Εθνικο Μετσοβιο Πολυτεχνειο
Σχολη Ηλεκτρολογων Μηχανικων και Μηχανικων Υπολογιστων
Τομεας Τεχνολογιας Πληροφορικης και Υπολογιστων

Lyrics and Vocal Melody Generation conditioned on Accompaniment

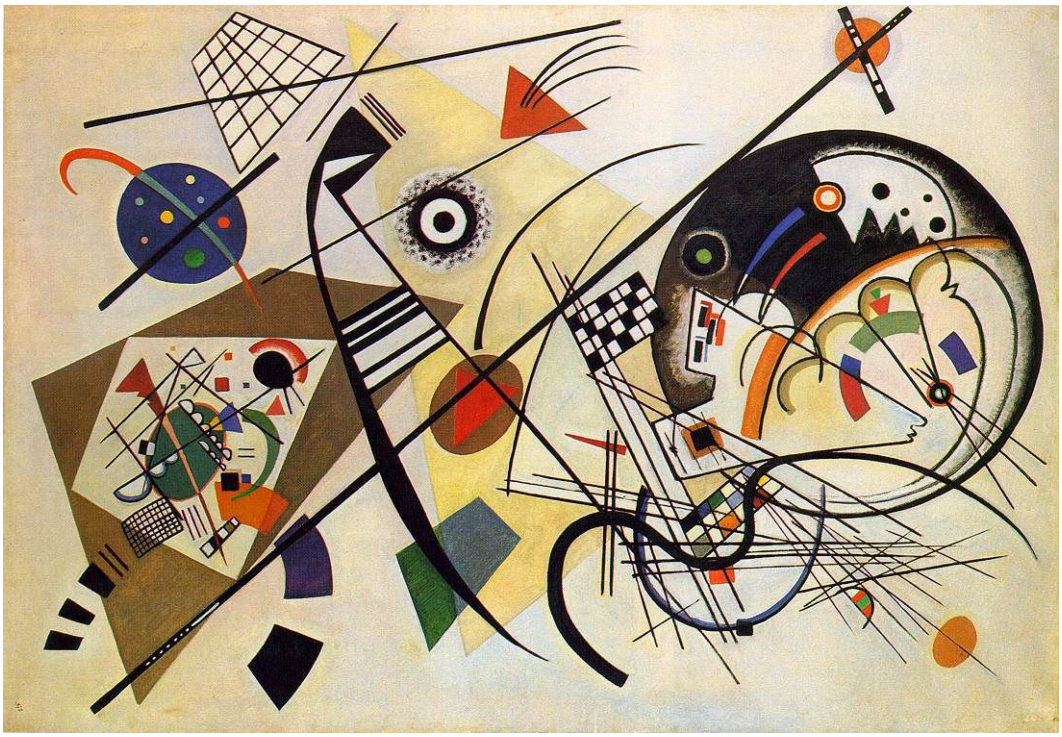
a symbolic music approach
using Deep Learning and Natural Language Processing techniques

---

Διπλωματικη Εργασια

του

ΜΕΛΙΣΤΑ ΘΩΜΑ



Επιβλέποντες: Αλέξανδρος Ποταμιάνος, Αναπληρωτής Καθηγητής
Θεόδωρος Γιαννακόπουλος, Ερευνητής Β, ΕΚΕΦΕ Δημόκριτος

Αθήνα, Μάρτιος 2021

Εθνικο Μετσοβιο Πολυτεχνειο
Σχολη Ηλεκτρολογων Μηχανικων και Μηχανικων Υπολογιστων
Τομεας Τεχνολογιας Πληροφορικης και Υπολογιστων

Lyrics and Vocal Melody Generation conditioned on Accompaniment

a symbolic music approach
using Deep Learning and Natural Language Processing techniques

---

Διπλωματικη Εργασια

του

ΜΕΛΙΣΤΑ ΘΩΜΑ

Επιβλέποντες: Αλέξανδρος Ποταμιάνος, Αναπληρωτής Καθηγητής
Θεόδωρος Γιαννακόπουλος, Ερευνητής Β, ΕΚΕΦΕ Δημόκριτος

Εγκρίθηκε από την τριμελή εξεταστική επιτροπή την 12η Μαρτίου 2021.

(Υπογραφή)	*(Υπογραφή)*	*(Υπογραφή)*
.....	.....	.....
Αλέξανδρος Ποταμιάνος	Γιώργος Στάμου	Θεόδωρος Γιαννακόπουλος
Αναπληρωτής Καθηγητής	Αναπληρωτής Καθηγητής	Ερευνητής Β, ΕΚΕΦΕ Δημόκριτος

Αθήνα, Μάρτιος 2021

Εθνικο Μετσοβιο Πολυτεχνειο
Σχολη Ηλεκτρολογων Μηχανικων και Μηχανικων Υπολογιστων
Τομεας Τεχνολογιας Πληροφορικης και Υπολογιστων

ΔΗΛΩΣΗ ΜΗ ΛΟΓΟΚΛΟΠΗΣ ΚΑΙ ΑΝΑΛΗΨΗΣ ΠΡΟΣΩΠΙΚΗΣ ΕΥΘΥΝΗΣ

Με πλήρη επίγνωση των συνεπειών του νόμου περί πνευματικών δικαιωμάτων, δηλώνω ενυπογράφως ότι είμαι αποκλειστικός συγγραφέας της παρούσας Πτυχιακής Εργασίας, για την ολοκλήρωση της οποίας κάθε βοήθεια είναι πλήρως αναγνωρισμένη και αναφέρεται λεπτομερώς στην εργασία αυτή. Έχω αναφέρει πλήρως και με σαφείς αναφορές, όλες τις πηγές χρήσης δεδομένων, απόψεων, θέσεων και προτάσεων, ιδεών και λεκτικών αναφορών, είτε κατά κυριολεξία είτε βάσει επιστημονικής παράφρασης. Αναλαμβάνω την προσωπική και ατομική ευθύνη ότι σε περίπτωση αποτυχίας στην υλοποίηση των ανωτέρω δηλωθέντων στοιχείων, είμαι υπόλογος έναντι λογοκλοπής, γεγονός που σημαίνει αποτυχία στην Πτυχιακή μου Εργασία και κατά συνέπεια αποτυχία απόκτησης του Τίτλου Σπουδών, πέραν των λοιπών συνεπειών του νόμου περί πνευματικών δικαιωμάτων. Δηλώνω, συνεπώς, ότι αυτή η Πτυχιακή Εργασία προετοιμάστηκε και ολοκληρώθηκε από εμένα προσωπικά και αποκλειστικά και ότι, αναλαμβάνω πλήρως όλες τις συνέπειες του νόμου στην περίπτωση κατά την οποία αποδειχθεί, διαχρονικά, ότι η εργασία αυτή ή τμήμα της δεν μου ανήκει διότι είναι προϊόν λογοκλοπής άλλης πνευματικής ιδιοκτησίας.

(Υπογραφή)

.....
Θωμάς Μελίστας

12 Μαρτίου 2021

Περίληψη

Θέμα της παρούσας διπλωματικής εργασίας είναι η αυτόματη παραγωγή στίχων και φωνητικής μελωδίας βάσει της μουσικής υπόκρουσης. Πρόκειται για ένα ανεξερεύνητο μέχρι στιγμής πρόβλημα. Τα τελευταία χρόνια υπάρχει ένα ολοένα αυξανόμενο ενδιαφέρον για την παραγωγή στίχων με γλωσσικά μοντέλα, λαμβάνοντας υπόψιν τις ιδιαιτερότητες στην δομή και το περιεχόμενο. Παράλληλα, έχει υπάρξει ενδιαφέρον για τη συσχέτιση στίχων και φωνητικής μελωδίας, ενώ έχουν αναπτυχθεί μοντέλα που μπορούν να προβλέπουν φωνητικές μελωδίες βάσει στίχων και το αντίστροφο.

Ενώ η έρευνα σε αυτόν τον τομέα φαίνεται αρκετά υποσχόμενη μέχρι στιγμής, αποτυγχάνει να λάβει υπόψιν της το γενικότερο μουσικό πλαίσιο. Στη σύγχρονη μουσική, το τραγούδι συνυπάρχει μαζί με την ορχηστρική μουσική και έχει δύο βασικές λειτουργίες. Προσθέτει μια μελωδία η οποία ταιριάζει με την μουσική, και ενδύει στιχουργικά ένα κομμάτι λέγοντας μια ιστορία και προκαλώντας συναισθήματα. Επίσης, η προηγούμενη έρευνα αρκείται στο να μελετάει τη συσχέτιση μιας ή μερικών προτάσεων στίχων και δεν αναλύει την δομή των στίχων και της μελωδίας σε ολόκληρα κομμάτια και τα μοτίβα που προκύπτουν.

Η παρούσα εργασία μοντελοποιεί το παραπάνω ως ένα πρόβλημα μετάφρασης ουσιαστικά από μία ακολουθία (μουσική) σε μια άλλη (φωνητικά), χρησιμοποιώντας για πρώτη φορά μοντέλα με μηχανισμούς προσοχής γραμμικής πολυπλοκότητας, τα οποία έχουν εκπαιδευτεί σε αναπαραστάσεις συμβολικής μουσικής. Χρησιμοποιούμε μουσικοθεωρητική ανάλυση για να μικρύνουμε το μέγεθος των ακολουθιών των δεδομένων μας και να τα κάνουμε ανεξάρτητα της μουσικής κλίμακας στην οποία είναι γραμμένα, πετυχαίνοντας έτσι πιο γρήγορη εκπαίδευση και πιο εύρωστα μοντέλα. Επίσης, δημιουργούμε και εφαρμόζουμε μια νέα αρχιτεκτονική για να χωρίσουμε την παραγωγή των στίχων και της μελωδίας, προσφέροντας τη δυνατότητα να χρησιμοποιηθεί κάποιο προεκπαιδευμένο γλωσσικό μοντέλο, αλλά και να χρησιμοποιηθούν δοσμένοι στίχοι, προαιρετικά. Τέλος, χρησιμοποιούμε ένα μοντέλο σύνθεσης φωνής για να διεξάγουμε ποιοτική αξιολόγηση των αποτελεσμάτων.

Από όσο γνωρίζουμε, αυτή είναι η πρώτη απόπειρα να μελετηθεί ταυτόχρονα στιχουργικά και μελωδικά η συσχέτιση τραγουδιού και μουσικής. Ο κώδικας και τα εκπαιδευμένα μοντέλα αυτής της εργασίας μιμούνται τη διαδικασία που ακολουθεί ένας τραγουδιστής/τραγουδοποιός για να προσθέσει φωνητικά σε ένα κομμάτι και πιστεύουμε ότι μπορεί να προσφέρει έμπνευση σε καλλιτέχνες και όχι μόνο.

Λέξεις Κλειδιά

δημιουργία στίχων και μουσικής, βαθιά μάθηση, επεξεργασία φυσικής γλώσσας, μετασχηματιστές, αποδοτικοί μηχανισμοί προσοχής, γλωσσικά μοντέλα, μουσική ανάλυση

Abstract

The purpose of this dissertation is to study the generation of lyrics and vocal melody for a given instrumental music piece. It is a novel, previously unexplored task. During the last few years, there has been increasing research interest over lyrics generation as a case of language modelling with domain specific structure and attributes, as well as regarding symbolic music generation. The correlation of lyrics and corresponding vocal melody has also recently started gaining attention and a few models that are able to generate lyrics conditioned on melody, and vice versa, have been developed.

While the above research directions are very promising, they fail to capture the general musical context of the songwriting process. In the majority of contemporary music, singing coexists with accompaniment and its function is to both provide a melodic line, that is grounded on the instrumental part and advances it musically, as well as to promote the unfolding of a story through lyrical imagery. Moreover, former research on the matter has followed a proof-of-concept approach, working on the level of one or a few sentences, which is insufficient for capturing the structure and the recurring musical and lyrical themes present in a song.

Our work models lyrics and vocal melody generation for a given music piece as a sequence-to-sequence task, using for the first time an efficient attention Transformer architecture trained on text event sequences, that describe entire songs. We build a symbolic music dataset, suitable for the described task, and we apply music theory analysis, compressing successfully our training data and making them key-independent. As a result, our models become faster to train and more robust. Furthermore, we come up with a novel architecture, that decouples lyric and melody generation, while also providing the ability to use any pretrained language model and optional conditioning on predefined lyrics. Finally, the output is used together with a singing voice synthesis model to create and add vocals to instrumental tracks, which we use for qualitative evaluation.

To the best of our knowledge, this is the first attempt to study both the melodic and lyrical content of singing in relation to the musical context it is found in, and through that, automate the process a singer or songwriter would follow, when presented with an instrumental music piece, in order to enrich it with vocals. We believe that our work can fuel human creativity and provide interesting musical ideas.

Keywords

lyrics and symbolic music generation, deep learning, natural language processing, transformers, efficient attention, language modeling, music analysis

Ευχαριστίες

Θα ήθελα καταρχάς να ευχαριστήσω τον καθηγητή κ. Αλέξανδρο Ποταμιάνο για την επίβλεψη αυτής της διπλωματικής εργασίας και για την ευκαιρία που μου έδωσε να την εκπονήσω στο εργαστήριο Επεξεργασίας Φωνής και Φυσικής Γλώσσας. Οι γνώσεις και οι συμβουλές του επί του αντικειμένου ήταν καθοριστικής σημασίας για την παρούσα εργασία, ενώ οι διαλέξεις του ήταν ένα απο τα βασικότερα ερεθίσματα που με ώθησαν να ασχοληθώ με τη Μηχανική Μάθηση.

Επίσης, θα ήθελα να ευχαριστήσω από καρδιάς και τον συνεπιβλέποντα της διπλωματικής μου εργασίας, κ. Θεόδωρο Γιαννακόπουλο, για την υποδειγματική συνεργασία που είχαμε. Χωρίς την καθοδήγηση, την υποστήριξη και το αμέριστο ενδιαφέρον του, η εκπόνηση αυτής της εργασίας θα ήταν απείρως δυσκολότερη.

Σε αυτό το σημείο, να ευχαριστήσω και τους υποψήφιους διδάκτορες Γιώργο Παρασκευόπουλο και Ευθύμη Γεωργίου για την βοήθεια τους επί ερευνητικών και πρακτικών ζητημάτων καθώς και για τις χρήσιμες συμβουλές τους.

Θα ήταν αδύνατον να είχα ολοκληρώσει την διπλωματική μου εργασία χωρίς την στήριξη κάποιων ανθρώπων, που ήταν εκεί για μένα, ακόμα και σε αυτή τη δύσκολη περίοδο. Δεν θα πω πολλά, αυτοί γνωρίζουν καλύτερα.

Θέλω να ευχαριστήσω την κοπέλα μου για την στήριξη που μου παρείχε όλο αυτό το διάστημα και βοήθησε στο να περάσουν ευχάριστα οι τελευταίοι μήνες. Τους φίλους μου, που μου προσέφεραν τη γνώμη τους και τα σχόλια τους και άκουγαν στοργικά την γκρίνια μου. Τέλος, θα ήθελα να ευχαριστήσω τους γονείς και τα αδέρφια μου.

Αθήνα, Μάρτιος 2021

Θωμάς Μελίστας

Περιεχόμενα

Περίληψη 1

Abstract 3

Ευχαριστίες 5

Πρόλογος 17

0 Εκτεταμένη Ελληνική Περίληψη 19
- 0.1 Εισαγωγή . . . 19
- 0.2 Θεωρητικό Υπόβαθρο . . . 20
- 0.3 Τα δεδομένα μας . . . 21
- 0.4 Αρχιτεκτονικές των Μοντέλων . . . 24
- 0.5 Πειράματα και Αποτελέσματα . . . 27
- 0.6 Συμπεράσματα και Μελλοντικές Κατευθύνσεις . . . 29

1 Introduction 31
- 1.1 Motivation and Originality of our Work . . . 31
- 1.2 Research Objective and Contributions . . . 33
- 1.3 Thesis outline . . . 34

2 Background 35
- 2.1 Introduction to Machine Learning . . . 35
  - 2.1.1 A Short History of Artificial Intelligence . . . 35
  - 2.1.2 Machine Learning . . . 36
  - 2.1.3 Basic Machine Learning Methods . . . 38
- 2.2 Basics of Deep Learning . . . 40
  - 2.2.1 Feed Forward Neural Networks . . . 40
  - 2.2.2 Activation Functions . . . 41
  - 2.2.3 Training . . . 45
- 2.3 Deep Learning for Natural Language Processing . . . 51
  - 2.3.1 Overview of the Field . . . 51
  - 2.3.2 Recurrent Neural Networks . . . 52
  - 2.3.3 Sequence-to-Sequence Modelling . . . 56
  - 2.3.4 The Attention Mechanism . . . 57
  - 2.3.5 Transformer . . . 58

2.4 Overcoming Memory Constraints . . . 62
2.4.1 The Pursuit for Efficient Attention . . . 62
2.4.2 Performer - FAVOR+ Attention . . . 62
2.4.3 Reversible Layers . . . 63
2.5 On Symbolic Music and Vocal Melody Generation . . . 64
2.5.1 Symbolic Music - MIDI . . . 64
2.5.2 Music Theory . . . 65
2.5.3 Symbolic Music Generation . . . 66
2.5.4 Conditional Vocal Melody Generation . . . 67
2.6 On Conditional Lyrics Generation . . . 67
2.7 Singing Voice Synthesis . . . 68

3 Building our Dataset **69**
3.1 The Lakh MIDI Dataset . . . 69
3.2 Shaping the Dataset for our Task . . . 70
3.2.1 Drawbacks of the Existing Dataset . . . 70
3.2.2 Creating a more Standardized Dataset . . . 70
3.2.3 Text Event Format . . . 71
3.3 Applying Music Theory Analysis . . . 73
3.3.1 Chord Reduction . . . 73
3.3.2 Roman Numeral Analysis . . . 75
3.4 Decoupling Lyrics and Melody . . . 77

4 Model Architectures for Lyrics and Vocal Melody Generation **79**
4.1 Sequence to Sequence Modelling . . . 79
4.1.1 Formulation . . . 79
4.1.2 Enhancements . . . 81
4.1.3 Decoding Strategy . . . 82
4.2 Decoupled Modelling - Combining Multiple Input Sequences . . . 83
4.2.1 Formulation . . . 83
4.2.2 Lyrics Language Model . . . 85

5 Experiments and Results **87**
5.1 Experiments . . . 87
5.1.1 Model Hyperparameters . . . 87
5.1.2 Training Hyperparameters . . . 89
5.1.3 Dataset size and total steps . . . 89
5.1.4 Training and Inference Speed . . . 90
5.2 Regarding the Generation Evaluation Metrics . . . 90
5.3 Comparison . . . 91
5.4 Qualitative Evaluation . . . 92

6 Conclusions and Future Work **95**
6.1 Conclusions 95
6.2 Future Work 96

Παραρτήματα **99**

A Training, Validation and Generation Metrics for our three models: Seq2Seq with Full Input, Seq2Seq with Reduced Chords Input, Decoupled with Reduced Chords Input **101**
A.1 Sequence-to-Sequence with Full Instrumental Input 101
A.2 Sequence-to-Sequence with Reduced Chords Instrumental Input 103
A.3 Decoupled with Reduced Chords Instrumental Input 104

B Examples of the Generated Sequences **105**

Βιβλιογραφία **113**

Κατάλογος Σχημάτων

1 Ακολουθία ορχηστρικού μέρους. 22
2 Ακολουθία φωνητικού μέρους. Περιέχει στίχους και μελωδία ταυτόχρονα. . . 22
3 Η απλή sequence-to-sequence αρχιτεκτονική με έναν κωδικοποιητή για το ορχηστρικό (αριστερά) και ένα αποκωδικοποιητή για το φωνητικό μέρος (δεξιά) 24
4 Η διαχωρισμένη αρχιτεκτονική μας με έναν κωδικοποιητή (κέντρο), έναν αποκωδικοποιητή για στίχους (αριστερά) και έναν αποκωδικοποιητή για φωνητική μελωδία (δεξιά) με δύο υπο-επίπεδα cross-attention. Ο κωδικοποιητής ρυθμίζει και τους δύο αποκωδικοποιητές, ενώ οι κωδικοποιήσεις του μοντέλου στίχων ρυθμίζουν τον αποκωδικοποιητή φωνητικής μελωδίας. 26

2.1 An illustration of the Perceptron with mathematical notation (right) and a drawing of a biological neuron (left) to draw analogies and show the researchers' inspiration. (Source: [38]) 38
2.2 An example of Support Vector Machines for classification. The red line is the maximum-margin hyperplane. (Source: Wikipedia) 39
2.3 A 3-layer feed forward neural network with three inputs, two hidden layers of 4 neurons each and one output layer. (Source: [38]) 40
2.4 A sigmoid non-linearity. All inputs are squashed in the range [0,1]. (Source: [38]) 42
2.5 A tanh non-linearity. All inputs are squashed in the range [-1,1]. (Source: [38]) 42
2.6 A ReLU non-linearity. Simply, zero when $z < 0$ and then linear with slope 1 when $z > 0$. (Source: [38]) 43
2.7 A GELU non-linearity. (Source: PyTorch documentation) 44
2.8 The surface of a non-convex loss function. The arrow shows a path to reach the global minimum. (Source: Medium) 47
2.9 Using dropout during a training step. (Source: [7]) 49
2.10 A residual connection skipping two layers. (Source: [30]) 51
2.11 An RNN looping through time (left) and the same network unfolded over time (right). (Source: Wikipedia) 53
2.12 The architecture of an LSTM unit. (Source: Wikipedia) 54
2.13 The architecture of a GRU unit. (Source: Wikipedia) 56
2.14 The encoder-decoder framework. The RNN units are unfolded over time. (Source: Github) 57
2.15 The attention mechanism of the original paper. (Source:[2]) 58

2.16 The Transformer Encoder-Decoder Architecture. (Source: [86]) 59
2.17 Schematics of Scaled Dot-Product Attention (left) and Multi-Head Attention with h attention heads (right). (Source: [86]) 60
2.18 The Performer Architecture - approximation via random features to avoid computation of A (source: [17]) 63
2.19 The forward **(a)** and reverse **(b)** computations on a reversible block (Source: [26]) 64
2.20 The piano roll representation of a MIDI file (Source: songaweek) 65
2.21 The chromatic 12-tone chromatic scale built on C4, with corresponding frequencies. (Source: Wikipedia) 65
2.22 The chord progression vi–ii–V–I in the key of C major, in standard notation $A_m - D_m - G_{maj} - C_{maj}$. (Source: Wikipedia) 66

3.1 Instrumental Text Events 71
3.2 Vocal Text Events (phonemes) corresponding to the lyrics: *ton sold in a marke(-e)t* 72
3.3 An example of two music measures before compressing them to chords and annotating them using roman numeral analysis (illustration with MuseScore) 74
3.4 The same two music measures of Figure 3.3, but with generated chords and their roman numeral representation. Notes are restricted in an octave range (illustration with MuseScore). 75
3.5 Instrumental Text Events with Roman Numeral Chords, Rests, Downbeats and Beats 76
3.6 Vocal Text Events with Roman Numeral Notes 76

4.1 Our Simple Sequence-to-Sequence Architecture with an instrumental encoder (left) and a vocal melody/lyrics decoder (right) 80
4.2 Our Decoupled Architecture with an instrumental encoder (center), one decoder for lyrics (left) and one decoder for vocal melody (right) with two cross-attention sublayers. The encoder conditions both decoders, while the encodings of the lyrics model condition the vocal melody decoder. 84
4.3 The Language Model that we finetune on lyrics, warm-starting with pretrained distilGPT-2 weights 86

5.1 Rhythmical/Musical Quality according to our Qualitative Evaluation Study 93
5.2 Relation to the Music according to our Qualitative Evaluation Study 94
5.3 Lyrical Content according to our Qualitative Evaluation Study 94

A.1 Train(orange) and Validation(blue) Loss for simple seq2seq architecture with full input - training for 6 epochs 101
A.2 BLEU metric for simple seq2seq architecture with full input 102
A.3 Valid Structure Metric of Vocal Melody sequence for simple seq2seq architecture with full input 102

A.4 Train(orange) and Validation(blue) Loss for simple seq2seq architecture with reduced input - training for 6 epochs 103
A.5 BLEU metric for simple seq2seq architecture with reduced input 103
A.6 Valid Structure Metric of Vocal Melody sequence for simple seq2seq architecture with reduced input 104
A.7 Train(orange) and Validation(blue) Loss for decoupled architecture with reduced input (sum of lyrics and melody losses) - training for 6 epochs . . . 104
A.8 Valid Structure Metric of Vocal Melody sequence for decoupled architecture with reduced input 104

B.1 Generated examples of the distilGPT-2 language model finetuned on lyrics 105
B.2 Generated examples of the language model in the decoupled architecture . 106

Κατάλογος Πινάκων

1 Μεγέθη ορχηστρικών και φωνητικών ακολουθιών 23
2 Μεγέθη ορχηστρικής ακολουθίας μετά τη συμπίεση σε ακόρντα 23
3 Μέγεθος συνόλων δεδομένων (90% για εκπαίδευσης), εποχές και συνολικά βήματα εκπαίδευσης και για τα τρία μοντέλα, καθώς και το Γλωσσικό Μοντέλο 27
4 Ταχύτητα εκπαίδευσης, συνολική διάρκεια εκπαίδευσης και ταχύτητα παραγωγής για τα τρία μοντέλα (χρόνοι μετρημένοι σε GPU NVIDIA Tesla T4) . . . 27

2.1 A summary table of popular attention mechanisms in chronological order . 58

3.1 Sequence Lengths for Instrumental and Vocal text event formats 73
3.2 Sequence Lengths and Percent Reduction of Instrumental Events 77

5.1 Model Hyperparameters for Sequence-to-Sequence with Full Instrumental Input, Sequence-to-Sequence with Reduced Chords Instrumental Input and Decoupled with Reduced Chords Instrumental Input. *language model . . 88
5.2 Model Hyperparameters for the Language Model that we finetune on Lyrics, we warm-start (load weights) from a pretrained distilGPT-2 model 88
5.3 Training Hyperparameters for all three Instrumental to Vocals Models (Seq2Seq (full), Seq2Seq (chords), Decoupled (chords)) and the finetuned on Lyrics Language Model 89
5.4 Size of Train and Validation Datasets (90/10 split), Epochs and Total Training Steps for all three Instrumental to Vocals Models (Seq2Seq (full), Seq2Seq (chords), Decoupled (chords)) and the finetuned on Lyrics Language Model 90
5.5 Training speed, total training duration and inference speed for Sequence-to-Sequence with Full Instrumental Input, Sequence-to-Sequence with Reduced Chords Instrumental Input and Decoupled with Reduced Chords Instrumental Input (times measured in an NVIDIA Tesla T4 GPU) 90
5.6 Training speed and total training duration for the Language Model that we finetune on Lyrics 91

Πρόλογος

Η παρούσα διπλωματική εργασία εκπονήθηκε στο εργαστήριο Επεξεργασίας Φωνής και Φυσικής Γλώσσας (NTUA Speech And Language Processing Group) της σχολής Ηλεκτρολόγων Μηχανικών και Μηχανικών Υπολογιστών του Εθνικού Μετσόβιου Πολυτεχνείου στην Αθήνα, κατά το έτος 2020-2021.

Πίνακας εξωφύλλου: Wassily Kandinsky, *Transverse Line*, 1923

Ο Kandinsky όντας συναισθητικός, παρομοίαζε συχνά τη ζωγραφική με τη μουσική, δανειζόμενος όρους όπως αυτοσχεδιασμός ή σύνθεση όταν αναφερόταν στο έργο του και η ίδια του η τέχνη μαρτυρούσε την αντίληψη του αυτή.

Colour is the keyboard, the eyes are the hammers,
the soul is the piano with many strings.
The artist is the hand which plays,
touching one key or another,
to cause vibrations in the soul.

WASSILY KANDINSKY

Κεφάλαιο 0

Εκτεταμένη Ελληνική Περίληψη

0.1 Εισαγωγή

Το αντικείμενο με το οποίο ασχολείται η παρούσα διπλωματική εργασία είναι η αυτόματη παραγωγή στίχων και φωνητικής μελωδίας βάσει της μουσικής υπόκρουσης. Πρόκειται για ένα ανεξερεύνητο μέχρι στιγμής πρόβλημα. Ενώ έχει παρατηρηθεί ενδιαφέρον για τη συσχέτιση στίχων και φωνητικής μελωδίας, η υπάρχουσα έρευνα δεν έχει λάβει υπόψιν της το ορχηστρικό κομμάτι των τραγουδιών. Η μοντέρνα μουσική βασίζεται πολύ στην συνύπαρξη μουσικής και φωνητικών, τα οποία πρέπει να σχετίζονται και να στηρίζονται στη μουσική, τόσο ρυθμικά, όσο και αρμονικά. Ακόμα, οι στίχοι έχουν άμεση σχέση με τη φωνητική μελωδία και το περιεχόμενο τους σχετίζεται με τη μουσική και το είδος της.

Συνεπώς, η συγκεκριμένη εργασία καλύπτει ένα σημαντικό κενό στην σχετική έρευνα. Ταυτόχρονα, το αντικείμενο αυτό, αποτελεί ένα περιβάλλον το οποίο διατίθεται για μελέτη σχετικά με τις δυνατότητες της τεχνητής ή αλλιώς υπολογιστικής δημιουργικότητας, καθώς και για την επίλυση δυσκολιών που προκύπτουν κατά την μοντελοποίηση μεγάλων και σύνθετων ακολουθιών.

Η παρούσα εργασία μοντελοποιεί το παραπάνω ως ένα πρόβλημα μετάφρασης ουσιαστικά, από μία ακολουθία (μουσική) σε μια άλλη (φωνητικά). Σε αντίθεση με προηγούμενες προσπάθειες που αρκούνται στο να μελετούν τη συσχέτιση μιας ή μερικών προτάσεων στίχων, η δουλειά μας αναλύει την δομή των στίχων και της μελωδίας σε ολόκληρα κομμάτια και συνεπώς τα μοτίβα που προκύπτουν. Το παρόν έργο είναι η πρώτη προσπάθεια, από όσο γνωρίζουμε, να παραχθούν τόσο στίχοι όσο και φωνητική μελωδία, για ένα ολόκληρο κομμάτι, χρησιμοποιώντας μια Transformer αρχιτεκτονική και χρησιμοποιώντας προηγούμενη γνώση από γλωσσικά μοντέλα.

Οι σημαντικότερες **συνεισφορές** μας στο σχετικό ερευνητικό πεδίο είναι οι εξής:

- Μελετάμε για πρώτη φορά την παραγωγή στίχων και φωνητικής μελωδίας βάσει της μουσικής υπόκρουσης
- Χρησιμοποιούμε για πρώτη φορά την αρχιτεκτονική Transformer στο πλαίσιο της παραγωγής στίχων ή φωνητικής μελωδίας. Χρησιμοποιούμε έναν μηχανισμό προσοχής γραμμικής πολυπλοκότητας, που μας επιτρέπει να εκπαιδεύουμε τα μοντέλα μας σε ακολουθίες έως και 50 φορές μεγαλύτερες από το συνηθισμένο.

- Με χρήση τεχνικών μουσικής ανάλυσης, συμπιέζουμε τα δεδομένα μας έως και 80% και τα κάνουμε ανεξάρτητα από την κλίμακα στην οποία βρίσκονται, εκπαιδεύοντας τα μοντέλα μας πιο γρήγορα
- Αναπτύσσουμε μια καινούρια ¨διαχωρισμένη¨ αρχιτεκτονική, που μας επιτρέπει να παράξουμε ξεχωριστά στίχους και φωνητική μελωδία, λαμβάνοντας υπόψιν την αλληλεξάρτησή τους. Αυτό μας δίνει τη δυνατότητα να χρησιμοποιήσουμε οποιοδήποτε προεκπαιδευμένο γλωσσικό μοντέλο και προαιρετικά να χρησιμοποιήσουμε δικούς μας στίχους.

Για να ερευνήσουμε τα παραπάνω:

- Δημιουργούμε ένα σύνολο δεδομένων που είναι κατάλληλο για το πρόβλημα, διαχωρίζοντας φωνητικά από μουσική και δημιουργώντας ακολουθίες κειμένου που περιγράφουν το κάθε τραγούδι
- Εκπαιδεύουμε ένα γλωσσικό μοντέλο σε αγγλικούς στίχους και το χρησιμοποιούμε στην αρχιτεκτονική μας για να βελτιώσουμε την ποιότητα των παραγόμενων στίχων
- Αναπτύσσουμε μια στρατηγική αποκωδικοποίησης, ειδικά για το πρόβλημα μας και κάνουμε τη δημιουργία πιο εύρωστη, επιβάλλοντας έγκυρη δομή. Επίσης, σχεδιάζουμε μια μέθοδο αξιολόγησης με βάση τη δομή.
- Χρησιμοποιούμε ένα μοντέλο σύνθεσης φωνής για τη δημιουργία φωνητικών. Στη συνέχεια, το χρησιμοποιούμε, μαζί με την υπόκρουση, για να πραγματοποιήσουμε μια ποιοτική μελέτη αξιολόγησης.

0.2 Θεωρητικό Υπόβαθρο

Ο **Transformer** [86] είναι μια αρχιτεκτονική βαθιάς μηχανικής μάθησης που εισήχθη για να λύσει το πρόβλημα της αυτόματης μετάφρασης. Είναι εμπνευσμένος από την επιτυχία του μηχανισμού προσοχής και είναι σε θέση να επεξεργάζεται δεδομένα παράλληλα. Αυτό τον καθιστά πολύ γρήγορο, ενώ αποδίδει καλύτερα από άλλες αρχιτεκτονικές. Πρόκειται για μια αρχιτεκτονική κωδικοποιητή-αποκωδικοποιητή ή οποίοι αποτελούνται από πολλά στοιβαγμένα ίδια επίπεδα. Κάθε επίπεδο του κωδικοποιητή αποτελείται από υπο-επίπεδα self-attention και feedfoward δικτύων. Ο αποκωδικοποιητής έχει την ίδια δομή, με την προσθήκη ενός cross-attention υπο-επιπέδου. Το υπο-επίπεδο self-attention δημιουργεί μια αναπαράσταση της ακολουθίας με βάση τα συμφραζόμενα, αναλύοντας την εξάρτηση μεταξύ των μερών της. Το υπο-επίπεδο cross-attention είναι υπεύθυνο για την ανάλυση της εξάρτησης μεταξύ των ακολουθιών εισόδου και εξόδου. Το αποτέλεσμα του τελευταίου επιπέδου του αποκωδικοποιητή μετατρέπεται τελικά σε πιθανότητες συμβόλων, χρησιμοποιώντας έναν γραμμικό μετασχηματισμό και μια συνάρτηση softmax.

