Εθνικό Μετσόβιο Πολυτεχνείο
Σχολή Ηλεκτρολόγων Μηχανικών
και Μηχανικών Υπολογιστών
Τομέας Σημάτων, Ελέγχου
και Ρομποτικής

Grounded Visual Question Answering Using Sequence-to-Sequence Modeling

ΔΙΠΛΩΜΑΤΙΚΗ ΕΡΓΑΣΙΑ

ΜΑΡΙΑ-ΒΑΣΙΛΙΚΗ ΝΙΚΑΝΔΡΟΥ

Επιβλέπων : Alexandros Potamianos
Associate Professor NTUA

Αθήνα, Ιούλιος 2019

Εθνικό Μετσόβιο Πολυτεχνείο
Σχολή Ηλεκτρολόγων Μηχανικών
και Μηχανικών Υπολογιστών
Τομέας Σημάτων, Ελέγχου
και Ρομποτικής

Grounded Visual Question Answering Using Sequence-to-Sequence Modeling

ΔΙΠΛΩΜΑΤΙΚΗ ΕΡΓΑΣΙΑ

ΜΑΡΙΑ-ΒΑΣΙΛΙΚΗ ΝΙΚΑΝΔΡΟΥ

Επιβλέπων : Alexandros Potamianos
Associate Professor NTUA

Εγκρίθηκε από την τριμελή εξεταστική επιτροπή την 5η Ιουλίου 2019.

Alexandros Potamianos Associate Professor NTUA	Konstantinos Tzafestas Associate Professor NTUA	Andreas-Georgios Stafylopatis Professor NTUA

Αθήνα, Ιούλιος 2019

Μαρία-Βασιλική Νικάνδρου

Διπλωματούχος Ηλεκτρολόγος Μηχανικός και Μηχανικός Υπολογιστών Ε.Μ.Π.

To my sister, Despina.

Αυτόματη Παραγωγή Απαντήσεων σε Ερωτήσεις Πάνω σε Εικόνες με Χρήση Βαθιάς Επιβλεπόμενης Μάθησης

Εκτεταμένη Περίληψη

Περιγραφή του Προβλήματος

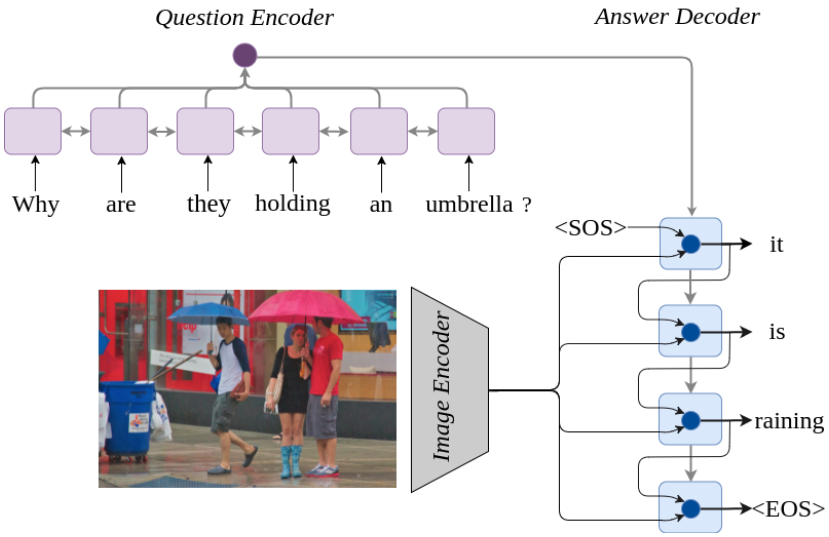
Η αυτόματη απάντηση οπτικών ερωτήσεων (Visual Question Answering, VQA) αποτελεί ένα πρόβλημα που διατέμνει τα πεδία της Επεξεργασίας Φυσικής Γλώσσας (Natural Language Processing, NLP) και της Όρασης Υπολογιστών (Computer Vision, CV). Τα τελευταία χρόνια η χρήση τεχνικών βαθιών νευρωνικών δικτύων έχει οδηγήσει σε σημαντική πρόοδο και στους δύο αυτούς τομείς ωθώντας τους ερευνητές να ασχοληθούν με προβλήματα που αφορούν την πολυτροπική μάθηση. Το VQA ορίζεται ως το πρόβλημα της αυτόματης παραγωγής μιας απάντησης σε φυσική γλώσσα δεδομένου μίας ερώτησης σε ελεύθερη μορφή σχετικά με μία εικόνα. Το πρόβλημα αυτό είναι ιδιαίτερα απαιτητικό καθώς προϋποθέτει τόσο την εξαγωγή χρήσιμης πληροφορίας από την ερώτηση και την εικόνα, όσο και την ικανότητα συλλογιστικής πάνω στην πληροφορία αυτή με απώτερο στόχο την πρόβλεψη μίας ορθής απάντησης. Στην παρούσα διπλωματική, επικεντρωνόμαστε στην παραγωγή απαντήσεων ελεύθερης μορφής για ερωτήσεις ανοιχτού τύπου. Καθώς η θεματική των οπτικών ερωτήσεων δεν περιορίζεται, η ορθή απόκριση απαιτεί ένα ευρύ φάσμα ικανοτήτων συλλογιστικής πάνω σε εικόνες.

Έπειτα από τη δημοσίευση συνόλων δεδομένων μεγάλης κλίμακας, έχουν προταθεί πολυάριθμα μοντέλα για την αντιμετώπιση του VQA προβλήματος. Η πλειοψηφία αυτών ακολουθεί μία παρόμοια οργάνωση: Η ερώτηση κωδικοποιείται μέσω ενός αναδρομικού νευρωνικού δικτύου, ενώ τα χαρακτηριστικά της εικόνας εξάγονται από ένα συνελικτικό νευρωνικό δίκτυο που έχει προεκπαιδευθεί για την αναγνώριση αντικειμένων. Στη συνέχεια, οι δύο αυτές αναπαραστάσεις συνδυάζονται με ένα μηχανισμό συγχώνευσης και οδηγούνται σε ένα υπο-δίκτυο που προβλέπει την απάντηση.

Οι περισσότερες σύγχρονες προσεγγίσεις διατυπώνουν το VQA ως ένα πρόβλημα ταξινόμησης. Παρότι οι απαντήσεις δεν είναι πλήρεις προτάσεις, κυμαίνονται από μεμονωμένες λέξεις ως σύντομες φράσεις. Το σύνολο των πιθανών απαντήσεων επιλέγεται ως οι απαντήσεις που εμφανίζονται συχνότερα στα δεδομένα εκπαίδευσης. Αυτό σημαίνει ότι απαντήσεις που αποτελούν σύντομες φράσεις αντιμετωπίζονται ως ξεχωριστές κλάσεις από τις επιμέρους λέξεις που αυτές περιλαμβάνουν. Για παράδειγμα, οι απαντήσεις "μαύρο και άσπρο", "μαύρο" και "άσπρο" θεωρούνται πλήρως ανεξάρτητες κλάσεις. Αυτό έρχεται σε αντίθεση με τη διαίσθησή ότι η έννοια μιας φράσης προκύπτει από τη σημασιολογία των επιμέρους λέξεων. Ένα αντικείμενο που είναι "μαύρο και άσπρο" μοιράζεται ένα κοινό χαρακτηριστικό με μαύρα και άσπρα αντικείμενα. Υποθέτουμε ότι η εκμάθηση της αναγνώρισης του χρώματος ενός "μαύρου και άσπρου" αντικειμένου μπορεί να ωφεληθεί από την εκμάθηση των εννοιών των δύο μεμονωμένων χρωμάτων από άλλα δείγματα του συνόλου δεδομένων.

Επιπλέον, η πρόβλεψη απαντήσεων σε επίπεδο φράσης περιορίζει την εκφραστικότητα των προτεινόμενων μοντέλων. Ο χώρος των πιθανών απαντήσεων είναι αυστηρά καθορισμένος εκ των προτέρων και δεν μπορούν να προκύψουν νέες απαντήσεις από την υπάρχουσα γνώση του μοντέλου. Για παράδειγμα, σε περίπτωση απαντήσεων σε ερωτήσεις σχετικά με το χρώμα των αντικειμένων, κάθε συνδυασμός χρωμάτων που δεν έχει προστεθεί στο σύνολο των πιθανών κατηγοριών είναι πρακτικά απαγορευμένος. Συνεπώς, η επιθυμία για υψηλή κάλυψη των απαντήσεων, οδηγείται σε γρήγορη διεύρυνση του αριθμού των πιθανών απαντήσεων.

Με κίνητρο αυτές τις παρατηρήσεις, προτείνουμε το Grounded Seq2Seq μοντέλο, που αντιμετωπίζει το VQA ως ένα πρόβλημα παραγωγής ακολουθίας (sequence generation). Η απάντηση παράγεται



Σχήμα 0.1: Αφηρημένη απεικόνισή του προτεινόμενου μοντέλου.

από ένα sequence-to-sequence μοντέλο (Ενότητα 3.3), που εξαρτάται από τις αναπαραστάσεις της ερώτησης και της εικόνας. Τα αποτελέσματα μας είναι ανταγωνιστικά με κυρίως για ερωτήσεις ανοιχτού τύπου, γεγονός που δείχνει ότι η ελεύθερη παραγωγή απαντήσεων είναι εφικτή για το ανοικτού τύπου VQA.

Περιγραφή του Μοντέλου

Το σχήμα 0.1 την αρχιτεκτονικής του προτεινόμενου μοντέλου. Το συνολικό σύστημα αποτελείται από τρεις βασικές μονάδες: τον κωδικοποιητή της ερώτησης, τον κωδικοποιητή της εικόνας και τον αποκωδικοποιητή της απάντησης.
Δεδομένου ενός ζευγαριού ερώτησης Q και εικόνας I, το σύστημα παράγει μία απάντηση Y ως εξής:

- Η ερώτηση $Q = [q_1, \dots q_N]$ κωδικοποιείται από ένα μονού επιπέδου, διπλής κατεύθυνσης LSTM (BiLSTM, Ενότητα 2.3.4). Αρχικά, οι λέξεις q_t της ερώτησης μήκους N αναπαρίστανται ως one-hot διανύσματα, όπου η θέση του "1" αντιστοιχεί στη θέση της λέξης στο λεξικό εισόδου V_q. Το one-hot διάνυσμα κάθε λέξης διέρχεται από ένα embedding επίπεδο, που έχει αρχικοποιηθεί με GloVe embeddings [57] και προσαρμόζεται κατά την εκπαίδευση. Το embedding κάθε λέξης περνά μέσα από το BiLSTM, που παράγει την νέα κρυφή κατάσταση για κάθε κατεύθυνση.

