National Technical University of Athens
School of Electrical and Computer Engineering
Division of Signals, Control and Robotics
Speech and Language Processing Group

Emotion Recognition in Conversation Using Prompt-Based Learning

Diploma Thesis

of

POLYTIMI-ANNA GKOTSI

Supervisor: Alexandros Potamianos
Associate Professor, NTUA

Athens, September 2022

National Technical University of Athens
School of Electrical and Computer Engineering
Division of Signals, Control and Robotics
Speech and Language Processing Group

Emotion Recognition in Conversation Using Prompt-Based Learning

Diploma Thesis

of

POLYTIMI-ANNA GKOTSI

Supervisor: Alexandros Potamianos
Associate Professor, NTUA

Approved by the examination committee on 27th September 2022.

(Signature)	*(Signature)*	*(Signature)*
....	....	....
Alexandros Potamianos	Stefanos Kollias	Constantinos Tzafestas
Associate Professor, NTUA	Professor, NTUA	Associate Professor, NTUA

Athens, September 2022

National Technical University of Athens
School of Electrical and Computer Engineering
Division of Signals, Control and Robotics
Speech and Language Processing Group

(Signature)

....
Polytimi-Anna Gkotsi

Electrical & Computer
Engineer

Περίληψη

Με τον τεράστιο όγκο δεδομένων συνομιλιών που δημοσιοποιούνται καθημερινά σε πλατφόρμες όπως το Twitter, το Reddit και το Facebook, η αυτόματη ανάλυσή τους με στόχο την εξόρυξη απόψεων και την κατανόηση της ανθρώπινης συμπεριφοράς γνωρίζει μεγάλη ζήτηση. Επιπλέον, η κατανόηση της συναισθηματικής κατάστασης του ανθρώπινου συνομιλητή είναι απαραίτητη για την ανάπτυξη πρακτόρων που διαθέτουν ενσυναίσθηση και μπορούν να αλληλεπιδρούν με τους ανθρώπους με τρόπο φυσικό για τους τελευταίους. Τέτοιες και άλλες πιθανές εφαρμογές έχουν οδηγήσει σε αυξανόμενο ερευνητικό ενδιαφέρον για το πρόβλημα της Αναγνώρισης Συναισθήμετος σε Συζητήσεις (Emotion Recognition in Conversation - ERC), το οποίο αποσκοπεί στον προσδιορισμό του συναισθήματος που εκφράζει κάθε εκφώνηση σε έναν δεδομένο διάλογο.

Πολλές προηγούμενες προσεγγίσεις στον τομέα χρησιμοποιούν προ-εκπαιδευμένα γλωσσικά μοντέλα, όπως τα BERT και RoBERTa, τα οποία προσαρμόζουν στο ERC πρόβλημα μέσω της μεθόδου fine-tuning. Ωστόσο, παρά την αποτελεσματικότητά της, η τελευταία είναι ιδιαίτερα δαπανηρή από άποψη υπολογιστικών και αποθηκευτικών πόρων, ενώ συχνά μπορεί να οδηγήσει σε υπερπροσαρμογή (overfitting). Μια εναλλακτική, πιο ελαφριά μέθοδος προσαρμογής που έχει προταθεί τα τελευταία χρόνια είναι η μάθηση βάσει prompt, η οποία διατηρεί παγωμένες τις προ-εκπαιδευμένες παραμέτρους του γλωσσικού μοντέλου και προσθέτει ένα μικρό σύνολο νέων, εκπαιδεύσιμων παραμέτρων στο επίπεδο εισόδου αυτού, που αποτελούν το ονομαζόμενο prompt. Καθώς πρόκειται ακόμη για μια πολύ νέα μέθοδο, η διαθέσιμη βιβλιογραφία, ιδίως στον τομέα της βελτιστοποίησης της μεθόδου για συγκεκριμένα προβλήματα, είναι περιορισμένη.

Στην παρούσα διπλωματική, στοχεύουμε στη μελέτη της μάθησης βάσει prompt ως μέθοδο προσαρμογής προ-εκπαιδευμένων γλωσσικών μοντέλων στο έργο του ERC. Ακολουθούμε δύο προσεγγίσεις και εκτελούμε εκτεταμένα πειράματα και για τις δύο.

Στην πρώτη μας προσέγγιση, στοχεύουμε να μελετήσουμε την εφαρμοσιμότητα της μάθησης βάσει prompt σε σύγκριση με το fine-tuning και να θέσουμε ένα βασικό μέτρο σύγκρισης (baseline) για τη μάθηση βάσει prompt για το ERC. Πειραματιζόμαστε με ένα απλό βασικό μοντέλο καθώς και με μοντέλα που χρησιμοποιούν δημοφιλείς μεθόδους που έχουν χρησιμοποιηθεί προηγουμένως στην βιβλιογραφία για την ενσωμάτωση πληροφοριών που αφορούν τον ομιλητή. Συμπεραίνουμε τελικά ότι η μάθηση βάσει prompt μπορεί πράγματι να συμβάλει στην προσαρμογή του προ-εκπαιδευμένου γλωσσικού μοντέλου μας, επιτυγχάνοντας μάλιστα επιδόσεις συγκρίσιμες με το fine-tuning για ένα από τα δύο σύνολα δεδομένων στα οποία πειραματιζόμαστε, με την απόδοσή της να εξαρτάται από το σύνολο δεδομένων, το μέγεθος του prompt, την αρχιτεκτονική και τη μέθοδο εκπαίδευσης.

Στη δεύτερη προσέγγισή μας, προτείνουμε μια μέθοδο για την ενσωμάτωση πρόσθετων πληροφοριών, χρήσιμων για την αναγνώριση συναισθήματος σε μια συζήτηση, απευθείας μέσω των prompts, χωρίς περαιτέρω αλλαγές στην αρχιτεκτονική και την είσοδο των προ-εκπαιδευμένων γλωσσικών μοντέλων. Πειραματιζόμαστε με την προσθήκη πληροφοριών συγκεκριμένων για κάθε ομιλητή και θέμα συζήτησης και παρατηρούμε αύξηση της απόδοσης σε πολλές περιπτώσεις, σε σύγκριση με το baseline μοντέλο μας. Η μέθοδός μας μπορεί εύκολα να επεκταθεί και σε άλλους τύπους πληροφοριών, εκτός από την ταυτότητα του ομιλητή και το θέμα συζήτησης, ακολουθώντας την ίδια λογική.

Λέξεις Κλειδιά

Μηχανική Μάθηση, Βαθιά Μάθηση, Επεξεργασία Φυσικής Γλώσσας, Αναγνώριση Συναισθήματος σε Συζητήσεις, Μάθηση Βάσει Prompt, Transformers, Προ-εκπαιδευμένα Γλωσσικά Μοντέλα, BERT, Prompt Ειδικά ανά Πληροφορία

Abstract

With the vast amount of conversational data that is made publicly available daily on platforms such as Twitter, Reddit, Facebook etc., its automatic analysis for the aim of mining opinions and understanding human behavior is in high demand. Additionally, a full grasp of the emotional state of the human interlocutor is necessary for the development of empathetic agents that can interact with humans in a manner natural to the latter. Such and other potential applications have lead to an increasing research interest on the task of Emotion Recognition in Conversation (ERC), which aims at determining the emotion each utterance in a given dialogue expresses.

A lot of previous approaches in the field utilize pre-trained language models such as BERT and RoBERTa as part of their proposed architecture, which they adapt to the specific task using the traditional fine-tuning method. However, despite its effectiveness, fine-tuning is very expensive in terms of computational and storage resources, while it can often lead to overfitting. An alternative, more lightweight method for the adaptation of pre-trained language models to downstream tasks that has been proposed in the recent years and aims at mitigating these issues, is prompt-based learning, which keeps the language model's pre-trained parameters frozen and adds a small set of new parameters in the mode's input level, called a prompt. Nevertheless, because it is still a very new method, there is limited work available, especially in the field of optimizing the method for a specific task.

In our work, we aim to study prompt-based learning as an adaptation method for the task of ERC. We follow two approaches and perform extensive experiments on both.

In our first approach, we aim to study the applicability of prompt-based learning in comparison to fine-tuning and set a baseline for prompt-based learning for Emotion Recognition in Conversation. We experiment with a simple baseline model as well as models utilizing popular methods previously used in related work for integrating speaker-specific information. We conclude that prompt-based learning can indeed contribute to the adaptation of our pre-trained language model, even yielding a performance comparable to fine-tuning for one of the two datasets we experiment on, with its performance depending on the dataset, prompt-size, architecture and training method.

In our second approach, we propose a method for integrating additional information, useful for recognizing emotion in a conversation, directly through the prompts, without further changes to the pre-trained language model's architecture and input. We experiment with adding speaker-specific and topic-specific information and observe an increase in performance in many cases, compared to our baseline. Our method may easily be extended to other types of information, besides speaker identity and topic, following the same logic.

Keywords

Machine Learning, Deep Learning, Natural Language Processing, Emotion Recognition in Conversation, Prompt-Based Learning, Transformers, Pre-trained Language Models, BERT, Information-Specific Prompts

Ευχαριστίες

Με την ολοκλήρωση της διπλωματικής μου εργασίας σηματοδοτείται το τέλος των προπτυχιακών σπουδών μου στο Εθνικό Μετσόβιο Πολυτεχνείο. Θα ήθελα στο πλαίσιο αυτό να εκφράσω τις θερμές μου ευχαριστίες στους ανθρώπους που με βοήθησαν για την περάτωση του παρόντος έργου.

Αρχικά, οφείλω ένα μεγάλο Ευχαριστώ στον καθηγητή και επιβλέποντά μου, Αλέξανδρο Ποταμιάνο. Η προθυμία και η συνεχής καθοδήγησή του, το αμείωτο ενδιαφέρον του και οι πολύτιμες παρατηρήσεις του, συντέλεσαν τόσο στην θεμελίωση της παρούσας εργασίας, όσο και στην επιστημονική μου ωρίμανση και διαμόρφωση. Παράλληλα, οι πλούσιες γνώσεις του, οι διαλέξεις του σε μαθήματα σχετιζόμενα με την μηχανική μάθηση και η αγάπη του για την έρευνα αποτέλεσαν σημαντικά ερεθίσματα που με ενέπνευσαν καθόλη την διάρκεια της εκπόνησης της διπλωματικής μου εργασίας.

Θα ήθελα επίσης να ευχαριστήσω τον διδακτορικό ερευνητή στην ομάδα του κ.Ποταμιάνου, Γιώργο Παρασκευόπουλο, οι συμβουλές και η καθοδήγηση του οποίου υπήρξαν ιδαίτερα πολύτιμες για την πορεία και ολοκλήρωση του παρόντος έργου.

Τέλος, θα ήθελα να ευχαριστήσω τους κοντινούς μου ανθρώπους, την οικογένειά μου καθώς και τους στενούς μου φίλους, οι οποίοι με στήριξαν καθ' όλη της διάρκεια του εγχειρήματός μου και ήταν πάντα διαθέσιμοι για να συζητήσουν μαζί μου, να με συμβουλεύσουν και να με βοηθήσουν.

Πολυτίμη-Άννα Γκότση

Αθήνα, Σεπτέμβριος 2022

Contents

Περίληψη 1

Abstract 3

Ευχαριστίες 5

List of Figures 11

List of Tables 15

0 Εκτεταμένη Ελληνική Περίληψη 17

0.1 Εισαγωγή 17

0.2 Θεωρητικό υπόβαθρο 18

0.2.1 Προ-εκπαιδευμένα γλωσσικά μοντέλα 18

0.2.2 Μέθοδοι προσαρμογής των προ-εκπαιδευμένων γλωσσικών μοντέλων σε επιμέρους προβλήματα 19

0.2.3 BERT 20

0.3 Μάθηση βάσει prompt 21

0.3.1 Ορισμός Μεθόδου 21

0.3.2 Κατηγορίες της Μεθόδου 22

0.3.3 Σχετική Βιβλιογραφία 22

0.4 Αναγνώριση συναισθήματος σε συζητήσεις 23

0.4.1 Ορισμός προβλήματος 23

0.4.2 Σχετική Βιβλιογραφία 23

0.5 Προτεινόμενες προσεγγίσεις 25

0.5.1 Συγκρίνοντας τη μάθηση βάσει prompt και το fine-tuning στο ERC 25

0.5.2 Prompts ειδικά ανά πληροφορία: Μία εναλλακτική προσέγγιση για την ενσωμάτωση επιπρόσθετης πληροφορίας 34

1 Introduction 43

1.1 Motivation 43

1.2 Contributions 44

1.3 Thesis Outline 44

2 Machine Learning and Deep Learning 47

2.1 Introduction 47

2.2 Types of machine learning 48

2.2.1 Supervised learning 48

2.2.2 Unsupervised learning 48

2.2.3 Semi-supervised learning 48

2.2.4 Reinforcement learning 49

2.3 Supervised learning . . . 49
2.3.1 Regression . . . 49
2.3.2 Classification . . . 50
2.4 Unsupervised learning . . . 53
2.4.1 Clustering . . . 53
2.4.2 Dimensionality reduction . . . 54
2.5 Artificial Neural Networks . . . 55
2.5.1 Activation function . . . 56
2.5.2 Loss function . . . 57
2.5.3 Back-propagation . . . 58
2.6 Additional machine learning concepts . . . 59
2.6.1 Generalization, overfitting and underfitting . . . 59
2.6.2 Evaluation metrics . . . 61
2.6.3 Transfer learning . . . 64
2.6.4 Multitask learning . . . 65
2.7 Deep learning models . . . 65
2.7.1 Feed-forward neural networks . . . 65
2.7.2 Convolutional neural networks . . . 66
2.7.3 Recurrent neural networks . . . 67
2.7.4 Attention mechanisms . . . 69
2.7.5 Transformers . . . 71

3 Word Representation Methods and Language Models 75
3.1 Introduction . . . 75
3.2 Word representation methods . . . 76
3.2.1 Traditional word representation . . . 76
3.2.2 Distributional word representation . . . 78
3.3 Language models . . . 80
3.3.1 Traditional language models . . . 80
3.3.2 Neural language models . . . 81
3.3.3 Large-scale pre-trained language models . . . 83
3.4 Adapting pre-trained language models . . . 86
3.4.1 Fine-tuning . . . 87
3.4.2 Model parameter ablation . . . 87
3.4.3 Adapter-tuning . . . 87
3.4.4 Prompt-based learning . . . 88

4 Prompt-based Learning 91
4.1 Introduction . . . 91
4.2 Definition . . . 91
4.3 Prompt engineering . . . 91
4.3.1 Determining prompts . . . 91
4.3.2 Prompt types . . . 92
4.4 Prompt-based learning and finetuning . . . 93
4.5 Previous work using soft prompts . . . 93

5 Emotion Recognition in Conversation **95**
5.1 Introduction 95
5.2 Definition 95
5.3 An overview of important variables in identifying emotion in conversation 95
5.3.1 Context 96
5.3.2 Inter- and intra-speaker dependencies 96
5.3.3 Emotional consistency 96
5.3.4 Speaker personality 97
5.3.5 Topic 97
5.4 Previous work 97
5.4.1 Sequence-based methods 97
5.4.2 Graph-based methods 98
5.5 Datasets 99
5.5.1 MELD 99
5.5.2 IEMOCAP 102

6 Proposed Approaches **105**
6.1 Comparing prompt-tuning and fine-tuning for ERC 105
6.1.1 Introduction 105
6.1.2 Method 106
6.1.3 Baselines 111
6.1.4 Experimental setup 113
6.1.5 Results and discussion 114
6.1.6 Additional discussion 118
6.2 Information-specific prompts: An alternative approach for integrating additional information 119
6.2.1 Introduction 119
6.2.2 Method 120
6.2.3 Baselines 125
6.2.4 Experimental setup 125
6.2.5 Results and discussion 126

7 Conclusions **133**
7.1 Discussion 133
7.2 Future work 134

Bibliography **137**

List of Abbreviations **145**

List of Figures

1 Γενικό διάγραμμα του μοντέλου μας που χρησιμοποιεί prompt στην περίπτωση που δεν προστίθενται ονομάτα ομιλητών. t είναι ο αριθμός των συμβόλων prompt, d είναι η διάσταση ενσωμάτωσης του BERT (768), n είναι ο αριθμός των ενσωματωμάτων που αντιστοιχούν στο κείμενο εισόδου μας, k είναι ο αριθμός των προηγούμενων εκφωνήσεων που χρησιμοποιούνται ως συμφραζόμενα και i ο δείκτης της τρέχουσας εκφώνησης που πρέπει να ταξινομηθεί. Το " |" συμβολίζει τη συνένωση. Οι μονάδες με μπλε χρώμα διατηρούνται παγωμένες κατά τη διάρκεια της εκπαίδευσης, ενώ οι μονάδες με κίτρινο χρώμα είναι εκπαιδεύσιμες. Σημειώνεται ότι η μονάδα ενσωμάτωσης prompt (prompt embedding) διαθέτει εκπαιδεύσιμα μέρη, αλλά δεν είναι εκπαιδεύσιμη στο σύνολό της: Βλ. Εικόνα 2 για μια λεπτομερή επισκόπηση. 27

2 Επισκόπηση του στρώματος ενσωμάτωσης για την περίπτωση μάθησης βάσει prompt. Η τελική έξοδος του στρώματος ενσωμάτωσης είναι μια συνένωση των ενσωματωμάτων prompt και κειμένου. Οι μονάδες με μπλε χρώμα διατηρούνται παγωμένες κατά τη διάρκεια της εκπαίδευσης, ενώ οι μονάδες με κίτρινο χρώμα είναι εκπαιδεύσιμες. . . . 29

3 Διάγραμμα κεφαλής ταξινόμησης, όπου το c συμβολίζει τον αριθμό κλάσεων στο σύνολο δεδομένων μας. 29

4 Επισκόπηση του στρώματος ενσωμάτωσης για την περίπτωση μάθησης βάσει prompt. Η τελική έξοδος στο επίπεδο ενσωμάτωσης προκύπτει από τη συνένωση των ειδικών για το πρόβλημα ενσωματωμάτων prompt (task-specific prompt embeddings), τον ειδικών ανά πληροφορία ενσωματωμάτων prompt (information-specific prompt-embeddings) (ειδικών ανά ομιλητή ή θέμα) και των ενσωματωμάτων κειμένου (text embeddings). Οι μονάδες με μπλε χρώμα διατηρούνται παγωμένες κατά τη διάρκεια της εκπαίδευσης, ενώ οι μονάδες με κίτρινο χρώμα είναι εκπαιδεύσιμες. 37

5 Γενικό διάγραμμα του μοντέλου μας με βάση τα prompts, όπου t είναι ο αριθμός των ειδικών για το πρόβλημα συμβόλων prompt, m είναι ο αριθμός των ειδικών ανά πληροφορία (information-specific) συμβόλων prompt, s είναι ο αριθμός των ομιλητών/θεμάτων, d είναι η διάσταση ενσωμάτωσης του BERT (768), n είναι ο αριθμός των συμβόλων prompt που αντιστοιχούν στο κείμενο εισόδου μας, k είναι ο αριθμός των προηγούμενων εκφωνήσεων που χρησιμοποιούνται ως συμφραζόμενα και i είναι ο δείκτης της τρέχουσας εκφώνησης, η οποία πρέπει να ταξινομηθεί. Για s ομιλητές/θέματα εκπαιδεύουμε s prompts που αποτελούνται από m σύμβολα prompt το καθένα, οπότε έχουμε συνολικά $s * m$ εκπαιδεύσιμα ενσωματώματα prompt, από τα οποία επιλέγουμε m κάθε φορά, ανάλογα με τον ομιλητή ή το θέμα. Οι μονάδες με μπλε χρώμα διατηρούνται παγωμένες κατά τη διάρκεια της εκπαίδευσης, ενώ οι μονάδες με κίτρινο χρώμα είναι εκπαιδεύσιμες. Σημειώνεται ότι τα ειδικά για το πρόβλημα και τα ειδικά ανά πληροφορία ενσωματώματα prompt έχουν εκπαιδεύσιμα τμήματα, αλλά δεν είναι εκπαιδεύσιμα στο σύνολό τους: Βλ. Εικόνα 4 για μια λεπτομερή επισκόπηση. 39

2.1 The dropout method for a simple neural network with two hidden layers. Left: The neural network before dropout. Right: The neural network after applying dropout. Image obtained from: [1] 61
2.2 An MLP with three input units, one output unit, and two hidden layers, each consisting of four perceptrons. Image obtained from: [2] 66
2.3 A CNN example. Image obtained from: [3] 67
2.4 An RNN, where X_t is the input vector and Y_t is the output vector for time step t. Image obtained from: [4] 67
2.5 The basic LSTM unit, where X_t is the input vector and h_t is the hidden state for time step t. Image obtained from: [5] 69
2.6 The encoder-decoder architecture using attention, as proposed by Bahdanau et al. in [6]. Image obtained from: [6] 70
2.7 The transformer encoder and decoder modules, as proposed by Vaswani et al. in [7]. Image obtained from: [7] 71

3.1 The Skip-gram and CBOW models. Image obtained from: [8] 79
3.2 The architecture of the FFNNLM proposed by Bengio et al. in [9]. Image obtained from: [9] 82
3.3 BERT input representations: The input is a sum of token embeddings, segment embeddings and position embeddings. Image obtained from [7]. 85
3.4 The pre-training and fine-tuning procedures of BERT. Image obtained from [7]. . 86
3.5 Right: Architecture of the adapter module. Left: The integration of the adapter module with the transformer. Image obtained from: [10]. 88

5.1 An example of a conversation for emotion recognition. Image obtained from: [11] . 95
5.2 An example dialogue in MELD, between two speakers. Source: [12] 100

6.1 Embedding layer overview for the prompt-based learning case. The final embedding output is a concatenation of the prompt and text embeddings. The modules in blue color are kept frozen during training, while the modules in yellow are trainable. . 108
6.2 Classification head diagram, where c represents the number of classes of the dataset. 109
6.3 General diagram of our prompt-based model in the case where no speaker-token prepending is used. t is the number of prompt tokens, d is BERT's embedding dimension (768), n is the number of embeddings corresponding to our text input, k is the number of past utterances used as context and i the index of the current utterance to be classified. "|" symbolizes concatenation. The modules in blue color are kept frozen during training, while the modules in yellow are trainable. The modules in blue color are kept frozen during training, while the modules in yellow are trainable for the prompt-based learning case. Note that the prompt embeddings block has trainable parts, but is not trainable as a whole: See Figure 6.1 for a detailed overview. 110
6.4 General diagram of our prompt-based model used for multitask training, where t is the number of prompt tokens, d is BERT's embedding dimension (768), n the number of embeddings corresponding to our text input, k is the number of past utterances used as context and i is the index of the current utterance to be classified. "|" symbolizes concatenation. The modules in blue color are kept frozen during training, while the modules in yellow are trainable. Note that the prompt embeddings block has trainable parts, but is not trainable as a whole: See Figure 6.1 for a detailed overview. 111

6.5 General diagram of our fine-tuned baseline model, in the case where no speaker-token prepending is used. d is BERT's embedding dimension (768), n is the number of embeddings corresponding to our text input, k is the number of past utterances used as context and i is the index of the current utterance to be classified. "|" symbolizes concatenation. The modules in blue color are kept frozen during training, while the modules in yellow are trainable. 112

6.6 General diagram of our baseline model with only a trainable classification head, where d is BERT's embedding dimension (768), n is the number of embeddings corresponding to our text input, k is the number of past utterances used as context and i is the index of the current utterance to be classified. "|" symbolizes concatenation. The modules in blue color are kept frozen during training, while the modules in yellow are trainable. 113

6.7 Embedding layer overview for the prompt-based learning case. The final embedding-level output consists of the concatenation of the task-specific prompt embeddings, the information-specific (speaker- or topic-specific) prompt embeddings and the text embeddings. The modules in blue color are kept frozen during training, while the modules in yellow are trainable. 121

6.8 General diagram of our prompt-based model, where t is the number of task specific prompt tokens, m is the number of information-specific (speaker-specific or topic-specific) prompt tokens, s is the number of speakers/topics, d is BERT's embedding dimension (768), n is the number of embeddings corresponding to our text input, k is the number of past utterances used as context and i is the index of the current utterance to be classified."|" symbolizes concatenation. For s speakers/topics we train s information-specific prompts consisting of m prompt tokens, so we have a total of $s * m$ trainable prompt embeddings, of which m are chosen every time, according to the speaker or topic. The modules in blue color are kept frozen during training, while the modules in yellow are trainable. Note that the task-specific and speaker-/topic-specific embedding blocks have trainable parts, but are not trainable as a whole: See Figure 6.7 for a detailed overview. 123

List of Tables

1 Σταθμισμένο F1 score (%) για τα σύνολα δεδομένων ελέγχου MELD και IEMOCAP 30
2 Σταθμισμένο F1-score (%) για τα σύνολα δεδομένων ελέγχου MELD και IEMOCAP 31
3 Σταθμισμένο F1-score (%) για το σύνολο δεδομένων ελέγχου MELD 32
4 Σταθμισμένο F1-score (%) για το σύνολο δεδομένων MELD 40
5 Σταθμισμένο F1-score (%) για το σύνολο δεδομένων IEMOCAP 40

5.1 General statistics for MELD. 100
5.2 Class distribution (%) for MELD. 101
5.3 # Utterances per speaker in MELD. 101
5.4 Emotion distribution per speaker for MELD dataset, for train data. 102
5.5 Emotion distribution per speaker for MELD dataset, for validation data. 102
5.6 Emotion distribution per speaker for MELD dataset, for test data. 102
5.7 General statistics for IEMOCAP. 103
5.8 Class distribution (%) for IEMOCAP. 103
5.9 # Utterances per speaker in IEMOCAP. 104
5.10 Emotion distribution per speaker for IEMOCAP dataset. 104

6.1 Weighted F1-score (%) for MELD dataset 114
6.2 Weighted F1-score (%) for IEMOCAP dataset 115
6.3 F1-score (%) per class for MELD test dataset 115
6.4 F1-score (%) per class for IEMOCAP test dataset 115
6.5 Weighted F1-score (%) for MELD dataset 116
6.6 Weighted F1-score (%) for IEMOCAP dataset 116
6.7 F1-score (%) per class for MELD test dataset 116
6.8 F1-score (%) per class for IEMOCAP test dataset 117
6.9 Weighted F1-score (%) for MELD dataset 118
6.10 F1-score (%) per class for MELD test dataset 118
6.11 Weighted F1-score (%) for MELD dataset 126
6.12 Weighted F1-score (%) for IEMOCAP dataset 126
6.13 F1-score (%) per class for MELD test dataset 127
6.14 F1-score (%) per class for IEMOCAP test dataset 127
6.15 Emotion distribution per topic for MELD dataset, for train data. 130
6.16 Emotion distribution per topic for MELD dataset, for test data. 130
6.17 Emotion distribution per topic for IEMOCAP dataset, for train data. 130
6.18 Emotion distribution per topic for IEMOCAP dataset, for test data. 130

Chapter 0

Εκτεταμένη Ελληνική Περίληψη

0.1 Εισαγωγή

Το συναίσθημα είναι έμφυτο στον άνθρωπο και αποτελεί σημαντικό παράγοντα που επηρεάζει την ανθρώπινη συμπεριφορά. Η αναγνώριση του ανθρώπινου συναισθήματος αποτελεί επομένως βασικό μέρος της κατανόησης του ανθρώπου. Στο πλαίσιο ενός διαλόγου, ο ακριβής προσδιορισμός του συναισθήματος μπορεί να είναι καθοριστικός για την ερμηνεία της συμπεριφοράς και της πρόθεσης κάθε συνομιλητή, καθώς και του νοήματος των λεγόμενών του. Με την άνοδο των διαδικτυακών πλατφορμών και των μέσων κοινωνικής δικτύωσης, όπως το Facebook, το Reddit, το Twitter κ.λπ. ένας συντριπτικός όγκος δεδομένων συνομιλίας καθίσταται καθημερινά διαθέσιμος δημόσια και η αυτόματη ανάλυσή τους με σκοπό την εξόρυξη απόψεων και τη μελέτη της ανθρώπινης συμπεριφοράς είναι περιζήτητη. Επιπλέον, η επιδίωξη της δημιουργίας διαλογικών πρακτόρων και βοηθών που διαθέτουν τεχνητή νοημοσύνη που μοιάζει με την ανθρώπινη και είναι σε θέση να συνομιλούν με τον άνθρωπο με τρόπο φυσικό προς αυτόν, απαιτεί την εις βάθος κατανόηση της συναισθηματικής κατάστασης του ανθρώπινου συνομιλητή. Σε ένα πολύ διαφορετικό πλαίσιο, η αναγνώριση του συναισθήματος όταν οι άνθρωποι συνομιλούν μπορεί να βοηθήσει στην ανάπτυξη εργαλείων ψυχολογικής ανάλυσης για την υποβοήθηση των γιατρών και μπορεί έτσι να υποστηρίξει την υγειονομική περίθαλψη.

Η αναγνώριση συναισθήματος σε συζητήσεις (Emotion Recognition in Conversation - ERC) μπορεί να οριστεί ως το πρόβλημα του προσδιορισμού του συναισθήματος κάθε εκφώνησης, δεδομένης μιας σειράς εκφωνήσεων που αποτελούν έναν διάλογο μεταξύ δύο ή περισσότερων ανθρώπινων ομιλητών. Λόγω των προαναφερθέντων δυνητικών εφαρμογών της, έχει αποκτήσει σημαντική δημοτικότητα τα τελευταία χρόνια, με όλο και περισσότερους ερευνητές να βελτιώνουν συνεχώς τα προηγούμενα συστήματα και να προτείνουν νέες αρχιτεκτονικές. Τα προτεινόμενα μοντέλα συχνά αξιοποιούν την ταυτότητα του ομιλητή [11] [13] [14] [15], τη μοντελοποίηση του θέματος συζήτησης [16], και την κωδικοποίηση της ενδοεξάρτησης και της αλληλεξάρτησης ομιλητών [17], προκειμένου να συλλάβουν αποτελεσματικά το συναίσθημα, και συνήθως βασίζονται σε μία από δύο κύριες προσεγγίσεις: Η πρώτη χρησιμοποιεί νευρωνικά δίκτυα γράφων, κωδικοποιώντας τις εκφωνήσεις και τις σχέσεις τους ως κόμβους και ακμές ενός γράφου και μοντελοποιώντας τις διάφορες εξαρτήσεις μέσω της συνάθροισης πληροφοριών από γειτονικούς κόμβους. Η δεύτερη αξιοποιεί τη ακολουθιακή φύση των συνομιλιών, χρησιμοποιώντας μοντέλα που αποτυπώνουν ρητά τις ακολουθιακές σχέσεις, όπως τα Επαναλαμβανόμενα Νευρωνικά Δίκτυα (Recurrent Neural Networks - RNNs) και τα προ-εκπαιδευμένα γλωσσικά μοντέλα που βασίζονται σε transformers (όπως τα μοντέλα BERT [18], RoBERTa [19], BART [20] κ.λπ.), και είναι ίσως η πιο συνηθισμένη στα συστήματα ERC τελευταίας τεχνολογίας.

Τα προ-εκπαιδευμένα γλωσσικά μοντέλα επιτρέπουν στους ερευνητές να επιτυγχάνουν πολύ καλές επιδόσεις όταν προσπαθούν να αναγνωρίσουν συναισθήματα σε συνομιλίες, λόγω της εκτεταμένης κατανόησης της ανθρώπινης γλώσσας που έχουν αποκτήσει κατά τη φάση της προ-εκπαίδευσής τους. Για την προσαρμογή τους στο περιβάλλον του ERC χρησιμοποιείται συνήθως fine-tuning, μέθοδος η οποία τροποποιεί όλες τις παραμέτρους του προ-εκπαιδευμένου γλωσσικού μοντέλου χρησιμοποιώντας ένα σύνολο δεδομένων σχεδιασμένο για το πρόβλημα του ERC. Ωστόσο, λαμβάνοντας υπόψη το γεγονός ότι τα εν λόγω μοντέλα αποτελούνται από εκατομμύρια ή και δισεκατομμύρια παραμέτρους, το fine-tuning αυτών μπορεί να αποτελεί μία πολύ δαπανηρή διαδικασία: Η εκπαίδευση απαιτεί τόσο πολύ χρόνο όσο και πολλούς υπολογιστικούς πόρους, ενώ τα μοντέλα που προκύπτουν καταλαμβάνουν μεγάλο αποθηκευτικό χώρο [21]. Επιπλέον, επειδή τα σύνολα δεδομένων που είναι διαθέσιμα για το πρόβλημα ERC είναι σχετικά μικρά σε σύγκριση με τον αριθμό των παραμέτρων που διαθέτουν τα προ-εκπαιδευμένα γλωσσικά μοντέλα, τα εκπαιδευμένα με fine-tuning μοντέλα συχνά υποφέρουν από υπερπροσαρμογή (overfitting) [22].

Για να αμβλυνθούν αυτά τα προβλήματα, έχει προταθεί η μάθηση βάσει prompt (προτροπών), ως μια γενική μέθοδος για την προσαρμογή προ-εκπαιδευμένων γλωσσικών μοντέλων σε επιμέρους προβλήματα. Αντί να τροποποιεί τις παραμέτρους του προ-εκπαιδευμένου μοντέλου, η μάθηση βάσει prompt τις διατηρεί παγωμένες και προσθέτει ένα σύνολο νέων παραμέτρων, γνωστό με την ορολογία prompt, στο επίπεδο εισόδου του μοντέλου, τις οποίες στη συνέχεια βελτιστοποιεί, προκειμένου να επηρεάσει τον τρόπο με τον οποίο το μοντέλο χειρίζεται την είσοδο, ώστε να προσαρμοστεί καλύτερα στο επιμέρους πρόβλημα. Επειδή ο αριθμός των νέων εισαγόμενων παραμέτρων που εκπαιδεύονται αποτελεί ένα πολύ μικρό ποσοστό του αριθμού των προ-εκπαιδευμένων παραμέτρων που η μέθοδος του fine-tuning παραδοσιακά βελτιστοποιεί, η μάθηση βάσει prompt είναι σημαντικά πιο ελαφριά και λιγότερο επιρρεπής στην πρόκληση υπερπροσαρμογής. Έχει έτσι προσελκύσει το ενδιαφέρον πολλών ερευνητών, οι οποίοι έχουν προτείνει διαφορετικές αρχιτεκτονικές και μεθόδους εκπαίδευσης. Ωστόσο, παρά το αυξανόμενο ενδιαφέρον της ερευνητικής κοινότητας, η μάθηση βάσει prompt εξακολουθεί να αποτελεί μια πολύ νέα μέθοδο και συνεπώς η διαθέσιμη βιβλιογραφία είναι περιορισμένη, ιδίως στον τομέα της βελτιστοποίησης της μεθόδου για ένα συγκεκριμένο πρόβλημα. Ιδιαίτερα στον τομέα του ERC, κατά τη περίοδο των πειραμάτων μας, δεν καταφέραμε να βρούμε καμία εργασία που να χρησιμοποιεί καθαρά μια προσέγγιση μάθησης βάσει prompt, η οποία να εκπαιδεύει μόνο τις παραμέτρους του prompt, διατηρώντας το προ-εκπαιδευμένο μοντέλο παγωμένο και παραμένοντας έτσι πραγματικά ελαφριά από άποψη πόρων.

Στην παρούσα διπλωματική εργασία στοχεύουμε στην μελέτη τη δυνατότητας εφαρμογής της μάθησης βάσει prompt για την προσαρμογή ενός μεγάλου προ-εκπαιδευμένου γλωσσικού μοντέλου στο έργο της Αναγνώρισης Συναισθήματος σε Συζητήσεις (ERC). Στο πρώτο μέρος της εργασίας μας, θέτουμε ένα βασικό μοντέλο σύγκρισης (baseline) για το βασιζόμενο σε prompts ERC και πειραματιζόμαστε με μεθόδους που χρησιμοποιούνται συνήθως σε προηγούμενα έργα για την ενσωμάτωση πληροφοριών που αφορούν τον ομιλητή και τη μοντελοποίηση της ενδοεξάρτησης και της αλληλεξάρτησης ομιλητών. Συγκρίνουμε τα μοντέλα μας με μοντέλα που χρησιμοποιούν fine-tuning και μετράμε τη διαφορά στην απόδοση μεταξύ των δύο μεθόδων προσαρμογής. Στη συνέχεια παρουσιάζουμε μία δεύτερη προσέγγιση, η οποία αποσκοπεί στην κωδικοποίηση πρόσθετων, χρήσιμων για το πρόβλημα του ERC πληροφοριών, όπως η ταυτότητα του ομιλητή ή το θέμα συζήτησης, απευθείας μέσω των prompts, χωρίς περαιτέρω αλλαγές στο μορφότυπο εισόδου ή στην αρχιτεκτονική του μοντέλου. Πραγματοποιούμε εκτενή πειράματα και για τις δύο προσεγγίσεις, καθώς και μια συζήτηση σχετικά με την εφαρμοσιμότητα της μάθησης βάσει prompt στο πρόβλημα του ERC.

0.2 Θεωρητικό υπόβαθρο

0.2.1 Προ-εκπαιδευμένα γλωσσικά μοντέλα

Τα γλωσσικά μοντέλα είναι μοντέλα που αναθέτουν πιθανότητες σε ακολουθίες λέξεων [23]. Αποτελούν το θεμέλιο της Επεξεργασίας Φυσικής Γλώσσας, καθώς προσφέρουν μια μέθοδο μετατροπής ποιοτικών πληροφοριών κειμένου σε κατανοητά από τη μηχανή ποσοτικά δεδομένα. Κατά την πάροδο των ετών, έχουν προταθεί διάφορα γλωσσικά μοντέλα. Με την πρόοδο της βαθιάς μάθησης, τα παραδοσιακά γλωσσικά μοντέλα που βασίζονται στη στατιστική αντικαταστάθηκαν από γλωσσικά μοντέλα που χρησιμοποιούν νευρωνικά δίκτυα, τα οποία τελικά οδήγησαν στα σημερινά μεγάλα προ-εκπαιδευμένα γλωσσικά μοντέλα, όπως το BERT [18] και το GPT-3 [24]. Τα προ-εκπαιδευμένα γλωσσικά μοντέλα αξιοποιούν τις τεράστιες ποσότητες δεδομένων κειμένου χωρίς ετικέτες για να εκπαιδευτούν, μέσω μη επιβλεπόμενης ή αυτο-επιβλεπόμενης μάθησης, προκειμένου να αποκτήσουν μια γενική κατανόηση της φυσικής γλώσσας. Μετά την εκπαίδευσή τους, μπορούν να προσαρμοστούν σε επιμέρους προβλήματα περνώντας από μερικούς ακόμη γύρους εκπαίδευσης, με εφαρμογή της μεθόδου fine-tuning, χρησιμοποιώντας μικρότερα σύνολα δεδομένων με ετικέτες.

0.2.2 Μέθοδοι προσαρμογής των προ-εκπαιδευμένων γλωσσικών μοντέλων σε επιμέρους προβλήματα

Παραδοσιακά, για την προσαρμογή των προ-εκπαιδευμένων γλωσσικών μοντέλων σε επιμέρους προβλήματα, χρησιμοποιείται fine-tuning. Ωστόσο, το fine-tuning δεν είναι πάντα βέλτιστη μέθοδος: Μπορεί να προκαλέσει υπερπροσαρμογή του προ-εκπαιδευμένου μοντέλου, ενώ τροποποιώντας τις παραμέτρους του τελευταίου μπορεί να το οδηγήσει στο να ξεχάσει σημαντικές γλωσσικές σχέσεις και αναπαραστάσεις, τις οποίες αυτό διδάχτηκε κατά το στάδιο της προ-εκπαίδευσης. Επιπλέον, το αυξανόμενο μέγεθος των προ-εκπαιδευμένων μοντέλων σήμερα σημαίνει ότι το fine-tuning αυτών των μοντέλων μπορεί να είναι σχετικά δαπανηρό και όχι πάντα η πιο αποδοτική από άποψη πόρων εναλλακτική. Για τους λόγους αυτούς, τα τελευταία χρόνια έχουν προταθεί διάφορες εναλλακτικές λύσεις για την προσαρμογή των προ-εκπαιδευμένων γλωσσικών μοντέλων, ορισμένες από τις οποίες είναι:

Αφαίρεση παραμέτρων του μοντέλου Προκειμένου να μειωθεί το αποτύπωμα στη μνήμη από την προσαρμογή προ-εκπαιδευμένων γλωσσικών μοντέλων, ορισμένοι ερευνητές έχουν προτείνει την αφαίρεση παραμέτρων του μοντέλου [25] [26]: Αντί να τροποποιήσει τις παραμέτρους του μοντέλου, αυτή η γραμμή εργασίας εκπαιδεύει δυαδικές μάσκες που καθορίζουν ποια βάρη του μοντέλου θα αφαιρεθούν [21] για κάθε πρόβλημα. Χρησιμοποιώντας αυτή τη μέθοδο, οι ερευνητές έχουν παρατηρήσει αποτελέσματα συγκρίσιμα με την μέθοδο του fine-tuning, αν και με μία τάση να είναι λίγο χαμηλότερα σε πολλές περιπτώσεις. Το μεγάλο πλεονέκτημα της μεθόδου αφαίρεσης παραμέτρων είναι ο μικρότερος απαιτούμενος αποθηκευτικός χώρος: Κατά το fine-tuning για την επίλυση πολλαπλών προβλημάτων ταυτόχρονα, ολόκληρο το fine-tuned μοντέλο πρέπει να αποθηκευτεί για κάθε πρόβλημα. Από την άλλη πλευρά, με τη χρήση δυαδικών μασκών, χρειάζεται να αποθηκευτεί μόνο ένα αντίγραφο ολόκληρου του προ-εκπαιδευμένου μοντέλου, καθώς και ένα σύνολο δυαδικών μασκών για κάθε πρόβλημα, μειώνοντας έτσι σε μεγάλο βαθμό τον απαραίτητο αποθηκευτικό χώρο [25].

Adapter-tuning Στη μέθοδο του adapter-tuning, μονάδες που βασίζονται σε υπολειμματικά δίκτυα (residual networks) εισάγονται μέσα στις προ-εκπαιδευμένες μονάδες-transformer του γλωσσικού μοντέλου. Οι νέες μονάδες, που ονομάζονται υπολειμματικοί προσαρμογείς (residual adapters), εισήχθησαν αρχικά από τους Rebuffi et al. [27] και εκπαιδεύονται χρησιμοποιώντας το σύνολο δεδομένων του επιμέρους προβλήματος, ενώ οι παράμετροι του αρχικού προ-εκπαιδευμένου μοντέλου διατηρούνται παγωμένες. Οι διαδικασία εισαγωγής και εκπαίδευσης υπολειμματικών προσαρμογέων αντί των παραμέτρων των προ-εκπαιδευμένων μοντέλων αναφέρεται συχνά ως adapter-tuning [21]. Συνολικά, η μέθοδος του adapter-tuning έχει επιτύχει επιδόσεις συγκρίσιμες με το fine-tuning, ενώ παράλληλα μειώνει σημαντικά τις εκπαιδεύσιμες παραμέτρους του μοντέλου και, ως αποτέλεσμα, τον απαιτούμενο αποθηκευτικό χώρο. Επιπλέον, οι προσαρμογείς επιτρέπουν την εύκολη και αποτελεσματική ανταλλαγή πληροφοριών μεταξύ διαφορετικών προβλημάτων, βελτιώνοντας συχνά την απόδοση.

Μάθηση βάσει prompt Η μάθηση βάσει prompt διατηρεί το προ-εκπαιδευμένο μοντέλο αμετάβλητο και εισάγει έναν μικρό αριθμό παραμέτρων, που αποτελούν το ονομαζόμενο prompt, είτε στο επίπεδο της κειμενικής εισόδου είτε απευθείας στο επίπεδο embedding (ενσωμάτωσης) του μοντέλου. Με το μέγεθος των σύγχρονων προ-εκπαιδευμένων γλωσσικών μοντέλων να αυξάνεται συνεχώς, η μάθηση βάσει prompt ερευνάται κυρίως λόγω των χαμηλότερων απαιτήσεών της όσον αφορά τους πόρους εκπαίδευσης και αποθήκευσης, σε σύγκριση με την παραδοσιακή μέθοδο του fine-tuning. Μια εκτενής ανάλυση της μάθησης βάσει prompt παρέχεται σε επόμενη ενότητα (Ενότητα 0.3).

0.2.3 BERT

Το BERT, που σημαίνει Bidirectional Encoder Representations from Transformers (Αμφίδρομες Αναπαραστάσεις Κωδικοποιητή για Transformers), προτάθηκε το 2018 από την Google στο [18]. Το BERT είναι ένα αμφίδρομο μοντέλο, το οποίο χρησιμοποιεί τόσο τα αριστερά όσο και τα δεξιά συμφραζόμενα για να δημιουργήσει αναπαραστάσεις λέξεων. Βασίζεται στον transformer, όπως αυτός παρουσιάστηκε από τους Vaswani et al. στο [7] και αποτελεί έναν αμφίδρομο κωδικοποιητή αρχιτεκτονικής transformer πολλαπλών επιπέδων. Η μικρότερη έκδοση του BERT, το bert-base, αποτελείται από 110 εκατομμύρια εκπαιδεύσιμες παραμέτρους, ενώ η μεγαλύτερη έκδοση του, το bert-large, αποτελείται από 340 εκατομμύρια εκπαιδεύσιμες παραμέτρους. Το BERT πέτυχε κορυφαία αποτελέσματα για πολλά προβλήματα Επεξεργασίας Φυσικής Γλώσσας κατά την δημιουργία του και έχει έκτοτε χρησιμοποιηθεί σε πολλά συστήματα για μια μεγάλη ποικιλία προβλημάτων. Με βάση το BERT και με τροποποιήσεις στην αρχιτεκτονική του ή στη φάση προ-εκπαίδευσης, έχουν δημιουργηθεί πολλά προ-εκπαιδευμένα γλωσσικά μοντέλα υψηλής απόδοσης, όπως τα RoBERTa [19], ELECTRA [28] και XLNet [29].

Επίπεδο εισόδου και ενσωμάτωσης Η εισόδος που δέχεται το BERT μπορεί να αποτελείται είτε από ένα είτε από δύο τμήματα κειμένου (που αναφέρονται στο [7] ως προτάσεις), τα οποία χωρίζονται από ένα ειδικό σύμβολο ([SEP]). Επιπλέον, ένα ειδικό σύμβολο ([CLS]) προστίθεται πάντα στην αρχή της εισόδου. Η τελική κρυφή αναπαράσταση αυτού του συμβόλου χρησιμοποιείται ως αναπαράσταση για ολόκληρη την ακολουθία σε προβλήματα ταξινόμησης.

Εκτός από τη χρήση ενσωματωμάτων (embeddings) που αντιστοιχούν στα σύμβολα στα οποία αντιστοιχίζεται το κείμενο εισόδου ("token embeddings" - ενσωματώματα συμβόλων), το BERT χρησιμοποιεί δύο επιπλέον τύπους εκπαιδεύσιμων ενσωματωμάτων: ενσωματώματα που ονομάζονται ενσωματώματα τμήματος ("segment embeddings") και χρησιμοποιούνται για τη διάκριση μεταξύ των συμβόλων που ανήκουν στο πρώτο τμήμα εισόδου και των συμβόλων που ανήκουν στο δεύτερο τμήμα εισόδου, και ενσωματώματα θέσης ("position embeddings"), τα οποία κωδικοποιούν την απόλυτη θέση κάθε συμβόλου στην είσοδο. Οι τελικές αναπαραστάσεις εισόδου που διαβιβάζονται στα στρώματα του κωδικοποιητή του BERT, είναι ένα άθροισμα των ενσωματωμάτων συμβόλων, τμήματος και θέσης.

Στάδιο προ-εκπαίδευσης Η εκπαίδευση του BERT αποτελείται από δύο στάδια, ένα στάδιο προ-εκπαίδευσης και ένα στάδιο fine-tuning. Στο στάδιο της προ-εκπαίδευσης, χρησιμοποιούνται δύο μη επιβλεπόμενα προβλήματα, προκειμένου να εκπαιδευτούν οι παράμετροι του μοντέλου χρησιμοποιώντας μη επισημασμένα δεδομένα, και συγκεκριμένα τα Masked Language Modeling (Masked LM) και Next Sentence Prediction (NSP):

- **Masked LM**: Κατά τη διάρκεια του Masked Language Modeling (Masked LM), επιλέγεται τύχαια και αποκρύπτεται το 15% όλων των συμβόλων σε κάθε πρόταση εισόδου και το μοντέλο καλείται να προβλέψει τις σωστές λέξεις που αποκρύφθηκαν. Προκειμένου να προ-εκπαιδευτεί το μοντέλο με τρόπο που να διατηρείται όσο το δυνατόν πιο κοντά στις συνθήκες στις οποίες θα πραγματοποιηθεί το fine-tuning, οι Delvin et al. επέλεξαν να αντικαταστήσουν τα σύμβολα που θα αποκρυφθούν με ένα ειδικό σύμβολο ($[MASK]$) στο 80% των περιπτώσεων, και ένα άλλο, τυχαίο σύμβολο στο 10% των περιπτώσεων. Στο υπόλοιπο 10% των περιπτώσεων το σύμβολο που έχει επιλεχθεί για απόκρυψη διατηρείται αμετάβλητο.

- **Next Sentence Prediction (NSP)**: Σε αυτό το πρόβλημα, παρουσιάζονται δύο προτάσεις στο μοντέλο και αυτό καλείται να προβλέψει αν η δεύτερη πρόταση ακολουθεί την πρώτη στο κείμενο. Για το σκοπό αυτό οι συγγραφείς επέλεξαν ζεύγη προτάσεων στα οποία η δεύτερη πρόταση πράγματι ακολουθούσε την πρώτη στο 50% των περιπτώσεων και στα οποία η δεύτερη πρόταση ήταν μια τυχαία πρόταση από το σώμα κειμένων στο υπόλοιπο 50% των περιπτώσεων.