Ο μηχανισμός προσοχής είναι το βασικότερο κομμάτι και δίνεται από τον εξής τύπο:

$$Attention(Q, K, V) = softmax(\frac{QK^\top}{\sqrt{d_k}})V \tag{1}$$

όπου οι πίνακες K και V αναφέρονται στον κωδικοποιητή ενώ ο πίνακας Q στον αποκωδικοποιητή, στην περίπτωση του cross-attention, ενώ κατά το self-attention οι πίνακες αφορούν το ίδιο μέρος του δικτύου. Στην πραγματικότητα, ο μηχανισμός που χρησιμοποιείται είναι πιο σύνθετος, αλλά δεν θα προχωρήσουμε σε λεπτομέρειες.

Από την εμφάνισή του, ο Transformer έχει φέρει επανάσταση στο ευρύτερο πεδίο της Επεξεργασίας Φυσικής Γλώσσας και έχει γίνει η επιλεγόμενη αρχιτεκτονική για πολλά προβλήματα. Κάποιες παραλλαγές της αρχιτεκτονικής κρατάνε μόνο τον κωδικοποιητή [21] ή τον αποκωδικοποιητή [69], εκπαιδεύοντας τους σε τεράστιο όγκο δεδομένων και χρησιμοποιώντας τους (και την γνώση που αποκομίζουν) ως γλωσσικά μοντέλα για παραγωγή κειμένου ή σε συνδυασμό με επιπλέον επίπεδα για άλλα προβλήματα.

Ένα σημαντικό μειονέκτημα του μηχανισμού προσοχής του Transformer είναι ότι είναι αναποτελεσματικός για μεγάλες ακολουθίες. Η χωρική και χρονική πολυπλοκότητα του είναι τετραγωνική, καθιστώντας ανέφικτο το να χρησιμοποιηθεί για μεγάλες ακολουθίες πρακτικά. Ο μηχανισμός προσοχής **FAVOR+** [17] έρχεται να λύσει αυτό το πρόβλημα και να κάνει την πολυπλοκότητα γραμμική, με μια ικανοποιητική προσέγγιση.

Ένας ακόμα τρόπος να μειώσουμε τη χρήση μνήμης είναι τα αναστρέψιμα επίπεδα (**reversible layers**) [26] με τα οποία είναι απαραίτητο να κρατάμε την έξοδο μόνο του τελευταίου επιπέδου. Αυτό επιτυγχάνεται κρατώντας ένα ζευγάρι εισόδου και εξόδου σε κάθε επίπεδο αντί για μόνο μια τιμή.

0.3 Τα δεδομένα μας

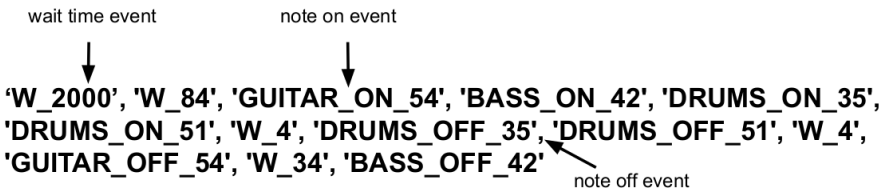
Η βάση του συνόλου δεδομένου μας είναι το Lakh MIDI Dataset (LMD) [70]. Τα δεδομένα αυτά βρίσκονται στην μορφή MIDI, η οποία είναι μια δυαδική μορφή συμβολικής αναπαράστασης μουσικής. Σε αντίθεση με την παρτιτούρα για παράδειγμα που περιέχει μια ιεραρχική δομή, τα αρχεία MIDI αποτελούνται από σύνολα ¨γεγονότων¨ που δηλώνουν σε ποιο χρονικό σημείο ξεκινάει να παίζει (ή σταματάει) μια νότα ή και άλλα μεταδεδομένα όπως οι στίχοι. Από το παραπάνω σύνολο δεδομένων, χρησιμοποιούμε 45, 129 αρχεία τα οποία φέρνουμε σε κατάλληλη μορφή ακολουθώντας την εξής διαδικασία:

- Ανίχνευσης γλώσσας, για τη διατήρηση μόνο αγγλικών στίχων.
- Διατήρηση μόνο συγκεκριμένων χαρακτήρων
- Αντιστοίχιση στίχων στην πλησιέστερη νότα και επιλογή του οργάνου με τις περισσότερες αντιστοιχίσεις
- Φιλτράρισμα κομματιών με λιγότερες από 50 συλλαβές στίχων.
- Περιορισμός όλων των νοτών σε εύρος οκάβων.
- Ομαδοποίηση όλων των οργάνων σε 8 κατηγορίες: Piano, Guitar, Bass, Strings, Wind, Synth, Drums, Effects

Επίσης για να έχουμε μια συνεπή και αναστρέψιμη αντιστοίχιση στίχων σε νότες, εφαρμόζουμε μια αυστηρή συλλαβοποίηση:

- Μετατρέπουμε κάθε λέξη σε φωνήματα
- Χωρίζουμε τις λέξεις σε συλλαβές, ώστε η καθεμία να περιέχει ένα φωνήεν
- Αν μια νότα αντιστοιχεί σε **n >1** συλλαβές, την χωρίζουμε σε **n** νότες ίσας διάρκειας
- Εάν μια συλλαβή εκτείνεται σε πάνω από **n >1** νότες, την αντιστοιχίζουμε στην πρώτη και αντιστοιχίζουμε τις επόμενες **n - 1** νότες σε ένα ειδικό σύμβολο.

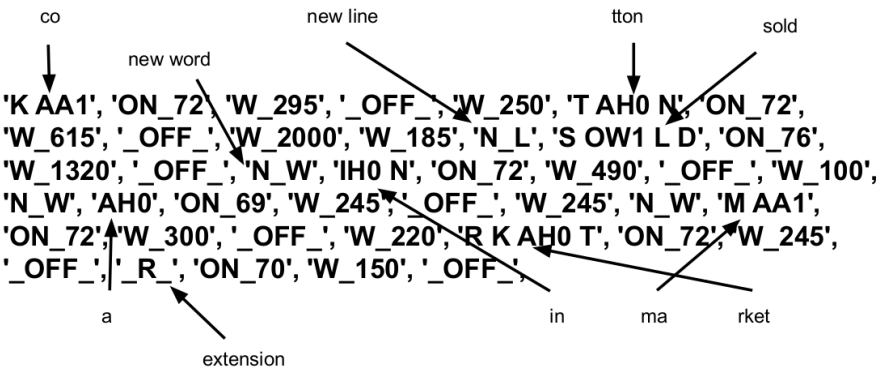
Μετά την παραπάνω διαδικασία έχουμε **8505** αρχεία. Από αυτά δημιουργούμε ακολουθίες της μορφής που φαίνεται στα Σχήματα 1 και 2, για το ορχηστρικό και το φωνητικό μέρος αντίστοιχα. Αυτές οι ακολουθίες αποτελούν τα δεδομένα που χρησιμοποιούμε στην εκπαίδευση.



Σχήμα 1: *Ακολουθία ορχηστρικού μέρους.*

Οι ακολουθίες αποτελούνται από γεγονότα των παρακάτω τύπων:

- **wait time** γεγονότα, μετράνε την παρέλευση του χρόνου (σε MIDI ticks)
- **note on** γεγονότα, δείχνουν ότι μια νότα (με αυτόν τον αριθμό) ξεκινάει να παίζεται
- **note off** γεγονότα, δείχνουν ότι μια νότα (με αυτόν τον αριθμό) σταματάει να παίζεται



Σχήμα 2: *Ακολουθία φωνητικού μέρους. Περιέχει στίχους και μελωδία ταυτόχρονα.*

Οι φωνητικές ακολουθίες έχουν επίσης:

- **syllable/phoneme** γεγονότα που περιλαμβάνουν τα φωνήματα που αντιστοιχούν στη νότα που ακολουθεί

- **extension** γεγονότα, όταν μια συλλαβή τραγουδιέται σε δύο ή περισσότερες νότες
- **boundary** γεγονότα, όπως κενό ή αλλαγή γραμμής, περιέχουν πληροφορία για τη δομή του κειμένου

Μεγέθη Ακολουθίας	Ορχηστρικό	Φωνητικά	Φωνητικά (χωρίς συλλαβές)
μέγιστο	59120	6115	5065
διάμεσο	13041	1645	1373
ελάχιστο	575	310	256

Πίνακας 1: *Μεγέθη ορχηστρικών και φωνητικών ακολουθιών*

Όπως φαίνεται στον Πίνακα 1, τα μεγέθη των ακολουθιών είναι πάρα πολύ μεγάλα. Παρόλο που χρησιμοποιούμε μια αρχιτεκτονική που αναπτύχθηκε για μοντελοποίηση μεγάλων ακολουθιών, η συμπίεση των ακολουθιών χωρίς απώλεια σημαντικής πληροφορίας είναι επιθυμητή. Η εκπαίδευση θα είναι ταχύτερη και μια πιο πυκνή αναπαράσταση μπορεί να κάνει το μοντέλο μας πιο εύρωστο.

Χρησιμοποιώντας την βιβλιοθήκη *music21* [19], εφαρμόζουμε μια μέθοδο συμπίεσης χρησιμοποιώντας ακόρντα (σύνολα νοτών). Αυτή η μέθοδος συγχωνεύει όλα τα διαφορετικά όργανα και κάθε νέα νότα οδηγεί σε ένα νέο ακόρντο. Με αυτόν τον τρόπο μπορούμε να μειώσουμε μια πολύπλοκη αναπαράσταση σε μια απλή διαδοχή ακόρντων, χωρίς να απορρίπτουμε πληροφορία και ταυτόχρονα την απλοποιούμε.

Μεγέθη Ακολουθίας	Ορχηστρικό	Μείωση
μέγιστο	11730	80.16%
διάμεσο	3220	75.31%
ελάχιστο	239	59.83%

Πίνακας 2: *Μεγέθη ορχηστρικής ακολουθίας μετά τη συμπίεση σε ακόρντα*

Όπως μπορούμε να δούμε στον Πίνακα 2, τα ακόρντα που δημιουργούνται μειώνουν τα μήκη της ορχηστρικής ακολουθίας κατά έναν μεγάλο παράγοντα, αλλά το να συμβολίσουμε απλά κάθε ακόρντο με τις νότες από τις οποίες αποτελείται, αυξάνει πολύ το μέγεθος των πιθανών συμβόλων. Επίσης, αν το ίδιο ακριβώς τραγούδι, μετατοπιζόταν σε άλλο μουσικό κλειδί θα αποτελούνταν από εντελώς διαφορετικά ακόρντα. Θα ήταν συνεπώς πιο χρήσιμο να μπορούμε να συμβολίζουμε τις σχετικές θέσεις και τη λειτουργία των ακόρντων. Για να λύσουμε αυτά τα ζητήματα, χρησιμοποιούμε έναν τύπο μουσικής ανάλυσης που ονομάζεται **ρωμαϊκή αριθμητική ανάλυση** και η βασική του ιδέα είναι ότι κάθε ακόρντο μπορεί να αναπαρασταθεί από έναν βαθμό της μουσικής κλίμακας στην οποία ανήκει. Την ίδια διαδικασία ακολουθούμε και για τη φωνητική μελωδία.

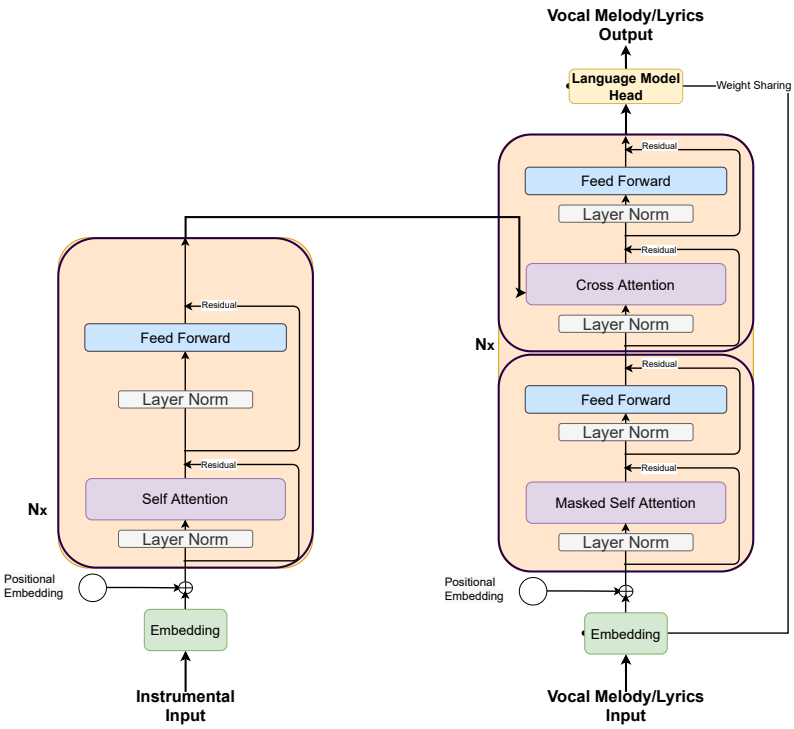
Τέλος, δημιουργούμε και ένα σύνολο δεδομένων όπως το παραπάνω, με τη διαφορά ότι δεν συμπεριλαμβάνουμε τα φωνήματα στην ακολουθία της μελωδίας, αλλά κρατάμε το αρχικό ολόκληρο κείμενο. Έτσι έχουμε τρεις ακολουθίες, την ορχηστρική, τη φωνητική μελωδία και τους στίχους σε μορφή κειμένου. Καθώς επιβάλλαμε τον διαχωρισμό σε συλλαβές που

εξηγήσαμε παραπάνω, είναι πολύ εύκολο να φτιάξουμε την φωνητική ακολουθία και αφού παράξουμε χωριστά τους στίχους.

Επίσης, δημιουργούμε ένα σύνολο δεδομένων από σκέτους στίχους που βρίσκουμε στο διαδίκτυο, φέρνοντας στους στη μορφή που εξηγήσαμε παραπάνω, για να προεκπαιδεύσουμε ένα γλωσσικό μοντέλο, όπως δείχνουμε στη συνέχεια.

0.4 Αρχιτεκτονικές των Μοντέλων

Σε αυτό το σημείο θα παρουσιάσουμε τις αρχιτεκτονικές των μοντέλων που θα χρησιμοποιήσουμε στα πειράματα μας. Πρόκειται για **(α)** μια απλή κωδικοποιητή-αποκωδικοποιητή ή αλλιώς sequence-to-sequence αρχιτεκτονική που μοντελοποιεί την φωνητική μελωδία και τους στίχους ως μέρος τη ίδιας ακολουθίας και **(β)** μια ¨διαχωρισμένη¨ αρχιτεκτονική που μοντελοποιεί ξεχωριστά την φωνητική μελωδία και τους στίχους, διατηρώντας την αλληλεξάρτησή τους.



Σχήμα 3: *Η απλή sequence-to-sequence αρχιτεκτονική με έναν κωδικοποιητή για το ορχηστρικό (αριστερά) και ένα αποκωδικοποιητή για το φωνητικό μέρος (δεξιά)*

Η απλή αρχιτεκτονική κωδικοποιητή-αποκωδικοποιητή βασίζεται στην αρχιτεκτονική Transformer που παρουσιάσαμε παραπάνω, με τη διαφορά της χρήσης του μηχανισμού προσοχής γραμμικής πολυπλοκότητας, FAVOR+, καθώς και τη χρήση αναστρέψιμων επιπέδων για επιπλέον ελάττωση στη χρήση μνήμης. Κάποιες επιπλέον προσθήκες σε σχέση με την απλή αρχιτεκτονική Transformer είναι η χρήση feed-forward chunking [42] στα feedforward υπο-επίπεδα, χρήση positional embedding τα βάρη των οποίων εκπαιδεύονται μαζί με το υπόλοιπο δίκτυο, αντί των τριγωνομετρικών συναρτήσεων που χρησιμοποιούνταν

προηγουμένως. Επίσης, το layer normalization [3] τοποθετείται πριν την είσοδο των υπολοίπων υπο-επιπέδων, κάτι που έχει αποδειχθεί πειραματικά και θεωρητικά να αποδίδει πολύ καλύτερα [12] [59]. Τέλος, χρησιμοποιούμε τη Gaussian Error Linear Unit (GELU) [31] ως συνάρτηση ενεργοποίησης στα feedforward υπο-επίπεδα. Η αρχιτεκτονική φαίνεται στο Σχήμα 3.

Συνοπτικά, θεωρώντας $\mathbf{X}_{1:n}$ την ακολουθία εισόδου και $\mathbf{Y}_{1:m}$ την ακολουθία εξόδου και τις παραμέτρους του κωδικοποιητή και του αποκωδικοποιητή θ_{enc} και θ_{dec} αντίστοιχα, το πρόβλημα που θέλουμε να λύσουμε έχει την μορφή:

$$p_{\theta_{\text{enc}},\theta_{\text{dec}}}(\mathbf{Y}_{1:m}|\mathbf{X}_{1:n}) \tag{2}$$

Ο κωδικοποιητής μοντελοποιεί τη συνάρτηση $f_{\theta_{\text{enc}}} : \mathbf{X}_{1:n} \rightarrow \overline{\mathbf{X}}_{1:n}$ και η Εξίσωση 2 μπορεί να γραφεί χρησιμοποιώντας τον κανόνα του Bayes ως:

$$p_{\theta_{\text{enc}},\theta_{\text{dec}}}(\mathbf{Y}_{1:m}|\mathbf{X}_{1:n}) = p_{\theta_{dec}}(\mathbf{Y}_{1:m}|\overline{\mathbf{X}}_{1:n}) = \prod_{i=1}^{m} p_{\theta_{\text{dec}}}(\mathbf{y}_i|\mathbf{Y}_{0:i-1}, \overline{\mathbf{X}}_{1:n}) \tag{3}$$

Είναι εύκολο να διαπιστωθεί πως οι ακολουθίες που χρησιμοποιούμε έχουν μια συγκεκριμένη δομή, ειδικά η φωνητική. Για παράδειγμα ένα σύνολο φωνημάτων ακολουθείται από ένα γεγονός *note on*, ή μετά από ένα *note on* ακολουθεί ένα *wait time*. Εκμεταλλευόμαστε αυτή την δομή και την επιβάλλουμε κατά την παραγωγή της ακολουθίας, αναπτύσσοντας έναν σχετικό αλγόριθμο. Έτσι, βοηθάμε το μοντέλο να παράγει κάθε φορά σωστές ακολουθίες.

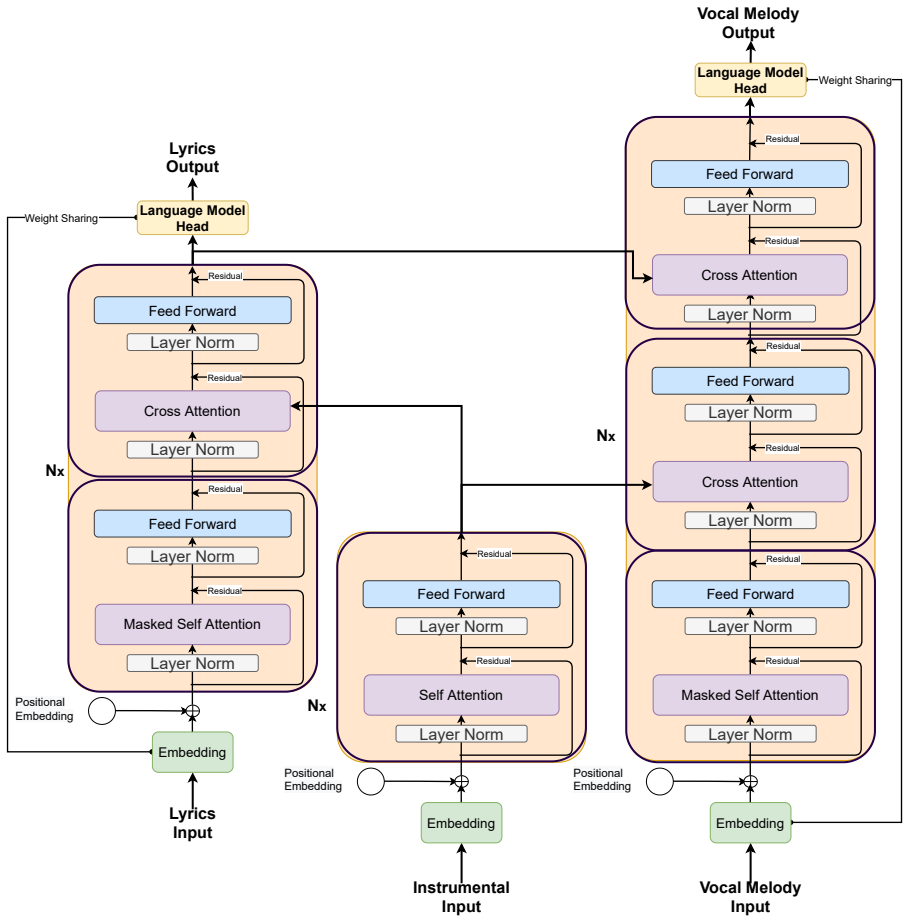
Η διαχωρισμένη αρχιτεκτονική που αναπτύσσουμε συνδέει τρεις ακολουθίες μεταξύ τους και χρησιμοποιώντας την θέλουμε δύο από αυτές να είναι οι έξοδοι.

Όπως φαίνεται στο Σχήμα 4, διατηρούμε τον κωδικοποιητή αμετάβλητο και αντί για έναν μόνο αποκωδικοποιητή, έχουμε έναν αποκωδικοποιητή για στίχους και έναν για φωνητική μελωδία, στον οποίο προσθέτουμε ένα δεύτερο επίπεδο cross-attention. Αυτή η τεχνική για τον συνδυασμό πολλαπλών εισόδων σε έναν αποκωδικοποιητή έχει μελετηθεί στο [47] και έχει αποδειχθεί ότι αποδίδει πολύ καλά. Η έξοδος του κωδικοποιητή του ορχηστρικού μέρους συμμετέχει τόσο στο cross-attention των στίχων όσο και στης φωνητικής μελωδίας, ενώ η έξοδος του τελευταίου επιπέδου του αποκωδικοποιητή των στίχων συμμετέχει στο δεύτερο υπο-επίπεδο cross-attention της μελωδίας.

Εκφράζοντας τα παραπάνω με μαθηματικούς όρους, δεδομένου ότι οι στίχοι αποτελούν την ακολουθία $\mathbf{Z}_{1:q}$ και οι παράμετροι του αποκωδικοποιητή στίχων είναι θ_{lm} έχουμε την πιθανότητα:

$$p_{\theta_{\text{enc}},\theta_{\text{lm}}}(\mathbf{Z}_{1:q}|\mathbf{X}_{1:n}) = p_{\theta_{lm}}(\mathbf{Z}_{1:q}|\overline{\mathbf{X}}_{1:n}) = \prod_{i=1}^{q} p_{\theta_{\text{lm}}}(\mathbf{z}_i|\mathbf{Z}_{0:i-1}, \overline{\mathbf{X}}_{1:n}) \tag{4}$$

Η ακολουθία $\mathbf{Z}_{1:q}$ πλέον κωδικοποιείται στην ακολουθία $\overline{\mathbf{Z}}_{1:q}$, που εξαρτάται από τους στίχους και το ορχηστρικό μέρος. Έτσι, η πιθανότητα για την φωνητική μελωδία δίνεται από



Σχήμα 4: *Η διαχωρισμένη αρχιτεκτονική μας με έναν κωδικοποιητή (κέντρο), έναν αποκωδικοποιητή για στίχους (αριστερά) και έναν αποκωδικοποιητή για φωνητική μελωδία (δεξιά) με δύο υπο-επίπεδα cross-attention. Ο κωδικοποιητής ρυθμίζει και τους δύο αποκωδικοποιητές, ενώ οι κωδικοποιήσεις του μουτέλου στίχων ρυθμίζουν του αποκωδικοποιητή φωνητικής μελωδίας.*

τον τύπο:

$$p_{\partial_{\text{enc}},\partial_{\text{lm}},\partial_{\text{dec}}}(\mathbf{Y}_{1:m}|\mathbf{X}_{1:n}\mathbf{Z}_{1:q}) = p_{\partial_{dec}}(\mathbf{Y}_{1:m}|\overline{\mathbf{X}}_{1:n},\overline{\mathbf{Z}}_{1:q}) = \prod_{i=1}^{m} p_{\partial_{\text{dec}}}(\mathbf{y}_i|\mathbf{Y}_{0:i-1},\overline{\mathbf{X}}_{1:n},\overline{\mathbf{Z}}_{1:q}) \quad (5)$$

Το μοντέλο εκπαιδεύεται με στόχο να ελαχιστοποιήσει το άθροισμα των cross-entropy loss συναρτήσεων των δύο ακολουθιών.

Ένα γλωσσικό μοντέλο που αποτελείται μόνο από έναν αποκωδικοποιητή χρησιμοποιείται για να βελτιώσει την ποιότητα των παραγόμενων στίχων. Χρησιμοποιούμε ένα προεκπαιδευμένο μοντέλο GPT-2 [69] σε αγγλικό κείμενο[1], το οποίο εκπαιδεύουμε περαιτέρω σε αγγλικούς στίχους ώστε να μάθει περισσότερα για τη δομή και το περιεχόμενο τους. Αφού το εκπαιδεύσουμε, το εισάγουμε στην παραπάνω αρχιτεκτονική, απλά προσθέτοντας ένα υπο-επίπεδο cross-attention με τυχαία αρχικοποίηση. Αυτό δεν επηρεάζει την απόδοση

[1] https://huggingface.co/distilgpt2

του μοντέλου όπως βλέπουμε πειραματικά. Αυτή η διαδικασία χρησιμοποιείται και σε άλλα προβλήματα με επιτυχία όπως φαίνεται στην βιβλιογραφία [76].

0.5 Πειράματα και Αποτελέσματα

Σε αυτό το σημείο θα περιγράψουμε εν συντομία τα πειράματά μας, και κάποια συμπεράσματα που εξάγουμε βάση αυτών, ενώ στη συνέχεια θα δούμε και τα αποτελέσματα της ποιοτικής μελέτης που διεξήγαμε. Ο κώδικας για όλη την παρούσα διπλωματική εργασία βρίσκεται στο GitHub[2], όπως και ο κώδικας που βασίσαμε μέρος της αρχιτεκτονικής μας (σε PyTorch)[3]. Χρησιμοποιήσαμε επίσης τη βιβλιοθήκη DeepSpeed [71] κατά την εκπαίδευση. Ο αναγνώστης μπορεί να επισκεφτεί τα Παραρτήματα για να δει σχετικές γραφικές παραστάσεις από την εκπαίδευση των μοντέλων, καθώς και κάποια παραδείγματα αποτελεσμάτων.

Εκπαιδεύουμε τρία μοντέλα συνολικά για παραγωγή στίχων και φωνητικής μελωδίας: **(α)** μια απλή αρχιτεκτονική κωδικοποιητή-αποκωδικοποιητή σε ακολουθίες νοτών και **(β)** ακόρντων, καθώς και **(γ)** μια διαχωρισμένη αρχιτεκτονική σε ακολουθίες ακόρντων. Επίσης, κάνουμε μια σύντομη εκπαίδευση του γλωσσικού μοντέλου σε σκέτους στίχους.

Εκπαίδευση	**Όλα τα Μοντέλα Μελωδίας/Στίχων**	**Γλωσσικό Μοντέλο**
Δείγματα Εκπαίδευσης	7654	237299 (29663 παρτίδες)
Δείγματα Αξιολόγησης	851	26367
Εποχές	6	1
Συνολικά Βήματα	45923	33370

Πίνακας 3: *Μέγεθος συνόλων δεδομένων (90% για εκπαίδευσης), εποχές και συνολικά βήματα εκπαίδευσης και για τα τρία μουτέλα, καθώς και το Γλωσσικό Μουτέλο*

Στον Πίνακα 3 φαίνονται στοιχεία για την εκπαίδευση όλων των μοντέλων, ενώ στον Πίνακα 4 φαίνεται η ταχύτητα εκπαίδευσης, αλλά και παραγωγής αποτελεσμάτων, για τα τρία μοντέλα. Παρατηρούμε ότι η συμπίεση της ακολουθίας των οργάνων είναι πολύ σημαντική και οδηγεί σε πολύ ταχύτερη εκπαίδευση (σχεδόν 3 φορές πιο γρήγορα). Να σημειωθεί ότι έχουμε μια μέση μείωση κατά 75% στο μήκος της ακολουθίας εισόδου (βλ. Πίνακα 2). Η διαχωρισμένη αρχιτεκτονική είναι πιο αργή από την απλή όπως αναμενόταν, καθώς είναι πιο περίπλοκη λόγω των επιπλέον υποστρωμάτων και του γλωσσικού μοντέλου. Ωστόσο, τα μεγαλύτερα μήκη ακολουθίας επηρεάζουν περισσότερο την ταχύτητα.

Ταχύτητα/Διάρκεια	**Απλή (νότες)**	**Απλή (ακόρντα)**	**Διαχωρισμένη (ακόρντα)**
Δείγματα/δευτερόλεπτο (μέση τιμή)	0.171	0.473	0.259
Ώρες εκπαίδευσης	74.59	26.96	49.25
Παραγόμενα μέρη/δευτερόλεπτο	5.89	6.71	9.09

Πίνακας 4: *Ταχύτητα εκπαίδευσης, συνολική διάρκεια εκπαίδευσης και ταχύτητα παραγωγής για τα τρία μουτέλα (χρόνοι μετρημένοι σε GPU NVIDIA Tesla T4)*

Ο αναγνώστης θα βρει στο Παράρτημα A και δύο μετρικές σχετικές με τη δημιουργία

[2] https://github.com/gulnazaki/thesis
[3] https://github.com/lucidrains/performer-pytorch

των ακολουθιών. Η πρώτη μετρική είναι η Bilingual Evaluation Understudy (BLEU) και χρησιμοποιείται κυρίως στην αυτόματη μετάφραση. Λόγω του ότι η μετρική αυτή είναι πολύ αυστηρή για το πρόβλημα μας, δεδομένου ότι ακόμα και δύο άνθρωποι θα προσέθεταν πολύ διαφορετικά φωνητικά σε ένα κομμάτι, δεν μας δίνει πολύ χρήσιμα αποτελέσματα. Για αυτόν τον λόγο, αναπτύσσουμε μια πολύ πιο χαλαρή μετρική που την ονομάζουμε Valid Structure Metric (VSM). Αυτή η μετρική μας πληροφορεί για τη δομή της ακολουθίας και αν και είναι αρκετά αδύναμη, δίνει μια καλή εκτίμηση για την πορεία της εκπαίδευσης. Φαίνεται συγκεκριμένα, ότι χρησιμοποιώντας ακόρντα, η μετρική αυτή βρίσκεται πιο σταθερά κοντά στη μονάδα (το μέγιστο).

Τέλος, θα συγκρίνουμε τα αποτελέσματα των μοντέλων μας διεξάγοντας μια **ποιοτική αξιολόγηση**. Ακολουθούμε την εξής διαδικασία:

- Επιλέγουμε τυχαία 5 MIDI αρχεία από το σύνολο δεδομένων αξιολόγησης.
- Συνθέτουμε ήχο από τα ορχηστρικά μέρη του MIDI, χρησιμοποιώντας μια σχετική βιβλιοθήκη[4] και το λογισμικό σύνθεσης FluidSynth[5].
- Παράγουμε φωνητική μελωδία και στίχους με τα τρία μοντέλα μας.
- Χρησιμοποιούμε την αρχιτεκτονική σύνθεσης φωνής Mellotron [85] για να δημιουργήσουμε φωνητικά σε ηχητική μορφή από τα αποτελέσματά μας.
- Αναμιγνύουμε το παραγόμενο τραγούδι με το ορχηστρικό μέρος και λαμβάνουμε συνολικά 15 αρχεία ήχου.
- Συμπεριλαμβάνουμε τους στίχους σε μορφή κειμένου.

Στη συνέχεια, ζητάμε από **15** άτομα να συγκρίνουν τα αποτελέσματα των 3 μοντέλων, για καθένα από τα 5 κομμάτια. Τους ζητάμε να τα συγκρίνουν ως προς τρεις άξονες:

1. **Μελωδικότητα/Ρυθμικότητα**: πόσο μουσικό ή ενδιαφέρον είναι το φωνητικό μέρος, όσον αφορά τον ρυθμό και τη μελωδία
2. **Σχέση με τη Μουσική**: πόσο καλά ταιριάζει το φωνητικό μέρος με τα όργανα, τόσο από άποψη αρμονίας όσο και συγχρονισμού/ρυθμού
3. **Περιεχόμενο Στίχων**: ποιότητα των παραγόμενων στίχων

Συμπεραίνουμε ότι η διαχωρισμένη αρχιτεκτονική είναι σίγουρα καλύτερη όσον αφορά τους στίχους, όπως αναμενόταν. Επίσης, ξεπερνά σημαντικά την απλή αρχιτεκτονική, όσον αφορά τη σχέση των φωνητικών με τη μουσική. Τέλος, είναι ελαφρώς καλύτερη στην παραγωγή μελωδιών. Συγκρίνοντας τα δύο μοντέλα κωδικοποιητή-αποκωδικοποιητή, είναι αρκετά παρόμοια, με αυτό που χρησιμοποιεί ακόρντα να είναι λίγο καλύτερο όσον αφορά τη σχέση των φωνητικών με τη μουσική και λίγο χειρότερο στην ποιότητα της φωνητικής μελωδίας.

[4] https://github.com/craffel/pretty-midi
[5] https://www.fluidsynth.org

Δεδομένου ότι έχουμε χρησιμοποιήσει σχετικά μικρό αριθμό κομματιών και συμμετεχόντων στη μελέτη μας, δεν μπορούμε να καταλήξουμε σε κάποιο συμπέρασμα με απόλυτη βεβαιότητα.

0.6 Συμπεράσματα και Μελλοντικές Κατευθύνσεις

Σε αυτή τη διατριβή διερευνούμε ένα πολύ ενδιαφέρον πρόβλημα, τη δημιουργία στίχων και φωνητικής μελωδίας για ένα συγκεκριμένο μουσικό κομμάτι. Από όσο γνωρίζουμε, η δουλειά μας είναι η πρώτη που ενσωματώνει το ορχηστρικό κομμάτι της συνοδείας κατά την μελέτη στίχων και φωνητικής μελωδίας. Η έρευνά μας, και ο σχετικός κώδικας, καθιστούν δυνατή τη δημιουργία φωνητικών για οποιοδήποτε ορχηστρικό κομμάτι.