  Οι κρυφές καταστάσεις από το εμπρός $\vec{\boldsymbol{h}}_t^e$ και το πίσω πέρασμα $\overleftarrow{\boldsymbol{h}}_t^e$ ενώνονται για να αποκτήσουμε την έξοδο του BiLSTM $\boldsymbol{h}_t^e$ σε κάθε χρονική στιγμή t :

$$\boldsymbol{h}_t^e = [\vec{\boldsymbol{h}}_t^e; \overleftarrow{\boldsymbol{h}}_t^e] \quad \text{for } t \in 1 \dots N \tag{0.1}$$

  Οι καταστάσεις $\boldsymbol{h}_t^e$ για κάθε $t[\in 1, \dots N]$ διαμορφώνουν έναν πίνακα $\boldsymbol{H} \in \mathbf{R}^{N \times d}$. Η αναπαράσταση της ερώτησης $\boldsymbol{e}$ υπολογίζεται ως το κυρτό άθροισμα των $\boldsymbol{h}_t^e$ με βάρη a_t^e, που υπολογίζονται μέσω ενός self-attention μηχανισμού [47]:

$$\boldsymbol{a}^e = \text{softmax}(\boldsymbol{w}_{a2}\text{tanh}(\boldsymbol{W}_{a1}\boldsymbol{H}^T)) \tag{0.2}$$

$$\boldsymbol{e} = \sum_{t=1}^{N} a_t^e \boldsymbol{h}_t^e \tag{0.3}$$

  όπου $\boldsymbol{W}_{a1} \in \mathbf{R}^{d \times d}$ και $\boldsymbol{w}_{a2} \in \mathbf{R}^{1 \times d}$ οι παράμετροι του μηχανισμού. Τα βάρη αυτά a_t^e ορίζουν τη συμβολή κάθε κρυφής κατάστασης $\boldsymbol{h}_t^e$ στην τελική αναπαράσταση της εικόνας $\boldsymbol{e}$.

- Η εικόνα I αναπαρίσταται στην είσοδο ως ένα σύνολο από R διανύσματα χαρακτηριστικών $\{\boldsymbol{i}_1, ...\boldsymbol{i}_R\}$ που εξάγονται από ένα προεκπαιδευμενο δίκτυο ανίχνευσης αντικειμένων. Συγκεκριμένα, τα χαρακτηριστικά που χρησιμοποιούμε έχουν εξαχθεί από ένα Faster RCNN ((Ενότητα 3.2)) δίκτυο εκπαιδευμένο πάνω στο σύνολο δεδομένων Visual Genome [43], τα οποία είναι διαθέσιμα από τους Anderson et al. [5]. Τα χαρακτηριστικά αυτά κωδικοποιούν bounding-box περιοχές μίας εικόνας, προσφέροντας έτσι τοπική πληροφορία σχετικά με τα αντικείμενα που είναι παρόντα σε αυτή. Κρατάμε για όλες τις εικόνες τα 36 bounding boxes με την υψηλότερη εμπιστοσύνη. Εν συντομία, μια εικόνα I αναπαρίσταται από ένα χάρτη $R = 36$ διανυσμάτων χαρακτηριστικών $\boldsymbol{i}_r \in \mathbf{R}^{2048}$ που αντιστοιχούν εξέχοντες περιοχές της εικόνας.

  Εφαρμόζουμε $L2$ κανονικοποίηση στα διανύσματα εισόδου I έτσι ώστε όλες οι τιμές τους να βρίσκονται σε ένα συγκρίσιμο εύρος και περνάμε τα κανονικοποιημένα χαρακτηριστικά από ένα επίπεδο με tanh μη γραμμικότητα, αποκτώντας την οπτική αναπαράσταση $V \in \mathbf{R}^{R\times d}$:

$$\boldsymbol{V} = \tanh(\frac{\boldsymbol{I}}{\|\boldsymbol{I}\|}\boldsymbol{W}_v) \tag{0.4}$$

  όπου $\boldsymbol{W}_v \in \mathbf{R}^{2048\times d}$ είναι ο πίνακας που προβάλει τα διανύσματα χαρακτηριστικών στη διάσταση την αναπαράστασης της ερώτησης.

- Η απάντηση $Y = [y_1, ...y_L]$ παράγεται από ένα μονού επιπέδου, μονής κατεύθυνσης LSTM με μέγεθος ίσο με d. Το LSTM αποκωδικοποίησης εξαρτάται από την ερώτηση μέσω της αρχικοποίησης της κρυφής κατάστασης με την αναπαράσταση της ερώτησης $\boldsymbol{e}$:

$$\boldsymbol{h}_0^d = \boldsymbol{e} \tag{0.5}$$

  Σε κάθε χρονική στιγμή $t \in [1, ...L]$, ο αποκωδικοποιητής δέχεται ως είσοδο την προηγούμενη λέξη y_{t-1} και ενημερώνει τις καταστάσεις του. Η τρέχουσα κρυφή κατάσταση $\boldsymbol{h}_t^d$ χρησιμοποιείται για να εστιάσει πάω στα χαρακτηριστικά της εικόνας $\boldsymbol{V}$. Η διαδικασία αυτή παράγει ένα οπτικό διάνυσμα $\bar{\boldsymbol{v}}_t$ όπως περιγράφεται από την Εξίσωση 0.8.

  Συνδυάζουμε το οπτικό διάνυσμα $\bar{\boldsymbol{v}}_t$ και την κρυφή κατάσταση $\boldsymbol{h}_t^d$ με το γινόμενο Hadamard $\circ$ με στόχο την απόκτηση ενός διανύσματος $\boldsymbol{f}_t$:

$$\boldsymbol{f}_t = \bar{\boldsymbol{v}}_t \circ \boldsymbol{h}_t^d \tag{0.6}$$

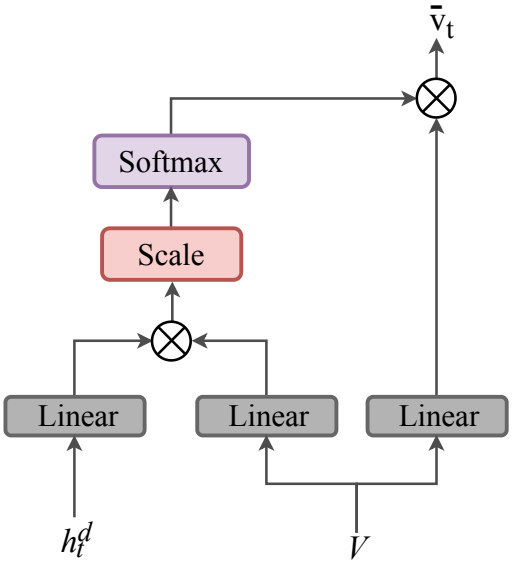
  Η επόμενη λέξη y_t υπολογίζεται ως η λέξη με την υψηλότερη πιθανότητα από την δεσμευμένη κατανομή πιθανότητας πάνω στο λεξικό των απαντήσεων V_a:

$$P(y_t|\boldsymbol{e}, y_1, ...y_{t-1}, \boldsymbol{f}_t) = \text{softmax}(\boldsymbol{W}_d\boldsymbol{f}_t) \tag{0.7}$$

  όπου $\boldsymbol{W}_d \in \mathbf{R}^{|V_a|\times d}$. Η αποκωδικοποίηση σταματά όταν προβλεφθεί η λέξη <*EOS*>. Στην πράξη, συγχωνεύουμε τα λεξικά των ερωτήσεων V_q και των απαντήσεων V_a σε ένα κοινό λεξικό V και χρησιμοποιούμε το ίδιο επίπεδο embedding για τις λέξεις των ερωτήσεων και των απαντήσεων έτσι ώστε το δίκτυο να μάθει μια κοινή αναπαράσταση για τις λέξεις V.

Μηχανισμός Χωρικής Προσοχής Ο μηχανισμός χωρικής προσοχής (attention) χρησιμοποιείται για την εστίαση πάνω στα χαρακτηριστικά των περιοχών της εικόνας κατά την αποκωδικοποίηση της απάντησης. Το σχήμα 0.2 αποτυπώνει γραφικά τον εν λόγω μηχανισμό χωρικού attention.

Σε κάθε χρονική στιγμή t, η εικόνα συνοψίζεται σε ένα d-διάστατο διάνυσμα $\bar{\boldsymbol{v}}_t$ χρησιμοποιώντας το μηχανισμό προσοχής, που επιτρέπει στο δίκτυο να εστιάσει στις πιο σχετικές υποπεριοχές της εικόνας. Ο μηχανισμός προσοχής που υιοθετούμε ακολουθεί τον ορισμό της κλιμακωτής προσοχής εσωτερικού γινομένου [68]. Η σημασία κάθε διανύσματος χαρακτηριστικών καθορίζεται από τη σχέση τους με ένα διάνυσμα-ερώτηση (*query vector*), που στην περίπτωση μας οδηγείται από την κρυφή κατάσταση του αποκωδικοποιητή $\boldsymbol{h}_t^d \in \mathbf{R}^d$.



Σχήμα 0.2: Γραφική απεικόνιση του μηχανισμού προσοχής.

Προκειμένου να υπολογίσουμε την τελική αναπαράσταση της εικόνας $\bar{\boldsymbol{v}}_t$, παίρνουμε πρώτα δύο γραμμικούς μετασχηματισμούς $\boldsymbol{V}_1$ και $\boldsymbol{V}_2$ της οπτικής αναπαράστασης $\boldsymbol{V}$. Ο πίνακας $\boldsymbol{V}_1$ χρησιμοποιείται για τον υπολογισμό της συγγένειας μεταξύ των διανυσμάτων χαρακτηριστικών κάθε περιοχή και του query διανύσματος $\boldsymbol{h}_t^d$. Τα σκορ της συγγένειας μετατρέπονται σε βάρη προσοχής $\boldsymbol{a}_t^d$ περνώντας μέσα από ένα softmax επίπεδο. Τέλος, τα βάρη προσοχής $\boldsymbol{a}_t^d$ εφαρμόζονται στα χαρακτηριστικά της εικόνας V_2 παράγοντας την αναπαράσταση $\bar{\boldsymbol{v}}_t$ ως εξής:

$$\boldsymbol{a}_t^d = \text{softmax}(\frac{\boldsymbol{h}_t^d \boldsymbol{V}_1^T}{\sqrt{d}}) \tag{0.8}$$

$$\bar{\boldsymbol{v}} = \boldsymbol{a}_t^d \boldsymbol{V}_2 \tag{0.9}$$

Ο παράγοντας $1/\sqrt{d}$ χρησιμοποιείται για να κλιμακώσει το αποτέλεσμα του εσωτερικού γινομένου και να αποτρέψει την είσοδο πολύ μεγάλων τιμών στης συνάρτησης, που μπορεί να οδηγήσει στο πρόβλημα vanishing gradient. Εστιάζοντας εκ νέου στην εικόνα σε κάθε βήμα, ο αποκωδικοποιητής έχει τη δυνατότητα να προσαρμόσει τα βάρη προσοχής λαμβάνοντας υπόψιν την προηγούμενη λέξη και να αποφύγει τις μακροπρόθεσμες εξαρτήσεις από το οπτικό περιεχόμενο.

Πειραματική Αξιολόγηση

Σύνολο Δεδομένων Αξιολογούμε το μοντέλο μας χρησιμοποιώντας τη δεύτερη έκδοση του συνόλου δεδομένων *Visual Question Answering under Changing Priors* (VQA-CP v2) [3]. Το εν λόγω σύνολο δεδομένων αποτελεί παραλλαγή του VQA v2 συνόλου δεδομένων [27] το οποίο είναι το πιο ευρέως διαδεδομένο σύνολο δεδομένων για VQA. Κάθε δείγμα αποτελεί ένα ζευγάρι εικόνας-ερώτησης συνοδευόμενο από 10 απαντήσεις, που έχουν συλλεχθεί από ξεχωριστούς ανθρώπους.