Στάδιο Fine-tuning Μετά την προ-εκπαίδευση του μοντέλου, τα προ-εκπαιδευμένα βάρη μπορούν να φορτωθούν και το μοντέλο μπορεί να υποστεί fine-tuning για την προσαρμογή του σε επιμέρους προβλήματα (όπως η απάντηση ερωτήσεων, η αναγνώριση συναισθήματος, η εξαγωγή συμπερασμάτων κ.λπ.) Για το σκοπό αυτό, το κείμενο εισόδου μετασχηματίζεται, ώστε να ταιριάζει με το πρότυπο εισόδου του BERT. Για παράδειγμα, για προβλήματα που σχετίζονται με την απάντηση ερωτήσεων, το ζεύγος δύο προτάσεων που δέχεται ως είσοδο το BERT αντιστοιχίζεται στο ζεύγος ερώτησης-συμφραζομένων αναφοράς, ενώ για εργασίες σήμανσης ακολουθιών (sequence tagging) και ταξινόμησης (classification) χρησιμοποιείται μόνο η πρώτη πρόταση εισόδου τοου BERT και η δεύτερη πρόταση παραμένει κενή, ενώ η τελική κρυφή αναπαράσταση του συμβόλου ($[CLS]$) μεταβιβάζεται σε μια κεφαλή ταξινόμησης.

0.3 Μάθηση βάσει prompt

Όπως έχουμε εξηγήσει νωρίτερα, η μάθηση βάσει prompt αποτελεί μια μέθοδο προσαρμογής για προ-εκπαιδευμένα γλωσσικά μοντέλα η οποία, αντί να τροποποιεί τις παραμέτρους του προ-εκπαιδευμένου μοντέλου, προσθέτει ένα νέο, μικρό σύνολο παραμέτρων, που ονομάζεται prompt, στην είσοδο του μοντέλου [30] [31]. Οι παράμετροι αυτές προστίθενται είτε στο επίπεδο της κειμενικής εισόδου, ως μια σειρά από σύμβολα prompt, είτε απευθείας στο χώρο ενσωμάτωσης, ως μια σειρά από ενσωματώματα prompt. Οι παράμετροι του προ-εκπαιδευμένου μοντέλου διατηρούνται συνήθως παγωμένες, γεγονός που μειώνει σημαντικά τον αριθμό των εκπαιδεύσιμων παραμέτρων, καθιστώντας τη μάθηση βάσει prompt μια ελαφριά από άποψη υπολογιστικών πόρων και χώρου εναλλακτική.

0.3.1 Ορισμός Μεθόδου

Παραδοσιακά, σε προβλήματα ταξινόμησης, το προ-εκπαιδευμένο μοντέλο υπολογίζει την πιθανότητα:

$$P_\theta(Y|X)$$

όπου Y είναι μια ακολουθία από σύμβολα που αντιπροσωπεύουν την ετικέτα της κλάσης, X είναι το κείμενο εισόδου και το θ αντιπροσωπεύει τις παραμέτρους (βάρη) του μοντέλου, οι οποίες βελτιστοποιούνται μέσω fine-tuning.

Χρησιμοποιώντας prompts, προστίθεται ένα σύνολο παραμέτρων θ_p, ενώ οι αρχικές παράμετροι του μοντέλου θ παραμένουν συνήθως παγωμένες και το μοντέλο πρέπει τώρα να υπολογίσει την πιθανότητα:

$$P_{\theta;\theta_p}(Y|X;P)$$

όπου P είναι τα σύμβολα prompt, παραμετροποιημένα από το σύνολο παραμέτρων θ_p, τα οποία παρέχονται στο μοντέλο ως πρόσθετη είσοδος [31].

0.3.2 Κατηγορίες της Μεθόδου

Διακριτά prompts

Τα prompts που προτάθηκαν για πρώτη φορά στη βιβλιογραφία και χρησιμοποιούνται πολύ συχνά είναι διακριτά, δηλαδή αποτελούνται από ένα σύνολο συμβόλων που αντιστοιχίζονται απευθείας σε μια υπάρχουσα λέξη στο λεξιλόγιο του μοντέλου [32], [33], [34]. Στην περίπτωση αυτή, οι παράμετροι θ_p του prompt αποτελούν υποσύνολο των παραμέτρων θ του μοντέλου, και συγκεκριμένα των παραμέτρων των ενσωματωμάτων λέξεων του μοντέλου θ_{emb} ($\theta_p \subseteq \theta_{emb} \subseteq \theta$). Με διακριτά prompt η είσοδος του μοντέλου έχει συνήθως τη μορφή:

$$P_1\ P_2\ ...\ P_i\ [X]\ P_{i+1}\ ...\ P_j\ [Z]\ P_{j+1}\ ...\ P_n,$$

όπου $P_i, i = 1, 2, ..., n$ είναι τα σύμβολα prompt που το καθένα αντιστοιχεί σε μία λέξη από το λεξιλόγιο του μοντέλου, $[X]$ είναι τα σύμβολα εισόδου που προέρχονται από το κείμενο εισόδου και $[Z]$ είναι ένα ή περισσότερα αποκρυμένα σύμβολα (mask-tokens), τα οποία πρέπει να συμπληρώσει το μοντέλο, με το πρόβλημα πρόβλεψης να διατυπώνεται ως πρόβλημα Masked Language Modeling.

Soft prompts

Επειδή η διαδικασία επιλογής του βέλτιστου διακριτού prompt για κάθε πρόβλημα μπορεί να είναι δύσκολη και υπο-βέλτιστη και καθώς, σε αντίθεση με τους ανθρώπους, τα προ-εκπαιδευμένα μοντέλα μπορούν να κατανοήσουν συνεχείς παραμέτρους που δεν αντιστοιχούν άμεσα σε φυσική γλώσσα, οι ερευνητές έχουν αναπτύξει ένα δεύτερο είδος prompts, που ονομάζονται soft prompts [31] [35] [31]. Τα τελευταία αποτελούνται από συνεχείς παραμέτρους, οι οποίες μπορούν να βελτιστοποιηθούν μέσω back-propagation απευθείας στο χώρο ενσωμάτωσης, γεγονός που τις καθιστά εύκολα εκπαιδεύσιμες και πλήρως προσαρμόσιμες σε κάθε πρόβλημα και γλωσσικό μοντέλο. Θα μπορούσαν να θεωρηθούν ως "soft" (μαλακές) λέξεις, με τα ενσωματώματα prompt να βρίσκονται μεταξύ των ενσωματωμάτων πραγματικών λέξεων του λεξιλογίου ($\theta_p \supseteq \theta$). Η αρχικοποίησή τους μπορεί να γίνει με διάφορους τρόπους, όπως η τυχαία αρχικοποίηση των παραμέτρων τους, η αρχικοποίηση με τα βάρη των ενσωματωμάτων τυχαίων λέξεων από το λεξιλόγιο του γλωσσικού μοντέλου ή η αρχικοποίηση με τα βάρη των ενσωματωμάτων των ετικετών που αντιστοιχούν σε διαφορετικές κλάσεις, στις περίπτωση προβλημάτων ταξινόμησης [31]

0.3.3 Σχετική Βιβλιογραφία

Στον τομέα της μάθησης βάσει prompt με τη χρήση soft prompts, έχουν προταθεί πολλές διαφορετικές παραλλαγές τα τελευταία χρόνια. Οι Lester el al. [31], χρησιμοποιούν μια ακολουθία prompts που βρίσκονται απευθείας στο χώρο ενσωμάτωσης του μοντέλου και συνενώνονται με τα ενσωματώματα των λέξεων που παράγονται από την κειμενική είσοδο του μοντέλου. Από τα αποτελέσματά τους παρατηρούν ότι η μέθοδός τους μπορεί να είναι πολύ αποτελεσματική για μοντέλα δισεκατομμυρίων παραμέτρων, αλλά υστερεί σε απόδοση σε σχέση με το fine-tuning, όταν το γλωσσικό μοντέλο είναι μικρότερο (π.χ. 100 εκατομμύρια παράμετροι). Για το λόγο αυτό, προτάθηκε αργότερα από τους Vu et al. η μεταφορά μάθησης (transfer learning) [36]: Τα prompts εκπαιδεύτηκαν αρχικά σε διαφορετικά προβλήματα, παρόμοια με το πρόβλημα-στόχο ή σε προβλήματα που απαιτούν συλλογισμό υψηλού επιπέδου σχετικά με τις σημασιολογικές σχέσεις μεταξύ των προτάσεων και χρησιμοποιήθηκαν στη συνέχεια για την αρχικοποίηση των prompts του προβλήματος-στόχου. Όπως παρατηρούν οι συγγραφείς, η μέθοδος αυτή μπορεί να επιτρέψει την αποτελεσματική μάθηση βάσει prompt ακόμη και για μοντέλα μικρότερης κλίμακας.

Άλλοι ερευνητές έχουν πειραματιστεί με μια τροποποιημένη εκδοχή της μάθησης βάσει prompt, χρησιμοποιώντας prompts όχι μόνο στο επίπεδο εισόδου αλλά και βαθύτερα στο μοντέλο. Για παράδειγμα, οι Li και Liang [21], χρησιμοποιούν ενεργοποιήσεις προθέματος (prefix activations) που προτάσσουν σε κάθε στρώμα στον κωδικοποιητή του μοντέλου, συμπεριλαμβανομένου του στρώματος εισόδου, εργαζόμενοι σε προβλήματα παραγωγής λόγου (language generation). Οι Liu et al. [37] χρησιμοποιούν επίσης prompts σε διαφορετικά στρώματα του προ-εκπαιδευμένου μοντέλου, προσθέτοντάς τα ως προθέματα, και επεκτείνουν τη μέθοδο των Li και Liang σε προβλήματα κατανόησης φυσικής γλώσσας (Natural Language Understanding - NLU). Παρατηρούν επίσης ότι, όταν χρησιμοποιούνται πλήρη σύνολα δεδομένων, τόσο μια κεφαλή γλωσσικής μοντελοποίησης όσο και μια τυχαία αρχικοποιημένη κεφαλή ταξινόμησης μπορούν να χρησιμοποιηθούν για την πρόβλεψη των τελικών ετικετών ταξινόμησης, στην περίπτωση της μάθησης βάσει prompt.

Τέλος, ορισμένοι ερευνητές έχουν προτείνει τη συνδυαστική χρήση διακριτών και soft prompts, με στόχο τη μεγιστοποίηση της απόδοσης. Για παράδειγμα, οι Liu et al. [35], πρότειναν την μέθοδο "P-tuning", η οποία χρησιμοποιεί συνεχή prompts που αποτελούν την έξοδο ενός εκπαιδεύσιμου κωδικοποιητή για prompts. Ο κωδικοποιητής για prompts χρησιμοποιείται για τη μοντελοποίηση της εξάρτησης μεταξύ των τμημάτων του prompt και για την αποφυγή τοπικών ελαχίστων και αποτελείται από ένα αμφίδρομο LSTM, ακολουθούμενο από ένα MLP με ενεργοποίηση ReLU. Εντός του μορφότυπου του prompt, οι Liu κ.ά. προσθέτουν επίσης ορισμένα διακριτά σύμβολα-άγκυρες που σχετίζονται με το συγκεκριμένο κάθε φορά πρόβλημα (για παράδειγμα το σύμβολο "?" για το πρόβλημα RTE [38]).

0.4 Αναγνώριση συναισθήματος σε συζητήσεις

Η αναγνώριση συναισθήματος σε συζητήσεις αποτελεί ένα πρόβλημα το οποίο έχει αποκτήσει αυξανόμενη δημοτικότητα τα τελευταία χρόνια λόγω των εφαρμογών του σε τομείς όπως η υγειονομική περίθαλψη, η ανάπτυξη πρακτόρων που διαθέτουν ενσυναίσθηση, καθώς και η εξόρυξη απόψεων από τον μεγάλο όγκο συνομιλιών μεταξύ χρηστών (σε κείμενο ή βίντεο) που είναι σήμερα διαθέσιμες στα μέσα κοινωνικής δικτύωσης.

0.4.1 Ορισμός προβλήματος

Δεδομένης μιας ακολουθίας εκφωνήσεων $u_1, u_2, ..., u_N$, και του ομιλητή p_i κάθε εκφώνησης $u_i, i = 1, 2..., N$, η Αναγνώριση Συναισθήματος σε Συζητήσεις (Emotion Recognition in Conversation - ERC) ορίζεται ως το πρόβλημα της αναγνώρισης της κλάσης του συναισθήματος κάθε εκφώνησης, από ένα σύνολο προκαθορισμένων κλάσεων (όπως χαρά, ενθουσιασμός, θυμός, απογοήτευση, λύπη, ουδετερότητα κ.λπ.) [39][40].

0.4.2 Σχετική Βιβλιογραφία

Τα τελευταία χρόνια το πρόβλημα της αναγνώρισης συναισθήματος σε συζητήσεις έχει λάβει όλο και μεγαλύτερη προσοχή και έχουν αναπτυχθεί πολλαπλές μέθοδοι για την επίλυσή του. Μια γραμμή εργασίας, πιο κοντά στη δική μας, χρησιμοποιεί ακολουθιακές μεθόδους, αντιμετωπίζοντας τις εκφωνήσεις στο ERC ως μια ακολουθία μέσα στο χρόνο.

Μέθοδοι βασισμένες σε RNN και LSTM

Οι πρώτες μέθοδοι σε αυτόν τον τομέα χρησιμοποιούν LSTMs και GRUs για την κωδικοποίηση μακροχρόνιας πληροφορίας. Για παράδειγμα, οι Majumder N. et al., με το DialogueRNN [39], χρησιμοποιούν ένα GRU για την κωδικοποίηση της γενικής πληροφορίας από τα συμφραζόμενα, ενώ οι κρυφές καταστάσεις του GRU περνούν μέσα από ένα επίπεδο προσοχής (attention layer), προκειμένου να ληφθούν αναπαραστάσεις των συμφραζομένων, οι οποίες, μαζί με τις εκφωνήσεις ενός συγκεκριμένου ομιλητή, τροφοδοτούν ένα GRU ειδικό ανά ομιλητή, για να ληφθούν αναπαραστάσεις εκφωνήσεων ειδικές ανά ομιλητή. Επιπλέον, οι συγγραφείς χρησιμοποιούν ένα τρίτο GRU για να λάβουν υπόψη τους τις προηγούμενες συναισθηματικές καταστάσεις των ομιλητών, πριν τελικά ταξινομήσουν την τρέχουσα εκφώνηση.

Αντί να μοντελοποιούν διαφορετικούς ομιλητές με διαφορετικά GRUs και να ενσωματώνουν τα συμφραζόμενα και την πληροφορία που αφορά την αλληλεξαρτήσεις ομιλητών με ένα γενικό GRU όπως το DialogueRNN, οι Zhang H. και Chai Y., δουλεύοντας με δυαδικές συνομιλίες, πρότειναν μια πιο άμεση αλληλεπίδραση μεταξύ των ομιλητών με το COIN [41]: Σε αυτό χρησιμοποιούν ένα bi-GRU που κωδικοποιεί τις εκφωνήσεις ανά ομιλητή ακολουθιακά, αλλά τροποποιεί την έξοδό του σε κάθε κατεύθυνση (προς τα εμπρός και προς τα πίσω) με βάση την κρυφή κατάσταση του bi-GRU του άλλου ομιλητή. Για την κωδικοποίηση των γενικών συμφραζομένων χρησιμοποιείται μια ακολουθία στρωμάτων bi-GRU σε συνδυασμό με ένα επίπεδο προσοχής, προκειμένου να συλληφθούν οι σημαντικότερες αλληλεξαρτήσεις.

Μέθοδοι βασισμένες σε transformers

Αν και εξακολουθούν να προτείνονται μέθοδοι που βασίζονται σε RNN, αναγνωρίζεται συχνά ότι η ικανότητά τους να διατηρούν σημαντικές πληροφορίες για πολύ μεγάλες ακολουθίες είναι περιορισμένη: Καθώς προστίθενται περισσότερες εκφράσεις, οι μακροπρόθεσμες εξαρτήσεις τείνουν να ξεχαστούν [17]. Αυτό μπορεί να είναι προβληματικό στην περίπτωση της αναγνώρισης συναισθήματος σε συζητήσεις, καθώς τα πιο μακρινά συμφραζόμενα μπορούν συχνά να παρέχουν κάποια ποσότητα (αν και όχι τόσο μεγάλη όσο τα πιο πρόσφατα συμφραζόμενα) πληροφοριών σχετικά με το συναίσθημα ενός ομιλητή [39]. Επιπλέον, οι μέθοδοι που βασίζονται σε RNN βασίζονται σε εξωτερικά μοντέλα εξαγωγής ενσωματωμάτων (π.χ. Glove [42], word2vec [8]), τα οποία δεν λαμβάνουν υπόψη τα συμφραζόμενα, εκτελώντας έτσι την εξαγωγή ενσωματωμάτων και τη μοντελοποίηση της ακολουθίας ανεξάρτητα [13], κάτι που μπορεί να μην είναι πάντα βέλτιστο. Για τους λόγους αυτούς έχουν εισαχθεί οι transformers στο πρόβλημα του ERC. Παρακινούμενες από την ανάγκη καλύτερης διατήρησης των μακροπρόθεσμων εξαρτήσεων, από την ικανότητα των transformers να χειρίζονται ταυτόχρονα την εξαγωγή ενσωματωνμάτων και τη μοντελοποίηση ακολουθιών και από τις μεγάλες βελτιώσεις που επιφέρουν τα προ-εκπαιδευμένα γλωσσικά μοντέλα με βάση τους transformers σε πολλέά προβλήματα του τομέα της επεξεργασίας φυσικής γλώσσας, υπήρξαν, τα τελευταία χρόνια, μια σειρά δημοσιεύσεων που προσπαθούν να αξιοποιήσουν μεγάλα προ-εκπαιδευμένα γλωσσικά μοντέλα με βάση τους transformers για την αποτελεσματική αναγνώριση συναισθήματος σε συζητήσεις.

Σε αυτά τα μοντέλα, οι εκφωνήσεις δίνονται συνήθως ως είσοδος με ακολουθιακή σειρά. Για την προσαρμογή του μοντέλου στο περιβάλλον του διαλόγου, χρησιμοποιούνται στη συνέχεια πρόσθετα στρώματα ή/και ενσωματώματα ή τροποποιούνται τα υπάρχοντα στρώματα, ώστε να συμπεριληφθούν πληροφορίες σημαντικές για το πρόβλημα της αναγνώρισης συναισθήματος. Μια άλλη δημοφιλής χρήση προ-εκπαιδευμένων μοντέλων που βασίζονται σε transformers είναι η εξαγωγή ενσωματωμάτων που κωδικοποιούν σε κάποιο βαθμό τα συμφραζόμενα, και η χρήση αυτών ως αναπαραστάσεις των εκφωνήσεων, οι οποίες μπορούν στη συνέχεια να χρησιμοποιηθούν σε μια δεύτερη, κύρια μονάδα, είτε ακολουθιακή είτε βασισμένη σε γράφους. για την τελική αναγνώριση του συναισθήματος.

Για παράδειγμα, με το Hitrans, οι Li, J. et al. [11] χρησιμοποιούν έναν προ-εκπαιδευμένο transformer κωδικοποιητή, για να λάβουν αναπαραστάσεις που κωδικοποιούν τα τοπικά συμφραζόμενα και προσθέτουν έναν δεύτερο transformer σε υψηλότερο επίπεδο, για να συμπεριλάβουν τα πιο μακρινά συμφραζόμενα, ο οποίος λαμβάνει ως είσοδο τις τοπικές αναπαραστάσεις. Χρησιμοποιούν επίσης εκμάθηση πολλαπλών εργασιών (multitask learning) για να συμπεριλάβουν πληροφορίες σχετικά με τον ομιλητή, προσθέτοντας ένα δεύτερο πρόβλημα, το οποίο ονομάζουν Pairwise Utterance Speaker Verification (ταυτοποίηση ομιλητών ανά ζεύγη), (ο στόχος του είναι να αποφανθεί το μοντέλο αν δύο εκφωνήσεις ανήκουν στον ίδιο ομιλητή), και χρησιμοποιώντας έναν biaffine ταξινομητή με είσοδο τις αναπαραστάσεις του transformer υψηλού επιπέδου.

Οι Shen W.et al., με το DialogXL [43], προσπαθούν να διατηρήσουν μια απλούστερη αρχιτεκτονική αξιοποιώντας ένα προ-εκπαιδευμένο μοντέλο βασισμένο σε transformer και κωδικοποιώντας όλες τις χρήσιμες πληροφορίες μέσω της αυτο-προσοχής πολλαπλών κεφαλών (multi-head self-attention) σε κάθε επίπεδο. Εφαρμόζουν διαφορετικούς μηχανισμούς απόκρυψης (masking) και χρησιμοποιούν την γενική αυτο-προσοχή (global self-attention) για εξαρτήσεις μεγάλης εμβέλειας, την τοπική αυτο-προσοχή (local self-attention) για τα συμφραζόμενα πιο κοντά στην τρέχουσα εκφώνηση, (δεδομένου ότι αυτά είναι συνήθως πιο σημαντικά για τον προσδιορισμό του τρέχοντος συναισθήματος), και τέλος μια αυτο-προσοχή ομιλητή (speaker self-attention) και μια αυτο-προσοχή ακροατή (listener self-attention) για την καταγραφή των εξαρτήσεων εντός και μεταξύ των ομιλητών.

Χρησιμοποιώντας και πάλι ένα προ-εκπαιδευμένο γλωσσικό μοντέλο βασισμένο σε transformer, οι Kim T. και Vossen P. [13] δημιούργησαν ένα ακόμα απλούστερο μοντέλο, το EMOBERTa, τοποθετώντας το όνομα του ομιλητή πριν από κάθε εκφώνηση και θέτοντας το προκύπτον κείμενο ως είσοδο σε ένα μοντέλο που βασίζεται στο BERT, προσθέτοντας μόνο ένα τυχαία αρχικοποιημένο γραμμικό επίπεδο (linear layer) ως επίπεδο εξόδου του μοντέλου τους.

Τέλος, οι Zhu, L. et al. [16], με το TODKAT, χρησιμοποιούν ως βάση τους ένα προ-εκπαιδευμένο γλωσσικό μοντέλο βασισμένο σε transformer, ενώ ενσωματώνουν πληροφορία για το θέμα της συζήτησης στην αρχιτεκτονική τους, μέσω ενός στρώματος ειδικού για το θέμα και πραγματοποιώντας fine-tuning. Προτείνουν επίσης την ενσωμάτωση της γνώσης κοινής λογικής (commonsense knowledge), λαμβάνοντας τις πιο παρόμοιες αναπαραστάσεις για κάθε εκφώνηση από μια μεγάλη βάση γνώσης και εφαρμόζοντας τον μηχανισμό προσοχής για την ενσωμάτωση της πιο χρήσιμης γνώσης.

0.5 Προτεινόμενες προσεγγίσεις

0.5.1 Συγκρίνοντας τη μάθηση βάσει prompt και το fine-tuning στο ERC

Εισαγωγή

Αν και έχουν προταθεί πολλαπλές παραλλαγές για τη χρήση prompts, η προσαρμογή αυτών των μεθόδων για συγκεκριμένα προβλήματα δεν έχει ερευνηθεί τόσο πολύ και υπάρχει περιορισμένη βιβλιογραφία που χρησιμοποιεί prompts σε μοντέλα ειδικά για κάποιο πρόβλημα. Η προσέγγισή μας αποσκοπεί επομένως στη μελέτη της δυνατότητας εφαρμογής της μάθησης βάσει prompt στο πρόβλημα του ERC, συγκριτικά με το fine-tuning και στην θεμελίωση ενός μέτρου σύγκρισης (baseline) για τη μάθηση βάσει prompt για την Αναγνώριση Συναισθήματος σε Συζητήσεις (ERC). Για τον σκοπό αυτό, πειραματιζόμαστε με μια πολύ απλή, βασική αρχιτεκτονική, καθώς και με μεθόδους που χρησιμοποιούνται συνήθως σε προηγούμενα έργα στον τομέα του ERC, όπως η ενσωμάτωση πληροφοριών που αφορούν τον ομιλητή, μέσω της εισόδου ή μέσω ενός βοηθητικού προβλήματος που σχετίζεται με τον ομιλητή. Αξιοποιούμε ένα προ-εκπαιδευμένο γλωσσικό μοντέλο και τροποποιούμε την αρχιτεκτονική του για να το προσαρμόσουμε τόσο σε ένα περιβάλλον μάθησης βάσει prompt όσο και σε ένα περιβάλλον fine-tuning, για το πρόβλημα του ERC.

Αρχικά συγκρίνουμε τις δύο μεθόδους προσαρμογής (fine-tuning και μάθηση βάσει prompt) χρησιμοποιώντας ένα πολύ απλό μοντέλο, θέτοντας έτσι το βασικό μας μέτρο σύγκρισης (baseline), και μελετάμε την επίδραση της χρήσης ενός prompt, καθώς και του μεγέθους αυτού. Στη συνέχεια πειραματιζόμαστε με την πρόσθεση των ονομάτων των ομιλητών στην είσοδο πριν από τις αντίστοιχες εκφωνήσεις. Πρόκειται για μια τεχνική που χρησιμοποιείται συχνά σε προηγούμενα μοντέλα για ERC [13] [14], καθώς επιτρέπει την ενσωμάτωση πληροφορίας που αφορά τους ομιλητές και βοηθά το μοντέλο να αναγνωρίζει ευκολότερα τις ενδοεξαρτήσεις και αλληλεξαρτήσεις ομιλητών. Όταν όμως προστίθενται τα ονόματα ομιλητών στις εκφωνήσεις, η μορφή της εισόδου του μοντέλου αλλάζει: δεν αποτελεί πλέον μια συνεχή ακολουθία προτάσεων, αφού αυτή διακόπτεται από ονόματα ομιλητών. Καθώς αυτή η μορφή εισόδου είναι διαφορετική από εκείνη που χρησιμοποιήθηκε στο στάδιο προ-εκπαίδευσης του γλωσσικού μας μοντέλου, το προ-εκπαιδευμένο μοντέλο πρέπει να προσαρμοστεί σε αυτή προκειμένου να αξιοποιήσει τις πρόσθετες πληροφορίες. Κατά το fine-tuning, το μοντέλο μπορεί να τροποποιήσει τις παραμέτρους του ανάλογα και έτσι να προσαρμοστεί αποτελεσματικά. Ωστόσο, όταν το μοντέλο είναι παγωμένο, όπως συμβαίνει συνήθως στη μάθηση βάσει prompt, αυτό δεν είναι δυνατό. Το μοντέλο μπορεί να αλλάξει μόνο τον τρόπο με τον οποίο ερμηνεύει την έξοδο του βασικού τμήματος του προ-εκπαιδευμένου μοντέλου (όταν χρησιμοποιείται μία εκπαιδεύσιμη μονάδα εξόδου), ή, μέσω των παραμέτρων του prompt, τον τρόπο με τον οποίο αλληλεπιδρούν τα διάφορα μέρη της εισόδου του. Επομένως, ο πειραματισμός με την προσθήκη των ονομάτων ομιλητών μπορεί να μας προσφέρει μια καλύτερη εικόνα για το βαθμό προσαρμοστικότητας που μπορούν να προσφέρουν τα prompts για ένα σύνθετο πρόβλημα όπως το ERC. Τέλος, πειραματιζόμαστε με τη χρήση ενός βοηθητικού προβλήματος αναγνώρισης ομιλητή, μέσω εκμάθησης πολλαπλών εργασιών (multi-task learning). Η ενσωμάτωση πληροφοριών που αφορούν τον ομιλητή μέσω εκμάθησης πολλαπλών εργασιών έχει προταθεί σε έργα όπως τα [11] [15] και έχει διαπιστωθεί ότι είναι αποτελεσματική σε ένα περιβάλλον fine-tuning. Ωστόσο, σε ένα περιβάλλον όπου οι εκπαιδεύσιμες παράμετροι είναι περιορισμένες και βρίσκονται στο επίπεδο εισόδου και όχι στο βασικό τμήμα του μοντέλου, όπως συμβαίνει στην περίπτωση της μάθησης βάσει prompt, η ικανότητα προσαρμογής του μοντέλου για τη συνδυαστική επίλυση δύο προβλημάτων δεν είναι εμφανής και αξίζει να διερευνηθεί.

Μέθοδος

Χρησιμοποιούμε το προ-εκπαιδευμένο γλωσσικό μοντέλο BERT ως το βασικό μας μοντέλο. Η βασική αρχιτεκτονική του μοντέλου μας μπορεί να χωριστεί σε τρεις κύριες μονάδες, την μονάδα ενσωμάτωσης (embedding), τη μονάδα κωδικοποιητή και την μονάδα εξόδου. Η μονάδα ενσωμάτωσης είναι η μόνη που διαφέρει μεταξύ της εκμάθησης βάσει prompt και του fine-tuning. Για τη μάθηση βάσει prompt, ακολουθούμε την αρχιτεκτονική που προτείνεται στο [31], κωδικοποιώντας τα prompts απευθείας στον χώρο ενσωμάτωσης του μοντέλου μας και χρησιμοποιώντας μια σειρά από ενσωματώματα prompt (prompt embeddings), τα οποία συνενώνουμε με τα ενσωματώματα που υπολογίζονται από το τμήμα ενσωμάτωσης του προ-εκπαιδευμένου μοντέλου μας και αντιστοιχούν στο κείμενο εισόδου. Στην περίπτωση του fine-tuning, το στρώμα ενσωμάτωσης του προ-εκπαιδευμένου μοντέλου μας χρησιμοποιείται αμετάβλητο, οπότε υπολογίζουμε μόνο τα ενσωματώματα που αντιστοιχούν στην είσοδο κειμένου, και όχι τα ενσωματώματα prompt. Η μονάδα κωδικοποιητή και η μονάδα εξόδου παραμένουν αμετάβλητα και για τις δύο μεθόδους προσαρμογής: Η πρώτη αποτελείται από 12 προ-εκπαιδευμένα στρώματα transformer κωδικοποιητή (πρόκειται για τον κωδικοποιητή BERT), ενώ για τη δεύτερη χρησιμοποιούμε δύο γραμμικά επίπεδα (linear layers) με ένα επίπεδο dropout μεταξύ τους, προκειμένου να αντιστοιχίσουμε την αναπαράσταση εξόδου του κωδικοποιητή σε μια ετικέτα (label). Εκτελούμε τα πειράματά μας σε δύο σύνολα δεδομένων για ERC, το MELD και το IEMOCAP.

Προκειμένου να μελετήσουμε την απόδοση της χρήσης prompt σε σύγκριση με το fine-tuning, συγκρίνουμε το μοντέλο μας με ένα παρόμοιο fine-tuned μοντέλο χωρίς παγωμένες παραμέτρους, του οποίου η μόνη διαφορά είναι πως δεν διαθέτει prompt στο στρώμα ενσωμάτωσης. Επιπλέον, αναπτύσσουμε ένα δεύτερο μοντέλο για λόγους σύγκρισης, ίδιο με το fine-tuned μοντέλο, αλλά με όλες τις παραμέτρους του παγωμένες, εκτός από τις παραμέτρους της κεφαλής ταξινόμησης (μονάδας εξόδου). Στόχος μας είναι να κατανοήσουμε πλήρως τη συμβολή των prompts στην παρατηρούμενη απόδοση των προτεινόμενων μοντέλων μας, διαχωρίζοντάς την από τη συμβολή της κεφαλής ταξινόμησης.

Στις επόμενες παραγράφους παρουσιάζουμε μια επισκόπηση της αρχιτεκτονικής των προτεινόμενων από εμάς μοντέλων που βασίζονται σε prompt. Μια σχηματική επισκόπηση του μοντέλου που χρησιμοποιεί prompt, στην απλή περίπτωση χωρίς πληροφορίες για τον ομιλητή, μπορεί να δει κανείς στο Σχήμα 1.

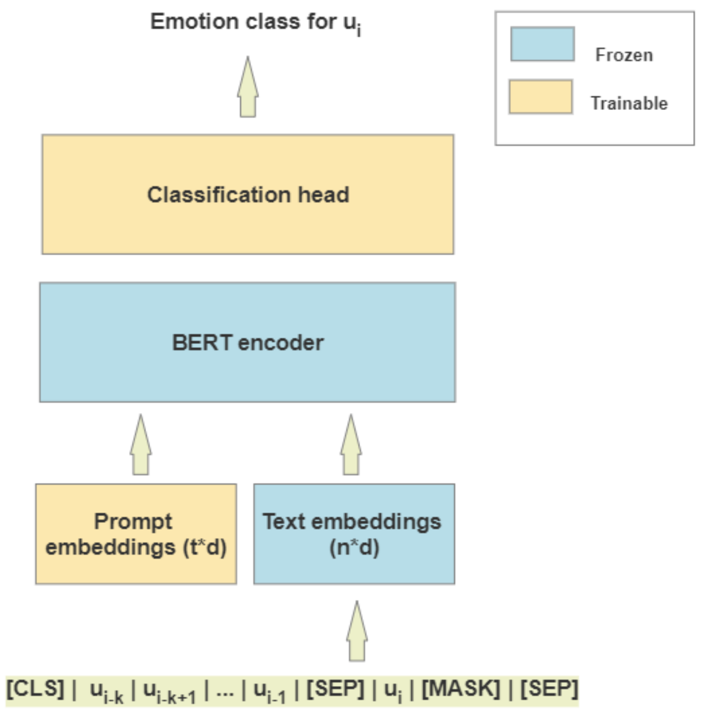


Figure 1. *Γενικό διάγραμμα του μοντέλου μας που χρησιμοποιεί prompt στην περίπτωση που δεν προστίθενται ονομάτα ομιλητών. t είναι ο αριθμός των συμβόλων prompt, d είναι η διάσταση ενσωμάτωσης του BERT (768), n είναι ο αριθμός των ενσωματωμάτων που αντιστοιχούν στο κείμενο εισόδου μας, k είναι ο αριθμός των προηγούμενων εκφωνήσεων που χρησιμοποιούνται ως συμφραζόμενα και i ο δείκτης της τρέχουσας εκφώνησης που πρέπει να ταξινομηθεί. Το " | " συμβολίζει τη συνένωση. Οι μονάδες με μπλε χρώμα διατηρούνται παγωμένες κατά τη διάρκεια της εκπαίδευσης, ενώ οι μονάδες με κίτρινο χρώμα είναι εκπαιδεύσιμες. Σημειώνεται ότι η μονάδα ενσωμάτωσης prompt (prompt embedding) διαθέτει εκπαιδεύσιμα μέρη, αλλά δεν είναι εκπαιδεύσιμη στο σύνολό της: Βλ. Εικόνα 2 για μια λεπτομερή επισκόπηση.*

Μορφότυπο κειμένου εισόδου

Χρησιμοποιούμε το μορφότυπο εισόδου δύο προτάσεων του BERT. Ως πρώτη πρόταση, συνενώνουμε τις k πιο πρόσφατες εκφωνήσεις πριν από την τρέχουσα εκφώνηση που πρόκειται να ταξινομηθεί, ώστε το μοντέλο να τις αξιοποιήσει ως συμφραζόμενα. Ως δεύτερη πρόταση ορίζουμε την τρέχουσα εκφώνση προς ταξινόμηση, ακολουθούμενη από ένα σύμβολο $[MASK]$. Η μετατροπή του προβλήματος ταξινόμησης σε πρόβλημα Masked LM φέρνει τη μορφή της εισόδου πιο κοντά στο στάδιο προ-εκπαίδευσης του γλωσσικού μας μοντέλου και διαπιστώσαμε ότι βελτιώνει την απόδοση. Διαχωρίζουμε τις δύο προτάσεις με το σύμβολο $[SEP]$ και προσθέτουμε ένα σύμβολο $[CLS]$ στην αρχή της ακολουθίας εισόδου μας και ένα σύμβολο $[SEP]$ στο τέλος.

Στην περίπτωση της προσθήκης του ονόματος ομιλητή, αλλάζουμε ελαφρώς τη μορφή εισόδου, αντιστοιχίζοντας κάθε ομιλητή του συνόλου δεδομένων μας σε ένα από τα αχρησιμοποίητα σύμβολα (tokens) του BERT και προσθέτοντας το αντίστοιχο σύμβολο (token) του ομιλητή πριν από κάθε εκφώνησή του. Θεωρούμε ότι αυτό αποτελεί μια καλύτερη εναλλακτική λύση από το να προσθέσουμε τα πραγματικά ονόματα των ομιλητών, καθώς αυτά μπορεί να μην υπάρχουν στο λεξιλόγιο του BERT, μπορεί να χωριστούν σε πολλαπλά tokens από τον tokenizer του BERT, ενώ μπορεί ακόμη και να μην υπάρχουν στο σύνολο δεδομένων (για παράδειγμα, οι ομιλητές δεν έχουν ονόματα στο IEMOCAP). Με την αντιστοίχιση των ομιλητών σε αχρησιμοποίητα σύμβολα, επιτυγχάνουμε μια αντιστοίχιση $1-1$ μεταξύ ομιλητών και συμβόλων, εφαρμόσιμη για όλα τα ονόματα ομιλητών και σύνολα δεδομένων.

Τέλος, χρησιμοποιούμε ένα ελαφρώς διαφορετικό μορφότυπο εισόδου στην περίπτωση εκμάθησης πολλαπλών εργασιών, για τα προβήματα ERC και αναγνώρισης ομιλητή: Επειδή η τρέχουσα εκφώνηση είναι συχνά πολύ σύντομη ή γενική και τα συμφραζόμενα του διαλόγου αποτελούνται από εκφωνήσεις πολλών ομιλητών, χρησιμοποιούμε δύο από τα αχρησιμοποίητα σύμβολα του BERT, αντιστοιχίζοντας το ένα από αυτά (το συμβολίζουμε ως "un_0") στον τρέχοντα ομιλητή και το άλλο (το συμβολίζουμε ως "un_1") σε όλους τους άλλους ομιλητές του διαλόγου. Με αυτόν τον τρόπο επιτρέπουμε στο μοντέλο να προσδιορίσει όλες τις εκφωνήσεις του ομιλητή που προσπαθεί να αναγνωρίσει. Προτάσσουμε το "u_0" σε όλες τις εκφωνήσεις του τρέχοντος ομιλητή και το "un_1" σε όλες τις άλλες. Επιπλέον, προσθέτουμε ένα δεύτερο σύμβολο $[MASK]$ στο τέλος της εισόδου μας, για να λάβουμε την πρόβλεψη του μοντέλου για την ταυτότητα του ομιλητή.

Στη συνέχεια, η είσοδος κειμένου μετατρέπεται σε σύμβολα (tokens) με τη χρήση του BertTokenizer και τα σύμβολα εισόδου (n συνολικά) περνούν από το προ-εκπαιδευμένο στρώμα ενσωμάτωσης του BERT, με αποτέλεσμα να προκύπτουν n ενωματώματα (embeddings). Σημειώνουμε ότι ο αριθμός k των εκφωνήσεων που θα χρησιμοποιηθούν ως συμφραζόμενα υπολογίζεται έτσι ώστε ο αριθμός n των ενσωματωμάτων που προκύπτει να έχει τη μέγιστη τιμή $512 - t$, όπου 512 είναι το μέγιστο μέγεθος εισόδου του BERT και t είναι ο αριθμός των συμβόλων prompt (prompt tokens) που θα χρησιμοποιηθούν, όπως εξηγείται στην επόμενη παράγραφο.

Σύμβολα prompt (prompt tokens)

Ακολουθώντας το [31], χρησιμοποιούμε μια ακολουθία από σύμβολα prompt (prompt tokens) που βρίσκονται απευθείας στο χώρο ενσωμάτωσης (τα ονομάζουμε ενσωματώματα συμβόλων prompt - prompt token embeddings), οδηγώντας σε ένα prompt διαστάσεων $t * d$, όπου το t αντιπροσωπεύει τον αριθμό των συμβόλων prompt και το d αντιπροσωπεύει τη διάσταση των ενσωματωμάτων (768 για το bert-base). Αρχικοποιούμε τα ενσωματώματα συμβόλων prompt με τα βάρη των ενσωματωμάτων των συμβόλων που αντιστοιχούν στις ετικέτες κλάσεων του συνόλου δεδομένων (π.χ. χαρά, λύπη κ.λπ.). Καθώς τα βάρη των ενσωματωμάτων συμβόλων prompt αλλάζουν κατά τη διάρκεια της εκπαίδευσης με τη χρήση back-propagation, δεν αντιστοιχούν πλέον σε πραγματικά σύμβολα (tokens) από το λεξιλόγιο του BERT, αλλά θα μπορούσαν να θεωρηθούν ως ένα είδος "soft" συμβόλων, που βρίσκονται βέλτιστα μεταξύ των λέξεων του BERT.

Προσθήκη ενσωματωμάτων θέσης και τμήματος

Όπως αναλύθηκε προηγουμένως (βλ. υποενότητα 0.2.3), εξετάζοντας την αρχιτεκτονική του στρώματος ενσωμάτωσης του BERT, μπορούμε να δούμε ότι χρησιμοποιούνται τρεις τύποι ενσωματωμάτων: ενσωματώματα συμβόλων (token embeddings), ενσωματώματα θέσης (position embeddings) και ενσωματώματα τμήματος (segment embeddings). Επομένως, στο επίπεδο ενσωμάτωσης, προσθέτουμε τα προ-εκπαιδευμένα ενσωματώματα θέσης και τμήματος του BERT στα ενσωματώματα συμβόλων prompt. Τα ενσωματώματα θέσης και τμήματος προστίθενται επίσης στα ενσωματώματα συμβόλων που αντιστοιχούν στην είσοδο κειμένου μας, από το προ-εκπαιδευμένο στρώμα ενσωμάτωσης του BERT. Στις επόμενες παραγράφους θα αναφερόμαστε στην τελική έξοδο ενσωμάτωσης μετά από αυτή την άθροιση ως ενσωματώματα prompt (prompt embeddings) στην περίπτωση του prompt και ενσωματώματα κειμένου (text embeddings) στην περίπτωση της εισόδου κειμένου.

Τελική έξοδος στρώματος ενσωμάτωσης

Έχοντας λάβει τόσο τα ενσωματώματα prompt όσο και τα ενσωματώματα κειμένου, τα συνενώνουμε, με αποτέλεσμα να λάβουμε $t+n$ ενσωματώματα, και περνάμε τις συνενωμένες αυτές αναπαραστάσεις ως είσοδο στη μονάδα κωδικοποιητή BERT. Το Σχήμα 2 παρέχει μια σχηματική επισκόπηση της εξόδου του στρώματος ενσωμάτωσης.

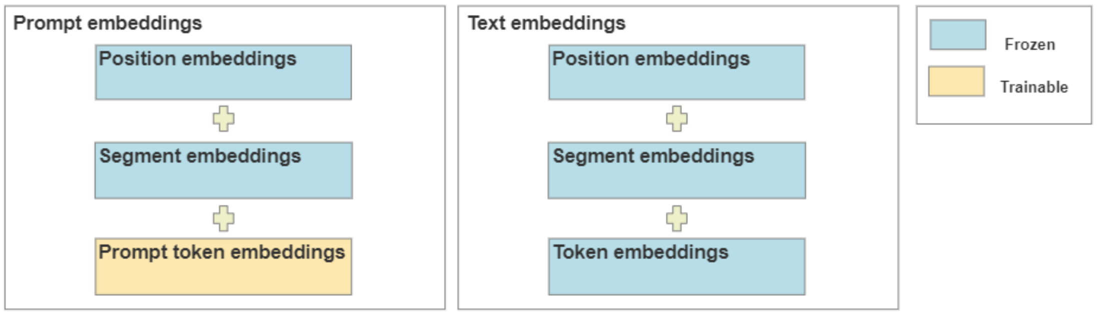
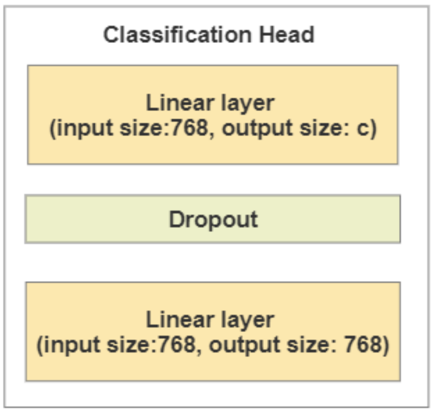


Figure 2. *Επισκόπηση του στρώματος ενσωμάτωσης για την περίπτωση μάθησης βάσει prompt. Η τελική έξοδος του στρώματος ενσωμάτωσης είναι μια συνένωση των ενσωματωμάτων prompt και κειμένου. Οι μονάδες με μπλε χρώμα διατηρούνται παγωμένες κατά τη διάρκεια της εκπαίδευσης, ενώ οι μονάδες με κίτρινο χρώμα είναι εκπαιδεύσιμες.*

Κεφαλή ταξινόμησης

Για να λάβουμε μια ετικέτα ταξινόμησης για κάθε δείγμα, χρησιμοποιούμε μια κεφαλή ταξινόμησης ως μονάδα εξόδου, που αποτελείται από δύο γραμμικά επίπεδα (linear layers) και ένα επίπεδο dropout μεταξύ τους, και περνάμε ως είσοδο στην κεφαλή ταξινόμησης την τελευταία κρυφή αναπαράσταση του συμβόλου $[MASK]$ της εισόδου του μοντέλου. Η κεφαλή ταξινόμησης ενεργεί επομένως ως συνάρτηση αντιστοίχισης (mapping function), αντιστοιχίζοντας την τελική αναπαράσταση του μοντέλου για το σύμβολο $[MASK]$ σε μία από τις ετικέτες συναισθήματος του συνόλου δεδομένων μας. Ένα σχηματικό διάγραμμα της κεφαλής ταξινόμησης που περιγράφηκε μπορεί κανείς να δει στο Σχήμα 3, όπου παρουσιάζονται επίσης οι διαστάσεις κάθε στρώματος. Σημειώνουμε ότι στην περίπτωση εκμάθησης πολλαπλών εργασιών (multitask learning), χρησιμοποιούμε μια δεύτερη κεφαλή ταξινόμησης για το πρόβλημα αναγνώρισης ομιλητή, ίδια με αυτή που περιγράψαμε μέχρι τώρα.



Classification Head
Linear layer
(input size:768, output size: c)
Dropout
Linear layer
(input size:768, output size: 768)

Figure 3. *Διάγραμμα κεφαλής ταξινόμησης, όπου το c συμβολίζει τον αριθμό κλάσεων στο σύνολο δεδομένων μας.*

Παγωμένα και εκπαιδεύσιμα τμήματα

Για να μελετήσουμε την επίδραση της μάθησης βάσει prompt χωρίς κανένα fine-tuning, παγώνουμε όλες τις παραμέτρους του μοντέλου, εκτός από τις παραμέτρους της κεφαλής ταξινόμησης και του prompt, τις οποίες εκπαιδεύουμε μέσω back-propagation. Πρέπει να σημειώσουμε ότι κρατάμε παγωμένες τις παραμέτρους των ενσωματωμάτων τμήματος και θέσης τόσο για την κειμενική είσοδο όσο και για τα prompts, ενώ επίσης δεν εκπαιδεύουμε τα ενσωματώματα συμβόλων του BERT, τα οποία χρησιμοποιούνται για την μετάβαση της κειμενικής εισόδου στο χώρο ενσωμάτωσης.

Σύνολο δεδομένων

Στην περίπτωση του απλού μοντέλου που χρησιμοπειεί prompt, χωρίς πληροφορίες σχετικά με τον ομιλητή, καθώς και του μοντέλου που χρησιμοποιεί prompt με παράλληλη προσθήκη του ονόματος του ομιλητή, πειραματιζόμαστε τόσο στο MELD όσο και στο IEMOCAP. Στην περίπτωση της εκμάθησης πολλαπλών εργασιών με το βοηθητικό πρόβλημα της αναγνώρισης ομιλητή πειραματιζόμαστε μόνο στο MELD: Όπως αναλύεται στην υποενότητα 5.5, στο MELD υπάρχουν έξι κύριοι ομιλητές, με περισσότερες από 60 εκφωνήσεις ο καθένας, και όλοι οι άλλοι ομιλητές έχουν πολύ λίγες εκφωνήσεις. Επομένως, δημιουργούμε επτά κλάσεις για το πρόβλημα αναγνώρισης ομιλητών, μία για κάθε κύριο ομιλητή και μία με τον χαρακτηρισμό "Other", στην οποία κατατάσσονται όλοι οι υπόλοιποι ομιλητές του συνόλου δεδομένων. Αυτό μετατρέπει το πρόβλημα αναγνώρισης ομιλητή σε πρόβλημα ταξινόμησης επτά κλάσεων. Δεν δοκιμάζουμε τη μέθοδό μας στο IEMOCAP, καθώς αυτό αποτελεί ένα σύνολο δεδομένων με μεγάλο μέρος του βασισμένο σε σενάρια, με τους ομιλητές να μην διαθέτουν μία συγκεκριμένη προσωπικότητα, αλλά να είναι ηθοποιοί που παίζουν διαφορετικούς ρόλους σε διαφορετικά σενάρια. Επιπλέον, όλοι οι ομιλητές του συνόλου ελέγχου (test) είναι διαφορετικοί από αυτούς του συνόλου εκπαίδευσης (train), το οποίο σημαίνει πως το μοντέλο δεν μπορεί να μάθει στοιχεία για συγκεκριμένους ομιλητές από το σύνολο εκπαίδευσης. Για το λόγο αυτό, πιστεύουμε πως δεν έχει νόημα να προσπαθήσουμε να αναγνωρίσουμε κάθε συγκεκριμένο ομιλητή στο IEMOCAP και, ως εκ τούτου, δεν εκτελούμε εκμάθηση πολλαπλών εργασιών σε αυτό.

Αποτελέσματα και συζήτηση

Ο Πίνακας 1 απεικονίζει το σταθμισμένο (weighted) F1-score για το MELD και το IEMOCAP στην περίπτωση που δεν χρησιμοποιείται η προσθήκη του ονόματος ομιλητή. Όλα τα αποτελέσματά μας έχουν προκύψει ως ο μέσος όρος τριών εκτελέσεων.