Εστιάζουμε στη συμβολική μουσική και συγκεκριμένα στη μορφή MIDI. Προτείνουμε μια αναπαράσταση σε μορφή κειμένου και εξερευνούμε δύο τύπων αρχιτεκτονικές που βασίζονται στο μοντέλο του Transformer. Μια απλή αρχιτεκτονική κωδικοποιητή-αποκωδικοποιητή και μια αρχιτεκτονική που μοντελοποιεί ξεχωριστά το μέρος των οργάνων, των στίχων και της φωνητικής μελωδίας, την οποία ονομάζουμε *διαχωρισμένη*. Ακολουθούμε ένα βήμα συμπίεσης της ακολουθίας εισόδου (οργάνων) σε ακόρντα, πετυχαίνοντας μείωση 75%, χρησιμοποιώντας μουσικοθεωρητική ανάλυση. Εκπαιδεύουμε τρία μοντέλα συνολικά για παραγωγή στίχων και φωνητικής μελωδίας: **(α)** μια απλή αρχιτεκτονική κωδικοποιητή-αποκωδικοποιητή σε ακολουθίες νοτών και **(β)** ακόρντων, καθώς και **(γ)** μια διαχωρισμένη αρχιτεκτονική σε ακολουθίες ακόρντων. Χρησιμοποιώντας ένα μοντέλο σύνθεσης φωνής, δημιουργούμε ήχο από την συμβολική αναπαράσταση που παράγουμε και διεξάγουμε μια ποιοτική έρευνα.

Παρατηρούμε ότι η συμπίεση που πραγματοποιήσαμε είναι καθοριστικής σημασίας για την ταχύτητα εκπαίδευσης των μοντέλων μας, ενώ δίνει και ελαφρώς καλύτερα αποτελέσματα. Επίσης, η νέα αρχιτεκτονική που προτείνουμε, έχει την καλύτερη απόδοση στην ποιοτική μελέτη και παράγει ιδιαίτερα ικανοποιητικούς στίχους. Αυτό είναι αναμενόμενο, εφόσον χρησιμοποιούμε ένα προεκπαιδευμένο γλωσσικό μοντέλο που εκπαιδεύουμε περαιτέρω σε στίχους.

Στο μέλλον, στοχεύουμε στη βελτίωση της απόδοσης των αρχιτεκτονικών που μελετήσαμε. Όσον αφορά την αναπαράσταση των δεδομένων, θα θέλαμε να δοκιμάσουμε χρήση των μουσικών αξιών (για παράδειγμα με βάση τα εξηκοστά τέταρτα) ή περαιτέρω απλοποίηση των ακόρντων. Όσον αφορά τις βελτιώσεις στην αρχιτεκτονική, θα μπορούσαμε να πειραματιστούμε με άλλες μεθόδους για να μειώσουμε τις απαιτήσεις σε μνήμη, για παράδειγμα χρησιμοποιώντας gradient checkpointing [13]. Αξίζει επίσης να μελετηθεί εάν η εξάρτηση των στίχων από το ορχηστρικό μέρος είναι χρήσιμη, πραγματοποιώντας ένα ablation study.

Για να μειώσουμε την πολυπλοκότητα της διαχωρισμένης αρχιτεκτονικής, θα μπορούσαμε επίσης να δοκιμάσουμε να χρησιμοποιήσουμε άλλους τρόπους για να ενσωματώσουμε προηγούμενες γνώσεις από ένα γλωσσικό μοντέλο. Αυτό προϋποθέτει εκπαίδευση ενός γλωσσικού μοντέλου από την αρχή με χωρισμό (tokenization) των λέξεων σε συλλαβές. Μια κομψή λύση που προσθέτει έναν όρο κανονικοποίησης ανάλογα με την κατανομή εξόδου του γλωσσικού μοντέλου, παρουσιάζεται στο [5].

Μια άλλη βελτίωση, συγκρίσιμη με τη χρήση προηγούμενων γνώσεων για τους στίχους, θα ήταν η προεκπαίδευση του κωδικοποιητή, με έναν μη εποπτευόμενο στόχο εκπαίδευσης, όπως στα γλωσσικά μοντέλα (για παράδειγμα BERT [21]). Θα ήταν χρήσιμο, καθώς ο αριθμός των αρχείων MIDI που περιέχουν μόνο μουσική είναι πολύ μεγάλος.

Τέλος, θα θέλαμε να ξεπεράσουμε τα όρια της συμβολικής μουσικής. Το σύνολο δεδομένων DALI [56] περιέχει κομμάτια ήχου, με συγχρονισμένους στίχους και φωνητική μελωδία. Η μορφή του είναι πολύ κοντά στο σύνολο δεδομένων που δημιουργήσαμε. Ένας απλός τρόπος να χρησιμοποιηθεί σε μία από τις υπάρχουσες αρχιτεκτονικές μας, θα ήταν να διαχωριστούν τα φωνητικά και να εξάγουμε temporal audio features από τη μουσική. Τέλος, θα μπορούσαμε να χρησιμοποιήσουμε αυτό το σύνολο δεδομένων για να μελετήσουμε κατευθείαν την παραγωγή φωνητικών σε μορφή ήχου.

Chapter 1

Introduction

Through the mountain
So many things
Between two of childhood and nothin' you needed a degree
On the time to give you a painter in your mind
One life or two
Through what a rainbow beauty and through life

Through many friends
Through a future of consciousness
And some deception
Reaching through just another man
And with this motherless man you'll give away
On the heart

Tell another chance and darkness
Feel a two in two
I will say goodbye
To a good man

Epoch 1, Step 33075, Loss: 1.7501434139

This introductory chapter aspires to make the reader familiar with the setting of our work and the task which the current dissertation deals with. At first, we manifest our motivation behind the following research direction and we make a short, first presentation of the broader research content and our work's position in it. We discuss the ways in which our work is original and we formulate our goals and research objectives as well as our contributions in the field. Lastly, we present the organization of this volume, making what is discussed in each chapter more clear to the reader.

1.1 Motivation and Originality of our Work

Singing is the act of producing musical tones with one's voice. Its origins are impossible to track, but they are said to predate the development of spoken language, while voice is presumed to be the original musical instrument [43]. The function and characteristics

of singing would vary tremendously between cultures and time periods, but its universal importance is undisputable. The merging of instrumental accompaniment and singing started to become more popular in late Renaissance Florence[1]. Music and singing continued to coexist in a theatrical context and gained the format we know today during the last hundred years. Nowadays, singing and instrumental music are so commonly tied, that when vocals are not present in a song it is characterized as instrumental and when a music performance includes singing and no instruments it is called *a capella*.

Singing has an intrinsic value in human culture and is considered very hard to model and automate, because it requires multiple skills, such as musical understanding and creativity, that are very hard to formulate. For this reason, it is generally believed to be a big step towards the long road of achieving Artificial General Intelligence[2]. A big part of research regarding singing has focused on information retrieval tasks, such as transcription of lyrics or melody. Another part, that is relevant to our work, focuses on generation and it can be further divided into generation of lyrics/vocal melody and singing voice synthesis. The latter is closely related to the more general task of speech synthesis and so has gained more attention. Generation of lyrics and vocal melody are less straightforward tasks and require, among others, the ability to be creative. For this reason, they have gained more interest recently, with the success of generational deep learning methods.

Motivated by the writer's adoration for music and enthusiasm to explore the challenges and opportunities presented when dealing with artificial creativity, the above task was chosen to be the subject of the current dissertation. With this work, not only do we study the relation of lyrics and vocals to music, but we create an end-to-end tool that can be used by musicians or regular users to provide new creative ideas, useful in the songwriting process.

Furthermore, our work aspires to fill a substantial gap that exists on the matter. We will present a thorough review of the relevant research in the next chapter, but we will quickly mention what we believe to be the most important aspects that have not been addressed and are worth investigating. As we mentioned above, the relation of both vocal melody and lyrics to music is very important and has not been yet taken into account when dealing with generation. Previous work also follows a proof-of-concept approach and deals with lyrics in the level of a few sentences, while working with full songs presents many additional challenges. Finally, recent architectures that are strongly considered state-of-the-art in many fields, such as the Transformer, have not been tested yet. *The present work is the first attempt, to our better knowledge, to generate both lyrics and vocal melody for a full given music piece in the symbolic domain, using a state-of-the-art architecture and utilizing prior knowledge from language models.*

[1] https://en.wikipedia.org/wiki/Florentine_Camerata
[2] https://en.wikipedia.org/wiki/Artificial_general_intelligence

1.2 Research Objective and Contributions

The main research objective of this work is to develop an end-to-end model, that given any instrumental music piece in MIDI format, namely a symbolic representation of all notes played, as input, can generate lyrics and a synchronized vocal melody, namely pitch and timing information of the singing music tones. Using a singing voice synthesis software we can mix the result with the synthesized music and create a complete piece with vocals. We model the instrumental input and vocal output as two sequences of text events and tackle the resulting problem as a sequence-to-sequence task, analogous to translation from one language to another.

Our main contributions to the field are the following:

- We research for the first time the generation of lyrics and vocal melody conditioned on instrumental accompaniment
- We use the Transformer architecture, for the first time in the context of lyrics and vocal melody generation. Using a linear attention mechanism, we are able to train our model to full song sequences, up to 50 times longer than those usually modelled in a single GPU.
- We apply music analysis to compress our symbolic music data up to 80% and make them key-independent, making our models more robust.
- We develop a novel architecture to decouple lyrics and vocal generation, while taking into account their interdependence and providing the ability to use any language model and optional conditioning on lyrics.

To research the above:

- We build a dataset that is suitable for our task, by separating vocals from instruments and creating text sequences that describe each song.
- We train a language model to an abundancy of english lyrics found online and use it in our architecture to improve the quality of generated lyrics.
- We develop a decoding strategy, specific to our task, to make the generation process more robust by enforcing valid structure. Also, we devise a generation evaluation method based on the sequence structure.
- We transform our results into a suitable format and use a singing voice synthesis model to generate vocals. We further use them, alongside the input instrumental, to conduct a qualitative evaluation study.

The code used for this thesis can be found on GitHub [3].

[3] https://github.com/gulnazaki/thesis

1.3 Thesis outline

This volume is organized in the following way:

In **Chapter 2** we present the theoretical background that is necessary to understand this work. At first we present an overview of the field of Machine Learning. We start by documenting the history and advancements in the broader field of Artificial Intelligence and afterwards we focus on the basic architectures and methods used in Deep Learning, a very successful subfield of Artificial Intelligence and Machine Learning. Next, we document the field of Natural Language Processing and the usage of Deep Learning methods. We analyze the attention mechanism and the Transformer architecture, which is the basis of our work. Following, we present the literature relevant to reducing the memory usage of the state-of-the-art methods, which is a drawback we had to overcome in our work. We also introduce concepts of symbolic music and music theory, such as MIDI, chords, scales and roman numeral analysis. Finally, we discuss related work, by presenting research that has studied tasks, close to ours. We present previous work on symbolic music, vocal melody and lyrics generation, as well as singing voice synthesis.

In **Chapter 3** we present the steps we followed to built a dataset for our task and the decisions we took. We talk about the original dataset we based our work on, the changes we had to make an the reasons we used the chosen format. Moreover, we will analyze the tools we used to generalize and compress our data based on music analysis. Finally, we mention the changes we made in our dataset to model lyrics and vocal melody separately and the lyrics dataset we assembled to pretrain our language model.

Chapter 4 is where the reader can find more details about the architecture of our sequence-to-sequence model and enhancements/improvements we decided to use. We present a decoding strategy that exploits the structure of the generated sequences. Also, we introduce a novel architecture that decouples melody and lyrics, while taking into account their codependency.

In **Chapter 5** we talk in detail about our experiments. We discuss the training process and the hyperparameters we chose, and we present and compare the results from all models. We also, discuss the generation evaluation metric we use and we introduce a structure metric that we devise. Finally, we analyze the qualitative evaluation study we conduct and discuss the outcomes.

Chapter 6 is the last chapter of this volume. We list some concluding remarks inferred from our work and we propose future work that can be done to improve current results, as well as possible directions we could take from here.

Chapter 2

Background

As mentioned in the Introduction, in this Chapter we will lay the foundations that our work has built upon. We present our theoretical background, regarding Machine Learning literature and especially the Deep Learning subfield. We analyze Deep Learning approaches to Natural Language Processing and focus on the base of our models, the Transformer architecture, and its usage in sequence-to-sequence modelling. We also talk about the drawbacks of commonly applied methods, regarding memory usage, and we explore recent work that deals with overcoming them.

Finally, we introduce some basic concepts that we use in this work, such as symbolic music representation and music theory concepts (notes, chords, scales and others). We also present related work, namely previous approaches to tasks relevant to ours, such as symbolic music generation, vocal melody generation and lyrics generation, as well as singing voice synthesis.

2.1 Introduction to Machine Learning

2.1.1 A Short History of Artificial Intelligence

Artificial Intelligence (AI) can be defined as intelligence demonstrated by machines, a type of intelligence that can be defined or computed, so it differs from natural intelligence encountered in life forms, which involves consciousness and emotionality. A first distinction is made to strong AI (Artificial General Intelligence), which relates to the hypothetical ability of a machine to understand and learn any intellectual task that is considered characteristic of human intelligence, and to weak AI, which focuses on solving a specific task and it is closer to the level that has been reached until today.

Humans have imagined that machines could have the above abilities much earlier than the development of computers. Artificial beings capable of thinking have appeared as storytelling devices in antiquity. Talos[1], a giant automaton appearing in Greek Mythology, is considered by many to be the first embodiment of this idea.

The starting point of the history of Artificial Intelligence is generally recognized to be the work of McCulloch and Pitts on *artificial neurons* [54]. In this paper, the authors describe a simple logical function computational model for a cell, called a neuron, in

[1] https://en.wikipedia.org/wiki/Talos

what is believed to be the first description of neural networks. The development of this model, and the connections to biology that are drawn, are of course a product of its time, since this period was marked by discoveries in neurobiology, information theory and cybernetics, as well as the insight given by Alan Turing's theory of computation[2].

AI started being a separate field in 1956, after being distinguished from cybernetics. During the next 50 years, new variations of this new technology were used to solve problems that were easy to formulate and hard for humans to solve, for example playing the game of checkers (1954), or proving logical theorems (1956). The most successful and famous application has been Deep Blue[3], a model playing chess, that in 1997 managed to beat world champion, Garry Kasparov.

During this period, there were a lot of different approaches to AI, as well as phases of growth and decline, but the invariant was the barrier that was presented when dealing with hard to formalize tasks, that humans can perform easily, almost without thinking, but machines cannot. An approach to solve these *real-world* tasks, was to insert hard-coded world knowledge into a model, using specific formal languages. This comprised a methodology called *knowledge based* AI. Because of the increasing difficulty and human effort needed to encode this knowledge, the research interest shifted away.

A different school of thought argued that the developed models should be able to acquire knowledge themselves, without depending on human experts and explicit representations. Comparable to the human learning procedure, they would be able to improve through experience and data. This approach is called *Machine Learning* (ML) and it can be thought of as a part or subfield of AI. Both fields share roots and goals, with ML focusing away from the previously mentioned symbolic representations, while borrowing methods and models from statistics and probability theory.

2.1.2 Machine Learning

Machine learning (ML) is a subfield of AI, that studies computer algorithms that improve automatically through data and experience. Machine learning algorithms build a model based on sample data in order to make predictions or decisions with no need for explicit task-specific programming. Machine Learning algorithms are used with great success for various applications, especially in scenarios in which it is challenging for a human to manually design algorithms to solve them, such as in Computer Vision and Natural Language Processing.

The term Machine Learning was coined in 1959 by Arthur Samuel, and a volume that summarizes these first attempts, mostly focused on pattern recognition is *Learning Machines* [61] by Nils Nilsson. A frequently quoted, formal definition of ML is given by Tom Mitchel [58]: "A computer program is said to learn from experience E with respect to some class of tasks T and performance measure P if its performance at tasks in T, as measured by P, improves with experience E."

The most common distinction of Machine Learning approaches depends on the feed-

[2] https://en.wikipedia.org/wiki/Theory_of_computation
[3] https://en.wikipedia.org/wiki/Deep_Blue_(chess_computer)

back available to the learning algorithm and the division is three-fold. The three categories are quickly discussed below, but it should be noted that there are approaches that do not fit into this categorisation or systems that use more than one approach, for example semi-supervised learning.

Supervised Learning

Supervised learning can be thought of as a family of algorithms that learn a function from input examples to target values given a set of data, for which the target responses are known. These input-output pairs are called *training data*, while the output is referred to as *ground truth*. Supervised models are designed to derive a mapping function g that approximates the implicit relationships of the training dataset, with the goal of making predictions about previously unseen inputs.

So, given a set of N training examples of the form $\{(x_1, y_1), ..., (x_N, y_N)\}$ such that x_i is the feature vector of the i-th example and y_i is its label (i.e., class or value), the learning algorithm seeks a function $g : X \to Y$, where X is the input space and Y is the output space. We can denote a scoring function $f : X \times Y \to \mathbb{R}$, such that g returns the y value with highest score: $g(x) = \arg\max_y f(x, y)$. To measure how well a function fits the training data, a loss function $\mathcal{L} : Y \times Y \to \mathbb{R}^{\geq 0}$ is defined. We want to minimize the sum of the loss function for all training examples, with the loss of predicting each output value being $\mathcal{L}(y_i, \hat{y})$, for a training example (x_i, y_i) and prediciton $\hat{y}$.

Supervised algorithms are split into two main categories, based on the desired output: **(a)** classification and **(b)** regression. The first refers to predicting outputs that are restricted to a distinct set of values, called class, while the second refers to the problem of estimating real-valued outputs within a predefined range.

Unsupervised Learning

Unsupervised learning is a family of algorithms that learn to infer patterns, without given target values for each learning example. The algorithm discovers the underlying structure or distribution of the data, through mimicry. A very common unsupervised learning problem is Clustering, in which a model groups or divides data points into categories based on a similarity measure, such that points that belong to the same cluster are more similar to points that belong to other clusters. Another set of techniques and tasks is Representation Learning, where the model discovers meaningful representations that can be later used to help in other tasks, replacing manual feature engineering. Lastly, many generative models can create new data that are drawn from a distribution of real data given as input.

Reinforcement Learning

Reinforcement learning is an area of ML that researches the way intelligent agents take actions in a dynamic environment, with the goal of maximizing a cumulative reward. As in the Unsupervise Learning, there is no need of labelled input-output pairs. The focus

of this approach is to find a balance between exploration (of uncharted territory) and exploitation (of current knowledge), while sub-optimal actions do not have to be explicitly corrected. The problem can be stated in the form of a Markov decision process. To give a more formal definition, let us denote:

- a set of environment and agent states, S
- a set of agent actions, A
- the probability of transition (at time t) from state s to state s' under action a $P_a(s, s') = \Pr(s_{t+1} = s' \mid s_t = s, a_t = a)$
- the immediate reward after transition from s to s' with action a $R_a(s, s')$

At each discrete time step t, the agent receives the current state s_t and reward r_t, and then chooses an action a_t from a set of available actions. The environment, given a_t, moves to a new state s_{t+1} and the agent receives the new reward r_{t+1} associated with the transition (s_t, a_t, s_{t+1}). The goal of the agent is to learn a policy: $\pi : A \times S \rightarrow [0, 1]$, $\pi(a, s) = \Pr(a_t = a \mid s_t = s)$ that maximizes the expected cumulative reward.

2.1.3 Basic Machine Learning Methods

The **Perceptron** is an algorithm for supervised learning of binary classifiers that was introduced in 1957 [75]. It is the basic building block of neural networks and it can be considered as the simplest case, a single layer neural network (with one neuron). It is a type of linear classifier, with a learnable weight vector. It can be denoted as a linear predictor function f that maps its input $\mathbf{x}$ vector to a binary output value $f(\mathbf{x})$:

$$f(\mathbf{x}) = \begin{cases} 1 & \text{if } \mathbf{w} \cdot \mathbf{x} + b > 0 \\ 0 & \text{otherwise} \end{cases} \tag{2.1}$$

where $\mathbf{w}$ is a vector of weights, $\mathbf{w} \cdot \mathbf{x}$ is the dot product equal to $\sum_{i=1}^{m} w_i x_i$, and b is the bias.

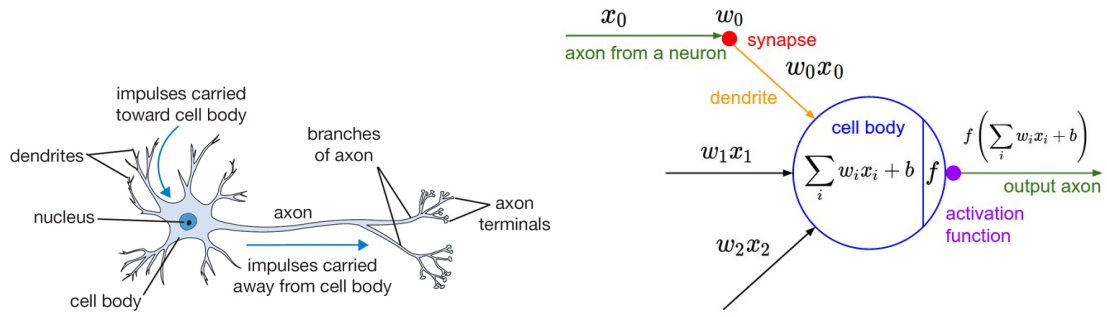


Figure 2.1: *An illustration of the Perceptron with mathematical notation (right) and a drawing of a biological neuron (left) to draw analogies and show the researchers' inspiration. (Source: [38])*

Regardless of the innovative idea of the Perceptron, it was quickly proved that it cannot be used for non-linearly separated classes, with the most infamous example being the

XOR problem. For this reason the idea of Perceptrons was dismissed for a lot of years, but it was revived much later with the idea of Multi-layer Perceptron and its great processing and representational power, more on that later. As can be seen in Figure 2.1 a vital part of the Perceptron is the activation function, which introduces a non-linearity which is very important, since it allows us to approximate arbitrarily complex functions. We will explore some basic activation functions in the next subsection.

Another idea of traditional ML, that is worth mentioning, is **Support Vector Machines (SVMs)**, one of the most robust prediction methods. The original algorithm was invented in 1963, but it got the form we use nowadays in 1995 [18]. One of the advantages of the SVMs is that they can efficiently classify non-linearly separable data, using what is called the kernel trick, implicitly mapping their inputs into high-dimensional feature spaces.

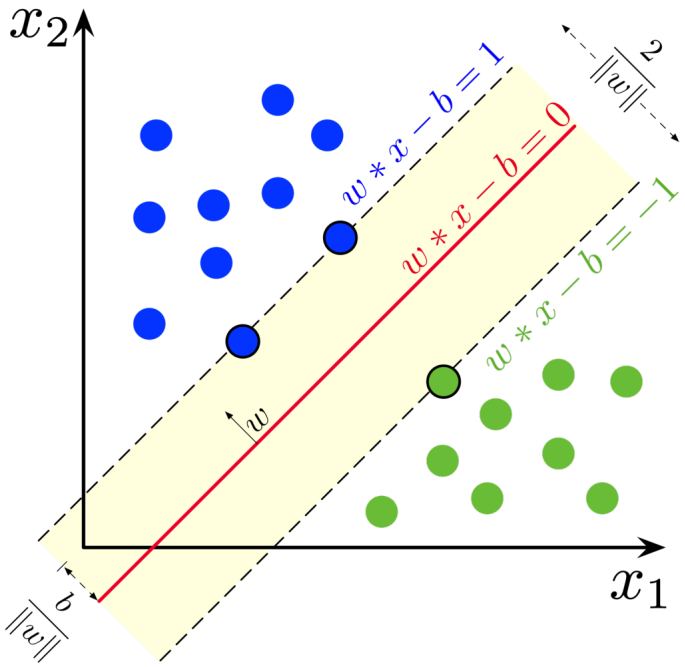


Figure 2.2: *An example of Support Vector Machines for classification. The red line is the maximum-margin hyperplane. (Source: Wikipedia)*

The basis of the SVM algorithm is that it constructs hyperplanes, to perform classification in the higher-dimensional space. Intuitively, to achieve a good separation, we expect to use the hyperplane that has the largest distance to the nearest training-data point of any class, because a larger margin, means a lower generalization error, when doing classification. If such a hyperplane exists, it is called the maximum-margin hyperplane.

In the general case, where data is not linearly separable, we use the *soft-margin* SVM algorithm. For input observations $\{x_1, x_2, ..., x_N\}$ with labels $\{y_1, y_2, ..., y_N\}$, where $y_i \in \{-1, 1\}$ we want to minimize the quantity:

$$\left[\frac{1}{n}\sum_{i=1}^{n}\max\left(0, 1 - y_i(\mathbf{w}^T\mathbf{x}_i - b)\right)\right] + \lambda\|\mathbf{w}\|^2 \tag{2.2}$$

where the parameter λ regards the trade-off between larger margin size and ensuring

that all data points lie in the correct side. Also, $\mathbf{w}$ is the normal vector to the hyperplane and b is the bias. These two define the predicted maximum-margin hyperplane. An example is seen in Figure 2.2.

2.2 Basics of Deep Learning

The distinction between Deep Learning and Machine Learning is arguably blurry. Deep Learning models are generally more complex and they involve a greater amount of learned function or concept composition than traditional Machine Learning does. The adjective "deep" refers to the usage of multiple layers in a network, but it is not limited to that. Also, Deep Learning is based on Neural Networks and Representation Learning in a greater extent, as we mention above. In this subsection we will discuss basic concepts (not limited to Deep Learning) and successful architectures that have been developed.

2.2.1 Feed Forward Neural Networks

Feed Forward Neural Networks (FFNs) was the first and simplest type of Artificial Neural Network devised. It can be thought therefore as the basis of Deep Learning. Another name for FFNNs is Multilayer Perceptrons (MLPs) and it reveals that they consist of multiple stacked layers of Perceptron units. These Perceptron units are connected without any feedback loops, in a Directed Acyclic Graph. One of the basic motivations and advantages of stacking Perceptron in a multi-layer network, is that if we introduce non-linearities through non-linear activation functions (more on that later), we are able to distinguish between non-linearly separable data.

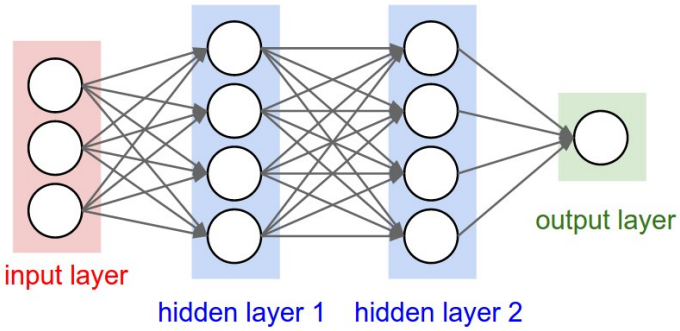


Figure 2.3: *A 3-layer feed forward neural network with three inputs, two hidden layers of 4 neurons each and one output layer. (Source: [38])*

Each Perceptron acts as a computational unit, taking as input the output of the previous layer. Each unit implements a function that converts the input vector to a scalar (as we mentioned above). A feed forward network consists of at least three layers of nodes: an input layer, a hidden layer and an output layer. The input layer takes as input the data, the hidden layers process the outputs of the previous layers and the output layer provides the final output. The flow of information from the input to the output is called forward propagation. Also typically, units of a single layer are not connected to each other. The number of hidden layers determines the depth of a model, while the

number of the units in the hidden layer determines the width of the model. An example can be seen in Figure 2.3.

FFNNs are applied with great success to many problem settings, either alone or as part of a more complex network. Apart from their experimental success they have also theoretical guarantees. The universal approximation theorem[4] states that every continuous function that maps intervals of real numbers to some output interval of real numbers can be approximated, arbitrarily closely, by a MLP with just one hidden layer and a sufficient number of neurons. This result requires a suitable activation function, but it holds for most that are used, one of the first proofs was for the sigmoid [20].

2.2.2 Activation Functions

As we discussed above, in order to classify non-linearly seperable data points with multiple layers, it is essential to introduce non-linearities. Non-linearities allow us to approximate arbitarily complex functions. We introduce them with activation functions. Activation functions take as input the output of a node and produce the final output, by making a non-linear decision. Common activation functions, introduce a first non-linearity at zero and some, like sigmoid, use a second non-linearity to restrict large inputs. If we add the activation function g to the single Perceptron, we can define a simple classifier with one node as follows:

$$\hat{y} = g(\mathbf{w} \cdot \mathbf{x} + b) \tag{2.3}$$

Some common activation functions are the following:

Sigmoid

The sigmoid or logistic function is defined as:

$$\sigma(z) = \frac{1}{1 + e^{-z}} \tag{2.4}$$

The sigmoid function is differentiable, defined for all real input values and has a non-negative derivative at each point. It takes a real-valued number and outputs a real number bounded in the range [0, 1]. In particular, large negative numbers become 0 and large positive numbers become 1. The sigmoid function has been widely used historically, because of the biological intuition of a neuron firing (fully for 1 and not at all for 0). Sigmoid functions are monotonic and have a first derivative which is bell shaped. The sigmoid function is convex for values less than 0, and it is concave for values more than 0.

In practice, the sigmoid is rarely used nowadays because of two drawbacks. First of all, when its value is close to 0 or 1, the gradient value is close to 0. This has the undesired effect of saturating or vanishing gradients. Secondly, the sigmoid output is not zero-centered. As a result, the input of the next neurons has always positive values. The

[4] https://en.wikipedia.org/wiki/Universal_approximation_theorem

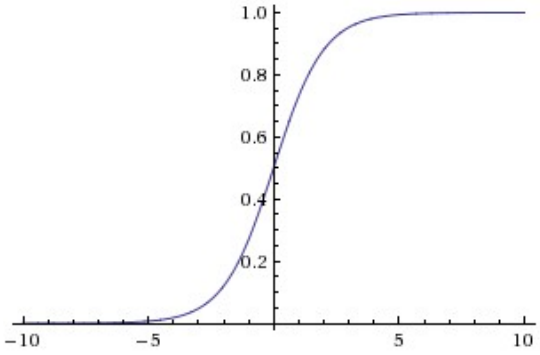


Figure 2.4: *A sigmoid non-linearity. All inputs are squashed in the range [0,1]. (Source: [38])*

gradient of weights will be either always positive, or always negative and an unwanted alternation of them is introduced into the network.

Hyperbolic Tangent (tanh)

The hyperbolic tangent or tanh function is defined as:

$$\tanh(z) = \frac{e^z - e^{-z}}{e^z + e^{-z}} \tag{2.5}$$

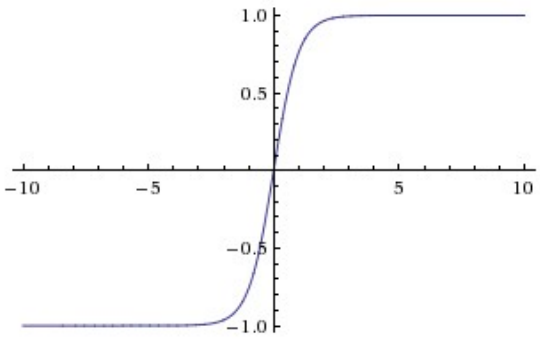


Figure 2.5: *A tanh non-linearity. All inputs are squashed in the range [-1,1]. (Source: [38])*

The tanh function is a scaled and shifted variation of the sigmoid. We can confirm that if we compare Equations 2.4 and 2.5. We get the relation $\tanh(z) = 2\sigma(2z) - 1$. So, similarly to the sigmoid, the hyperbolic tangent is a bounded, differentiable, real function. It is defined for all real input values and has a non-negative derivative at each point. It has the same drawback of gradients close to zero for large input values, but its advantage is that it is zero-centered.

Rectified Linear Unit (ReLU)

The ReLU activation function is described by the following simple mathematical form:

$$f(z) = max(0, z) \tag{2.6}$$

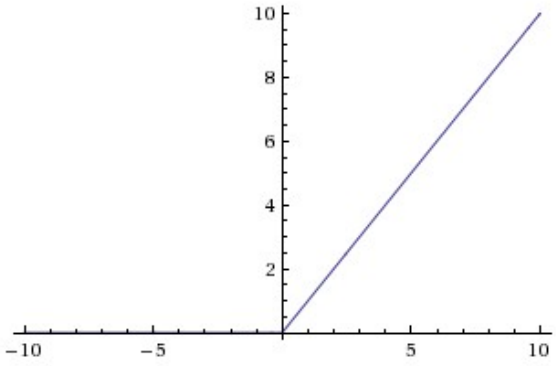


Figure 2.6: *A ReLU non-linearity. Simply, zero when $z < 0$ and then linear with slope 1 when $z > 0$. (Source: [38])*

This activation function simply thresholds the input at zero, keeping only positive inputs. It is otherwise known as the ramp function. It is one of the most commonly used activation functions nowadays. First of all, it does not involve any computationally expensive operations (like exponentials), in contrast to the sigmoid and tanh fuctions. This makes its computation extremely fast. Also, it is found to accelerate convergence by a large factor in many cases, compared to the previously mentioned functions. Moreover, it avoids the vanishing gradient problem and it is sparsely activated, making it more likely that neurons learn more meaningful aspects of the problem. Finally, it is scale-invariant.

Some disadvantages are the following. ReLU is not differentiable at zero, it is not zero-centered and unbounded. A common problem is the "dying ReLU" problem. ReLU units can be fragile during training and can “die”. This means that it is possible for neurons to be pushed into states that they become inactive for essentially all inputs. Then, no gradients flow backward through the neuron, and so the neuron becomes stuck in a perpetually inactive state. For example, if a high learning rate is used, as much as 40% of the network can be “dead”, meaning that so many neurons will never activate across the entire training dataset.

Leaky ReLU

Leaky ReLU is an attempt to fix the problem of "dying ReLU". It is described in mathematical form as:

$$f(z) = \begin{cases} z & \text{if } z > 0, \\ az & \text{otherwise} \end{cases} \tag{2.7}$$

where a is a small constant (commonly 0.01). Another variant called Parametric ReLU uses a learnable a coefficient instead. Instead of the function being zero when $z < 0$, a leaky ReLU will instead have a small negative slope.