Πρόσφατες δημοσιεύσεις [27, 2, 14, 34] έχουν δείξει ότι η ύπαρξη γλωσσικής μεροληψίας (language biases) στα σύνολα δεδομένων επιτρέπει στα μοντέλα να μαντεύουν τη σωστή απάντηση αγνοώντας την οπτική πληροφορία. Ως αποτέλεσμα, ένα μοντέλο που επιτυγχάνει καλή απόδοση, δεν αντιμετωπίζει απαραιτήτως το the VQA πρόβλημα, γεγονός που μπορεί να αποτελέσει τροχοπέδη για την πραγματική πρόοδο. Το VQA v2 σύνολο δεδομένων επιχειρεί να καταστείλει το ρόλο των εκ των προτέρων γλωσσικών κατανομών συλλέγοντας για κάθε ερώτηση 2 συμπληρωματικές εικόνες που οδηγούν σε διαφορετικές απαντήσεις. Παρά την προσπάθεια αυτή, μεροληψία (bias) στην κατανομή των ερωτήσεων ή των απαντήσεων εξακολουθούν να υπάρχουν, όπως φαίνεται στον πίνακα 0.1. Η παραλλαγή VQA-CP v2 προτάθηκε για την αντιμετώπιση αυτού του ζητήματος. Δημιουργήθηκε ανα-

Μέθοδος	Σύνολο Δεδομένων	Ακρίβεια			
		Σύνολο	Ναι/Όχι	Αριθμός	Άλλο
μόνο γλώσσα [7]	VQA v2	43.01	67.95	30.97	27.20
	VQA-CP v2	15.95	35.09	11.63	07.11
γλώσσα+εικόνα [7]	VQA v2	51.61	73.06	34.41	39.85
	VQA-CP v2	19.73	34.25	11.39	14.41
γλώσσα+εικόνα+attention [73]	VQA v2	52.02	68.89	34.55	43.80
	VQA-CP v2	24.96	38.35	11.14	21.74

Πίνακας 0.1: Σύγκριση της απόδοσης VQA μοντέλων πάνω στο VQA-CP v2 σύνολο αξιολόγησης και στο VQA v2 σύνολο επαλήθευσης [3].

διοργανώνοντας τις υποδιαίρεσης του VQA v2 συνόλου, έτσι ώστε οι κατανομή των απαντήσεων κάθε τύπου ερώτησης να διαφέρει μεταξύ του υποσυνόλου εκπαίδευσης και αξιολόγησης. Η αναντιστοιχία των κατανομών των απαντήσεων εκπαίδευσης και αξιολόγησης εξετάζει πραγματικά πόσο καλά ένα μοντέλο μπορεί να επεξεργαστεί την οπτική είσοδο για να συμπεράνει τη σωστή απάντηση. Ο πίνακας 0.1 αποτυπώνει τη σημαντική μείωση της απόδοσης των μοντέλων όταν αξιολογούνται στο VQA-CP v2.

Παρακάτω απαριθμούμε ορισμένες βασικές λεπτομέρειες του συνόλου δεδομένων VQA-CP v2:

- Το σύνολο *εκπαίδευσης* αποτελείται από 121K εικόνες, 438K ερωτήσεις και 4.4M απαντήσεις, ενώ το σύνολο *αξιολόγησης* αποτελείται 98K εικόνες, 220K ερωτήσεις και 2.2M απαντήσεις.

- Οι ερωτήσεις εκπαίδευσης έχουν ένα λεξιλόγιο μεγέθους 14.5K, ενώ οι απαντήσεις εκπαίδευσης έχουν ένα λεξιλόγιο μεγέθους 37K. Το κοινό λεξιλόγιο ανέρχεται σε 40K λέξεις.

- Το μέγιστο μήκος των ερωτήσεων είναι 25 λέξεις. Παρότι το μέγιστο μήκος των απαντήσεων φτάνει τις 24 λέξεις, η πλειοψηφία των απαντήσεων αποτελείται από μόλις μία έως τρεις λέξεις.

Σχετική Βιβλιογραφία Έπειτα από την κυκλοφορία μεγάλης κλίμακας VQA συνόλων δεδομένων [7, 27], πολυάριθμες προσεγγίσεις έχουν προταθεί για την αντιμετώπιση του προβλήματος. Η εισαγωγή μηχανισμών attention στα VQA συστήματα έχει επιφέρει σημαντική ώθηση της απόδοσης τους. Μια αξιοσημείωτη εργασία πάνω στο attention αποτελεί το μοντέλο Stacked Attention Network (SAN) [73], που αξιοποιεί δύο συνεχόμενα επίπεδα χωρικού attention με στόχο την εξαγωγή πιο λεπτομερούς πληροφορίας. Το μοντέλο Grounded VQA (GVQA) [3] είναι μια υβριδική αρχιτεκτονική που επεκτείνει το SAN δίκτυο ως προς την πρόβλεψη των απαντήσεων. Οι "Ναι/Όχι" ερωτήσεις αντιμετωπίζονται ως ένα δυαδικό πρόβλημα οπτικής επαλήθευσης, ενώ οι υπόλοιπες ερωτήσεις συνδυάζουν την πρόβλεψη οπτικών εννοιών και υποψήφιων απαντήσεων. Anderson et al. [5] εφαρμόζουν μια εκδοχή χωρικού attention πάνω σε ένα σύνολο τοπικών χαρακτηριστικών από ένα Faster RCNN δίκτυο. Πρόσφατα οι Ramakrishnan et al. [59] εισήγαγαν ένα σχήμα adversarial εκπαίδευσης για την εκπαίδευση μοντέλων με την ίδια αρχιτεκτονική όπως τα [73] και [5], τα οποία όμως βασίζονται λιγότερο σε γλωσσικά biases. Ειδικότερα, εκπαιδεύουν ως αντίπαλο ένα μοντέλο χρησιμοποιώντας μόνο την γλωσσική είσοδο και προσθέτοντας έναν όρο εντροπίας στη συνάρτηση κόστους που υπολογίζει το κέρδος πληροφορίας από τη συμπερίληψη της εικόνας.

Η συγχώνευση των χαρακτηριστικών της εικόνας και της ερώτησης πραγματοποιείται συχνά με απλές πράξεις όπως το γινόμενο Hadamard και η ένωση (concatenation) [7]. Παρόλα αυτά, έχουν διερευνηθεί και πιο πολύπλοκες προσεγγίσεις συγχώνευσης. Αυτές οι προσεγγίσεις επικεντρώνονται στην αποδοτική προσέγγιση του εξωτερικού γινομένου μεταξύ των χαρακτηριστικών της εικόνας και της ερώτησης χρησιμοποιώντας τις τεχνικές compact bilinear pooling [23], προσέγγισης με tensors χαμηλής τάξης [39] or Tucker αποσύνθεσης [10]. Οι Andreas et al. [6] προτείνουν τη δυναμική σύνθεση ενός δικτύου με μέρη από ένα σύνολο προεκπαιδευμένων υπο-δικτύων με στόχο την προσαρμογή του attention και της συγχώνευσης ανάλογα με την ερώτηση.

Υπερπαράμετροι του Μοντέλου Ο πίνακας 0.2 συνοψίζει τις βασικές υπερπαραμέτρους του Grounded Seq2Seq μοντέλου. Χρησιμοποιούμε κοινό λεξιλόγιο για τις ερωτήσεις και τις απαντήσεις εφαρμόζοντας την τεχνική δέσμευσης βαρών (weight tying) στα embedding επίπεδα του κωδικοποιητή και του αποκωδικοποιητή. Με αυτό τον τρόπο μειώνουμε τον αριθμό των παραμέτρων και επιβάλουμε τη σημασιολογική ευθυγράμμιση της ερώτησης και της απάντησης. Χρησιμοποιούμε για μέγεθος batch ίσο με 128. Θέτουμε το μέγιστο μήκος των ερωτήσεων ίσο με 23 και το μέγιστο μήκος των απαντήσεων ίσο με 5.

Αρχικοποίηση Embeddings		300-d GloVe
Μέγεθος Λεξιλογίου		10K words
Κωδικοποιητής Ερώτησης	Μέγεθος LSTM	512
	Μέγιστο μήκος	23
Αποκωδικοποιητής Απάντησης	Μέγεθος LSTM	1024
	Μέγιστο μήκος	5

Πίνακας 0.2: Υπερπαράμετροι του μοντέλου Grounded Seq2Seq.

Ρυθμίσεις Εκπαίδευσης Κατά την εκπαίδευση, διατηρούμε το ένα τρίτο των δεδομένων εκπαίδευσης για επαλήθευση και παρακολουθούμε την ακρίβεια στο σύνολο αυτό προκειμένου να εφαρμόσουμε πρόωρη διακοπή ύστερα από 5 εποχές σταθερής ή μειούμενης ακρίβειας. Ακόμη εφαρμόζουμε dropout με πιθανότητα 0.2 μετά από κάθε επίπεδο και teacher forcing, δίνοντας ως είσοδο στον αποκωδικοποιητή τη σωστή προηγούμενη λέξη, Χρησιμοποιούμε τον αλγόριθμο βελτιστοποίησης Adam με ρυθμό μάθησης 0.001. Τέλος, πραγματοποιούμε επαύξηση των δεδομένων μας για την κατηγορία απαντήσεων "Άλλο": Σε κάθε εποχή, διαλέγουμε τυχαία ως ετικέτα μία από τις 10 διαθέσιμες απαντήσεις. Με αυτόν τον τρόπο, παρουσιάζουμε στο μοντέλο μας παραφρασμένες απαντήσεις για κάθε απάντηση, το οποίο λειτουργεί ως μέθοδος regularization.

Μετρική Αξιολόγησης Αξιολογούμε το μοντέλο μας χρησιμοποιώντας την καθιερωμένη VQA μετρική ακρίβειας [7]. Όπως έχει προαναφερθεί, κάθε ερώτηση συνοδεύεται από 10 απαντήσεις. Μια απάντηση a λαμβάνει ένα τέλειο σκορ $s(a)$ αν ταιριάζει με τουλάχιστον τρεις από τις διαθέσιμες απαντήσεις:

$$score(a) = \min(\frac{\#\text{ άνθρωποι που απάντησαν a}}{3}, 1) \tag{0.10}$$

Πειραματικά Αποτελέσματα Ο πίνακας 0.3 παρουσιάζει την απόδοση του προτεινόμενου μοντέλου στο VQA-CP v2 σύνολο σε σύγκριση με baseline και state-of-the-art μοντέλα. Ως baseline μοντέλα θεωρούμε τα δύο πρώτα, ονομαστικά τα "d-LSTM" και "d-LSTM Q + I". Το "d-LSTM" αναφέρεται σε ένα μοντέλο που χρησιμοποιεί μόνο τη γλωσσική πληροφορία εισόδου και αγνοώντας την εικόνα, όπου η ερώτηση κωδικοποιείται από ένα βαθύ LSTM δύο στρωμάτων. Το "d-LSTM Q + I" χρησιμοποιεί την ίδια διάταξη για την ερώτηση και ένα προεκπαιδευμένο CNN μοντέλο για την εξαγωγή ενός καθολικού διανύσματος χαρακτηριστικών για την εικόνα. Τα διανύσματα χαρακτηριστικών της ερώτησης και της εικόνας ενώνονται και οδηγούνται σε ένα ταξινομητή. Είναι εμφανές ότι το μοντέλο μας αποδίδει σημαντικά καλύτερα από αυτά τα αφελή baseline μοντέλα. Το καθολικό διάνυσμα χαρακτηριστικών μπορεί να είναι επαρκές για την κατηγοριοποίηση της εικόνας αλλά αποτελεί μία υπερβολικά γενική αναπαράσταση της απαραίτητης πληροφορίας για την απάντηση μιας συγκεκριμένης ερώτησης.