Table 1. *Σταθμισμένο F1 score (%) για τα σύνολα δεδομένων ελέγχου MELD και IEMOCAP*

Μοντέλο	MELD	IEMOCAP
Fine-tuned baseline	57.06	64.44
Baseline με μόνο κεφαλή ταξινόμησης	54.90	56.46
Με χρήση prompt, 7 (στο MELD) / 6 (στο IEMOCAP) σύμβολα prompt	56.23	58.95
Με χρήση prompt, 30 σύμβολα prompt	56.70	59.94
Με χρήση prompt, 40 σύμβολα prompt	56.94	59.21
Με χρήση prompt, 60 σύμβολα prompt	56.53	59.24

Παρατηρώντας τον Πίνακα 1 μπορούμε να δούμε ότι, για το MELD, το baseline μοντέλο που χρησιμοποιεί fine-tuning οδηγεί σε F1-score ίσο με 57,06%, ενώ το καλύτερο από τα μοντέλα που βασίζονται σε prompts, το μοντέλο με 40 σύμβολα prompt, οδηγεί σε F1-score ίσο με 56,94%. Αυτό σημαίνει ότι, για το σύνολο δεδομένων MELD, το μοντέλο με χρήση prompt είναι συγκρίσιμο με το fine-tuned μοντέλο. Ωστόσο, για το σύνολο δεδομένων IEMOCAP, το baseline μοντέλο με fine-tuning επιτυγχάνει υψηλότερο F1-score από εκείνο του μοντέλου με την καλύτερη επίδοση που χρησιμοποιεί prompt (και συγκεκριμένα 30 σύμβολα prompt), κατά 4,50%, γεγονός που υποδηλώνει την υπεροχή της μεθόδου fine-tuning σε αυτή την περίπτωση.

Ένας λόγος για τη διαφορά μεταξύ των δύο συνόλων δεδομένων θα μπορούσε να είναι η προέλευσή τους: Ενώ οι εκφωνήσεις του IEMOCAP έχουν σχεδιαστεί με σκοπό να προκαλέσουν ένα συγκεκριμένο συναίσθημα και εκφωνούνται σε ένα σενάριο με δύο ομιλητές, το MELD προέρχεται από μια τηλεοπτική σειρά, όπου μιλούν πολλοί ομιλητές και το συναίσθημα κάθε εκφώνησης μπορεί να είναι πιο ήπιο ή να εκφράζεται με πολύ διαφορετικούς τρόπους από διαφορετικούς ομιλητές και σε διαφορετικές καταστάσεις. Ο αριθμός των ομιλητών του MELD είναι επίσης σημαντικά μεγαλύτερος από αυτόν του IEMOCAP. Θα μπορούσαμε επομένως να υποθέσουμε ότι το fine-tuning, το οποίο τροποποιεί εκατομμύρια παραμέτρους σύμφωνα με τα δεδομένα και επιτρέπει στο μοντέλο να προσαρμόζεται πιο στενά στα δεδομένα, έχει πλεονέκτημα στο IEMOCAP, σε σύγκριση με την λιγότερο προσαρμοστική μάθηση βάσει prompt. Από την άλλη πλευρά, στο MELD, λόγω της μεγαλύτερης ποικιλομορφίας και δυσκολίας του, η στενότερη προσαρμογή του μοντέλου στα δεδομένα εκπαίδευσης δεν οδηγεί απαραίτητα σε καλύτερη ικανότητα γενίκευσης και, επομένως, το fine-tuning δεν έχει το ίδιο ισχυρό πλεονέκτημα.

Προσθήκη συμβόλων ομιλητή

Ο Πίνακας 2 παρουσιάζει το σταθμισμένο (weighted) F1-score για το MELD και το IEMOCAP, στην περίπτωση που χρησιμοποιείται η προσθήκη συμβόλων ομιλητή. Όλα τα αποτελέσματά μας αποτελούν μέσο όρο τριών εκτελέσεων.

Table 2. *Σταθμισμένο F1-score (%) για τα σύνολα δεδομένων ελέγχου MELD και IEMOCAP*

Μοντέλο	MELD	IEMOCAP
Με χρήση prompt, 7 (στο MELD) / 6 (στο IEMOCAP) σύμβολα prompt	56.23	58.95
Με χρήση prompt, 30 σύμβολα prompt	56.70	59.94
Fine-tuned, με σύμβολα ομιλητή	56.14	64.99
Μόνο κεφαλή ταξινόμησης, με σύμβολα ομιλητή	54.90	56.16
Με χρήση prompt και σύμβολα ομιλητή, 7 (στο MELD) / 6 (στο IEMOCAP) σύμβολα prompt	56.64	59.04
Με χρήση prompt και σύμβολα ομιλητή, 30 σύμβολα prompt	57.63	59.12

Συγκρίνοντας τα αποτελέσματα των απλών μοντέλων με χρήση prompt και των μοντέλων με χρήση prompt που χρησιμοποιούν επιπλέον την προσθήκη συμβόλων ομιλητή, βλέπουμε ότι η τελευταία τείνει να βελτιώνει την απόδοση (με εξαίρεση την περίπτωση του μοντέλου που χρησιμοποιεί 30 σύμβολα prompt για το IEMOCAP, όπου παρατηρείται μια μικρή πτώση της απόδοσης). Συγκεκριμένα για το MELD, μπορούμε να δούμε από τον Πίνακα 2 ότι, όταν χρησιμοποιείται η προθήκη των συμβόλων ομιλητή, το F1-score αυξάνεται κατά 0,41% στην περίπτωση των 6 συμβόλων prompt και κατά 0,93% στην περίπτωση των 30 συμβόλων prompt, γεγονός που υποδηλώνει ότι ο μεγαλύτερος αριθμός παραμέτρων στα σύμβολα prompt επιτρέπει στο μοντέλο να προσαρμοστεί στη νέα μορφή εισόδου και να ενσωματώσει πιο αποτελεσματικά τις πληροφορίες που αφορούν τον ομιλητή. Ωστόσο, για το IEMOCAP, μπορούμε να δούμε ότι, αν και το μοντέλο με 30 σύμβολα prompt που χρησιμοποιεί προσθήκη συμβόλων ομιλητή αποδίδει καλύτερα σε σύγκριση με εκείνο με 6 σύμβολα prompt που χρησιμοποιεί σύμβολα ομιλητή, η προσθήκη συμβόλων ομιλητή έχει συνολικά αρνητική επίδραση, με το baseline μοντέλο με 30 σύμβολα prompt και χωρίς πληροφορία ομιλητή να επιτυγχάνει υψηλότερη απόδοση κατά 0,90%. Παρομοίως, παρατηρούμε ότι για το μοντέλο που χρησιμοπεί 6 σύμβολα prompt, η προσθήκη των συμβόλων ομιλητή επίσης δεν οδήγησε σε αξιοσημείωτη βελτίωση των επιδόσεων. Εξετάζοντας τα fine-tuned μοντέλα ως σύγκριση (Πίνακες 1 και 2), η προσθήκη συμβόλων ομιλητή στην περίπτωση του MELD οδηγεί σε πτώση της απόδοσης κατά 0,92%, ενώ στην περίπτωση του IEMOCAP αυξάνει την απόδοση 0,55%.

Τα παραπάνω αποτελέσματα υποδηλώνουν ότι η κωδικοποίηση πληροφοριών για τον εκάστοτε ομιλητή μπορεί πράγματι να είναι χρήσιμη για το πρόβλημα του ERC, όπως προτείνεται και στην βιβλιογραφία. Ωστόσο, η επιτυχής αξιοποίηση αυτών των πληροφοριών εξαρτάται από τη μέθοδο προσαρμογής του προ-εκπαιδευμένου γλωσσικού μοντέλου και την αρχιτεκτονική του μοντέλου που χρησιμοποιείται. Στην περίπτωση του MELD, η μάθηση βάσει prompt φαίνεται να είναι πιο επιτυχής στην αξιοποίηση των πρόσθετων πληροφοριών σε σύγκριση με το fine-tuning, γεγονός που υποδηλώνει ότι τα prompts έχουν την ικανότητα να προσαρμόζουν το μοντέλο στη νέα μορφή εισόδου, ενώ μπορεί να είναι ακόμη και πιο αποτελεσματικά από το fine-tuning στην αξιοποίηση των πρόσθετων πληροφοριών. Για το IEMOCAP, το fine-tuning επιφέρει καλύτερα αποτελέσματα, ενώ η μάθηση βάσει prompt φαίνεται να αποτυγχάνει στην αξιοποίηση των παρεχόμενων πληροφοριών που αφορούν τον ομιλητή. Επομένως, μπορούμε να συμπεράνουμε ότι, κατά την ενσωμάτωση των πληροφοριών που αφορούν συγκεκριμένους ομιλητές και παρέχονται μέσω της αλλαγής του μορφότυπου εισόδου, ούτε η μάθηση βάσει prompt ούτε το fine-tuning είναι ανώτερα και το ποια μέθοδος προσαρμογής είναι βέλτιστη εξαρτάται σε μεγάλο βαθμό από το σύνολο δεδομένων.

Εκμάθηση πολλαπλών εργασιών με το βοηθητικό πρόβλημα της αναγνώρισης ομιλητή

Οι παρακάτω πίνακες παρουσιάζουν τα αποτελέσματα για το MELD, για το προτεινόμενο μοντέλο που βασίζεται σε prompt και χρησιμοποιεί την αναγνώριση ομιλητή ως βοηθητικό πρόβλημα. Παρουσιάζουμε επίσης τα αποτελέσματα για το αντίστοιχο fine-tuned μοντέλο, καθώς και το απλό μοντέλο μας με βάση prompt που αποτελείται από 30 prompt tokens και δεν διαθέτει πληροφορίες για τον ομιλητή και το αντίστοιχό του fine-tuned μοντέλο, για λόγους σύγκρισης. Το σταθμισμένο F1-score παρουσιάζεται στον Πίνακα 3. Όλα τα αποτελέσματά μας αποτελούν τον μέσο όρο τριών εκτελέσεων.

Table 3. *Σταθμισμένο F1-score (%) για το σύνολο δεδομένων ελέγχου MELD*

Model	F1 score (Test)
Fine-tuned baseline	57.06
Με χρήση prompt, 30 σύμβολα prompt	56.70
Εκμάθηση πολλαπλών εργασιών για συναίσθημα+ομιλητή, fine-tuned	56.79
Εκμάθηση πολλαπλών εργασιών για συναίσθημα+ομιλητή, με χρήση prompt	54.65

Από τον Πίνακα 3 μπορούμε να δούμε ότι η χρήση εκμάθησης πολλαπλών εργασιών για την αναγνώριση συναισθήματος και την αναγνώριση ομιλητή δεν βελτιώνει την απόδοση για το fine-tuned μοντέλο και οδηγεί σε σημαντική μείωση της απόδοσης για το μοντέλο που χρησιμοποιεί prompt. Αυτό θα μπορούσε να υποδηλώνει ότι η αναγνώριση ομιλητή, όπως ορίζεται στη μέθοδό μας, δεν είναι κατάλληλη ως βοηθητικό πρόβλημα για τη βελτίωση της απόδοσης για το ERC. Επιπλέον, συγκρίνοντας τα αποτελέσματα του μοντέλου που χρησιμοποιεί prompts και του fine-tuned μοντέλου, τα οποία εκπαιδεύτηκαν με τη χρήση εκμάθησης πολλαπλών εργασιών, το χαμηλότερο F1-score του μοντέλου που βασίζεται σε prompts (κατά 2,14%) υποδηλώνει ότι η ικανότητα των prompts να προσαρμόσουν το BERT σε ένα πιο σύνθετο περιβάλλον, όπως το περιβάλλον εκμάθησης πολλαπλών εργασιών, δεν είναι αρκετή και ενδεχομένως απαιτούνται εκπαιδεύσιμες παράμετροι και στο εσωτερικό του μοντέλου, ώστε να είναι δυνατή μία μεγαλύτερη τροποποίηση του μοντέλου.

Συνολική συζήτηση

- **Η επίδραση του μεγέθους του prompt:**

  Από τα αποτελέσματα που παρουσιάστηκαν μπορούμε να συμπεράνουμε ότι το μέγεθος του prompt μπορεί να επηρεάσει την απόδοση. Τόσο για το MELD όσο και για το IEMOCAP, παρατηρούμε ότι τα μοντέλα που βασίζονται σε prompt τείνουν να αποδίδουν χαμηλότερα F1-score για πολύ μικρό αριθμό συμβόλων prompt (6-7 σύμβολα). Τα αποτελέσματα βελτιώνονται με την αύξηση των χρησιμοποιούμενων συμβόλων prompt, με 30 σύμβολα για το IEMOCAP και 40 σύμβολα για το MELD να οδηγούν στη βέλτιστη απόδοση, και στη συνέχεια μειώνονται σταδιακά και πάλι όταν αυξάνεται περαιτέρω ο αριθμός συμβόλων prompt (σε 40 και στη συνέχεια σε 60 για το IEMOCAP, σε 60 για το MELD). Αυτή η τάση μπορεί να παρατηρηθεί για τα μοντέλα που βασίζονται σε prompt χωρίς την προσθήκη πληροφορίας σχετικής με τον ομιλητή, καθώς και για τα μοντέλα που βασίζονται σε prompt και χρησιμοποιούν προσθήκη συμβόλων ομιλητή (όπου τα μοντέλα με 6-7 σύμβολα prompt οδηγούν σε χειρότερες επιδόσεις σε σύγκριση με εκείνα με 30 σύμβολα prompt).

  Αυτές οι παρατηρήσεις υποδηλώνουν ότι υπάρχει ένας αριθμός συμβόλων prompt που είναι βέλτιστος για ένα συγκεκριμένο προ-εκπαιδευμένο γλωσσικό μοντέλο σε ένα συγκεκριμένο πρόβλημα. Πιστεύουμε ότι η χαμηλότερη απόδοση όταν χρησιμοποιούνται πολύ λίγα σύμβολα prompt μπορεί να αποδοθεί στο γεγονός ότι δεν υπάρχουν αρκετές εκπαιδεύσιμες παράμετροι για να κωδικοποιήσουν όλη την πληροφορία που απαιτείται για την επαρκή προσαρμογή του BERT στο έργο του ERC, ενώ τα πάρα πολλά σύμβολα prompt οδηγούν στο αντίθετο πρόβλημα, με πάρα πολλές παραμέτρους που πρέπει να εκπαιδευτούν από τα διαθέσιμα δεδομένα, οδηγώντας ενδεχομένως σε μεγαλύτερο πρόβλημα υπερπροσαρμογής (overfitting).

- **Τι μέρος της παρατηρούμενης επίδοσης μπορεί να αποδοθεί στην χρήση prompts;**

  Στην αρχιτεκτονική που περιγράψαμε χρησιμοποιήσαμε μια εκπαιδεύσιμη κεφαλή ταξινόμησης για να αντιστοιχίσουμε την αναπαράσταση του συμβόλου $[MASK]$ του BERT σε μία από τις κλάσεις συναισθημάτων. Καθώς η κεφαλή ταξινόμησης είναι εκπαιδεύσιμη, είναι λογικό να υποθέσουμε ότι συμβάλλει και αυτή στην προσαρμογή του προ-εκπαιδευμένου γλωσσικού μοντέλου στο πρόβλημα του ERC. Προκειμένου να διαχωρίσουμε τη συμβολή της κεφαλής ταξινόμησης από τη συμβολή του prompt, μπορούμε να συγκρίνουμε την απόδοση των μοντέλων μας που βασίζονται σε prompt με εκείνη των μοντέλων τα οποία περιλαμβάνουν μόνο έναν παγωμένο κωδικοποιητή BERT και μια εκπαιδεύσιμη κεφαλή ταξινόμησης και δεν χρησιμοποιούν prompts ούτε fine-tuning για τις υπόλοιπες παραμέτρους τους.

  Από τους Πίνακες 1 και 2 μπορούμε να παρατηρήσουμε ότι, τόσο για το MELD όσο και για το IEMOCAP, τα μοντέλα που βασίζονται σε prompts τείνουν να επιτυγχάνουν υψηλότερο F1-score κατά 2%-3%, σε σύγκριση με τα μοντέλα με μόνο μια εκπαιδεύσιμη κεφαλή ταξινόμησης, γεγονός που μας οδηγεί στο συμπέρασμα ότι τα prompts πράγματι συμβάλλουν στην προσαρμογή ενός προ-εκπαιδευμένου γλωσσικού μοντέλου σε επιμέρους προβλήματα. Επιπλέον, αξίζει να σημειωθεί ότι, όταν στην κειμενική είσοδο χρησιμοποιείται η προσθήκη συμβόλων ομιλητή, τα μοντέλα που χρησιμοποιούν μόνο μία εκπαιδεύσιμη κεφαλή ταξινόμησης δεν μπορούν να αξιοποιήσουν την πρόσθετη πληροφορία και οι επιδόσεις παραμένουν οι ίδιες ή πέφτουν σε σύγκριση με τη μη χρήση προσθήκης συμβόλων ομιλητή. Αυτό υποδηλώνει ότι, κατά την προσαρμογή του προ-εκπαιδευμένου γλωσσικού μοντέλου στο νέο μορφότυπο εισόδου που χρησιμοποιείται για την προσθήκη των συμβόλων ομιλητή και διαφέρει από εκείνο της φάσης προ-εκπαίδευσης του BERT, η συμβολή των prompts διαδραματίζει τον κύριο ρόλο. Μια εξήγηση γι' αυτό θα μπορούσε να είναι ότι τα prompts επηρεάζουν τις αναπαραστάσεις που δημιουργεί το μοντέλο, μέσω των συμφραζομένων, ήδη από το χαμηλότερο επίπεδο του κωδικοποιητή BERT, ενώ η κεφαλή ταξινόμησης μπορεί να αλλάξει μόνο την αντιστοίχιση της αναπαράστασης του συμβόλου [$MASK$] στις κλάσεις συναισθημάτων και δεν μπορεί να επηρεάσει τις κρυφές αναπαραστάσεις του μοντέλου και τις αλληλεπιδράσεις τους στα επίπεδα του κωδικοποιητή BERT.

0.5.2 Prompts ειδικά ανά πληροφορία: Μία εναλλακτική προσέγγιση για την ενσωμάτωση επιπρόσθετης πληροφορίας

Εισαγωγή

Η μοντελοποίηση του θέματος συζήτησης καθώς και η ταυτότητα του ομιλητή έχουν χρησιμοποιηθεί συχνά στην βιβλιογραφία για να βελτιώσουν την επίδοση στο πρόβλημα της Αναγνώρισης Συναισθήματος σε Συζητήσεις (Emotion Recognition in Conversation - ERC). Ο λόγος για αυτό είναι ότι τα διαφορετικά θέματα συζήτησης χαρακτηρίζονται συχνά από διαφορετικά γλωσσικά μοτίβα [44] και μπορούν έτσι να επηρεάσουν τόσο το νόημα της εκφώνησης όσο και το συναίσθημα που εκφράζουν συγκεκριμένες εκφράσεις, ενώ διαφορετικοί ομιλητές μπορεί επίσης να χρησιμοποιούν διαφορετικές λέξεις ή εκφράσεις για να εκφράσουν το ίδιο συναίσθημα. Η μοντελοποίηση του θέματος συζήτησης και η γνώση της ταυτότητας του ομιλητή μπορούν έτσι να βοηθήσουν ένα μοντέλο να ερμηνεύσει αποτελεσματικότερα κάθε εκφώνηση, προκειμένου να αποκωδικοποιήσει το συναίσθημά της. Η ενσωμάτωση της πληροφορίας για τον εκάστοτε ομιλητή επιτυγχάνεται συχνά μέσω της προσθήκης των ονομάτων ομιλητών στις αντίστοιχες εκφωνήσεις στην είσοδο κειμένου [13] [14], ενώ η χρήση ενός βοηθητικού προβλήματος αναγνώρισης ομιλητή έχει επίσης χρησιμοποιηθεί σε προηγούμενα έργα [11] [15]. Στην περίπτωση της μοντελοποίησης του θέματος συζήτησης, ένα γλωσσικό μοντέλο με ένα πρόσθετο στρώμα εξειδικευμένο για την ανίχνευση θεμάτων έχει προταθεί από τους Zhu κ.ά. στο [16].

Παρακινούμενοι από τη θετική επίδραση που φαίνεται να έχουν για το ERC τόσο η πληροφορία για τον ομιλητή όσο και η πληροφορία για το θέμα συζήτησης, προτείνουμε ένα μοντέλο που βασίζεται σε ένα προ-εκπαιδευμένο γλωσσικό μοντέλο και προσαρμόζεται στο πρόβλημα του ERC μέσω μάθησης βάσει prompt, στο οποίο η πληροφορία για τον ομιλητή ή για το θέμα μπορεί να παρέχεται απευθείας μέσω των prompts (τα οποία ονομάζουμε ειδικα ανά πληροφορία prompts (information-specific prompts)), χωρίς να απαιτείται αλλαγή στο μορφότυπο εισόδου ή χρήση πρόσθετων στρωμάτων στο μοντέλο. Η μέθοδός μας είναι ανεξάρτητη από τον τύπο πληροφορίας, πράγμα που σημαίνει ότι μπορεί να εφαρμοστεί για διαφορετικά είδη πληροφορίας, συμπεριλαμβανομένων, μεταξύ άλλων, της ταυτότητας του ομιλητή και του θέματος.

Μέθοδος

Στην ενότητα 0.5.1, χρησιμοποιούμε το προ-εκπαιδευμένο γλωσσικό μοντέλο BERT ως το κύριο μέρος του μοντέλου μας. Το μοντέλο που δημιουργούμε αποτελείται από τρεις κύριες μονάδες, το στρώμα ενσωμάτωσης (embedding), την μονάδα κωδικοποιητή και την μονάδα εξόδου. Το στρώμα ενσωμάτωσης (embedding) αποτελείται από τρία μέρη: Το πρώτο μέρος είναι ένα prompt ειδικό για το πρόβλημα (task-specific prompt): Παράγει ενσωματώματα prompt με τον ίδιο τρόπο όπως στην ενότητα 0.5.1, τα οποία εκπαιδεύουμε σε ολόκληρο το σύνολο δεδομένων και έχει ως σκοπό την προσαρμογή του γλωσσικού μοντέλου στο πρόβλημα του ERC. Το δεύτερο μέρος είναι ειδικό για το θέμα ή για τον ομιλητή, ανάλογα με το αν εργαζόμαστε με ταυτότητες ομιλητών ή θέματα συζήτησης: Υποθέτοντας ότι εργαζόμαστε με s ομιλητές ή s θέματα συζήτησης στα δεδομένα εκπαίδευσης, εκπαιδεύουμε s διαφορετικά prompts, καθένα από τα οποία αντιστοιχεί σε έναν ομιλητή ή σε ένα θέμα και εκπαιδεύεται μόνο με τα δεδομένα αυτού του ομιλητή ή θέματος. Αυτό το τμήμα της μονάδας ενσωμάτωσης είναι λοιπόν υπεύθυνο για την επιλογή του ενός από τα s prompt που αντιστοιχεί κάθε φορά στον συγκεκριμένο ομιλητή ή στο συγκεκριμένο θέμα, για να χρησιμοποιηθεί αυτό μαζί με το ειδικό για το πρόβλημα (task-specific) prompt. Σκοπός αυτού του δεύτερου μέρους της μονάδας ενσωμάτωσης είναι τελικά να επηρεάσει τον τρόπο με τον οποίο το μοντέλο αντιμετωπίζει την είσοδο κειμένου, ανάλογα με το θέμα της συζήτησης ή τον ομιλητή της εκφώνησης. Πιστεύουμε πως κάτι τέτοιο είναι επωφελές, καθώς τόσο η ταυτότητα του ομιλητή όσο και το θέμα συζήτησης μπορούν να επηρεάσουν τον τρόπο με τον οποίο εκφράζονται τα συναισθήματα μέσω της γλώσσας, όπως εξηγήθηκε προηγουμένως. Το τρίτο μέρος της μονάδας ενσωμάτωσης είναι υπεύθυνο για τον υπολογισμό των ενσωματωμάτων που αντιστοιχούν στην είσοδο κειμένου του μοντέλου. Η μονάδα κωδικοποιητή και η μονάδα εξόδου του μοντέλου μας διατηρούνται τα ίδια όπως και στην υποενότητα 0.5.1: Ο κωδικοποιητής αποτελείται από 12 προ-εκπαιδευμένα στρώματα transformer κωδικοποιητή (πρόκειται για τον κωδικοποιητή BERT), ενώ ως μονάδα εξόδου χρησιμοποιούμε δύο γραμμικά επίπεδα (linear layers) με ένα επίπεδο dropout μεταξύ τους, προκειμένου να αντιστοιχίσουμε την αναπαράσταση εξόδου του κωδικοποιητή BERT σε μια ετικέτα συναισθήματος. Σημειώνουμε ότι εργαζόμαστε με prompts που αφορούν συγκεκριμένο θέμα ή ομιλητή, καθώς αυτό το είδος πληροφορίας είναι ήδη διαθέσιμο στα σύνολα δεδομένων μας ή μπορεί εύκολα να εξαχθεί για το πρόβλημα του ERC, αλλά η ίδια λογική θα μπορούσε να εφαρμοστεί και σε άλλους τύπους πληροφορίας.

Πειραματιζόμαστε στα δύο σύνολα δεδομένων ERC που περιγράφονται στην ενότητα 5.5, MELD και IEMOCAP και παρουσιάζουμε τα αποτελέσματά μας για τη χρήση τόσο ειδικών για το πρόβλημα prompts (task-specific prompts) όσο και ειδικών ανά θέμα/ομιλητή prompts (speaker-/topic-specific prompts), ενώ πειραματιζόμαστε επίσης με τη χρήση μόνο ειδικών ανά ομιλητή/ θέμα prompts και καθόλου ειδικών για το πρόβλημα prompts. Συγκρίνουμε τα αποτελέσματά μας με εκείνα ενός απλού μοντέλου βασισμένου σε prompts, χωρίς prompts ειδικά ανά θέμα/ομιλητή, προκειμένου να προσδιορίσουμε κατά πόσον η προσθήκη prompts συγκεκριμένων ανά ομιλητή ή θέμα μπορεί να ενσωματώσει με επιτυχία την χρήσιμη πρόσθετη πληροφορία (σχετική με τον ομιλητή ή το θέμα συζήτησης), ώστε να βελτιωθεί η επίδοση του μοντέλου.

Στις επόμενες παραγράφους παρέχουμε μια λεπτομερή περιγραφή της αρχιτεκτονικής του προτεινόμενου μοντέλου μας καθώς και της μεθόδου που χρησιμοποιούμε για την εξαγωγή του θέματος κάθε εκφώνησης. Μια σχηματική επισκόπηση του προτεινόμενου μοντέλου μας απεικονίζεται στο Σχήμα 5.

Μορφότυπο κειμένου εισόδου

Χρησιμοποιούμε το ίδιο μορφότυπο κειμένου εισόδου όπως στην ενότητα 6.1, εκτελώντας Masked LM και χρησιμοποιώντας μια είσοδο δύο προτάσεων, όπου η πρώτη πρόταση αποτελεί μια συνένωση προηγούμενων εκφωνήσεων, που θα χρησιμοποιηθούν από το μοντέλο ως συμφραζόμενα, και η δεύτερη πρόταση είναι η τρέχουσα εκφώνηση που πρέπει να ταξινομηθεί, ακολουθούμενη από ένα σύμβολο $[MASK]$. Το τελικό κείμενο εισόδου διαμορφώνεται ως εξής:

$$[CLS]|u_{i-k}|u_{i-k+1}|...|u_{i-1}|[SEP]|u_i|[MASK]|[SEP]$$

όπου i είναι ο δείκτης της τρέχουσας εκφώνησης προς ταξινόμηση, k είναι ο αριθμός των προηγούμενων εκφωνήσεων που χρησιμοποιούνται ως συμφραζόμενα, u_i είναι η εκφώνηση με δείκτη i και $|$ είναι το σύμβολο της συνένωσης.

Η είσοδος αυτή περνάει στο τμήμα του στρώματος ενσωμάτωσης του μοντέλου μας που είναι υπεύθυνο για τον υπολογισμό των ενσωματωμάτων που αντιστοιχούν στα σύμβολα (tokens) της κειμενικής εισόδου και που ονομάζουμε ενσωματώματα κειμένου (text embeddings). Για το σκοπό αυτό, χρησιμοποιείται αμετάβλητο το προ-εκπαιδευμένο στρώμα ενσωμάτωσης του BERT, το οποίο αντιστοιχίζει τα σύμβολα στην είσοδο στα αντίστοιχα προ-εκπαιδευμένα ενσωματώματα κειμένου και στη συνέχεια προσθέτει σε αυτά τα προ-εκπαιδευμένα ενσωματώματα τμήματος και θέσης (segment και position embeddings), δίνοντας στην έξοδο τα τελικά ενσωματώματα κειμένου (text embeddings).

Ειδικά για το πρόβλημα σύμβολα prompt

Χρησιμοποιούμε t ενωματώματα συμβόλων prompt (prompt token embeddings) τα οποία εκπαιδεύονται με χρήση όλου του συνόλου δεδομένων και έχουν ως στόχο την προσαρμογή του προ-εκπαιδευμένου μοντέλου στο πρόβλημα του ERC. Αυτά είναι τα ίδια με τα ενωματώματα συμβόλων prompt που χρησιμοποιούμε στην υποενότητα 0.5.1, τα οποία βρίσκονται απευθείας στον χώρο ενσωμάτωσης του BERT και οδηγούν στα τελικά ενσωματώματα prompt (prompt embeddings), μετά την άθροιση τους με τα ενσωματώματα τμήματος και θέσης του BERT.

Ειδικά ανά πληροφορία σύμβολα prompt

Εκτός από τα t ειδικά για το πρόβλημα (task-specific) σύμβολα prompt που περιγράφηκαν παραπάνω, χρησιμοποιούμε m ακόμα σύμβολα prompt, τα οποία είναι ειδικά ανά την πρόσθετη πηγή πληροφοριών που προσπαθούμε να ενσωματώσουμε (δηλαδή ειδικά ανά ομιλητή ή θέμα συζήτησης) και τα οποία ονομάζουμε σύμβολα prompt ειδικά ανά πληροφορία (information-specific prompt tokens).

Καθώς αυτά είναι ειδικά ανά θέμα συζήτησης ή ομιλητή, δεν εκπαιδεύονται στο σύνολο των δεδομένων: Υποθέτοντας ότι διαθέτουμε s θέματα συζήτησης ή s ομιλητές (ή ομάδες ομιλητών, με τους ομιλητές μέσα σε κάθε ομάδα μοντελοποιούνται όλοι μαζί), θα έχουμε s ομάδες από m ενσωματώματα συμβόλων prompt (prompt token embeddings) η καθεμία, με αποτέλεσμα έναν τελικό πίνακα από ενσωματώματα συμβόλων prompt διαστάσεων $(s + m) * d$, όπου d είναι η διάσταση ενσωμάτωσης του BERT (768 για το bert-base). Για κάθε τρέχουσα εκφώνηση που πρόκειται να ταξινομηθεί, προσδιορίζουμε τον ομιλητή ή το θέμα της εκφώνησης και επιλέγουμε τα m ενσωματώματα συμβόλων prompt που αντιστοιχούν σε αυτόν τον ομιλητή/θέμα. Αυτά τα m ενσωματώματα συμβόλων prompt χρησιμοποιούνται στη συνέχεια μαζί με τα t ειδικά για το πρόβλημα ενσωματώματα συμβόλων prompt (task-specific prompt token embeddings) για την ταξινόμηση της τρέχουσας εκφώνησης, αφού πρώτα προστεθούν τα ενσωματώματα θέσης και τμήματος του BERT.

Τελική έξοδος επιπέδου ενσωμάτωσης

Για την ταξινόμηση μίας εκφώνησης, αφού λάβουμε τα ενσωματώματα κειμένου, τα ειδικά για το πρόβλημα ενσωματώματα prompt και τα ειδικά ανά ομιλητή/θέμα ενσωματώματα prompt, συνενώνουμε τα τρία, με αποτέλεσμα να προκύπτουν $t + m + n$ ενσωματώματα (όπου n ο αριθμός των ενσωματωμάτων που αντιστοιχούν στο κείμενο εισόδου, επιλεγμένος έτσι ώστε ο συνολικός αριθμός των ενσωματωμάτων να μην υπερβαίνει το μέγιστο μέγεθος εισόδου του BERT: $t + m + n \leq 512$), τα οποία στη συνέχεια διαβιβάζονται στην προ-εκπαιδευμένη μονάδα κωδικοποιητή BERT. Μια σχηματική αναπαράσταση της τελικής εξόδου του προτεινόμενου μοντέλου στο επίπεδο ενσωμάτωσης απεικονίζεται στο Σχήμα 4.

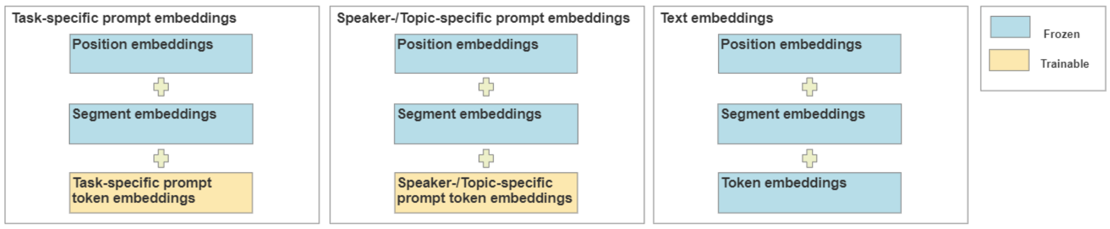


Figure 4. *Επισκόπηση του στρώματος ενσωμάτωσης για την περίπτωση μάθησης βάσει prompt. Η τελική έξοδος στο επίπεδο ενσωμάτωσης προκύπτει από τη συνένωση των ειδικών για το πρόβλημα ενσωματωμάτων prompt (task-specific prompt embeddings), τον ειδικών ανά πληροφορία ενσωματωμάτων prompt (information-specific prompt-embeddings) (ειδικών ανά ομιλητή ή θέμα) και των ενσωματωμάτων κειμένου (text embeddings). Οι μονάδες με μπλε χρώμα διατηρούνται παγωμένες κατά τη διάρκεια της εκπαίδευσης, ενώ οι μονάδες με κίτρινο χρώμα είναι εκπαιδεύσιμες.*

Κωδικοποιητής BERT και κεφαλή ταξινόμησης

Χρησιμοποιούμε τον προ-εκπαιδευμένο κωδικοποιητή BERT και προσθέτουμε πάνω από αυτόν μία κεφαλή ταξινόμησης, προκειμένου να αντιστοιχίσουμε την τελική κρυφή αναπαράσταση του συμβόλου $[MASK]$ της εισόδου του μοντέλου σε μία από τις κλάσεις του συνόλου δεδομένων μας. Χρησιμοποιούμε την ίδια κεφαλή ταξινόμησης με την υπο ενότητα 0.5.1, αποτελούμενη από δύο γραμμικά επίπεδα (linear layers) με ένα επίπεδο dropout ανάμεσά τους. (Βλ. Σχήμα 3 για μία λεπτομερή αναπαράσταση).

Παγωμένα και εκπαιδεύσιμα τμήματα

Παγώνουμε όλες τις παραμέτρους του μοντέλου, εκτός από τις παραμέτρους της κεφαλής ταξινόμησης και του prompt (τόσο τα ειδικά για το πρόβλημα (task-specific) όσο και τα ειδικά ανά πληροφορία (information-specific) ενσωματώματα συμβόλων prompt παραμένουν εκπαιδεύσιμα), τις οποίες εκπαιδεύουμε μέσω back-propagation. Σημειώνουμε και πάλι ότι διατηρούμε παγωμένες τις παραμέτρους των ενσωατωμάτων τμήματος και θέσης τόσο για το κείμενο εισόδου όσο και για τα σύμβολα prompt, και επίσης δεν εκπαιδεύουμε τα ενσωματώματα συμβόλων (token embeddings) του BERT, τα οποία χρησιμοποιούνται για την αντιστοίχιση των συμβόλων του κειμένου εισόδου στο χώρο ενσωμάτωσης.

Μέθοδος εξαγωγής θέματος

Ενώ η ταυτότητα του ομιλητή παρέχεται τόσο στο IEMOCAP όσο και στο MELD, αυτό δεν ισχύει για το θέμα κάθε εκφώνησης. Προκειμένου να χρησιμοποιήσουμε prompts ειδικά ανά θέμα, εκτελούμε λοιπόν πρώτα μοντελοποίηση θέματος, ώστε να εξάγουμε το κύριο θέμα κάθε εκφώνησης. Χρησιμοποιούμε την μέθοδο Latent Dirichlet Allocation (LDA) [45] για να εκτελέσουμε τη μοντελοποίηση θέματος [46]. Σημειώνουμε ότι για κάθε εκφώνηση, χρησιμοποιούμε αυτήν καθώς και τα συμφραζόμενά της (τις k προηγούμενες εκφωνήσεις του διαλόγου που δίνονται ως είσοδος στο μοντέλο μαζί με την τρέχουσα εκφώνηση) ως έναν ενιαίο σύνολο (ως έγγραφο για μοντελοποίηση θέματος), προκειμένου να είμαστε σε θέση να προσδιορίσουμε το τρέχον θέμα της συνομιλίας πιο αποτελεσματικά. Αυτό είναι απαραίτητο, καθώς οι εκφωνήσεις σε έναν διάλογο είναι συχνά μικρές και γενικές (για παράδειγμα: "Ναι." ή "Δεν ξέρω!"), οπότε δεν είναι δυνατόν να προσδιοριστεί το θέμα χρησιμοποιώντας μόνο την τρέχουσα εκφώνηση.

Έτσι, θεωρώντας κάθε εκφώνηση και τα συμφραζόμενά της ως ένα έγγραφο, υπολογίζουμε τον πίνακα όρων του εγγράφου (Document Term Matrix - DTM) για το σύνολο εκπαίδευσης του συνόλου δεδομένων, ο οποίος είναι ένας πίνακας που περιγράφει τη συχνότητα κάθε λέξης σε κάθε ένα από τα έγγραφα και χρησιμοποιούμε την μέθοδο LDA για να εκτελέσουμε την μοντελοποίηση θέματος. Η LDA υποθέτει μια συλλογή από k θέματα, όπου το k πρέπει να είναι προκαθορισμένο. Θεωρεί κάθε έγγραφο ως μια συλλογή θεμάτων και κάθε θέμα ως μια συλλογή λέξεων και, μέσω μιας στατιστικής διαδικασίας, υπολογίζει το μείγμα θεμάτων από το οποίο αποτελείται κάθε έγγραφο. Αυτό μπορεί να θεωρηθεί ως μια ήπια (soft) κατηγοριοποίηση κάθε εγγράφου στα διάφορα θέματα. Για να λάβουμε το κύριο θέμα κάθε εγγράφου, επιλέγουμε το θέμα με το μεγαλύτερο βάρος. Σημειώνουμε ότι εκτελούμε την ανάλυση LDA χρησιμοποιώντας μόνο τα δεδομένα της εκπαίδευσής μας. Κατά τη διάρκεια του testing, απλώς ταξινομούμε κάθε έγγραφο στα υπάρχοντα θέματα, χρησιμοποιώντας το μοντέλο LDA των δεδομένων εκπαίδευσης.

Συνάθροιση prompt

Όταν πραγματοποιούμε μοντελοποίηση θέματος για να λάβουμε το κύριο θέμα κάθε εκφώνησης, μπορούμε να υποθέσουμε μια αυστηρή (hard) ταξινόμηση, αναθέτοντας ένα θέμα σε κάθε εκφώνηση (το θέμα με τη μέγιστη πιθανότητα) και επιλέγοντας τα m αντίστοιχα σύμβολα prompt, ή μπορούμε να υποθέσουμε μια ήπια (soft) ταξινόμηση μεταξύ των θεμάτων, χρησιμοποιώντας τις πιθανότητες που υπολογίζονται από τη μέθοδο εξαγωγής θέματος. Στην τελευταία περίπτωση, υποθέτουμε ότι κάθε εκφώνηση ανήκει σε κάθε θέμα με πιθανότητα p. Αντί τότε να χρησιμοποιούμε το prompt που αντιστοιχεί στο κύριο θέμα της εκφώνησης κατά τη διάρκεια του ελέγχου (testing), μπορούμε να υπολογίσουμε μία συνάθροιση (ensemble) από prompts, λαμβάνοντας τον σταθμισμένο μέσο όρο των παραμέτρων όλων των s prompts που αντιστοιχούν στα s θέματα (κάθενα από τα οποία αποτελείται από m prompt tokens), χρησιμοποιώντας τις πιθανότητες των θεμάτων ως βάρη. Ονομάζουμε αυτή την τεχνική "συνάθροιση prompt" (prompt-ensembling). Πειραματιζόμαστε με την τεχνική της συνάθροισης prompt και παρέχουμε τα αποτελέσματα στην ενότητα 0.5.2. Σημειώνουμε ότι η συνάθροιση πραγματοποιείται μόνο κατά τη διάρκεια του ελέγχου (testing), ενώ εξακολουθούμε να υποθέτουμε μια αυστηρή ταξινόμηση κατά τη διάρκεια της εκπαίδευσης, προκειμένου να εκπαιδεύσουμε τα ειδικά ανά θέμα prompts.

Σύνολο δεδομένων

Η μέθοδός μας βασίζεται στην εκπαίδευση ενός ειδικού ανά ομιλητή ή θέμα prompt χρησιμοποιώντας το σύνολο εκπαίδευσης και, στη συνέχεια, στη χρήση αυτού του προ-εκπαιδευμένου prompt κατά τη διάρκεια του testing. Αυτό σημαίνει ότι υποθέτει ότι ο ομιλητής ή το θέμα που εμφανίζεται κατά τη διάρκεια του ελέγχου (testing) υπάρχει και στο σύνολο εκπαίδευσης. Όταν εργαζόμαστε με το θέμα συζήτησης, αυτό δεν αποτελεί πρόβλημα, καθώς κάθε εκφώνηση μπορεί να αντιστοιχηθεί στο θέμα του συνόλου εκπαίδευσης στο οποίο είναι πιο κοντά. Ωστόσο, όταν βρεθεί ένας άγνωστος ομιλητής στο σύνολο test, ο οποίος δεν υπάρχει στο σύνολο εκπαίδευσης, δεν διαθέτουμε εκπαιδευμένο prompt για αυτόν τον ομιλητή. Για το λόγο αυτό, στην περίπτωση των prompt ειδικών ανά ομιλητή, δοκιμάζουμε τη μέθοδό μας στο MELD και όχι στο IEMOCAP, καθώς στο IEMOCAP υπάρχουν διαφορετικοί ομιλητές σε κάθε ένα από τα σύνολα εκπαίδευσης (train), ανάπτυξης (validation) και ελέγχου (testing). Από την άλλη πλευρά, στο MELD, οι έξι κύριοι ομιλητές είναι παρόντες και στα τρία μέρη δεδομένων (train, validation και test). Επομένως, χρησιμοποιούμε m σύμβολα prompt για κάθε έναν από αυτούς και ομαδοποιούμε όλους τους υπόλοιπους ομιλητές του MELD σε έναν, χρησιμοποιώντας m κοινά, ειδικά ανά ομιλητή σύμβολα prompt για αυτούς.

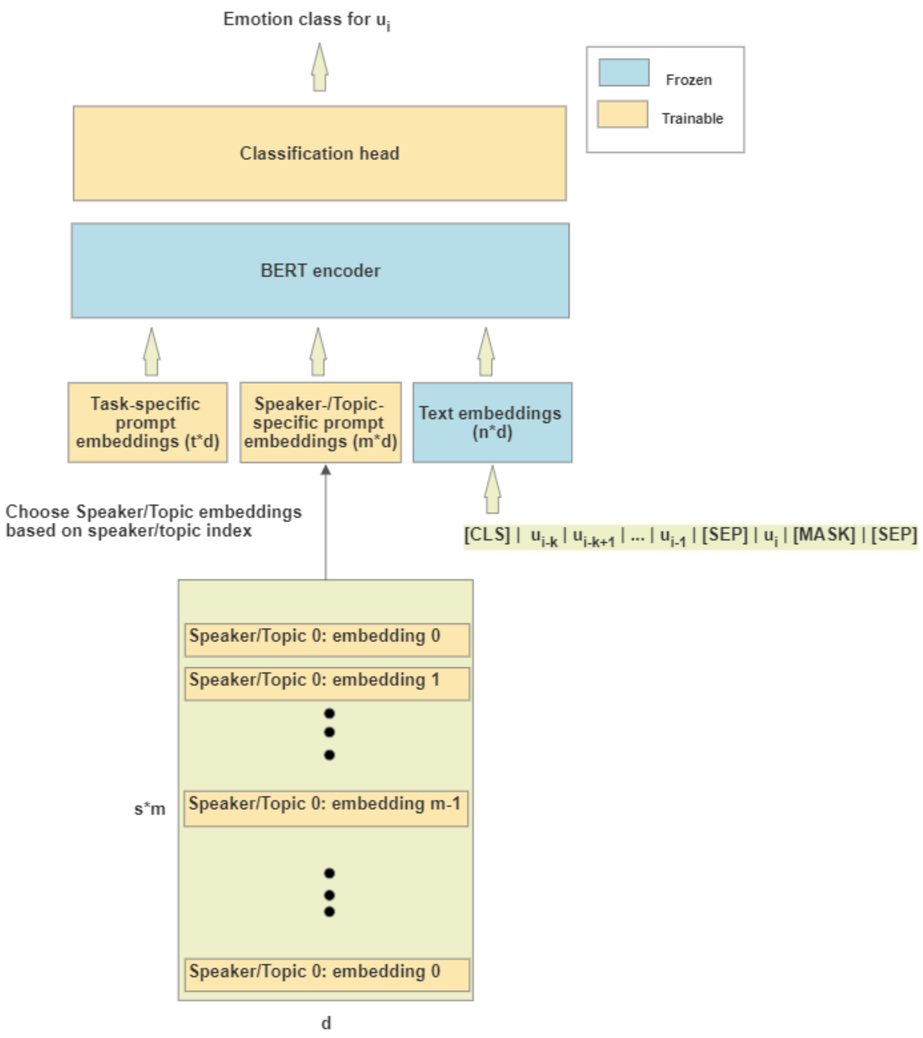

Emotion class for u_i
Frozen
Trainable
Classification head
BERT encoder
Task-specific prompt embeddings (t*d)
Speaker-/Topic-specific prompt embeddings (m*d)
Text embeddings (n*d)
Choose Speaker/Topic embeddings based on speaker/topic index
[CLS] | u_{i-k} | u_{i-k+1} | ... | u_{i-1} | [SEP] | u_i | [MASK] | [SEP]
Speaker/Topic 0: embedding 0
Speaker/Topic 0: embedding 1
Speaker/Topic 0: embedding m-1
Speaker/Topic 0: embedding 0
s*m
d

Figure 5. *Γενικό διάγραμμα του μοντέλου μας με βάση τα prompts, όπου t είναι ο αριθμός των ειδικών για το πρόβλημα συμβόλων prompt, m είναι ο αριθμός των ειδικών ανά πληροφορία (information-specific) συμβόλων prompt, s είναι ο αριθμός των ομιλητών/θεμάτων, d είναι η διάσταση ενσωμάτωσης του BERT (768), n είναι ο αριθμός των συμβόλων prompt που αντιστοιχούν στο κείμενο εισόδου μας, k είναι ο αριθμός των προηγούμενων εκφωνήσεων που χρησιμοποιούνται ως συμφραζόμενα και i είναι ο δείκτης της τρέχουσας εκφώνησης, η οποία πρέπει να ταξινομηθεί. Για s ομιλητές/θέματα εκπαιδεύουμε s prompts που αποτελούνται από m σύμβολα prompt το καθένα, οπότε έχουμε συνολικά $s * m$ εκπαιδεύσιμα ενσωματώματα prompt, από τα οποία επιλέγουμε m κάθε φορά, ανάλογα με τον ομιλητή ή το θέμα. Οι μονάδες με μπλε χρώμα διατηρούνται παγωμένες κατά τη διάρκεια της εκπαίδευσης, ενώ οι μονάδες με κίτρινο χρώμα είναι εκπαιδεύσιμες. Σημειώνεται ότι τα ειδικά για το πρόβλημα και τα ειδικά ανά πληροφορία ενσωματώματα prompt έχουν εκπαιδεύσιμα τμήματα, αλλά δεν είναι εκπαιδεύσιμα στο σύνολό τους: Βλ. Εικόνα 4 για μια λεπτομερή επισκόπηση.*

Αποτελέσματα και συζήτηση

Οι Πίνακες 4 και 5 παρουσιάζουν το σταθμισμένο F1-score για τα MELD και IEMOCAP, για το προτεινόμενο από εμάς μοντέλο καθώς και για τα baseline μοντέλα μας. Όλα τα αποτελέσματα έχουν υπολογιστεί ως ο μέσος όρος τριών τρεξιμάτων.