Gaussian Error Linear Unit (GELU)

The GELU activation function [31] is used in some of the most recent state-of-the-art language model architectures, such as BERT [21] and GPT-2 [69], with great success. We also use it in our architectures. It is defined as:

$$\text{GELU}(z) = zP(Z \leq z) = z\Phi(z) = z \cdot \frac{1}{2}\left[1 + \text{erf}(z/\sqrt{2})\right] \tag{2.8}$$

if $Z \sim \mathcal{N}(0, 1)$. $\Phi(Z)$ is the cumulative distribution function of the standard normal distribution. An approximation of GELU is given by:

$$\text{GELU}(z) = 0.5z\left(1 + \tanh\left[\sqrt{2/\pi}\left(z + 0.044715z^3\right)\right]\right) \tag{2.9}$$

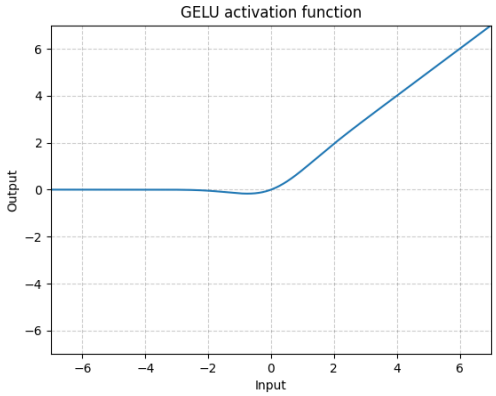


Figure 2.7: *A GELU non-linearity. (Source: PyTorch documentation)*

The GELU non-linearity weights inputs by their percentile, rather than gates inputs by their sign as in ReLUs. Consequently the GELU can be thought of as a smoother ReLU. The GELU function has a negative coefficient, which shifts to a positive coefficient. So when z is greater than zero, the output will be z, except when z is in the range [0, 1], where it slightly leans to a smaller output value. That formulation relates to stochastic regularizers because it is a modified expectation of adaptive dropout, providing neurons with more abstract probabilistic view. The GELU function largely avoids the vanishing gradient problem.

Softmax

The softmax function is simply a generalization of the logistic or sigmoid function to multiple dimensions. It is often used as the last activation function of a neural network to normalize its output to a probability distribution over some classes. It takes as input a vector z of K real numbers, and normalizes it into a probability distribution consisting of K probabilities proportional to the input exponentials. Given an input vector z and a weighting vector w, we have:

$$P(y = j \mid z) = \frac{e^{z^T w_j}}{\sum_{k=1}^{K} e^{z^T wk}} \tag{2.10}$$

2.2.3 Training

Artificial Neural Networks are trained by an optimization method, that aims to select a set of model parameters that minimizes the prediction error. We will discuss a way to quantify how well the model fits to the training data, some optimization algorithms and other notions and enhancements used in the training process.

Loss Functions

As we quickly discussed in Subsection 2.1.2, a way to measure the performance of supervised learning is the loss function. The loss function computes a non-negative value that measures the inconsistency between the predicted and the target output. So, if we denote our model as a function f with parameters w and the i-th training example pair as (x_i, y_i) the loss for each example is $\mathcal{L}(y_i, f(x_i; w))$. As we want to quantify the total loss over the entire dataset of size N, we want to minimize $J(w)$ (objective function or empirical risk), given by:

$$J(w) = \frac{1}{N} \sum_{i=1}^{N} \mathcal{L}(y_i, f(x_i; w)) \tag{2.11}$$

Next, we denote the predicted output as $\hat{y_i}^w$ instead of $f(x_i; w)$. For models that output a probability between 0 and 1, the **binary cross-entropy** loss is commonly used:

$$J(w) = -\frac{1}{N} \sum_{i=1}^{N} (y_i \log(\hat{y_i}^w) + (1 - y_i) \log(1 - \hat{y_i}^w)) \tag{2.12}$$

For the general case of multi-class output, the dissimilarity between the empirical distribution p and the predicted distribution q can be expressed by the cross-entropy H. This requires an output probability for each class and usually for this a softmax layer is used. The **cross-entropy** loss, for M classes, is given as:

$$J(w) = -\frac{1}{N} \sum_{i=1}^{N} H(p_i, q_i) = -\frac{1}{N} \sum_{i=1}^{N} \sum_{c=1}^{M} (y_{i,c} \log(\hat{y_{i,c}}^w)) \tag{2.13}$$

For regression tasks, in which a model outputs a real value instead of a probability, **mean squared error** loss can be used:

$$J(w) = \frac{1}{N} \sum_{i=1}^{N} (y_i - \hat{y_i}^w)^2 \tag{2.14}$$

More loss functions exist, but these are the most commonly used ones. Note that the performance of a model on a particular task is actually measured by another metric, different from the loss J. These performance measures, however, are often not suitable for gradient-based optimization. So, we minimize a selected loss function and expect that we will improve the performance of the model, by minimizing the loss.

Optimization

Training a model can be actually thought of as the optimization problem of finding the model parameters w that minimize the loss function $J(w)$ averaged across the training dataset. Most optimization algorithms are gradient-based. Let us denote a differentiable function $f: \mathbb{R}^n \to \mathbb{R}$, then its gradient $\nabla f: \mathbb{R}^n \to \mathbb{R}^n$ is defined, at $p = (x_1, ..., x_n)$ as the vector:

$$\nabla f(p) = \begin{bmatrix} \frac{\partial f}{\partial x_1}(p) \\ \vdots \\ \frac{\partial f}{\partial x_n}(p) \end{bmatrix} \tag{2.15}$$

The gradient vector can be interpreted as the *direction and rate of fastest increase*. Since, we want to minimize the loss function, we compute its gradient with respect to the weights of the model and then the weights of the model are updated in the opposite direction. We will now present some common optimization algorithms.

Gradient Descent (GD) is a very basic optimization algorithm and one of the most popular ones. It follows an iterative process. The model parameters w are randomly initialized and at each iteration the gradient of the loss function is computed for the entire dataset with respect to the parameters w and then the parameters are updated accordingly. Let $J(w)$ be the loss function over the entire dataset and $a \in \mathbb{R}$ be a small value (a hyperparameter), called learning rate. Then, the parameters of the model at the i-th iteration are updated as:

$$w_{i+1} = w_i - a\nabla_w J(w_i) \tag{2.16}$$

The Gradient Descent algorithm is very simple and effective, but it has a major drawback. Computing the loss over the entire dataset at each iteration is computationally expensive and inefficient, especially for large datasets.

Stochastic Gradient Descent (SGD) is a variant of Gradient Descent that solves the aforementioned problem. It computes the gradient of the loss function over a subset of samples, not for the whole dataset. It makes an estimation of the gradient, instead of computing the true gradient using all samples, therefore the name "stochastic".

In its simplest form, the gradient is computed over each unique training example, but this can lead to very noisy gradients and cause the loss function to fluctuate. For this reason, a variation called **mini-batch** SGD is commonly used in practice. A mini batch of B training data points is picked and the average gradient over those B points is calculated and used:

$$w_{i+1} = w_i - a\frac{1}{B}\sum_{b=1}^{B} \nabla_w J_b(w_i) \tag{2.17}$$

This method is still fast to compute and gives a much better estimate of the true gra-

dient. The larger the batch size, the more accurate the estimation of the gradient, which leads to smoother convergence and allows for larger learning rates.

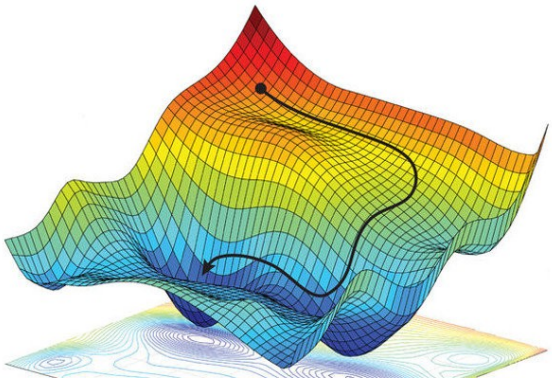


Figure 2.8: *The surface of a non-convex loss function. The arrow shows a path to reach the global minimum. (Source: Medium)*

It should be noted, that when we train a deep neural network, the loss surface becomes non-convex because of the introduced non-linearities. This means that there is no guarantee that a gradient based method will converge to a global minimum, where the loss function is zero. The **learning rate** becomes a very important training hyperparameter and effectively controls the speed at which the model learns. If the learning rate is too large, we may overshoot the minima and bounce back and forth on the loss surface or the model can become unstable and diverge. If it is too small, it may take too long to reach a minimum, or the algorithm might get stuck at a suboptimal local minimum.

In practice, learning rate is chosen based on experience and the model that we train. But, in order to deal more systematically with the problem of choosing a suitable learning rate, it is common to use different values during training. One way of doing this, is scheduling the rate based on the timestep. For example Linear Warmup is a technique that linearly increases the learning rate for the first K steps and then keeps it stable. This is shown to reduce volatility in the early stages of training. Learning Rate Decay, on the other hand, starts with a large rate and then slowly decreases it. The intuition behind this is that decaying the learning rate helps the network converge to a local minimum and avoid oscillation. Another way to change the learning rate is to choose an adaptive learning rate algorithm that calculates it depending on the gradient magnitude, the speed of learning, the size of particular weights or other measuers. Some famous adaptive gradient descent algorithms are Adam, Adadelta, Adagrad and RMSprop.

Adam [40] is one of the most commonly used optimization algorithms with adaptive learning rate. Adam is short for Adaptive Moment Estimation. It keeps a learning rate for every parameter in the network and separately adapts them during training. It uses estimations of first and second moments of gradient to adapt the learning rate for each weight. The n-th moment of a variable is the expected value of that variable to the power of n. We can formally define the model's and Adam's parameters update as:

$$w^{(i+1)} \leftarrow w^{(i)} - a\frac{\hat{m}_w}{\sqrt{\hat{v}_w} + \epsilon}$$

$$\begin{aligned}
\hat{m}_w &= \frac{m_w^{(i+1)}}{1-\beta_1^{i+1}} \\
\hat{v}_w &= \frac{v_w^{(i+1)}}{1-\beta_2^{i+1}} \\
m_w^{(i+1)} &\leftarrow \beta_1 m_w^{(i)} + (1-\beta_1)\nabla_w J^{(i)} \\
v_w^{(i+1)} &\leftarrow \beta_2 v_w^{(i)} + (1-\beta_2)(\nabla_w J^{(i)})^2
\end{aligned} \tag{2.18}$$

where m and v are the first and seconds moments respectively, a is the step size/learning rate, ϵ is a very small value (typically 1e-8) to avoid zero division and β_1 (typically 0.9), β_2 (typically 0.999) are the forgetting factors for gradients and second moments of gradients, respectively. Adam is well suited for noisy gradients, it is computationally efficient and has little memory requirements.

AdamW [48] is an improvement to the Adam algorithm. The basic idea of AdamW is that it modifies the typical implementation of weight decay [44] in Adam, by decoupling weight decay from the gradient update.

Weight decay, otherwise known as L_2 **Regularization**, penalizes larger model parameter (weight) norms, by adding an extra part to the loss: $J'(w) = J(w) + \lambda w^T w$, with λ being a hyperparameter. It is a regularization technique, used to mitigate the problem of overfitting, by encouraging smaller weights and therefore less complex functions. **Overfitting** is a common training problem. It happens when a model learns too much detail and therefore noise in the training data, to the extent that it negatively impacts the performance on unseen data.

AdamW adjusts the weight decay term to appear in the gradient update:

$$w_{i+1,j} = w_{i,j} - a\left(\frac{1}{\sqrt{\hat{v}_i} + \epsilon} \cdot \hat{m}_i + \lambda_{i,j} w_{i,j}\right) \tag{2.19}$$

The weight decay or regularization term does not end up in the moving averages and is thus only proportional to the weight itself. The authors show experimentally that AdamW yields better training loss and that the models generalize much better than models trained with Adam.

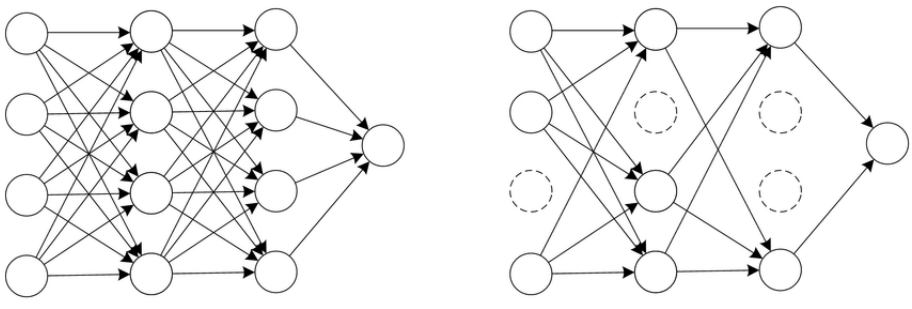
Backpropagation

As we discussed above, in order to use a gradient based optimization algorithm, it is essential to compute the gradient of the loss function with respect to the weights of the network. Every neural network can be illustrated as a directed graph where each neuron corresponds to a node and each weight to an edge. The backpropagation algorithm [45] [77] computes the gradient of the loss function for each input-output pair, with respect to each weight. It is the backbone of training neural networks.

Computing the gradient for a network is not a trivial step, especially for large and complex networks. Fortunately, the backpropagation algorithm can efficiently compute the gradients for all parts of the network. It works by computing the gradient of the function with respect to each weight, using the chain rule. It computes the gradient one layer at a time, iterating backward from the last layer and caching intermediate results to avoid redundant calculations. Its idea is based on dynamic programming.

Dropout

Dropout [34] is another regularization method for neural networks. Its basic idea is very simple, yet it is very effective and commonly used. During training, each unit is dropped with a probability $p > 0$, meaning that it ignored during both the forward and the backward pass, while its activation becomes zero. This is illustrated in Figure 2.9. During inference, all neurons are used and no unit is dropped, but they are scaled by a factor $frac1p$ to account for the missing activations during training. Dropout forces the network to not rely on specific nodes and prevents units from forming co-dependencies amongst each other. It can otherwise be thought of as a way to perform model averaging.



(a) Standard Neural Network (b) Network after Dropout

Figure 2.9: *Using dropout during a training step. (Source: [7])*

Normalization

Training a deep neural network is complicated by the fact that the distribution of each layer's inputs changes during training, as the parameters of the previous layers change. The problem that arises is called internal covariate shift and as a result it requires lower learning rates, making training slower, and careful parameter initialization, making it harder to train models with saturating non-linearities. A way to address this problem is to normalize layer inputs.

Batch Normalization [37] is a method that performs this normalization for each training mini-batch. A normalization step fixes the mean and variance of each layer input. It improves gradient flow through the network, reducing the dependence of gradients on the scale of the parameters and their initial values. It allows to use higher learning rates and permits less careful initialization. It can also act as a regularizer, eliminating the need for dropout, in some cases.

Let us denote b a mini-batch of size B of the training dataset. The empirical mean and variance of b could thus be denoted as:

$$\mu_b = \frac{1}{B}\sum_{i=1}^{B} x_i \quad and \quad \sigma_b^2 = \frac{1}{B}\sum_{i=1}^{B}(x_i - \mu_b)^2 \tag{2.20}$$

For a layer of the network with input $x = (x^{(1)}, ..., x^{(d)})$, each dimension of its input is normalized separately, as:

$$\hat{x}_i^{(k)} = \frac{x_i^{(k)} - \mu_b^{(k)}}{\sqrt{\sigma_b^{(k)^2} + \epsilon}} \tag{2.21}$$

where $k \in [1, d]$ the dimension and $i \in [1, B]$ the sample of the mini-batch b. ϵ is a small constant to avoid zero divisions. The new normalized activation $\hat{x}^{(k)}$ has zero mean and unit variance. To restore the representation power of the network, a final transformation step follows:

$$y_i^{(k)} = \gamma^{(k)}\hat{x}_i^{(k)} + \beta^{(k)} \tag{2.22}$$

with parameters $\gamma^{(k)}$ $\beta^{(k)}$ being learned in the optimization process.

Layer Normalization [3] is another normalization technique that was introduced to normalize the training of recurrent networks, where batch normalization falls behind. It also solves the challenging task of selecting a suitable mini-batch size and omits the dependence between training examples. Layer normalization is, in a sense, the transpose of batch normalization, since it computes the mean and variance from all of the summed inputs to the neurons in a layer on a single training case. Unlike batch normalization, layer normalization performs exactly the same computation at training and test time. Layer normalization is widely used, since it stabilizes the hidden state dynamics in recurrent networks, reduces the training time and generalizes well.

The layer normalization statistics (mean and variance) are computed over all the hidden units in the same layer as follows:

$$\begin{aligned} \mu^l &= \frac{1}{H}\sum_{i=1}^{H} a_i^l \\ \sigma^l &= \sqrt{\frac{1}{H}\sum_{i=1}^{H}\left(a_i^l - \mu^l\right)^2} \end{aligned} \tag{2.23}$$

where H is the number of hidden units in a layer. Under layer normalization, all the hidden units in a layer share the same normalization terms μ^l and σ^l, but different training cases have different normalization terms. After computing the normalization statistics we follow the same procedure as in batch normalization.

Residual Connections

Residual or skip connections are paths (shortcuts) in neural networks that jump over one or more layers. They have been introduced in [30] to create an architecture called Residual Neural Networks (ResNets). This addition made ResNets the state-of-the-art model in many Computer Vision tasks, and since then the residual connections have been an important component in a wide variety of architectures.

The main motivation behind the development of residual connections was to combat the problem of vanishing gradients, that made the training of deep architectures notoriously difficult. As we discussed above, the backpropagation algorithm is used to compute the gradient for all the layers of a network, using the chain rule. When there are many layers with gradients less than 1, the total product approaches zero, making learning infeasible. By using a skip connection, an alternative path for the gradient is provided. These connections simply use the identity function and add the original input to the output of the layers they skip. This can be seen in Figure 2.10. Supposing a layer (or more) perform the function $\mathcal{F}$, then the output with a residual connection, given an input x, will be:

$$\mathcal{F}(x) + x$$

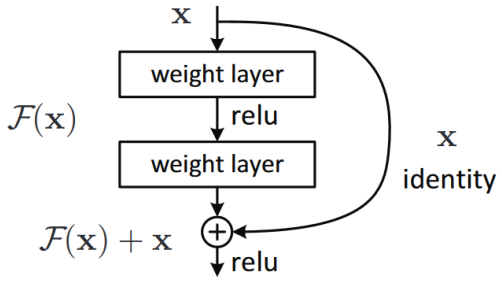


Figure 2.10: *A residual connection skipping two layers. (Source: [30])*

Except from providing an uninterrupted gradient flow from the first layer to the last, residual connections enable feature reusability and stabilize training and convergence. Skipping effectively simplifies the network, using fewer layers in the initial training stages, while the skipped layers are gradually restored, as the model learns the feature space.

2.3 Deep Learning for Natural Language Processing

2.3.1 Overview of the Field

Natural Language Processing (NLP) is a subfield of computer science, artificial intelligence and linguistics, concerned with the interactions between computers and human (natural) languages. NLP deals with extracting meaningful information from natural language input and producing natural language output. It analyzes human language based on semantics and employs various parsing techniques to achieve that. Analysis of language has as purpose the production of meaningful representations, while generation aims to produce (language) text from a representation. NLP deals not only with written

language, but also spoken language, for example speech recognition is considered an NLP problem.

The roots of NLP can be traced back to the 1950s. In 1950, Alan Turing published the article "Computing Machinery and Intelligence", which proposed as a criterion of intelligence a task that involves the automated interpretation and generation of natural language, today known as Turing test. This started the so called **Symbolic NLP** phase, which lasted until the early 1990's. The idea of Symbolic NLP can be summarized in the *Chinese room experiment*[5], a hypothetical setting devised by John Searle. *Given a collection of rules (for example a Chinese phrasebook, with questions and matching answers), the computer emulates natural language understanding (and other tasks) by applying those rules to the data it is confronted with.* This means that a computer has no meaningful representations of words or other notions. Instead, its understanding or generation of answers is emulated, by following a set of carefully crafted (hard-coded) rules.

The next phase is called **Statistical NLP** and it was a big step towards achieving something that resembles understanding of language. With the introduction of machine learning algorithms, a revolution started in the field of NLP. Statistical NLP, instead of focusing on grammar or other rules, focused on existing language data. This approach is statistical and inductive in nature. Instead of using rules, a computer could use statistical inference to automatically learn such rules through the analysis of large corpora of typical real-world examples.

The current phase of NLP is called by many **Neural NLP**, because of the usage of neural networks and deep learning instead of traditional ML methods, like in other related fields. It started in the 2010's, with the widespread usage of representation learning based methods and currently achieves state-of-the-art results. The paradigm has not changed significantly, patterns are inferred from language text, but there is no need for elaborate feature engineering, since meaningful representations of words are inferred directly from data. This led to many tasks that were previously solved as a chain of different subtasks to be solved directly, in an end-to-end manner. For instance, the term "neural machine translation" (NMT) emphasizes the fact that neural NLP directly learns sequence-to-sequence transformations, obviating the need for intermediate steps such as word alignment and language modeling that were used in statistical machine translation (SMT).

2.3.2 Recurrent Neural Networks

Recurrent Neural Networks (RNNs) are a type of artificial neural networks in which the connections between nodes form a directed graph along a temporal sequence. Unlike feedforward neural networks, that operate under the assumption that all training instances are independent, RNNs produce their output taking into account previously presented information. This allows them to exhibit temporal dynamic behavior and for this reason RNNs are used to model sequential data (for example text or audio). The term

[5] https://en.wikipedia.org/wiki/Chinese_room

"recurrent" refers to the way they process a sequence, performing the same task for every element. The basic idea of RNNs is to infer the temporal dynamics of the data sequence by keeping an internal state, also known as hidden layer (or state), which they update on each time-step, for each token of the sequence.

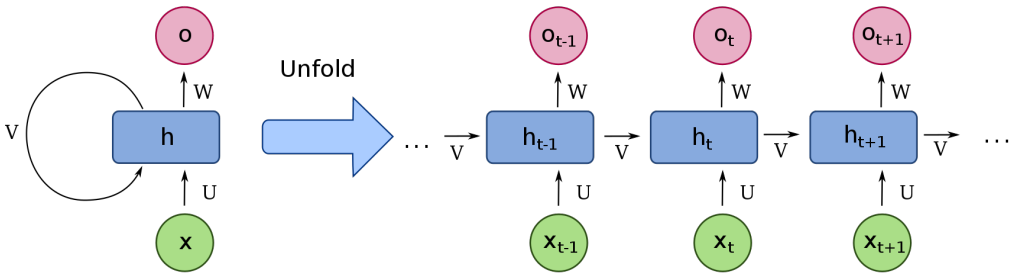


Figure 2.11: *An RNN looping through time (left) and the same network unfolded over time (right). (Source: Wikipedia)*

An RNN is essentialy a very simple network. At each timestep t, it receives an input vector x_t and a hidden state from the previous timestep h_{t-1} and produces a new hidden state h_t and an output o_t. The hidden state at each timestep is updated based on the previous hidden state and the input vector, while the output depends on the current hidden state. To better visualize this, we can "unfold" the model over time, as in Figure 2.11. Then we can see it as multiple copies of the same network, each passing hidden state information to its successor. The hidden state h_t and the output o_t for each timestep t are computed as:

$$\begin{aligned} h_t &= f_h(V_h h_{t-1} + U_h x_t + b_h) \\ o_t &= f_o(W_o h_t + b_o) \end{aligned} \tag{2.24}$$

where f_h, b_h, and f_o, b_o are activation functions and biases for the hidden state and the output respectively, while U_h, V_h and W_o are learnable weight matrices. One major advantage of RNNs is that they can process sequences of arbitrary lengths, wnd while doing that the size of the model remains the same, since the same weights are applied at each timestep.

Some basic modifications of this architecture are the following:

Deep RNNs are simply created by stacking multiple RNNs on top of each other, with the output of each layer being the input to the next one. The inspiration for this was the rise of deep neural networks, and the intuition behind it is that a deep RNN can capture better the hierarchy present in the input sequences.

Bidirectional RNNs were developed to tackle a drawback in the simple RNN architecture. The simple architecture conditions only on previous input tokens to predict the current

output. This is enough (or even necessary) for some tasks, but insufficient for others, such as encoding a sentence. It is possible to use the context of the whole sequence, not just the previous tokens, by adding an RNN that reads backwards. A combination of a forward and backward RNN is called bidirectional. At each timestep it maintains two hidden states, one ($\overrightarrow{h_t}$) for the left-to-right and another ($\overleftarrow{h_t}$) for the right-to-left propagation. These are then concatenated into h_t to compute the output o_t at the current timestep.

RNNs have a very basic problem, that appears when dealing with long sequences. In order to train the network, the gradient of the loss needs to be propagated not only through the depth of the model but also through previous timesteps (backpropagation through time). A common issue is that these long-term dependencies cause the gradients to become either very large (explode) or very small (vanish). This makes learning very hard and the model prone to degeneration. To mitigate this issue, two popular architectures were proposed which we will shortly present below.

Long Short-Term Memory

Long Short-Term Memory (LSTM) [35] is a type of recurrent neural network that addresses the vanishing gradient problem in RNNs through additional cell states and gates. LSTMs are able to effectively learn long-term dependencies and have largely replaces the simple RNN architecture. Intuitively, they solve the vanishing gradient problem through additional additive components, and forget gate activations, that allow the gradients to flow through the network without vanishing as quickly.

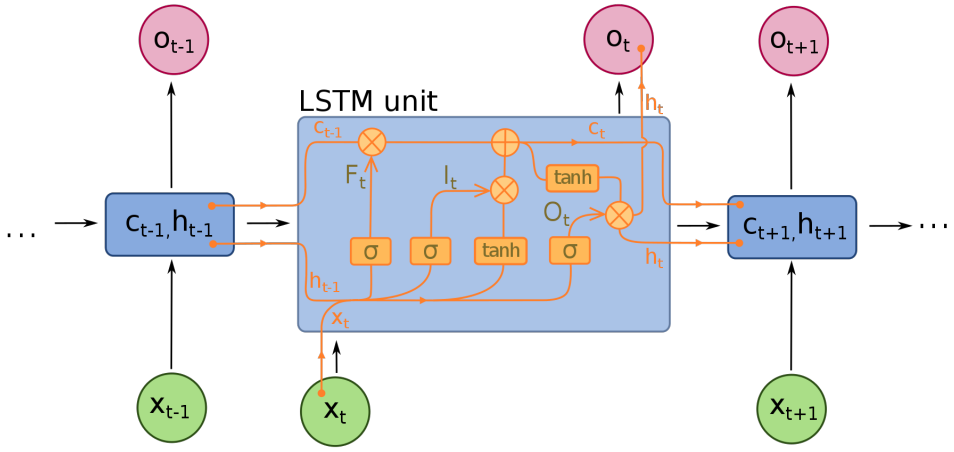


Figure 2.12: *The architecture of an LSTM unit. (Source: Wikipedia)*

The internal architecture of an LSTM unit (cell) is depicted in Figure 2.12. We will shortly describe the additional components and their function. An LSTM cell contains:

- A **forget gate (f_t)**, which controls which information will be be discarded from the cell state. The current input x_t and the previous hidden state h_{t1} pass through a sigmoid activation, in order to scale the values between 0 and 1. Values closer to 0 will be forgotten, while values closer to 1 will be kept in the cell state.

- An **input gate (i_t)**, which controls which values of the input are important and will be used to the new cell state.
- An **output gate (o_t)**, which controls how much information of the cell should be passed to the output and hidden state, based on the current input and previous hidden state.
- A **memory cell (c_t)**, which is updated based on past information and the current input. The forget gate and the input gate filter the past and new information, before it enters the memory cell.
- A **hidden state (h_t)**, which is used to encode all the past information of the sequence.

To formulate the function of and LSTM unit and better show how the above components interact, we show how to compute the above for the timestep t:

$$\begin{aligned} f_t &= \sigma(W_f x_t + U_f h_{t-1} + b_f) \\ i_t &= \sigma(W_i x_t + U_i h_{t-1} + b_i) \\ o_t &= \sigma(W_o x_t + U_o h_{t-1} + b_o) \\ \tilde{c}_t &= \tanh(W_c x_t + U_c h_{t-1} + b_c) \\ c_t &= f_t \odot c_{t-1} + i_t \odot \tilde{c}_t \\ h_t &= o_t \odot \tanh(c_t) \end{aligned} \tag{2.25}$$

where W and U are the weight matrices to be learned (different for f, i, o and c) and $\odot$ denotes the element-wise product between vectors (Hadamard product).

Gated Recurrent Units

Gated Recurrent Units (GRU) were introduced in [16] and are a variant of the original LSTM architecture. It actually has fewer parameters than an LSTM, since it omits the output gate and the cell state, but has an equivalent performance on most tasks. GRUs effectively solve the vanishing gradient problem using update and reset gates, which decide what information should pass to the output.

The **update gate** z_t acts like the input and forget gates of the LSTM. It is responsible for determining what percentage of $ht1$ should be propagated to the next timestep. On the other hand, the **reset gate** r_t determines how much of the past information should the model forget. The inner architecture of a GRU unit can be seen in Figure 2.13 and the equations that describe its function are:

$$\begin{aligned} z_t &= \sigma(W_z x_t + U_z h_{t-1} + b_z) \\ r_t &= \sigma(W_r x_t + U_r h_{t-1} + b_r) \\ \tilde{h}_t &= \tanh(W_h x_t + U_h(r_t \odot h_{t-1}) + b_h) \\ h_t &= (1 - z_t) \odot h_{t-1} + z_t \odot \tilde{h}_t \end{aligned} \tag{2.26}$$

where W and U are learnable weight matrices (for z, r and h).

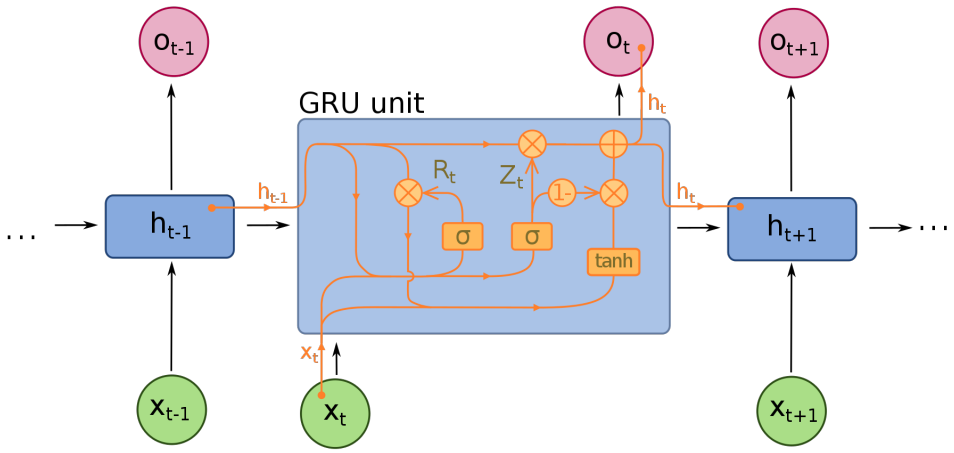


Figure 2.13: *The architecture of a GRU unit. (Source: Wikipedia)*

2.3.3 Sequence-to-Sequence Modelling

Sequence-to-sequence (seq2seq) is a family of machine learning approaches used in NLP and other fields. It deals with the task of generating an output sequence given an input sequence, when the two sequences have different lengths and not an explicit one-to-one correspondence. For example, one very typical task that is solved using a seq2seq approach, is Neural Machine Translation (NMT). Other tasks that are solved in a similar manner are Document Summarization, Conversational Models and Image Captioning, to name a few.

Since the two sequences do not have the same structure, using simply a sequential architecture, like the RNN we described above, is not sufficient. To deal with this, the seq2seq architecture was introduced [84]. Seq2seq is based on an Encoder-Decoder framework. The encoder encodes the input sequence into a hidden (contextualized) representation, while the decoder takes this representation as input to generate the final output. In the original implementation, RNNs (specifically deep LSTMs) are used as the encoder and the decoder, while the encoded representation is a vector of fixed dimensionality.

If we denote the input sequence as $x_1, \ldots, x_n$, the output sequence as $y_1, \ldots, y_m$ and the fixed size vector as c (context vector), we can formally express the task of generating the output sequence given the input, by the conditional probability:

$$P(y_1, \ldots, y_m \mid x_1, \ldots, x_n) = \prod_{i=1}^{m} P(y_i \mid c, y_1, \ldots, y_{i-1}) \tag{2.27}$$

In more detail, the encoder RNN processes the input sequence one token at a time and the final hidden state is used as the context vector c. Then, the hidden state of the decoder RNN is initialized with c. The first input to the decoder is a special token called <bos> (beginning of sequence) and the next inputs are the outputs of the previous timestep. Another technique that is used during training, called **teacher forcing**, uses the target sequence tokens as input instead of the predicted output and it usually leads to faster convergence. During inference (or test time) the first technique is used, since

there is no ground truth.

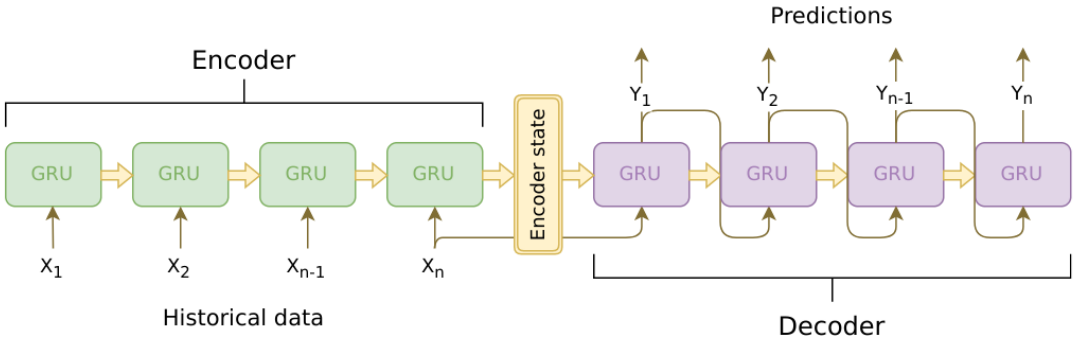


Figure 2.14: *The encoder-decoder framework. The RNN units are unfolded over time. (Source: Github)*

While this approach is very helpful when solving tasks without making assumptions on the sequence structure, it has a major drawback. The input can only be accessed through the context vector. Encoding a sequence of arbitrary length into a fixed-size representation creates an information bottleneck. Also, when dealing with long sequences, information in the beginning can be significantly forgotten by the time the last token is processed.

2.3.4 The Attention Mechanism

The attention mechanism was introduced in [2] for the task of Neural Machine Translation. It improves the performance of the aforementioned encoder-decoder architecture, by solving the bottleneck problem that arises when using a fixed size context vector. Instead of discarding the intermediate hidden states of the encoder and using its final state as the context vector, the attention mechanism develops a dynamic context vector by combining all the encoder hidden states. This also creates shortcuts between the context vector and the entire source input, which solves the problem of forgetting too. Attention can be interpreted as the ability to focus on relevant tokens of the input by computing weights of importance for their representations. This addition made the NMT approach competitive with previous methods such as statistical MT. Since then, it is widely used in most state-of-the-art architectures, not limited to NLP.