Σύγκριση με State-of-the-Art Μοντέλα Κατά τη σύγκριση του προτεινόμενου Grounded Seq2Seq με state-of-the-Art (sota) μοντέλα, παρατηρούμε ότι το μοντέλο μας επιφέρει ανταγωνιστικά αποτελέσματα, ιδίως στην κατηγορία "Άλλο". Παρόλο που η κατηγορία αυτή είναι ιδιαίτερα ευρεία, το αποτέλεσμα βρίσκεται σε συμφωνία με το κίνητρο της εργασίας. Υποθέτουμε ότι τα παραδείγματα αυτής

Μοντέλο	Σύνολο	Ναι/Όχι	Αριθμός	Άλλο
d-LSTM Q [7]	15.95	35.09	11.63	7.11
d-LSTM Q + I [48]	19.73	34.25	11.39	14.41
SAN [73]	24.96	38.35	11.14	21.74
GVQA [3]	31.30	57.99	13.68	22.14
MCB [23]	36.33	41.01	11.96	40.57
UpDn [5]	39.74	42.27	11.93	**46.05**
UpDn+Adv [59]	**41.17**	**65.49**	**15.48**	35.48
Grounded Seq2Seq	**36.42**	**40.29**	**12.07**	**41.08**

Πίνακας 0.3: Σύγκριση με baseline and state-of-the-art μοντέλα πάνω στο VQA-CP v2 σύνολο δεδομένων.

της κατηγορίας επωφελούνται σε μεγαλύτερο βαθμό από την παραγωγή ακολουθιών ως απαντήσεις, επειδή επιδέχονται απαντήσεις μεγαλύτερου μήκους. Υπογραμμίζεται ότι κανένα μοντέλο δεν επιτυγχάνει τη βέλτιστη απόδοση σε όλες τις υποκατηγορίες. Παρατηρούμε ότι τα μοντέλα *GVQA* και *UpDn+Adv* έχουν καλύτερη επίδοση σε απαντήσεις "Ναι/Όχι" με μεγάλη διαφορά. Παρόλα αυτά, η επίδοση τους στην κατηγορία "Άλλο" είναι μέτρια. Σε σύγκριση με τα baselines, sota μοντέλα έχουν τη λιγότερη βελτίωση σε ερωτήσεις που απαντώνται με αριθμούς. Το μέτρημα απαιτεί ένα ξεχωριστό είδος συλλογιστικής, που έχει κινητοποιήσει μια σειρά εργασιών που εστιάζουν αποκλειστικά σε αυτό το πρόβλημα [67, 75]. Ένας περαιτέρω παράγοντας που έχει ιδιαίτερη επίδραση στην συνολική απόδοση είναι η ποιότητα των οπτικών χαρακτηριστικών. Ειδικότερα, μοντέλα που αξιοποιούν χαρακτηριστικά από το Faster RCNN (*UpDn*, *UpDn+Adv*, *Grounded Seq2Seq*) ξεπερνούν συστηματικά μοντέλα που κάνουν χρήση χαρακτηριστικών από συνελικτικά δίκτυα εκπαιδευμένα για κατηγοριοποίηση εικόνων (*SAN*, *GVQA* και *MCB*).

Μελέτη του Μοντέλου Εκτελούμε μια μελέτη με στόχο να απομονώσουμε τα στοιχεία του μοντέλου μας και να αξιολογήσουμε την επίδρασή τους στην απόδοση του προτεινόμενου μοντέλου. Αυτή η μελέτη περιλαμβάνει πειράματα όπου μια ή και οι δύο είσοδοι αντικαθίστανται από τυχαία διανύσματα για να εκτιμηθεί η συνεισφορά της γλωσσικής και της οπτικής εισόδου, καθώς και η τυχαία απόδοση για κάθε τύπο ερωτήματος. Ο πίνακας 0.4 παρουσιάζει τα αποτελέσματα των πειραμάτων.

Μοντέλο	Σύνολο	Ναι/Όχι	Αριθμός	Άλλο
(A): Τυχαίες Είσοδοι	16.84	56.34	0.38	0.65
(B): Μόνο Εικόνα	17.44	57.90	0.39	0.91
(C): Μόνο Ερώτηση	18.66	38.29	10.25	10.38
(D): Ερώτηση και Εικόνα ως Αρχικοποίηση	35.51	38.57	11.10	40.30
(E): UpDn Μηχανισμός Attention	33.20	39.56	11.54	35.45
(F): Αφαίρεση του Μηχανισμού self-attention	35.92	40.83	11.93	40.03
(G): Συγχώνευση Ένωσης	35.42	39.47	12.30	39.35
(H): Συγχώνευση MUTAN	35.67	40.00	11.46	39.70
Grounded Seq2Seq	**36.42**	**40.29**	**12.07**	**41.08**

Πίνακας 0.4: Αποτελέσματα από τη μελέτη του μοντέλου.

Τα πειράματα με τυχαία ή περιορισμένη είσοδο (*A, B, C*) προσφέρουν κατά βάση πληροφορία για τη φύση του προβλήματος. Κατ' αρχάς, παρατηρούμε ότι η χρήση μόνο της εικόνας έχει παρόμοια επίδοση με τις τυχαίες εισόδους, ενώ η χρήση μόνο της ερώτησης έχει καλύτερη αν και χαμηλή επίδοση. Αυτό επιβεβαιώνει παρατηρήσεις στη βιβλιογραφία σχετικά με το ρόλο του language bias στα VQA συστήματα [3]. Η απόδοση των μοντέλων (*A, B*) στην κατηγορία "Ναι/Όχι" οφείλεται στο γεγονός ότι τα μοντέλα απαντούν σχεδόν πάντα *'όχι'*, που αποτελεί την πιο συχνή απάντηση στα δεδομένα εκπαίδευσης. Είναι φανερό ότι για την επίδοση στην κατηγορία"Άλλο" απαιτείται η ενσωμάτωση και

των δύο εισόδων. Στην κατηγορία "Αριθμός", παρατηρούμε μικρή διαφορά ανάμεσα στο μοντέλο (*C*) και αυτά που αξιοποιούν την πληροφορία τόσο από την ερώτηση όσο και από την απάντηση.

Τα μοντέλα (*C*) και (*F*), διερευνούν τη διάταξη του κωδικοποιητή της ερώτησης. Σε σύγκριση με το *d-LSTM Q* baseline (βλ. Πίνακα 0.3), η έκδοση του μοντέλου μας που χρησιμοποιεί μόνο γλώσσα επιτυγχάνει $+3\%$ απόλυτη βελτίωση. Αυτό υποδηλώνει ότι η χρήση ενός διπλής κατεύθυνσης LSTM με self-attention παρέχει μια καλύτερη αναπαράσταση της ερώτησης. Η σημασία του μηχανισμού self-attention επιβεβαιώνεται και από τη μειωμένη επίδοση του μοντέλου (*F*). Συγκρίνοντας τα πειράματα (*D*) και (*E*) με το Grounded Seq2Seq, παρατηρούμε ότι η χρήση του επιλεγμένου μηχανισμού attention σε κάθε βήμα αποκωδικοποίησης έχει θετική επίδραση στην απόδοση του μοντέλου.

Τέλος, ο πειραματισμός με διαφορετικές μεθόδους συγχώνευσης δείχνει ότι το γινόμενο Hadamard αποδίδει καλύτερα από την ένωση (*G*) και τη συγχώνευση MUTAN (*H*) κατά 1.00% και 0.75% αντίστοιχα. Η ελαφρώς χειρότερη απόδοση του μοντέλου (*G*) μπορεί να αποδοθεί στο γεγονός ότι η πράξη της ένωσης αυξάνει το μέγεθος του διανύσματος χαρακτηριστικών χωρίς να συλλαμβάνει τις αλληλεπιδράσεις που συμβαίνουν μεταξύ των εισόδων. Η συγχώνευση MUTAN, από την άλλη, είναι μια μέθοδος παραγοντοποίησης του εξωτερικού γινομένου, που αυξάνει σημαντικά την πολυπλοκότητα του μοντέλου. Δεδομένου ότι χρησιμοποιήσαμε τις τιμές των παραμέτρων από την πρωτότυπη δημοσίευση [10], η απόδοση του μοντέλου (*H*) θα επωφελούνταν πιθανώς από περαιτέρω πειραματισμό με τις τιμές των υπερπαραμέτρων.

Συμπεράσματα

Σε αυτή τη διπλωματική εργασία, εξερευνούμε την αποτελεσματικότητα ενός sequence-to-sequence μοντέλου για το VQA πρόβλημα. Η προτεινόμενη μέθοδος βασίζεται στη χρήση αναδρομικών νευρωνικών δικτύων που εξαρτώνται τόσο από τη γλωσσική όσο και από την οπτική είσοδο. Ο αποκωδικοποιητής εφαρμόζει attention πάνω στα χαρακτηριστικά της εικόνας σε κάθε βήμα αποκωδικοποίσης. Η πειραματική μας μελέτη εξετάζει τη συμβολή των επιμέρους τμημάτων του μοντέλου. Εν κατακλείδι, επιτυγχάνουμε ανταγωνιστικά αποτελέσματα δείχνοντας τη δυνατότητα για πρόβλεψη απαντήσεων ελεύθερης μορφής.

Λέξεις κλειδιά

απάντηση οπτικών ερωτήσεων, μοντέλα ακολουθίας-σε-ακολουθία, γειωμένη συλλογιστική, βαθιά μάθηση, πολυτροπική μάθηση

Abstract

Visual Question Answering (VQA) constitutes a task at the intersection of Natural Language Processing and Computer Vision. Given an open-ended, natural-language question and an image, the goal is to predict the correct answer. In recent years, VQA has attracted a lot of interest from the research community, as it transcends the preliminary of extracting meaningful representations from each modality. In addition to that, it requires the capability to reason over the inputs and to infer their relations. Answering arbitrary visual questions is a challenging problem, as it assesses a large range of skills extending across the linguistic and the visual domains.

One limitation of typical, modern VQA systems is that they approach the task as a classification problem over a limited set of pre-defined answers. The goal of this Diploma Thesis is to alleviate the above problem, via proposing a sequence generation model that can produce answers of arbitrary length. The proposed method is a sequence-to-sequence network conditioned on textual and visual information from the question and the image respectively. The question encoder is a bidirectional Recurrent Neural Network augmented with a self-attention mechanism, while the image feature extractor is comprised of a pretrained Convolutional Neural Network. The answer is generated following a greedy decoding process, which uses a Recurrent Neural Network cell as the decoder. At each decoding step, the answer decoder attends to the visual features by employing a cross-modal, scaled dot-product attention mechanism. We conduct an ablation study that investigates the contribution of each module. Our results indicate that the feedback loop, which allows access to image features at each decoding step, is an effective conditioning mechanism.

The proposed model is evaluated on the VQA-CP v2 dataset, which tests the capacity to reason over the visual input without depending on language-based statistical biases. Our model shows significant improvement of prediction accuracy compared to baseline approaches. We also measure the performance of the proposed model against state-of-the-art VQA systems. The reported performance is comparable to that of existing models in the literature, showcasing the feasibility of free-form answer generation for open-ended VQA.