Table 4. *Σταθμισμένο F1-score (%) για το σύνολο δεδομένων MELD*

Model	F1 score (Test)
Με χρήση prompt, 30 σύμβολα prompt	56.70
Με χρήση prompt, 40 σύμβολα prompt	56.94
Μόνο speaker-specific prompts, 40 speaker-specific σύμβολα prompt	55.68
Task- + Speaker-specific prompts, 30 task-/10 speaker-specific σύμβολα prompt	57.08
Μόνο topic-specific prompts, αυστηρή ταξινόμηση 40 topic-specific σύμβολα prompt	54.96
Task- + Topic-specific prompts, αυστηρή ταξινόμηση 30 task-/10 topic-specific σύμβολα prompt	56.63
Task- + Topic-specific prompts, συνάθροιση prompt 30 task-/10 topic-specific σύμβολα prompt	56.82

Table 5. *Σταθμισμένο F1-score (%) για το σύνολο δεδομένων IEMOCAP*

Model	F1 score (Test)
Με χρήση prompt, 30 σύμβολα prompt	59.94
Με χρήση prompt, 40 σύμβολα prompt	59.21
Μόνο topic-specific prompts, αυστηρή ταξινόμηση 40 topic-specific σύμβολα prompt	58.69
Task- + Topic-specific prompts, αυστηρή ταξινόμηση 30 task-/10 topic-specific σύμβολα prompt	61.13
Task- + Topic-specific prompts, συνάθροιση prompt 30 task-/10 topic-specific σύμβολα prompt	61.12

Ειδικά ανά ομιλητή prompts

Όσον αφορά τη χρήση prompts ειδικών ανά ομιλητή, πειραματιστήκαμε μόνο με το MELD, επομένως τα αποτελέσματά μας περιέχονται στον πίνακα 4. Μεταξύ των μοντέλων που χρησιμοποιούν prompts ειδικά ανά ομιλητή, η βέλτιστη απόδοση (F1-score ίσο με 57,08%) επιτυγχάνεται από το μοντέλο που χρησιμοποιεί 30 task-specific σύμβολα prompt που παραμένουν ίδια για όλο το σύνολο δεδομένων και 10 σύμβολα prompt ειδικά ανά ομιλητή (speaker-specific prompt tokens) που αλλάζουν ανάλογα με τον ομιλητή. Αυτό το μοντέλο επιτυγχάνει επίσης καλύτερο F1-score σε σύγκριση με τα baseline μοντέλα που βασίζονται σε prompts και χρησιμοποιούν τόσο 30 όσο και 40 task-specific σύμβολα prompt και καθόλου information-specific σύμβολα prompt, γεγονός που υποδηλώνει ότι η πληροφορία για τον ομιλητή μπορεί πράγματι να κωδικοποιηθεί αποτελεσματικά μέσω της χρήσης prompts ειδικών ανά ομιλητή.

Ειδικά ανά θέμα prompts

Χρησιμοποιώντας ειδικά ανά θέμα (topic-specific) prompts, πειραματιστήκαμε τόσο στο MELD όσο και στο IEMOCAP. Για το MELD, το μοντέλο που αξιοποιεί το θέμα με τις καλύτερες επιδόσεις είναι το μοντέλο που χρησιμοποιεί 30 ειδικά για το πρόβλημα (task-sepcific) σύμβολα prompt και 10 ειδικά ανά θέμα σύμβολα prompt, με το prompt για το θέμα να υπολογίζεται ως συνάθροιση των διαφόρων ειδικών για το θέμα prompts, σύμφωνα με τα βάρη των θεμάτων που υπολογίζει η μέθοδος LDA. Ωστόσο, αυτό το μοντέλο επιτυγχάνει F1-score σχεδόν ίδιο με το μοντέλο με βάση prompt που χρησιμοποιεί μόνο 40 ειδικά για το πρόβλημα σύμβολα prompt και κανένα ειδικό ανά θέμα σύμβολο prompt. Αυτό σημαίνει ότι, για το MELD, η προσθήκη πληροφορίας για το θέμα μέσω prompts δεν φαίνεται να είναι επωφελής.

Τα αποτελέσματά μας είναι διαφορετικά στην περίπτωση του IEMOCAP: Σε σύγκριση με το μοντέλο που βασίζεται σε prompts και χρησιμοποιεί μόνο 30 ειδικά για το πρόβλημα σύμβολα prompt, τα μοντέλα που χρησιμοποιούν 30 ειδικά για το πρόβλημα και 10 ειδικά ανά θέμα σύμβολα prompt επιτυγχάνουν F1-score μεγαλύτερο κατά 1,19% στην περίπτωση του μοντέλου που χρησιμοποιεί αυστηρή (hard) ταξινόμηση και κατά 1,18% στην περίπτωση του μοντέλου που χρησιμοποιεί συνάθροιση prompt (prompt-ensembling). Σε σύγκριση με το μοντέλο που χρησιμοποιεί 40 ειδικά για το πρόβλημα σύμβολα prompt και καθόλου ειδικά ανά θέμα σύμβολα prompt, η διαφορά αυτή είναι ακόμη μεγαλύτερη. Συνεπώς, οι πληροφορίες που αφορούν το θέμα συζήτησης φαίνεται να είναι σημαντικές για το IEMOCAP και αξιοποιούνται με επιτυχία μέσω ειδικών ανά θέμα prompts.

Ένας λόγος που εξηγεί τη διαφορά στις επιδόσεις των ειδικών ανά θέμα prompts στα δύο σύνολα δεδομένων θα μπορούσε να είναι το γεγονός ότι υπάρχουν μεγαλύτερες διαφορές στην κατανομή των συναισθημάτων μεταξύ των διαφόρων θεμάτων στο IEMOCAP, παρά στο MELD, όπως παρατηρήσαμε αναλύοντας την κατανομή των συναισθημάτων ανά θέμα για κάθε σύνολο δεδομένων. (Ο ενδιαφερόμενος αναγνώστης μπορεί να μελετήσει τους Πίνακες 6.15, 6.16. 6.17 και 6.18). Ένας δεύτερος παράγοντας θα μπορούσε να είναι ότι η μοντελοποίηση θέματος στο IEMOCAP κωδικοποιεί περισσότερη πληροφορία, με τα διαφορετικά θέματα να φέρουν ορισμένα διαφορετικά γλωσσικά μοτίβα που επηρεάζουν το νόημα των εκφωνήσεων και τον τρόπο μετάδοσης του συναισθήματος, ενώ τα θέματα στο MELD είναι πιο παρόμοια ή γενικά και συνεπώς πιο φτωχά σε πληροφορία. Το ισχυρισμό αυτό επιβεβαιώσαμε παρατηρώντας τις πιο κοινές λέξεις για κάθε ένα από τα θέματα που εξήχθησαν από το σύνολο εκπαίδευσης, για το MELD και για το IEMOCAP, απ' όπου διαπιστώσαμε ότι, στην περίπτωση του MELD, πολλές από τις πιο συχνές λέξεις για κάθε θέμα είναι λέξεις που χρησιμοποιούνται στον προφορικό καθημερινό λόγο ("uhh", "ah", "huh" κ.λπ.), ενώ υπάρχουν επίσης πολλές κοινές λέξεις μεταξύ των διαφόρων θεμάτων. Αυτό θα μπορούσε να σημαίνει ότι, λόγω της προέλευσης του MELD, που είναι καθημερινές συνομιλίες από μια τηλεοπτική σειρά, οι διάλογοι δεν είναι τόσο σύνθετοι και δεν έχουν πάντα ένα προφανές θέμα ή σκοπό και επομένως η μοντελοποίηση θέματος δεν παρέχει χρήσιμες πληροφορίες. Από την άλλη πλευρά, το IEMOCAP, έχοντας σχεδιαστεί ρητά για το πρόβλημα αναγνώρισης συναισθήματος, θα μπορούσε να περιλαμβάνει κείμενο πιο κατάλληλο για μοντελοποίηση θέματος, με πλουσιότερες σχετικές πληροφορίες.

Χρήση μόνο ειδικών ανά πληροφορία prompts έναντι της χρήση ειδικών για το πρόβλημα και ειδικών ανά πληροφορία prompts

Μελετώντας τα διαφορετικά μοντέλα που χρησιμοποιούν prompts που αφορούν τον ομιλητή και το θέμα, μπορούμε να δούμε ότι, τόσο για το MELD όσο και για το IEMOCAP, η χρήση 40 συμβόλων prompt που αλλάζουν όλα ανάλογα με τον ομιλητή ή το θέμα και καθόλου συμβόλων prompt ειδικών για το πρόβλημα που παραμένουν ίδια για όλο το σύνολο εκπαίδευσης, αποδίδει χαμηλότερες επιδόσεις από αυτές που επιτυγχάνουν τα μοντέλα που χρησιμοποιούν ειδικά για το πρόβλημα καθώς και ειδικά ανά ομιλητή/θέμα prompt. Αυτό θα μπορούσε να υποδηλώνει ότι η χρήση ολόκληρου του συνόλου δεδομένων, άρα περισσότερων δεδομένων, μας επιτρέπει να καθορίσουμε τις παραμέτρους των prompt πιο αποτελεσματικά, προκειμένου να προσαρμόσουμε το προ-εκπαιδευμένο γλωσσικό μοντέλο μας στο πρόβλημα του ERC, και τα επιπλέον σύμβολα prompt που είναι ειδικά για τον ομιλητή/θέμα μπορούν στη συνέχεια να χρησιμοποιηθούν για την πρόσθετη κωδικοποίηση της πρόσθετης πληροφορίας που σχετίζεται με τον ομιλητή/θέμα. Αντίθετα, όταν χρησιμοποιούνται μόνο prompts ειδικά για τον ομιλητή/θέμα, τα δεδομένα ανά ομιλητή ή θέμα δεν επαρκούν για την αποτελεσματική εκπαίδευση όλων των παραμέτρων του prompt.

Αυστηρή ταξινόμηση έναντι συνάθροισης prompt

Όπως αναλύθηκε νωρίτερα, στην περίπτωση του θέματος, πειραματιζόμαστε με την επιλογή ενός συγκεκριμένου prompt για κάθε εκφώνηση του συνόλου δοκιμών (test), του prompt που αντιστοιχεί στο θέμα στο οποίο η εκφώνηση ανήκει με τη μεγαλύτερη πιθανότητα, καθώς και με τη χρήση της πιθανότητας κάθε θέματος να είναι το θέμα της εκφώνησης, προκειμένου να υπολογιστεί ένας σταθμισμένος μέσος όρος των παραμέτρων των prompts που αντιστοιχούν στα διαφορετικά θέματα (συνάθροιση prompt). Συγκρίνοντας τα αποτελέσματα για τις δύο μεθόδους, παρατηρούμε ότι η συνάθροιση prompt οδηγεί σε ίση απόδοση για το IEMOCAP και σε ανώτερη απόδοση για το MELD, αν και μόνο κατά 0,19%, γεγονός που μας οδηγεί στο συμπέρασμα ότι η συνάθροιση prompt μπορεί να είναι επωφελής, αλλά δεν επηρεάζει σε μεγάλο βαθμό την απόδοση στις παρούσες συνθήκες των πειραμάτων μας.

Chapter 1

Introduction

1.1 Motivation

Emotion is innate to humans and it constitutes a major factor affecting human behaviour. Understanding human emotion is therefore a key part of understanding humans. In a conversational context, accurately determining emotion can be vital to interpreting each speaker's conduct, intent and conveyed meaning. With the rise of online platforms and social media, such as Facebook, Reddit, Twitter etc. an overwhelming amount of conversational data becomes publicly available daily and its automatic analysis for the purpose of mining opinions and studying human behaviour is widely sought after. In addition, the pursuit of the creation of conversational agents and AI assistants that possess human-like artificial intelligence and are able to converse in a human-native way, demands a thorough understanding of the human interlocutor's emotional state. In a much different context, recognizing emotion when human's are conversing can aid the development of psychological analysis tools for assisting doctors and can thus support health-care.

Emotion Recognition in Conversation (ERC) is defined as the task of determining the emotion of each utterance, given a series of utterances that constitute a dialogue between two or more human speakers. Due to its aforementioned potential applications, it has gained significant popularity in the recent years, with an increasing amount of researchers constantly improving previous systems and suggesting new architectures that can aid the task. Suggested models often leverage speaker-identity [11] [13] [14] [15], topic modeling [16], intra-dependency and inter-dependency encoding [17], in order to effectively capture emotion, and they are usually based in one of two main approaches. The first uses graph neural networks, encoding utterances and their relations as nodes and edges of a graph and modeling dependencies through the aggregation of information from neighbouring nodes. The second leverages the sequential nature of conversations, utilizing models that capture sequential relationships explicitly, such as Recurrent Neural Networks (RNNs) and transformer-based pre-trained language models (such as BERT [18], RoBERTa [19], BART [20] etc.), and is perhaps the most common in the latest state of the art works.

Pre-trained language models allow researchers to achieve very good performance when attempting to recognize emotion in conversations, due to the extensive understanding of human language they have obtained during their pre-training phase. To adapt them to the ERC setting, fine-tuning is usually employed, which tunes all of the pre-trained language model's parameters using a dataset designed for the task of ERC. However, considering the fact that such models consist of millions or even billions of parameters, fine-tuning can be very expensive. The training is both time- and space-demanding, while the resulting models also occupy a large storage space [21]. In addition, because the datasets available for the ERC task are relatively small in comparison to the number of parameters pre-trained language models possess, the trained models often suffer from overfitting [22].

To mitigate these problems, prompt-based learning has been proposed as a general method for adapting pre-trained language models to downstream tasks. Instead of tuning the pre-trained model's parameters, prompt-based learning keeps them frozen, and adds a set of new parameters, known as a prompt, in the input level of the model, which it then optimizes, in order to influence the way the model handles input, to better accommodate the downstream task. Because the number of trainable parameters introduced constitutes a very small percentage of the number of pre-trained parameters fine-tuning traditionally optimizes, prompt-based learning is significantly more lightweight and less prone to cause overfitting. It has thus captured the interest of multiple researchers, that have suggested different prompt architectures and training methods. However, despite the increasing interest of the research community, prompt-based learning is still a very new method and there is thus limited work available, especially in the field of optimizing the method for a specific task. Particularly in the field of Emotion Recognition in Conversation, at the time of our experiments, we managed to find no work utilizing a pure prompt-based learning approach, that only trains the prompt parameters, keeping the model frozen and remaining truly lightweight.

In this thesis we aim to study the applicability of prompt-based learning to the adaptation of a large pre-trained language model to the task of Emotion Recognition of Conversation. In the first part of our work, we set a baseline for prompt-based ERC and experiment with methods commonly used in previous work for integrating speaker-specific information and modeling inter- and intra-dependencies. We compare our models to fine-tuned models, and measure the difference in performance between the two adaptation methods. We then provide as second approach, which aims at encoding additional, useful for the ERC task information, such as speaker identity or topic, directly through prompts, without any further changes to the input format or model architecture. We perform extensive experiments for both approaches and provide a discussion on the applicability of prompt-based learning to the task of ERC.

1.2 Contributions

Through our work we contribute the following:

- We provide a baseline for the task of Emotion Recognition in Conversation using a purely prompt-based approach with no tuning of the pre-trained model's parameters.
- We employ previous methods for integrating speaker-specific information and encoding inter- and intra-dependencies that are commonly utilized in the fine-tuning setting, and study whether a language model adapted through prompt-based learning can still benefit from such methods, or a different approach is needed.
- We compare the performance of prompt-based and fine-tuned models in the ERC task and determine that prompt-based learning can indeed prove an effective alternative adaptation approach for the specific task, and as such is worth investigating further.
- We propose an approach for integrating additional information to the task of ERC directly through prompts, with no additional changes to the model's architecture or input format. Our approach is information-agnostic, in the sense that, while we experiement with speaker-identity and topic, it can easily be extended to other types of information that are deemed to be beneficial for ERC, following the logic presented in this work.

1.3 Thesis Outline

In Chapter 2, Machine Learning and Deep Learning, we provide the theoretical background to prepare the reader for the next chapters, familiarizing them with the concepts of machine learning and deep learning and presenting the basic machine learning types. We explain some fundamental notions related to the training of machine learning models, and present some commonly used methods and models both in the field of traditional machine learning and in the field of deep learning.

In Chapter 3, Word Representation Methods and Language Models, we provide an overview of the field of Natural Language Processing, presenting different representation methods and language models that have been developed throughout the years to model and understand human language. We then delve into pre-trained language models, presenting some of the most popular architectures and the different methods that have been proposed for adapting them to downstream tasks .

In Chapter 4, Prompt-based Learning, we analyze the adaptation method called prompt-based learning, which we utilize in this work, providing details regarding its architecture and benefits and presenting an overview of previous work related to our usage of prompt-based learning.

In Chapter 5, Emotion Recognition in Conversation, we define the problem of Emotion Recognition in Conversation (ERC), which we tackle in the following chapter. We also provide an overview of the most important variables for identifying emotion in a conversational setting and present the main approaches followed in previous work. Finally, we present two datasets created for the task, which we use to perform our experiments.

In Chapter 6, Proposed Approaches, we analyze the two approaches our work follows, explaining their purpose, as well as our method and presenting the results obtained from our experiments. We also provide a discussion over our results, explaining our conclusions.

In Chapter 7, Conclusions, we provide a summary of our methods and findings and suggest some potential approaches for future work.

Chapter 2

Machine Learning and Deep Learning

2.1 Introduction

Artificial intelligence (AI), the creation of machines that can "think", has been an exciting concept for humanity for many years. Even in ancient Greece, people devised the existence of creatures such as Talos and Pandora, which may be considered as artificially intelligent machines [47]. The development of Artificial Intelligence however ultimately depended on the development of computer systems. The earlier efforts in AI usually evolved around formal tasks, where formal languages where used to express statements, about which the computer could reason using logical inference rules. However, the success of such systems was limited, due to the difficulty of sufficiently describing all possible knowledge for a given task, especially in the case of more intuitive to humans tasks, such as the recognition of a voice or an image.

This led to the idea of designing AI systems that will be capable of recognizing patterns from the data on their own, in order to learn through experience and extract knowledge that will be useful to them, for the purpose of solving specific tasks. This is known as Machine Learning (ML). Many machine learning algorithms attempt to extract a mapping from the representation of the data to the output, learning how each piece of information included in a representation (feature) correlates with various outputs [47]. For this purpose, a set of features can be extracted and provided to the machine learning algorithm. A problem with this approach is that it is not always easy to know what the best features that will provide the most useful information for the solution of each task are. To solve this, machine learning can be used not only to learn the mapping between a representation and an output, but also to learn the representation itself (an approach known as representation learning).

Representation learning algorithms have the ability to identify the set of features best suited for each task from the raw data provided (training data) and can lead to very high performance for various applications. However, extracting high-level, abstract features from raw data can be challenging, due to data complexity and to the many different factors influencing the data's variation [47]. For this reason, deep model architectures are used, that build representations based on order, expressing the more complex representations of higher model layers with the use of simpler representations of lower layers. This technique, called Deep Learning (DL), allows the solution of many complex tasks nowadays, from image segmentation and machine translation to emotion recognition and the creation of conversational agents. It is researched extensively and it is utilized in an increasing number of applications, both of research and commercial nature.

In the following sections we first define the four main types of machine learning. We then analyze the two most popular machine learning types, supervised and unsupervised learning, and present some of the most well-known traditional algorithms in each category. We consequently provide an analysis of some basic parts of the neural network architecture and training, before moving on to an analysis of some additional basic concepts in the field of machine learning and deep learning. Finally, we present some of the most commonly used deep learning models.

2.2 Types of machine learning

Based on the setting used for training a machine learning algorithm, we can define four main types of machine learning:

- Supervised learning
- Unsupervised learning
- Semi-Supervised learning
- Reinforcement learning

In the following sub-sections we explain the basic attributes that characterize each of the above learning categories.

2.2.1 Supervised learning

In supervised learning, the machine learning algorithm is trained using labeled data, in other words data for which the correct output is determined. The machine is provided with a dataset which includes both the inputs and the desired outputs and the algorithm uses this data to determine how to predict the correct outputs, based on the inputs provided.

More formally, in supervised learning, given a set of input variables x and output variables y, an algorithm is used in order to learn a mapping function f from the input X to the output Y, such that:

$$Y = f(X) \tag{2.1}$$

wherein both the labeled data X and their corresponding outputs Y are provided and used in the learning process.

Due to the fact that the data needs to be labeled, usually by a human agent, supervised learning datasets are often small or medium in size, which can pose a limitation for the performance of supervised learning models.

2.2.2 Unsupervised learning

In unsupervised learning, all of the data is unlabeled (only the input X and not the output Y is available). The unsupervised learning algorithms' goal is to learn the underlying structure or distribution in the data.

Because unsupervised learning's data is unlabeled, no human labor is required to create the datasets, which are thus often substantially larger compared to datasets used for supervised learning. In addition, unsupervised learning algorithms can more easily adapt to new data, by dynamically shifting the underlying data structures they have learned.

2.2.3 Semi-supervised learning

In semi-supervised learning, a large amount of input data is available, but only some of this data is labeled. It is a combination of supervised and unsupervised learning and it is used in situations where only few labeled training samples but a large number of unlabeled training data are available [48]. Semi-supervised learning trains an initial model on the few labeled examples and uses the additional unlabeled data to better estimate the shape of the underlying data distribution and generalize better to new samples.

2.2.4 Reinforcement learning

Reinforcement learning provides the machine learning algorithm with a work environment that contains a set of actions, rewards and end values. The machine learning algorithm attempts to explore different options, while evaluating each result to decide on the best action. When the algorithm finds a correct solution a reward is provided, while in the case of a non positive outcome the algorithm iterates again, searching for a better result. The reward provided may not be an absolute value; it may depend on the effectiveness of the result. The program thus learns to adapt its approach in order to achieve the best possible reward in the environment provided.

2.3 Supervised learning

Supervised learning algorithms can be separated into two types, based on the type of value they predict: regression and classification:

2.3.1 Regression

Regression is the task of determining a mapping function from independent (input) variables to dependent (output) variables. The output values which the regression algorithm attempts to predict are continuous. Common regression problems include weather prediction, stock price prediction, house price prediction etc. A simple model used for regression tasks is linear regression:

Linear Regression

Let $\mathbf{x} \in \mathbb{R}^n$ be an input vector and $y \in \mathbb{R}$ be a scalar value to be predicted as its output. In linear regression, y is considered to be a linear function of the input $\mathbf{x}$. We define $\hat{y}$ as the value approximating y that the linear regression model predicts. We can then express $\hat{y}$ as:

$$\hat{y} = \mathbf{w}^\top \mathbf{x} \tag{2.2}$$

where $\mathbf{w} \in \mathbb{R}^n$ is a vector of parameters [47].

The vector $\mathbf{w}$ may be considered as a set of weights, each determining how the corresponding feature x_i with which it is multiplied ultimately affects the output. The larger the absolute value of the i-th weight w_i is, the bigger an effect the x_i has for $\hat{y}$, while a weight equal to zero means that the corresponding feature does not influence the model's prediction.

In order to determine the optimal parameters $\mathbf{w}$ for the linear regression model, we must first determine a measure for its performance. We can define the mean squared error (MSE) as:

$$MSE = \frac{1}{m}\sum_i (\hat{\mathbf{y}} - \mathbf{y})_i^2 \tag{2.3}$$

where $\hat{\mathbf{y}}$ are the predictions of the model for out data, $\mathbf{y}$ are the correct outputs and m is the number of samples we have. The goal is thus to minimize the MSE, with an MSE equal to zero meaning that all our models predictions are accurate ($\hat{\mathbf{y}} = \mathbf{y}$). A simple way of minimizing the MSE is to solve for where its gradient is zero:

$$\nabla_{\mathbf{w}} MSE = 0 \tag{2.4}$$

which leads to the solution:

$$\mathbf{w} = (\mathbf{X}^\top \mathbf{X})^{-1} \mathbf{X} \mathbf{y} \tag{2.5}$$

where $\mathbf{X}$ is the matrix consisting of all input vectors x and $\mathbf{y}$ is the vector consisting of the corresponding outputs.

2.3.2 Classification

Classification is the task of determining a mapping function from independent (input) variables to discrete output variables. The output variables are usually called labels. In other words, classification is the task where the computer program is asked to specify to which class from a set of classes each input belongs to. Common classification problems include image recognition, sentiment analysis, spam email detection, etc. In the following paragraphs we present some of the most popular traditional machine learning models, used for classification:

Logistic Regression

In the case where we perform classification between only two classes, the problem is called binary classification. For binary classification, we can model the probability of a data sample with a feature vector $\boldsymbol{\varphi}$ belonging to class C_1 as:

$$p(C_1|\boldsymbol{\varphi}) = y(\boldsymbol{\varphi}) = \sigma(\mathbf{w}^\top \boldsymbol{\varphi}) \tag{2.6}$$

where $\mathbf{w}$ is a vector of parameters and:

$$\sigma(\alpha) = \frac{1}{1 + exp(-\alpha)} \tag{2.7}$$

is the logistic sigmoid function.

The probability of the data sample belonging to the second class C_2, can then be calculated as:

$$p(C_2|\boldsymbol{\varphi}) = 1 - p(C_1|\boldsymbol{\varphi}) \tag{2.8}$$

To determine the parameters of the logistic regression model we can use maximum likelihood: We can write the likelihood function as:

$$p(\mathbf{t}|\mathbf{w}) = \prod_{n=1}^{N} p(C_1|\varphi_n)^{t_n} (1 - p(C_1|\varphi_n))^{1-t_n} \tag{2.9}$$

where N is the number of training samples, $t_n \in \{0, 1\}$ is the class label for sample n, φ_n is the feature vector for the sample n and $\mathbf{t} = (t_1, ...t_N)^\top$ is the vector of labels for all samples. By taking the negative logarithm of the likelihood function, we can then define the error function to be minimized as:

$$E(\mathbf{w}) = -lnp(\mathbf{t}|\mathbf{w}) = -\sum_{n=1}^{N}\{t_n lnp(C_1|\varphi_n) + (1 - t_n)ln(1 - p(C_1|\varphi_n))\} \tag{2.10}$$

which is called the crossenteropy error function.

The logistic regression model described can only be used for binary classification. However logistic regression can be extended to more than two classes: One way for achieving that is using the one-to-rest approach, which trains a logistic regression model for each class, modeling the probabilty of a sample belonging to this class versus the probability of the sample beloning to any of the other classes. The predicted class is then the class whose logistic regression model yields the highest probability. A second approach is multinomial logistic regression, a simple extension of the binary logistic regression, which, instead of the logistic sigmoid function, uses the softmax function:

$$p(C_k|\boldsymbol{\varphi}) = \frac{exp\{a_k\}}{\sum_j exp\{a_j\}} \tag{2.11}$$

with:

$$a_k = \mathbf{w}_k^\top \boldsymbol{\varphi} \tag{2.12}$$

where $\mathbf{w}_k$ is the parameter vector for class k and $p(C_k|\boldsymbol{\varphi})$ is the probability of a sample with a feature vector $\boldsymbol{\varphi}$ belonging to class k [49].

Naive Bayes classifier

The Naive Bayes classifier is a probabilistic machine learning model used for classification, which is based on the Bayes Theorem. For a class y and a feature vector $\mathbf{x} = (x_1, x_2, ..., x_n)$, the Bayes Theorem can be expressed as:

$$P(y|\mathbf{x}) = \frac{P(\mathbf{x}|y)}{P(\mathbf{x})} \tag{2.13}$$

Assuming conditional independence for the features of $\mathbf{x}$, we can then write:

$$P(y|\mathbf{x}) = P(y|x_1, x_2, ..., x_n) = \frac{P(x_1|y)P(x_2|y)...P(x_n|y)P(y)}{P(x_1)P(x_2)...P(x_n)} \tag{2.14}$$

Considering that the demoninator of the above fraction remains the same for all values of y (for all classes), we can classify each sample to the class $\hat{y}$ where:

$$\hat{y} = argmax_y(P(y)\prod_{i=1}^{n} P(x_i|y)) \tag{2.15}$$

When the input variables are continuous, we can assume that the features are sampled from a Gaussian distribution. We then model the probability $P(x_i|y)$ as:

$$P(x_i|y) = \frac{1}{\sqrt{2\pi\sigma_y^2}} exp(-\frac{(x_i-\mu_y)}{2\sigma_y^2}) \tag{2.16}$$

and obtain the Gaussian Naive Bayes classifier.

The Naive Bayes classifier is often used because of its speed and simplicity. However, because it assumes a conditional independence of the input features, it often leads to inferior performance, as this assumption is not valid in most real-world problems.

K-Nearest Neighbours

The K-Nearest Neighbours (KNN) algorithm is a non-parametric classifier, which makes predictions about the grouping of a data point based on its proximity to other data points. The algorithm calculates the distance of a test sample from all the training points and selects the K points that are the closest to the test point. It then classifies the test point to the class which has the most training data points, from the K points previously selected.

While different distance measures can be used, the most common is the Euclidean distance, which, for two points $x = (x_1, x_2, ..., x_n)$ and $y = (y_1, y_2, ..., y_n)$ can be defined as:

$$d(x, y) = \sqrt{(x_1 - y_1)^2 + (x_2 - y_2)^2 + ... + (x_n - y_n)^2} \tag{2.17}$$

We must note that, while the KNN algorithm is most commonly used for classification, it may also be applied to regression tasks, in which case the predicted value can be calculated as the mean of the K nearest training data points.

Support Vector Machines

Support vector machines (SVM) can be used for both classification and regression problems, however the algorithm is most commonly used for classification. Let $\mathbf{x}_i, i = 1, 2, ..., N$ be the feature vectors of the training set, for a binary classification problem with two linearly separable classes, ω_1 and ω_2. (We call two classes linearly separable if we can separate the data belonging to each class using a linear function. In the 2-d dimension, this corresponds to a straight line). The goal of SVM is to calculate the optimal hyperplane:

$$g(\mathbf{x}) = \mathbf{w}^\top \boldsymbol{\varphi}(\mathbf{x}) + w_0 = 0 \tag{2.18}$$

that correctly classifies all training samples, were $\boldsymbol{\varphi}(\mathbf{x})$ is a feature-space transformation [49][50]. The optimal hyperplane is the hyperplane with the biggest margin, where we define the margin as the smallest distance between the decision boundary and any of the samples. The location of the optimal hyperplane is calculated using a subset of the training data points. The data points of this subset are called support vectors, which is were the name of the algorithm originates from.

In the above definition we have assumed that the two classes are linearly separable. We note that this assumption is valid, as it can be proven that, using an appropriate mapping $\varphi()$ to a sufficiently high dimension, data belonging to one of two classes can always be separated by a hyperplane (is linearly separable in the higher dimension space) [51].

The SVM algorithm discussed above can be used for binary classification problems. In the case of multiple classes, we can extend SVM using the one-to-one or the one-to-rest approach. The first approach uses $\frac{m(m-1)}{2}$ SVMs for m classes, calculating a hyperplane to separate between every two classes, without taking into account the data of all the other classes. The second approach uses m SVMs for m classes, each separating the data of one class from the data of all the other classes.

2.4 Unsupervised learning

Two of the main tasks unsupervised learning algorithms are utilized for are clustering and dimensionality reduction:

2.4.1 Clustering

Clustering can be defined as the task of grouping unlabeled data, based on their similarities. It is often used in applications such as social media analysis, anomaly detection, search result grouping etc. Clustering algorithms can be separated into multiple different types, based on the method used to group data. Some of the most prominent are:

- **Hierarchical clustering**:

  Hierarchical clustering algorithms are divided into Agglomerative and Divisive clustering algorithms. The first follow a bottom-up approach, initially considering each data point as a different cluster and iteratively merging similar clusters, until the desired number of clusters is reached. The second follow a top-down approach: They begin with all the data in the same cluster and split a cluster into two in each iteration, based on dissimilarity.

- **Centroid-based or Partition clustering**:

  In this type of algorithms, the data points are divided into a predefined number of clusters, and a vector is calculated, which represents each cluster. The distance of each data point to each of the vectors representing the clusters is calculated, and the data point is assigned to the cluster with the minimal distance. Distance measures often used are the Euclidean, the Manhattan and the Minkowski distance. One of the most popular algorithms of this category is the K-Means algorithm, which will be described later in this sub-section.

- **Density-based clustering**:

  Density-based clustering algorithms prioritize density over distance. Data is organized into regions with high concentrations of data objects that are surrounded by low concentration regions.

- **Probabilistic clustering**:

  In probabilistic clustering algorithms, data are clustered based on the probability that they belong to the same distribution, with each data point beeing assigned to the cluster to whose distribution it belongs with the biggest probability. These probabilities are estimated through the use of a suitably formulated optimization problem. The most commonly used distributions are the Normal and Gaussian distributions.

Based on the type of grouping the clustering algorithms calculate for each data point, they can additionally be divided into two categories:

- **Hard clustering**:

  Hard clustering algorithms group data, assigning only one cluster to each data point: Each data point either belongs to a cluster or it does not.

- **Soft clustering**:

  Soft clustering algorithms group data using probability: They calculate the likelihood for each data point to belong to each cluster.

K-Means

K-Means is an unsupervised clustering algorithm that partitions the data into K pre-defined, non-overlapping clusters, with each data point belonging to only one cluster (hard clustering).

Considering N data points $\mathbf{x}_1, \mathbf{x}_2, ..., \mathbf{x}_N$, and K clusters, it assigns a vector $\boldsymbol{\mu}_k$ to each cluster. Its goal is then to minimize the objective function:

$$J = \sum_{n=1}^{N} \sum_{k=1}^{K} r_{nk} ||\mathbf{x}_n - \boldsymbol{\mu}_k||^2 \tag{2.19}$$

where, for each data point $\mathbf{x}_n, n = 1, 2, ..., N$ and cluster $k, k = 1, 2, ..., K$, $r_{nk} = 1$ if $\mathbf{x}_n$ is assigned to cluster k and $r_{nk} = 0$ otherwise. The objective function J thus represents the sum of the distances of each data point to the vector $\boldsymbol{\mu}_k$ of its assigned cluster [49].

The algorithm calculates the values for r_{nk} and μ_k which minimize J. For this purpose, after choosing some starting values for μ_k, is follows an iterative process, during which, in each step it first minimizes J with respect to r_{nk}, keeping $\boldsymbol{\mu}_k$ fixed and then minimizes J with respect to $\boldsymbol{\mu}_k$, keeping r_{nk} fixed. Ultimately, in the first phase, for a data point $\mathbf{x}_n$, r_{nk} is set to 1 for the index k of the cluster with the smallest value $||\mathbf{x}_n - \boldsymbol{\mu}_k||^2$ and is set to 0 for all other cluster indexes. In the second phase, the new value of $\boldsymbol{\mu}_k$ for each cluster k is calculated as:

$$\boldsymbol{\mu}_k = \frac{\sum_n r_{nk} \mathbf{x}_n}{\sum_n r_{nk}} \tag{2.20}$$

The iterative process continues, until the values r_{nk} remain the same, so until the assignment of each data point to a cluster does not change.

2.4.2 Dimensionality reduction

Dimensionality reduction is the task of reducing the dimension of a set of data features. In machine learning, data can often contain hundreds of columns (features), increasing the difficulty of both visualizing the training data and training a model on it and leading to a set of problems that occur when working with very-high dimensional data, known as the Curse of Dimensionality.

One of those problems, is data sparsity: When the dimension of the data is very high, the combinations of values between each sample's attributes that occur in the training set and from which the machine learning model learns to predict the output, becomes a small subset of the combinations of values between attributes that can occur. Training a machine learning model on such sparse data can lead to overfitting, as the model may learn according to the values of attributes in the train set and achieve a high performance during training, but will not be able to generalize effectively on the test set, if many new, largely different combinations of attribute values occur in it.

A second problem is that, as the dimensionality of the data increases, the distances between different data points in the feature space may converge to the same value. This renders the distance-based measures that many machine learning algorithms (like KNN or K-Means) use trivial and unable to encode similarity in a meaningful way.

The purpose of dimensionality reduction techniques is thus to reduce the dimension of data feature vectors, while maintaining the data's integrity as much as possible. In the following paragraph we present one of the most common techniques used for dimensionality reduction, PCA:

PCA

The Principal Component Analysis (PCA) technique performs an orthogonal projection of the data to a lower dimensional linear space, in such a way as to maximize the variance of the low-dimensional data, encoding as much information of the high dimensional representations as possible. Using statistical analysis, PCA calculates the first principle component as the direction at which the projections of the data present the biggest variance. The rest of the principal components can be defined incrementally, each as the next direction to maximize variance, with the restraint that it is orthogonal to the components already specified. To obtain a lower dimensional representation of our data we can then only keep the projections of the data to the M first principal components, were M is the desired dimensionality.

2.5 Artificial Neural Networks

The Artificial Neural Networks (ANNs) lie at the basis of deep learning. Their structure is inspired by the human brain's biological neuron's, which is the reason for their name. Traditionally, ANNs consist of nodes (neurons) which are organized in layers. Specifically, ANNs are comprised of an input and an output layer and, between those, one or more hidden layers. Each node may be connected to one or more other nodes, and each connection is associated with a weight. To obtain the output of each node, the sum of the output of the nodes of the previous layers with which it is connected is calculated. However, this calculation is a linear function of the outputs of previous layers. If we consider this as the output of the node, then all operations in the neural network will be linear, thus reducing the output of the neural network to a linear function of the input and a single layer. To be able to stack multiple layers of neurons and learn complex relations from the data of each task, we thus need to use a non-linear function to calculate the neuron output, which is called an activation function. In addition, when training the neural network, we need to define a metric which will allow us to calculate the error between our model's predictions and the true labels for our dataset. This is called a loss function, and the neural network's objective is to minimize it during training. For this, an algorithm able to compute the necessary gradients is used, which is called back-propagation. In the following subsections we analyze the notions mentioned above, namely the activation function, the loss function and back-propagation.

2.5.1 Activation function

As analyzed, the activation function is a non-linear function that takes as input the weighted sum of the outputs of previous nodes (or the input data in the first layer) and calculates the output of a neural network's node. Activation functions are usually differentiable, in order to be able to be used with the back-propagation algorithm (back-propagation is analyzed in following paragraphs) and they typically differ between the input and hidden layers of the neural network and the output layer of the neural network.

Some of the most popular activation functions are:

- **Linear/Identity activation function**:

  The linear activation function is defined as:

  $$f(x) = x \tag{2.21}$$

  Becasue this activation function is linear, it is not commonly used in neural networks, as it does not provide the needed non-linearity, leading to all network layers being equivalent to one layer, as the output layer becomes a linear function of the input layer. A second problem with this function is that is cannot be used with back-propagation, as is has a constant derivative, which does not depend on the input.

- **Sigmoid/Logistic activation function**:

  The sigmoid activation function is defined as:

  $$f(x) = \frac{1}{1 + e^{-x}} \tag{2.22}$$

  It is most suitable for models that predict a probability, as it has an output range of $(0, 1)$. It provides a smooth gradient when differentiated and does not lead to abrupt changes in the output. However, because of the fact that its gradient's values tend to be very small for the biggest range of input x, it suffers from the vanishing gradient problem. (This problem occurs when gradients become very small and as a result the neural network's weights are prevented for changing their value during back-propagation). In addition, it is not symmetric around zero, a property important for the stability of the training of neural networks. Finally it is computationally expensive. For these reasons, it is not often used for neural networks today.

- **Tanh (Hyperbolic tangent) activation function**:

  The tahn activation function is defined as:

  $$f(x) = \frac{e^x - e^{-x}}{e^x + e^{-x}} \tag{2.23}$$

It is very similar to the sigmoid activation function, but it is zero-centered. It is mostly used in the hidden layers of neural networks, as its values have a range of $(-1, 1($ and the mean hidden layer output is thus close to zero, which makes the neural network's training easier. However, as the sigmoid function, the tanh activation function suffers from the vanishing gradient problem.

- **Rectified Linear Unit (ReLU) activation function**:

  The Rectified Linear Unit (ReLU) activation function is defined as:

$$f(x) = max(0, x) \tag{2.24}$$

  The ReLU activation function is very efficient computationally, and it does not lead to saturation or cause the vanishing gradient problem. However, because it has a zero gradient for all negative inputs, some of the neural network's neurons are not updated during back-propagation, which can lead to dead neurons which are never activated during the training process. This is known as "the dying ReLU problem".

- **Leaky ReLU activation function**:

  The Leaky ReLU activation function is defined as:

$$f(x) = \begin{cases} x & \text{for } x > 0 \\ ax & \text{for } x \leq 0 \end{cases} \tag{2.25}$$

  where a is a small, non-zero constant, ususally around 0.01.

  The leaky ReLU actication function has the same benefits as ReLU, while it allows the usage of back-propagation for negative inputs as well.

2.5.2 Loss function

The loss function can be defined as an overall measure of the loss associated with specific decisions or actions. When training a model in supervised learning, we need a metric to be able to calculate the distance between our model's predictions and the true labels. We can thus use a loss function that maps the set of predicted labels $\hat{y}$ and the set of true labels y to an real number, representing the total loss (error) of our model. The goal of the training then becomes to minimize this loss function.

We present some of the most widely used loss functions in the following paragraphs. In the equations below, $\mathbf{w}$ represents the vector of parameters of the model, $\hat{y_i}$ represents the model's prediction for sample i and y_i represents the actual label for sample i, where we make predictions for N samples.

Loss functions used for regression problems:

- **Mean Squared Error (MSE)**:

  The Mean Squared Error (MSE) loss function is defined as:

$$L(\mathbf{w}) = \frac{\sum_{i=1}^{N}(y_i - \hat{y_i})^2}{N} \tag{2.26}$$

Due to the usage of squares in the MSE loss function, predictions that are further from the true labels contribute much more to the total error compared to predictions that are less deviated. In addition, the sign of the difference does not affect the result (again due to squaring) and only the magnitude is considered.

- **Mean Absolute Error (MAE)**:

  The Mean Absolute Error (MAE) is defined as:

$$L(\mathbf{w}) = \frac{\sum_{i=1}^{N}|y_i - \hat{y_i}|}{N} \tag{2.27}$$

  Like the MSE loss, the MAE loss only considers the magnitude and not the direction of the difference between the true and predicted label. Due to the fact that it does not use squaring, it does not penalize bigger differences as heavily as the MSE loss, and is thus more robust to outliers.

Loss functions used for classification problems:

- **Kullback-Leibler Divergence**:

  The Kullback-Leibler divergence is defined as:

$$KL(P, Q) = -\sum_{x} P(\mathbf{x})log\frac{Q(\mathbf{x})}{P(\mathbf{x})} \tag{2.28}$$

  where $P(\mathbf{x})$ and $Q(\mathbf{x})$ are the true and predicted distributions for the input $\mathbf{x}$ respectively.

- **Cross-entropy loss**:

  The cross-entropy loss function can be defined as:

$$H(P, Q) = -\sum_{\mathbf{x}} P(\mathbf{x})logQ(\mathbf{x}) \tag{2.29}$$

  where $P(\mathbf{x})$ and $Q(\mathbf{x})$ are the true and predicted distributions for the input $\mathbf{x}$ respectively. The cross-entropy loss function is the most commonly used loss function for classification problems.

2.5.3 Back-propagation

When working with neural networks, we use a loss function to measure the loss (error) of our network, which we attempt to minimize during training. For this purpose, we need to compute the gradient of the loss function, with respect to the neural network's weights. In the hidden layers of a neural network, the input to every node is determined by the previous layers, while the output of the node affects the final model predictions and thus the value of the loss function. This leads to a coupling of the parameters between layers and makes the calculation of gradients with respect to each parameter very complex. Therefore, we need a method to efficiently compute the gradients. The method we use is the back-propagation algorithm, popularized by Rumelhart et al. in [52], although, as a method, it was introduced as early as the 1960s.

After each forward pass in the neural network, the back-propagation algorithm performs a backward pass, working with one layer of the network at a time, from the last layer to the starting layer and using the chain rule to compute the gradient of the loss function with respect to each model weight, leveraging the results of the previous layers. An important advantage of this algorithm is that is offers an understanding of the effect that a change in a model weight can have to the loss function and overall model behaviour.

2.6 Additional machine learning concepts

In the following sub-sections we analyze some of the most basic concepts related to machine learning and the training of machine learning models.

2.6.1 Generalization, overfitting and underfitting

Machine learing models are trained using a train set. During training, the training error is calculated and the objective is the minimization of this training error, so the achievement of the best possible performance on the train data. However, ultimately, the success of a trained machine learning model is evaluated not only based on its performance on the data seen during its training, but also based on its performance on new, previously unseen data, which are called test data. The ability of a machine learning model to achieve accurate predictions for unseen data is called generalization, and the error obtained on this data is called the generalization or test error [47].

To understand why generalization is important, we must consider that training data may often not be representative of all possible cases for a specific problem, consisting only a subset of the possible input, especially in cases of noisy or sparse data and small datasets. Test data may thus present new, unseen by the model values and relations between features. In such cases, the machine learning model can have a small training error, but be unable to handle the unseen data effectively, obtaining a bigger test error.

Ultimately, the machine learning model can face two problems, affecting its overall performance:

- **Underfitting**:

  It occurs when the model is not able to capture the relation between the input features and the output and thus sufficiently model the training data. In this case, both the training error and the test error will not be sufficiently small.

- **Overfitting**:

  In this case the model can adapt to the training data well, but cannot generalize, meaning that, while it is able to achieve a small tranining error, the test error is substantially larger. Overfitting often occurs when the model learns the training data two closely, modeling properties of the training set such as noise that do not occur in the test set, instead of modeling the general data distribution.

Preventing overfitting

Overfitting is a problem that occurs very often while training machine learning models, especially deep learning models that can have millions of trainable parameters and are thus capable of fitting the training data too closely. To prevent it, a variety of methods have been developed. Some of the most widely used methods that are utilized to prevent overfitting are regularization, dropout and early stopping. We present these methods in the following paragraphs:

Regularization: Regularization aims at reducing overftting by forcing the model to learn smaller weights and thus leading to a less flexible model, that does not adapt to the training data too closely. It does so by adding a norm penalty term (regularization term) to the loss function, which imposes a bigger penalty for a bigger norm of the parameter values. For a model with parameters $\mathbf{w}$ and a loss function $L(\mathbf{w})$, the loss function after regularization is defined as:

$$\tilde{L}(\mathbf{w}) = L(\mathbf{w}) + \alpha\Omega(\mathbf{w}) \tag{2.30}$$

where $\Omega(\mathbf{w})$ is the norm penalty term and a is a hyperparameter that determines the contribution of the penalty term to the final value of the loss function.

Based on the norm used by the norm penalty term, different regularization methods can be defined. Two of the most popular are:

- **L1 Regularization**:

  The L1 norm penalty, also known as Lasso regression, uses the L1 norm (also called Manhattan distance), and is thus defined as:

  $$\Omega(\mathbf{w}) = \sum_{i=1}^{N} |w_i| \tag{2.31}$$

  where $w_i, i = 1, 2, ..., N$ are the model's parameters.

- **L2 Regularization**:

  The L2 norm penalty, also known as weight decay or ridge regression, uses the L2 or Euclidean norm, and is thus defined as:

  $$\Omega(\mathbf{w}) = \sum_{i=1}^{N} w_i^2 \tag{2.32}$$

  where $w_i, i = 1, 2, ..., N$ are the model's parameters.

Comparing the two regulation methods, we can conclude the following: Due to squaring, L2 regularization penalizes parameters with bigger values much more strongly, while it only affects smaller values lightly. It can therefore reduce overfitting by decreasing model complexity, but, since it does not lead any parameters to become equal to zero and only decreases them, it does not reduce the total parameter number. On the other hand, L1 regularization affects all values equally, decreasing all non zero parameters and leading some of them to assume an optimal value of zero. For this reason, L1 regularization often leads to more sparse models than L2 regularization and, in addition to overfitting, it may also be used for feature selection, with the features that are passed through zero-valued weights being discarded [47].

Dropout: Dropout is a regularization method widely used for preventing overfitting in neural networks. During training, for each training example, it ignores (sets to zero) a number of neuron outputs, each randomly chosen with a probability of p. Dropout can be thought of as training an ensemble of networks, the networks that occur after the random neurons are "dropped". In this way, the final output does not rely too much on a specific neuron connection. A schematic presentation of the dropout method is depicted in Figure 2.1.

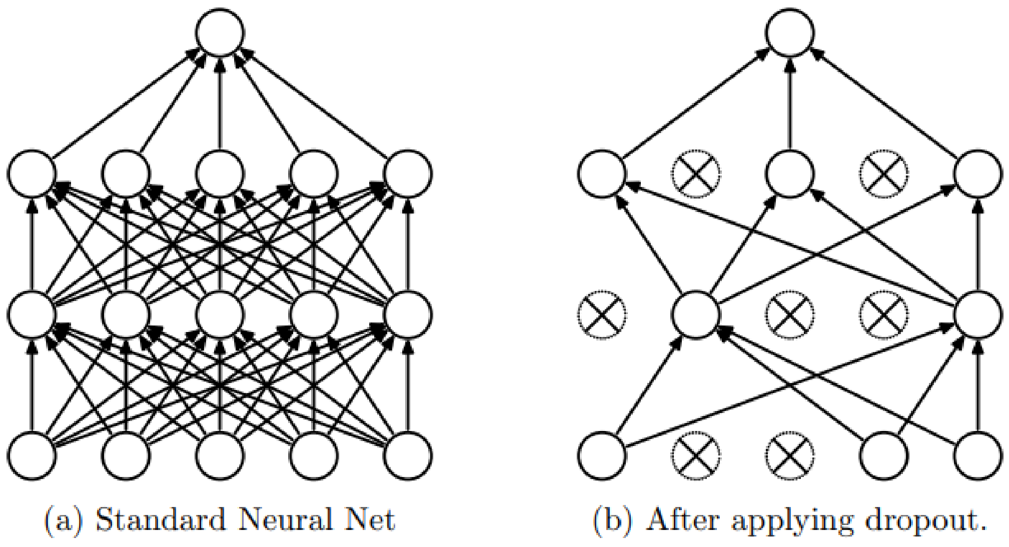


Figure 2.1. *The dropout method for a simple neural network with two hidden layers. Left: The neural network before dropout. Right: The neural network after applying dropout. Image obtained from: [1]*

Early stopping: While training a machine learning model, up to a point, both the training error and the validation error (the test error on the validation set) tend to decrease in each epoch. However, as the training continues and the model is learning the training data more and more closely, the validation error begins to rise again (this is a sign that overfitting is starting to occur). For this reason, we use early stopping: We keep a copy of the model every time the validation error improves, and choose these and not the latest parameters as our final model. In most cases, we also keep track of the validation error and if it has not decreased for a predefined number of epochs we stop the training completely.

2.6.2 Evaluation metrics

When training a machine learning model, we need to be able to measure its performance. We have already discussed the usage of loss functions for this purpose. However, when evaluating model performance for a specific application, we may need to prioritize different criteria, other than the total training or validation error. For example, when training a model to detect cancer from images, it is more important for the model to identify all cancer cases, than identify all non-cancer cases correctly, as misdiagnosing a healthy person will only lead him to perform a check-up, while misdiagnosing a cancer patient could lead to his cancer remaining undetected for a longer period of time, with life-threatening consequences. In addition, when working with a largely imbalanced dataset, a model that learns to always predict the majority class could appear to obtain a good total performance. However, it would fail to identify any class, other than the majority class. Therefore, a variety of evaluation metrics are used for evaluating machine learning model performance, depending on the dataset and application. We present some of the most popular evaluation metrics used in classification problems in the following paragraphs.