Formally, we have an input sequence $x_1, \ldots, x_n$ and an output sequence $y_1, \ldots, y_m$. The encoder is a bidirectional RNN with hidden state h_i (the concatenation of $\overrightarrow{h_t}$ and $\overleftarrow{h_t}$) at timestep i and the decoder is a unidirectional RNN with hidden state $s_t = f(s_{t-1}, y_{t-1}, c_t)$ at timestep t. The context vector c_t now is dynamic and is given by the following equation:

$$c_t = \sum_{i=1}^{n} a_{t,i} h_i \tag{2.28}$$

The weight $a_{t,i}$ refers to the pair (x_i, y_t) and is a measure of how well they match. It is computed by:

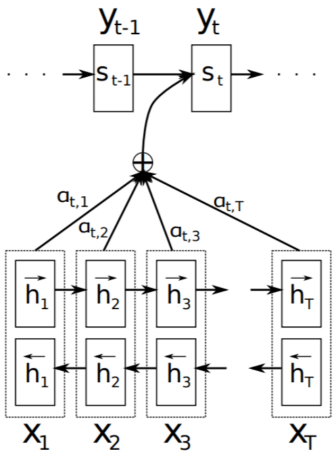


Figure 2.15: *The attention mechanism of the original paper. (Source:[2])*

$$a_{t,i} = \text{softmax}(e_{t,i}) = \frac{\exp(e_{t,i})}{\sum_{j=1}^{n} \exp(e_{t,j})} \tag{2.29}$$

where

$$e_{t,i} = \text{score}(s_{t-1}, h_i)$$

is the score function that measures the alignment. There are many choices for an alignment score function. In the original paper the score function is implemented by a feedforward neural network with a single hidden layer. This network is jointly trained with the rest of the model. In this case, using a tanh activation function, the score is:

$$\text{score}(s_t, h_i) = \mathbf{v}_a^\top \tanh(\mathbf{W}_a[s_t; h_i]) \tag{2.30}$$

where $\mathbf{v}_a$ and $\mathbf{W}_a$ are learned weight matrices. This type of attention mechanism is called **additive attention**. An overview of alternative attention mechanisms and the corresponding score function can be found in Table 2.1.

Name	Score Function	Introduced In
Content-base attention	$\text{score}(s_t, h_i) = \text{cosine}[s_t, h_i]$	[28]
Additive	$\text{score}(s_t, h_i) = \mathbf{v}_a^\top \tanh(\mathbf{W}_a[s_t; h_i])$	[2]
Location-Based	$a_{t,i} = \text{softmax}(\mathbf{W}_a s_t)$	[49]
General	$\text{score}(s_t, h_i) = s_t^\top \mathbf{W}_a h_i$	[49]
Dot-Product	$\text{score}(s_t, h_i) = s_t^\top h_i$	[49]
Scaled Dot-Product	$\text{score}(s_t, h_i) = \frac{s_t^\top h_i}{\sqrt{n}}$	[86]

Table 2.1: *A summary table of popular attention mechanisms in chronological order*

2.3.5 Transformer

The Transformer is a deep neural network architecture introduced in [86], providing a seq2seq approach to the task of NMT. Inspired by the success of the attention mechanism and trying to combat the slow training speeds enforced by the sequential nature of recur-

rent networks, the Transformer relies only on attention and so is able to process data in parallel. This makes it much faster than any sequential architecture, while it performs much better.

The Transformer is an encoder-decoder architecture. The encoder and the decoder consist of N stacked layers, where the output of each layer is the input to the next. The layers are identical, with different weights. Each encoder layer consists of a self-attention and a fully connected feedforward sublayer interleaved by a residual connection and layer normalization. The decoder has the same structure, with the addition of a cross-attention sublayer between the self-attention and the feed-forward. The self-attention sublayer is capable of creating a contextualized representation of the sequence, by analyzing the dependency between tokens of the sequence. The cross-attention sublayer is responsible for analyzing the dependency between the input and the output sequences. It should be noted that the self-attention in the encoder is "bidirectional", while the self-attention of the decoder is "unidirectional". This is implemented by using triangular masking and its intention is to avoid attending to future tokens of the sequence, which has adverse effects when predicting the current token. The cross-attention sublayer of each decoder layer is conditioned by the last layer output of the encoder. The ouput of the last decoder layer is finally converted to token probabilities, using a linear transformation and a softmax.

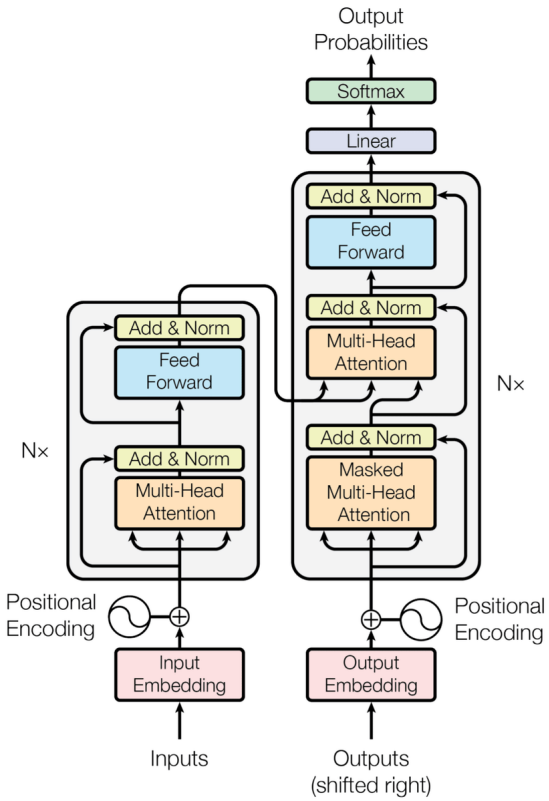


Figure 2.16: *The Transformer Encoder-Decoder Architecture. (Source: [86])*

The Feed Forward sublayer is fully connected and has a hidden layer with dimensionality d_{ff} (usually $d_{ff} = 4dmodel$). It can be thought of as two linear transformations with a non-linearity in between (ReLU in the original paper) applied to each position separately

and identically:

$$FFN(x) = max(0, xW_1 + b_1)W_2 + b_2 \tag{2.31}$$

W_1 and W_2 are weight matrices learned by the model. This step performs an important non-linear transformation, which creates a new representation based on the previous attention sublayers. While experimentally it has been proven very helpful, its exact function is still being researched [25].

Positional Encodings are used to give a sense of order in the sequences, since the Transformer is not a sequential architecture. They are ultimately a mapping of the sequence index to a vector. In the original paper, sinusoidal functions of different frequencies were used (fixed embedding), while a popular variation is using simple learned embeddings based on the index.

Scaled Dot-Product Attention is the backbone of the Transformer. We will first describe how it is applied in cross-attention and then we will show how it can be used for self-attention too.

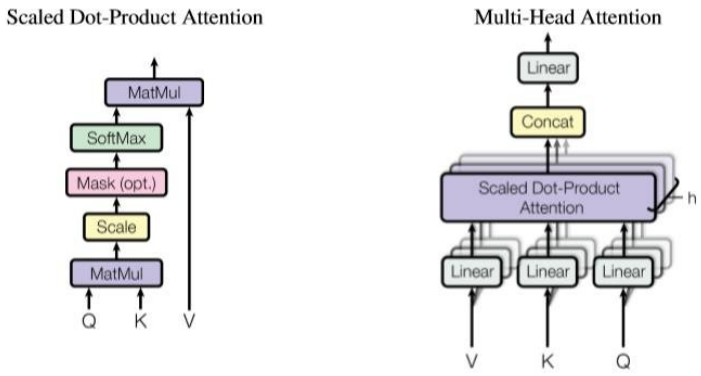


Figure 2.17: *Schematics of Scaled Dot-Product Attention (left) and Multi-Head Attention with h attention heads (right). (Source: [86])*

We can think of the attention function as mapping a query and a set of key-value pairs to an output. The output is computed as a weighted sum of the values, where the weight of each value is computed by a compatibility (score) function between the query and the corresponding key (similar to Subsection 2.3.4). At first, the key K and value V matrices are created from the input and the query Q matrix is created from the output. They are computed by multiplying them with learned weight matrices. The output is computed as:

$$Attention(Q, K, V) = softmax(\frac{QK^\top}{\sqrt{d_k}})V \tag{2.32}$$

where d_k is the dimension of keys. The $\frac{1}{\sqrt{d_k}}$ scaling factor avoids too large inputs to the softmax, which lead to extremely small gradients.

Instead of performing the above attention function once, the d-dimensional queries, keys and values are projected with h different, learned, linear projections to dimensions d_q, d_k and d_v. The attention function is applied h times in parallel and then combined. Each application of the attention function is an attention head, while this method is called **Multi-Head Attention**. Formally, given the weight matrices $W_i^Q \in \mathbb{R}^{d \times d_q}$, $W_i^K \in \mathbb{R}^{d \times d_k}$, $W_i^V \in \mathbb{R}^{d \times d_v}$, $W^O \in \mathbb{R}^{hd_v \times d}$ the multi-head attention is given by:

$$\begin{aligned} \text{MultiHead}(Q, K, V) &= [\text{head}_1; \ldots; \text{head}_h]W^O \\ \text{head}_i &= \text{Attention}(QW_i^Q, KW_i^K, VW_i^V) \end{aligned} \tag{2.33}$$

In the original paper, $h = 8$ and $d_q = d_k = d_v = d/h = 64$. All the above weight matrices are learned by the model.

Note that, in cross-attention, K and V come from the last layer of the encoder and Q comes from the previous decoder layer, while in self-attention, simply all three matrices come from the same part (encoder or decoder).

Transformer Variants

Since its appearance, the Transformer has revolutionized the wider field of Natural Language Processing and became the state-of-the-art architecture for many NLP tasks. It has even been used with great success for image and speech. Many variations of the original architecture exist for seq2seq modelling, but it is worth mentioning a line of research that keeps only one of the two parts (encoder or decoder) and trains them with a semi-supervised objective.

The first approach keeps only the encoder part to create a language representation model. The most successful work in this direction is **BERT** [21]. Its basic idea is that by pretraining a bidirectional encoder on unlabeled data, meaningful representations can be derived, which can later be used (with a little finetuning and some extra layers) on downstream tasks, such as question answering or summarization. The task of masked language modelling (hiding a word and predicting it based on context) and next sentence prediction were devised to pretrain the model. The success of BERT led to many similar models being developed (it even created a field of study called BERTology). Also, pretraining large language models on huge corpora of data (usually by large companies) and then making their weights publicly available, to be finetuned on specific tasks became the new paradigm.

Another line of work is focused on language models for text generation (unidirectional or causal). The most successful work has been the **GPT-**x ($x \in \{1, 2, 3\}$) architecture [68] [69] [9]. All three models are quite similar, with an increasing number of parameters, larger training datasets and few enhancements. In contrast to BERT, GPT keeps the decoder part of the model, which is trained on the task of predicting a token based on the previous tokens of the sequence. It uses masked self-attention like the original decoder. Pretrained models on huge language corpora are made publicly available, following the paradigm of BERT. GPT is usually finetuned to generate text with specific attributes, like

poetry or movie reviews (domain adaptation). It can also be finetuned for other tasks such as translation or question answering, with or without adding extra parts to the model. It is considered by many as a general-purpose learner and the latest models can generate text indistinguishable from text written by humans.

2.4 Overcoming Memory Constraints

2.4.1 The Pursuit for Efficient Attention

One major drawback of the attention mechanism is that it is inefficient for long sequences. Suppose we have a sequence of L tokens and the dimension of the Transformer model is d. Also, the dimensions of matrices Q, K and V are $L \times d$ (in the simple case). To compute the attention matrix given by the Equation 2.32, we have to compute the $QK^\top$ matrix, which is of size $L \times L$. So, the memory usage and time complexity of the Transformer scales quadratically with L making it unfeasible to use it for sequences longer than 1024 tokens, practically. Notice, that this multiplication has to be computed because of the softmax, otherwise we could make the $K^\top V$ multiplication first, which is of size $d \times d$.

Recently, many solutions have been many proposed to address the above drawback. Most solutions make assumptions about the attention matrix to reduce complexity. Instead of regular dense attention, [15] uses sparse attention with $O(L\sqrt{L})$ complexity, [42] uses locality-sensitive hashing to group together tokens with similar embeddings and reduces complexity to $O(L \cdot \log L)$ and [39] substitutes softmax with low-rank kernels to achieve $O(L)$ complexity.

2.4.2 Performer - FAVOR+ Attention

Performer [17] is a very recent architecture that manages to achieve linear complexity by accurately estimating full-rank softmax attention, without relying on any priors or changing the attention mechanism, like previous approaches. This is succeeded by using a novel approach called *Fast Attention Via positive Orthogonal Random features* (FAVOR+), which provably can be used to make a robust and unbiased estimation of attention.

To formally show how this is possible, we can rewrite Equation 2.32 as:

$$Att(Q, K, V) = D^{-1}AV, \quad A = \exp(\frac{QK^\top}{\sqrt{d_k}}), \quad D = diag(A1_L). \tag{2.34}$$

with $exp()$ applied elementwise, 1_L being an all-ones vector of length L and $diag()$ a diagonal matrix with the input vector as the diagonal.

Matrix A is of the form $A(i,j) = K(q_i^\top, k_j^\top)$, q_i being the i^{th} query and k_j being the j^{th} key row and K the kernel: $K(x,y) = \mathbb{E}[\varphi(x)^\top \varphi(y)]$. By using the positive orthogonal random features mapping $\varphi()$, we have Q' and K' with rows given as $\varphi(q_i^\top)^\top$ as and $\varphi(k_i^\top)^\top$ respectively. Now matrix A does not have to be computed and by reordering the computations, linear complexity on L is achieved, as shown in:

$$\widehat{Att}(Q, K, V) = \widehat{D}^{-1}(Q'((K')^\top V)), \quad D = diag(Q'((K')^\top 1_L). \tag{2.35}$$

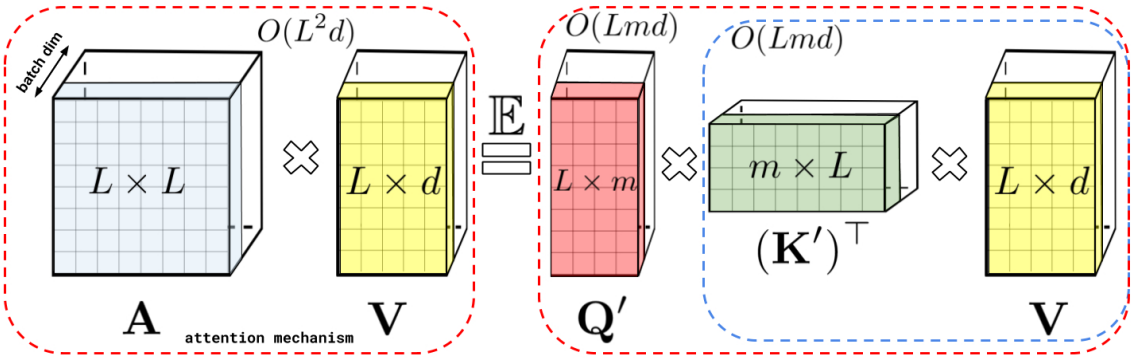


Figure 2.18: *The Performer Architecture - approximation via random features to avoid computation of A (source: [17])*

The authors show experimentally that the Performer achieves similar results to the original Transformer architecture, using the same hyperparameters. Also, it is shown that loading the weights of a pretrained Transformer model to this new architecture is possible, but a small finetuning is necessary to recover the original accuracy. Finally, redrawing the random features periodically is crucial to get a good approximation of the attention mechanism.

2.4.3 Reversible Layers

Reversible layers were introduced in [26] and they focus on reducing the memory consumption of deep neural networks. They are heavily inspired by deep residual networks and provide a simple but clever method to minimize the memory requirements for activation storage.

When using gradient descent (or other optimization algorithms) to train a model with N layers, the activations of all layers are saved to be used in the backward pass. With reversible layers, one needs only the last activations, which can be used to recover the activations of all previous layers, during back-propagation.

To achieve this, a reversible layer uses pairs of inputs (x_1, x_2) and outputs (y_1, y_2) and two residual functions $\mathcal{F}$, $\mathcal{G}$. The outputs, during the forward pass, can be computed as:

$$y_1 = x_1 + \mathcal{F}(x_2), \qquad y_2 = x_2 + \mathcal{G}(y_1) \tag{2.36}$$

This way a layer can be reversed simply by subtracting the residual functions. The inputs of the layer, during the backward pass, can be computed as:

$$x_1 = y_1 - \mathcal{F}(x_2), \qquad x_2 = y_2 - \mathcal{G}(y_1) \tag{2.37}$$

A very straightforward way to apply this idea to the Transformer has been introduced and used with great success in the Reformer architecture [42], along with other mechanism to reduce memory usage. The basic idea is to create a reversible block (as described above) that contains one attention sublayer as the $\mathcal{F}$ function and one feedforward sublayer as the $\mathcal{G}$ function. Now, the activations for only one out of the N Transformer layers

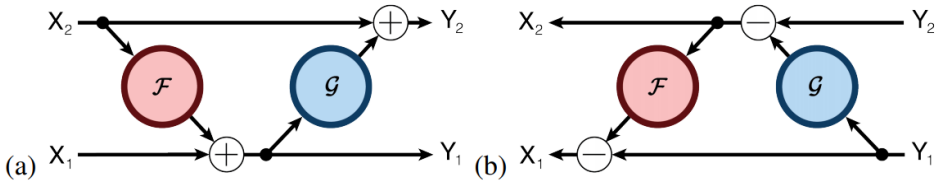


Figure 2.19: *The forward **(a)** and reverse **(b)** computations on a reversible block (Source: [26])*

have to be stored.

2.5 On Symbolic Music and Vocal Melody Generation

2.5.1 Symbolic Music - MIDI

Symbolic music refers to a notation-based representation of music, which contains information about a music piece (such as pitches, rhythm, instruments and more), but no audio. The two most common symbolic music (digital) formats are **(a)** digital score and **(b)** MIDI.

A digital score is similar to an analog score (or sheet). Scores with various musical notations have been developed over the years (traced back to 2000 BC), while the modern staff notation, which we are all familiar with, took its final form around the 19th century and got digitised in the 1990's. The score format imposes a certain structure. A music piece is divided into bars (or measures). A bar is a segment of time corresponding to a specific number of beats. Notes are represented by their pitch and their duration which corresponds to a note value, which is a fractional power of two. Rests are denoted similarly. Without analyzing sheet music further, one can understand that it has a certain structure and it is useful mostly for musicians.

MIDI on the other hand, is closer to an event log of all notes played by all instruments, than to a hierarchical structured representation of a score. MIDI stands for Musical Instrument Digital Interface and is a protocol that allows computers, musical instruments and other devices to communicate.

A single MIDI link can carry up to sixteen channels of information, different digital instruments for example. The basic element of MIDI is the event message. These are data that specify instructions, like a note onset with its pitch and velocity (loudness) or clock signals (which set tempo) and more. When a musician plays a MIDI instrument, all of the key presses, knob turns and other actions are converted into MIDI data. These data can also be stored to files and the original song that was played can be reconstructed from them. Early mobile phones used MIDI files as ringtones, because of their small size. MIDI files can also be annotated with other metadata events, such as lyrics. Today, MIDI files are used widely and some that contain synchronized lyrics are used for karaoke and related videogames.

Since a MIDI file is just binary data, a common way to visualize it is the piano roll representation, a two dimensional representation, with the horizontal axis denoting time

and the vertical being the note range. An example can be seen in Figure 2.20.

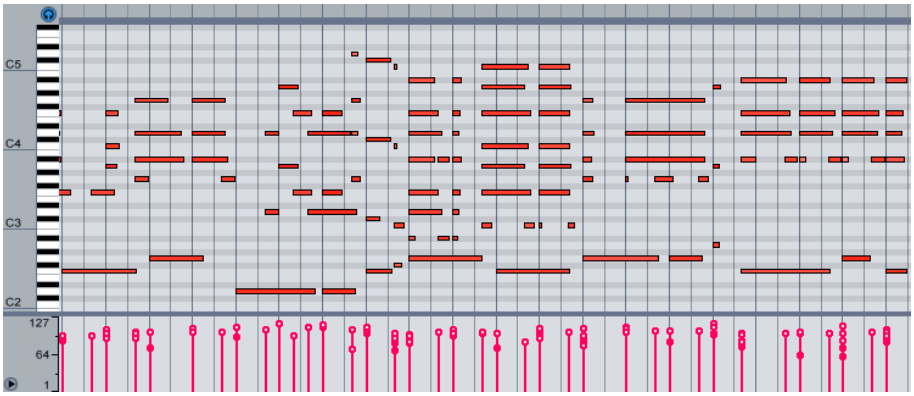


Figure 2.20: *The piano roll representation of a MIDI file (Source: songaweek)*

2.5.2 Music Theory

We will shortly present some concepts from music theory that we use in the next chapters.

Notes can represent the pitch and duration of a sound in musical notation. A note can be represented by a letter (A to G), an optional accidental symbol (♯ or ♭) and a subscript number (0 to 10) denoting the octave. A_4 is the standard tuning pitch and is equal to 440 Hz. An octave contains 12 tones, or steps. All 12 tones can be seen in Figure 2.21.

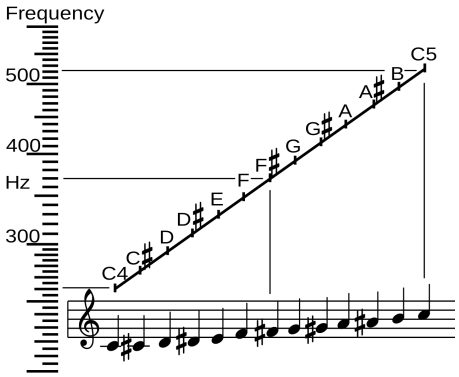


Figure 2.21: *The chromatic 12-tone chromatic scale built on C4, with corresponding frequencies. (Source: Wikipedia)*

Enharmonics or enharmonic equivalents are equivalent notes that have different "spellings". For example **E**♯ and **F** correspond to the same pitch, but they are spelled differently. While a specific spelling is commonly used, there are contexts in which another spelling is preferred.

Chords are (harmonic) sets of notes that are played simultaneously. Chords can be symbolized by simply enumerating the containing notes. For example, **C E G** (we will not use

the octave). Another notation that is used in popular music is using a letter indicating the root note, a symbol indicating the quality and optional additional symbols for more complex chords. Chord qualities are the qualities of the note intervals and they can be major, minor, augmented, (half-)diminished and dominant. In our example our chord can be written as **C** or $\mathbf{C}_{\text{maj}}$ (major).

Scales (or keys) are sets of notes ordered by pitch. A specific scale is defined by its characteristic interval pattern (like chord qualities) and by a note, known as its first degree (or tonic). For example, C major indicates a major scale with a C tonic. Most common modern Western scales consist of 7 notes (per octave). Each note is called a degree of the scale. Each scale contains specific accidentals, which are called the key signature. Most contemporary popular music is written in one scale, meaning that chords that consist of notes of that scale are used. Sometimes the scale can change mid-song. This key or tonal change is called modulation.

Roman Numerals is another representation of chords. They are commonly used in harmonic analysis to denote the scale degree on which the chord is built on. Roman numerals (I, II, III, IV, ...), can denote scale degrees themselves. More commonly, however, they represent the chord whose root note is that scale degree. For instance, III denotes either the third scale degree or, more commonly, the chord built on it. Typically, uppercase Roman numerals (such as I, IV, V) are used to represent major chords, while lowercase Roman numerals (such as ii, iii, vi) are used to represent minor chords. So for example, in the key (scale) of C major, the chord **E G B** or $\mathbf{E}_m$ is represented as **iii** (lowercase), since it is the third degree of the scale and minor, while the chord **F A C** or $\mathbf{F}_{\text{maj}}$ is represented as **IV** (uppercase), since it is the fourth degree of the scale and major. Roman numerals are sometimes complemented by numbers to denote inversion (different note ordering) of the chords. The musical analysis process that creates this representation for a chord is called Roman Numeral Analysis. A more thorough example can be found in Figure 2.22.

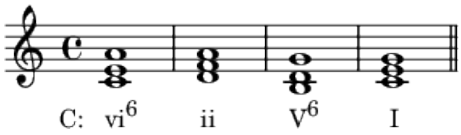


Figure 2.22: *The chord progression vi-ii-V-I in the key of C major, in standard notation* $A_m - D_m - G_{maj} - C_{maj}$. *(Source: Wikipedia)*

2.5.3 Symbolic Music Generation

Music generation is by no means a new area of interest among computer scientists and artists, with the first example being the Illiac Suite [32] in the late 1950's, utilising generative grammars and Markov chains. Similar methods based on knowledge and rule-based approaches were popular the following years, with the first use of Artificial Neural Networks [46] in the late 80's setting the paradigm until today. Music generation would mostly be thought of as a sequence modelling task, with recurrent architectures applied

to various symbolic representations.

Another line of work would create representations directly from spectograms and waveforms, following the rise of deep features in the wider field of audio and music processing with the most notable examples being SampleRNN [55] and Jukebox [22].

We will now mention some notable works in the field of symbolic music generation. MidiNet [91] uses convolutional GANs [27] to generates monophonic MIDI melodies of fixed length by dividing each bar to fixed timesteps in a piano roll representation. Another popular approach that creates a text event represenation of MIDI events (inspiration for our approach) is used in PerformanceRNN [82] , Music Transformer [36] and MuseNet [64] with great success on expressivity. A MIDI file is namely completely represented by a text sequence. MuseGAN [23] uses a GAN-based model for multi-track sequence generation on piano-roll format extracted from MIDI and MusicVAE [74] employs a hierarchical Variational Autoencoder model [41] to generate multi-track sequences, controllable through latent space manipulation.

2.5.4 Conditional Vocal Melody Generation

Conditional Vocal Melody Generation is the task of generating a monophonic melody to match a predefined lyrics text. In [4] the concatenation of syllable and word lyrics embeddings and a context window of the melody are encoded by two separate GRU encoders and together condition a melody multi-layer decoder that explicitly outputs a note, its duration and alignment information for each lyric syllable. [52] takes a similar approach. A skip-gram model [57] is used to get word and syllable embeddings that are combined and used as input for an LSTM encoder that uses a context vector to attend to three separate decoders, for note, duration and rest. By connecting the encoder's hidden vectors to a language model head it is possible to also generate lyrics.

Another work [94] uses an LSTM that takes as input lyrics embeddings and noise vectors to generate MIDI sequences in the above format, that are provided to another LSTM together with the original text embeddings and are classified as real or fake if they match the training distribution, functioning as a GAN. Lastly, [92] applies mutual information maximization to a standard encoder-decoder LSTM architecture with attention.

2.6 On Conditional Lyrics Generation

An area of focus regarding lyrics generation is replicating the lyrical structure and content of a specific genre or music style. For example in [60] a Transformer-based denoising autoencoder architecture is used to create rap lyrics from given content words, achieving rhyming by choosing words with large vowel overlap to end each line, while [73] generates parody lyrics that adhere to the original song syllable and rhyming constraints with suitable token masking. In [87] and [88] an LSTM Variational Autoencoder(VAE) is used to create latent representations of lyrics, which are used together with embeddings extracted from audio in order to condition lyrics generation on a specific music style.

Another area, more relevant to our work is the modelling and temporal conditioning

of lyrics to a specific vocal melody in the symbolic domain. An encoder-decoder LSTM architecture is used in [50], where only the rhythmic quality of the melody is taken into account. In [89] conditioning is done by concatenating each lyric syllable to the corresponding melody note and its local window, before feeding it as input to an LSTM language model. Finally, [14] utilizes Sequence Generative Adversarial Networks [93], an architecture that overcomes the discrete sequence generator differentiation problem with gradient policy update, trained on pairs of lyrics lines and their corresponding melody.

2.7 Singing Voice Synthesis

Singing voice synthesis is the task of generating audio for specific melody and lyrics. Note that while our work deals with symbolic representations and not audio synthesis, we employ a singing voice synthesis model to generate audio for our results, which we use in the qualitative evaluation study. We will shortly discuss some approaches to this task.

Singing voice synthesis is closely related to the more general task of speech synthesis (or Text-to-Speech) and so has gained more attention, with the very first attempt taking place in 1961 at Bell Labs[6]. A very successful (commercial) application of singing voice synthesis is the VOCALOID[7]. Its core mechanism is concatenative synthesis in the frequency domain, using short professionaly recorded samples. Another popular commercial system is Sinsy[8], which uses mostly Hidden Markov models.

Similar to other NLP tasks, voice synthesis used to be comprised of many components, such as text analysis, phoneme analysis, intonation and others, while lately there has been a shift towards more end-to-end approaches, that directly generate mel-spectrograms or even waveforms. DeepSinger [72] is a multi-lingual model based on the Transformer architecture that generates spectrograms which are converted to waveforms with the Griffin-Lim algorithm[9]. Another model, called WGANSing [11] uses a Wasserstein-GAN [1] that works on vocoder (and other) features.

We decide to use an architecture called Mellotron [85], since we find it to perform very well among the alternatives and a pretrained model is publicly available. Mellotron is a multi-speaker voice synthesis model, that is based on the Tacotron 2 architecture [81], a recurrent seq2seq feature prediction network that maps character embeddings to mel-scale spectrograms. Unlike other methods, Mellotron is trained only with read speech data without alignments between text and audio. Nevertheless, it manages to generate singing mel-spectrograms, by explicitly conditioning on rhythm and continuous pitch contours from an audio signal or music score. The mel-spectrograms are finally converted into audio, using WaveGlow [65], a flow-based network for audio synthesis.

We are able to use Mellotron as the final part of our pipeline, by reformatting our generated output into a suitable format.

[6] https://www.historyofinformation.com/detail.php?entryid=4445
[7] https://www.upf.edu/web/mtg/news/-/asset_publisher/WM181VyAQipW/content/id/231857712/
[8] http://www.sinsy.jp/
[9] https://paperswithcode.com/method/griffin-lim-algorithm

Chapter 3

Building our Dataset

In this Chapter we analyze the Lakh MIDI Dataset that constitutes the basis of our dataset. We present the decisions and the steps we took to make the irregular and arbitrarily lyrics annotations well structured and formulated for our task. We talk about the tools we used and the way we applied music theory and music analysis to reshape our data in a more compact and meaningful format. Lastly, we demonstrate our strategy to separate lyrics and melody and the data we collected to finetune a pretrained language model on lyrics.

3.1 The Lakh MIDI Dataset

The core of our dataset is the Lakh MIDI Dataset (LMD) [70]. LMD is a collection of 176,581 unique MIDI files, scraped from various publicly-available online sources and its main goal is to facilitate large-scale music information retrieval. It is the largest dataset of symbolic music that is publicly available. A subset of 45,129 files have been automatically matched to entries in the Million Song Dataset (MSD) [8] and small excerpts (around 30 seconds or more) of them have also been automatically aligned to available audio previews (linked to the MSD). The Million Song Dataset is a freely-available collection of audio features and metadata for a million contemporary popular music tracks. The matching and alignment of the files was done using a Dynamic Time Warping [6] based algorithm and for each match there a confidence score shows how probable it is that the result is correct.

To make sure we don't include any duplicate songs, which is unfortunately common and creates an imbalance in our data, we used the subset of MIDI files that were matched to the MSD and included only one MIDI file for each entry. We chose the ones with the highest confidence scores among all others that were valid for our task, meaning those that include lyrics and comply with other constraints that we discuss later.

3.2 Shaping the Dataset for our Task

3.2.1 Drawbacks of the Existing Dataset

The above dataset is not oriented towards analysis or processing of vocal melody and lyrics. For this reason many files don't include vocal melody at all, or if they include one it is sometimes considered as another instrument without synchronized lyrics included. Another issue is that in MIDI files the notion of annotating a specific channel/instrument as vocals does not exist and lyrics are included as metadata of the whole song. Also, for the files that do contain lyrics, there is no consistent annotation. The files are uploaded by different users. For example, some of the files that include lyrics are used for karaoke or relevant video games, where vocals are important, and others are transcribed or used by singers and musicians.

This high irregularity of the annotations mostly affects lyrics and it can lead to the following: **(a)** different sentence and verse separators or special characters, **(b)** some lyrics to be included as MIDI lyrics events, while others were included as text events (usually reserved for metadata such as copyrights or artist name) and most importantly **(c)** inconsistency in the way the lyrics were partitioned in order to correspond to notes. To give some examples of the latter, the phrase *hello world* could be split into *hell-o-world* or *hel-lo-wo-rld* or the whole phrase could correspond to just one note. This depends not only on the annotator, but on the way it is sung too. This not desired, since it increases our token vocabulary size and restricts us on training and on generation.

3.2.2 Creating a more Standardized Dataset

For the above reasons, we applied the following preprocessing steps to get a more structured dataset, making it suitable for our task:

- Language detection software [1] was used to keep only English lyrics.
- Lyrics events that were misplaced or mixed with text events had to be reordered, based on their timing information.
- Many tracks denoted line or part boundaries differently, so we used a strict multiline format (for example substituting full stops with new lines). The only characters we keep are:
  - letters
  - single spaces
  - single and double new lines
  - commas
  - apostrophes
  - hyphens

[1] https://github.com/aboSamoor/polyglot

- MIDI files don't include instrument to lyrics mapping. Sometimes, the synchronization is not absolute, so we assigned each lyric to the closest note, given a threshold, and chose the instrument with the most matches.
- Tracks with less than 50 lyrics syllables were not used.
- We restrict all notes in the piano octave range.
- All instruments are grouped to 8 instrument classes: *Piano, Guitar, Bass, Strings, Wind, Synth, Drums, Effects*

As we mentioned above, the division of lyrics to singing notes can vary, depending on the singer or the annotator. Sometimes more than one syllables or even words correspond to one music note, while one syllable is possible to belong to more than one notes. To make this division consistent and reversible at inference time we enforce a strict syllabified format:

- We do grapheme-to-phoneme conversion, using Phonetisaurus [62] and the CMU pronouncing dictionary [29]
- We split words to syllables, each containing one vowel
- If a note corresponds to **n >1** syllables we divide it to **n** equal duration notes of the same pitch
- If a syllable spans **n >1** notes we match it to the first note and assign the next **n - 1** notes to a special symbol

3.2.3 Text Event Format

After the above process we are left with **8505** MIDI tracks.