Key words

visual question answering, sequence-to-sequence, grounded reasoning, deep learning, multimodal learning

Acknowledgements

Throughout the writing of this thesis, I have received a great deal of support and assistance. First and foremost, I would first like to thank my supervisor, Prof. Alexandros Potamianos, for offering me the opportunity to work on a subject of my interest and for his guidance along the way. It has been a valuable learning experience. I am also grateful for him introducing me through his lectures to the fields of Machine Learning and Natural Language Processing, in a way that captured my interest and acted as a catalyst for my following academic choices.

Next, I would like to express my gratefulness for Giorgos Paraskevopoulos for his time and energy, but most importantly for his direct input. I would like to acknowledge my colleagues for the insightful discussions and companionship along the years, and especially Eleftheria for the experiences we shared during our studies.

Finally, I would like to thank my family and my friends for supporting me. The encouragement they have provided is greatly appreciated. Special thanks to my sister, who has always been a bright example of thoughtfulness and determination.

Maria-Vasiliki Nikandrou,

Athens, July 5, 2019

Contents

List of Tables

List of Figures

Chapter 1

Introduction

1.1 Visually Grounded Language

The development of machines that understand the meaning of natural language and that are able to interpret the content of visual scenes, constitute two central objectives of artificial intelligence. In recent years, research in the respective fields of natural language processing (NLP) and computer vision (CV) has made remarkable progress. Deep neural network techniques have pushed the boundaries of performance in tasks across the board. Although the areas of NLP and CV are usually studied in isolation, there is increasing interest in approaches attempting to capture how they interact. These approaches fall under the broader category of *multimodal learning*, which focuses on processing and combining information from the diverse modes in which something exists or is experienced.

Efforts to bridge natural language and visual understanding have been motivated by the aspiration to enable human-machine interactions about the physical world. As humans communicate with each other primarily through language, it is expected that autonomous, intelligent machines would also be capable to reason over visual information using language. Most modern language models are based on the distributional hypothesis of semantics [30], which states that words that occur in similar linguistic contexts are likely to have related meanings. Semantic representations of words learned using only textual corpora have so far been successfully applied to core NLP tasks. However, the limitations of this overarching approach are called into question by the following argument: Models are trying to extract the meaning of words, which are simply symbols, from their relation to words in their context, which are also symbols. As a result, they are lacking the ability to connect word representations to the real-world objects they refer to [51]. The challenge of grounding the meaning of words in perceptual experience or action is known as the *symbol grounding problem* [29].

The importance of grounding semantic representations to physical objects is supported by cognitive evidence. Humans acquire a common understanding about the meaning of words that is tied to their interactions with the environment. Studies in child language acquisition suggest that there is a strong correlation between learning new words and learning the corresponding concepts in the real world [26]. Moreover, experimental findings indicate that in early developmental stages children generalize their knowledge about object names based on the similarity of visual attributes [45]. This line of research has inspired several NLP approaches. Recent work has shown that grounding semantic representations can boost the performance of models on NLP tasks, which are traditionally tackled by purely linguistic approaches such as word similarity [63], natural language inference [41] and sentiment classification [38].

Advances in deep learning have given rise to tasks that lie at the intersection of language and vision and particularly those that aim at leveraging natural language to express visual understanding capabilities. Such tasks include image captioning [32], visual question answering [7] and visual dialog [17]. In the scope of this thesis, we focus on the task of visual question answering and propose a model for grounded answer generation.

1.2 Visual Question Answering

Visual question answering (VQA) is the task of automatically answering free-form, natural-language questions by reasoning over a given image. During the past few years, VQA has attracted considerable attention in the research community. The success of image classification and object detection systems has shifted the interest towards more demanding visual understanding tasks. VQA is particularly challenging because it requires a system to demonstrate varied computer vision capabilities, such as object recognition, attribute classification, counting and more. Current VQA datasets pose no limitations to the visual questions asked, meaning that they can demand external knowledge and common-sense reasoning. In brief, VQA is a multimodal task, that involves the understanding of natural language, the processing of visual inputs and the ability to reason over both modalities as well as their interconnections.

The nature of the task allows VQA to serve as a benchmark for visual understanding. Answering arbitrary questions involves diverse low-level computer vision tasks. As a result, in a VQA setting we can evaluate how well machines are able to process and reason over visual inputs. It is thus understandable that open-domain VQA on real-world images is considered as a Visual Turing Test. Beyond theoretical motivations, VQA can be applied in real-world scenarios. Machines with visual reasoning capabilities will improve human-machine communication by creating common reference points in the physical world. An immediate application of such VQA systems would be to assist visually impaired individuals by enabling them to acquire specific information on the fly. For example, VQA systems can help blind people either online by providing requested details about visual content or in real life by helping them navigate dynamic environments.

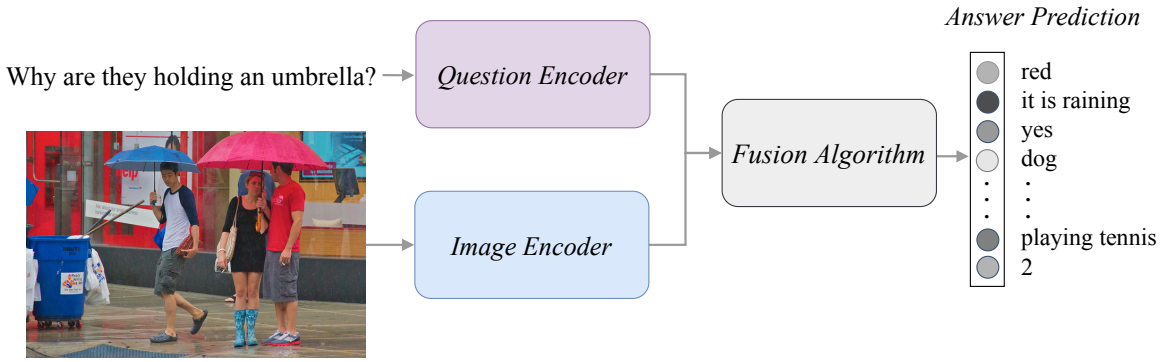


Figure 1.1: Abstract diagram of the VQA pipeline.

Since the release of large-scale datasets [7, 27, 43], a multitude of approaches has been proposed. Most VQA algorithms define a common series of steps, as it is shown in Figure 1.1: First, image and question representations are extracted by separate modules. The input representations are then combined using either simple operations, attention mechanisms or more complex fusion methods to predict an answer. Finally, the fused representation is passed to a classifier that predicts an answer. More details about specific models are given in our summary of related work (Sec. 5.2).

Addressing VQA as a classification task poses a limitation to current systems. The prediction of answers on a phrase-level contradicts our intuition suggesting the meaning of a phrase emerges from the syntax and the semantics of the words that comprise it. Consequently, this approach restricts the expressiveness of proposed models to closed-world settings, where the space of possible answers is known in advance. The goal of this thesis is to explore the feasibility of generating answers, instead of performing classification among a set of known answers. To this end, we adapt the sequence-to-sequence framework [65] to VQA by conditioning the answer generation to both input modalities. We use Faster RCNN features [60] to encode the image from salient regions and recurrent layers to encode the question input as well as to decode the answer. We address the issue of grounding the answer generation process to the visual input by creating a feedback loop during decoding. At each

decoding step, the next word is predicted by attending to the image features and fusing with the current decoder state.

The results of the experiments, which were conducted as part of this thesis, indicate that our conditional answer decoder proposes a viable alternative to the closed-world assumption of current VQA systems. Through an ablation study we provide empirical evidence supporting the importance of recurrent grounding for model performance. Furthermore, our results demonstrate significant improvement compared to baseline approaches and are competitive with the state-of-the-art for open-ended visual questions.

1.3 Thesis Outline

The thesis is structured as follows:

- Chapter 2 provides the machine learning background. Specifically, we first introduce the basic concepts of supervised machine learning and then focus on the deep learning background, that is most relevant to our work.
- Chapter 3 summarizes the modality-specific approaches that we utilize. We describe the general application of recurrent neural network for sentence encoding, the convolutional neural network model that is used to extract image features and the sequence-to-sequence framework.
- Chapter 4 presents the main motivation behind our approach and the system architecture of the proposed Grounded Seq2Seq model.
- Chapter 5 displays the setting and the results for the different experiments that were conducted. We also provide some examples of the results of the proposed model.
- Chapter 6 concludes the thesis and proposes directions for future work.

Chapter 2

Machine Learning Background

2.1 Machine Learning

Machine learning (ML) is the scientific study of algorithms that allow computers to automatically extract knowledge through data. Computers can thus learn to perform tasks by generalizing from the experience acquired from data rather than being explicitly programmed. ML is considered a subfield of artificial intelligence (AI) that has strong ties to computational statistics and mathematical optimization. The term *Machine Learning* was first introduced in 1959 by Arthur Samuel, a pioneer in the fields of computer gaming and AI.

ML applications span a wide range of tasks associated with human intelligence. One active area of research, that makes use of ML methods, is natural language processing (NLP). NLP is concerned with developing computational algorithms to automatically analyze, understand and generate human language. ML algorithms have also been a subject of computer vision (CV) research. CV deals with building algorithms for computers to understand the content of digital images and solve problems such as face recognition, object detection and image segmentation. Visual question answering, which is the subject of focus for the present thesis, is a task at the intersection of NLP and CV. Therefore, after providing a brief, general background on ML, we will focus on the basics of deep learning in the context of both NLP and CV.

2.1.1 Types of Learning Algorithms

ML algorithms can be broadly divided into three categories depending on the information that is provided for learning: supervised, unsupervised and reinforcement learning.

Supervised Learning In the setting of supervised learning, an algorithm learns a function from input examples to target values given a set of data, for which the target responses are known. The set of input-output pairs used for learning is usually referred to as the training dataset. Supervised models are designed to derive a mapping function that approximates the implicit relationships in the training examples with the aim of making accurate predictions about novel inputs. Supervised ML algorithms can be grouped into two major categories based on the desired output:

- Classification, which refers to the problem of predicting outputs that are restricted to a distinct set of values (classes). The target output is often called a *label*. If there are only two possible classes, the task is referred to as binary classification. For more than two classes, each input can be assigned exactly one (multi-class classification) or multiple labels (multi-label classification) of these classes.

- Regression, which refers to the problem of estimating real-valued outputs within a predefined range.

Supervised methods are very powerful and they are thus utilized in the majority of ML applications. Many of the recent deep learning breakthroughs in tasks such as object classification, text generation and machine translation, fall under this category. A more detailed background of supervised learning algorithms will be presented in Sections 2.2-2.3.

Unsupervised Learning In unsupervised learning, algorithms learn to infer patterns within a dataset without being presented with target values for each learning example. This leaves the algorithm to discover the underlying structure or distribution of the data. Two of the most common unsupervised learning problems are clustering and representation learning.

- Clustering is the task of finding groupings in the data based on a predesignated similarity measure such that objects that belong to the same group are more similar to each other than to those in other groups. Clustering is often used for exploratory data analysis, which aims at providing insight into a dataset by identifying patterns, trends and outliers.

- Representation learning comprises a set of techniques for discovering representations of raw data that are conducive to classification or prediction tasks. This replaces manual feature engineering, which can be a difficult and expensive process since it requires domain knowledge. Moreover, unsupervised methods for representation learning often perform dimensionality reduction, i.e. finding representations of the input that lie in a low-dimensional space.