In a binary classification problem, we can define the class that the classifier's purpose is to identify as "positive" and the other as "negative". For example, a classifier that recognizes photos of people, would then classify them as positive and it would classify all other photos as negative. We call "True Positive (TP)" the number of predictions where both the true and the predicted class are "positve" and "True Negative (TN)" the number of predictions where both the true and the predicted class are "negative". We call "False Positive (FP)" the number of predictions where the predicted class is "positive", but the true class is "negative" and "False Negative (FN)" the number of predictions where the predicted class is "negative", but the true class is "positive".

Based on these definitions, we can then define the following evaluation metrics, for the binary classification case:

Accuracy

Accuracy is defined as:

$$Accuary = \frac{TP + TN}{TP + FP + TN + FN} \tag{2.33}$$

It is the ratio of the number of correct predictions to the total number of data instances. Accuracy measures our model's overall performance. As it treats both classes in the same way, it is not suitable for cases where we consider the accurate prediction of one class to be more important. In addition, accuracy does not take into account data imbalance, so it can be misleading in such cases.

Precision

Precision is defined as:

$$Precision = \frac{TP}{TP + FP} \tag{2.34}$$

For the class we want to identify ("positive" class) precision is a measure of how many of the instances the model classified as "positive" were truly "positive".

Recall

Recall is defined as:

$$Recall = \frac{TP}{TP + FN} \tag{2.35}$$

For the class we want to identify ("positive" class) recall is a measure of how many of the "positive" instances of our data the model identified correctly as "positive".

F1-score

F1-score is defined as:

$$F1\text{-}score = 2 * \frac{Precision * Recall}{Precision + Recall} \tag{2.36}$$

or in terms of the TP, FP, FN values:

$$F1\text{-}score = 2 * \frac{TP}{TP + \frac{1}{2} * (FP + FN)} \tag{2.37}$$

F1-score's purpose is to provide a unified metric that combines precision and recall, requiring both of them to be high, for it to have a high value as well.

We have defined the above metrics for the binary classification case. In the case of multiple classes, accuracy can easily be extended as the ratio of the number of instances classified correctly to the total number of instances in our data. However, precision, recall and F1-score measure the model's performance for a "positive" class against the other "negative" class. We can define all classes except from the class for which we want to evaluate the model's performance as "negative", which will allow as to measure the model's performance for the "positive" class. However, this will provide us with a metric value for each class instead of a unified metric value for all classes, able to measure our model's total performance. To obtain the latter we can use one of the following averaging techniques:

Macro-average

The macro-average computes each per-class metric independently and then averages them, treating all classes equally to obtain the final result. In the case of three classes, A, B, C, it is defined as:

$$Macro\text{-}Avg = \frac{M_A + M_B + M_C}{3} \tag{2.38}$$

where M_i the metric value (precision, recall or f1-score) for class i. Its extension to more classes is considered to be straightforward. Because macro-average treats each class as equally important for the final result, it is often used in cases of imbalanced datasets, were we want our model to perform well on all classes, regardless of the class distribution.

Micro-average

The micro-average aggregates the predictions of all classes in order to calculate the average metric. We provide its definition for precision, recall and f1-score in the case of three classes, A, B, C. Its extension to more classes is considered to be straightforward.

- For precision:

$$Macro\text{-}Precision = \frac{TP_A + TP_B + TP_C}{TP_A + TP_B + TP_C + FP_A + FP_B + FP_C} \tag{2.39}$$

- For recall:

$$Macro\text{-}Recall = \frac{TP_A + TP_B + TP_C}{TP_A + TP_B + TP_C + FN_A + FN_B + FN_C} \tag{2.40}$$

- For F1-score:

$$Micro\text{-}F1\text{-}score = \frac{TP}{TP + \frac{1}{2} * (FP + FN)} \tag{2.41}$$

  where TP, FP, FN are the total number of true positives, false positives and false negatives, for all classes.

Because micro-average treats all data instances as equally important, in the case of a largely imbalanced dataset, the classes with more instances will have a bigger effect on the final score. It is thus not suitable as a metric in such cases, if we care about per-class performance.

Weighted-average

The weighted average is simply a weighted mean of the metric values for each class, were the weight for each class is calculated as the fraction of instances in the data that belong to this class to the total number of instances. In the case of three classes A, B, C it is thus defined as:

$$Weighted\text{-}Avg = w_A * M_A + w_B * M_B + w_C * M_C \tag{2.42}$$

where M_i the metric value (precision, recall or f1-score) for class i and:

$$w_i = \frac{\text{number of instances that belong to class } i}{\text{total number of instances}} \tag{2.43}$$

The extension to more classes is considered to be straightforward.

2.6.3 Transfer learning

Transfer learning is a machine learning method that aims to leverage the knowledge obtained from one task to improve performance in a different, usually relevant problem. In transfer learning, a model or a part of a model trained on a specific task is reused as a starting point for the training of a model for a second task. This is particularly useful in cases where the data available for a target task is few, but there is a related task with significantly more data. In such a case, the representations learned from the first task may be useful for the target task as well, helping it generalize better or faster, even if there are limited example instances for the target task.

2.6.4 Multitask learning

Multitask learning is a method that aims to improve generalization, by training a machine learning model to solve multiple tasks at the same time. In multitask learning, part of the model is shared among multiple tasks (usually the lower layers of the model), learning from all training examples, while part of the model is task-specific, learning from only the specific task's examples (usually the higher layers of the model). This often achieves better generalization, as the part of the model that is shared across tasks can learn shared representations that improve performance on the individual tasks (assuming that the combination of tasks is chosen appropriately) [47].

2.7 Deep learning models

In this section we present some of the most popular types of deep learning models, that have since their creation contributed to the rise of deep learning and the achievement of high performance in multiple problems.

2.7.1 Feed-forward neural networks

Feed-forward neural networks (FFNN) are artificial neural networks (ANNs) in which the connections between nodes do not form a cycle. In feed-forward neural networks, the information is processed in one direction only, from the input layer to the hidden layers and then to the output layer. They were the first kinδ of ANNs to be proposed and are primarily utilized in supervised learning problems, with non-sequential, independent data.

The simplest type of feed-forward neural network is the perceptron. The perceptron has no hidden units. It consists of only one node, which calculates the weighted sum of the input features and then passes the sum through an activation function. For N input features, N weights $w_i, i = 1, 2, ..., N$, and an activation function f, the perceptron's output can be calculated as:

$$y = f(\sum_{i=1}^{N} w_i x_i + b) \tag{2.44}$$

where b is a bias term added to the weighted sum of input features.

The perceptron is a linear model, and it is able to classify linearly separable data. In the case of non-linearly separable data, we need to be able to learn more complex, non-linear functions, using models that consist of more neurons. One of those, and a direct extension of the perceptron is the multi-layer perceptron (MLP).

The multi-layer perceptron is an artificial neural network consisting of multiple perceptrons. It is organized in an input layer, at least one hidden layer and an output layer, with each layer including multiple perceptrons. An example of an MLP can be seen in Figure 2.2. The MLP is able to learn non-linear functions and can be used for both supervised regression and supervised classification problems.

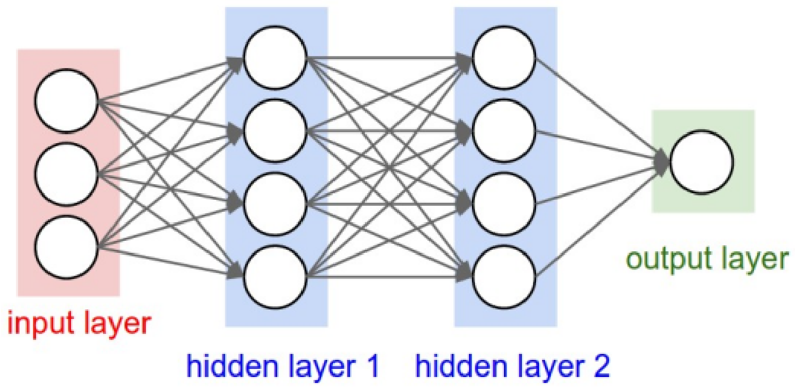


Figure 2.2. *An MLP with three input units, one output unit, and two hidden layers, each consisting of four perceptrons. Image obtained from: [2]*

2.7.2 Convolutional neural networks

Convolutional Neural Networks (CNNs), originally introduced by LeCun et al. [53] are a type of artificial neural network that operates on image input, using convolution. They have an architecture that resembles that of the human visual cortex, in the sense that each of their neurons processes only a specific region of the input, with different such regions overlapping to create the complete input image. The layer of CNNs that performs convolution is similar to a perceptron layer, where each neuron is only connected to a fixed-size input image region, and all neurons of the layer share the same weights. Convolutional neural networks operate in a hierarchical way, with the first layers extracting simpler low-level features from the input (such as lines), and the patterns becoming more complex and related to bigger parts of the image (such as shapes or objects), as we move towards the layers deeper in the network. There are three types of layers that neural networks consist of, convolutional, pooling and fully-connected. The main network is made up of a repetition of convolutional layers, often followed by pooling layers, while the fully-connected layer is applied at the end of the network. In more detail:

- **Convolutional layer**: The convolutional layer contains a set of filters with learnable parameters. Each of the filters is convolved with the image, by being slid across it, with the dot-product between each filter and input element being calculated for every position. The result of this process is the generation of an activation map for each filter. Convolutional layers are often followed by the application of a non-linear activation function to their output, usually a ReLU.

- **Pooling layer**: The polling layer performs a downsampling of its input, reducing its parameters. It uses a kernel which performs an aggregation over the input region it is applied to and is slid ac cross the whole input. By performing a dimensionality reduction, the pooling layer improves the efficiency and reduces the complexity of the network, often also limiting overfitting. The two main types of aggregation performed are max pooling, where the kernel selects the pixel with the maximum value from every input region it is applied on, and average pooling, where the kernel calculates the average value of all pixels of each region.

- **Fully-connected layer**: This layer is the final layer of the convolutional neural network and it is responsible for performing the final classification, using the high level features calculated by the previous CNN layers.

An example of a CNN is presented in Figure 2.3.

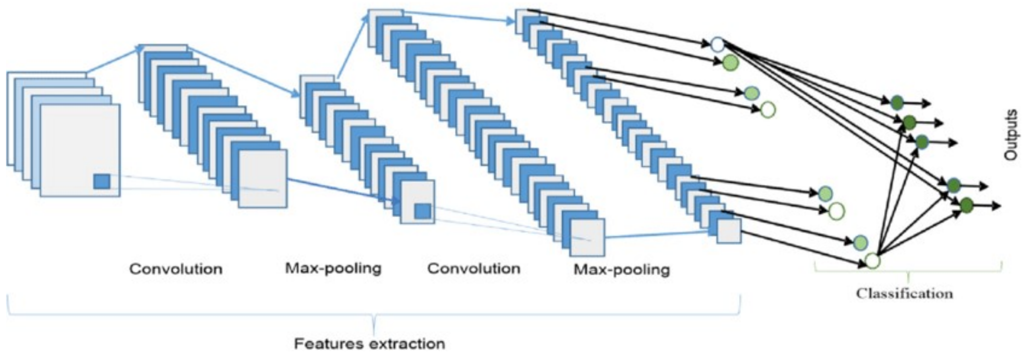


Figure 2.3. *A CNN example. Image obtained from: [3]*

2.7.3 Recurrent neural networks

Feed-forward neural networks perform well on independent data: They process the input independently and do not take into account sequential or temporal relationships between different data points. However, when the data is sequential or is a time series, information about previous inputs is important to the next. For example, the order of words in a sentence can be crucial to its meaning. To model such data, Recurrent Neural Networks (RNNs) are used.

Originally proposed as early as the 1980s [54] [55], an RNN is an artificial neural network that has a feedback loop, which allows data from previous input to be passed back into the node. RNNs can thus be considered as having some sort of "internal memory" called a hidden state, which is continuously updated from the new input and holds past information which it uses in order to influence the new output. Unrolling the feed-back loop of an RNN through time, we can think of RNNs as multiple copies of the same network, one for each input point, each passing information to the next, in a sequential order. We can see both the rolled and the unrolled version of the RNN architecture in Figure 2.4. Because of their ability to model sequential and time series data, RNNs are widely used in applications such as language translation, natural language processing, speech recognition, and image captioning.

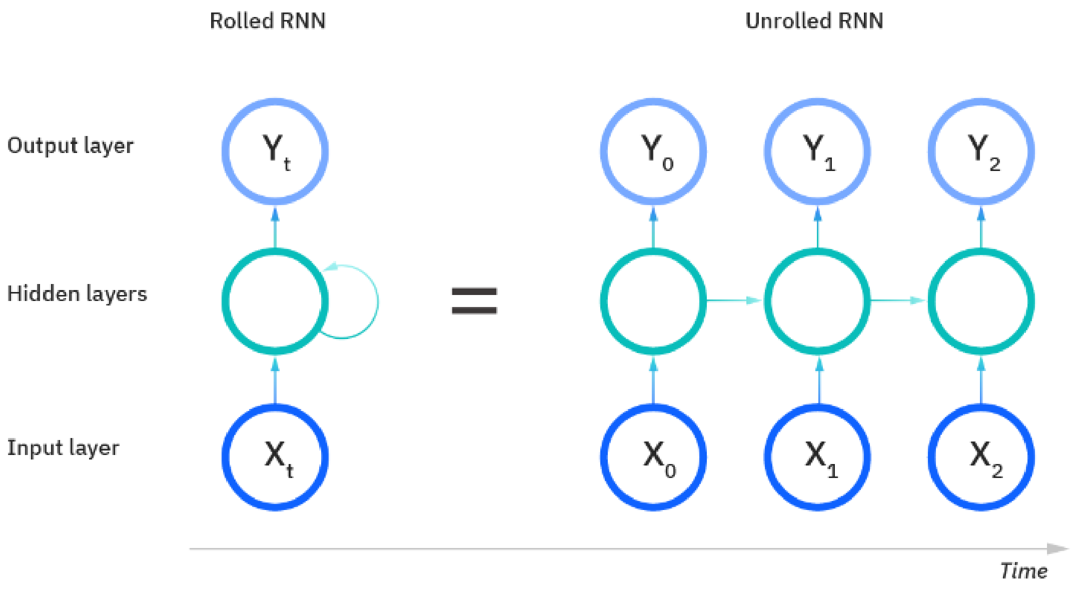


Figure 2.4. *An RNN, where X_t is the input vector and Y_t is the output vector for time step t. Image obtained from: [4]*

Mathematically, we can describe the vanilla RNN architecture using the following equations:

$$h_t = f_h(W_{hx}x_t + W_{hh}h_{t-1} + b_h) \tag{2.45}$$

$$y_t = f_y(W_{yh}h_t + b_y) \tag{2.46}$$

where x_t is the input vector, h_t is the hidden state and y_t is the output for time step t, b_h is a bias term for the hidden state, b_y is a bias term for the output and f_h and f_y are activation functions. W_{hx}, W_{hh} and W_{yh} are weight matrices (input-to-hidden, hidden-to-hidden and hidden-to-output respectively). Note that we consider time in terms of input points: Each new input leads to a new time step.

Long Short-Term Memory networks

While RNN's can work with sequential data, they cannot remember long-term dependencies. The reason for this, is that they suffer from the vanishing gradient problem: During back-propagation, a repeated multiplication of gradients occurs, as we back-propagate through layers in time. This can lead to a term that tends to become zero exponentially fast, making it difficult to maintain information from layers further in the past. In addition, it can lead to a term that tends to become infinite, making the network unstable (exploding gradient problem).

For this reason, Long Short-Term Memory (LSTM) networks have been proposed [56]. In LSTMs, in addition to the hidden state, a cell state is added, which runs through all network layers (through), changed only by some minor linear interactions, and able to easily maintain past information. In order to determine the information that should be added or removed from the cell state, the LSTM uses structures, called gates. Mathematically, we can define the LSTM architecture using the following equations, where x_t is the input vector, c_t is the cell state and h_t is the hidden state for time step t, $W_f, U_f, W_i, U_i, W_o, U_o, W_c$ and U_c are weight matrices and b_f, b_i, b_o and b_c are bias terms:

$$f_t = \sigma(W_f x_t + U_f h_{t-1} + b_f) \tag{2.47}$$

$$i_t = \sigma(W_i x_t + U_i h_{t-1} + b_i) \tag{2.48}$$

$$o_t = \sigma(W_o x_t + U_o h_{t-1} + b_o) \tag{2.49}$$

$$\hat{c}_t = tanh(W_c x_t + U_c h_{t-1} + b_c) \tag{2.50}$$

$$c_t = f_t \odot c_{t-1} + i_t \odot \hat{c}_t \tag{2.51}$$

$$h_t = o_t \odot tanh(c_t) \tag{2.52}$$

A schematic overview of the LSTM unit's architecture can be seen in Figure 2.5. We describe the function of the above equations below:

Updating the cell state c_t: The forget gate f_t is responsible for determining which information should be kept and which should be discarded from the cell state. The current input x_t and hidden state of the previous time step h_{t-1} are multiplied with the appropriate weight matrices, added together with a bias term and passed through a sigmoid function, which limits the result f_t to the range of $(0, 1)$ (Equation 2.47). The calculated value f_t is afterwards multiplied with the cell state, with values close to zero meaning to forget and values close to one meaning to remember all past information in the cell state (Equation 2.51).

While the forget state determines the information to be forgotten, the input gate i_t determines the information to be stored in the cell state. To produce its result, the current input x_t and the hidden state of the previous time step h_{t-1} are passed together with a bias term through a sigmoid function, after being multiplied by the appropriate weight matrices. Again, the sigmoid function limits the input gate's values to the range of $(0, 1)$ (Equation 2.48). The input gate determines which information will be kept from a vector of candidate values, $\hat{c}_t$, which is created by passing the input and hidden state as well as a bias term through a tanh function, after multiplying them with the corresponding weight matrices (Equation 2.50).

The final cell state c_t for time step t, is calculated by adding the information to be remembered from the past cell state, determined by the forget gate, and the new information to be stored, determined by the input gate (Equation 2.51).

Updating the hidden state h_t: The information from the cell state c_t to be contained in the hidden state h_t is determined by the output gate o_t. The output gate's value is calculated using a sigmoid function, which accepts the input x_t and hidden state h_{t-1}, multiplied by the corresponding matrices, together with a bias term, and limits the final value to the range of $(0, 1)$ (Equation 2.49). Its output is then multiplied with the newly updated cell state c_t, after the latter is passed though a tanh function, in order to determine the parts of the cell state that will be passed into the next hidden state (Equation 2.52). The hidden state h_t is also the LSTM model's output.

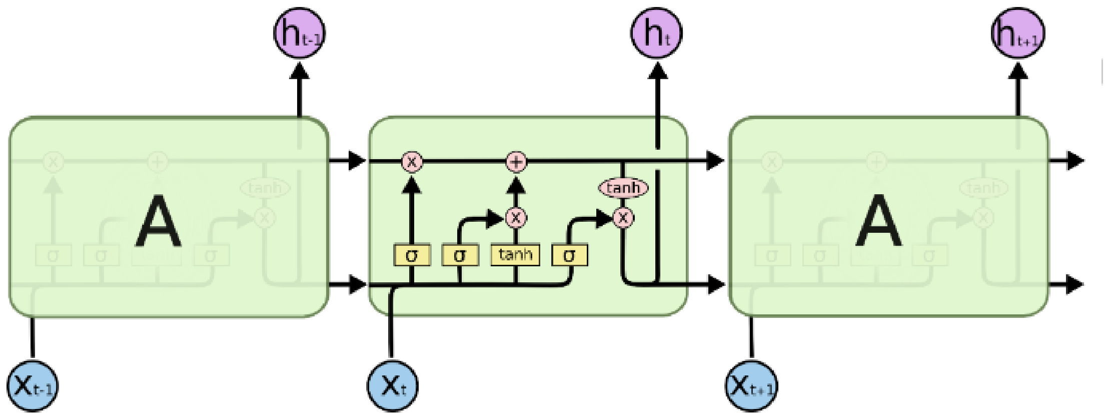


Figure 2.5. *The basic LSTM unit, where X_t is the input vector and h_t is the hidden state for time step t. Image obtained from: [5]*

2.7.4 Attention mechanisms

Despite the additional control LSTM networks provide regarding the abilty to determine the information to be remembered compared to the vanilla RNN, the encoding of the entire sequence into one vector (the last hidden state) still leads to the forgetting of information further in the past, in the case of very long sequences. A mechanism that provides a solution to this problem, called attention, was proposed by Bahdanau et al in [6], initially for the task of neural machine translation, in order to deal with the need for an effective translation of very long sentences. Since then, attention has been used widely, in multiple fields of machine learning, such as computer vision and natural language processing, as well as in multimodal problems that combine the two fields. Specifically in the field of NLP, the attention mechanism gave rise to the creation of the transformer [7], which enabled researchers to achieve state of the art performance in multiple tasks and is the basis of today's large pre-trained language models.

Attention refers to focusing more or less on different parts of an input. For an encoder-decoder architecture, considering an RNN encoder and decoder, where the decoder receives an input context vector c_t for time step t, c_t can be calculated as:

$$c_t = \sum_j a_{tj} h_j \tag{2.53}$$

where h_t is the encoder's hidden state for time step t and a_{tj} are the attention weights, that determine the importance of each of the encoder's outputs through time, for the calculation of the decoder's input context vector. The attention weights are calculated as:

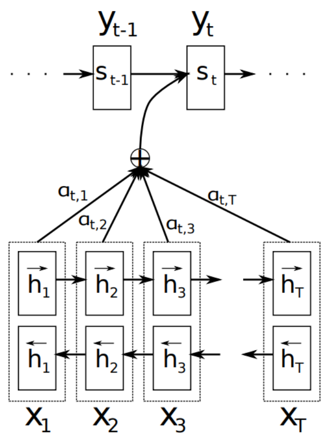
$$a_{tj} = softmax(e_{tj}) = \frac{exp(e_{tj})}{\sum_k exp(e_{tk})} \tag{2.54}$$

$$e_{tj} = f(s_{t-1}, h_j) \tag{2.55}$$

where f is an alignment function, often learned by a trainable model, such as a small FFNN. Some examples of alignment functions are:

- Additive [6]: $f(s_t, h_i) = v_a^\top tanh(W_a[s_t; h_i])$
- Dot-Product [57]: $f(s_t, h_i) = s_t^\top h_i$
- Scaled Dot-Product [7]: $f(s_t, h_i) = \frac{s_t^\top h_i}{\sqrt{n}}$
- General [57]: $f(s_t, h_i) = s_t^\top W_a h_i$

An illustration of the encoder-decoder architecture using attention, as proposed in [6] and described in the above equations, is provided in Figure 2.6.



y_{t-1}
y_t
s_{t-1}
s_t
$a_{t,1}$
$a_{t,2}$
$a_{t,3}$
$a_{t,T}$
$\overrightarrow{h}_1$ $\overrightarrow{h}_2$ $\overrightarrow{h}_3$ $\overrightarrow{h}_T$
$\overleftarrow{h}_1$ $\overleftarrow{h}_2$ $\overleftarrow{h}_3$ $\overleftarrow{h}_T$
x_1 x_2 x_3 x_T

Figure 2.6. *The encoder-decoder architecture using attention, as proposed by Bahdanau et al. in [6]. Image obtained from: [6]*

Since the creation of the first attention mechanism, different attention types have been proposed and employed for different models and problems. One popular type of attention is self-attention or intra-attention, which relates different parts within a sequence, in order to compute its representation, thus allowing the sequence to "attend" to itself. Another popular attention type is multi-head attention, which consists of multiple independent attention outputs, calculated from an attention mechanism that runs multiple times in parallel.

2.7.5 Transformers

The creation of the attention mechanism lead to the proposal of multiple model architectures, the most well known of which is the transformer. Before the transformer, models able to handle sequences effectively were based on complex recurrent and convolutional networks with an encoder-decoder structure and with many of them utilizing attention. In contrast to such approaches, Vaswani et al. in [7] suggested a new type of model, based fully on attention and without any recurrence or convolution mechanism, more parallelizable and faster to train. They called this model the transformer and with it they achieved state of the art results in translation tasks.

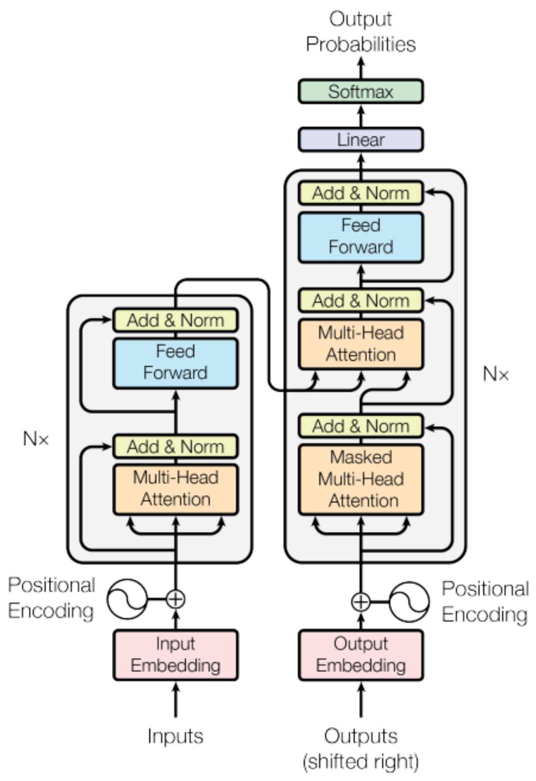


Figure 2.7. *The transformer encoder and decoder modules, as proposed by Vaswani et al. in [7]. Image obtained from: [7]*

Encoder and decoder architecture

The transformer model introduced by Vaswani et al. uses an autoregressive encoder-decoder structure, where the encoder maps an input sequence **x** to a sequential representation **z**, and the decoder uses **z** to generate an output sequence **y**. A graphical representation of the transformer encoder-decoder module is provided in Figure 2.7.

Encoder: The encoder part of the transformer consists of repeating layers of encoder modules. Each such module includes two sub-layers, the first of which is a multi-head self-attention mechanism and the second a fully connected feed-forward neural network. Both sub-layers are followed by layer normalization, while a residual connection from the input to the output of each layer is used.

Decoder: The decoder again consists of repeating decoder modules. Each decoder module has three parts: The first is a multi-head self-attention layer, which uses masking in order to prevent every position from attending to following parts of the sequence. The second is an multi-head attention layer, which attends over the encoder's output. Finally, the third layer is a fully connected feed-forward neural network. As is the case for the encoder, all three layers are followed by a layer normalization.

We note that the fully connected feed-forward network in both the encoder and decoder modules, consists of two linear layers, with a ReLU activation function between them.

Positional encoding

As mentioned earlier, the transformer does not use recurrence or convolution to encode the sequential order of the input sequence. In order to achieve that, is uses positional encodings, which leverage the sine and cosine functions, in order to calculate a representation of dimension d_m, equal to the model's representations, summed together with the latter. For a position pos and dimension i, the sine and cosine functions are defined as:

$$PE_(pos, 2i) = sin(pos/100000^{\frac{2i}{d_m}}) \tag{2.56}$$

$$PE_(pos, 2i+1) = cos(pos/100000^{\frac{2i}{d_m}}) \tag{2.57}$$

Attention

We can define attention as an operation on a set of keys K, queries Q and values V, where the alignment function calculates the attention weights for the keys, based on the queries, using softmax, and the attention weights are then used to attend over the set of values, in order to compute the final output of the attention mechanism. As an alignment function, the authors introduce the scaled dot-product, thus computing the final output as:

$$Attention(Q, K, V) = softmax(\frac{QK^\top}{\sqrt{d_k}})V \tag{2.58}$$

where d_k is the keys' dimension.

In each attention layer, the transformer uses multi-head attention, performing attention in parallel multiple times and concatenating the output representations, before projecting them to the model's embedding dimension d_m. For each of the multiple parallel attention mechanisms, the keys, queries and values of dimension d_m are projected to dimensions d_k, d_k and d_u, using a different learnable projection matrix every time. The multi-head attention's output is thus calculated as:

$$MultiHead(Q, K, V) = Concat(head_1, ..., head_h)W^O \tag{2.59}$$

where

$$head_i = Attention(QW_i^Q, KW_i^K, VW_i^V) \tag{2.60}$$

where W_i^Q, W_i^K, W_i^V and W^O are the projection matrices for the attention head i and h is the number of attention heads.

As analyzed earlier, the authors use attention in three different types of sub-layers:

- In the encoder self-attention sub-layers, the keys, queries and values are set as equal to the output of the previous transformer encoder module.
- In the decoder self-attention sub-layer, the keys, queries and values originate from the previous decoder module's output, but a masking mechanism is used, in order to prevent the queries at each position from interacting with the keys from positions later in the sequence. This is achieved by setting the softmax input values that we do not wish to be attended to equal to $-\infty$.
- In the decoder attention layers that attend to the encoder output, the queries originate from the previous decoder module's output, while the keys and values come from the encoder output.

Since its creation, the transformer model has been used in different variations, either as a whole or with only the encoder part, for multiple applications in the field of computer vision and natural language processing. In addition, it is the basis of today's large pre-trained language models, such as BERT [18] and GPT-3 [24], which are widely used to yield state of the art results in multiple problems.

Chapter 3

Word Representation Methods and Language Models

3.1 Introduction

Natural language Processing (NLP) is the field of Computer Science which aims to help computers process, analyze and generate natural language text or speech, with the goal of achieving a human-like understanding [58]. For this purpose, NLP uses computational logistics, statistics and machine learning.

With the rise of the Internet and social media, bigger and bigger amounts of natural language text are made available everyday, regarding user preferences, sales, politics, community matters, economics etc. This creates the need for effective ways to automatically process text and extract information, thus increasing the demand for efficient NLP methods and maximizing the research and industrial interest in NLP tasks. At the same time, the development and increasing usage of virtual assistants, human-like agents etc. creates a need for computers to be able to understand the semantics of human language, as well as human intents and emotions, and be able to generate human-like speech, in order to effectively communicate with humans in natural language.

Natural language text consists of words, which in turn consist of characters. When using words as input for computations however, this natural representation of words can be very ineffective, firstly because of the different lengths words possess, which makes it hard for machine leaning models to process them, and secondly because of the sparsity of such a representation, with most sequences of characters not forming an existing word [59]. Words are therefore first converted to numerical representations, before being passed as input to machine learning models.

However, words do not occur independently in natural language: They make up sentences, which in turn are combined into paragraphs etc. The syntactic position of a word as well as its previous or following words in a sentence can largely affect its meaning in each specific context and the understanding of the semantic and syntactic relationships between words is essential for the extraction of the intended message. For this reason, representing words is not enough for Natural Language Processing: Language Models (LMs) must be developed to encode and understand word sequences.

In the following sections we first provide an overview of the most important word representation methods throughout the history of Natural Language Processing. We then analyze language models, explaining the basics of more traditional to more modern architectures. Finally, we present the most popular methods used for adapting modern pre-trained language models to different NLP tasks.

3.2 Word representation methods

Converting words to numerical representations is not an easy task, considering the fact that each language can consist of hundreds of thousands of words, with many occurring with different meanings in different contexts, or with different variations: The word representation methods ought to be as computationally efficient as possible, while at the same time being able to handle large numbers of different, even unknown words, and they ought to encode as much useful information as possible, regarding the semantic meaning of words.

In the following subsections we present some of the most common word representation methods, from traditional methods such as one-hot encoding to more modern ones, based on distributional semantics (see Section 3.2.2), such as GloVe and ELMo.

3.2.1 Traditional word representation

Some of the first, more traditional word representation methods include categorical word representations (label encoding, one-hot encoding) and TF-IDF:

Categorical word representation

Categorical word representation methods map each word to a simple representation, such as a numerical value or a vector. Two common methods in this category are Label encoding (Section 3.2.1) and One-hot encoding (Section 3.2.1).

Label encoding In label encoding [60], a numerical value is assigned to every word of the used vocabulary, let this be denoted as V. For example, assuming a vocabulary:

$V = \{"mountain", "sea", "valley"\}$,

then the values 0, 1, 2 could be assigned to the words respectively. The words would thus be represented as:

$w^{mountain} = 0$, $w^{sea} = 1$, $w^{valley} = 2$.

While label encoding could be considered as the most simple and straightforward technique to represent text, is does not generally yield good performance. The reason for that is that, when the data is not naturally ordinal, it introduces an artificial ordering among the words, defining a random "ranking". This can be rather problematic when the word representations are used for neural network architectures, as larger values could affect the model's weights more during training [60], while the numerical distance between word values does not encode useful, semantic or syntactic, information.

One-hot encoding Considering a vocabulary V of size $|V|$, one-hot encoding maps each word w of V to a vector $\boldsymbol{w} = [w_1, w_2, .., w_{|v|}]$ of dimension $|V|$, where each element $w_i, i = 1, ..., n$ has a value of either 0 or 1. Specifically, each word w of the vocabulary is mapped to an index i, and only the i-th element of $\boldsymbol{w}$ is equal to 1, with all other elements being zero [61]. For example, assuming again a vocabulary:

$V = \{"mountain", "sea", "valley"\}$,

its words could be represented as:

$$\boldsymbol{w}^{\ mountain} = \begin{bmatrix}1\\0\\0\end{bmatrix}, \mathrm{w}^{sea} = \begin{bmatrix}0\\1\\0\end{bmatrix}, \mathrm{w}^{valley} = \begin{bmatrix}0\\0\\1\end{bmatrix}$$

While the computation of the one-hot encoding representations is highly efficient, the one-hot encoding method presents some major disadvantages. Firstly, it does not encode word similarity, as all words have the same distance from each other, resulting in representations with poor semantic and syntactic information [59] [60]. In addition, when the vocabulary size is large, which is most often the case, the one hot vectors have a very high dimension. This leads to expensive computations when the one hot vectors are used in neural network architectures, as well as an increased complexity, as a large input size means that a large number of model parameters must be trained, which can lead to overfitting.

TF-IDF

TF-IDF (Term Frequency - Inverse Document Frequency) is a measure of the importance of a term to a document, in a collection of documents. It is a combination of two numerical statistics, TF (term frequency) and IDF (Inverse Document Frequency).

TF (term frequency), credited to Hans Peter Luhn's research on term frequency [62], measures how often the occurrence of a word happens in a document. There are different ways in which term frequency is defined, some of which are:

- The fraction of the total number of occurrences of a word w in a document d to the total number of words in the document (with each occurrence of a word counted separately):

$$tf(w,d) = \frac{f_{w,d}}{\sum_{w' \in d} f_{w,d'}} \tag{3.1}$$

  where $f_{w,d}$ the number of occurrences of a word w in the document d.

- The total number of occurrences of a word w in the document d:

$$tf(w,d) = f_{w,d} \tag{3.2}$$

- The logarithmically scaled frequency of occurrence of a word w in the document d::

$$tf(w,d) = log(1 + f_{w,d}) \tag{3.3}$$

IDF (inverse document frequency), originally proposed by Karen Spärck Jones in [63], is a measure of how common a word is in a collection of documents D. It is defined as the logarithm of the fraction of the total number of documents, divided by the number of documents containing the word w [61]:

$$ifd(w, D) = \frac{|D|}{\sum_{d \in D} i_{w,d}} \tag{3.4}$$

where $i_{w,d} = 1$ if the word w appears in the document d and $i_{w,d} = 0$ if the word w does not appear in the document d.

Combining TF and IDF, we obtain TF-IDF, which, for a word w, a collection of documents D and a document d within that collection is defined as:

$$tf\text{-}idf(w, d, D) = tf(w, d) * idf(w, D) \tag{3.5}$$

The higher the tf-idf value is for a word, the more important this word is in the document collection, while the lower the value is, the word is considered less relevant.

TF-IDF is simple to understand and very efficient to calculate. However, it does not capture the semantic meaning or the syntactic features of words, and it ignores the order of words in the document, resulting often in less meaningful representations [61]. In addition, since a different tf-idf value is calculated for each pair of word and document, it is a very high dimensional representation, resulting in la ack of storage efficiency, and making computations using tf-idf vectors more expensive.

3.2.2 Distributional word representation

Distributional representation methods are derived from the research area of distributional semantics, which is based on the idea that the words that occur in similar contexts (i.e. have similar distributions) have a similar meaning [64]. Based on word co-occurrences [65], they map words to vectors that encode information about the context in which each word tends to appear.

Non-contextual word embeddings

Based on distributional semantics, word embeddings provided an important breakthrough for performance in NLP tasks. Word embeddings are dense, low-dimensional vectors that are learned based on the context in which words occur. They encode word similarity, while maintaining efficiency, due to their lower dimension compared to sparse word representations (such as one-hot encoding or TF-IDF) [61].

Different methods for the calculation of word embeddings have been proposed, employing deep learning models. Bengio et al. [66] created a NNLM model consiting of a one-layer feed-forward neural network, which learned word representations through the training of the language model, using the previous words to calculate the representation of the next.. While other approaches have been proposed as well, the most popular and perhaps the most effective, have been Word2Vec [8] and GloVe [42].

Word2Vec Word2Vec, proposed by Mikolov et al. in [8], is a shallow neural network model with two hidden layers, which is trained on a large corpus and uses the words' context in order to calculate a dense word vector to represent each word. The word representations computed encode linguistic patterns of words [8] and the resulting vector space unfolds the similarity between tokens [64], with words that occur in similar contexts corresponding to vectors that are located closer to one another.

Two similar models can be used to calculate Word2Vec representations, the Continuous Bag of Words (CBOW) model and the Skip-gram model. Both models use a window whose length is predefined, which is slided along the corpus, and are trained with the words inside that window in each step [67].

- **Continuous Bag of Words (CBOW):** In the CBOW model the center word is predicted based on its context, within the window. Much like a feed-forward neural network, the model consists of an input layer, a hidden layer and an output (softmax) layer, with the difference that the hidden layer does not feature a non-linearity, and is simple a projection / averaging layer, which combines the distributed representations of all context words, in order to predict the middle word.
- **Skip-gram:** The Skip-Gram model predicts the context of a given word, based on that word [68]. It consists of an input layer which corresponds to the target word, a hidden layer without a non-linearity and an output layer that uses softmax and corresponds to the predicted context words. It uses back-propagation to adjust its representation based on the correlation between the model's output and the context words [64] [69].

Comparing the two models, Skip-gram is more effective when the amount of training data is smaller and can calculate better representations for rare words, while CBOW is a lot faster, and has better performance for words that are more common [64]. The architecture of Skip-gram and CBOW is depicted in Figure 3.1:

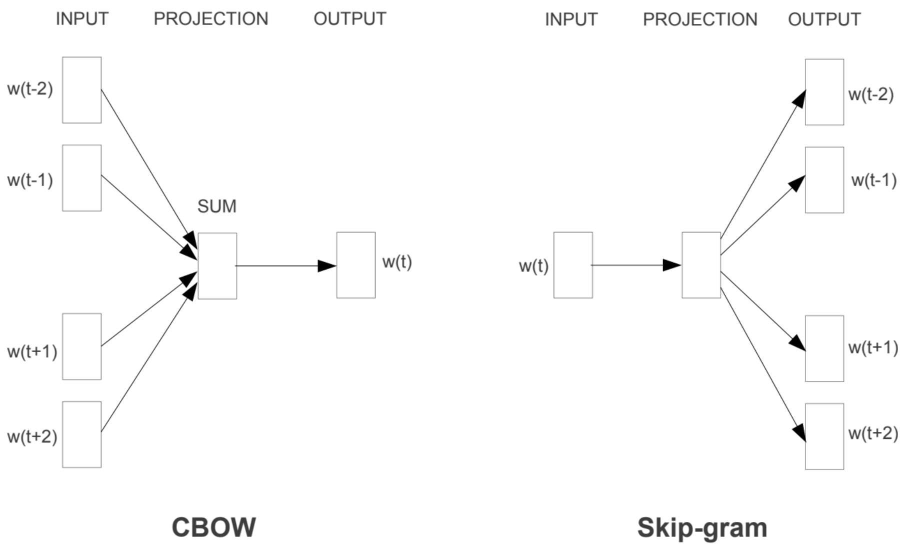


Figure 3.1. *The Skip-gram and CBOW models. Image obtained from: [8]*

GloVe GloVe was proposed by Pennington et al. in [42] and is an unsupervised learning algorithm for obtaining vector representations for words. Training is performed on aggregated global word-word co-occurrence statistics from a corpus and the resulting representations showcase interesting linear substructures of the word vector space. In contrast to Word2Vec which is based on the usage of a sliding window in the corpus, GloVe can capture global statistics in the corpus, by leveraging global information. It is based on the idea that the co-occurrence ratios of words can encode some meaning about these words, and the training objective is the calculation of word vectors, such that their dot product is equal to the logarithm of the words' probability of co-occurrence. Considering the fact that the logarithm of a ratio is equal to the difference of logarithms, the co-occurrence information is ultimately encoded through vector differences in the vector space [70].

Contextual word representations

While representations such as Word2Vec and GloVe use context to calculate word representations, the resulting word vectors are static, always the same for one word, regardless of the context in which the word occurs and the semantic and syntactic features of the word in the specific context. To tackle these challenges, Peters et al. proposed ELMO [71].

ELMO ELMO consists of three modules, a CNN, a bidirectional LSTM and the embedding module. It accepts character based input, with character embeddings passed to the CNN layers, which extract statistical, temporal and spatial information. The CNN representations are then forwarded to a bidirectional two-layer LSTM, which encodes sequential information through time in its hidden states, considering the input sequence from both directions. The final word embeddings are composed by obtaining a linear combination of all hidden states of the bidirectional LSTM, in order to encode the different types of information that the different layers of the LSTM capture [71]. The weight of each hidden state for the final embeddings is task specific. This can improve task performance, as for different tasks the information captured by higher or lower levels may differ in significance, but it also means that a different tuning for the model is needed for each task.

By using the entire input sentence as input [71] instead of a fixed length window, ELMO can successfully leverage the whole context, in order to compute context-dependent representations for both common and very rare words. For this reason, it achieved state of the art performance in most NLP benchmarks at the time of its creation.

3.3 Language models

Language models are models that assign probabilities to sequences of words [23]. They make up the foundation of Natural Language Processing as they offer a method for converting qualitative textual information into machine-understandable quantitative data. Throughout the years, different language models have been proposed. With the advancement of deep learning, traditional language models based on statistics where replaced by language models utilizing neural networks, which finally gave rise to the large pre-trained language models that are used today, such as BERT [18] and GPT-3 [24].

In this section we provide an analysis of the most popular traditional and modern language models, as well as the most common methods used nowadays for the adaptation of language models to specific NLP tasks.

3.3.1 Traditional language models

Traditional language models are count-based: They used a statistics to construct the joint probability distribution of a word sequence. One such model is the n-gram model.

N-gram

A sequence of n words is called an n-gram (a sequence of 2 words is called a 2-gram, a sequence of 3 words is called a 3-gram etc). By extension, in literature, the language model calculating the probability of an n-gram is also called an n-gram [23].

Given a large corpus, one could estimate the probability of a word occurring after a sequence of words by counting: In the sequence "the wolf ate the ____", the probability of one word "sheep" being the missing word, could be calculated as the fraction of the number of times the word sequence "the word ate the sheep" was seen in the corpus to the number of times the word sequence "the wolf ate the" was seen. Formally, we could calculate the probability of a word w_i being the next word in a sequence as:

$$P(w_i|w1...w_{i-1}) = \frac{count(w_1...w_i)}{count(w_1...w_{i-1})} \tag{3.6}$$

While this method could provide good results for a large enough corpus and a common enough word sequence, it is not efficient in cases of rare sequences that do not often occur.

To model the probability of a word sequence more effectively, the n-gram model uses the Markov condition, assuming that the probability of a word only depends on its n-1 previous words. Mathematically, the n-gram language model calculates the probability:

$$P(w_1, w_2, ..., w_k) = p(w_1) \prod_{i=2}^{n} P(w_i|w_{i-n+1}...w_{i-1}) \tag{3.7}$$

which is the probability of the words $w_1, w_2, ..., w_k$ appearing in a sequence, for example in a sentence.

2-gram For example, when $n = 2$, the probability of a word sequence with k words can be calculated as:

$$P(w_1, w_2, ..., w_k) = p(w_1) \prod_{i=2}^{n} P(w_i|w_{i-1}) \tag{3.8}$$

where the assumption is made that each word only depends one the word occurring before it.

3.3.2 Neural language models

Neural language models are language models based on neural networks. They obtain better results than the classical statistical models, as they are able to consider larger context sizes with a linear only increase in their parameters [72], and they learn dense word representations that help overcome the curse of dimensionality [73].

Feed-forward neural network language models

Two of the first researchers to propose the usage of neural networks for language modeling where Xu and Rudnicky. In [74], they proposed a single-layer neural network (with no hidden layers) with an input and output size equal to the vocabulary size, where the i-th input unit is equal to 1 if the input word is w_i. They achieved better results that their baseline N-gram, but their model lacked a good generalization ability and the ability to capture context-dependent features [75].

A more popular work using feed-forward neural networks was published a few years later by Bengio et al. [9]. In their model, words are represented by learnable word vectors that exist in a look-up matrix of dimensions $|V| * m$, where $|V|$ is the vocabulary size and m is the word embedding dimension. The model takes as input the previous words within a specified window from the current word, uses the look-up matrix to obtain the word representations and then passes the concatenated representation through a hidden layer, before the final softmax output layer. The model's architecture is presented in Figure 3.2.

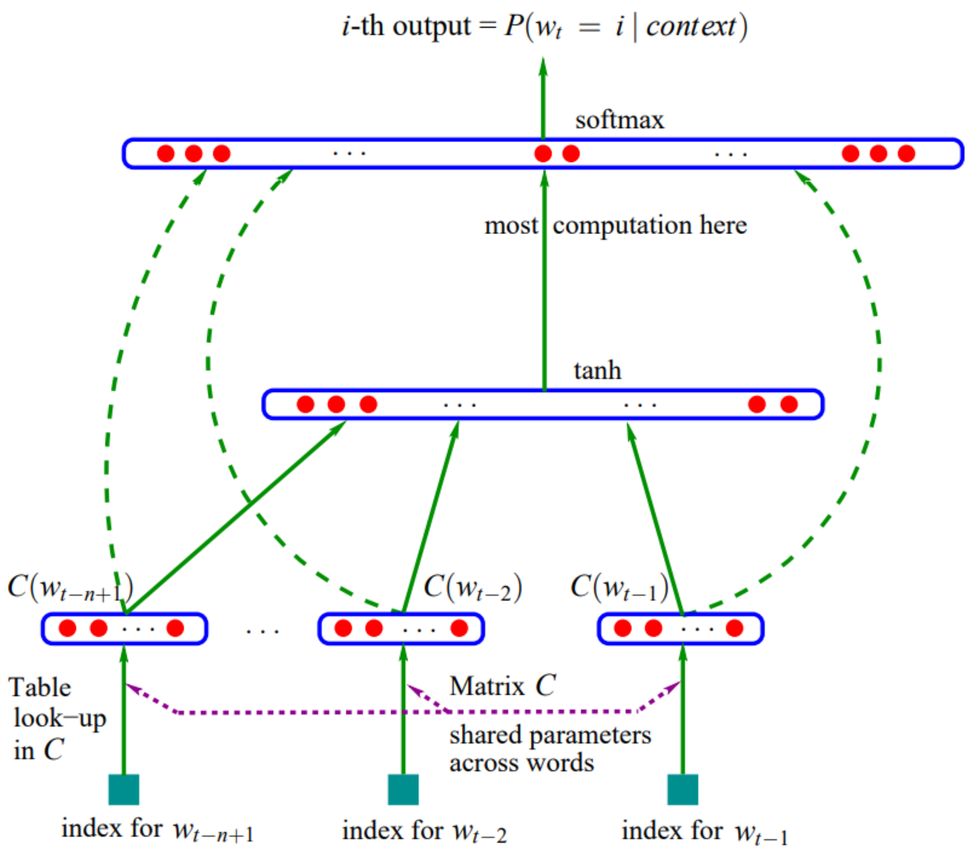


Figure 3.2. *The architecture of the FFNNLM proposed by Bengio et al. in [9]. Image obtained from: [9]*

While the FFNNLM proposed by Bengio et al. in [9] achieved a significant increase in performance, it has significant limitations. It uses a fixed context size, specified in training time and it must learn a large number of parameters, because of its feed-forward architecture.

RNN-LSTM language models

Recurrent neural network language models (RNNLMs) where later proposed as an alternative to feed-forward neural network language models (FFNNLM). In contrast to the latter, RNN language models can model sequential information such as word sequences of arbitrary length and do not require a fixed window size. In addition, because of parameter sharing, they have less parameters that need to be learned through training.

The first RNNLM was proposed by Mikolov et al. in [76] and was extended in [77]. The model consists of an input layer, a hidden layer and an output layer and receives as input at time t the concatenation of the vector representing the current word $w(t)$, and the output from the neurons in the hidden layer at time $t-1$.

Although the results obtained suggest the model's superiority to feed-forward neural network language models and n-gram models, the model's ability to encode long-range sequences is limited, as is a common problem with RNN models. For this reason, LSTM language models were later proposed, which, using the larger control of LSTMs over the storing and forgetting of past information, obtained better performance.

3.3.3 Large-scale pre-trained language models

With the rise of deep learning, enabled by the advancement in computing power, deeper model architectures have shown their superiority in machine learning applications such as computer vision. Such deep models, having a very large number of trainable parameters, require a large amount of data in order to be trained efficiently. However, for most supervised NLP tasks, datasets are rather small, due to the expensive annotation costs, especially for syntax and semantically related tasks [78]. Large-scale unlabeled corpora, however, are simpler to create. They may thus first be utilized to teach the model a sufficient language representation and understanding, which can then be used for other downstream tasks. Because the corpora is unlabeled, supervised learning cannot be used to train the language models. For this reason, new tasks must be devised, that allow the model to learn in an unsupervised or self-supervised setting. Such tasks include Language Modeling, where the model attempts to maximize the maximum likelihood of the probability of tokens appearing in a given context window, Masked Language Modeling, where a number of input tokens is masked an the model must determine the correct token for that position using the context and Next Sentence Prediction, where the model is given two sentences and must determine whether the second follows the first in the original corpora. Using such tasks, the language model can develop a general understanding of natural language and learn to extract meaningful representations from its input. It may then adapted to the supervised setting of each downstream task using a second training stage, this time in the tasks supervised setting.