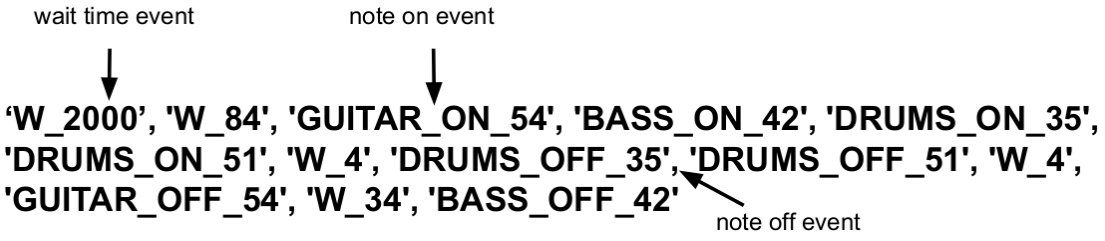


Figure 3.1: *Instrumental Text Events*

We further analyze them and create text event sequences for the instrumental and the vocal part. We use the extracted phonemes instead of word syllables to make the size of the token vocabulary smaller and also account for rhyming, since words or syllables that are spelled different can contain the same phonemes. We use MIDI ticks to denote time and MIDI pitches for notes (C4 being MIDI pitch 60). The format of these text events is

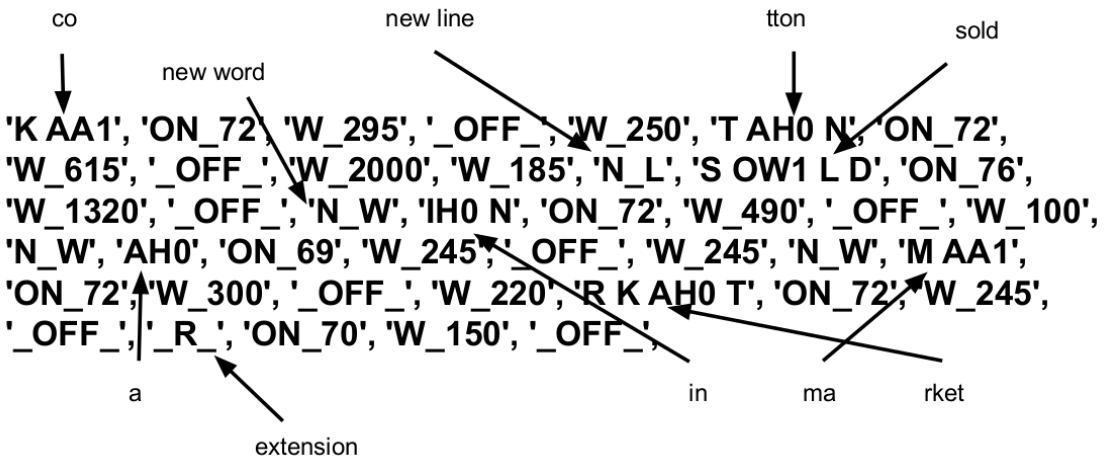


Figure 3.2: *Vocal Text Events (phonemes) corresponding to the lyrics:*
cotton
sold in a marke(-e)t

analyzed below.

For both sequences we have the following events:

- **wait time events** measure time passing in MIDI ticks, they can mean rest if no note is being played or duration of current note and makes it easy to model polyphonic music. Largest value is *2000*, so longer events are represented by adding waits.
- **note on events** signify that a note of this MIDI pitch (C4 = 60) starts.
- **note off events** signify that a note of this MIDI pitch ends.

For instrumental sequences (Figure 3.1):

- The instrument class name is appended to the corresponding note on event. We have 8 instrument classes in total.

For vocal sequences (Figure 3.2):

- **syllable/phoneme events** signify the lyric syllable that belongs to the following note
- **extension events** are used when a syllable is sung in two or more notes
- **boundary events** are used for providing structure information and making it possible to recreate the original text and include:
  - **new double line**
  - **new line**
  - **new word**
  - **comma**

- Since vocals are monophonic and one note is played at each time we need only one **note off event**.

Notice that synchronization of instruments and vocals is not explicit, but it happens implicitly, since timing information (wait events) is the same for both sequences.

Each MIDI tick corresponds to $\frac{60}{T * R}$ seconds, with T representing tempo in *Beats Per Minute* and R being the resolution of the file in *Pulses Per Quarter Note* (PPQN) with a minimum value of 24 and maximum of 960 for most. The resolution can vary a lot between MIDI files, which largely impacts the distribution of wait tick times. To alleviate this issue we normalize the resolution by using 960 PPQN for all files.

At first, we also used the MIDI velocity of each note, but as it increased the vocabulary size, which is disadvantageous in low-resource conditions [80], and couldn't be used in the final singing voice synthesis, we decided to discard it.

Sequence Length	Instrumental	Vocal	Vocal (w/o syllables)
max	59120	6115	5065
median	13041	1645	1373
min	575	310	256

Table 3.1: *Sequence Lengths for Instrumental and Vocal text event formats*

In Table 3.1 we can see the maximum, median and minimum of sequence lengths for instrumental and vocal text event sequences. For the latter we can also see the sequence lengths when we do not include syllables/phonemes. It is evident that the size of the instrumental especially is very large. Even though we use an architecture that was developed for usage in long sequence scenarios, compressing the instrumental sequence without losing important information is for our profit. Training will be faster and a more dense representation can make our model more robust.

3.3 Applying Music Theory Analysis

3.3.1 Chord Reduction

We adopt many techniques to combat the large sequence lengths of our data, which we will analyze later. But even though we are able to fit our models in a single GPU, the training process still requires a lot of time. Also, the information contained in the instrumental text events is very dense and it can be argued that a singer does not have to know every single note played by every instrument to sing on a track. For the reasons above, we choose to create a new representation from raw midi data, that at the same time is compact, meaningful and does not discard important information.

To achieve this we use the *music21* library [19], a set of tools for computer-aided musicology. Music21 can parse symbolic music in many formats, such as MIDI or MusicXML[2]

[2] https://www.musicxml.com/publications/

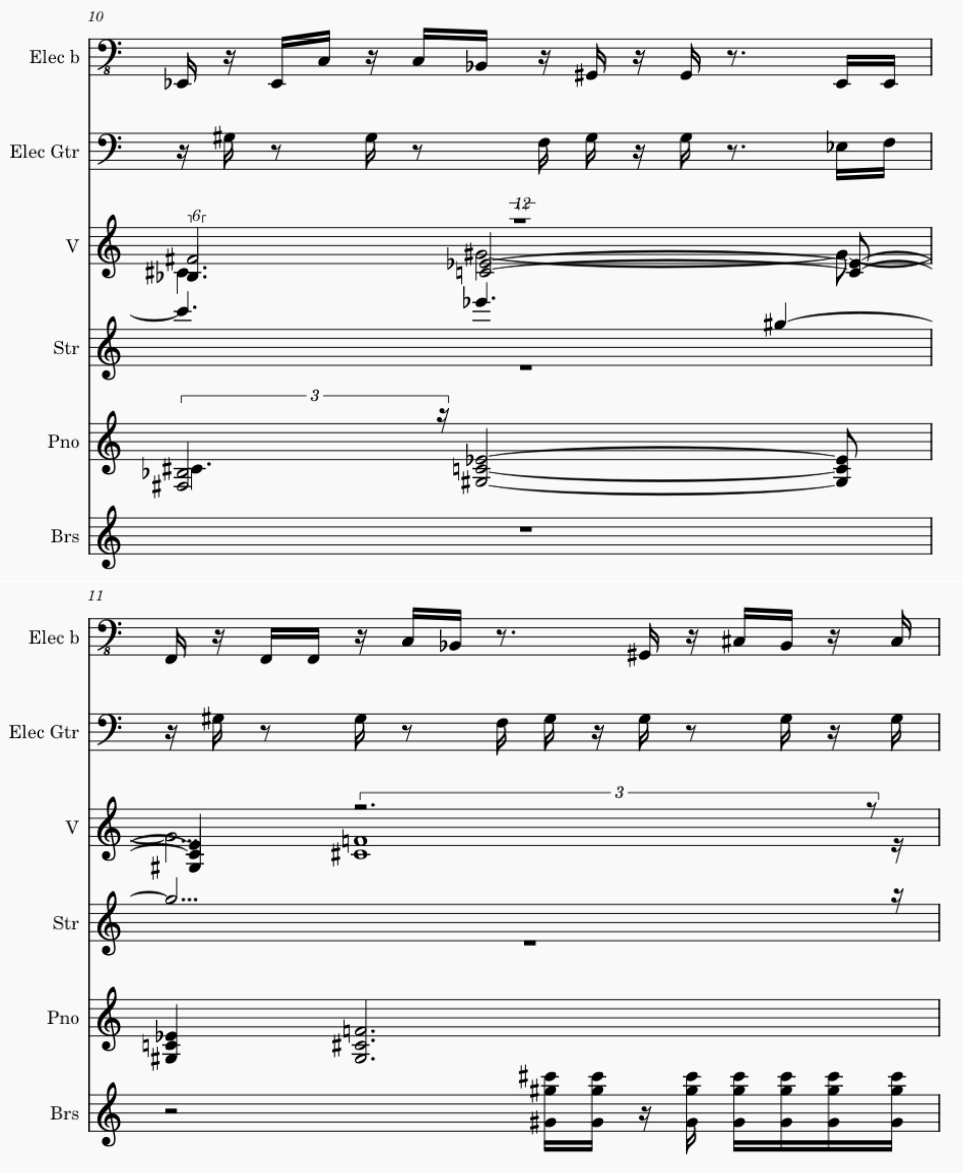


Figure 3.3: *An example of two music measures before compressing them to chords and annotating them using roman numeral analysis (illustration with MuseScore)*

(digital score) among others. It contains very useful notions of music theory, such as harmony, chords, key signatures and modalities.

First of all we analyze and remove vocal and percussion instruments from our files using our preprocessing algorithm and we parse them with music21 to get a score representation. Then, we use a method that is implemented by the library, called *chordify*[3]. This method merges all different parts/instruments into one and can create a chordal reduction of polyphonic music, where each change to a new pitch results in a new chord. This way we can reduce a complex score to a simple chord succession, discarding no information and simplifying it simultaneously. To visualize this simplification we present the score representation of two music measures before in Figure 3.3 and after this compression in Figure 3.4.

[3] http://web.mit.edu/music21/doc/moduleReference/moduleStream.html#music21.stream.Stream.chordify

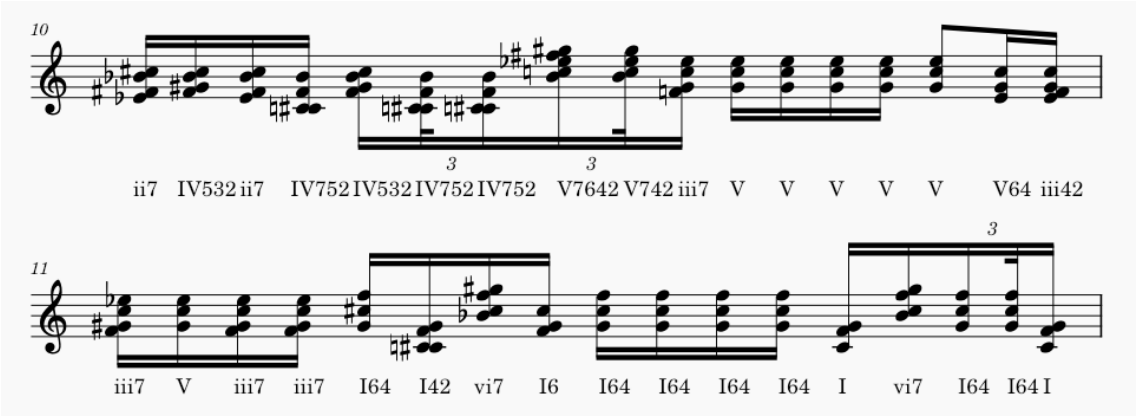


Figure 3.4: *The same two music measures of Figure 3.3, but with generated chords and their roman numeral representation. Notes are restricted in an octave range (illustration with MuseScore).*

The reason we do not include the percussion part is because it would insert noise during this process. Since different parts of a drum kit are represented simply as different notes, music21 has the drawback of interpreting them as notes and it is not able to extract information correctly. Also, it can be argued that the exact parts played by a drummer are not crucial to the singer. Instead, we will use more basic rhythmical notions such as beats and downbeats (the first beat of the measure), which will give us more abstract, but concrete rhythmical information.

3.3.2 Roman Numeral Analysis

The created chords reduce the instrumental sequence lengths by a large factor, but if we now create a separate token for each chord as a combination of pitches, the size of the vocabulary will become huge, because of all the possible note combinations and permutations (voicings). Also, the exact same song, transposed to a different music key would consist of completely different chords. It would be more useful to model the relative positions and function of the chords, instead. To solve these issues, we use *Roman Numeral Analysis* (RNA). RNA is a very common type of musical analysis and its core idea is that chords can be represented by a degree of the musical scale they belong to. It is presented in more detail in 2.5.2 along with other basic music theory concepts.

At first, we get a key estimation of the current song using the Krumhansl-Schmuckler algorithm[4]. This algorithm determines the key a song is in by comparing its pitch class distribution to ideal pitch distributions for each key. It is implemented in music21. It should be noted that sometimes the key changes during a song (modulation), even though this is not that common for pop or rock music, that makes up most of our dataset. Identifying modulations algorithmically is not easy and complicates our representation, so we identify the most common key and we implicitly represent modulations by analyzing the changed tonality chords in the original key.

Following, we get the roman numeral representation of every chord based on the

[4] http://rnhart.net/articles/key-finding/

estimated key, including inversions (specified accordingly). The enharmonics are also respelled to get simpler representations, in the context of the key. One very important advantage of this method, except for providing a smaller vocabulary size, is that every chord is independent of the key the song is written in. This is actually a crucial data augmentation procedure, because all our dataset is being transposed to a common but abstract key. Previous work that takes this into account is limited. To our knowledge only [4] transposes the dataset to the commonly used key of *C* or *Am* and [90] trasposes to all keys.

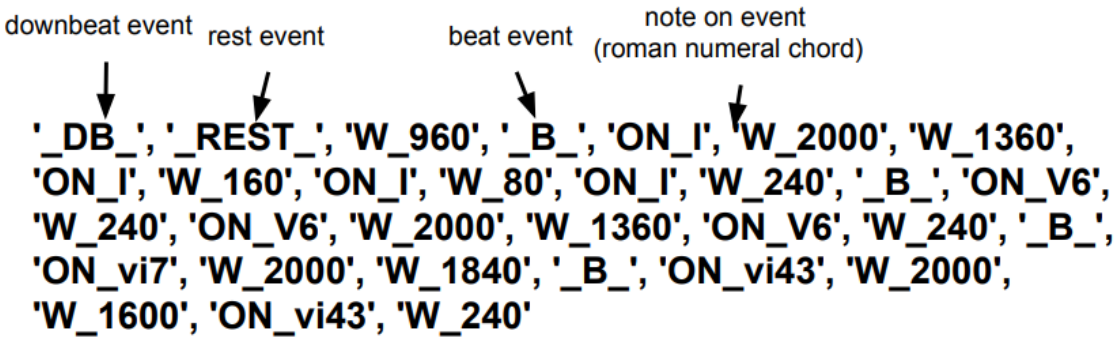


Figure 3.5: *Instrumental Text Events with Roman Numeral Chords, Rests, Downbeats and Beats*

We also apply a procedure similar to RNA for the vocal melody notes, by converting each absolute MIDI pitch to a corresponding scale degree. Since octave information is also useful to get a more expressive vocal performance, we indicate which octave a note is in by appending a number that specifies how many octaves above or below from the tonic of the key the note lies.

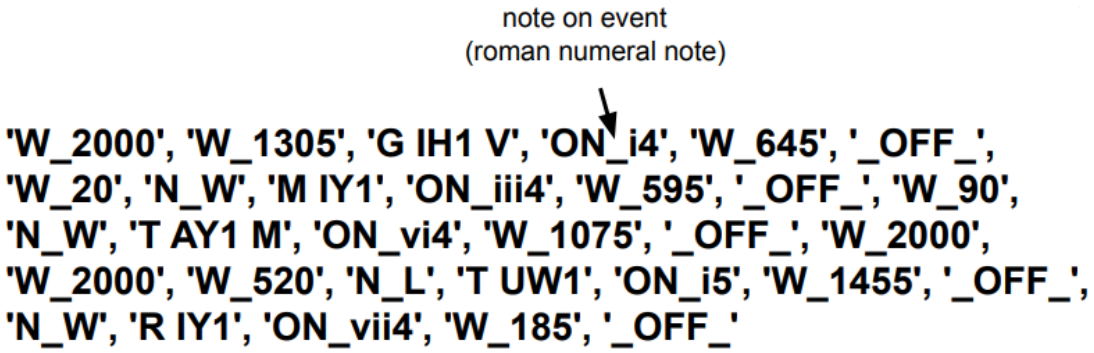


Figure 3.6: *Vocal Text Events with Roman Numeral Notes*

Since the instrumental data are now chords with no overlap it is very uncommon for a note on event not to succeed a note off event directly. For this reason we introduce a new event that explicitly represents the few rests that remain. Also, as we discussed above we discard percussion and substitute it with two new events that correspond to downbeat, meaning the first beat of a measure, and beat, a more fine grained division (mostly quarter notes). In Figures 3.5 and 3.6 the reader can see examples of the new format.

Sequence Length	Instrumental	Reduction
max	11730	80.16%
median	3220	75.31%
min	239	59.83%

Table 3.2: *Sequence Lengths and Percent Reduction of Instrumental Events*

Table 3.2 proves that we achieve a very important reduction in the length of our instrumental sequences. The size of vocal sequences remains the same.

3.4 Decoupling Lyrics and Melody

As we will discuss more thoroughly in the next chapter (Section 4.2), being able to model lyrics and vocal melody separately, while also accounting for their correlation, can have many advantages. As the reader can see in the previous sections, so far we have merged the two sequences in one. Each note is preceded by the phonemes/syllable that it vocalizes, which has the advantage of an explicit and simpler one-to-one correspondence.

To shape our dataset in a suitable format for this separate modelling we first follow the preprocessing steps we analyzed in Section 3.2. The difference is that we do not substitute syllables with phonemes. Instead, we concatenate all syllables and create a single lyrics text for the whole song. Then, we remove the **syllable/phoneme events** from the vocal event text sequences, except from the **extension events** that are needed to specify that a syllable extends to more than one note. The new concatenated then can be used with **any** tokenization method, which does not restrict us and allows us to use any pretrained language model. To further emphasize the importance of the syllabification process we described, if we had not enforced specific tokenization rules, that are also reversible, we would not be able to know how to map the new lyrics text to vocal melody notes.

We then assemble a lyrics dataset to finetune a pretrained english language model. The purpose of this is twofold. The language model will use the attention mechanism of the Performer, which as we mentioned in Subsection 2.4.2 requires some training steps to be able to utilize loaded weights from a pretrained model. Also, lyrics have specific structure and content (repetition of words or phrases or onomatopoeia for example). So, the finetuning also serves the function of Domain Adaptation.

We use three publicly available datasets[5] [6] [7] that include lyrics. We keep only english lyrics and enforce a multiline format as in our target dataset. We also remove metadata regarding structure such as the words *Verse* or *Chorus*. The dataset we create contains lyrics for 263,666 songs.

[5] https://www.kaggle.com/neisse/scrapped-lyrics-from-6-genres
[6] https://www.kaggle.com/edenbd/150k-lyrics-labeled-with-spotify-valence
[7] https://www.kaggle.com/deepshah16/song-lyrics-dataset

Chapter 4

Model Architectures for Lyrics and Vocal Melody Generation

In this Chapter we present the model architectures that we use in our experiments. First, we formulate the sequence-to-sequence task we want to solve. The architecture we use is based on Transformers, substituting the original softmax attention with the FAVOR+ attention mechanism and including some enhancements we found to be helpful to reduce memory usage. We use improvements to the original architecture, documented in literature. We also explore a decoding technique specific to our task to make the generation process more robust. Next, we present a novel architecture for modelling multiple sequences, while we also formulate the input combination technique we are using. Finally, we show the architecture of the pretrained language model that we finetune on the lyrics dataset.

4.1 Sequence to Sequence Modelling

4.1.1 Formulation

Let us define a few concepts first to show how we train a sequence-to-sequence model on our dataset. We will not analyze the attention mechanism of the Transformer or the Performer architecture. The reader can refer to Subsections 2.3.5 and 2.4.2 for more details.

We have an instrumental sequence $\mathbf{I}_{1:n}$ of variable length n as input and a vocal sequence $\mathbf{O}_{1:m}$ of variable length m as output. Each token of the input sequence is a one-hot vector of dimension d_n and each token of the output sequence is an one-hot vector of dimension d_m, where d_n and d_m are the sizes of the input and output vocabularies respectively. Next, we have two mapping functions for each sequence $g_{\text{enc}} : \mathbb{R}^{d_n} \to \mathbb{R}^d$ and $h_{\text{enc}} : \mathbb{R} \to \mathbb{R}^d$ for the input and $g_{\text{dec}} : \mathbb{R}^{d_m} \to \mathbb{R}^d$ and $h_{\text{dec}} : \mathbb{R} \to \mathbb{R}^d$ for the output. These functions map sequences $\mathbf{I}_{1:n}$ and $\mathbf{O}_{1:m}$ to the sequences $\mathbf{X}_{1:n}$ and $\mathbf{Y}_{1:m}$ of the shared dimension d respectively, as:

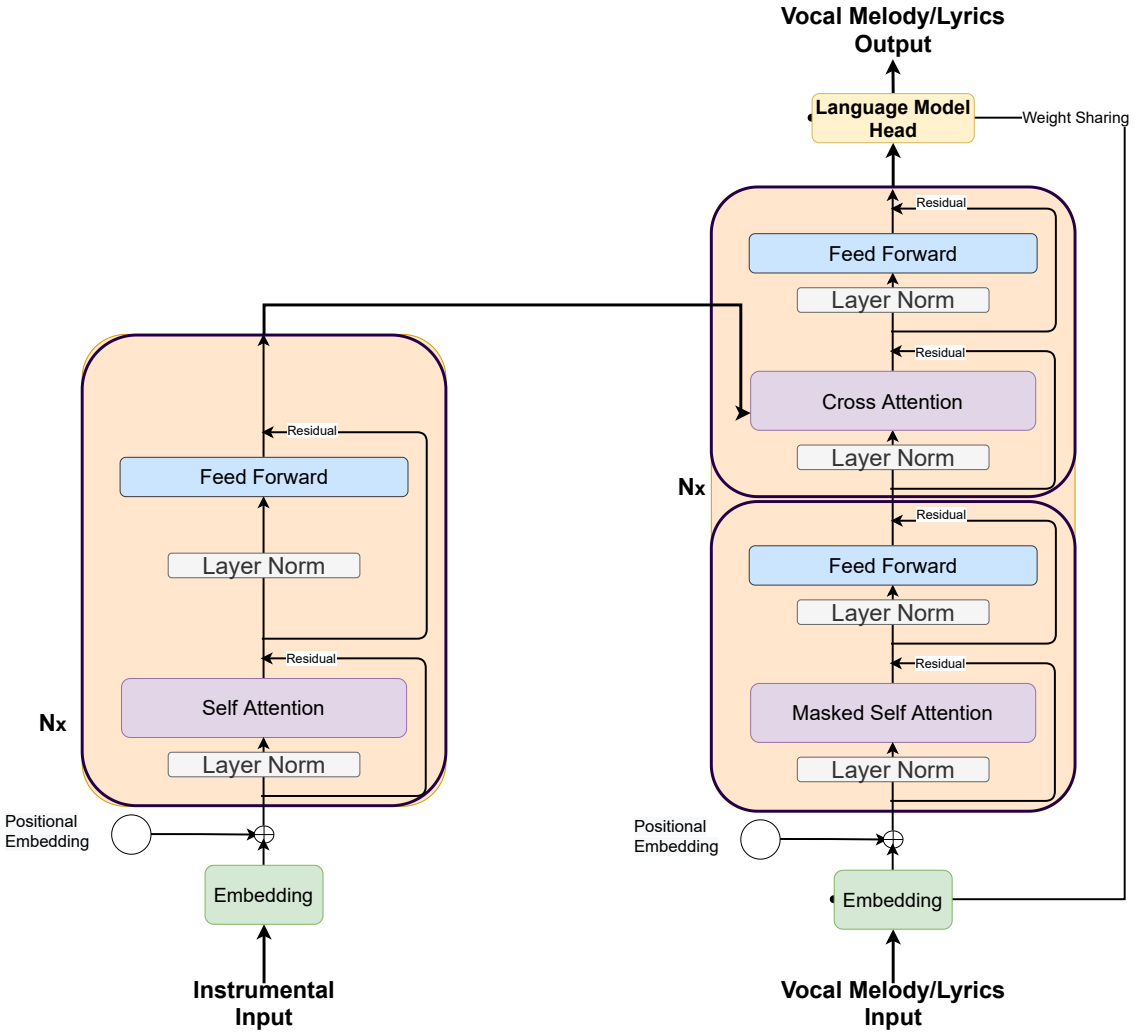


Figure 4.1: *Our Simple Sequence-to-Sequence Architecture with an instrumental encoder (left) and a vocal melody/lyrics decoder (right)*

$$\mathbf{X}_{1:n} = g_{\text{enc}}(\mathbf{I}_{1:n}) + h_{\text{enc}}(p_i) \qquad \mathbf{Y}_{1:m} = g_{\text{dec}}(\mathbf{O}_{1:n}) + h_{\text{dec}}(p_i) \tag{4.1}$$

where p_i is the index of each token in a sequence. g_{enc} and h_{enc} are the token embeddings and positional embeddings of the encoder (equivalent for the decoder) and are learnable.

The model we present here is trained to implement the following mapping:

$$f : \mathbf{X}_{1:n} \rightarrow \mathbf{Y}_{1:m} \tag{4.2}$$

It is composed of an encoder (left on Figure 4.1) with parameters ∂_{enc} and a decoder (right on Figure 4.1) with parameters ∂_{dec} and it models the conditional probability distribution:

$$p_{\partial_{\text{enc}},\partial_{\text{dec}}}(\mathbf{Y}_{1:m}|\mathbf{X}_{1:n}) \tag{4.3}$$

The encoder part encodes the input sequence $\mathbf{X}_{1:n}$ to a new sequence $\overline{\mathbf{X}}_{1:n}$ of the same dimension, thus defining the mapping:

$$f_{\partial_{\text{enc}}} : \mathbf{X}_{1:n} \rightarrow \overline{\mathbf{X}}_{1:n} \tag{4.4}$$

which now provides for each token contextual information of the sequence.

We can rewrite Equation 4.3, to depend only on the decoder part and the above contextualized sequence, as:

$$p_{\partial_{dec}}(\mathbf{Y}_{1:m}|\overline{\mathbf{X}}_{1:n}) \tag{4.5}$$

Or using Bayes rule:

$$p_{\partial_{dec}}(\mathbf{Y}_{1:m}|\overline{\mathbf{X}}_{1:n}) = \prod_{i=1}^{m} p_{\partial_{\text{dec}}}(\mathbf{y}_i|\mathbf{Y}_{0:i-1}, \overline{\mathbf{X}}_{1:n}) \tag{4.6}$$

Note that the probability for $\mathbf{y}_i$ does not depend on $\mathbf{Y}_{i:m}$.

The sequence $\mathbf{Y}_{1:m}$ includes vectors of dimension d. To bring them to dimension d_m, which is required both for training as well as decoding (more on that later) we use the transpose of the weight matrix of g_{dec}, as in the original Transformer implementation. This has been also explored in [67]. So we get the sequence $\mathbf{L}_{1:m}$:

$$\mathbf{L}_{1:m} = g'_{\text{dec}}(\mathbf{Y}_{1:m}) \tag{4.7}$$

Finally, we train our network by minimizing the cross-entropy loss between the sequence $\mathbf{L}_{1:m}$ and the original vocal sequence $\mathbf{O}_{0:m-1}$ shifted by 1 (otherwise the model would learn to copy the input):

$$L = \sum_{i=1}^{m} (\mathbf{o}_{i-1}) \log (\mathbf{l}_i) \tag{4.8}$$

As we can see, training is done with the teacher forcing technique.

4.1.2 Enhancements

As we already mentioned, a very important step to reduce the memory usage of the original Transformer architecture, which otherwise scales quadratically with the sequence length, is substituting the softmax attention with the FAVOR+ attention mechanism introduced by the Performer (Subsection 2.4.2).

Another important addition to make modelling such long sequences in a single GPU feasible, is using reversible layers (see Subsection 2.4.3) instead of normal ones. Each layer of the encoder will now correspond to a reversible block. Our decoder contains an

additional cross-attention sublayer, so to make it possible for it to use reversible layers, we include an extra feed-forward sublayer between self and cross-attention, creating two reversible blocks for each layer. Notice, that while this can be considered a significant change from the original architecture, recent research on reordering [66] or adding feed-forward sublayers [51], as well as on substituting attention with feed-forward blocks [53], has shown that feedforward layers mostly act advantageously.

Another technique for reducing memory footprint also introduced in Reformer [42] is used. It is called *feed-forward chunking*. Its idea is simple, it means that the input to a feed-forward layer is divided into groups and not processed entirely in parallel to avoid large memory consumption, producing the same result.

One more addition is the usage of learnable positional embeddings, instead of fixed sinusoidal ones. Learnable positional embeddings were introduced in [24] and have been used for state-of-the-art models, such as BERT [21] or GPT-2 [69].

In the original Transformer architecture, layer normalization (see Subsection 2.2.3) occurs after each sublayer and residual connection addition, in what is called *post-norm*. Another way to do layer normalization is putting it immediately before each sublayer, creating *pre-norm* residual units. The improvement from using *pre-norm* has been documented in [12] and further analyzed in [59]. We use *pre-norm*, since it improves performance, robustness and most importantly is stable when using large learning rates, with no need to do warm-up (starting with a very small learning rate and gradually increasing it), in contrast to *post-norm*.

Finally, we use Gaussian Error Linear Unit (GELU) instead of the more simple Rectified Linear Units (ReLU) as the activation function of the feed-forward networks. GELU has been used in BERT and GPT-2 with great success. Activation functions are documented in more detail in Subsection 2.2.2.

4.1.3 Decoding Strategy

The reader can affirm by examining our data, that in contrast to other token sequences (for example human language) there exists a very specific structure in the vocal sequences (as well as on the instrumental but we will not focus on this). To make the above more specific:

- The possible events that follow a **wait time** event are: **note off** events if a note is on (meaning a note on event was encountered more recently than a note off event) or one of: **phoneme**, **extension** or **boundary** events if no note is on.
- If the **wait time event** has the maximum value it can also be followed by a **wait time** event, since this is the way we model longer times.
- If we encounter a **note on** event only **wait time** events can follow.
- If we encounter a **note off** event **wait time**, **phoneme**, **extension** or **boundary** events can follow.
- **Boundary** events are always succeeded by **phoneme** or **extension** events.

- **Phoneme** and **extension** events are always succeeded by **note on** events.

We take advantage of these structural constrains, by masking the logits vector accordingly, allowing only valid events to be selected. This masking is done by setting all invalid events probabilities to $-inf$ before our decoding technique, which is *top-k sampling* followed by a softmax and sampling from the multinomial probability distribution.

Using these rules at generation time is very helpful. First of all, we explicitly enforce that the generated sequence will be in the correct format. Moreover, when we mask the invalid events before our decoding technique, we can argue that generation is more robust, since the probability distribution is only over valid events.

4.2 Decoupled Modelling - Combining Multiple Input Sequences

4.2.1 Formulation

The fact that lyric syllables and melody are jointly modelled as part of the same sequence is quite limiting, since it does not provide a lot of flexibility and control on the training or the generation process. Now we will describe the new task we solve, which can be thought of as a multi-source sequence-to-sequence task, with the difference of having two outputs instead of one.

As can be seen in Figure 4.2, we keep the instrumental encoder unchanged and instead of a single decoder, we have one decoder for lyrics and one for vocal melody, in which we add a second cross-attention layer. This technique to combine multiple input sequences in a decoder has been studied in [47] and has been shown to perform very well among the alternatives. The output of the instrumental encoder cross-attends to both the lyrics and the first cross-attention sublayer of the vocal melody decoder, while last layer output of the lyrics decoder cross-attends to the second.

We have still one instrumental sequence, let us denote it as $\mathbf{I}_{1:n}$ and one vocal sequence, that does not contain lyrical information (phonemes) now, let us denote it as $\mathbf{O}_{1:m}$. In our new architecture we have an extra part, which is the lyrics text and we will refer to the part of the network that is responsible for it as the *language model* (leftmost on Figure 4.2). Lyrics are tokenized by the tokenization method of the language method to the sequence $\mathbf{K}_{1:q}$ of length q and dimension d_q.