Reinforcement Learning In reinforcement learning, a software agent learns to perform a task in interaction with its environment. The agent is presented with examples without labels, as in unsupervised learning. However, learning is accomplished by trial and error using feedback from its actions. Although both supervised and reinforcement learning use external signals to learn an input-output mapping, supervised learning has access to the target outputs, while reinforcement learning uses positive or negative feedback to discover the actions that maximize its rewards. Reinforcement learning algorithms are widely used in autonomous vehicles and logic games, such as the DeepMind's AlphaGo, an AI system that managed to beat an expert at the ancient Chinese game Go.

2.2 Supervised Machine Learning Algorithms

The subject of this thesis, namely visual question answering, falls under the category of supervised classification tasks. Consequently, the following sections will provide background primarily on supervised learning methods.

Definition Given a dataset of N training examples $D = \{(\boldsymbol{x}_n, \boldsymbol{y}_n), n = 1, ..., N\}$, the task is to learn a function $f : X \to Y$ mapping the input X to the output space Y. How well the function f fits the training data, i.e. how accurately it maps X to Y, is quantified by a *loss function* $\mathcal{L} : Y \times Y \to \mathbb{R}^{\geq 0}$. For instance, given a training example $(\boldsymbol{x}_i, \boldsymbol{y}_i)$ the loss of predicting the value $\hat{\boldsymbol{y}}_i = f(\boldsymbol{x}_i)$ is computed by $\mathcal{L}(\hat{\boldsymbol{y}}_i, \boldsymbol{y}_i)$.

The accuracy of a learning algorithm is measured by its ability to make an accurate prediction $\boldsymbol{y}^*$, when it is presented with a novel input $\boldsymbol{x}^* \notin D$. This is referred to as the *generalization* ability of the algorithm.

The "no free lunch" (NFL) theorem [72] states that all optimization problem algorithms perform equally well when averaged over all possible problems. This implies that no one algorithm works best for learning all possible target functions. Note that the NFL theorem only applies to problems drawn uniformly at random from the space of all problems, which is not the case for real-world problems. This highlights the importance of *inductive bias*, i.e. making assumptions about the about the nature of the target function, when selecting an algorithm for a particular problem. Some other factors to consider when choosing a learning algorithm are accuracy, model complexity, training time, number of parameters and number of features.

2.2.1 Generalization

Definition Generalization refers to a model's ability to perform well on new, previously unseen inputs, drawn from the same probability distribution as the ones used to train the model.

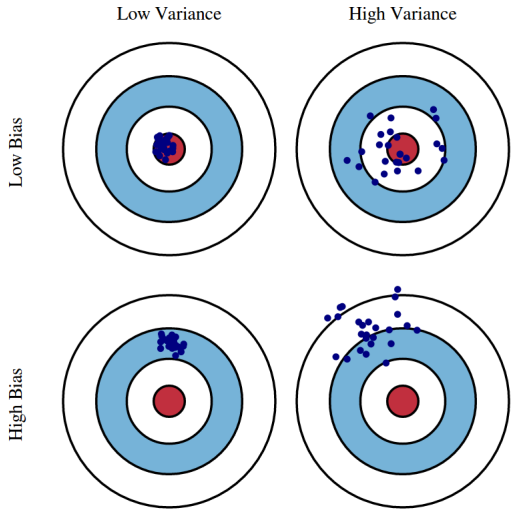


Figure 2.1: Graphical illustration of bias and variance. Figure from: [22]

The entirety of available data is typically split into two parts: the *training set* used for training the model and the *test set* reserved for evaluating the model. It is important to draw a distinction between the training and the generalization error. The training error is calculated on the training data set and it is minimized using an optimization technique. The generalization error is formally the expectation of the model's error were we to apply it to an infinite, previously unseen inputs. In practice, it is approximated by the model's error on the test set.

Ideally, we would like an algorithm to have a low training and generalization error. The prediction errors of any ML algorithm contain a *bias* and a *variance* error term. In order to achieve good model performance, the sum of these errors needs to be minimized. However, there is a trade-off between a model's ability to minimize the bias and the variance simultaneously.

The Bias-Variance Trade-Off

Suppose we have a training data set $D = \{(\boldsymbol{x}_n, \boldsymbol{y}_n), n = 1, ..., N\}$. The training data is actually generated by some function f, such that $\boldsymbol{y} = f(\boldsymbol{x}) + \epsilon$, where ϵ is normally distributed noise with zero mean: $\epsilon \sim N(0, \sigma_\epsilon)$.

The goal of the learning process is to find a model $\hat{f}(\boldsymbol{x})$ to approximate $f(\boldsymbol{x})$ as well as possible. The expected squared prediction error at point $\boldsymbol{x}$ is:

$$Err[\boldsymbol{x}] = E[(\hat{f}(\boldsymbol{x}) - f(\boldsymbol{x}))^2]$$

The prediction error can be decomposed into three terms: [35]:

$$Err[\boldsymbol{x}] = (E[\hat{f}(\boldsymbol{x})] - f(\boldsymbol{x}))^2 + E[(\hat{f}(\boldsymbol{x}) - E[\hat{f}(\boldsymbol{x})])^2] + \sigma_\epsilon^2$$
$$Err[\boldsymbol{x}] = \text{Bias}^2 + \text{Variance} + \text{Irreducible Error}$$

- The bias is the difference between the average model prediction and the correct value. A model with high bias pays very little attention to the training data and oversimplifies the model. It always leads to high error on training and test data.
- The variance term corresponds the variance of the approximating function $\hat{f}$ over all the training data D. It represents the model sensitivity to the choice of the training data D.
- The irreducible error is produced by the noise in the data and cannot be reduced regardless of the learning algorithm.

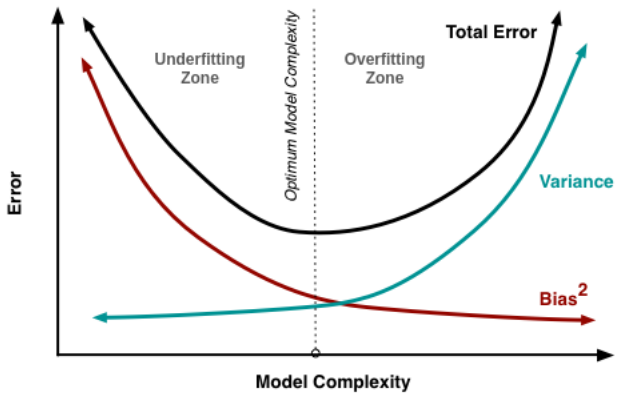


Figure 2.2: The bias-variance trade-off with respect to model complexity. Figure from: [22]

Figure 2.1 presents a bulls-eye diagram to visualize the bias and variance in a 2-dimensional space. The center ring is the target model that predicts correctly the output values. Moving away from the bulls-eye the predictions become more and more inaccurate. The dots represent the training data points.

The reason that there is a trade-off between achieving low bias and low variance, is that these states are interrelated with model complexity (see Fig. 2.2). Low variance models tend to be simpler, while low bias models are usually more complex. The trade-off becomes clear when considering the two extreme cases. A model with zero variance, e.g. a constant function, cannot capture the underlying pattern of the data. Such a simple model would probably have high bias and perform poorly on both the training and the test sets. On the other hand, a model with zero bias, would have to be complex enough to fit every data point in the training set. Although the training error would be zero, such a model would have high variance and a high generalization error.

Underfitting vs. Overfitting

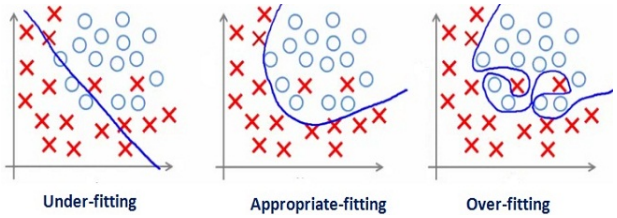


Figure 2.3: Approximating functions of increasing complexity from left to right. Figure from: [1]

The difference between the training and the generalization error is often called the *generalization gap*. The goal of ML algorithms is to minimize both the training error and the generalization gap. In order to achieve that we need to avoid the phenomenons known as underfitting and overfitting (see Fig. 2.3). Whether a model will under- or overfit depends on the its complexity (see Fig. 2.2) and the availability of sufficient training data.

Underfitting occurs when the model is not able to reduce the training error. An underfitted model has not captured the relationship between the inputs and the target outputs. Underfitting is related to low variance - high bias errors.

Overfitting occurs when the model is able to reduce the training error, but the generalization gap is substantial. A model will overfit, if it is complex enough to adapt to the noise in the training data. Overfitting is related to high variance - low bias errors. A common strategy to combat overfitting is to withhold a subset of the training examples known as the *validation set*, on which different models can be evaluated until we determine the appropriate model complexity.

2.2.2 Logistic Regression

Logistic regression (LR) is a linear classification model. LR computes the probabilities for a classification problems with two possible outcomes by applying the logistic function to the output of a linear function f. The logistic function, also known as the sigmoid function, squeezes a vector into a range of $(0, 1)$. For a binary classification problem, the probability of one of the classes for a feature vector $\boldsymbol{x} \in \mathbb{R}^d$ is computed as:

$$P(y = 1|\boldsymbol{x}) = \frac{1}{1 + e^{-f(\boldsymbol{x})}} \tag{2.1}$$

where f is a linear function with w_i parameters:

$$f(\boldsymbol{x}) = w_0 + w_1 x_1 + ... + w_d x_d \tag{2.2}$$

The probability of the other class would be $P(y = 0|\boldsymbol{x}) = 1 - P(y = 1|\boldsymbol{x})$. Each input is assigned the class label for with the posterior probability is greater than 0.5.

The parameters of the linear function are computed by minimizing the cross-entropy loss J, which is defined as:

$$J(\boldsymbol{w}) = -[y \log(P(y = 1|\boldsymbol{x})) + (1 - y) \log(1 - P(y = 1|\boldsymbol{x}))] \tag{2.3}$$

This optimization problem can be solved using the gradient descent algorithm, which will be described further in Section 2.3.3.

2.2.3 Support Vector Machines

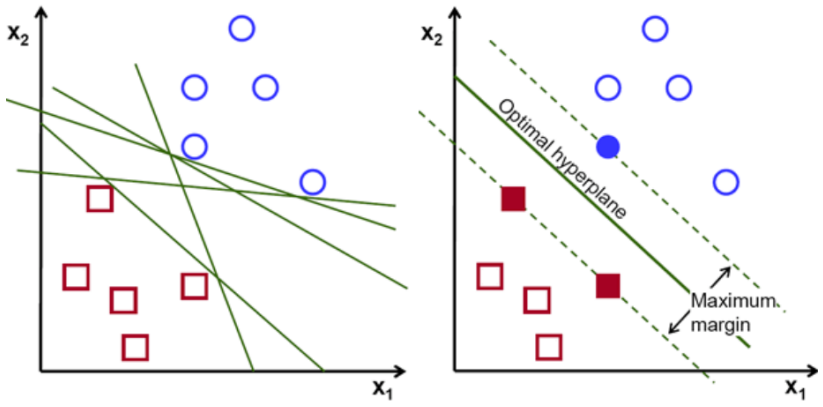


Figure 2.4: Left: Examples of hyperplanes that separate the data points. Right: The hyperplane chosen by the SVM algorithm.