This idea, together with the creation of the attention mechanism and the transformer, gave rise to the large-scale pre-trained language models, which are first trained using unsupervised or self-supervised objectives on text consisting of billions of words and are then adapted to each specific task using supervised training with the smaller task-specific dataset (See Section 3.4 for an overview of different adaptation methods). Two of the most popular language models that have become the predecessor of multiple other successful language models are the GPT [79] and BERT [18]:

GPT

GPT-1 The first Generative Pre-trained Transformer (GPT-1) was created by OpenAI in 2018 and presented in [79]. In this paper, the authors proposed using unlabeled data to learn a generative language model, which can then be fine-tuned for downstream NLP tasks, such as sentiment analysis or textual entailment. GPT-1 consists of a 12-layer transformer decoder with masked self-attention so as to prevent the access of the language model to context words to the right of the current word. It was trained on an enormous BooksCorpus dataset and was able to learn long range dependencies and gain extensive knowledge from a variety of contiguous text corpora.

According to [79], in the pre-training stage, GPT was trained using gradient descend, with a standard language modeling objective, to maximize the likelihood:

$$L_1(U) = \sum_i log(P(u_i|u_{i-k,\ldots,u_{i-1};\Theta})) \tag{3.9}$$

where k is the size of the context window size, P is the conditional probability, modeled using the transformer decoder with parameters Θ and U is the unlabeled corpus used as the training dataset.

During fine-tuning, to adapt the model to the supervised setting of a downstream task, a labeled dataset C is used, with each sample consisting of input tokens $x^i, ..., x^m$ and a label y. The input tokens are fed into the model's decoder layers, and the output h_l^m of the final layer is passed to a linear classification layer with parameters W_y, which calculates the probability:

$$P(y|x^1, ..., x^m) = softmax(h_l^m W_y) \tag{3.10}$$

The goal is thus to maximize the objective:

$$L_2(C) = \sum_{x,y} logP(y|x^i, ..., x^m) \tag{3.11}$$

The above objective is used together with the language modeling objective L_1, as the authors found that this improves the models generalization ability and accelerates convergence. The final training objective for the finetuning stage is thus formulated as:

$$L_3(C) = L_2(C) + \lambda * L_1(C) \tag{3.12}$$

where λ is set to 0.5.

GPT-2 In 2019, OpenAI proposed GPT-2 [80]. In comparison to GPT-1, GPT-2 consists of more parameters (1.5 billion instead of 117 million), and was trained in a larger and very diverse dataset, surpassing the performance of GPT-1 in language understanding and zero-shot settings.

GPT-3 The third version of the Generative Pre-trained Transformer, GPT-3, [24] was created by OpenAI in 2020. The motivation for the creation of GPT-3 was the construction of a powerful and fast language model that would be able to understand and solve tasks without the need for fine-tuning, after only seeing a few examples. GPT-3 is even larger than its predecessors, consisting of 175 billion parameters and 96 transformer decoder layers (instead of 12 for GPT-1 and 48 for GPT-2). It was trained on a massive dataset collected from the internet and it is known for its ability to create human-like text, as well as solve tasks on which it was not specifically trained, including solving simple arithmetic problems and writing code pieces.

BERT

BERT, meaning Bidirectional Encoder Representations from Transformers, was introduced in 2018 by Google with [18]. In contrast to the unidirectional GPT, which allows every token to attend only to its previous ones, BERT is bidirectional, using both left and right context to obtain word representations. It is based on the transformer as presented by Vaswani et al. in [7] and it is a multi-layer bidirectional transformer encoder. The authors created two versions of BERT, bert-base, with 12 transformer encoder layers, 12 attention heads and a hidden size of dimension equal to 768, and bert-large, with 24 transformer encoder layers, 16 attention heads and a hidden size of dimension equal to 1024. In total, bert-base consists of 110 million and bert-large consists of 340 million trainable parameters.

Input and embedding layer BERT's input format can consist of either one or two text segments (refered to in [7] as sentences), separated by a special ($[SEP]$) token. In addition, a special token ($[CLS]$) is always added to the beginning of the input. The final hidden representation of this token is used as an aggregate for the whole sequence in classification tasks.

BERT uses WordPiece embeddings [81], wherein rare words are split into meaningful subwords, with a total vocabulary of 30.000 tokens. In addition to the word embeddings that correspond to the tokens the input text is mapped to, BERT uses two additional embedding types: To differentiate between the tokens belonging to the first input segment and the tokens belonging to the second input segment, a learned embedding called "segment embedding" is added to each work embedding. Finally, position embeddings are added too, whihch encode the absolute position of each token in the input. Thus, the final input representations that are passed to the transformer encoder layers, are a sum of token embeddings, segment embeddings and position embeddings, as presented in Figure 3.3.

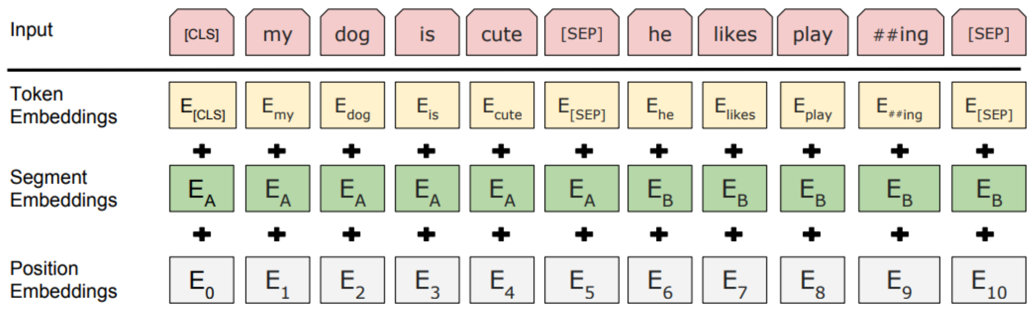

Input
[CLS] my dog is cute [SEP] he likes play ##ing [SEP]
Token Embeddings
$E_{[CLS]}$ E_{my} E_{dog} E_{is} E_{cute} $E_{[SEP]}$ E_{he} E_{likes} E_{play} $E_{\#\#ing}$ $E_{[SEP]}$
Segment Embeddings
E_A E_A E_A E_A E_A E_A E_B E_B E_B E_B E_B
Position Embeddings
E_0 E_1 E_2 E_3 E_4 E_5 E_6 E_7 E_8 E_9 E_{10}

Figure 3.3. *BERT input representations: The input is a sum of token embeddings, segment embeddings and position embeddings. Image obtained from [7].*

Pre-training phase As presented in Figure 3.4, BERT's training consists of two stages, a pre-training and a fine-tuning stage. In the pre-training phase, two unsupervised tasks are used, in order to train the model's parameters using unlabeled data, namely Masked Language Modeling (Masked LM) and Next Sentence Prediction (NSP). The datasets used for pre-training are the BookCorpus, which consists of 800 million words and the English Wikepedia from which only text pieces were extracted and 2.500 million words were used. In the following paragraphs we provide further details about BERTs two pre-training tasks, Masked LM and NSP:

- **Masked LM**: To train the bidirectional model, Devlin et al. could not use bidirectional conditioning without reducing the prediction task to a trivial one (as the word to be predicted would indirectly already be seen by the model). For this reason they proposed a different pre-training task, Masked Language Modeling (Masked LM). During Masked LM, 15% of all tokens in each input sentence are chosen and masked randomly and the model is asked to predict the correct words that were masked. In order to pre-train the model in a way that will keep in as close to the fine-tuning settings as possible, Delvin et al. choose to replace the masked words with a special ($[MASK]$) token 80% of the time, a random token 10% of the time and the token to be masked, unchanged, the rest 10% of the time.

- **Next Sentence Prediction (NSP)**: The motivation for the NSP task, was the importance of the undertanding of the relationship between two sentences by the model, for tasks such as Question Answering (QA) or Natural Language Inference (NLI). In this task, the model is presented with two sentences and is asked to predict whether the second sentence follows the first sentence in the text. For this purpose the authors chose pairs of sentences in which the second sentence was the sentence following the first 50% of the time, and the second sentence was a random dentence from the corpus the rest 50% of the time.

Fine-tuning phase After the model is pre-trained the pre-trained weights can be loaded and the model can be fine-tuned for downstream tasks (such as question answering, sentiment analysis, natural language inference etc.). For this purpose, the text input is transformed, in order to match the input template of BERT. For example, for question answering tasks, the two sentence pair that BERT accepts as input is mapped to a question-reference context pair, while for sequence tagging and classification tasks only the first input sentence of BERT is used and the second sentence is kept empty, while the final hidden representation of the ($[CLS]$) token is passed to a classification head.

BERT's fine-tuning can be performed effectively using much smaller datasets than those used in the pre-training stage and it demands less computation resources that BERT's pretraining, rendering it a process easily repeatable for different tasks and in different settings.

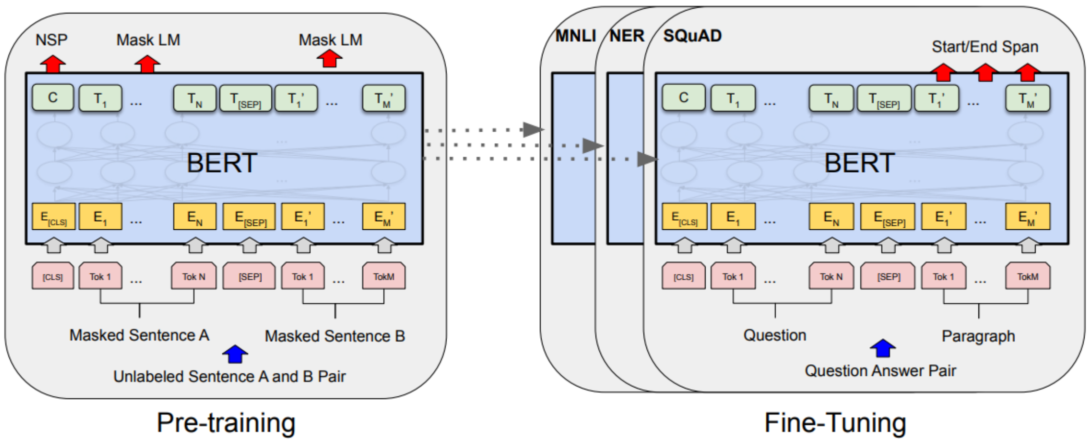


Figure 3.4. *The pre-training and fine-tuning procedures of BERT. Image obtained from [7].*

The importance of BERT At the time of its release BERT achived state of the art results for multiple NLP tasks and it has since been used by many researches for a large variety of tasks and settings. Based on BERT and with modifications in its architecture or pre-training phase, many high performing pre-trained language models have been created since, such as RoBERTa [19], ELECTRA [28] and XLNet [29].

3.4 Adapting pre-trained language models

As analyzed so far in this chapter, pre-trained language models are trained on a vast amount of unlabeled data, with unsupervised or self-supervised objectives, using high computational resources and with a training time lasting for multiple days. As the datasets for most downstream NLP tasks are not large enough to train the language models such as BERT or GPT from start and most research labs do not possess the hardware resources needed for such training, these models are not trained separately for each task. Instead, the language understanding they have obtained through the pre-training phase is leveraged and the pre-trained weights are loaded into the models, which are then adapted to particular tasks using the smaller downstream task datasets.

Traditionally, to adapt the pre-trained language models to downstream tasks, fine-tuning has been used. However, fine-tuning is not always optimal: It can cause the model to overfit and by modifying the pre-trained models parameters it can lead the model to forget important language relationships and representations learned though the pre-training phase. In addition, the increasing size of pre-trained models today means that even fine-tuning these models can be relatively expensive and not always the most computationally efficient alternative. For this reasons, in the recent years, a variety of different alternatives to fine-tuning have been proposed to adapt pre-trained language models, with the purpose of tackling the problems that occur when fine-tuning is used.

In the following sections we analyze some of the most popular methods used for adapting pre-trained language models to different tasks.

3.4.1 Fine-tuning

Fine-tuning is the most popular method for adapting pre-trained language models to downstream tasks, and the method proposed during the creation of language models such as GPT [79] and BERT [18]. During fine-tuning, the pretrained model weights are loaded and the model is trained for some additional epochs, using a dataset of the desired task. Usually, all of the language model's parameters are re-trained and optimized during training, in order to minimize the task loss.

Fine-tuning is very straightforward to perform and it has been used extensively in the literature so far, yielding good results. Some of the disadvantages it poses and have already been mentioned are, that fine-tuned models are prone to overfit the data, as well as the fact that it is a demanding procedure, both in computational and storage resources: Due to the very large size of modern pre-trained language models, tuning all of their parameters requires a lot of memory during training and can be time consuming, while storing the fine-tuned language models can also be expensive.

3.4.2 Model parameter ablation

In order to reduce the memory footprint of adapting pre-trained language models, some researchers have proposed ablating (removing) model parameters [25] [26]: Instead of modifying model parameters, this line of work trains binary masks that determine which model weights will be ablated [21] for each task.

Using this method, researchers have observed results comparable to fine-tuning, although with a tendency to be a little lower in many tasks. The large advantage of this method is memory efficiency: When solving multiple tasks, the whole fine-tuned model needs to be stored for each task. On the other hand, with masking, only one copy of the whole pre-trained model and a set of binary masks for each task need to be stored, thus largely decreasing the necessary storage space [25].

3.4.3 Adapter-tuning

Instead of removing model parameters, another line of work focuses on inserting a small number of new parameters. In NLP small modules based on residual networks are inserted inside the transformer layers of pre-trained modules. These models, called residual adapters, were originally introduced by Rebuffi et al. [27] and are trained using the downstream task's dataset, while the initial pre-trained model's parameters are kept frozen. The processes of inserting and training residual adapters instead of the pre-trained models parameters is often referred to as adapter-tuning [21].

The exact placement and architecture of the adapter modules inside the transformer layer can differ in the literature and may affect task performance [82]. For example, Houlsby et al. [10] suggested using an adapter module consisting of two feed forward layers separated by a nonlinearity, with an internal skip-connection and a bottleneck (a projection of the initial features to a smaller dimension and then back to the original dimension). They proposed inserting this module twice in each transformer layer, once after the multihead attention sub-layer and one after the feed-forward sub-layer of the transformer block. Their proposed module architecture and placement is presented in Figure 3.5. While an adapter module architecture similar to the one described is common in the literature, the placement of the module often differs from [10]. For example, Pfeiffer et al. [83] only place an adapter module after the feed-forward sub-layer of the transformer blocks.

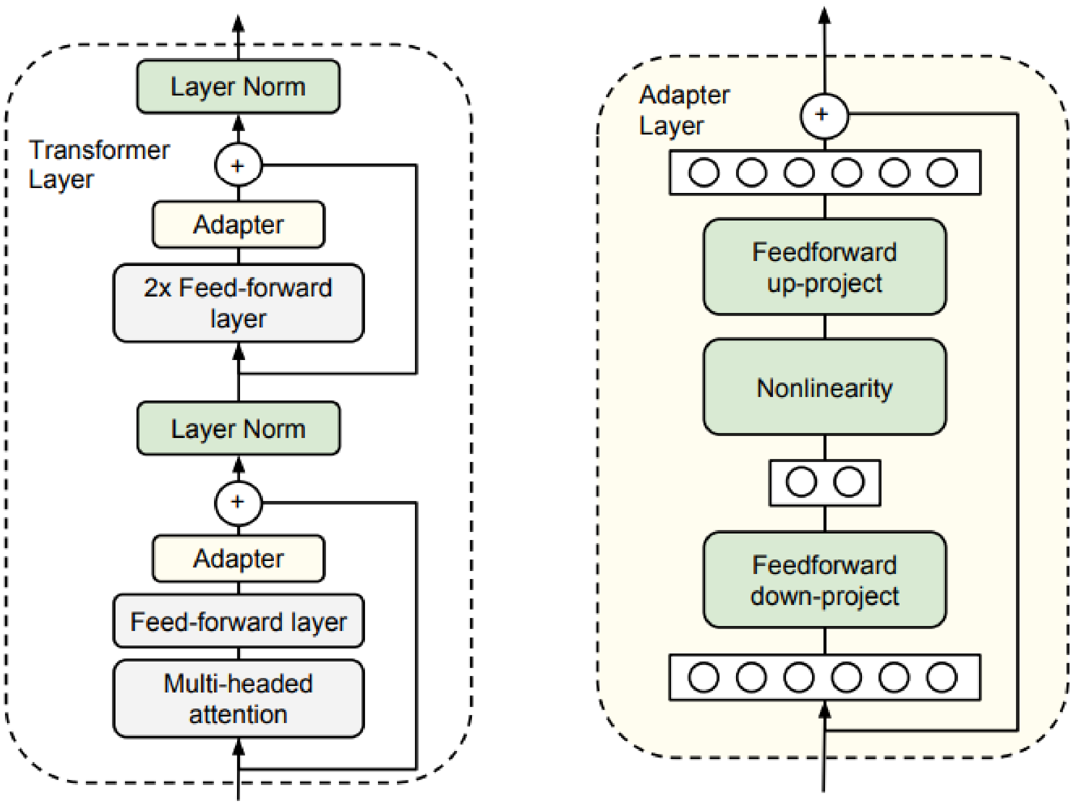


Figure 3.5. *Right: Architecture of the adapter module. Left: The integration of the adapter module with the transformer. Image obtained from: [10].*

Overall, adapter-tuning has achieved performance comparable to fine-tuning, while significantly reducing trainable model parameters and as a result storage space for language models trained on downstream tasks. In addition, adapters allow an easy and effective information sharing between tasks. In contrast to multitask learning which can often lead to catastrophic forgetting (the forgetting of information learned at the early stages of training) or catastrophic inference (the decrease of task performance when adding new tasks), adapters can be learned separately for different tasks and then combined using attention [83]. The separation of the task-specific information learning and the information combination, alleviates the aforementioned problems of multitask learning [82], improving task performance.

3.4.4 Prompt-based learning

All methods described so far perform changes inside the pre-trained language model, by tuning all of its parameters, removing some of them or inserting a small amount of new parameters.

However, in the latest years an alternative approach has been proposed, which instead changes the pre-trained language model's input. This method keeps the whole pre-trained model unchanged and inserts a small number of parameters, called a prompt, either in the level of the text input or directly in the embedding level of the model. With the size of modern pre-trained language models constantly increasing, prompt-based learning, as this approach is often referred to, is mostly researched because of its inexpensiveness in terms of training and storage resources, compared to the traditional fine-tuning. An extensive analysis of prompt-based learning is provided in the following chapter (Chapter 4).

Chapter 4

Prompt-based Learning

4.1 Introduction

For years the paradigm for adapting pre-trained language models to downstream tasks has been fine-tuning the pre-trained model on the downstream data [84] [30]. However, fine-tuning requires the training of all of the millions or even billions of parameters that pre-trained language models (GPT-3 [24], T5 [85] etc.) consist of nowadays. This requires a vast amount of resources for training, as well as storage of the task-specific fine-tuned models [21]. In addition, to efficiently fine-tune the model, achieving generalizability, a big amount of downstream data is needed in order to prevent overfitting [22], which is not always available for the downstream task.

To tackle these issues, prompt-based learning has been proposed as an alternative. Instead of modyfing the pre-trained model's parameters, prompt-based learning adds a new, small set of parameters, called a prompt, to the model's input [30] [31]. These are added either to the text input, as a series of prompt tokens, or directly in the embedding space, as a series of prompt embeddings. The pre-trained model's parameters are usually kept frozen, which rapidly decreases the number of trainable parameters, making prompt-based learning a lightweight alternative.

4.2 Definition

Traditionally, in classification tasks, the pre-trained model calculates the probability:

$$P_\theta(Y|X)$$

where Y is a sequence of tokens representing the class label, X is the text input and ϑ represents the parameters (weights) of the model, which are tuned using fine-tuning.

Using prompts, a set of parameters θ_p, is added, while the initial model parameters ϑ usually remain frozen, and the model must now calculate the probability:

$$P_{\theta;\theta_p}(Y|X;P)$$

where P are the prompt tokens, parameterized by θ_p, which are passed into the model as additional input [31].

4.3 Prompt engineering

4.3.1 Determining prompts

As mentioned, prompts are comprised by a set of prompt tokens, parameterized by θ_p. These tokens can be determined either manually [86] [32], using predefined prompts, usually intuitive to the way humans understand the task, or learned automatically using a variety of methods [30]. Examples of such methods are:

- Paraphrasing-based methods: These methods take a predefined prompt and paraphrase it into multiple, semantically similar prompts, out of which the one achieving the best performance is selected
- Mining-based methods: Such methods search a large text corpus (e.g. Wikipedia) for strings containing the training inputs x and the training outputs y, and find either the middle words or dependency paths between the inputs and outputs [30] [33].

4.3.2 Prompt types

Discrete (hard) prompts

The prompts first proposed in the bibliography and very often used are discrete prompts (also called hard prompts), i.e. prompts consisting of tokens that are directly mapped to an existing word in the model's vocabulary [32], [33], [34]. The prompt parameters θ_p in this case are a subset of the pre-trained language model's parameters θ, and specifically of the model's word embedding parameters θ_{emb} ($\theta_p \subseteq \theta_{emb} \subseteq \theta$). With discrete prompts the input format usually has the form:

$$P_1 \ P_2 \ ... \ P_i \ [X] \ P_{i+1} \ ... \ P_j \ [Z] \ P_{j+1} \ ... \ P_n,$$

where $P_i, i = 1, 2, .., n$ are the prompt tokens each corresponding to a word in the model's vocabulary, $[X]$ are the input tokens (derived from the input text) and $[Z]$ is one ore more missing tokens that the model must fill (mask-tokens), with the task being formulated as a language modeling task.

For example, for an emotion recognition task, the template could be: "In the sentence [X] the speaker is feeling [Z]", where [X] would be the words of the speaker whose emotion is to be recognized and [Z] would be filled with the model's prediction.

Soft prompts

Discrete prompts offer explainability, as humans can easily understand their meaning. However, the process of choosing the optimal template for each task can be difficult and sub-optimal. Motivated by the challenges hard prompts pose as well as the fact that, in contrast to humans, pre-trained models can understand continuous parameters that do not directly correspond to natural language, researchers have developed a second type of prompts, called soft prompts [31] [35] [31]. These consist of continuous parameters, which can be optimized through back-propagation directly in the embedding space, making them easily trainable and fully adaptable to each task and language model. They could be thought of as "soft" words, with the prompt-embeddings lying between the vocabulary's actual word embeddings ($\theta_p \supseteq \theta$).

The initialization of soft prompts can be performed in different ways: The most straightforward, is a random initialization of their parameters. However, over the years, other methods have been proposed as well, such as the initialization with embeddings of random words from the language model's vocabulary, or with the embeddings of the labels corresponding to different classes, in classification tasks [31].

Mixture of hard and soft prompts

Instead of choosing between discrete and soft prompts, some researchers [35] [87] have suggested merging the aforementioned two types, in order to improve performance, using some tunable soft prompts, whose parameters can be optimized to maximize performance, but adding also some hard prompt tokens, corresponding directly to words of the model's vocabulary and determined specifically for each task.

4.4 Prompt-based learning and finetuning

Prompt-based learning is often used as a lightweight alternative to fine-tuning, keeping the pre-trained language model's parameters frozen. However, this is not necessary. Prompts can also be used additionally to finetuning, with the purpose of simply improving performance. In the following paragraphs we present the alternative usages of prompt-based learning:

Alternative to finetuning

As mentioned, in prompt-based learning, the pre-trained model parameters, θ, are usually kept frozen, especially in the case of very large pre-trained language models (such as GPT-3 [24] and T5 [85]), and the prompt parameters θ_p are optimized, either through back-propagation when working with soft-prompts, or with other manual or automatic methods, when working with discrete prompts. This reduces the number of parameters to be trained rapidly in comparison to finetuning, making prompt-based learning a much more efficient alternative, in terms of computational and storage resources. In addition, since the parameters learned during the pre-training stage of the model are not altered, the language understanding that the model has obtained from the pre-training stage remains unchanged. This can be beneficial, as finetuning the pre-trained model's parameters using an insufficient amount of data could damage the ability of the model to understand language in a broader way, resulting in a worse generalization ability.

Aditionally to finetuning

While the pre-trained lagnuage model's parameters θ are often kept frozen, especially when extremely large pre-trained language models are used, this is not always the case: Some researchers use prompts as additional information that boosts performance, but finetune all or some of the model's parameters at the same time [34] [35] [88], especially when using smaller models (for example bert-base), where finetuning is not as demanding in terms of resources and space. The final choice, which parameters to freeze or not, ultimately depends on the task, the available data and resources, etc.

4.5 Previous work using soft prompts

In the field of prompt-based learning using soft prompts, many different variations have been proposed in the latest years. Lester et al. [31], use a sequence of prompts that lie directly in the embedding space of the model and are concatenated with the word embeddings that are produced from the text input that is given to the model. From their results they observe that their method can be very effective for billion parameter models, but lacks in performance in comparison to finetuning, when the language model is smaller (i.e. 100 million parameters). For this reason, transfer learning was later proposed by Vu et al. [36]: Prompts where first trained on different tasks similar to the downstream task or tasks that involve high level reasoning about semantic relationships among sentences. The pre-trained prompts were then used to initialize the prompts for the target task. As the authors observe, this method can allow prompt-based learning to be effective even for smaller scale models.

Other researchers have experimented with a modified version of prompt-based learning, using prompts not just at the input layer but also deeper in the model. For example, Li and Liang [21], used prefix activations prepended to each layer in the encoder of the model, including the input layer, working on language generation tasks. Liu et al. [37] also used prompts in different layers of the pre-trained model, prepending them as prefixes, and extended Li and Liangs method to natural language understanding (NLU) tasks. They also observed that, in full-data settings, both a language modeling head and a randomly initialized classification head can be used to predict the final classification labels, in the case of prompt-based learning.

Finally, as mentioned earlier, some researchers have proposed the combined usage of hard and soft prompts, with the aim of maximizing performance. For example, Liu et al. [35], proposed "P-tuning", which uses continuous prompts that are the output of a trainable prompt encoder. The prompt encoder is used to model the dependency between the prompt embeddings and to avoid local minima and it consists of a bidirectional LSTM, followed by a RELU activated MLP. Within the prompt template, Liu et al. also added some task-related anchor tokens (for example the token "?" for the RTE task [38]).

Chapter 5

Emotion Recognition in Conversation

5.1 Introduction

Emotion recognition is a task that has gained increasing popularity in the recent past years due to its applications in fields such health-care, the development of empathetic agents, as well as opinion mining based on the big amount of conversations between users (in text or video) that are nowadays available in the social media. Many architectures utilizing different types of deep learning models have been proposed, with researchers experimenting with different datasets, information sources (speaker-identity, external knowledge, topic etc). and constantly improving the performance achieved on the task. In the current chapter we provide an overview of the task of Emotion Recognition in Conversation: its definition, the most important information modelled for the task, previous work, and the datasets used.

5.2 Definition

Given a sequence of utterances $u_1, u_2, \ldots, u_N$, and the speaker p_i of each utterance $u_i, i = 1, 2..., N$, Emotion Recognition in Conversation (ERC) is the task of identifying the emotion label of each utterance from a set of predefined labels (such as happy, excited, angry, frustrated, sad, neutral etc.) [39][40].

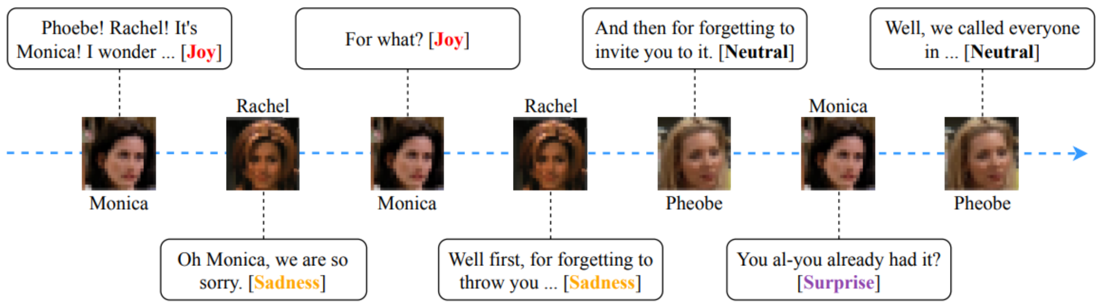

Phoebe! Rachel! It's Monica! I wonder ... [Joy]
For what? [Joy]
And then for forgetting to invite you to it. [Neutral]
Well, we called everyone in ... [Neutral]
Rachel
Rachel
Monica
Monica
Monica
Pheobe
Pheobe
Oh Monica, we are so sorry. [Sadness]
Well first, for forgetting to throw you ... [Sadness]
You al-you already had it? [Surprise]

Figure 5.1. *An example of a conversation for emotion recognition. Image obtained from: [11] .*

5.3 An overview of important variables in identifying emotion in conversation

When attempting to identify the emotional state of a speaker uttering a phrase in a conversation, various sources of information can be taken into account:

5.3.1 Context

In contrast to traditional emotion recognition, when transferring to a conversational setting, the utterance whose emotion one attempts to identify is often very short and does not include a lot of information when isolated. Therefore, in order to identify the emotion it represents, the utterance's context must be taken into account [11] [89]. For example, the utterance "Yeah", could show any emotion, from happiness to anger or sadness, depending on the dialogue that has occurred before [17], and so the context is an important source of information.

When looking at context, there are two options: Either to consider both future and past utterances or to only take into account past utterances. The first option can sometimes improve performance [13] by allowing a better understanding of the whole conversation and of the speaker's general emotional state or by helping the model fill in information that is part of the speaker's backround knowledge and explicitly appears later in the conversation [17]. However, it is not always a realistic option. For example, while in the task of opinion mining all utterances are available, when dealing with real-time emotion recognition, as is the case with an empathetic agent, it is not possible to have knowledge of future utterances.

Another question that arises when handling context is, how far in the past (or future) one should look. As a conversation progresses and its sub-topic gradually changes, the emotional state of the speaker can also change, and it is usually a response to the last few utterances exchanged. Therefore, it is natural to expect that the utterances closer to the current utterance are more important for identifying its emotional label. However, utterances further in the past can also contribute to the current utterance's emotion [39] and should therefore be considered. One possible cause for this could be that people tend to keep their current emotional state, if not affected strongly by an external factor [17] [90], and so clues from utterances further away could carry some helpful information.

5.3.2 Inter- and intra-speaker dependencies

During a dialogue, there are two dependencies one may model [17]. The first is intra-speaker dependency (or self-dependency [17]) and it refers to the effect speakers have on their own emotion, and specifically to their tendency to remain in one emotional state and resist to changes from external factors. The second dependency, inter-speaker dependency, refers to the changes caused to a speaker's emotion, due to their interaction with other speakers. Both of these dependencies can largely affect a speaker's emotion and should thus be modeled appropriately.

5.3.3 Emotional consistency

Due to the fact that speakers tend to maintain their emotional state, one can observe that neighboring utterances of a speaker often express the same or a similar emotion [17] [90]. This has sometimes been explicitly modeled. For example Wang, Y. et al. [90] used a CRF (Conditional Random Field) as the top layer of their model, in order to take into account emotional consistency, and treated the ERC task as a sequence tagging problem. In general, even when not modeling emotional consistency explicitly, it is always taken into account through the context, which is why models often have a higher performance in identifying the emotion of a utterance in cases where the same speaker's previous utterance expresses the same emotion [39], rather than in cases of abrupt emotional changes.

5.3.4 Speaker personality

It is natural that different speakers in a conversation will have different personalities, causing them to react differently in the same situation. For example, the same joke about someone could be found funny by the person towards whom it was intended, but may cause a different recipient to take offence. Another example is that some people tend to avoid conflict by responding calmly in challenging situations, while others may be more irritable, stressful or impatient [91]. Acquiring knowledge about different speaker's personalities could therefore be beneficial in understanding their emotional states.

5.3.5 Topic

While the topic of the conversation does not directly influence the speakers' emotions, it can be indicative to some point: Topics tend to carry certain language patterns [44] and can thus affect both the utterance's meaning and the particular emotion conveyed by specific expressions. Topic modeling has therefore sometimes been used as an additional source of information to enrich the model's representations [16].

5.4 Previous work

5.4.1 Sequence-based methods

In the last few years the task of emotion recognition has received an increasing amount of attention and multiple methods have been developed to solve this task. One line of work, closer to ours, uses sequence based methods, treating the ERC data as a sequence through time.

RNN- and LSTM-based methods

The first methods in this field used LSTMs and GRUs to encode long term information. For example, Majumder N. et al., with DialogueRNN [39], used a GRU to encode global context information, while the hidden states of this GRU were passed through an attention layer to obtain context representations that, together with the utterances of a specific speaker were fed into a speaker-specific GRU to obtain speaker specific representations of utterances. Finally, the authors used a third GRU to take into account the speakers' previous emotional states, before finally classifying the emotion of the current user.

Instead of modeling different speakers by different GRUs and integrating the global context and intra-speaker information with the global GRU like DialogueRNN, Zhang H. and Chai Y., working with dyadic conversations, proposed a more direct interaction between speakers with COIN [41]: They used a bi-GRU that encodes utterances per speaker sequentially, but regulated its output in each direction (forward and backward GRU) with the hidden state of the bi-GRU of the other speaker. For encoding global context they used a sequence of bi-GRU layers combined with an attention layer, in order to capture the most important inter-dependencies.

Transformer-based methods

While RNN-based methods are still proposed, is is often recognized that their ability to maintain important information for very long sequences is limited: As more utterances are added, long-term dependencies tend to be forgotten [17]. This can be problematic in the case of emotion recognition in conversations, as further away context can often provide some amount (although not as big as more recent context) of information about a speaker's emotion [39]. In addition RNN-based methods rely on external word embedding extractors (e.g. Glove [42], word2vec [8]) that do not take context into account, thus performing word embedding extraction and sequence modeling independently [13], which may not always be optimal. For these reasons Transformers have been introduced to the ERC task. Motivated by the need to better maintain long-term dependencies, by the ability of transformers to handle embedding extraction and sequence modeling at once and by the large improvements transformer-based pre-trained language models bring to many downstream tasks in NLP, there have been, in the last years, a series of works attempting to leverage large pre-trained transformer-based language models to effectively classify emotion in conversations.

In these models, utterances are usually given as input in sequential order . To adapt the model to the dialogue setting, extra layers or/and embeddings are then used or existing layers are manipulated, to include information important to the emotion recognition task. Another popular use of pre-trained transformer-based models is to extract embeddings that encode context to some degree as utterance representations, which can then be used in a second, main module, either sequential or graph based.

For example, with Hitrans, Li, J. et al. [11] used a pre-trained transformer encoder to obtain representations that include local context and added a second higher-level transformer to include global context by taking as input the local representations. They also used multitask learning to include speaker-specific information, by adding a second task of Pairwise Utterance Speaker Verification (the objective is to classify whether two utterances belong to the same speaker), using a biaffine classifier with the high-level transformer's representations as input.

Shen W.et al., with DialogXL [43], attempted to maintain a simpler architecture by leveraging a transformer-based pre-trained model and encoding all useful information through multi-headed self-attention in each layer. They applied different masking mechanisms and used global self-attention for long range dependencies, local self-attention for capturing context closer to the current utterance, as this is usually more important in determining the current emotion, and finally a speaker self-attention and a listener self-attention to capture intra- and inter-speaker dependencies.

Using again a transformer-based pre-trained language model, Kim T. and Vossen P. [13] created an even simpler model, EMOBERTa, by prepending speaker names to the utterances and giving them as input to a BERT-like model, adding only a randomly initialized linear layer as the output layer of their model.

Finally, Zhu, L. et al. [16], with TODKAT, utilized a transformer-based pre-trained language model which they fine-tuned, as their basis, but integrated topic information to their architecture with the usage of a topic modeling layer. They also proposed the incorporation of commonsense knowledge by obtaining the most similar to each utterance representations from a large knowledge base and applying attention to integrate the most useful knowledge.

5.4.2 Graph-based methods

Graph based methods model utterances and their relations using nodes and edges of graphs and aggregate information from neighboring nodes to model context and speaker dependencies.

One of the first popular models in this field is DialogueGCN [17]. Its authors attempted to model speaker-level context using neighborhood based convolution, with a Graph Convolution Network (GCN). To model the sequential nature of conversations they firstly used GRUs, obtaining a context-aware sequential representation for each utterance. Using these representations, they constructed a graph, where each utterance is a node and edges connect utterances within a window of specified length from the utterance of each node. Edge weights were calculated using attention and different relationships for each edge were defined, based on the speakers of the utterances (nodes) that the edge connects and the temporal relationship between those utterances, thus modeling both speaker and sequential dependencies between utterances. Finally, a two level, relation-aware convolution was performed to obtain the final representations, enriched with both sequential and speaker information, which, together with the representations obtained by the GRUs, were used for emotion classification.

Inspired by DialogueGCN, Hu J. et al. [92], adapted a similar model to a multimodal setting, using again utterances as nodes of the graph, but this time representing each utterance with three nodes, one for each modality (text, audio, visual) and creating edges between nodes of the same utterance but of different modality, and edges between utterances of the same modality. Instead of using relation types to model speaker dependencies, they encoded speaker information by concatenating speaker embeddings with sequential representations obtained by LSTMs and using the concatenated representations as the representations of the utterances in the graph.

Following a different approach from the latter two models, another team of researchers [40] introduced more refined edges to their graph, using four types of relations: influences of intent from the speaker's own future utterances, influences of action from the same and other speakers' past utterances and self-influences within each utterance. To obtain representations for the different types of relations they used external commonsense knowledge, as they observed that training embeddings did not perform as well.

5.5 Datasets

Throughout our research we perform our experiements on two datasets for Emotion Recognition in Conversation, MELD and IEMOCAP. In the following sub-sections we perform a short analysis on these datasets.

5.5.1 MELD

General information

Multimodal Emotion-Lines Dataset (MELD) [12] is a multimodal dataset for Emotion Recognition in Conversation, containing multi-party conversations in text, audio and visual modalities. It is an extension of the EmotionLines dataset, which contains dialogues from the TV series Friends. MELD was annotated using all three modalities and its utterances where classified as one of the seven emotions: anger, disgust, fear, joy, neutral, sadness. It is separated in train, development and test splits.

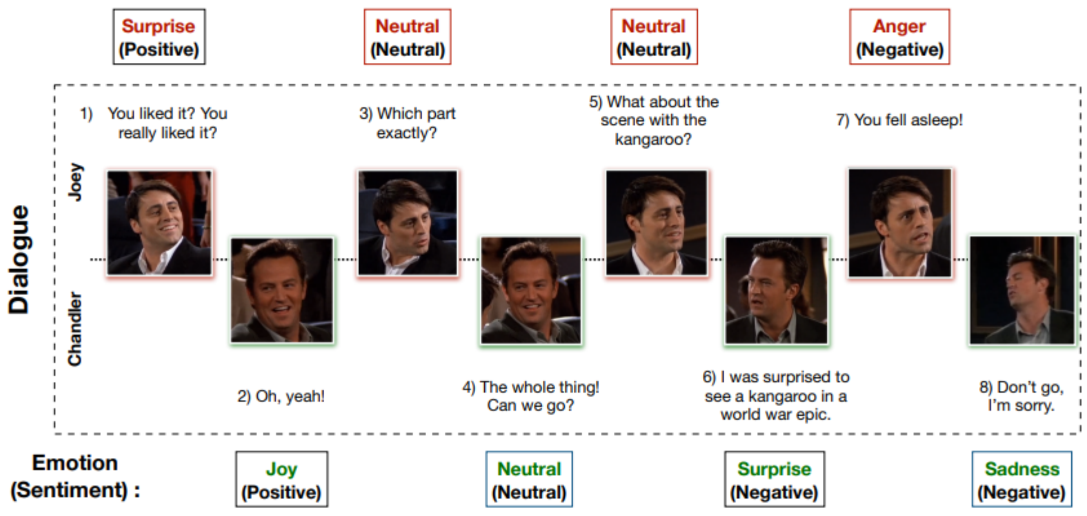


Figure 5.2. *An example dialogue in MELD, between two speakers. Source: [12]*

General dataset statistics

MELD is organized in dialogues, each of which contains a number of utterances. In total, there are more than 13000 utterances in MELD, making it a quite large dataset compared to others in the field of ERC. The following table shows some general statistics regarding dialogues and utterances for the training, development and test split in MELD, as provided by [12] or calculated by us:

Table 5.1. *General statistics for MELD.*

	train	development	test
# dialogues	1039	114	280
# utterances	9989	1109	2610
Average/Max utterance length	8.0/69	7.9/37	8.2/45
# speakers	260	47	100
Average # of utterances per dialogue	9.6	9.7	9.3
Average # of emotions per dialogue	3.3	3.3	3.2
Average/Max # of speakers per dialogue	2.7/9	3.0/8	2.6/8

From the above table we an see that MELD follows a 73%-8%-19% analogy for the train-development-test split. Its utterances are rather short and there are often more that two speakers in each conversation.

Emotion class distribution statisitcs

The following table depicts the Emotion class distribution for MELD, for each of the train, development and test splits:

Table 5.2. *Class distribution (%) for MELD.*

	train	development	test
anger	11.10	13.80	13.22
disgust	2.71	1.98	2.61
fear	2.68	3.61	1.92
joy	17.45	14.70	15.40
neutral	47.15	42.38	48.12
sadness	6.84	10.00	7.97
surprise	12.06	13.53	10.77

As the above table shows, MELD is very imbalanced, with the majority of utterances belonging to the neutral emotion class, and the classes of disgust and fear having the smallest number of samples. This imbalance is natural to the emotions people usually feel (it is the most common for a person to be in a neutral state when having a conversation) but it can make the training of our model more difficult, making it harder for it to distinguish the minority classes and biasing it towards the majority class.

Comparing the distribution of emotions between the train, development and test split, we can see that it is very similar for the three splits.

Speaker distribution and statistics

As seen in Table 5.1, there are hundreds of speakers in MELD. However, only six of them have more than 60 utterances each. These are the six main characters of the Friends TV show, namely Chandler, Joey, Monica, Phoebe, Rachel and Ross. Considering these six speakers as the main speakers of our dataset, we can label all others as "Other". The following table depicts the number of utterances for each of the six main speakers and for all the other speakers grouped together:

Table 5.3. *# Utterances per speaker in MELD.*

	train	development	test
Chandler	1283	101	379
Joey	1510	149	411
Monica	1299	137	346
Phoebe	1321	185	291
Rachel	1435	164	356
Ross	1458	217	373
Other	1683	156	454

From the above table we can see that the six main speakers have a similar number of utterances with each other, which is around the same as the total of utterances from all other speakers. This means, that, regarding speaker distribution, considering seven groups of speakers (the six main speakers and the all others as a seventh group), MELD is almost balanced.

In the following tables, we present the distribution of emotion for each of the six main speakers and all other speakers grouped together:

Table 5.4. *Emotion distribution per speaker for MELD dataset, for train data.*

Speaker	Joy	Surprise	Fear	Anger	Disgust	Sadness	Neutral
Chandler	0.156	0.113	0.042	0.102	0.035	0.051	0.500
Joey	0.185	0.132	0.025	0.112	0.026	0.054	0.467
Monica	0.184	0.137	0.020	0.123	0.042	0.063	0.431
Phoebe	0.171	0.123	0.017	0.130	0.029	0.079	0.451
Rachel	0.169	0.144	0.030	0.111	0.025	0.104	0.417
Ross	0.180	0.098	0.029	0.111	0.016	0.071	0.493
Other	0.175	0.102	0.025	0.092	0.021	0.057	0.529

Table 5.5. *Emotion distribution per speaker for MELD dataset, for validation data.*

Speaker	Joy	Surprise	Fear	Anger	Disgust	Sadness	Neutral
Chandler	0.168	0.178	0.069	0.099	0.000	0.079	0.406
Joey	0.114	0.107	0.047	0.134	0.007	0.101	0.490
Monica	0.175	0.146	0.036	0.124	0.015	0.095	0.409
Phoebe	0.130	0.119	0.022	0.200	0.005	0.097	0.427
Rachel	0.176	0.189	0.067	0.091	0.030	0.110	0.335
Ross	0.106	0.101	0.018	0.189	0.055	0.106	0.424
Other	0.186	0.135	0.013	0.083	0.006	0.103	0.474

Table 5.6. *Emotion distribution per speaker for MELD dataset, for test data.*

Speaker	Joy	Surprise	Fear	Anger	Disgust	Sadness	Neutral
Chandler	0.090	0.113	0.034	0.142	0.037	0.074	0.509
Joey	0.180	0.097	0.017	0.119	0.022	0.068	0.496
Monica	0.162	0.118	0.020	0.191	0.038	0.058	0.413
Phoebe	0.172	0.131	0.031	0.082	0.038	0.069	0.450
Rachel	0.188	0.107	0.008	0.160	0.022	0.115	0.400
Ross	0.150	0.102	0.019	0.142	0.027	0.075	0.485
Other	0.143	0.095	0.009	0.093	0.007	0.077	0.577

Observing the above tables we can see that the train and test data are more similar in terms of the distribution of emotions per speaker, compared to the validation data. A reason for this could be the smaller size of the validation set. We can also see that each speaker's distribution of emotions is largely affected by the fact that the dataset is imbalanced, with the prevalent emotion being that of neutrality, which is that the of majority class.

5.5.2 IEMOCAP

General information

IEMOCAP [93] is a multimodal dataset containing dyadic conversations between actors. It is separated in five sessions, in each of which a different male and female actor take part, and it contains scripted scenarios, which the actors where asked to memorize beforehand, as well as improvised conversations, based on hypothetical scenarios, designed to elicit specific emotions. It contains all three modalities (visual, audio, text) and its utterances elicit the emotions of happiness, sadness, anger, surprise, fear, disgust, frustration, excitement and neutrality.

We note that, following the literature for the task of Emotion Recognition in Conversations [39] [17], we take only six of the aforementioned emotions into account: neutrality, sadness, happiness, anger, frustration and excitement.

IEMOCAP is not originally separated into clear training, development and test sets, but for ERC the last two speakers, (meaning all of the the fifth session) are used as the test set [17] [11]. Regarding the split of the training set into training and validation, we choose to keep the first three sessions for training and the fourth for validation, thus ensuring that the training and validation set do not have common speakers, as is the case for the training and test set.

General dataset statistics

The following table shows some general statistics regarding dialogues and utterances for the training, development and test split (as chosen by us, following previous works):

Table 5.7. *General statistics for IEMOCAP.*

	train	development	test
# sessions	3	1	1
# dialogues	90	30	31
# utterances	4246	1512	1622
Average/Max utterance length	12.0/89	12.9/90	13.3/107
# speakers	6	2	2
Average # of utterances per dialogue	47.2	50.4	52.3
Average # of emotions per dialogue	3.4	3.4	3.4
Average/Max # of speakers per dialogue	2/2	2/2	2/2

From the above table we an see that the chosen split for IEMOCAP follows a 57.5%-20.5%-22.0% analogy for the train-development-test set. The dataset is strictly dyadic and its utterances are usually quite short (although longer than those of MELD), but there are exceptions of very long utterances.

Emotion class distribution statistics

The following table depicts the Emotion class distribution for IEMOCAP, for each of the train, development and test splits:

Table 5.8. *Class distribution (%) for IEMOCAP.*

	train	development	test
anger	14.27	21.63	10.48
excitement	11.87	15.74	18.43
frustration	23.25	31.81	23.49
happiness	9.11	4.30	8.81
neutral	25.11	17.06	23.67
sadness	16.39	9.46	15.10

As we can see from the above table, the emotion distribution demonstrates strong similarities between the train, development and test split (for example happiness is the most rare emotion), but there are also differences in the distributions (for example the neutral emotion is more prevalent in the train and test set and not in the validation set). In addition, we notice that the dataset is unbalanced, although not as largely as MELD (see Table 5.2).

Speaker distribution and statistics

As analyzed earlier, IEMOCAP includes ten speakers, six of which belong to our train set, and two to each of the development and test sets. The following table depicts the number of utterances for each of the speakers. As the speakers in IEMOCAP do not have names, we label them as F_i and M_i, where i the session number and F/M are used to annotate the female of male speaker of the conversation.

Table 5.9. *# Utterances per speaker in IEMOCAP.*

speaker	# utterances
F_0	635
M_0	730
F_1	646
M_1	702
F_2	721
M_2	812
F_3	716
M_3	796
F_4	751
M_4	871

From the above table we can see that ten speakers have a similar number of utterances making IEMOCAP relatively balanced with regard to speaker distribution.

Finally, in the following table, we present the distribution of emotion for each of the ten speakers for the train (F_0 - F_2), validation (F_3 - M_3) and test (F_4 - M_4) data:

Table 5.10. *Emotion distribution per speaker for IEMOCAP dataset.*

Speaker	Anger	Excitement	Frustration	Happiness	Neutral	Sadness
F_0	0.231	0.099	0.169	0.109	0.269	0.123
M_0	0.112	0.110	0.237	0.090	0.292	0.159
F_1	0.104	0.149	0.255	0.108	0.209	0.175
M_1	0.100	0.162	0.228	0.067	0.323	0.120
F_2	0.128	0.067	0.276	0.111	0.180	0.239
M_2	0.182	0.127	0.225	0.068	0.234	0.164
F_3	0.286	0.215	0.263	0.043	0.106	0.087
M_3	0.153	0.106	0.368	0.043	0.229	0.102
F_4	0.104	0.109	0.214	0.103	0.294	0.176
M_4	0.106	0.249	0.253	0.076	0.187	0.130

We can see that the distribution of emotions follows some similar patters for different speakers, however there are also many differences between speakers.