As before we project it to the shared dimension d, using learnable token embeddings $g_{\text{lm}} : \mathbb{R}^{d_q} \to \mathbb{R}^d$ and positional embeddings $h_{\text{lm}} : \mathbb{R} \to \mathbb{R}^d$. We get the sequence $\mathbf{Z}_{1:q}$

$$\mathbf{Z}_{1:q} = g_{\text{lm}}(\mathbf{K}_{1:q}) + h_{\text{lm}}(\text{p}(\mathbf{K}_{1:q})) \tag{4.9}$$

We condition the language model to the instrumental input. We denote the parameters of the language model as ∂_{lm} and the modelled conditional probability distribution as:

$$p_{\partial_{\text{enc}},\partial_{\text{lm}}}(\mathbf{Z}_{1:q}|\mathbf{X}_{1:n}) \tag{4.10}$$

We substitute now with the encodings of the instrumental encoder part $\overline{\mathbf{X}}_{1:n}$ and get a probability depending only on ∂_{lm}:

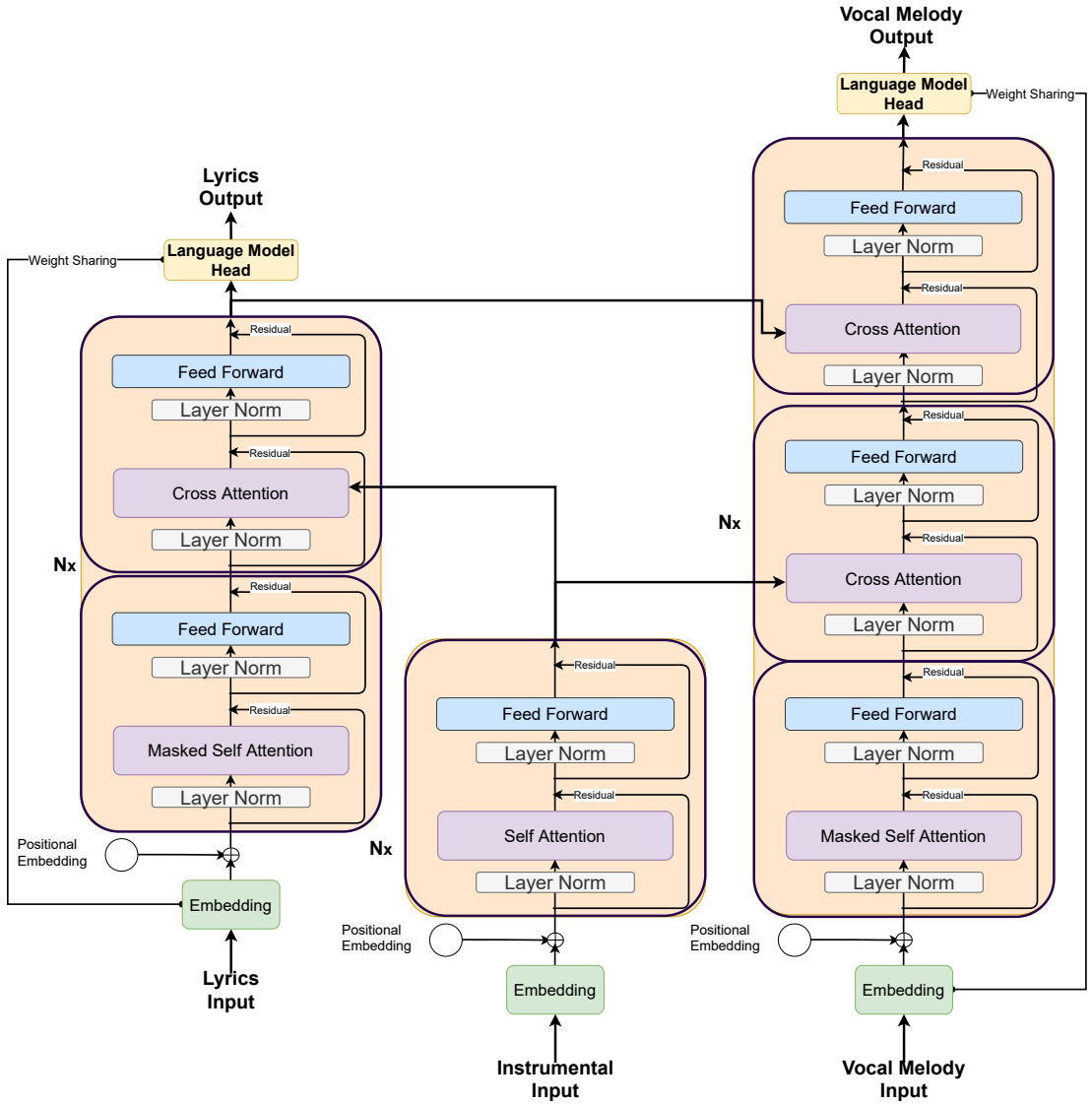


Figure 4.2: *Our Decoupled Architecture with an instrumental encoder (center), one decoder for lyrics (left) and one decoder for vocal melody (right) with two cross-attention sublayers. The encoder conditions both decoders, while the encodings of the lyrics model condition the vocal melody decoder.*

$$p_{\partial_{lm}}(\mathbf{Z}_{1:q}|\overline{\mathbf{X}}_{1:n}) \tag{4.11}$$

Using Bayes rule:

$$p_{\partial_{lm}}(\mathbf{Z}_{1:q}|\overline{\mathbf{X}}_{1:n}) = \prod_{i=1}^{q} p_{\partial_{\text{lm}}}(\mathbf{z}_i|\mathbf{Z}_{0:i-1}, \overline{\mathbf{X}}_{1:n}) \tag{4.12}$$

The sequence $\mathbf{Z}_{1:q}$ is now encoded into the sequence $\overline{\mathbf{Z}}_{1:q}$ which depends on both the lyrics sequence as well as on the instrumental one.

We can now use the transpose of the weight matrix of g_{lm} and get the logits vector $\mathbf{L}'_{\mathbf{1:q}}$ of dimensions d_q:

$$\mathbf{L'_{1:q}} = g'_{\text{lm}}(\mathbf{Z}_{1:q}) \tag{4.13}$$

We train this part of the network by minimizing the cross-entropy loss between the sequence $\mathbf{L'_{1:q}}$ and the original lyrics sequence $\mathbf{K}_{0:q-1}$ shifted by 1:

$$L_1 = \sum_{i=1}^{q} (\mathbf{k}_{i-1}) \log (\mathbf{l'}_i) \tag{4.14}$$

Next, we focus on the vocal melody decoder.

As the reader can see in Figure 4.2 and as we mentioned already, the vocal melody depends both on the instrumental and the lyrics. The conditional probability for vocal melody is:

$$p_{\partial_{\text{enc}},\partial_{\text{lm}},\partial_{\text{dec}}}(\mathbf{Y}_{1:m}|\mathbf{X}_{1:n}\mathbf{Z}_{1:q}) \tag{4.15}$$

We use the encodings from the instrumental encoder part as well as the lyrics encodings and we rewrite the previous equation as:

$$p_{\partial_{dec}}(\mathbf{Y}_{1:m}|\overline{\mathbf{X}}_{1:n}, \overline{\mathbf{Z}}_{1:q}) \tag{4.16}$$

$$p_{\partial_{dec}}(\mathbf{Y}_{1:m}|\overline{\mathbf{X}}_{1:n}, \overline{\mathbf{Z}}_{1:q}) = \prod_{i=1}^{m} p_{\partial_{\text{dec}}}(\mathbf{y}_i|\mathbf{Y}_{0:i-1}, \overline{\mathbf{X}}_{1:n}, \overline{\mathbf{Z}}_{1:q}) \tag{4.17}$$

We then use Equation 4.7 to get the logits vector $\mathbf{L}_{1:m}$ and we calculate the cross-entropy loss for this sequence as we did before (Equation 4.8).

Finally, we train the network to minimize the sum of the above two cross-entropy losses.

At inference time, the two separate sequences can be merged. Finally, one more advantage of this method is that we can either generate both lyrics and melody or specify in advance full lyrics or a part of them, to conditionally generate melody.

4.2.2 Lyrics Language Model

One of our main motivations behind the above decoupling was the ability to use a pretrained language model as prior. With the release of language models trained on large text corpora, it is common practice to use the knowledge they have acquired, either by loading a pretrained model's weights and finetuning on a downstream task (*warm-starting*) or with more sophisticated methods, such as fusion [83] or knowledge distillation [5].

Since all pretrained language models we researched use a different tokenization method than the one we are using (syllable tokenization), we cannot use any of the methods that

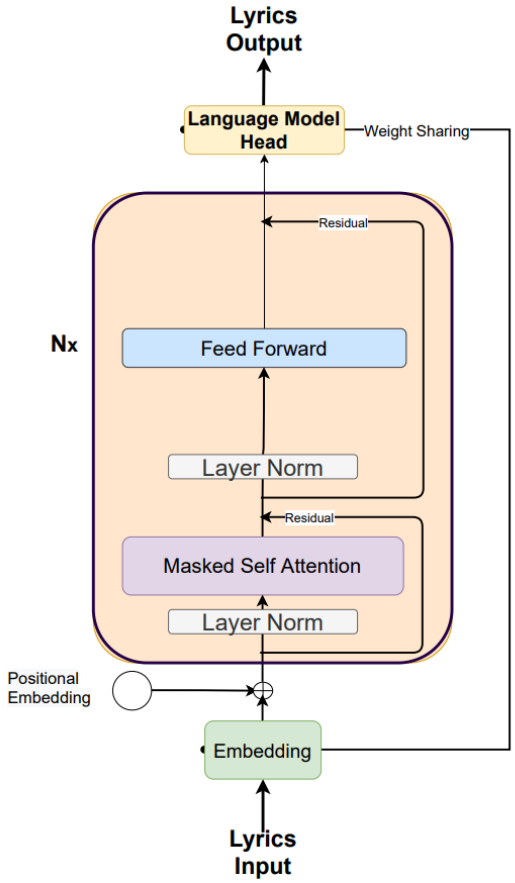


Figure 4.3: *The Language Model that we finetune on lyrics, warm-starting with pretrained distilGPT-2 weights*

require a one-to-one correspondence between tokens. So, we use a different network (lyrics decoder) that conditions our sequence (vocal melody) with cross-attention.

We decide to use the distilled version of GPT-2 (distilGPT-2), since it is one of the best performing causal language models available and since it is small it fits our low-resource requirements. Knowledge distillation was introduced in [10] and [33] and used with great success in distilBERT [78], being one of the most successful ways to compress a model. Its basic idea is that a smaller student model is trained to reproduce the outputs of a larger, teacher model. Note that distilGPT-2 is 34% smaller than the smallest version of GPT-2, and two times faster on average. GPT-2 uses a Byte-Pair Encoding tokenizer [79].

We use the model's weights to warm-start a Performer decoder with reversible layers, which we will later use in the decoupled model shown in Figure 4.2. As discussed in the Performer paper and as we found out in practice, a small amount of finetuning is necessary for the model to get on par with the original model. Also, we will use reversible layers as we do in the decoupled model.

The language model in Figure 4.3 is unconditional and causal (each token attends only to previous tokens) and is trained to minimize the cross-entropy loss like before. For the final model we add a randomly initialized cross-attention layer, as it has been showcased in [76]. As we confirm experimentally next, this random initialized layer does not degrade the quality of generated lyrics, even from the very beginning of the training.

Chapter 5

Experiments and Results

In this Chapter we discuss some experimental details and the model and training hyperparameters we used, as well as the impact of the model parameters on training speed. We further analyze the evaluation metrics we use. Following, we do a short comparison based on the above and analyze the qualitative evaluation study we conducted. The reader can visit Appendix A for plots of losses, and evaluation metrics. Appendix B contains examples of generated sequences.

Our code[1] is on GitHub. Most of the architecture code is based on a Pytorch implementation[2] of Performer, that uses the attention mechanism introduced by the latter on the original Transformer architecture, alongside some enhancements. We also use DeepSpeed [71], a deep learning optimization library, in our training setup.

5.1 Experiments

5.1.1 Model Hyperparameters

Table 5.1 summarizes the main hyperparameters for the three models we train, while Table 5.2 refers to the hyperparameters of the language model we finetune on the lyrics dataset we assemble. We use a distilled version of GPT-2 (*distilGPT-2*) to perform warm-start, by loading its weights to a model that uses the FAVOR+ attention mechanism and has reversible layers. DistilGPT-2 is pretrained on a large corpus of regular English text and its weights are publicly available online[3].

As we already mentioned in Chapter 4 all our models have the following settings:

- FAVOR+ attention mechanism
- reversible layers
- feed-forward chunking (10 chunks)
- learnable token embeddings

[1] https://github.com/gulnazaki/thesis
[2] https://github.com/lucidrains/performer-pytorch
[3] https://huggingface.co/distilgpt2

Hyperparameters	Seq2Seq (Full)	Seq2Seq (Chords)	Decoupled (Chords)
dimension	512		768
depth	6		
encoder heads	8		6
decoder heads	8		6
lm* heads	-		12
vocabulary size (encoder)	3411	22443	
vocabulary size (decoder)	11984	12066	2122
vocabulary size (lm*)	-		50257
maximum length (encoder)	50000	11730	
maximum length (decoder)	6115		5065
maximum length (lm*)	-		1024
token embedding dropout	0.1		
feed-forward dropout	0.1		
attention dropout	0.1		

Table 5.1: *Model Hyperparameters for Sequence-to-Sequence with Full Instrumental Input, Sequence-to-Sequence with Reduced Chords Instrumental Input and Decoupled with Reduced Chords Instrumental Input. *language model*

Hyperparameters	Lyrics Language Model
dimension	768
depth	6
heads	12
vocabulary size	50257
maximum sequence length	1024
token embedding dropout	0.1
feed-forward dropout	0.1
attention dropout	0.1

Table 5.2: *Model Hyperparameters for the Language Model that we finetune on Lyrics, we warm-start (load weights) from a pretrained distilGPT-2 model*

- learnable positional embeddings
- pre-norm (layer normalization before each sublayer)
- GELU activation function in the feed-forward sublayers

Also

- the dimension of the feed-forward hidden layers is $4d$, d being the model/embedding dimension
- the random features for the FAVOR+ attention mechanism (see Subsection 2.4.2) are redrawn every 1000 steps

Note that while we try to have the same hyperparameters in all three models for a more fair comparison, we are limited in the decoupled model by the hyperparameters of the pretrained distilGPT-2. Specifically, it has been trained with a dimension of 768 and 12 attention heads. We use 6 heads for the encoder and decoder, since the dimension

has to be divisible by the number of heads. Using a dimension larger than 768 on all models was not an option, since we could not fit the full instrumental input model in a single GPU.

5.1.2 Training Hyperparameters

Training Hyperparameters	All Instrumental to Vocals Models	Lyrics Language Model
Batch size	8	64
Mini-Batch size	1	8
Gradient Clipping	0.5	
Optimizer	AdamW	
Betas	0.9, 0.98	
Learning Rate	0.001	
Weight Decay	0.1	
Linear Warmup steps	100	

Table 5.3: *Training Hyperparameters for all three Instrumental to Vocals Models (Seq2Seq (full), Seq2Seq (chords), Decoupled (chords)) and the finetuned on Lyrics Language Model*

In Table 5.3, the reader can see the configuration we used for the training of our models. We used the same hyperparameters on all models except from the finetuning of our language model, in which we used a much larger batch size because of the small memory requirements.

We manage to use a batch size 8 times the mini-batch size by performing gradient accumulation (updating only every 8 steps).

We use gradient clipping (introduced in [63] and analyzed thoroughly in [95]). Gradient clipping solves the exploding gradient problem and smoothens the gradient landscape, by restricting the gradient norm: if $\|\mathbf{g}\| > v$ then $\mathbf{g} \leftarrow \frac{\mathbf{g}^v}{\|\mathbf{g}\|}$ where v is a norm threshold, 0.5 here.

We use the *AdamW* optimizer (see Subsection 2.2.3) with beta parameters of 0.9 and 0.98. We report that using this optimizer instead of regular Adam led to a much faster convergence. We use a λ of 0.1 for the weight decay.

Lastly, we perform a very quick warm-up scheduling on the learning rate. We start with a learning rate of 0 and gradually increase it until the final value of 0.001. Notice that since we are using pre-norm, a slow warm-up, or small learning rates are not required.

5.1.3 Dataset size and total steps

We train all three instrumental to vocals models with same size dataset, using a 90% of the dataset for training and the 10% for validation and evaluating the generation performance. We also train all models for the same amount of training steps, since we have convergence at around this point and for fair comparison.

Likewise, we split our dataset to 90% for training and 10% for validation and evaluating the generation performance, when finetuning the pre-trained distilGPT-2 on lyrics. Since, the dataset we have assembled is very large and finetuning does not need to be extensive, we train it for a little more than one epoch. Details are presented on Table 5.4.

The reader can find the training and validation loss for all models plotted with time in Appendix A.

Training	All Instrumental to Vocals Models	Lyrics Language Model
Train size	7654	237299 (29663 batches)
Validation size	851	26367
Epochs	6	1
Total Steps	45923	33370

Table 5.4: *Size of Train and Validation Datasets (90/10 split), Epochs and Total Training Steps for all three Instrumental to Vocals Models (Seq2Seq (full), Seq2Seq (chords), Decoupled (chords)) and the finetuned on Lyrics Language Model*

5.1.4 Training and Inference Speed

At this point we document the training and inference times for our models. Tables 5.5 and 5.6 present the training speed, using the average of processed training samples per second and providing an estimate of the total training time based on that. We notice that reducing the instrumental input sequence is very important and leads to much faster training (almost 3 times faster). Notice that we have an average reduction of 75% on the length of the input sequence (see Table 3.2). The decoupled architecture is slower than the simple sequence-to-sequence on chords (as expected), since it is more complex because of the extra sublayers and the language model part. But still, larger sequence lengths impact the training speed more. Also, we notice that finetuning the lyrics language model is a fast process.

Next, we present the inference speed in generated tokens per second. In the sequence-to-sequence architectures we calculate speed on the tokens of the unique output sequence, while in the decoupled architecture we take an average on the sum of lyrics and vocal melody tokens. It should be noted that the decoding strategy that we analyzed in 4.1.3 makes generation slower, which is why the decoupled architecture that uses these decoding constrains only on the vocal melody decoder is faster.

Speed/Duration	Seq2Seq (Full)	Seq2Seq (Chords)	Decoupled (Chords)
Samples/second (average)	0.171	0.473	0.259
Total hours of training	74.59	26.96	49.25
Generated tokens/second	5.89	6.71	9.09

Table 5.5: *Training speed, total training duration and inference speed for Sequence-to-Sequence with Full Instrumental Input, Sequence-to-Sequence with Reduced Chords Instrumental Input and Decoupled with Reduced Chords Instrumental Input (times measured in an NVIDIA Tesla T4 GPU)*

5.2 Regarding the Generation Evaluation Metrics

To get an estimate of our model's performance during training we need to evaluate the generation capabilities, besides the loss on the validation dataset. A metric most

Speed/Duration	Lyrics Language Model
Samples/second (average)	3.59
Total hours of training	2.58

Table 5.6: *Training speed and total training duration for the Language Model that we finetune on Lyrics*

commonly used in Machine Translation (a common seq2seq task) is the Bilingual Evaluation Understudy (BLEU). BLEU metric is constrained between 0 and 1, and larger values indicate that a generated sequence is closer to ground truth. The generated sequence is evaluated based on n-gram matches to ground truth and can be defined as:

$$BLEU = BP \cdot \exp\left(\sum_{n=1}^{N}(w_n \log p_n)\right) \tag{5.1}$$

where BP is the brevity penalty on the length of the utterance, p_n is the probability that the n-grams in a generated response occur in ground truth, N is the max number of grams and w_n is the weight for each n-gram.

The reader can find BLEU metric plotted with time in Appendix A. Arguably, its values are small and irregular, and there is no significant improvement with time. This does not surprise us, since vocals generation based on instrumental input is by no means a task with a "correct answer". Given the same instrumental song, a thousand songwriters would come up with a thousand different vocal melodies and lyrics.

For the above reason we devise a much weaker metric, that nevertheless is informative of the generation quality. We talked about the structure of the vocal sequence in detail in Subsection 4.1.3. There are specific rules/constraints regarding the possible event types, based on a few previous event types of the sequence. To get an estimate of how well the generated sequence obeys to these rules, we remove the decoding constrains we mentioned and instead we generate unconstrained sequences. Then, using the same algorithm we described, we count how many of these events obey the structural constraints and we divide by the total length of the sequence getting a value between 0 and 1. We call this metric Valid Structure Metric (VSM) and we write:

$$VSM = \frac{\sum_{i=1}^{m} \mathbf{v}(\mathbf{O}_i, \mathbf{O}_{1:\mathrm{m}})}{m} \tag{5.2}$$

where $\mathbf{v}$ is a function that is equal to 1 if the current event is valid and equal to 0 otherwise, and $\mathbf{O}_{1:\mathrm{m}}$ is a vocal sequence of length m.

We notice that although VSM takes values close to one early in training, it is not consistently 1 and has some negatives spikes, especially when using the full instrumental input. So, the need for constraining on generation is justified.

5.3 Comparison

To recap what we noted above and what we see in the plots in Appendix A, reducing the size of the instrumental input is very important for the training speed of the model.

When examining the plotting of the losses we also note that validation loss converges to a smaller value.

Regarding the decoupled architecture, we note that, while the model is more complex than the simple sequence-to-sequence one, the training speed is not affected as much as it is affected from larger instrumental input sequences. Also, we see from the VSM metric that generated sequences with the decoupled architecture have a valid structure metric more consistently close to 1, with much fewer downward spikes.

5.4 Qualitative Evaluation

In Subsection 5.2 we discussed the generation evaluation metrics we used to get an estimation of the quality of generation sequences. It is very hard to formulate musicality and because of this, automatic evaluation of the results is not representative of the resulting quality. BLEU metric can not give us a good estimation and the metric we devised depends only on the structure of the sequences and gives us no estimation for the melodic or lyrical content.

For the above reasons, and since musicality is subjective, we conduct a qualitative evaluation of our results to compare the different models we developed. The raw format of the sequences is impossible to judge, so we convert it to a much more human friendly, audio format. We take the following steps:

- We randomly pick 5 MIDI instrumentals from the evaluation dataset.
- We synthesize audio from the above instrumentals, using code from a MIDI processing library[4] and the software synthesizer FluidSynth[5].
- We generate vocal melody and lyrics with our 3 models (seq2seq with full input, seq2seq with reduced (chords) input and decoupled with reduced input).
- We use Mellotron (see Subsection 2.7) to perform singing voice synthesis on the results (using custom code to make our event sequence format compatible).
- We mix the generated singing with the instrumental audio and get a total of 15 audio files.
- We also include the generated lyrics. Note that for the seq2seq architecture, we train a phoneme to grapheme model[6] on the CMU pronouncing dictionary to turn the phonemes to a more friendly text format.

Then we ask **15** people to compare the results of the 3 models, for each one of the 5 tracks. We shuffle the audio files for each track to avoid bias in the answers.

For each track, the participants have to choose one or more audio files that they believe were better in terms of:

[4] https://github.com/craffel/pretty-midi
[5] https://www.fluidsynth.org
[6] https://github.com/cmusphinx/cmudict

1. **Rhythmic/Melodic Quality**: how musical or interesting is the vocal part, in terms of rhythm and melody
2. **Relation to the Music**: how well the vocal part fits with the instrumental, both in terms of harmony (in tune) and synchronization (in tempo)
3. **Lyrical Content**: the quality of the generated lyrics

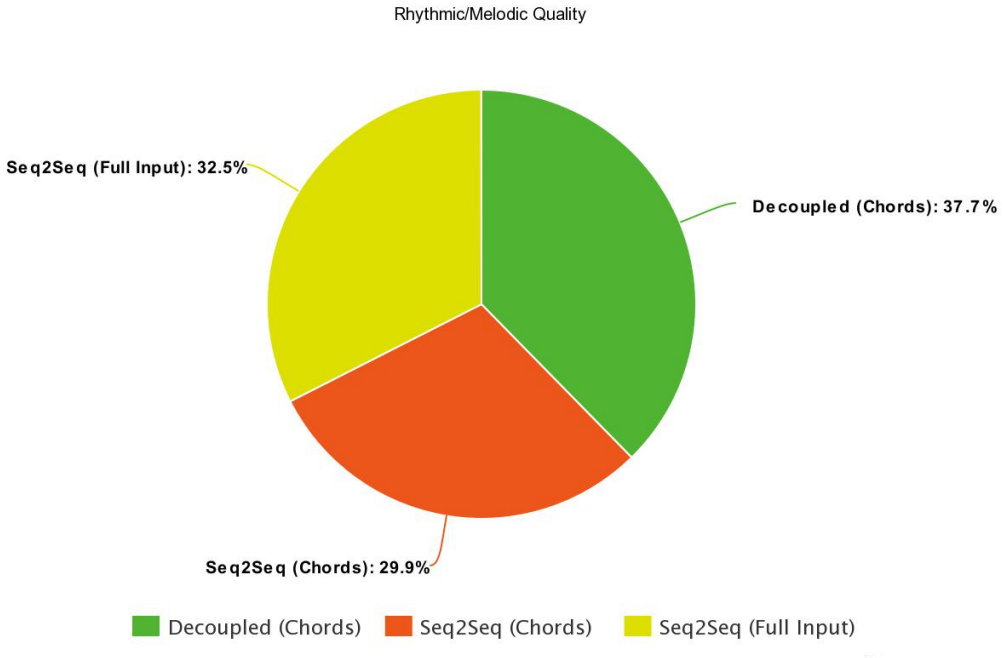


Figure 5.1: *Rhythmical/Musical Quality according to our Qualitative Evaluation Study*

We gather their answers and analyze them. The reader can see the results in Figures 5.1, 5.2 and 5.3.

We conclude that the decoupled architecture is definitely better when it comes to lyrics, as expected. Also, it significantly outperforms the seq2seq architectures, regarding the relation of vocals to music. Finally, it is slightly better to generating musical vocal sequences. Comparing the two seq2seq models we note that they are quite similar, with the one using chords being a little better regarding instrumental/vocal relation and a little worse on the quality of the vocal melody.

Since we have used a relatively small number of tracks and participants in our study, we cannot come to a conclusion with very much confidence.

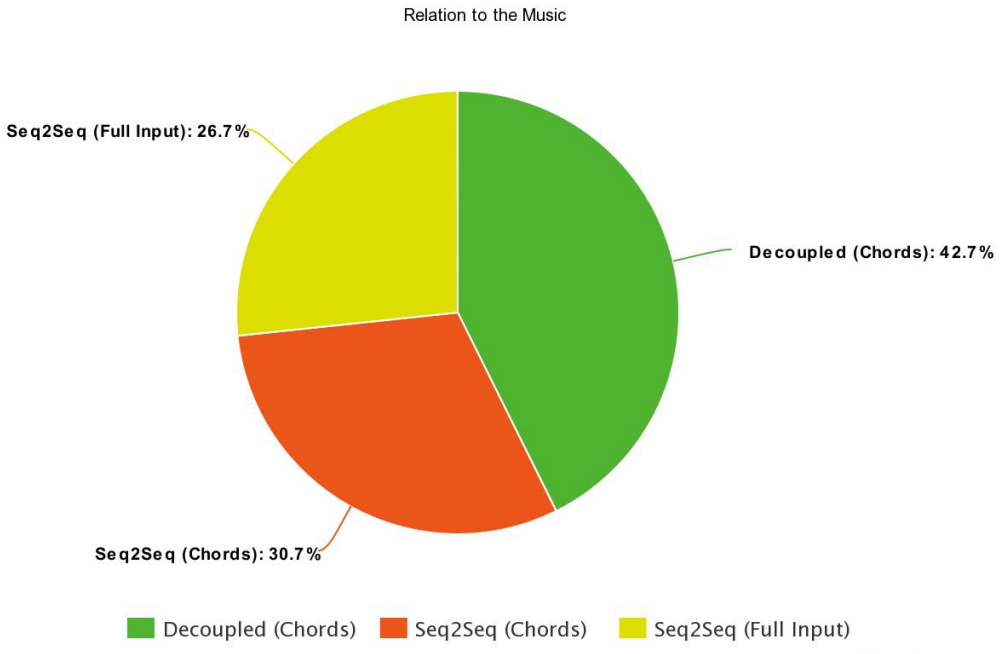


Figure 5.2: *Relation to the Music according to our Qualitative Evaluation Study*

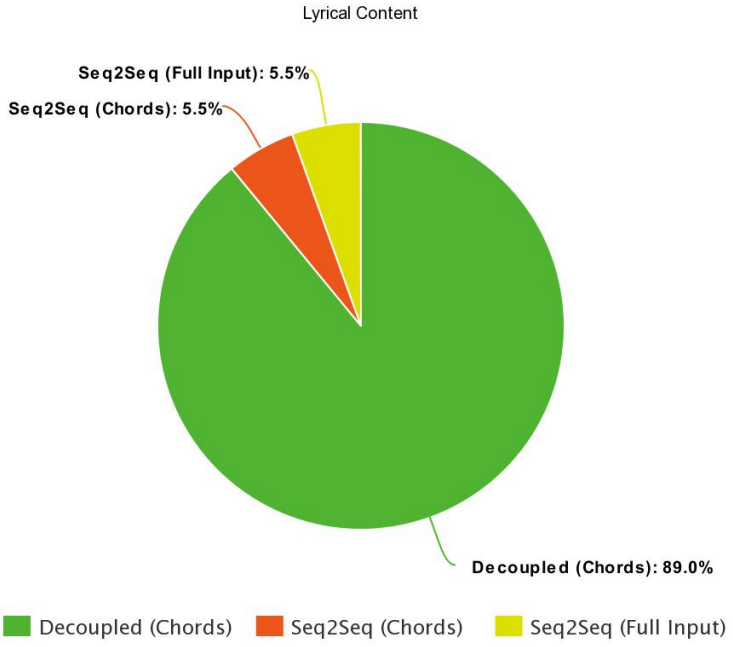


Figure 5.3: *Lyrical Content according to our Qualitative Evaluation Study*

Chapter 6

Conclusions and Future Work

In this last Chapter we will wrap up our work with some conclusions. Also, we present some ideas that could improve our approach to the task we explore or give us new and exciting directions to gaze at.

6.1 Conclusions

In this thesis we explore a very interesting task, that of lyrics and vocal melody generation for a specific instrumental music piece. We focus on symbolic music and specifically on the MIDI format. To the best of our knowledge, our work is the first to incorporate the musical context of the accompaniment, when studying the generation of lyrics or vocal melody. Our research, and the code that we release along, make it possible to generate end-to-end a complete vocal performance for any instrumental MIDI file we can imagine, for the first time. We turn the generated vocals from a symbolic representation to audio, using a singing voice synthesis model.

We propose a text event approach, inspired by the recent, successful work in the field of symbolic music generation. Based on a dataset that consists of publicly available MIDI files, we separate vocals and lyrics from the instrumental part and create a dataset in a more structured format, suitable for the problem we research. We also make available the code to recreate it from the original.

We experiment with two types of model architectures. In the first one, we merge lyrics with vocal melody. We have two sequences, one consisting of instrumental events, the input sequence, and another one that consists of vocal melody events, alongside lyrics syllables, which is the output. Then, we model conditional vocal melody and lyrics generation as a sequence-to-sequence task, using a typical encoder-decoder Transformer architecture.

The second type of model architecture that we experiment with is one that we introduce, inspired by research in multi-source and multi-modal sequence modelling, and motivated by the benefits of incorporating prior knowledge from language models. We separate lyrics from vocal melody and use a Transformer architecture that consists of an instrumental encoder, one vocal melody decoder and one lyrics decoder part. For the lyrics decoder we use a pretrained GPT-2 model, which we finetune on plain lyrics. The encodings of the instrumental part are used to encode both the lyrics and the vocal melody

decoder, using cross-attention sublayers in the latter two. We also use the encodings of the lyrics language model part, before decoding it to text tokens, to condition the vocal melody decoder, by adding a second cross-attention sublayer to it. The generated lyrics and vocal melody sequences are merged into one. We call this architecture decoupled.

The sequences that we model can be up to 50 times longer than the barrier of 1024 tokens that most Transformer based architectures can model, because of the quadratical memory requirements of the regular softmax attention mechanism. For this reason, we use the linear FAVOR+ attention mechanism introduced recently, along with other enhancements to reduce memory footprint, such as reversible layers. Furthermore, we apply a music analysis step, that reduces the instrumental input up to 80%, by substituting music notes with chords. We also substitute notes and chords with roman numerals, a music key-independent representation, for more robustness. We find that this reduced representation substantially reduces training time.

Finally, we conduct a qualitative comparative evaluation study to compare the results from three models: **(a)** a sequence-to-sequence model trained with full instrumental notes input, **(b)** a sequence-to-sequence model trained with reduced instrumental chords and **(c)** a decoupled architecture model trained with reduced instrumental chords. We find that the latter significantly outperforms the others in terms of lyrics content, as expected, while also being slightly superior in the musical quality of the generated vocal melody as well as on its relation to the accompaniment.

We hope that with work we can encourage more researchers to study conditional vocal melody and lyrics generation and bring new ideas to this exciting field. Also, we are confident that our work can be used by artists and others to create and inspire.

6.2 Future Work

In the future, we aim to improve the performance on the task we study here by experimenting with some different settings, regarding the sequence representations and the model architecture. A difference in representation, that is easy to incorporate in the existing pipeline, is substituting MIDI ticks as the time unit with musical durations (for example using a 64th note duration as base). Another idea is to keep only notes that belong to the derived music key, or further simplifying chords, when doing chordal reduction.

Regarding improvements in the model architecture, we could experiment with other methods to reduce memory footprint, for example using gradient checkpointing [13] instead of reversible layers, since reversibility enforces slower training speed. Some other ideas regard the decoupled architecture we introduced. We can experiment with a different setup, such as conditioning lyrics to vocal melody, in contrast to the current model. It is also worth studying whether the conditioning of lyrics to the instrumental sequence is important, by performing an ablation study.

To reduce the complexity of the decoupled architecture we could also try using other ways to incorporate prior knowledge from a language model. Since the vocal tokenization of lyrics, meaning the mapping of lyrics text to notes, is different from commonly used

tokenization algorithms, we cannot think of a more efficient way to use a pretrained language model. Instead, we can train a language model, using a syllable based tokenization method, on a large corpus of english text from the beginning. Training a language model with restricted resources could mean that it will not perform as well as a pretrained state of the art model, but using other ways to leverage it, could improve the performance overall. An elegant solution that performs a type of knowledge distillation technique, by adding a regularization term depending on the language model output distribution, is showcased in [5]. It can be easily utilized in our model if the regularization applies only to syllable tokens.

Another improvement, comparable to the usage of prior knowledge for lyrics, would be the pretraining of the instrumental encoder, with an unsupervised training objective, like bidirectional language representation model encoders (for example BERT). Since the amount of instrumental-only MIDI tracks is very large, this is worth exploring.

Finally, we aspire to cross the limits of symbolic music. DALI [56] is a large dataset that contains audio tracks, with synchronized lyrics and their time-aligned vocal melody notes. Its format is very close to the dataset we built, with the difference of containing raw music audio information. Using source separation to isolate vocals and extracting temporal audio features and using them instead of the symbolic instrumental embeddings, is one straightforward way to use one of our existing architectures. Finally, we could use this dataset to build an end-to-end audio vocals generation pipeline, since singing audio is contained with synchronized note and lyrics mapping.

Παραρτήματα

Appendix A

Training, Validation and Generation Metrics for our three models: Seq2Seq with Full Input, Seq2Seq with Reduced Chords Input, Decoupled with Reduced Chords Input

In this appendix, we present diagrams for: **(a)** Training and Validation Loss, **(b)** BLEU metric, **(c)** Valid Structure Metric. These metrics are explained in Chapter 5 along with a comparison for our three models based on these results.