Support Vector Machines (SVMs) are linear classification models as well. Suppose we have a binary classification problem and that the training data are linearly separable in the feature space. One could choose any of such solutions as the decision boundary (see Fig. 2.4). The objective of the SVM algorithm is to find the hyperplane with the maximum margin, i.e. to maximize the smallest distance between the hyperplane and any of the samples [11].

A hyperplane in a d dimensional space can be described mathematically by the equation:

$$\boldsymbol{w}^T \boldsymbol{x} + b = 0 \tag{2.4}$$

where $\boldsymbol{x} \in \mathbb{R}^d$ are the input features, $\boldsymbol{w} \in \mathbb{R}^d$ and $b \in \mathbb{R}$ are the parameters. Without the bias term b the model would be limited to hyperplanes that go through the origin. We assume that the training

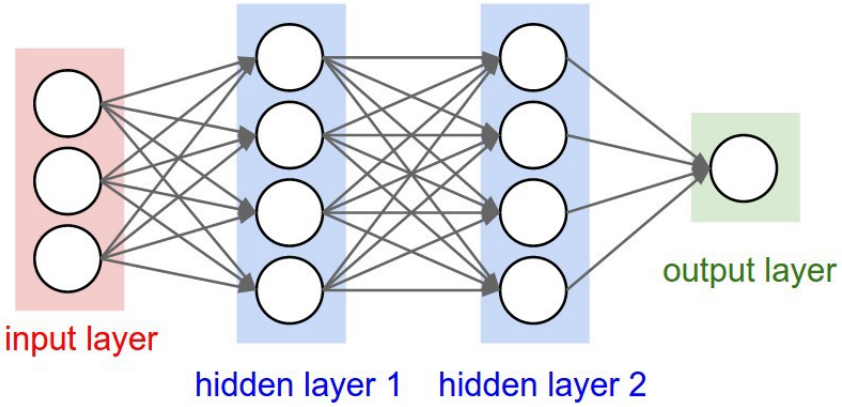


Figure 2.5: Overview of an artificial neural network. Figure from [37].

data set with class labels $y \in \{-1, +1\}$, that is linearly separable. There exists at least one choice of the parameters $\boldsymbol{w}$ and b such that for the function $f(\boldsymbol{x}) = \boldsymbol{w}^T\boldsymbol{x} + b$ we get:

$$f(\boldsymbol{x}) > 0 \quad \forall\, \boldsymbol{x} \text{ with label } y = +1 \tag{2.5}$$
$$f(\boldsymbol{x}) < 0 \quad \forall\, \boldsymbol{x} \text{ with label } y = -1 \tag{2.6}$$

That means that $y \cdot f(\boldsymbol{x}) > 0$ for all training data points $\boldsymbol{x}_i$. The distance of a point $\boldsymbol{x}$ from the hyperplane is given by $|f(\boldsymbol{x})|/\|\boldsymbol{w}\|$. We can scale $\boldsymbol{w}$, so that the value of $f(\boldsymbol{x})$ is equal to 1 for the nearest points with $y = 1$ and equal to -1 for the nearest points with $y = -1$. The maximum margin then becomes equal to $2/\|\boldsymbol{w}\|$ requiring that:

$$f(\boldsymbol{x}) \geq -1 \quad \forall\, \boldsymbol{x} \text{ with label } y = +1 \tag{2.7}$$
$$f(\boldsymbol{x}) \leq -1 \quad \forall\, \boldsymbol{x} \text{ with label } y = -1 \tag{2.8}$$

The task is now formulated as finding the parameters $\boldsymbol{w}$, b such that

$$min \quad J(\boldsymbol{w}) = \frac{1}{2}\|\boldsymbol{w}\|^2 \tag{2.9}$$
$$\text{subject to} \quad y_i(\boldsymbol{w}^T\boldsymbol{x}_i + b) \geq 1, \quad \forall \boldsymbol{x}_i \tag{2.10}$$

This is an example of a quadratic programming problem which is solved using the method of Lagrange multipliers [11]. The SVM algorithm can be extended for datasets that are not linearly separable by using the kernel trick, projecting data points into high-dimensional feature spaces, where they become linearly separable.

2.3 Deep Learning

In order to give an overview of deep learning, we will first introduce the concept of artificial neural networks.

Artificial neural networks (ANNs) are a computational learning paradigm loosely inspired by the biology of the human brain [54]. The term "artificial neural networks" does not describe a particular algorithm, but rather a framework to design and train ML models for a wide variety of tasks. ANNs can process any kind of real-world data that is represented as a vector, be it text, image, sound etc. That is because ANNs are able to extract features from raw input data and to be trained in an end-to-end manner. This eliminates the need for engineering hand-crafted features, a process which requires domain expertise and is often both task-specific and time-consuming.

Deep learning (DL) is used to describe the set of techniques for learning in deep neural networks [54]. Deep ANNs are typically composed of multiple layers of units. The attribute "deep" refers to

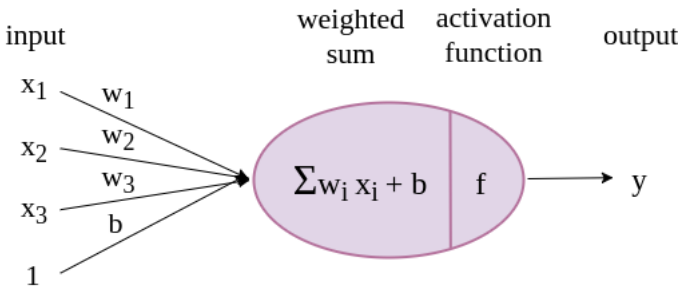


Figure 2.6: The perceptron. The terms w_i represent the neural weights and b represents the bias.

the number of layers in a neural network. The first layer is called the input layer and the last one is called the output layer. Any layer between the input and the output layer is called hidden (see Fig 2.5). ANNs with one hidden layer are described as *shallow*, while ANNs with two or more stacked hidden layers are described as *deep*. Deep neural networks are able to learn a hierarchy of features, as the features are transformed from one layer to the next, forming more and more complex representations.

According to the *Universal Approximation Theorem* [16], an ANN with only one hidden layer can approximate any continuous function with arbitrary precision. More formally, given any continuous function $f(\boldsymbol{x})$ and some $\epsilon > 0$, there exists an neural network $\hat{f}(\boldsymbol{x})$ with one hidden layer such that $|f(\boldsymbol{x}) - \hat{f}(\boldsymbol{x})| < \epsilon \; \forall \boldsymbol{x}$. Even though shallow neural networks are universal function approximators, in practice deeper networks are necessary to learn most tasks. The layer of a shallow network that approximates f may be infeasibly large or such an ANN may fail to generalize correctly [24].

In the past decade, DL has experienced an explosive growth. However, the basic ideas around DL started forming in the 1940s. There are two main contributing factors for the recent success of DL. On the one hand, the availability of larger, labeled datasets has played a key role. Deep neural networks require massive amounts of data for training. On the other hand, deeper networks and larger datasets also require a lot of computing power. The efficient training of DL models was enabled by the high parallelization of computations, that is achieved by multi-core hardware architectures such as Graphics Processing Units (GPUs).

2.3.1 The Perceptron

The simplest possible type of ANNs is the *perceptron* [61]. The perceptron is a network with a single unit that can be used to for binary classification problems assuming that the two classes are linearly separable. The algorithm was developed in the 1950s by Frank Rosenblatt. As it is illustrated in Figure 2.6, the perceptron first sums up the weighted inputs and a bias, and then applies to the weighted sum an activation function such as the sign or the sigmoid function.

Suppose we have a training data set $D = \{(\boldsymbol{x}_n, y_n), n = 1, ..., N\}$, where $\boldsymbol{x}_n \in \mathbb{R}^d$ and $y_n \in \{-1, 1\}$. The two classes can be separated by a hyperplane with parameters $\boldsymbol{w} \in \mathbb{R}$, $b \in \mathbb{R}$ (see Eq. 2.4-2.6). To simplify the notation we will incorporate the bias term b to the weights $\boldsymbol{w}$ by expanding the input and weight vectors such that

$$\boldsymbol{x} = [1, x_1, ...x_d]^T \tag{2.11}$$

$$\boldsymbol{w} = [b, w_1, ...w_d]^T \tag{2.12}$$

Let M be the set of all data points missclassified by a perceptron with parameters $\boldsymbol{w}$. The cost function is defined as:

$$J(\boldsymbol{w}) = \sum_{\boldsymbol{x} \in M} y \boldsymbol{w}^T \boldsymbol{x} \tag{2.13}$$

The cost function is non-negative and it becomes zero when the weights $\boldsymbol{w}$ correspond to a solution, that is when no point in the dataset is missclassified. The algorithm for finding a solution is similar to gradient descent. Starting with a random initialization of $\boldsymbol{w}$, we iteratively update the weights in

the direction of the negative gradient of the cost function $J(\boldsymbol{w})$. However, the cost function is not differentiable at all points in the weight space. As we adjust the weights $\boldsymbol{w}$, there is a point where a data sample x that was previously missclassified, is classified correctly by the new hyperplane. At this point the number of samples in M changes and the gradient at this point is not defined.

To recapitulate the steps in the perceptron algorithm, starting with an initialization of the weights $\boldsymbol{w}(0)$ we apply iteratively the following update rule:

$$\boldsymbol{w}(t+1) = \boldsymbol{w}(t) - \alpha(t)\boldsymbol{x}(t) \tag{2.14}$$

where $\alpha(t)$ is the *learning rate* at iteration t, i.e. a constant in the range $(0, 1]$ that controls the size of the update. The update process is repeated until the cost function becomes equal to zero. It has been proven that the perceptron algorithm converges to a solution after a finite number of steps [66].

The learning rate is an example of what is known as *hyperparameters* of ML algorithms, that means parameters whose values are not learned by the algorithm itself. Hyperparameters are properties of the algorithm that have to be set before the learning process and they are usually chosen empirically.

The perceptron can also be trained in an online fashion. This means that each data sample in D is examined successively and in case it is misclassified the update rule (Eq. 2.14) is applied. Once all data samples in D have been examined, we say that an *epoch* has been completed. This process is repeated for a finite number of epochs.

Comparing the Perceptron to Logistic Regression and SVMs

In previous sections, we have described two more linear classification methods, namely logistic regression (Sec. 2.2.2) and SVMs (Sec. 2.2.3).

Both the perceptron and the SVM algorithm aim at finding a hyperplane that classifies correctly the samples from both classes. Nonetheless, the solution of the perceptron algorithm is not unique. Since there exist multiple hyperplanes that separate the classes, the algorithm could converge to any of them depending on the weight initialization and the learning rate. On the contrary, the SVM algorithm produces the hyperplane that maximizes the distance of each sample from the decision boundary (see Fig. 2.4). Another difference of these algorithms is that the perceptron can be trained online, while the SVM requires all the training samples at once.