Chapter 6

Proposed Approaches

6.1 Comparing prompt-tuning and fine-tuning for ERC

6.1.1 Introduction

Following our analysis so far, we recognize the potential of using prompts, as an alternative to fine-tuning, as well as an additional bias to adapt a pre-trained language model to a downstream task. However, while multiple variations for using prompts have been proposed, the adaptation of such methods for individual downstream tasks has not been researched as much and there is limited work using prompts in task-specific models. Our approach is thus aimed to study the applicability of prompt-based learning to the task of ERC in comparison to fine-tuning and set a baseline for prompt-based learning for Emotion Recognition in Conversation. For this purpose, we experiment with a very simple, baseline architecture, as well as commonly used methods in previous works on ERC, such as the integration of speaker-specific information through the input or through an auxiliary speaker-related task. We leverage a pre-trained language model and modify its architecture to adapt it to both a prompt-based learning and a fine-tuning setting, for the task of ERC.

We first compare the two adaptation methods using a very simple model, thus setting our baseline, and study the effect of prompt usage and prompt size. We then experiment with the prepending of speaker names in the input. This is a technique commonly used in previous work for ERC [13] [14], as it allows the integration of speaker information and helps the model recognize inter- and intra-dependencies more easily. When prepending speaker names to utterances, the input format of the model is changed: The context is no longer a sequence of continuous sentences; it is interrupted by speaker names. This input format is different from that used in the pre-training phase of our language model, which means that the pre-trained model must be adapted to the new input format in order to leverage the additional information. When fine-tuning, as in previous work using this method, the model can shift its parameters accordingly, and thus adapt effectively. However, when the model is frozen, as is usually the case with prompt-based learning, this is not possible. The model can only change the way the main language model's output is interpreted at the top level (when using a trainable output layer), or the way different parts of the input interact, through the prompt parameters. Experimenting with speaker prepending can therefore provide insight into the degree of adaptability prompts can offer for a more complex task, such as Emotion Recognition in Conversations. Finally, we experiment with the usage of an auxiliary speaker-identification task and multitask training. The integration of speaker-specific information through such a task has been proposed in works such as [11] [15] and has been found to be effective in a fine-tuning setting. However, in a setting where the trainable parameters are limited and confined only to the input level of the model, as is the case of prompt-based learning, the capacity for adaptation of the model for the combined solution of two tasks is not apparent and is worth investigating.

6.1.2 Method

We utilize the pre-trained BERT language model as our backbone model. The core architecture of our model can be broken down into three main modules, the embedding layer, the encoder module and the output layer. The embedding layer is the only one that differs between the prompt-based learning and the fine-tuning setting. For prompt-based learning, we follow the architecture proposed by [31], encoding our prompts directly in the embedding space of our model and using a series of prompt-embeddings which we prepend to the embeddings calculated by our pre-trained model's embedding layer, that correspond to the text input. In the case of fine-tuning, the embedding layer of our pre-trained model is used unchanged, so it only calculates the embeddings that correspond to the text input. The encoder module and the output layer remain unchanged for both adaptation methods: The first consists of 12 pre-trained transformer encoder layers (this is the BERT encoder), while for the second we employ two linear layers with a dropout layer between them, in order to map the encoder's output representation to a label. We perform our experiments on the two ERC datasets described in 5.5, MELD and IEMOCAP.

In the following paragraphs we present an analytical overview of our proposed prompt-based models' architecture, while in sub-section 6.1.3 we present the models we employ for comparison: the fine-tuned model as well as a model with only a trainable output layer. A schematic overview of our prompt-based model can be seen in Figure 6.3 for the task of ERC and in Figure 6.4 for the case of multitask training.

Text input format

We use the two-sentence input format of BERT. As the first sentence, we concatenate the k most recent utterances before the current utterance to be classified, in order for the model to leverage those as context. We set the current utterance to be classified as the second sentence, followed by a $[MASK]$ token. Reformulating the classification task into a language modeling task brings the input format closer to our language model's pre-training phase, and we found that it improves performance. We separate the two sentences with as $[SEP]$ token and add a $[CLS]$ token at the beginning of our input sequence and a $[SEP]$ token at the end. The final text input is formulated as following:

$$[CLS]|u_{i-k}|u_{i-k+1}|...|u_{i-1}|[SEP]|u_i|[MASK]|[SEP]$$

where i is the index of current utterance to be classified, k is the number of previous utterances used as context, u_i is the utterance with index i and $|$ is the symbol for concatenation.

In the case of speaker information prepending, the described input format is slightly changed: Instead of prepending the speaker's name to each utterance of the text input, as is usually the case in previous work, we choose to map each speaker of our dataset to one of BERT's unused tokens and prepend the speaker's corresponding token before each of the speaker's utterances. (There are 994 unused tokens in BERT's vocabulary). We consider this to be a better alternative than prepending the speaker's actual names, as these may not exist in BERT's vocabulary, they may be split into multiple tokens by BERT's tokenizer, and they may even not exist in the dataset (for example, speakers do not have names in IEMOCAP). By mapping speakers to unused tokens, we achieve a $1-1$ mapping between speakers and tokens, applicable to all speaker names and datasets.

We must note that we use the same mapping for speakers in the train, validation and test set, so the same speaker is mapped to the same token in validation and test time, as the one to which the speaker was mapped during train time. This allows the model to use speaker-specific information learned in train time during testing too, in the case of datasets that have the same speakers in the train and test set, like MELD. In IEMOCAP, as the test speakers are different than those of the train set, this has no effect.

The final text input is formulated as:

$$[CLS]|un_l|u_{i-k}|un_m|u_{i-k+1}|...|un_o|u_{i-1}|[SEP]|un_p|u_i|[MASK]|[SEP]$$

where i is the index of current utterance to be classified, k is the number of previous utterances used as context, u_i is the utterance with index i, un_j represents BERT's unused token with index j and l, m, o, p are example token indexes. Concatenation is represented by the symbol $|$.

Finally, we use a slightly different input format in the case of multitask training for the tasks of ERC and Speaker Identification: As we attempt to identify the speaker of each utterance, it is important to provide the model with the information of which utterances of the context (the first sentence of our input) belong to the current utterance's speaker. This will help the model identify the speaker, as the current utterance on its own is often too short or generic, and the dialog context consists of multiple speakers' utterances, so if we do not specify the change in turns, the model will not be able to discriminate between the different speakers easily. For this reason, we use two of BERT's unused tokens, mapping one of them (we symbolize it as "un_0") to the current speaker and the other (we symbolize it as "un_1") to all other speakers in the dialog. We prepend "u_0" to all utterances of the current utterance's speaker and "un_1" to all other utterances. In addition, we add a second $[MASK]$ token at the end of our input, to represent the model's prediction for the speaker identity. The final input is formulated as following:

$$[CLS]|un_0|u_{i-k}|un_1|u_{i-k+1}|...|un_1|u_{i-1}|[SEP]|un_0|u_i|[MASK]|[MASK]|[SEP]$$

(where the particular utterances that belong to the current speaker as chosen randomly for the context utterances.)

The text input is then tokenized using BertTokenizer and the input tokens (n in total) are passed through BERT's pre-trained embedding layer, resulting in n embeddings. We note that the number k of utterances to be used as context is calculated so that the resulting number n of embeddings has the maximum value of $512 - t$, where 512 is BERT's maximum input size and t is the number of prompt tokens to be used, as explained in the following paragraph.

Prompt tokens

Following [31], we use a sequence of prompt tokens that lie directly in the embedding space (we call them prompt token embeddings), leading to a prompt of dimensions $t * d$, where t represents the number of prompt tokens and d represents the embedding dimension (768 for bert-base).

By writing that the prompt tokens lie in the embedding space we mean that these are created in the embedding level of the model and not the text token / input level, by initializing an embedding matrix of dimensions $t * d$, using the embedding weights that token embeddings of words from the BERT's vocabulary have. The initialization can either be random, so using random words from BERT's vocabulary, or specified, for example using the token embeddings of the words corresponding to the labels of the classification task [31]. We choose to follow the second approach, initializing the prompt token embeddings with the weights of the token embeddings corresponding to the dataset's class labels (i.e joy, sadness etc.). As the prompt token embedding weights are changed during training using back-propagation, they no longer correspond to actual tokens from BERT's vocabulary, but they could be considered as some kind of "soft" tokens, lying optimally in between BERT's words.

Position and segment embeddings addition

As analyzed earlier (see sub-section 3.3.3), looking into BERT's embedding layer architecture, we can see that three types of embeddings are used: token embeddings, position embeddings, and token type embeddings. Therefore, in the embedding level, we add BERT's pre-trained position and segment embeddings to the prompt token embeddings. The position and segment embeddings are also added to the token embeddings that correspond to our text input, using BERT's pre-trained embedding layer. We will refer to the final embedding output after this summation as prompt embeddings and text embeddings in the following paragraphs.

Final embedding-level output

Having obtained both prompt and text embeddings, we then concatenate them, resulting in $t + n$ embedding representations, and pass the concatenated representations as input to the BERT encoder's layers. Figure 6.1 provides an schematic overview of the embedding layer's output.

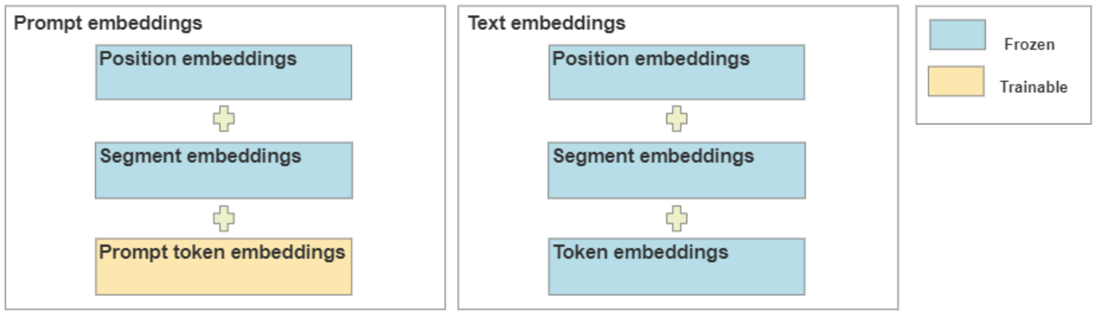


Figure 6.1. *Embedding layer overview for the prompt-based learning case. The final embedding output is a concatenation of the prompt and text embeddings. The modules in blue color are kept frozen during training, while the modules in yellow are trainable.*

Classification head

To obtain a classification label for each sample we use a classification head as our output layer, consisting of two linear layers and a dropout layer between them, and pass as input to the classification head the last hidden representation of our input's [MASK] token. The classification head therefore acts as a mapping function, mapping the model's output for the [MASK] token to one of the class labels of our classification task. We note that the lower linear layer has an input and output shape of 768 (BERT's hidden representation dimension) and the upper linear layer has an input shape of 768 and an output shape of c, where c is the number of classes in our dataset. A schematic diagram of the classification head described can be seen in Figure 6.2.

In the case of multitask training, we use a second classification head for the speaker-identification task, which consists again of two linear layers and a dropout layer between them. The lower linear layer has an input and an output dimension of d, where d is the size of BERT's hidden representations (768). The upper linear layer has an input size of d, and an output size of s where s is the number of speaker classes.

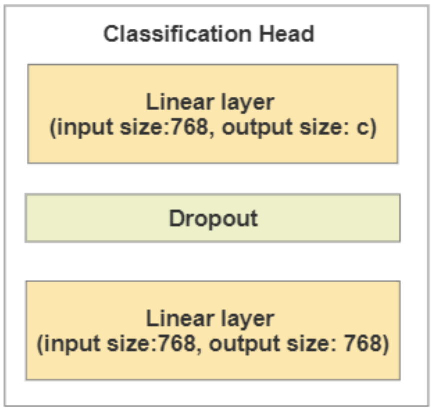

Classification Head
Linear layer
(input size:768, output size: c)
Dropout
Linear layer
(input size:768, output size: 768)

Figure 6.2. *Classification head diagram, where c represents the number of classes of the dataset.*

Frozen and trainable parts

To study the effect of prompt-based learning as an alternative to fine-tuning, we freeze all model parameters, except from the classification head's and the prompt's parameters, which we train through back-propagation. We must note that we keep the segment and position embeddings' parameters frozed for both the text input and the prompt tokens, and we also do not train BERT's token embeddings which are used to map the text input to the embedding space.

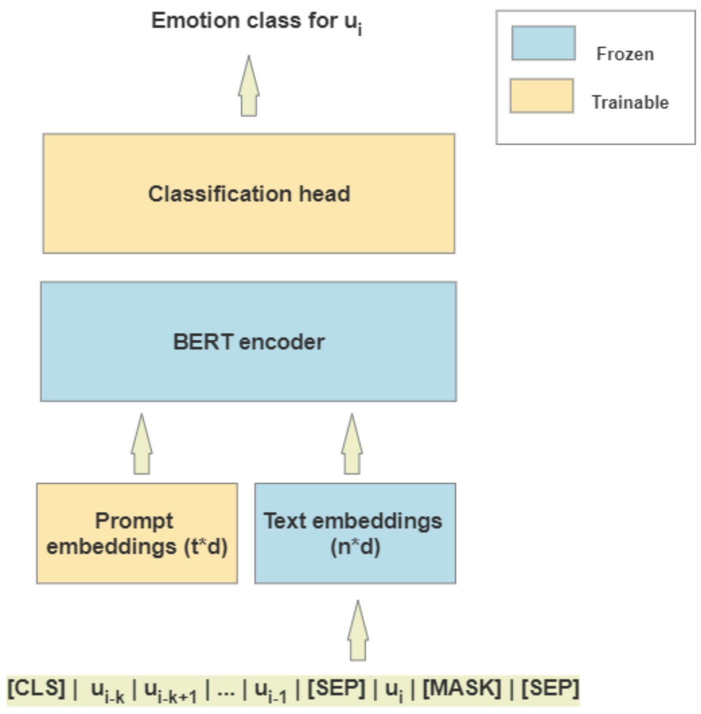


Figure 6.3. *General diagram of our prompt-based model in the case where no speaker-token prepending is used. t is the number of prompt tokens, d is BERT's embedding dimension (768), n is the number of embeddings corresponding to our text input, k is the number of past utterances used as context and i the index of the current utterance to be classified. "|" symbolizes concatenation. The modules in blue color are kept frozen during training, while the modules in yellow are trainable. The modules in blue color are kept frozen during training, while the modules in yellow are trainable for the prompt-based learning case. Note that the prompt embeddings block has trainable parts, but is not trainable as a whole: See Figure 6.1 for a detailed overview.*

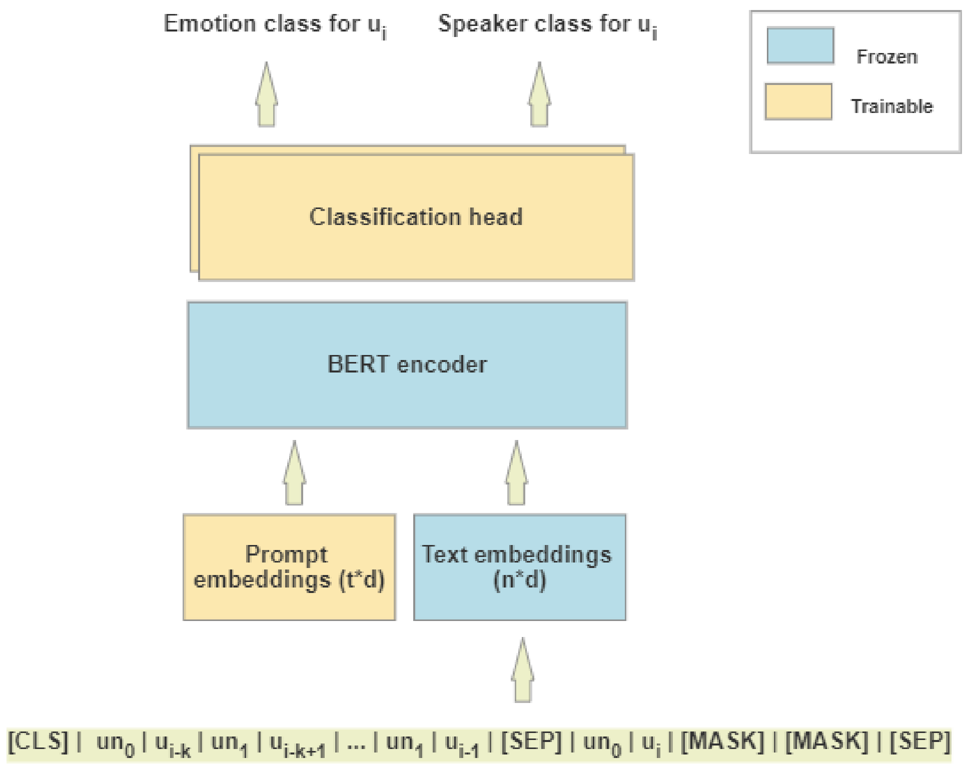


Figure 6.4. *General diagram of our prompt-based model used for multitask training, where t is the number of prompt tokens, d is BERT's embedding dimension (768), n the number of embeddings corresponding to our text input, k is the number of past utterances used as context and i is the index of the current utterance to be classified. "|" symbolizes concatenation. The modules in blue color are kept frozen during training, while the modules in yellow are trainable. Note that the prompt embeddings block has trainable parts, but is not trainable as a whole: See Figure 6.1 for a detailed overview.*

6.1.3 Baselines

Fine-tuned model

To study the performance of the usage of prompts in comparison to fine-tuning, we use a model similar to the prompt-based model described so far, but with a different embedding layer, which only consists of the text embeddings module. We use the same input format with the prompt-based learning case for our text input, both in the simple case and in the cases of speaker-token prepending and speaker-identification. As we do not use prompt tokens, we choose the number of k past, context-utterances so that the maximum value of the embeddings that correspond to the input n is no bigger that 512 (BERT's maximum input size). We also employ the same classification head as the one described for our prompt-based model. We train this model using fine-tuning, so we do not freeze any parameters during training. Figure 6.5 depicts the general architecture of the model used for fine-tuning in the simple case, with no speaker-specific information.

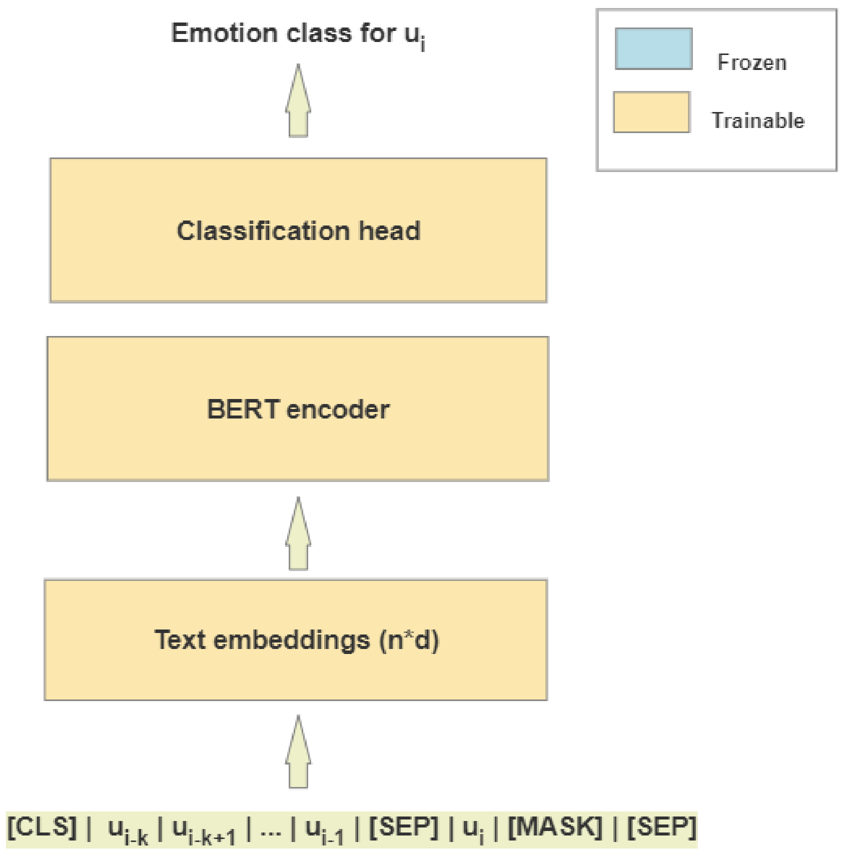


Figure 6.5. *General diagram of our fine-tuned baseline model, in the case where no speaker-token prepending is used. d is BERT's embedding dimension (768), n is the number of embeddings corresponding to our text input, k is the number of past utterances used as context and i is the index of the current utterance to be classified. "|" symbolizes concatenation. The modules in blue color are kept frozen during training, while the modules in yellow are trainable.*

Model with trainable classification head

In order to be able to fully understand the contribution of prompts to the observed performance of our proposed models, we must also separate the prompt embeddings' from the classification head's contribution. For this reason, we deploy as second model as our baseline, same as the fine-tuned model, but we keep all of this model's parameters frozen, except from the classification head's parameters. Figure 6.6 depicts the general model architecture.

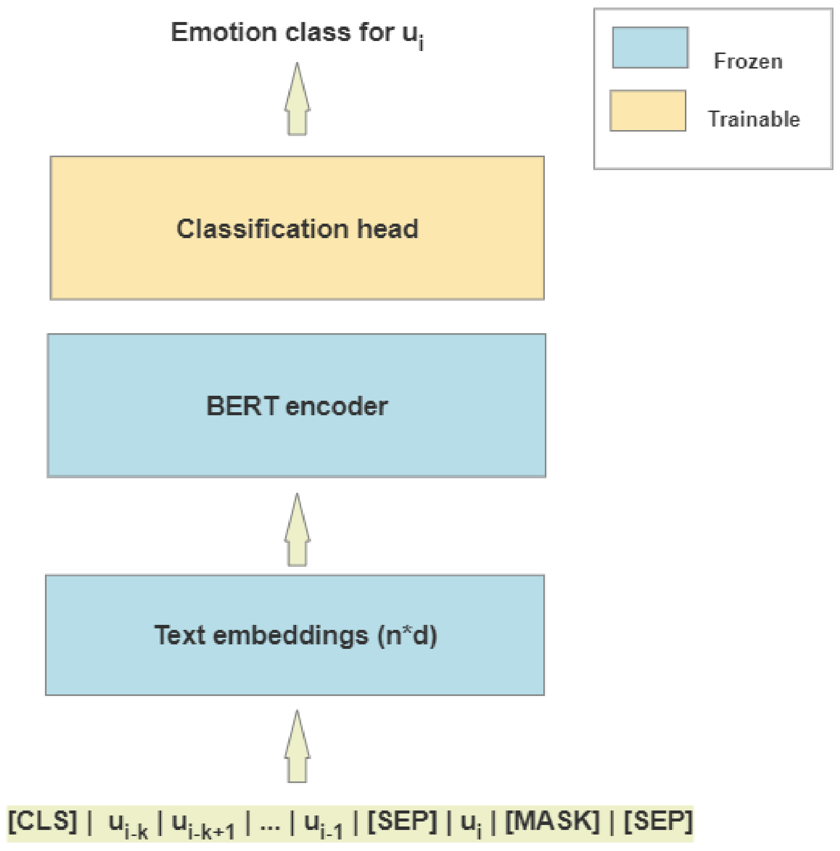


Figure 6.6. *General diagram of our baseline model with only a trainable classification head, where d is BERT's embedding dimension (768), n is the number of embeddings corresponding to our text input, k is the number of past utterances used as context and i is the index of the current utterance to be classified. "|" symbolizes concatenation. The modules in blue color are kept frozen during training, while the modules in yellow are trainable.*

6.1.4 Experimental setup

Dataset

In the case of the simple prompt-based model with no speaker-specific information and the prompt-based model with speaker-token prepending we experiment on both MELD and IEMOCAP. In the case of multitask training with the auxiliary task of speaker-identification we experiment only on MELD: As analyzed in sub-section 5.5, there are six main speakers in MELD, with more than 60 utterances each, and all other speakers have very few utterances. We therefore create seven classes for the speaker identification task, one for each main speaker and one annotated as "Other", in which all other speakers of the dataset are classified. This transforms the speaker identification task to a seven-class classification task. We do not test our method on IEMOCAP, as this dataset is for the most part a scripted, acted dataset, with different scenarios. This means that the speakers do not have a specific personality, but are actors playing different roles in different scripts. In addition, all speakers of the test set are unseen, so the model cannot learn speaker-specific clues about them from the train set. For this reason, we believe that there is no point in attempting to identify each particular speaker, and therefore do not perform multitask training using speaker-identification as an auxiliary task.

Loss function

For our simple prompt-based model and our prompt-based model using speaker-token prepending we use crossentropy loss as our loss function, which is defined in sub-section 2.3.2. In the case of multitask training we use a multitask loss function following the method of homoscedastic uncertainty [94], as in [11]. The loss is calculated as:

$$L = \frac{1}{2(\sigma_1)^2} L_e + \frac{1}{2(\sigma_2)^2} L_s + log(\sigma_1 \sigma_2) \tag{6.1}$$

where σ_1 and σ_2 are trainable noise parameters.

Training setup

We experiment with different learning rates for each of the models and keep the ones that achieve the best results, that is, a learning rate of $1e^{-4}$ for the prompt-based model and the model with only a trainable classification head and a learning rate of $1e^{-5}$ for the fine-tuned model, except from its classification head and the multitask loss (in the case of multitask training), for which we use a learning rate of $1e^{-4}$. We train for a maximum of 30 epochs for the prompt based model and the model with only a trainable classification head and for a maximum of 5 epochs for the fine-tuned model, and keep the model version of the epoch that achieved the highest weighted F1-score.

Prompt token number

For the prompt-based model, we experiment with different numbers of prompt tokens to determine the best value for t. Our experiment results are presented in Section 6.1.5. Specifically we experiment with using $t = c$ prompt tokens, where c the number of classes in each dataset, so $c = 7$ for MELD and $c = 6$ for IEMOCAP, as well as using $t = 30$, 40 and 60 prompt tokens for the simple case and $t = 30$ prompt tokens for the case of speaker-token prepending. In the case of multitask training we use $t = 30$ prompt tokens. We initialize each of the prompt tokens with the embedding of the label of one of the c classes. (When $t > c$, multiple prompt tokens will be initialized with the same embedding).

6.1.5 Results and discussion

The bellow Tables present the results for our experiments on MELD and IEMOCAP, in the case where no speaker-token prepending is used. Tables 6.1 and 6.2 depict the weighted F1-score for MELD and IEMOCAP, while Tables 6.3 and 6.4 depict the per-class F1-score for MELD and IEMOCAP respectively. All our results are an average of three runs.

Table 6.1. *Weighted F1-score (%) for MELD dataset*

Model	MELD
Fine-tuned baseline	57.06
Class. head only baseline	54.90
Prompt-based, 7 prompt tokens	56.23
Prompt-based, 30 prompt tokens	56.70
Prompt-based, 40 prompt tokens	56.94
Prompt-based, 60 prompt tokens	56.53

Table 6.2. *Weighted F1-score (%) for IEMOCAP dataset*

Model	IEMOCAP
Fine-tuned baseline	64.44
Class. head only baseline	56.46
Prompt-based, 6 prompt tokens	58.95
Prompt-based, 30 prompt token	59.94
Prompt-based, 40 prompt tokens	59.21
Prompt-based, 60 prompt tokens	59.24

Table 6.3. *F1-score (%) per class for MELD test dataset*

Model	Anger	Disgust	Fear	Joy	Neutral	Sadness	Surprise
Fine-tuned baseline	41.33	15.66	17.66	51.33	72.00	31.67	53.33
Class. head only baseline	39.00	13.33	10.67	44.00	72.00	25.33	52.33
Prompt-based, 7 prompt tokens	40.00	15.67	9.00	46.33	73.00	32.33	52.00
Prompt-based, 30 prompt tokens	40.33	20.67	14.33	47.0	72.67	33.67	52.00
Prompt-based, 40 prompt tokens	40.33	20.67	16.00	46.67	73.00	35.00	51.67
Prompt-based, 60 prompt tokens	41.33	17.33	14.0	47.00	72.00	34.33	52.00

Table 6.4. *F1-score (%) per class for IEMOCAP test dataset*

Model	Anger	Excitement	Frustration	Happiness	Neutral	Sadness
Fine-tuned baseline	62.00	69.00	59.33	54.00	63.37	75.67
Class. head only baseline	55.33	58.33	57.00	33.33	56.33	67.33
Prompt-based, 6 prompt tokens	56.67	62.00	59.67	35.00	58.67	69.67
Prompt-based, 30 prompt tokens	54.33	64.67	62.00	39.00	57.67	71.33
Prompt-based, 40 prompt tokens	54.00	62.00	60.33	40.67	58.00	70.67
Prompt-based, 60 prompt tokens	54.33	62.33	60.00	40.67	57.67	71.33

Observing Table 6.1 we can see that, for MELD, the fine-tuned baseline model yields an F1-score of 57.06%, while the best of the prompt-based models, the model with 40 prompt tokens, leads to an F1-score of 56.94%. This means that for the MELD dataset, the prompt based model is comparable to the fine-tuned model. However, from Table 6.2 we see that for the IEMOCAP dataset, the fine-tuned baseline achieves a F1-score higher than that of the best performing prompt-based model (that uses 30 prompt tokens), by 4.50%, suggesting the fine-tuning adaptation method's superiority in this case.

One reason for the difference between the two datasets could be their origin: While IEMOCAP utterances have been designed with the purpose of eliciting a specific emotion and are uttered in a two-speaker scenario, MELD is derived from a TV-series, where multiple speakers are talking and the emotion of each utterance could be more subtle or expressed in very different ways by different speakers and in different situations. The number of speakers in MELD is also substantially larger than that in IEMOCAP. We could therefore assume that fine-tuning, which tunes millions of parameters according the data and allows the model to fit the data more closely has an advantage in IEMOCAP, compared to the less adaptive prompt-based learning. On the other hand, in MELD, because of the bigger diversity and difficulty of the dataset, a closer fitting of the model to the training data does not necessarily lead to a better generalization ability, and fine-tuning is thus not as advantageous.

Speaker-token prepending

The bellow Tables present the results for MELD and IEMOCAP, in the case where speaker-token prepending is used. Tables 6.5 and 6.6 depict the weighted F1-score for MELD and IEMOCAP, while Tables 6.7 and 6.8 depict the per-class F1-score for MELD and IEMOCAP respectively. We also include some of the results from Tables 6.1 and 6.2, for easier comparison. All our results are an average of three runs.

Table 6.5. *Weighted F1-score (%) for MELD dataset*

Model	F1-score (Test)
Prompt-based, 7 prompt tokens	56.23
Prompt-based, 30 prompt tokens	56.70
Fine-tuned speaker-token prepending	56.14
Class. head only speaker-token prepending	54.90
Prompt-based speaker-token prepending, 7 prompt tokens	56.64
Prompt-based speaker-token prepending, 30 prompt tokens	57.63

Table 6.6. *Weighted F1-score (%) for IEMOCAP dataset*

Model	F1-score (Test)
Prompt-based, 6 prompt tokens	58.95
Prompt-based, 30 prompt tokens	59.94
Fine-tuned speaker-token prepending	64.99
Class. head only speaker-token prepending	56.16
Prompt-based speaker-token prepending, 6 prompt tokens	59.04
Prompt-based speaker-token prepending, 30 prompt tokens	59.12

Table 6.7. *F1-score (%) per class for MELD test dataset*

Model	Anger	Disgust	Fear	Joy	Neutral	Sadness	Surprise
Prompt-based, 7 prompt tokens	40.00	15.67	9.00	46.33	73.00	32.33	52.00
Prompt-based, 30 prompt tokens	40.33	20.67	14.33	47.0	72.67	33.67	52.00
Fine-tuned speaker prepending	43.00	24.33	17.67	48.33	70.33	34.00	52.00
Class. head only speaker prepending	40.67	9.67	11.00	43.67	72.00	28.00	50.00
Prompt-based speaker-token prepending, 7 prompt tokens	38.00	16.67	11.00	47.67	73.00	32.67	55.00
Prompt-based speaker-token prepending, 30 prompt tokens	40.67	25.00	15.67	48.67	73.00	34.67	54.33

Table 6.8. *F1-score (%) per class for IEMOCAP test dataset*

Model	Anger	Excitement	Frustration	Happiness	Neutral	Sadness
Prompt-based, 6 prompt tokens	56.67	62.00	59.67	35.00	58.67	69.67
Prompt-based, 30 prompt tokens	54.33	64.67	62.00	39.00	57.67	71.33
Fine-tuned speaker prepending	63.33	69.00	61.67	50.67	63.33	77.00
Class. head only speaker prepending	50.67	56.33	55.67	35.67	59.00	67.67
Prompt-based speaker-token prepending, 6 prompt tokens	52.00	62.67	59.33	42.00	58.33	70.67
Prompt-based speaker-token prepending, 30 prompt tokens	55.67	61.33	60.67	41.33	57.67	69.67

Comparing the results of the simple prompt-based models and those of the prompt-based models using speaker-token prepending, we can see that speaker-token prepending tends to improve performance (with the exception of the case of the prompt-based model with 30 prompt tokens for IEMOCAP, in Table 6.6, where a slight drop in performance is observed). Specifically for MELD, we can see from Table 6.5 that, when using speaker-token prepending, the F1-score rises by 0.41% in the case of 6 prompt tokens and by 0.93% in the case of 30 prompt tokens, suggesting that a larger number of prompt token parameters allows the model to adapt to the new input format and integrate the speaker-specific information more effectively. However, for IEMOCAP, we can see in Table 6.6 that, while the 30 prompt token model that uses speaker-token prepending yields a better performance compared to the one with 6 prompt tokens, the prepending of prompt tokens has an overall negative effect, with the baseline prompt-based model with 30 prompt tokens achieving a higher performance by 0.90% compared to the one using speaker-token prepending. Similarly, we observe that for the prompt-based model with 6 prompt tokens, speaker-token prepending did not lead to notable improvement in performance either. Looking into the fine-tuned models as a comparison (Tables 6.1, 6.2, 6.5 and 6.6), in the case of MELD, speaker-token prepending actually leads to a 0.92% drop in performance, while in the case of IEMOCAP the performance increased by 0.55%.

The above results suggest that encoding speaker-specific information can indeed be helpful for the ERC task, as previous work suggests. However, the successful utilization of this information depends on the adaptation method and model architecture used. In the case of MELD, prompt-based learning seems to be more successful in leveraging the additional information compared to fine-tuning, suggesting that prompts do have the capacity to adapt the model to the new input format and can even be more effective than fine-tuning in leveraging the additional information. For IEMOCAP, fine-tuning is more effective, while prompt-based learning seems to fail in utilizing the speaker-specific information provided. We can therefore conclude that, in integrating speaker-specific information provided through our changed input format, neither prompt-based learning nor fine-tuning is superior, and the optimal method largely depends on the dataset.

Multitask training with the auxiliary task of speaker-identification

The bellow Tables present the results for MELD, for our proposed prompt-based model that uses speaker-identification as an auxiliary task. We also show the results for the corresponding fine-tuned model as well as our simple prompt-based model with 30 prompt tokens and no speaker-specific information and its corresponding fine-tuned model, for comparison purposes. Table 6.9

presents the weighted F1-score, while Table 6.10 depicts the per-class F1-score. All our results are an average of three runs.

Table 6.9. *Weighted F1-score (%) for MELD dataset*

Model	F1-score (Test)
Fine-tuned baseline	57.06
Prompt-based, 30 prompt tokens	56.70
Emotion+Speaker multitask, fine-tuned	56.79
Emotion+Speaker multitask, prompt-based	54.65

Table 6.10. *F1-score (%) per class for MELD test dataset*

Model	Anger	Disgust	Fear	Joy	Neutral	Sadness	Surprise
Fine-tuned baseline	41.33	15.66	17.66	51.33	72.00	31.67	53.33
Prompt-based, 30 prompt tokens	40.33	20.67	14.33	47.00	72.67	33.67	52.00
Emotion+Speaker multitask, fine-tuned	40.67	24.00	15.33	48.33	71.67	36.66	53.33
Emotion+Speaker multitask, prompt-based	37.33	12.33	11.67	44.00	71.67	32.67	48.67

From Table 6.9 we can see that using multitask training for emotion recognition and speaker identification does not improve performance for the fine-tuned model and leads to an important decrease in performance for the prompt-based model. This could suggest that speaker identification, as defined in our method, is not very suitable as an auxiliary task to improve performance for Emotion Recognition in Conversation. In addition, comparing the results of the prompt-based and the fine-tuned model, both trained with the multitask objective, the lower F1-score of the prompt-based model (by 2.14%) suggests that the capacity of prompts to adapt BERT to a more complex setting, such as the multitask setting, is not enough, and trainable parameters inside the model may be needed for a more in-depth tuning.

6.1.6 Additional discussion

The effect of prompt size

From the presented results we can conclude that prompt size can impact performance. Both for MELD and IEMOCAP, we observe that the prompt-based models tend to yield lower F1-scores for a very small number of prompt tokens (6-7 prompt tokens). The results improve by increasing the prompt tokens used, with 30 prompt tokens for IEMOCAP and 40 prompt tokens for MELD leading to the optimal performance, and then gradually decline again when further increasing prompt tokens (to 40 and then 60 for IEMOCAP, to 60 for MELD). This trend can be observed for the simple prompt-based models, as well as the prompt-based models using speaker-token prepending (where the models with 6-7 prompt tokens lead to inferior performance compared to those with 30 prompt tokens).

These observations suggest that there is a prompt token number optimal for a specific language model in a specific task. We believe that the lower performance when using very few prompt tokens can be attributed to the fact that there are not enough trainable parameters to encode all the information needed to sufficiently adapt BERT to the task of ERC, while too many prompt tokens lead to the opposite problem, with too many parameters to be trained from the available data, possibly leading to a bigger overfitting problem.

How much of the observed performance can be attributed to the usage of prompts?

In our described architecture we have used a trainable classification head to map the representation of BERT's $[MASK]$ token to one of the emotion classes. Because the classification head is trainable, it is reasonable to assume that is contributes to the pre-trained language model's adaptation to the ERC task. In order to separate the classification head's from the prompt's contribution, we can compare the performance of our prompt-based models with that of the models we have employed that only include a frozen BERT encoder and a trainable classification head, and neither use prompts nor are fine-tuned.

From Tables 6.1, 6.2, 6.5 and 6.6 we can observe that, for both MELD and IEMOCAP, the prompt-based models tend to obtain a F1-score higher by 2%-3%, compared to the models with only a classification head, which leads us to the conclusion that prompts do indeed contribute to a pre-trained language model's adaptation to downstream tasks. In addition, it is worth noticing that, when speaker-token prepending is used in the text input, the models using a classification head fail to leverage the additional information, and the performance remains the same or drops compared to not using speaker-token prepending. This suggests that, in adapting the language model to the new input format used for speaker-token prepending, which differs from that of BERT's pre-training phase, the prompts' contribution plays the major role. An explanation for this could be that the prompts influence the model's representations through context as early as the lowest level of the BERT encoder, while the classification head can only change the $[MASK]$ token's representation's mapping to the emotion classes, and cannot affect the model's hidden representations and their interactions throughout the encoder layers.

Comparing per class performance

We can see from Tabels 6.3 and 6.7 that, in the case of MELD, the performance for each class largely depends on the emotion distribution of the dataset, with the minority classes having the lowest performance. In IEMOCAP (Tables 6.4, 6.8), which is not as unbalanced as MELD, the differences in performance among different classes are not as large and they do not depend on the class distribution as strongly. The emotion more easily recognized in IEMOCAP seems to be sadness. A reason for that could be that it is more different from the rest of the emotions (we could consider excitement to be closer to happiness and anger to frustration, while sadness does not have as strong similarities with the rest of the emotions).

6.2 Information-specific prompts: An alternative approach for integrating additional information

6.2.1 Introduction

For the task of Emotion Recognition in Conversations (ERC), topic modeling as well as speaker identity have previously been used to help the model predict emotions. The reason for that is that topics tend to carry certain language patterns [44] and can thus affect both the utterance's meaning and the particular emotion conveyed by specific expressions, while different speakers may also use different words or expressions to convey emotion. The modeling of topic and the knowledge of speaker identity can thus help a model interpret each utterance more effectively to decode its emotion. The integration of speaker-specific information is usually achieved through the prepending of speaker names to the corresponding utterances in the text input [13] [14], while the usage of an auxiliary speaker-identification task has also been utilized in previous work [11]

[15]. In the case of topic modeling, a topic-augmented language model with an additional layer specialized for topic detection has been proposed by Zhu et al. in [16]. Motivated by the positive effect both speaker-specific and topic-specific information seems to have for Emotion Recognition in Conversation, we propose a model based on a pre-trained language model and adapted to the ERC task through prompt-based learning, in which speaker-specific or topic-specific information can be provided directly through the prompts (which we call information-specific prompts), without the need for a change in the input format or a usage of additional layers. Our method is information-type agnostic, meaning that it can be implemented for different types of information, including but not limited to speaker identity and topic.

6.2.2 Method

As in Section 6.1, we use the pre-trained BERT language model as our backbone. Our model consists of three main modules, the embedding layer, the encoder module and the output module. The embedding layer is made up of three parts: The first part is a task-specific prompt: It outputs prompt embeddings in the same way as in Section 6.1, which we train on the whole train set and have the purpose of adapting the language model to the ERC task. The second part is topic- or speaker-specific, depending on whether we are working with speaker identities or topic: Assuming that we are working with s speakers or s topics in our training data, we train s different prompts, each corresponding to one speaker or topic and trained only using that speaker's or topic's data. This part of the embedding module is responsible for choosing one of the s speaker- or topic-specific prompts every time, and outputing it to be used together with the task-specific prompt. The purpose of this second part of the embedding module is to influence the way the model treats the text input according to the conversation's topic or the utterance's speaker. We believe this to be beneficial, as both the identity of the speaker as well as the topic may influence the way in which emotions are expressed through language, as explained earlier. The third part of the embedding module is responsible for calculating the embeddings that correspond to the model's text input. The encoder module and output layer of our module, are kept the same as in 6.1: The encoder consists of 12 pre-trained transformer encoder layers (this is the BERT encoder), while, as an output module we use two linear layers with a dropout layer between them, in order to map the encoder's output representation to a label. We experiment on the two ERC datasets described in 5.5, MELD and IEMOCAP. We note that we work with topic-specific and speaker-specific prompts, as this type of information is already available in our datasets or can easily be extracted for the task of ERC, but the same logic could be applied to other types of information.

In the following paragraphs we provide an in-depth description of our proposed model's architecture as well as the method we use to extract the topic of each utterance. A schematic overview of our proposed model is depicted in Figure 6.8.

Text input format

We use the same text input format as in 6.1, performing masked language modeling and using a two sentence input, where the first sentence is a concatenation of past utterances, to be used by the model as context, and the second utterance is the current utterance to be classified, followed by a $[MASK]$ token. The final text input is formulated as following:

$$[CLS]|u_{i-k}|u_{i-k+1}|...|u_{i-1}|[SEP]|u_i|[MASK]|[SEP]$$

where i is the index of current utterance to be classified, k is the number of previous utterances used as context, u_i is the utterance with index i and $|$ the symbol for concatenation.

This input is passed into the part our model's embedding layer responsible for calculating the embeddings that correspond to the text input tokens, which we call text embeddings. For this purpose, BERT's pre-trained embedding layer is used unchanged, which maps the input tokens to the corresponding pre-trained token embeddings, and then adds to them the pre-trained segment and position embeddings, outputing the final text embeddings.

Task specific prompt tokens

We use t prompt token embeddings that are trained using the whole dataset, with the purpose of adapting the model to the ERC task. These are the same as the prompt token embeddings used in 6.1, lying directly in the embedding space and resulting in the final prompt embeddings, after the summation of the trainable prompt token embeddings with BERT's position and segment embeddings.

Information-specific prompt tokens

In addition to the t task-specific prompt tokens described above, we use m more prompt tokens, that are specific to the additional source of information we attempt to integrate (i.e. speaker- or topic-specific). We call these information-specific prompt tokens.

As these prompt tokens are topic- or speaker-specific, they are not trained on the whole dataset: Assuming that we have s speakers (or speaker groups, the speakers within each are modeled together) or s topics, we keep s groups of m prompt token embeddings each, resulting in a total embedding array of dimensions $(s+m)*d$, where d is BERT's embedding dimension (768 for bert-base). For each current utterance to be classified, we determine the utterance's speaker or topic and choose the m prompt token embeddings corresponding to this speaker/topic. These m prompt token embeddings are then used together with the t task-specific prompt token embeddings for the current utterance's classification, after BERT's position and segment embeddings are summed with them.

Final embedding-level output

To classify an utterance, having obtained the text embeddings, the task-specific prompt embeddings and the speaker-/topic-specific prompt embeddings for it, we concatenate the three, resulting in $t+m+n$ embeddings (where n the number of embeddings coressponding to the text input, chosen so as the total number of embeddings does not surpass BERT's maximum input size: $t+m+n \leq 512$), which are then passed to the pre-trained BERT encoder module. A schematic representation of our proposed model's final embedding-level output is depicted in Figure 6.7.

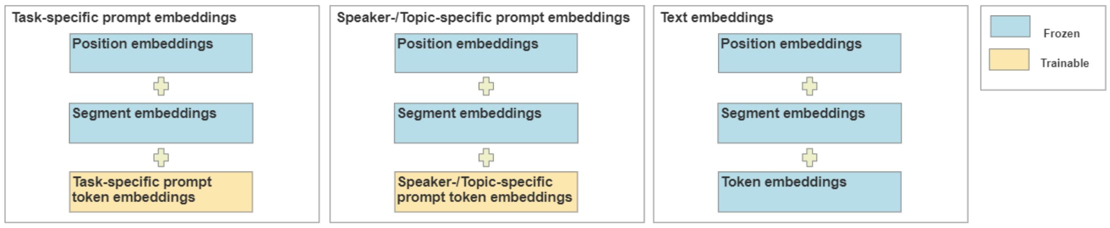


Figure 6.7. *Embedding layer overview for the prompt-based learning case. The final embedding-level output consists of the concatenation of the task-specific prompt embeddings, the information-specific (speaker- or topic-specific) prompt embeddings and the text embeddings. The modules in blue color are kept frozen during training, while the modules in yellow are trainable.*

BERT encoder and classification head

We use the pre-trained BERT encoder and add a classification head on top, in order to map the final hidden representation of our input's MASK token to one of the classes. We use the same classification head, consisting of two linear layers with a dropout layer between them, as in 6.1. (See Figure 6.2 for a detailed representation).

Frozen and trainable parts

We freeze all model parameters, except from the classification head's and the prompt's parameters (both the task-specific prompt token embeddings and the information-specific prompt token embeddings remain trainable), which we train through back-propagation. We note again that we keep the segment and position embeddings' parameters frozen for both the text input and the prompt tokens, and we also do not train BERT's token embeddings which are used to map the text input tokens to the embedding space.

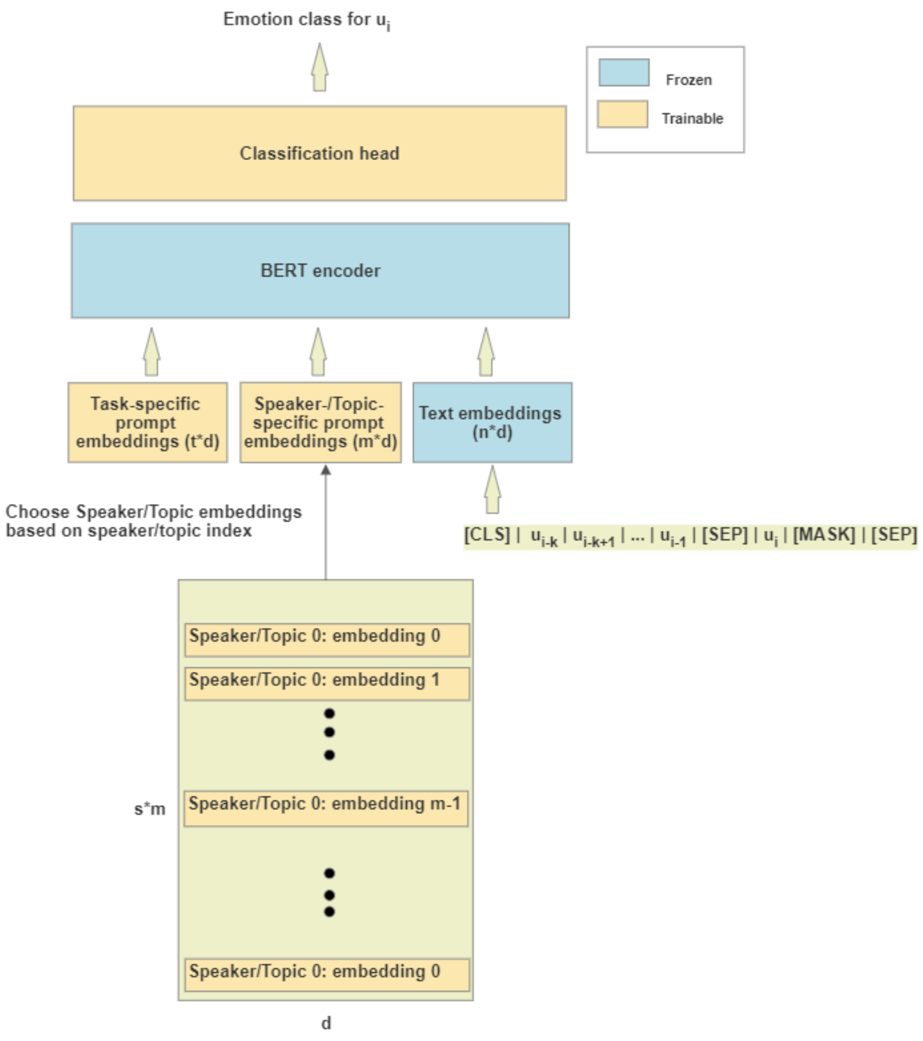

Emotion class for u_i
Frozen
Trainable
Classification head
BERT encoder
Task-specific prompt embeddings (t*d)
Speaker-/Topic-specific prompt embeddings (m*d)
Text embeddings (n*d)
Choose Speaker/Topic embeddings based on speaker/topic index
[CLS] | u_{i-k} | u_{i-k+1} | ... | u_{i-1} | [SEP] | u_i | [MASK] | [SEP]
Speaker/Topic 0: embedding 0
Speaker/Topic 0: embedding 1
Speaker/Topic 0: embedding m-1
Speaker/Topic 0: embedding 0
s*m
d

Figure 6.8. *General diagram of our prompt-based model, where t is the number of task specific prompt tokens, m is the number of information-specific (speaker-specific or topic-specific) prompt tokens, s is the number of speakers/topics, d is BERT's embedding dimension (768), n is the number of embeddings corresponding to our text input, k is the number of past utterances used as context and i is the index of the current utterance to be classified. "|" symbolizes concatenation. For s speakers/topics we train s information-specific prompts consisting of m prompt tokens, so we have a total of $s * m$ trainable prompt embeddings, of which m are chosen every time, according to the speaker or topic. The modules in blue color are kept frozen during training, while the modules in yellow are trainable. Note that the task-specific and speaker-/topic-specific embedding blocks have trainable parts, but are not trainable as a whole: See Figure 6.7 for a detailed overview.*

Topic extraction method

While the speaker identity is provided in both IEMOCAP and MELD, this is not the case for the topic of each utterance. In order to use topic-specific prompts, we thus first perform topic modeling to extract each utterance's main topic.