A.1 Sequence-to-Sequence with Full Instrumental Input

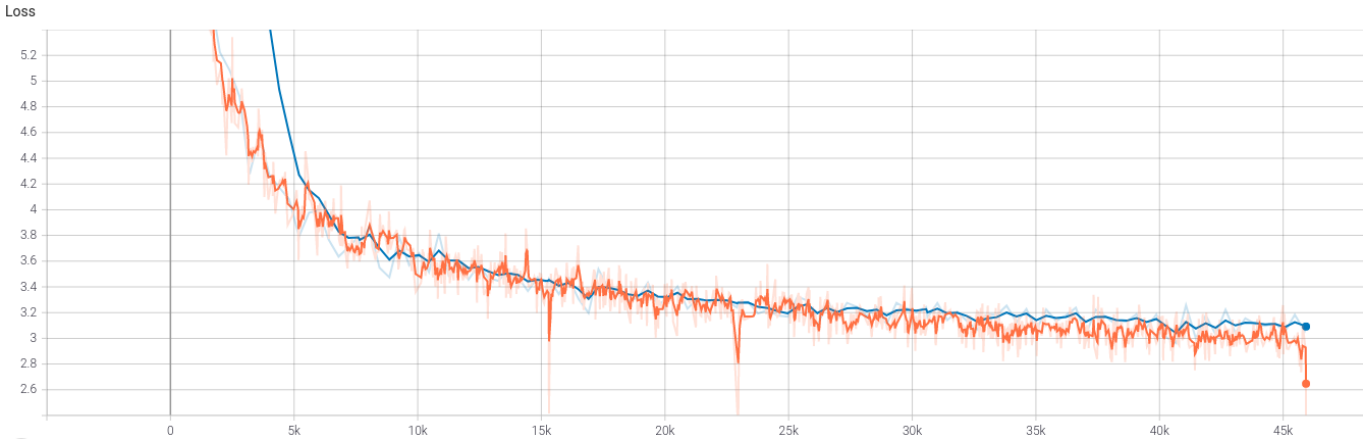


Figure A.1: *Train(orange) and Validation(blue) Loss for simple seq2seq architecture with full input - training for 6 epochs*

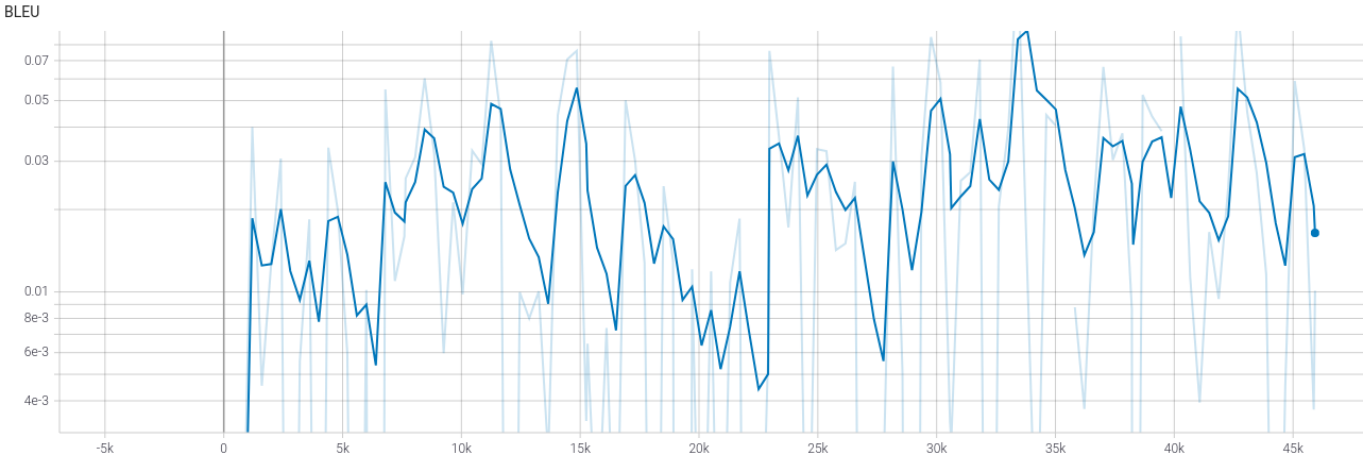


Figure A.2: *BLEU metric for simple seq2seq architecture with full input*

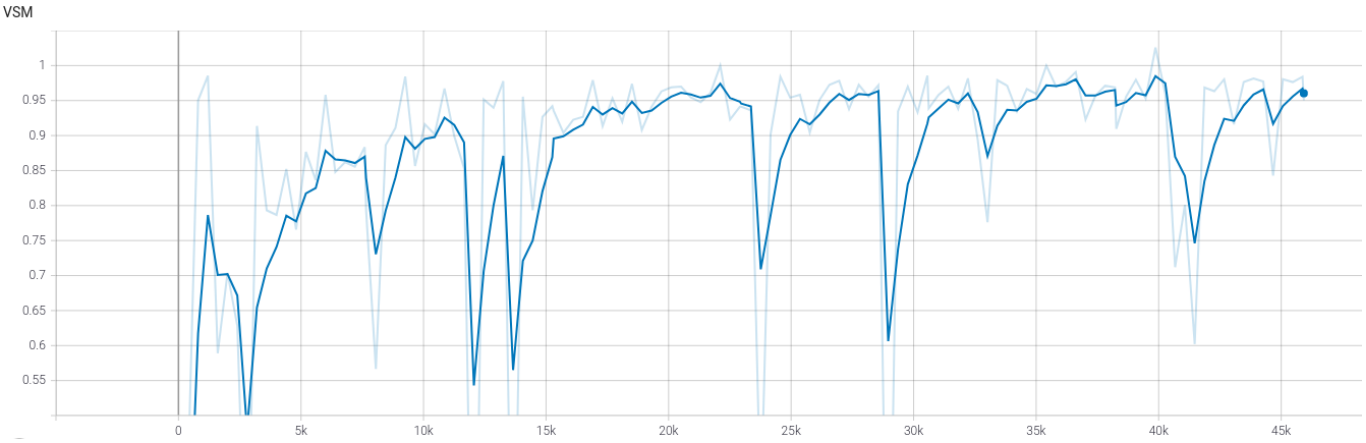


Figure A.3: *Valid Structure Metric of Vocal Melody sequence for simple seq2seq architecture with full input*

A.2 Sequence-to-Sequence with Reduced Chords Instrumental Input

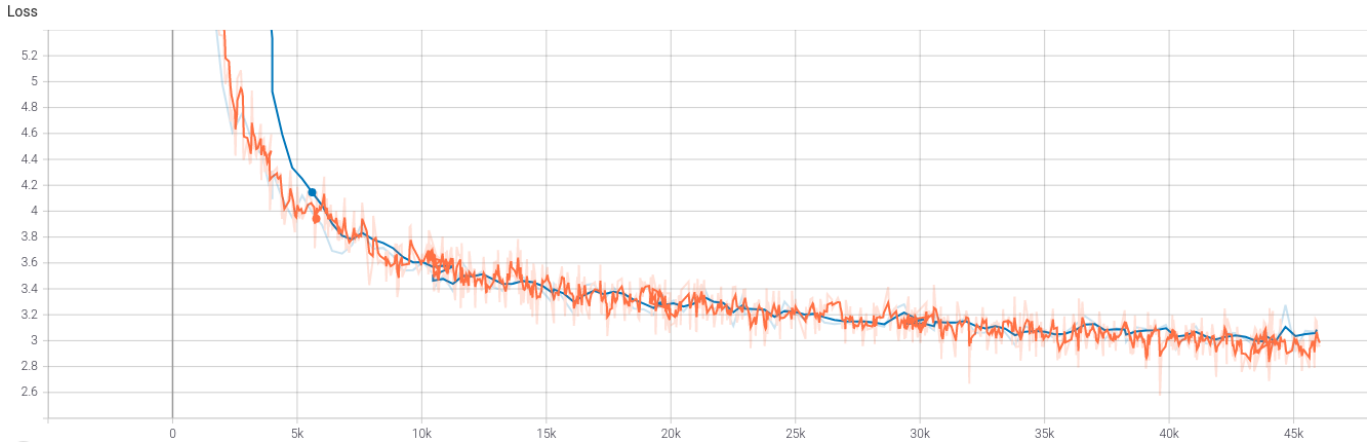


Figure A.4: *Train(orange) and Validation(blue) Loss for simple seq2seq architecture with reduced input - training for 6 epochs*

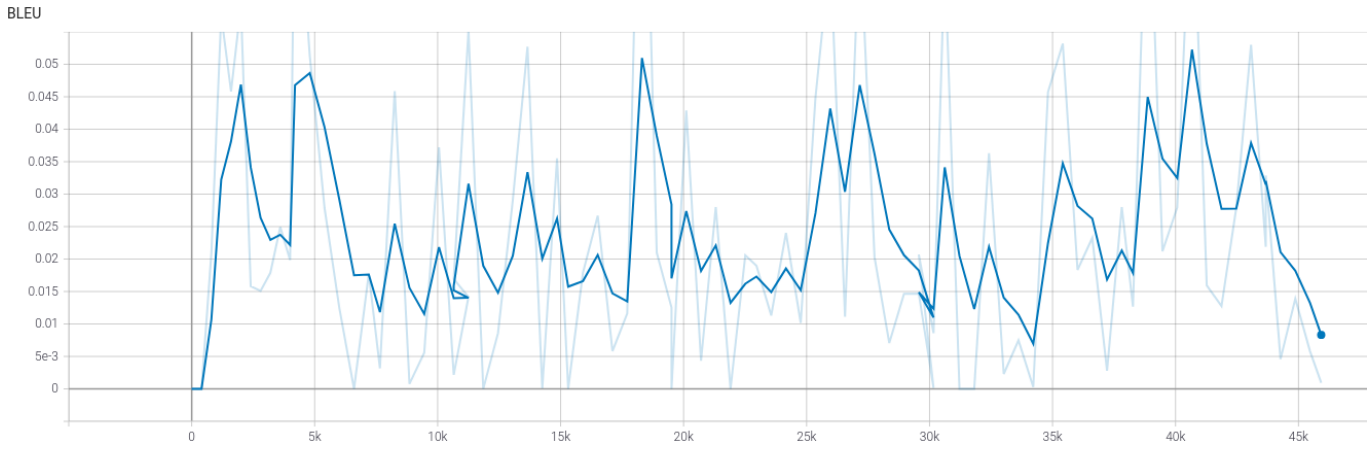


Figure A.5: *BLEU metric for simple seq2seq architecture with reduced input*

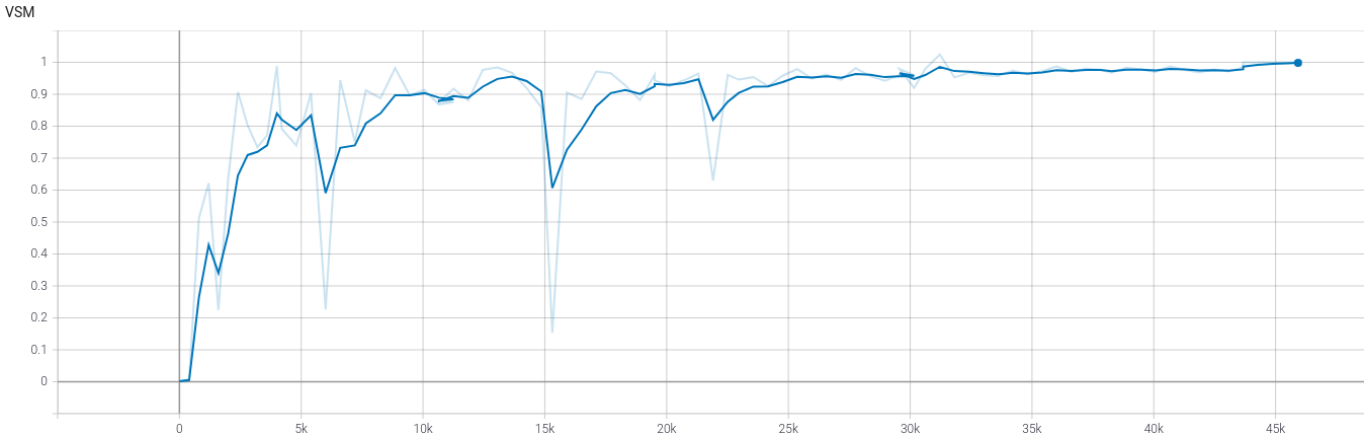


Figure A.6: *Valid Structure Metric of Vocal Melody sequence for simple seq2seq architecture with reduced input*

A.3 Decoupled with Reduced Chords Instrumental Input

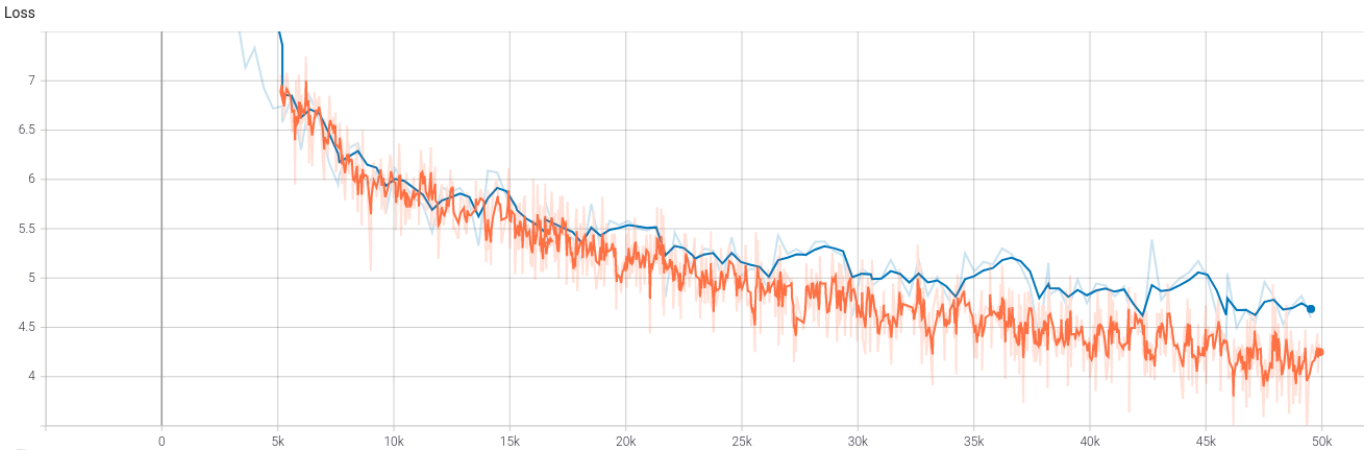


Figure A.7: *Train(orange) and Validation(blue) Loss for decoupled architecture with reduced input (sum of lyrics and melody losses) - training for 6 epochs*

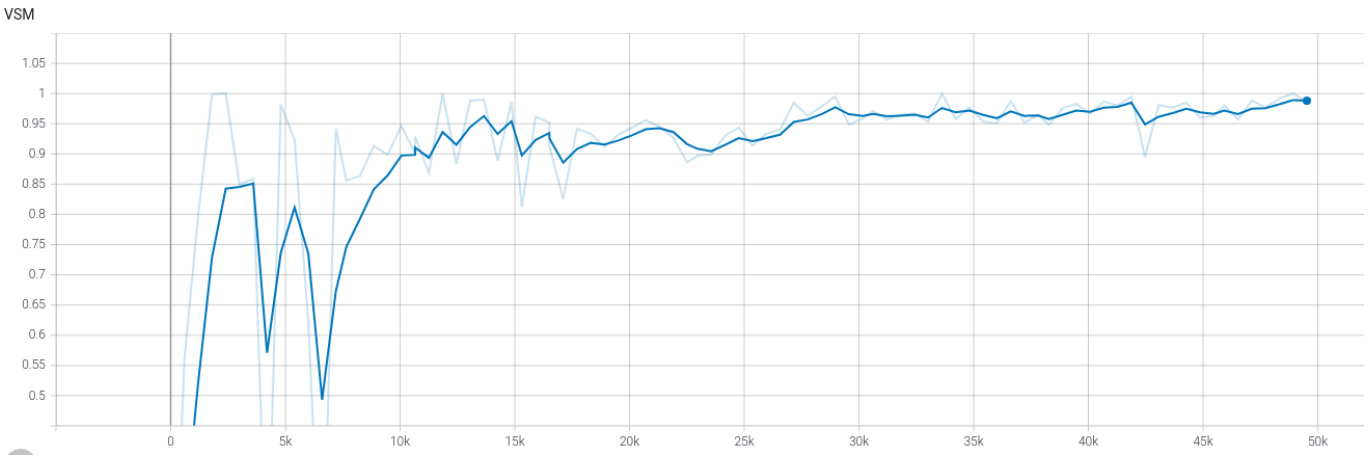


Figure A.8: *Valid Structure Metric of Vocal Melody sequence for decoupled architecture with reduced input*

Appendix B

Examples of the Generated Sequences

In this appendix, we present some examples of the lyrics that are generated with our finetuned language model **before** and **after** incorporating it in our decoupled architecture. The reader can also find some audio examples of generated vocals, mixed with the instrumental part, on SoundCloud.

```
Now if you're fighting now
And if it's just come along
Now it's always near me
We love you
And if it would have come along
I'll cross a crossroad
And then when the light starts my feet

And if it's on it's gone along
And if it's gone and gone
Through the cold simply wasn't gone
So if it still have me

And if it's just come along
And if it's sure to conquer well
I'll cross the crossroad
If it's home running on my feet

And if we come along
To the moon is the road
Did you ever cross beating me along
And if it's always time
It's sure to cross me
With all along up
Yes I'll cross the cross the cross
Then if it would cross
And it's always on and on my mind

And if it's always
And if it's always<|endoftext|>
```

```
You bought the kid made me a hand
You moved us against my sand
A eyed son with you herself in a bottle of sand
You can still remember the hand that he used to remember

You used to raise our hands
Used to raise your hands to jigold feet back, wooden shoes
You used to raise your legs
You are fire fill the air
Used to raise your soul
You were running hot and made
Used to raise a broken wooden punch
You called me from beating heart
You used to raise your air
You were taking your tongue, changing your heart
You took the bullet
You were burning gold
Don't you just ask about you gave its attention

You need any evidence
But it's burning
You were burning fire
You were just burning
You're made of me burning for your heart
But it's plain
You changed your little more step closer
You are with your heart
But we're burning, you<|endoftext|>
```

Figure B.1: *Generated examples of the distilGPT-2 language model finetuned on lyrics*

```
You fall in the one say
when I get high
whenever I fall to
Tale as the clouds
all I get high
And the sun
my feet runnin' when the sun goes
till the evening's high
Just my fall

Two people gettin' kinda blue
Cause you're my poor boy
and I get high
You gettin' high
with ev'ryone's
with lovin' nobody bless

you'll cry
only get my high with a poor poor boy blues
Go crazy then
and I'll shine
Once I look,
But darling
```

```
Don't forget how you forget this girl
when I'm afraid of love
Yeah,  oh I never met you now
But when he's over so cold
one that everything I feel
Yeah doesn't matter how I truly feel
for you now
yeah yeah
it's so good to move on
One day in the distance and all alone
onely day it never seemed so good
here to reason
one day
never needed time but suddenly I'm so good to win
yeah yeah yeah
cause I'm so good to doubt
It's so good yeah
it's so good to think so good
```

Figure B.2: *Generated examples of the language model in the decoupled architecture*

Βιβλιογραφία

[1] Martin Arjovsky, Soumith Chintala και Léon Bottou. *Wasserstein GAN*, 2017.

[2] Dzmitry Bahdanau, Kyunghyun Cho και Yoshua Bengio. *Neural Machine Translation by Jointly Learning to Align and Translate*, 2016.

[3] Jimmy Lei Ba, Jamie Ryan Kiros και Geoffrey E. Hinton. *Layer Normalization*, 2016.

[4] Hangbo Bao, Shaohan Huang, Furu Wei κ.ά. . *Neural Melody Composition from Lyrics*, 2018.

[5] Christos Baziotis, Barry Haddow και Alexandra Birch. *Language Model Prior for Low-Resource Neural Machine Translation*, 2020.

[6] R. Bellman και R. Kalaba. *On adaptive control processes*. *IRE Transactions on Automatic Control*, 4(2):1–9, 1959.

[7] Amineben khalifa και Hichem Frigui. *Multiple Instance Fuzzy Inference Neural Networks*. 2016.

[8] Thierry Bertin-Mahieux, Daniel P.W. Ellis, Brian Whitman και Paul Lamere. *The Million Song Dataset*. *Proceedings of the 12th International Conference on Music Information Retrieval (ISMIR 2011)*, 2011.

[9] Tom B. Brown, Benjamin Mann, Nick Ryder κ.ά. . *Language Models are Few-Shot Learners*, 2020.

[10] Cristian Buciluundefined, Rich Caruana και Alexandru Niculescu-Mizil. *Model Compression*. *Proceedings of the 12th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, KDD '06, σελίδα 535–541, New York, NY, USA, 2006. Association for Computing Machinery.

[11] Pritish Chandna, Merlijn Blaauw, Jordi Bonada και Emilia Gomez. *WGANSing: A Multi-Voice Singing Voice Synthesizer Based on the Wasserstein-GAN*. *2019 27th European Signal Processing Conference (EUSIPCO)*, 2019.

[12] Mia Xu Chen, Orhan Firat, Ankur Bapna κ.ά. . *The Best of Both Worlds: Combining Recent Advances in Neural Machine Translation*. *Proceedings of the 56th Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, σελίδες 76–86, Melbourne, Australia, 2018. Association for Computational Linguistics.

[13] Tianqi Chen, Bing Xu, Chiyuan Zhang και Carlos Guestrin. *Training Deep Nets with Sublinear Memory Cost*, 2016.

[14] Yihao Chen και Alexander Lerch. *Melody-Conditioned Lyrics Generation with SeqGANs*, 2020.

[15] Rewon Child, Scott Gray, Alec Radford και Ilya Sutskever. *Generating Long Sequences with Sparse Transformers*, 2019.

[16] Kyunghyun Cho, Bartvan Merrienboer, Caglar Gulcehre κ.ά. . *Learning Phrase Representations using RNN Encoder-Decoder for Statistical Machine Translation*, 2014.

[17] Krzysztof Choromanski, Valerii Likhosherstov, David Dohan κ.ά. . *Rethinking Attention with Performers*, 2021.

[18] Corinna Cortes και Vladimir Vapnik. *Support-Vector Networks*. *Mach. Learn.*, 20(3):273–297, 1995.

[19] Michael Scott Cuthbert και Christopher Ariza. *Music21: A Toolkit for Computer-Aided Musicology and Symbolic Music Data*. *ISMIR*J. Stephen Downie και Remco C. Veltkamp, επιμελητές, σελίδες 637–642. International Society for Music Information Retrieval, 2010.

[20] G. Cybenko. *Approximation by superpositions of a sigmoidal function*. *Mathematics of Control, Signals, and Systems (MCSS)*, 2(4):303–314, 1989.

[21] Jacob Devlin, Ming Wei Chang, Kenton Lee και Kristina Toutanova. *BERT: Pre-training of Deep Bidirectional Transformers for Language Understanding*, 2019.

[22] Prafulla Dhariwal, Heewoo Jun, Christine Payne κ.ά. . *Jukebox: A Generative Model for Music*, 2020.

[23] Hao Wen Dong, Wen Yi Hsiao, Li Chia Yang και Yi Hsuan Yang. *MuseGAN: Multi-track Sequential Generative Adversarial Networks for Symbolic Music Generation and Accompaniment*, 2017.

[24] Jonas Gehring, Michael Auli, David Grangier κ.ά. . *Convolutional Sequence to Sequence Learning*, 2017.

[25] Mor Geva, Roei Schuster, Jonathan Berant και Omer Levy. *Transformer Feed-Forward Layers Are Key-Value Memories*, 2020.

[26] Aidan N. Gomez, Mengye Ren, Raquel Urtasun και Roger B. Grosse. *The Reversible Residual Network: Backpropagation Without Storing Activations*, 2017.

[27] Ian J. Goodfellow, Jean Pouget-Abadie, Mehdi Mirza κ.ά. . *Generative Adversarial Networks*, 2014.

[28] Alex Graves, Greg Wayne και Ivo Danihelka. *Neural Turing Machines*, 2014.

[29] The Speech Group. *The CMU Pronouncing Dictionary*, 2014.

[30] Kaiming He, Xiangyu Zhang, Shaoqing Ren και Jian Sun. *Deep Residual Learning for Image Recognition*, 2015.

[31] Dan Hendrycks και Kevin Gimpel. *Gaussian Error Linear Units (GELUs)*, 2020.

[32] Lejaren Arthur Hiller και Leonard M. Isaacson. *Experimental Music; Composition with an Electronic Computer*. Greenwood Publishing Group Inc., USA, 1979.

[33] Geoffrey Hinton, Oriol Vinyals και Jeff Dean. *Distilling the Knowledge in a Neural Network*, 2015.

[34] Geoffrey E. Hinton, Nitish Srivastava, Alex Krizhevsky κ.ά. . *Improving neural networks by preventing co-adaptation of feature detectors*, 2012.

[35] Sepp Hochreiter και Jürgen Schmidhuber. *Long short-term memory*. *Neural computation*, 9(8):1735–1780, 1997.

[36] Cheng Zhi Anna Huang, Ashish Vaswani, Jakob Uszkoreit κ.ά. . *Music Transformer: Generating Music with Long-Term Structure*. *arXiv preprint arXiv:1809.04281*, 2018.

[37] Sergey Ioffe και Christian Szegedy. *Batch Normalization: Accelerating Deep Network Training by Reducing Internal Covariate Shift*, 2015.

[38] Andrej Karpathy. *Convolutional neural networks for visual recognition*.

[39] Angelos Katharopoulos, Apoorv Vyas, Nikolaos Pappas και François Fleuret. *Transformers are RNNs: Fast Autoregressive Transformers with Linear Attention*, 2020.

[40] Diederik P. Kingma και Jimmy Ba. *Adam: A Method for Stochastic Optimization*, 2017.

[41] Diederik P Kingma και Max Welling. *Auto-Encoding Variational Bayes*, 2014.

[42] Nikita Kitaev, Łukasz Kaiser και Anselm Levskaya. *Reformer: The Efficient Transformer*, 2020.

[43] John Koopman. *A Brief History of Singing*. http://www2.lawrence.edu/fast/KOOPMAJO/brief.html, 1999. Online; accessed March 2021.

[44] Anders Krogh και John Hertz. *A Simple Weight Decay Can Improve Generalization*. *Advances in Neural Information Processing Systems*J. Moody, S. Hanson και R. P. Lippmann, επιμελητές, τόμος 4. Morgan-Kaufmann, 1992.

[45] Y. Lecun, L. Bottou, Y. Bengio και P. Haffner. *Gradient-based learning applied to document recognition*. *Proceedings of the IEEE*, 86(11):2278–2324, 1998.

[46] Lewis. *Creation by refinement: a creativity paradigm for gradient descent learning networks*. *IEEE 1988 International Conference on Neural Networks*, σελίδες 229–233 ολ.2, 1988.

[47] Jindřich Libovický, Jindřich Helcl και David Mareček. *Input Combination Strategies for Multi-Source Transformer Decoder*. *Proceedings of the Third Conference on Machine Translation: Research Papers*, σελίδες 253–260, Brussels, Belgium, 2018. Association for Computational Linguistics.

[48] Ilya Loshchilov και Frank Hutter. *Decoupled Weight Decay Regularization*, 2019.

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

[50] Xu Lu, Jie Wang, Bojin Zhuang κ.ά. . *A Syllable-Structured, Contextually-Based Conditionally Generation of Chinese Lyrics*, 2019.

[51] Yiping Lu, Zhuohan Li, Di He κ.ά. . *Understanding and Improving Transformer From a Multi-Particle Dynamic System Point of View*, 2019.

[52] Gurunath Reddy Madhumani, Yi Yu, Florian Harscoët κ.ά. . *Automatic Neural Lyrics and Melody Composition*, 2020.

[53] Swetha Mandava, Szymon Migacz και Alex Fit Florea. *Pay Attention when Required*, 2020.

[54] Warren S. McCulloch και Walter Pitts. *A logical calculus of the ideas immanent in nervous activity*. *The bulletin of mathematical biophysics*, 5(4):115–133, 1943.

[55] Soroush Mehri, Kundan Kumar, Ishaan Gulrajani κ.ά. . *SampleRNN: An Unconditional End-to-End Neural Audio Generation Model*, 2017.

[56] Gabriel Meseguer-Brocal, Alice Cohen-Hadria και Geoffroy Peeters. *DALI: A Large Dataset Of Synchronized Audio, Lyrics And Notes, Automatically Created Using Teacher-student Machine Learning Paradigm*. *19th International Society for Music Information Retrieval Conference*, Paris, France, 2018.

[57] Tomas Mikolov, Kai Chen, Greg Corrado και Jeffrey Dean. *Efficient Estimation of Word Representations in Vector Space*, 2013.

[58] Thomas M. Mitchell. *Machine Learning*. McGraw-Hill, Inc., USA, 1η έκδοση, 1997.

[59] Toan Q Nguyen και Julian Salazar. *Transformers without tears: Improving the normalization of self-attention*. *arXiv preprint arXiv:1910.05895*, 2019.

[60] Nikola I. Nikolov, Eric Malmi, Curtis G. Northcutt και Loreto Parisi. *Rapformer: Conditional Rap Lyrics Generation with Denoising Autoencoders*, 2020.

[61] N. J. Nilsson. *Learning Machines*. McGraw-Hill, New York, 1965.

[62] JOSEF NOVAK, Nobuaki Minematsu και Keikichi Hirose. *Phonetisaurus: Exploring grapheme-to-phoneme conversion with joint n-gram models in the WFST framework*. *Natural Language Engineering*, -1:1–32, 2015.

[63] Razvan Pascanu, Tomas Mikolov και Yoshua Bengio. *On the difficulty of training Recurrent Neural Networks*, 2013.

[64] Christine Payne. *MuseNet*, 2019.

[65] Ryan Prenger, Rafael Valle και Bryan Catanzaro. *WaveGlow: A Flow-based Generative Network for Speech Synthesis*, 2018.

[66] Ofir Press, Noah A. Smith και Omer Levy. *Improving Transformer Models by Reordering their Sublayers*, 2020.

[67] Ofir Press και Lior Wolf. *Using the Output Embedding to Improve Language Models*. *Proceedings of the 15th Conference of the European Chapter of the Association for Computational Linguistics: Volume 2, Short Papers*, σελίδες 157–163, Valencia, Spain, 2017. Association for Computational Linguistics.

[68] Alec Radford, Karthik Narasimhan, Tim Salimans και Ilya Sutskever. *Improving language understanding by generative pre-training*.

[69] Alec Radford, Jeff Wu, Rewon Child κ.ά. . *Language Models are Unsupervised Multitask Learners*. 2019.

[70] Colin Raffel. *Learning-Based Methods for Comparing Sequences, with Applications to Audio-to-MIDI Alignment and Matching*. PhD Thesis, 2016.

[71] Jeff Rasley, Samyam Rajbhandari, Olatunji Ruwase και Yuxiong He. *DeepSpeed: System Optimizations Enable Training Deep Learning Models with Over 100 Billion Parameters*. *Proceedings of the 26th ACM SIGKDD International Conference on Knowledge Discovery amp; Data Mining*, KDD '20, σελίδα 3505–3506, New York, NY, USA, 2020. Association for Computing Machinery.

[72] Yi Ren, Xu Tan, Tao Qin κ.ά. . *DeepSinger: Singing Voice Synthesis with Data Mined From the Web*, 2020.

[73] Mark Riedl. *Weird AI Yankovic: Generating Parody Lyrics*, 2020.

[74] Adam Roberts, Jesse Engel, Colin Raffel κ.ά. . *A Hierarchical Latent Vector Model for Learning Long-Term Structure in Music*, 2019.

[75] F. Rosenblatt. *The perceptron: A probabilistic model for information storage and organization in the brain*. *Psychological Review*, 65(6):386–408, 1958.

[76] Sascha Rothe, Shashi Narayan και Aliaksei Severyn. *Leveraging Pre-trained Checkpoints for Sequence Generation Tasks*, 2020.

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

[78] Victor Sanh, Lysandre Debut, Julien Chaumond και Thomas Wolf. *DistilBERT, a distilled version of BERT: smaller, faster, cheaper and lighter*, 2020.

[79] Rico Sennrich, Barry Haddow και Alexandra Birch. *Neural Machine Translation of Rare Words with Subword Units*, 2016.

[80] Rico Sennrich και Biao Zhang. *Revisiting Low-Resource Neural Machine Translation: A Case Study*. *Proceedings of the 57th Annual Meeting of the Association for Computational Linguistics*, σελίδες 211–221, Florence, Italy, 2019. Association for Computational Linguistics.

[81] Jonathan Shen, Ruoming Pang, Ron J. Weiss κ.ά. . *Natural TTS Synthesis by Conditioning WaveNet on Mel Spectrogram Predictions*, 2018.

[82] Ian Simon και Sageev Oore. *Performance RNN: Generating Music with Expressive Timing and Dynamics*. https://magenta.tensorflow.org/performance-rnn, 2017.

[83] Felix Stahlberg, James Cross και Veselin Stoyanov. *Simple Fusion: Return of the Language Model*. *Proceedings of the Third Conference on Machine Translation: Research Papers*, σελίδες 204–211, Brussels, Belgium, 2018. Association for Computational Linguistics.

[84] Ilya Sutskever, Oriol Vinyals και Quoc V. Le. *Sequence to Sequence Learning with Neural Networks*, 2014.

[85] Rafael Valle, Jason Li, Ryan Prenger και Bryan Catanzaro. *Mellotron: Multispeaker expressive voice synthesis by conditioning on rhythm, pitch and global style tokens*, 2019.

[86] Ashish Vaswani, Noam Shazeer, Niki Parmar κ.ά. . *Attention Is All You Need*, 2017.

[87] Olga Vechtomova, Hareesh Bahuleyan, Amirpasha Ghabussi και Vineet John. *Generating lyrics with variational autoencoder and multi-modal artist embeddings*, 2018.

[88] Olga Vechtomova, Gaurav Sahu και Dhruv Kumar. *Generation of lyrics lines conditioned on music audio clips*, 2020.

[89] Kento Watanabe, Yuichiroh Matsubayashi, Satoru Fukayama κ.ά. . *A Melody-Conditioned Lyrics Language Model*. *Proceedings of the 2018 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies, Volume 1 (Long Papers)*, σελίδες 163–172, New Orleans, Louisiana, 2018. Association for Computational Linguistics.

[90] Gerhard Widmer, Maarten Grachten και Stefan Lattner. *Imposing Higher-Level Structure in Polyphonic Music Generation Using Convolutional Restricted Boltzmann Machines and Constraints*. *Journal of Creative Music Systems*, 2(2), 2018.

[91] Li Chia Yang, Szu Yu Chou και Yi Hsuan Yang. *MidiNet: A Convolutional Generative Adversarial Network for Symbolic-domain Music Generation*, 2017.

[92] Ruibin Yuan, Ge Zhang, Anqiao Yang και Xinyue Zhang. *Diverse Melody Generation from Chinese Lyrics via Mutual Information Maximization*, 2020.

[93] Lantao Yu, Weinan Zhang, Jun Wang και Yong Yu. *SeqGAN: Sequence Generative Adversarial Nets with Policy Gradient*, 2017.

[94] Yi Yu και Simon Canales. *Conditional LSTM-GAN for Melody Generation from Lyrics*, 2019.

[95] Jingzhao Zhang, Tianxing He, Suvrit Sra και Ali Jadbabaie. *Why gradient clipping accelerates training: A theoretical justification for adaptivity*, 2020.