We use Latent Dirichlet Allocation (LDA) [45] to perform topic modeling [46]. We note that for each utterance, we use this utterance and its context (the k previous utterances in the dialog that are given as input to the model together with the current utterance) as a whole (as a document for our topic modeling analysis), in order to be able to identify the conversation's current topic more effectively. This is necessary, as utterances in a dialog are often small and general (for example: "Yeah." or "I don't know!"), so it is not possible to determine the topic using only the current utterance.

Thus, considering each utterance and its context as a document, we calculate the Document Term Matrix (DTM) for the dataset's train set. The DTM is a matrix that describes the frequency of each word in each of the documents, having a shape of $a*b$, where a is the number of documents and b is the number of words. The number b of words may be equal the total amount of words found in the documents, or it can be smaller, when criteria are applied, based on the appearance frequency of a word. For example, words appearing in less than a specified percentage of all documents are often not included.

After obtaining the DTM matrix, we can then use LDA to perform topic clustering. LDA assumes a collection of k topics, where k must be predefined. Each topic is considered as a collection of words and is assumed to have been drawn from a Dirichlet distribution $\vec{\beta}_k \sim Dirichlet(\eta)$ over the vocabulary, where η is a parameter controlling the distribution of words per topic. Lower values of η mean that the topics will likely have fewer words and higher values mean that topics will likely have more words. Given the topics, LDA assumes a generative process for each document d. During this process, each document is assumed to be a collection of topics and for each document a distribution over topics is drawn $\vec{\theta}_d \sim Dirichlet(\vec{\alpha})$. $\vec{a}$ controls the number of topics expected in the document, with lower values leading to fewer topics and higher values to more topics in the document. Then, for each word n in the document d a topic assignment $z_{d,n} \sim Mult(\theta_d) \in \{1, ..., K\}$ is drawn and the observed word $w_{d,n} \sim Mult(\beta_{z_{d,n}})$ is drawn for the selected topic ($Mult$ represents the Multivariate distribution) [95] [96]. In short, the LDA algorithm begins with a random initialization of the word distribution for each topic, and follows an iterative process with two repeating steps for document: In the first step, for each document d, first θ_d and then $z_{d,n}$ for all words n in d are updated iteratively, until convergence. Then, in the second step, the word distribution β_k for each topic is updated. The process repeats itself, until convergence.

LDA thus calculates the mixture of topics of which each document consists. This can be seen as a soft clustering of each document between each of the topics. To obtain the main topic of each document, we choose the topic with the biggest weight. We note that we perform the LDA analysis using only our train data. During test time, we simply classify each document to the existing topics, using the training data's LDA model.

Prompt ensembles

When performing topic clustering to obtain each utterance's main topic, we can assume a hard clustering, assigning one topic to each utterance (the topic with the maximum probability) and choosing the m corresponding topic-specific prompt tokens, or we can assume a soft clustering between topics, using the probabilities calculated by our topic extraction method. In the latter case, we assume that each utterance belongs to each topic with a probability p. Instead of then using one topic-specific prompt for the utterance during test time, we can calculate a prompt ensemble, by obtaining the weighted average of all s topic-specific prompts (each consisting of m prompt tokens), using the topic probabilities as weights. We call this technique prompt-ensembling. We experiment with this alternative as well and provide the results in 6.2.5. We note that the averaging is only performed during test time, while we still assume a hard clustering during training, in order to train the topic-specific prompts.

6.2.3 Baselines

Simple prompt-based model

We compare our results with those of the simple prompt-based model of 6.1, with $t' = 30$ prompt tokens, in order to determine whether the addition of speaker- or topic-specific prompts can successfully integrate useful additional speaker-/topic-specific information, in order to boost performance. To be certain whether the impovement we see in performance (if we see any) is achieved due to the successful integration of speaker-/topic-specific information and not solely because of the addition of m extra prompt tokens, we also compare our results to those of the prompt-based model of 6.1, with $t' = 40 (= t + m)$ prompt tokens.

6.2.4 Experimental setup

Dataset

Our method is based on training a speaker- or topic- specific prompt using the train set, and then using this pre-trained prompt during test time. This means that it assumes that the speaker or the topic seen during test time also exists in the train set. When working with topic, this does not present a problem, as any utterance can be mapped to the topic of the train set that it is closer to. However, when an unknown speaker is found in the test set that does not exist in the train set, there is no trained prompt for this speaker.

For this reason, in the case of speaker-specific prompts, we test our method on MELD and not IEMOCAP, as in IEMOCAP there are different speakers in each of the train, validation and test set. On the other hand, in MELD, the six main speakers are present in all three data parts (train, validation and test). We therefore use m speaker-specific prompt tokens for each of these six speakers. We group all other speakers of MELD into one, using m common speaker-specific prompt tokens for them. We thus keep only $s = 7$ different groups of prompt tokens. We choose to perform this grouping for two reasons: The first is that there are hundreds of speakers in MELD, so using m prompts for each of the speakers separately would demand a much bigger memory and would result in a lot more parameters. The second reason is that, as all speakers in MELD other than the six main ones have very few utterances, there would not be enough data to train the prompts correctly. With this approach, we thus train prompts to help the model treat each of the six main speakers differently, and we let the model treat all other speakers in a more general way. When working with topic-specific prompts we experiment on both MELD and IEMOCAP.

Training and testing setup

After experimenting with different learning rates to obtain the best results, we choose a learning rate of $1e^{-4}$ for the classification head and the task specific prompt tokens of our model and a learning rate of $3e^{-4}$ for the topic-/speaker-specific prompt tokens. We train for a maximum of 30 epochs and keep the model version of the epoch that achieved the highest weighted F1-score. We use crossentropy loss as our loss function (defined in sub-section 2.3.2).

We must note that the clustering to one of the training topics for the samples of the development and the test set in the case of topic-specific prompts can easily be performed during test time, using the LDA model which we obtained in the training phase. However we choose to perform the clustering beforehand and keep the topic (or topic distribution in the case of prompt-ensembling) as additional preprocessed information, for bigger speed during our experiments.

Prompt token number

We use $t = 30$ task-specific prompt tokens and initialize each of them with the embedding of the label of one of the c classes of our dataset. We also use $m = 10$ topic-/speaker-specific prompt tokens, for which we perform random initialization. The value of m was chosen after experimenting with multiple values, in order to achieve optimum performance. In addition, as mentioned earlier, we use $s = 7$ different groups of m speaker-specific prompt tokens, while, in the case of topic-specific prompt tokens, we use $s = 7$ different groups of m topic-specific prompt tokens for MELD and $s = 4$ different groups of m topic-specific prompt tokens for IEMOCAP, as these numbers were found to achieve the best performance.

We additionally experiment with using $t = 0$ task-specific prompt tokens and $m = 40$ topic-/speaker-specific prompt tokens, for which we perform random initialization. This means that we convert all 40 prompt tokens of our model to speaker-/topic-specific.

6.2.5 Results and discussion

The bellow Tables present the results for our proposed model as well as our baseline models as described in sub-section 6.2.3, for MELD and IEMOCAP. Tables 6.11 and 6.12 present the weighted F1-score for MELD and IEMOCAP, while Tables 6.13 and 6.14 depict the per-class F1-score for MELD and IEMOCAP respectively. All our results are an average of three runs.

Table 6.11. *Weighted F1-score (%) for MELD dataset*

Model	F1-score (Test)
Prompt-based, 30 prompt tokens	56.70
Prompt-based, 40 prompt tokens	56.94
Only speaker-specific prompts, 40 speaker-specific prompt tokens	55.68
Task- + Speaker-specific prompts, 30 task-/10 speaker-specific prompt tokens	57.08
Only topic-specific prompts, hard class. 40 topic-specific prompt tokens	54.96
Task- + Topic-specific prompts, hard class. 30 task-/10 topic-specific prompt tokens	56.63
Task- + Topic-specific prompts, prompt-ensembling 30 task-/10 topic-specific prompt tokens	56.82

Table 6.12. *Weighted F1-score (%) for IEMOCAP dataset*

Model	F1-score (Test)
Prompt-based, 30 prompt tokens	59.94
Prompt-based, 40 prompt tokens	59.21
Only topic-specific prompts, hard class. 40 topic-specific prompt tokens	58.69
Task- + Topic-specific prompts, hard class. 30 task-/10 topic-specific prompt tokens	61.13
Task- + Topic-specific prompts, prompt-ensembling 30 task-/10 topic-specific prompt tokens	61.12

Table 6.13. *F1-score (%) per class for MELD test dataset*

Model	Anger	Disgust	Fear	Joy	Neutral	Sadness	Surprise
Prompt-based, 30 prompt tokens	40.33	20.67	14.33	47.0	72.67	33.67	52.00
Prompt-based, 40 prompt tokens	40.33	20.67	16.00	46.67	73.00	35.00	51.67
Only speaker-specific prompts, 40 speaker-specific prompt tokens	39.67	10.33	12.00	46.67	72.67	27.67	52.33
Task- + Speaker-specific prompts, 30 task-/10 speaker-specific prompt tokens	40.00	19.33	17.00	48.67	73.67	33.00	52.00
Only topic-specific prompts, hard class. 40 topic-specific prompt tokens	38.67	10.67	12.67	43.33	72.67	31.67	51.67
Task- + Topic-specific prompts, hard class. 30 task-/10 topic-specific prompt tokens	38.67	18.00	14.67	47.33	73.33	32.67	51.00
Task- + Topic-specific prompts, prompt-ensembling 30 task-/10 topic-specific prompt tokens	39.33	16.00	15.33	49.33	73.33	32.33	51.00

Table 6.14. *F1-score (%) per class for IEMOCAP test dataset*

Model	Anger	Excitement	Frustration	Happiness	Neutral	Sadness
Prompt-based, 30 prompt tokens	54.33	64.67	62.00	39.00	57.67	71.33
Prompt-based, 40 prompt tokens	54.00	62.00	60.33	40.67	58.00	70.67
Only topic-specific prompts, hard class. 40 topic-specific prompt tokens	55.33	60.33	60.33	39.67	57.67	69.33
Task- + Topic-specific prompts, hard class. 30 task-/10 topic-specific prompt tokens	57.33	61.33	61.33	42.67	61.33	73.33
Task- + Topic-specific prompts, prompt-ensembling 30 task-/10 topic-specific prompt tokens	58.33	63.00	62.33	41.00	60.67	72.33

Speaker-specific prompts

Regarding the usage of speaker-specific prompts, we have only experimented on MELD, so our results are contained in Tables 6.11 and 6.13. Between the models using speaker-specific prompts, the optimum performance (F1-score equal to 57.08%) is achieved by the model using 30 task-specific prompt tokens that remain the same for the whole dataset, and 10-speaker specific prompt tokens that change according to the speaker. This model also achieves a better F1-score compared to the baseline prompt-based models, both using 30 and 40 task-specific prompt tokens, thus suggesting that speaker information can indeed be encoded effectively using prompts.

In order to obtain a better understanding of the information speaker identity may provide in MELD, we can study the emotion class distribution for each of the six main speakers, as well as the group of the rest of the speakers, as these are presented in Tables 5.4 and 5.6. We can see that the emotion class distribution for each speaker often presents similarities in the train and in the test set, which could allow the model to use an emotional bias learned for every speaker during training in test time too. Comparing the emotion class distribution between different speakers, we can see that the different speakers have a similar emotion distribution. This is partly due to the fact the MELD is very imbalanced, meaning we cannot see big percentages in minority classes, and we are more likely to see large percentages in the majority class. For this reason, while learning different speaker's tendencies towards some particular emotion may provide some benefit, we do not expect it to be a major factor in determining emotion, in most cases. We note however that the are some cases were some emotional clues can be extracted: For example Joey tends to be happy very often (compared to the others) in both the train and the test set, Rachel is sad more often that the other speakers etc. Overall, considering all of the above, we believe that the performance gain achieved through the usage of speaker-specific prompts may be based less on learning different emotion distributions for different speakers, and more on learning to interpret each utterance differently according to its speaker, with different speakers using different words and ways of expression.

Topic-specific prompts

Using topic-specific prompts, we have experimented on both MELD and IEMOCAP. For MELD, the topic-enriched model with the best performance is the model using 30 task-specific and 10 topic-specific prompt tokens, with the topic-specific prompt being calculated as an ensemble of the different topic-specific prompts, according to the topic clustering weights. However, this model achieves an F1-score, almost the same as the prompt-based model that uses only 40 task-specific prompt tokens and no topic-specific prompt tokens. This means that, for MELD, the addition of topic-specific information through prompts does not seem to be beneficial.

Our results are different in the case of IEMOCAP: Compared to the prompt based model using only 30 task-specifc prompt tokens, the models utilizing 30 task-specific and 10 topic-specific prompt tokens achieve a F1-score larger by 1.19% in the case of the model using hard classification and by 1.18% in the case of the model using prompt ensembles. Compared to the 40 task-specific prompt token model, this difference is even larger. Topic-specific information thus seems to be important for IEMOCAP, and is successfully utilized through topic-specific prompts.

To understand why the usage of topic-specific prompts yields such different results for MELD and IEMOCAP, we provide an overview of the distribution of emotions per topic for the MELD and IEMOCAP train and test data, in Tables 6.15, 6.16. 6.17 and 6.18. From Tables 6.15 6.16 we can see that the different topics in MELD have a similar distribution of emotions, partly due to the imbalance of our dataset. This largely decreases the emotional clues that could potentially be extracted when performing topic clustering. Observing Tables 6.17 6.18 we can see that in IEMOCAP the differences in the distribution of emotions between the different topics are bigger and occur more often than in MELD, so the knowledge of topic may carry a bigger significance for identifying emotion.

We can thus conclude that the bigger differences in the distribution of emotions between the different topics in IEMOCAP, rather than in MELD, could be one factor explaining the difference in the performance of topic-specific prompts in the two datasets. A second factor could be that topic-modeling in IEMOCAP encodes more meaning, with different topics carrying certain different language patterns that affect the utterances meaning and way of conveying emotion, while topics in MELD are more similar or generic and thus poor on information. To support this claim, we present the 20 most common words for each of the topics extracted from the train set, for MELD and for IEMOCAP:

In the case of MELD:

- Topic 1:
  "place", "need", "guy", "took", "bob", "umm", "listen", "morning", "big", "make", "monica", "ok", "said", "time", "rachel", "little", "thank", "whoa", "chandler", "yes"
- Topic 2:
  "thinking", "ah", "ow", "believe", "say", "sure", "happy", "time", "actually", "went", "day", "date", "place", "wanna", "guy", "people", "bye", "little", "huh", "monica"
- Topic 3:
  "iss", "little", "phoebe", "actually", "quit", "whoa", "listen", "maybe", "umm", "ac", "ok", "guy", "stupid", "make", "night", "time", "wait", "ee", "bs", "ph"
- Topic 4:
  "people", "stuff", "chandler", "remember", "stop", "love", "fine", "need", "talk", "wow", "sure", "hello", "man", "thing", "monica", "yes", "thought", "dr", "oo", "umm"
- Topic 5:
  "man", "stop", "huh", "chandler", "gotta", "somebody", "money", "minutes", "coming", "du", "work", "umm", "thank", "say", "janice", "time", "feel", "thing", "love", "wanna"
- Topic 6:
  "play", "people", "thank", "wow", "ben", "knows", "man", "maybe", "say", "umm", "believe", "phoebe", "ok", "chandler", "guy", "love", "ha", "little", "yes", "ah"
- Topic 7:
  "ross", "hours", "thing", "guy", "wait", "ra", "ch", "big", "thank", "rachel", "gotta", "huh", "knew", "la", "time", "love", "ok", "phoebe", "say", "um"

In the case of IEMOCAP:

- Topic 1:
  "child", "man", "tell", "thing", "lot", "drunk", "point", "okay", "people", "time", "flashlight", "snap", "job", "yes", "really", "think", "good", "got", "little", "yeah"
- Topic 2:
  "phone", "yeah", "new", "tell", "line", "let", "thing", "help", "id", "look", "work", "yes", "got", "maybe", "uh", "sir", "sorry", "need", "um", "okay"
- Topic 3:
  "kind", "larry", "care", "ask", "good", "got", "god", "long", "girl", "father", "laughter", "years", "annie", "marry", "really", "time", "yes", "business", "think", "yeah"
- Topic 4:
  "think", "years", "supposed", "let", "champagne", "thing", "guess", "feel", "ask", "thought", "remember", "things", "come", "got", "okay", "time", "look", "yeah", "said", "fish"

We observe that in the case of MELD, a lot of the most common words for each topic are words used in oral everyday speech ("uhh", "ah", "huh" etc.), while there are also a lot of common words between the different topics. This could mean that, because of the origin of MELD, which is everyday conversations from a TV series, the dialogues are not as sophisticated and do not always have an apparent topic or purpose and therefore topic modeling does not provide useful information. IEMOCAP, on the other hand, having been explicitly designed for the emotion recognition task, could include text more suitable for topic classification, with richer topic-specific information.

Table 6.15. *Emotion distribution per topic for MELD dataset, for train data.*

Topic	Joy	Surprise	Fear	Anger	Disgust	Sadness	Neutral
1	0.169	0.121	0.025	0.117	0.022	0.056	0.490
2	0.168	0.115	0.023	0.149	0.032	0.066	0.447
3	0.192	0.108	0.031	0.110	0.029	0.084	0.446
4	0.143	0.123	0.035	0.102	0.027	0.070	0.498
5	0.178	0.114	0.023	0.109	0.026	0.079	0.470
6	0.195	0.138	0.028	0.082	0.036	0.056	0.466
7	0.179	0.120	0.021	0.112	0.016	0.074	0.479

Table 6.16. *Emotion distribution per topic for MELD dataset, for test data.*

Topic	Joy	Surprise	Fear	Anger	Disgust	Sadness	Neutral
1	0.194	0.103	0.043	0.100	0.026	0.057	0.479
2	0.154	0.123	0.045	0.123	0.031	0.062	0.462
3	0.190	0.130	0.030	0.083	0.035	0.073	0.459
4	0.202	0.122	0.025	0.082	0.032	0.038	0.500
5	0.118	0.131	0.032	0.163	0.036	0.072	0.448
6	0.168	0.106	0.033	0.121	0.035	0.067	0.470
7	0.189	0.158	0.031	0.119	0.025	0.033	0.444

Table 6.17. *Emotion distribution per topic for IEMOCAP dataset, for train data.*

Topic	Anger	Excitement	Frustration	Happiness	Neutral	Sadness
1	0.192	0.220	0.177	0.048	0.231	0.132
2	0.196	0.007	0.329	0.030	0.315	0.123
3	0.089	0.216	0.180	0.199	0.217	0.099
4	0.048	0.063	0.220	0.153	0.221	0.296

Table 6.18. *Emotion distribution per topic for IEMOCAP dataset, for test data.*

Topic	Anger	Excitement	Frustration	Happiness	Neutral	Sadness
1	0.168	0.005	0.298	0.040	0.366	0.123
2	0.152	0.157	0.165	0.161	0.249	0.116
3	0.118	0.101	0.320	0.024	0.195	0.243
4	0.190	0.120	0.158	0.088	0.301	0.143

Using only information-specific prompts versus using both task-specific and information-specific prompts

Studying the different models employed with speaker- and topic-specific prompts, we can see that, for both MELD and IEMOCAP, using 40 prompt tokens that all change according to the speaker or topic and no task-specific prompt tokens that remain the same for the whole train set, yields inferior performance to that achieved by the models which use both task-specific and speaker-/topic-specific prompts. This could suggest that utilizing the whole dataset, thus more data, allows us to tune our prompt's parameters more efficiently, in order to adapt our pre-trained language model to the ERC task, and the extra information-specific prompt tokens can then be used to additionally encode the extra speaker-/topic-related information, whereas when using only speaker-/topic-specific prompts, the data per speaker or topic is not enough to train all of the prompt parameters efficiently.

Hard classification versus prompt-ensembling

As analyzed in sub-section 6.2.2, in the case of topic, we experiment with choosing one topic-specific prompt for each test utterance, the prompt corresponding to the topic to which the utterance belongs with the biggest probability, and with using the probability of each topic being the topic of the utterance, in order to perform a weighted averaging of the prompt weights of the different topic-specific prompts (prompt-ensembling). Comparing the results for both methods, we observe that prompt ensembling leads to equal performance for IEMOCAP and superior performance for MELD, although only by 0.19%, leading us to the conclusion that prompt-ensembling can be beneficial, but does not largely affect performance in the current setting.

Chapter 7

Conclusions

7.1 Discussion

In this work, we explore the potential of prompt-based learning for the adaptation of a pre-trained language model to the task of Emotion Recognition in Conversation (ERC). Prompt-based learning has been proposed as an alternative to fine-tuning in the recent years, with the purpose of reducing the training and storage resources needed for the very large pre-trained language models (which consist of millions or billions of parameters) and has given rise to an increasing amount of related research. There is still however limited work on the utilization of prompt-based learning in specific tasks. We follow two approaches, performing extensive experiments on two popular datasets in the field of ERC, namely IEMOCAP and MELD. To the best of our knowledge, at the time of this thesis, there are no previous works in the field of ERC following a purely prompt-based learning approach, with no tuning of the language model's pre-trained parameters.

In our first approach, we aim to to study the applicability of prompt-based learning in comparison to fine-tuning for the task of ERC and set a baseline for prompt-based learning for Emotion Recognition in Conversation. For this purpose, we create a model which is based on a frozen pre-trained language model and uses a trainable soft prompt, consisting of prompt tokens encoded and optimized directly in the embedding space of the model. Using different prompt sizes, we experiment with a simple model, whose only input is the dataset's utterances. We also experiment with commonly used methods in earlier work in ERC that aim to improve model performance by encoding speaker-specific information, such as the prepending of speaker identity to the input utterances as well as a multitask training with an auxiliary task of speaker-identification. We compare our models with similar models that use fine-tuning instead of prompt-based learning, as well as models with only a trainable output module. We conclude that prompt-based learning can indeed be an effective method to adapt a pre-trained language model to the task of ERC, even in cases where the input format is very different from that of the pre-training phase, such as the case where the speaker identity is prepended to the utterances. A factor affecting performance seems to be the prompt size, with prompts of size neither too big nor too small leading to a better performance. We see that neither prompt-based learning nor fine-tuning is always preferable, with each method's performance depending on the dataset and specific model architecture or input format. In addition, we can confirm that prompts do help the model predict emotion correctly, with our prompt-based models prevailing over the models with only a trainable output module. Finally, regarding the usage of multitask learning with the additional speaker-identification task, we conclude that it does not seem to benefit our models neither in the case of fine-tuning nor in the case of prompt-based learning, with the latter leading to substantially worse results.

Our second approach aims at integrating additional information, useful to the task of ERC, directly through prompts, which we call information-specific, and not through changes in the input format or pre-trained model architecture. We create again a prompt-based model based on a frozen pre-trained language model, which uses trainable soft prompts, but this time we use two prompts: The first is task-specific, aimed at adapting our model to the ERC task and trained on the whole dataset, while the second is information specific and changes according to the additional information for each current utterance. We experiment with topic-specific and speaker-specific prompts each trained for a different speaker or topic. A benefit of our approach is that it is information-agnostic, in the sense that it can be used with different kinds of information that may be deemed useful, by directly extending the logic presented in the current work for speaker and topic information. Through our experiments we conclude that prompts can indeed encode additional, useful to a task information, improving model performance. We can also infer that the combined use of a task-specific and an information-specific prompt is beneficial, as including only the latter leads to a drop in performance. We attribute this to the fact that task-specific prompts, trained on the whole dataset and thus having "seen" more training data, are important for the effective adaptation of our model to the ERC task.

Overall, we conclude that there is potential in using prompt-based learning for the task of Emotion Recogntion in Conversation, especially considering the fact that it is more leightweight than fine-tuning. We therefore believe prompt-based learning for ERC to be a field worth being further researched, in order to determine the optimal architecture and pre-trained model that will allow prompt-based models to reach a state of the art performance in the task.

7.2 Future work

With the end of this thesis, we wish to provide an overview of some potential directions that we believe to be worth exploring, regarding prompt-based learning for Emotion Recognition in Conversation. We would like to suggest the following aspects:

- **Combination of different information-specific prompts**: In our work, we use only a speaker-specific or only a topic-specific prompt, together with a task-specific prompt. A different approach would be to combine both the speaker-specific and the topic-specific prompts. The optimal way to combine the prompts (through weighted sum, concatenation, etc.) could also be investigated.

- **Sound-based speaker-clustering**: In our work we train prompts for some pre-determined speakers that are present in both the training and the test set and group the rest of the speakers into only one speaker-group. We then use this pre-determined grouping in order to determine the speaker-specific prompt to be used in test-time. However, this means that we cannot handle unseen speakers in the test set effectively. An interesting approach would be to cluster unseen speakers to one of the speakers or speaker groups for which we have trained a prompt, based on the audio that corresponds to the speaker's utterance. We can utilize this audio to extract speaker-specific embeddings and then use these embeddings to perform the desired clustering.

- **Combination of prompt-based learning and fine-tuning**: While we have followed the approach of keeping the pre-trained part of our model frozen in order to keep our method lightweight, the usage of a prompt and the tuning of the pre-trained model's parameters combined is often used in some tasks, and found to improve performance. The potential of the combination of the two adaptation methods with the purpose of achieving optimal performance could thus be attempted.

- **Transfer learning for prompts**: Previous work in the field of prompt-based learning has used transfer learning for prompt initialization, reporting beneficial results [36]. An approach could thus be to train our model on a related task, and then use the trained prompt as the initial task-specific prompt embeddings for the task of ERC. The speaker-specific prompts could also be pre-trained in a speaker-related task such as speaker-identification.

- **Experimentation with different pre-trained language models**: While we have worked with BERT, many approaches in ERC work with RoBERTa, BART, or other language models. To obtain a better insight of the effect of prompt-based learning for the adaptation of pre-trained language models for ERC, a variety of language models, with different architectures and sizes could replace BERT in our work and more experiments could be performed.

Bibliography

[1] Nitish Srivastava, Geoffrey Hinton, Alex Krizhevsky, Ilya Sutskever και Ruslan Salakhutdinov. *Dropout: A Simple Way to Prevent Neural Networks from Overfitting. Journal of Machine Learning Research*, 15(56):1929–1958, 2014.

[2] John McGonagle, Alonso García και Saruque Mollick. *Feedforward Neural Networks.*

[3] Md. Zahangir Alom, Tarek Taha, Chris Yakopcic, Stefan Westberg, Paheding Sidike, Mst Nasrin, Mahmudul Hasan, Brian Essen, Abdul Awwal και Vijayan Asari. *A State-of-the-Art Survey on Deep Learning Theory and Architectures. Electronics*, 8:292, 2019.

[4] IBM Cloud Education. *What are recurrent neural networks?* `https://www.ibm.com/cloud/learn/recurrent-neural-networks`, 2020.

[5] *Understanding LSTM networks.* `http://colah.github.io/posts/2015-08-Understanding-LSTMs/`, 2015.

[6] Dzmitry Bahdanau, Kyunghyun Cho και Yoshua Bengio. *Neural Machine Translation by Jointly Learning to Align and Translate. CoRR*, abs/1409.0473, 2015.

[7] Ashish Vaswani, Noam Shazeer, Niki Parmar, Jakob Uszkoreit, Llion Jones, Aidan N. Gomez, Lukasz Kaiser και Illia Polosukhin. *Attention Is All You Need. CoRR*, abs/1706.03762, 2017.

[8] Tomas Mikolov, Kai Chen, G.s Corrado και Jeffrey Dean. *Efficient Estimation of Word Representations in Vector Space. Proceedings of Workshop at ICLR*, 2013, 2013.

[9] Yoshua Bengio, Réjean Ducharme, Pascal Vincent και Christian Janvin. *A Neural Probabilistic Language Model. J. Mach. Learn. Res.*, 3(null):1137–1155, 2003.

[10] Neil Houlsby, Andrei Giurgiu, Stanislaw Jastrzebski, Bruna Morrone, Quentin De Laroussilhe, Andrea Gesmundo, Mona Attariyan και Sylvain Gelly. *Parameter-Efficient Transfer Learning for NLP. Proceedings of the 36th International Conference on Machine Learning*Kamalika Chaudhuri και Ruslan Salakhutdinov, επιμελητές, τόμος 97 στο *Proceedings of Machine Learning Research*, σελίδες 2790–2799. PMLR, 2019.

[11] Jingye Li, Donghong Ji, Fei Li, Meishan Zhang και Yijiang Liu. *HiTrans: A Transformer-Based Context-and Speaker-Sensitive Model for Emotion Detection in Conversations.*

[12] Soujanya Poria, Devamanyu Hazarika, Navonil Majumder, Gautam Naik, Erik Cambria και Rada Mihalcea. *MELD: A Multimodal Multi-Party Dataset for Emotion Recognition in Conversations.* 2018.

[13] Taewoon Kim και Piek Vossen. *EmoBERTa: Speaker-Aware Emotion Recognition in Conversation with RoBERTa.* 2021.

[14] Shimin Li, Hang Yan και Xipeng Qiu. *Contrast and Generation Make BART a Good Dialogue Emotion Recognizer.* 2021. AAAI 2022.

[15] Jingye Li, Meishan Zhang, Donghong Ji και Yijiang Liu. *Multi-Task Learning with Auxiliary Speaker Identification for Conversational Emotion Recognition.* 2020.

[16] Lixing Zhu, Gabriele Pergola, Lin Gui, Deyu Zhou και Yulan He. *Topic-Driven and Knowledge-Aware Transformer for Dialogue Emotion Detection.* 2021.

[17] Deepanway Ghosal, Navonil Majumder, Soujanya Poria, Niyati Chhaya και Alexander Gelbukh. *DialogueGCN: A Graph Convolutional Neural Network for Emotion Recognition in Conversation.* 2019.

[18] Jacob Devlin, Ming-Wei Chang, Kenton Lee και Kristina Toutanova. *BERT: Pre-training of Deep Bidirectional Transformers for Language Understanding.* *CoRR*, abs/1810.04805, 2018.

[19] Yinhan Liu, Myle Ott, Naman Goyal, Jingfei Du, Mandar Joshi, Danqi Chen, Omer Levy, Mike Lewis, Luke Zettlemoyer και Veselin Stoyanov. *RoBERTa: A Robustly Optimized BERT Pretraining Approach.* *CoRR*, abs/1907.11692, 2019.

[20] Mike Lewis, Yinhan Liu, Naman Goyal, Marjan Ghazvininejad, Abdelrahman Mohamed, Omer Levy, Veselin Stoyanov και Luke Zettlemoyer. *BART: Denoising Sequence-to-Sequence Pre-training for Natural Language Generation, Translation, and Comprehension.* *CoRR*, abs/1910.13461, 2019.

[21] Xiang Lisa Li και Percy Liang. *Prefix-Tuning: Optimizing Continuous Prompts for Generation.* 2021.

[22] Antonio Valerio, Miceli Barone, Barry Haddow, Ulrich Germann και Rico Sennrich. *Regularization techniques for fine-tuning in neural machine translation.*

[23] Daniel Jurafsky και James H. Martin. *Speech and Language Processing: An Introduction to Natural Language Processing, Computational Linguistics, and Speech Recognition.* Prentice-Hall, 2000.

[24] Tom Brown, Benjamin Mann, Nick Ryder, Melanie Subbiah, Jared D Kaplan, Prafulla Dhariwal, Arvind Neelakantan, Pranav Shyam, Girish Sastry, Amanda Askell, Sandhini Agarwal, Ariel Herbert-Voss, Gretchen Krueger, Tom Henighan, Rewon Child, Aditya Ramesh, Daniel Ziegler, Jeffrey Wu, Clemens Winter, Chris Hesse, Mark Chen, Eric Sigler, Mateusz Litwin, Scott Gray, Benjamin Chess, Jack Clark, Christopher Berner, Sam McCandlish, Alec Radford, Ilya Sutskever και Dario Amodei. *Language Models are Few-Shot Learners.* *Advances in Neural Information Processing Systems*H. Larochelle, M. Ranzato, R. Hadsell, M.F. Balcan και H. Lin, επιμελητές, τόμος 33, σελίδες 1877–1901. Curran Associates, Inc., 2020.

[25] Mengjie Zhao, Tao Lin, Martin Jaggi και Hinrich Schütze. *Masking as an Efficient Alternative to Finetuning for Pretrained Language Models.* *CoRR*, abs/2004.12406, 2020.

[26] Evani Radiya-Dixit και Xin Wang. *How fine can fine-tuning be? Learning efficient language models.* *Proceedings of the Twenty Third International Conference on Artificial Intelligence and Statistics*Silvia Chiappa και Roberto Calandra, επιμελητές, τόμος 108 στο *Proceedings of Machine Learning Research*, σελίδες 2435–2443. PMLR, 2020.

[27] Sylvestre-Alvise Rebuffi, Hakan Bilen και Andrea Vedaldi. *Learning multiple visual domains with residual adapters.* *CoRR*, abs/1705.08045, 2017.

[28] Kevin Clark, Minh-Thang Luong, Quoc V. Le και Christopher D. Manning. *ELECTRA: Pre-training Text Encoders as Discriminators Rather Than Generators.* *CoRR*, abs/2003.10555, 2020.

[29] Zhilin Yang, Zihang Dai, Yiming Yang, Jaime G. Carbonell, Ruslan Salakhutdinov και Quoc V. Le. *XLNet: Generalized Autoregressive Pretraining for Language Understanding. CoRR*, abs/1906.08237, 2019.

[30] Pengfei Liu, Weizhe Yuan, Jinlan Fu, Zhengbao Jiang, Hiroaki Hayashi και Graham Neubig. *Pre-train, Prompt, and Predict: A Systematic Survey of Prompting Methods in Natural Language Processing.* 2021.

[31] Brian Lester, Rami Al-Rfou και Noah Constant. *The Power of Scale for Parameter-Efficient Prompt Tuning.* 2021.

[32] Jinming Zhao, Ruichen Li, Qin Jin, Xinchao Wang και Haizhou Li. *MEmoBERT: Pre-training Model with Prompt-based Learning for Multimodal Emotion Recognition.* 2021.

[33] Zhengbao Jiang, Frank F. Xu, Jun Araki και Graham Neubig. *How Can We Know What Language Models Know? Transactions of the Association for Computational Linguistics*, 8:423–438, 2020.

[34] Anonymous Author. *Unified Multi-modal Pre-training for Few-shot Sentiment Analysis with Prompt-based Learning*, 2022.

[35] Xiao Liu, Yanan Zheng, Zhengxiao Du, Ming Ding, Yujie Qian, Zhilin Yang και Jie Tang. *GPT Understands, Too.* 2021.

[36] Tu Vu, Brian Lester, Noah Constant, Rami Al-Rfou, Daniel Cer και Google Research. *SPoT: Better Frozen Model Adaptation through Soft Prompt Transfer.*

[37] Xiao Liu, Kaixuan Ji, Yicheng Fu, Weng Lam Tam, Zhengxiao Du, Zhilin Yang και Jie Tang. *P-Tuning v2: Prompt Tuning Can Be Comparable to Fine-tuning Universally Across Scales and Tasks.* 2021.

[38] Ido Dagan, Oren Glickman και Bernardo Magnini. *The PASCAL Recognising Textual Entailment Challenge. MLCW*, 2005.

[39] Navonil Majumder, Soujanya Poria, Devamanyu Hazarika, Rada Mihalcea, Alexander Gelbukh και Erik Cambria. *DialogueRNN: An Attentive RNN for Emotion Detection in Conversations.* 2018.

[40] Jiangnan Li, Zheng Lin, Peng Fu και Weiping Wang. *Past, Present, and Future: Conversational Emotion Recognition through Structural Modeling of Psychological Knowledge.*

[41] Haidong Zhang και Yekun Chai. *COIN: Conversational Interactive Networks for Emotion Recognition in Conversation*, 2021.

[42] Jeffrey Pennington, Richard Socher και Christopher D Manning. *GloVe: Global Vectors for Word Representation.*

[43] Weizhou Shen, Junqing Chen, Xiaojun Quan και Zhixian Xie. *DialogXL: All-in-One XLNet for Multi-Party Conversation Emotion Recognition*, 2021.

[44] Iulian Vlad Serban, Alessandro Sordoni, Ryan Lowe, Laurent Charlin, Joelle Pineau, Aaron Courville και Yoshua Bengio. *A Hierarchical Latent Variable Encoder-Decoder Model for Generating Dialogues.* 2016.

[45] David M. Blei, Andrew Y. Ng και Michael I. Jordan. *Latent Dirichlet Allocation. J. Mach. Learn. Res.*, 3(null):993–1022, 2003.

[46] Mimi Dutta. *Topic Modelling With LDA -A Hands-on Introduction.* 7.

[47] Ian J. Goodfellow, Yoshua Bengio και Aaron Courville. *Deep Learning.* MIT Press, Cambridge, MA, USA, 2016. `http://www.deeplearningbook.org`.

[48] Olivier Chapelle, Bernhard Schölkopf και Alexander Zien. *Semi-Supervised Learning (Adaptive Computation and Machine Learning).* The MIT Press, 2006.

[49] Christopher M. Bishop. *Pattern Recognition and Machine Learning (Information Science and Statistics).* Springer-Verlag, Berlin, Heidelberg, 2006.

[50] Sergios Theodoridis και Konstantinos Koutroumbas. *Pattern Recognition, Fourth Edition.* Academic Press, Inc., USA, 4thη έκδοση, 2008.

[51] Richard O. Duda, Peter E. Hart και David G. Stork. *Pattern Classification.* Wiley, New York, 2η έκδοση, 2001.

[52] David E. Rumelhart, Geoffrey E. Hinton και Ronald J. Williams. *Learning Representations by Back-propagating Errors. Nature*, 323(6088):533–536, 1986.

[53] Yann Lecun, Patrick Haffner και Y. Bengio. *Object Recognition with Gradient-Based Learning.* 2000.

[54] David E. Rumelhart, Geoffrey E. Hinton και Ronald J. Williams. *Learning Representations by Back-Propagating Errors*, σελίδα 696–699. MIT Press, Cambridge, MA, USA, 1988.

[55] J J Hopfield. *Neural networks and physical systems with emergent collective computational abilities. Proceedings of the National Academy of Sciences*, 79(8):2554–2558, 1982.

[56] Sepp Hochreiter και Jürgen Schmidhuber. *Long Short-term Memory. Neural computation*, 9:1735–80, 1997.

[57] Minh-Thang Luong, Hieu Pham και Christopher D. Manning. *Effective Approaches to Attention-based Neural Machine Translation. CoRR*, abs/1508.04025, 2015.

[58] Sethunya Joseph, Kutlwano Sedimo, Freeson Kaniwa, Hlomani Hlomani και Keletso Letsholo. *Natural Language Processing: A Review. Natural Language Processing: A Review*, 6:207–210, 2016.

[59] Zhiyuan Liu, Yankai Lin και Maosong Sun. *Representation Learning for Natural Language Processing. CoRR*, abs/2102.03732, 2021.

[60] John Hancock και Taghi Khoshgoftaar. *Survey on categorical data for neural networks. Journal of Big Data*, 7, 2020.

[61] Usman Naseem, Imran Razzak, Shah Khalid Khan και Mukesh Prasad. *A Comprehensive Survey on Word Representation Models: From Classical to State-Of-The-Art Word Representation Language Models. CoRR*, abs/2010.15036, 2020.

[62] H. P. Luhn. *A Statistical Approach to Mechanized Encoding and Searching of Literary Information. IBM Journal of Research and Development*, 1(4):309–317, 1957.

[63] K. Jones. *A Statistical Interpretation of Term Specificity in Retrieval. Journal of Documentation*, 60:493–502, 2004.

[64] Lorenzo Ferrone και Fabio Massimo Zanzotto. *Symbolic, Distributed and Distributional Representations for Natural Language Processing in the Era of Deep Learning: a Survey. CoRR*, abs/1702.00764, 2017.

[65] Li Lucy και Jon Gauthier. *Are distributional representations ready for the real world? Evaluating word vectors for grounded perceptual meaning. CoRR*, abs/1705.11168, 2017.

[66] Yoshua Bengio, Réjean Ducharme και Pascal Vincent. *A Neural Probabilistic Language Model. Advances in Neural Information Processing Systems*T. Leen, T. Dietterich και V. Tresp, επιμελητές, τόμος 13. MIT Press, 2000.

[67] Edgar Altszyler, Mariano Sigman και Diego Fernández Slezak. *Comparative study of LSA vs Word2vec embeddings in small corpora: a case study in dreams database. CoRR*, abs/1610.01520, 2016.

[68] Zhiyuan Liu, Yankai Lin και Maosong Sun. *Representation Learning for Natural Language Processing.* 2020.

[69] Marwa Naili, Anja Habacha και Henda Ben Ghezala. *Comparative study of word embedding methods in topic segmentation. Procedia Computer Science*, 112:340–349, 2017.

[70] Jeffrey Pennington. *GloVe: Global Vectors for Word Representation*, 2014.

[71] Matthew Peters, Mark Neumann, Mohit Iyyer, Matt Gardner, Christopher Clark, Kenton Lee και Luke Zettlemoyer. *Deep contextualized word representations.* 2018.

[72] Yoav Goldberg και Graeme Hirst. *Neural Network Methods in Natural Language Processing.* Morgan amp; Claypool Publishers, 2017.

[73] Y. Bengio. *Neural net language models. Scholarpedia*, 3(1):3881, 2008. revision #140963.

[74] Wei Xu και Alex Rudnicky. *Can artificial neural networks learn language models? INTERSPEECH*, 2000.

[75] Kun Jing και Jungang Xu. *A Survey on Neural Network Language Models. CoRR*, abs/1906.03591, 2019.

[76] Tomas Mikolov, Martin Karafiát, Lukas Burget, Jan Cernocký και Sanjeev Khudanpur. *Recurrent neural network based language model.* τόμος 2, σελίδες 1045–1048, 2010.

[77] Tomas Mikolov, Stefan Kombrink, Lukas Burget, J.H. Cernocky και Sanjeev Khudanpur. *Extensions of recurrent neural network language model.* σελίδες 5528 – 5531, 2011.

[78] Xipeng Qiu, Tianxiang Sun, Yige Xu, Yunfan Shao, Ning Dai και Xuanjing Huang. *Pre-trained Models for Natural Language Processing: A Survey. CoRR*, abs/2003.08271, 2020.

[79] Alec Radford και Karthik Narasimhan. *Improving Language Understanding by Generative Pre-Training.* 2018.

[80] Alec Radford, Jeff Wu, Rewon Child, David Luan, Dario Amodei και Ilya Sutskever. *Language Models are Unsupervised Multitask Learners.* 2019.

[81] Yonghui Wu, Mike Schuster, Zhifeng Chen, Quoc V. Le, Mohammad Norouzi, Wolfgang Macherey, Maxim Krikun, Yuan Cao, Qin Gao, Klaus Macherey, Jeff Klingner, Apurva Shah, Melvin Johnson, Xiaobing Liu, Lukasz Kaiser, Stephan Gouws, Yoshikiyo Kato, Taku Kudo, Hideto Kazawa, Keith Stevens, George Kurian, Nishant Patil, Wei Wang, Cliff Young, Jason

Smith, Jason Riesa, Alex Rudnick, Oriol Vinyals, Greg Corrado, Macduff Hughes και Jeffrey Dean. *Google's Neural Machine Translation System: Bridging the Gap between Human and Machine Translation. CoRR*, abs/1609.08144, 2016.

[82] Jonas Pfeiffer, Andreas Rücklé, Clifton Poth, Aishwarya Kamath, Ivan Vulic, Sebastian Ruder, Kyunghyun Cho και Iryna Gurevych. *AdapterHub: A Framework for Adapting Transformers. CoRR*, abs/2007.07779, 2020.

[83] Jonas Pfeiffer, Aishwarya Kamath, Andreas Rücklé, Kyunghyun Cho και Iryna Gurevych. *AdapterFusion: Non-Destructive Task Composition for Transfer Learning. CoRR*, abs/2005.00247, 2020.

[84] Xu Han, Zhengyan Zhang, Ning Ding, Yuxian Gu, Xiao Liu, Yuqi Huo, Jiezhong Qiu, Yuan Yao, Ao Zhang, Liang Zhang, Wentao Han, Minlie Huang, Qin Jin, Yanyan Lan, Yang Liu, Zhiyuan Liu, Zhiwu Lu, Xipeng Qiu, Ruihua Song, Jie Tang, Ji Rong Wen, Jinhui Yuan, Wayne Xin Zhao και Jun Zhu. *Pre-Trained Models: Past, Present and Future.* 2021.

[85] Colin Raffel, Noam Shazeer, Adam Roberts, Katherine Lee, Sharan Narang, Michael Matena, Yanqi Zhou, Wei Li και Peter J. Liu. *Exploring the Limits of Transfer Learning with a Unified Text-to-Text Transformer. CoRR*, abs/1910.10683, 2019.

[86] Tom B. Brown, Benjamin Mann, Nick Ryder, Melanie Subbiah, Jared Kaplan, Prafulla Dhariwal, Arvind Neelakantan, Pranav Shyam, Girish Sastry, Amanda Askell, Sandhini Agarwal, Ariel Herbert-Voss, Gretchen Krueger, Tom Henighan, Rewon Child, Aditya Ramesh, Daniel M. Ziegler, Jeffrey Wu, Clemens Winter, Christopher Hesse, Mark Chen, Eric Sigler, Mateusz Litwin, Scott Gray, Benjamin Chess, Jack Clark, Christopher Berner, Sam McCandlish, Alec Radford, Ilya Sutskever και Dario Amodei. *Language Models are Few-Shot Learners.* 2020.

[87] Xu Han, Weilin Zhao, Ning Ding, Zhiyuan Liu και Maosong Sun. *PTR: Prompt Tuning with Rules for Text Classification. CoRR*, abs/2105.11259, 2021.

[88] Eyal Ben-David, Nadav Oved και Roi Reichart. *PADA: A Prompt-based Autoregressive Approach for Adaptation to Unseen Domains. CoRR*, abs/2102.12206, 2021.

[89] Yuzhao Mao, Guang Liu, Xiaojie Wang, Weiguo Gao, Xuan Li και Ping An. *DialogueTRM: Exploring Multi-Modal Emotion Dynamics in Conversations*, 2021.

[90] Yan Wang, Jiayu Zhang, Jun Ma, Shaojun Wang και Jing Xiao. *Contextualized Emotion Recognition in Conversation as Sequence Tagging*, 2020.

[91] Dong Zhang, Liangqing Wu, Changlong Sun, Shoushan Li, Qiaoming Zhu και Guodong Zhou. *Modeling both Context- and Speaker-Sensitive Dependence for Emotion Detection in Multi-speaker Conversations*, 2019.

[92] Jingwen Hu, Yuchen Liu, Jinming Zhao και Qin Jin. *MMGCN: Multimodal Fusion via Deep Graph Convolution Network for Emotion Recognition in Conversation.*

[93] Carlos Busso, Murtaza Bulut, Chi Chun Lee, Abe Kazemzadeh, Emily Mower, Samuel Kim, Jeannette N Chang, Sungbok Lee και Shrikanth S Narayanan. *IEMOCAP: Interactive emotional dyadic motion capture database*, 2007.

[94] Alex Kendall, Yarin Gal και Roberto Cipolla. *Multi-Task Learning Using Uncertainty to Weigh Losses for Scene Geometry and Semantics.* 2017.

[95] David M Blei και John D Lafferty. *Topic models. Text mining*, σελίδες 101–124. Chapman and Hall/CRC, 2009.

[96] Matthew Hoffman, Francis Bach και David Blei. *Online Learning for Latent Dirichlet Allocation. Advances in Neural Information Processing Systems*J. Lafferty, C. Williams, J. Shawe-Taylor, R. Zemel και A. Culotta, επιμελητές, τόμος 23. Curran Associates, Inc., 2010.

List of Abbreviations

AI	Artifical Intelligence
BERT	Bidirectional Encoder Representations from Transformers
CNN	Convolutional Neural Network
DL	Deep Learning
DTM	Document Term Matrix
ERC	Emotion Recognition in Conversation
FFNN	Feed-Forward Neural Network
FFNNLM	Feed-Forward Neural Network Language Model
FN	False Negative
FP	False Positive
GPT	Generative Pre-trained Transformer
KNN	K-Nearest Neighbours
LDA	Latent Dirichlet Allocation
LM	Language Model / Language Modeling
LSTM	Long Short-Term Memory
ML	Machine Learning
MAE	Mean Absolute Error
MLP	Multi-Layer Perceptron
MSE	Mean Squared Error
NLI	Natural Language Inference
NLP	Natural Language Processing
NSP	Next Sentence Prediction
PCA	Principal Component Analysis
ReLU	Rectified Linear Unit
RNN	Recurrent Neural Network
RNNLM	Recurrent Neural Network Language Model
RTE	Recognizing Textual Entailment
TF-IDF	Term Frequency - Inverse Document Frequency
TN	True Negative
TP	True Positive
QA	Question Answering