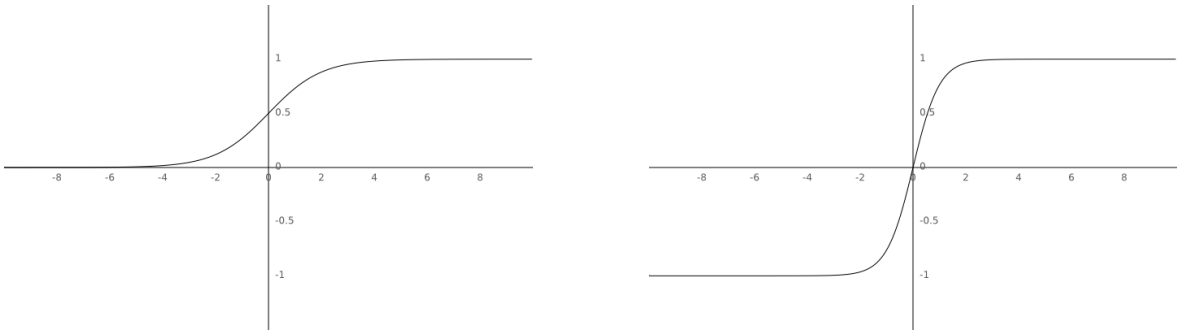
A perceptron with a sigmoid (logistic) activation function is equivalent to logistic regression. Using the sigmoid function allows us to interpret the output of the perceptron as the probability of the input belonging to one of the classes. The prediction is based on whether the output is greater than 0.5 or not. For instance, if the output of the perceptron is interpreted as the probability $P(y_i = 1|\boldsymbol{x}_i; \boldsymbol{w})$, we can formulate the cross-entropy loss (Eq. 2.3) and train the model as we would a logistic regression classifier.

The perceptron can be considered as the building block of feedforward networks, which we will discuss next.

2.3.2 Feedforward Networks

Feedforward networks, also called multilayer perceptrons (MLPs), consist of multiple stacked layers of computational *units*, which are connected without any feedback loops. Each unit is a perceptron that takes as input the output of the previous layer and implements a vector-to-scalar function. There are typically no connections between units of a single layer. So, feedforward networks can represented as acyclic graphs that compose together many different functions.

A feedforward networks is made of an input layer that accepts the input data, hidden layers that process outputs from the previous layer and an output layer that provides the final output. The flow of information from the input to the output is called *forward propagation*. The number of hidden layers determines the depth of a model, while the number of the units in the hidden layers determines the width of the model.



(a) $\sigma(x)$ **(b)** $tanh(x)$

Figure 2.7: Left: The sigmoid function. Right: The tanh function.

The goal of a feedforward networks is to approximate a functions f that maps the input $\boldsymbol{x}$ to a target y. Linear models, such as logistic regression and the perceptron, can learn only linear functions in the form of $y = \boldsymbol{w}^T \boldsymbol{x}$, where $\boldsymbol{w}$ are the model parameters. Feedforward networks, however, are not limited to linear functions thanks to the use of non-linear activation functions. These models can learn non-linear transformations ϕ of the input such that $y = \boldsymbol{w}^T \phi(\boldsymbol{x}; \boldsymbol{\theta})$. The function ϕ with parameters $\boldsymbol{\theta}$ is defined by the hidden layers of the network. Both $\boldsymbol{\theta}$ and $\boldsymbol{w}$ are learned during training using a gradient-based method.

Activation Functions

As noted above, the purpose of activation functions is to introduce non-linearities to the network. Non-linear activation functions are a prerequisite for the universal approximation theorem. Without non-linearities the model capacity would not grow as the depth of the model increases. Lets consider, for example, a linear neural network with one hidden layer of two units $\boldsymbol{w}_1$ and $\boldsymbol{w}_2$ and one output unit $\boldsymbol{w}_o$:

$$\begin{aligned} y &= w_{o1}(w_{11}x_1 + \dots w_{1n}x_n) + w_{o2}(w_{21}x_1 + \dots w_{2n}x_n \\ &= (w_{o1}w_{11} + w_{o2}w_{21})\, x_1 + \cdots + (w_{o1}w_{1n} + w_{o2}w_{2n})\, x_n \\ &= \bar{w}_1 x_1 + \cdots + \bar{w}_n x_n \end{aligned} \tag{2.15}$$

It's obvious that a feedforward network without linear activation functions is equivalent to a linear neural network without hidden layers. In this section, we present some of the most commonly used activation functions for deep neural networks.

Sigmoid

The sigmoid function is defined as:

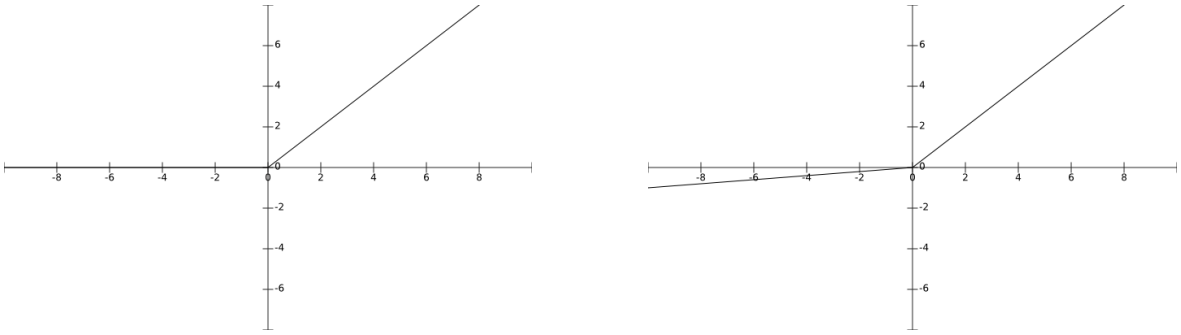
$$\sigma(x) = \frac{1}{1 + e^{-x}} \tag{2.16}$$

As seen in Figure 2.7a, the sigmoid function squashes real-valued numbers to the range $(0, 1)$. In particular, large negative numbers become 0 and large positive numbers become 1. This causes the problem of *vanishing gradients*. In the saturation regions the gradient is almost zero, which means that during training the weights will barely be adjusted. Moreover, the use of sigmoid activations makes the learning process sensitive to the initialization of the weights. If the initial weights are too large or too small, the small gradients will hinder the learning.

Hyperbolic tangent

The hyperbolic tangent (tanh) function is defined as:

$$tanh(x) = \frac{e^x - e^{-x}}{e^x + e^{-x}} \tag{2.17}$$



(a) $f(x) = \max(0, x)$ **(b)** $f(x) = \max(0.1x, x)$

Figure 2.8: Left: The ReLU function. Right: The leaky ReLU function.

The tanh function (see Fig. 2.7b) is similiar to the sigmoid function, but it is zero-centered. Tanh squashes real-valued numbers to the range $(-1, 1)$. In fact, the tanh function is a scaled sigmoid:

$$tanh(x) = 2\sigma(2x) - 1 \tag{2.18}$$

Therefore, the tanh activation function suffers from the vanishing gradient problem as well.

Rectified Linear Unit

The rectified linear unit (ReLU) function is defined as:

$$f(x) = max(0, x) \tag{2.19}$$

This ReLU (see Fig. 2.8a) is the most common activation function in feedforward neural networks. The popularity of the ReLU is attributed to the fact that it is a piecewise linear function, which facilitates the optimization with gradient-based methods. In addition, ReLUs are computationally efficient compared to sigmoid or tanh activations, which involve expensive operations with exponentials. Although ReLUs do not suffer from the vanishing gradient problem, a complication that can arise from their use is the problem of the *"dying ReLU"*. A ReLU dies when the weights are updated in such a way that the output of the unit becomes zero for any input. This means that the gradient flowing through the unit will be zero and the weights will not be updated. As a result, that unit will not activate (i.e. become non-zero) from that point on. Another drawback of ReLUs is that the output value is not bound, which can lead to activations blowing up in deep networks.

In order to overcome the problem of the "dying" ReLU, the leaky ReLU was proposed [52]. The leaky ReLU (see Fig. 2.8b) has a small slope instead of becoming zero for negative values and it defined as:

$$f(x) = \begin{cases} \alpha x & \text{if } x < 0 \\ x & \text{if } x \geq 0 \end{cases} \tag{2.20}$$

where α is a small positive constant.

2.3.3 Training Artificial Neural Networks

Loss Functions

ANNs are trained using an optimization method, which aims to find the set of model parameters that minimizes the prediction error. The prediction error of a model with $\boldsymbol{w}$ parameters is estimated by a *loss function* $J(\boldsymbol{w})$. The loss function computes a non-negative value that measures the inconsistency between the predicted and the target output.

Most deep ANNs are trained under the framework of *maximum likelihood estimation* (MLE). MLE is a method of estimating the best model parameters $\hat{\boldsymbol{w}}$ by maximizing the likelihood function,

or equivalently by minimizing the negative likelihood. Suppose we have a training dataset $D = \{(\boldsymbol{x}_n, \boldsymbol{y}_n), n = 1, \ldots, N\}$ The maximum likelihood estimate is defined as:

$$\hat{\boldsymbol{w}} = \arg\max_{\boldsymbol{W}} \prod_{i=1}^{N} P(y = y_i | \boldsymbol{x}_i; \boldsymbol{w}) \tag{2.21}$$

In practice, it is common to take the logarithm of the likelihood function. This monotone transformation makes the computation of the derivatives more convenient without changing the maxima:

$$\hat{\boldsymbol{w}} = \arg\max_{\boldsymbol{W}} \sum_{i=1}^{N} \log P(y = y_i | \boldsymbol{x}_i; \boldsymbol{W}) \tag{2.22}$$

Under the MLE framework the loss function estimates how closely the distribution of predictions matches the distribution of the targets in the training data. This dissimilarity between the empirical distribution p and the predicted distribution q can be expressed by the *cross-entropy* H. The loss function is computed as:

$$J(\boldsymbol{w}) = H(p, q) = -\frac{1}{N} \sum_{i}^{N} log P(y = y_i | \boldsymbol{x}_i; \boldsymbol{w}) \tag{2.23}$$

Note that the performance of an ANN on a particular task is typically evaluated by a measure P different that the loss J. This measure, however, is often unsuitable for gradient-based optimization. As a result, we minimize a carefully selected loss function and expect that by minimizing the loss we will optimize the performance measure P.

Suppose we have a classification task with K possible classes. We want the output layer to calculate the probability distribution over the K classes. In order to convert the values z_i of the output layer to probabilities, we use the *softmax* function. The softmax function is defined as:

$$\text{softmax}(z_i) = \frac{e_i^z}{\sum_{j=1}^{K} e^{z_j}} \tag{2.24}$$

The values z_i are first exponentiated to become positive and then normalized so that the sum of the output values for all classes adds up to 1. The softmax function can be seen as a generalization of the sigmoid function, which is often used to compute the probability distribution in case of a binary classification problem.

Optimization

Training an ANN is essentially the optimization problem of finding the set of model parameters $\boldsymbol{w}$ that minimizes the average loss function $J(\boldsymbol{w})$ over the training data. This problem is usually addressed using *gradient descent*. Gradient descent is an iterative method. Parameters $\boldsymbol{w}$ are randomly initialized and at each iteration they are adjusted by taking a step in the parameter space following the direction of the negative gradient of the loss function $\boldsymbol{\nabla} J(\boldsymbol{w})$. The biggest difference between training a linear model like the perceptron and a deep neural network is that the non-linearity of deep models causes the loss surface to become non-convex (see Fig. 2.9). This means that there is no guarantee that a gradient based method will converge to a global minimum, where the loss function is zero. In fact, provably solving the optimization problem of deep neural networks belongs to the class of an NP-hard problems [36]. That means that there probably exists no algorithm that finds the optimal set of parameters for a deep NN in polynomial time.

The parameters $\boldsymbol{w}$ at the i-th iteration are updated according to the rule:

$$\boldsymbol{w}_{i+1} = \boldsymbol{w}_i - \alpha \boldsymbol{\nabla} J(\boldsymbol{w}_i) \tag{2.25}$$

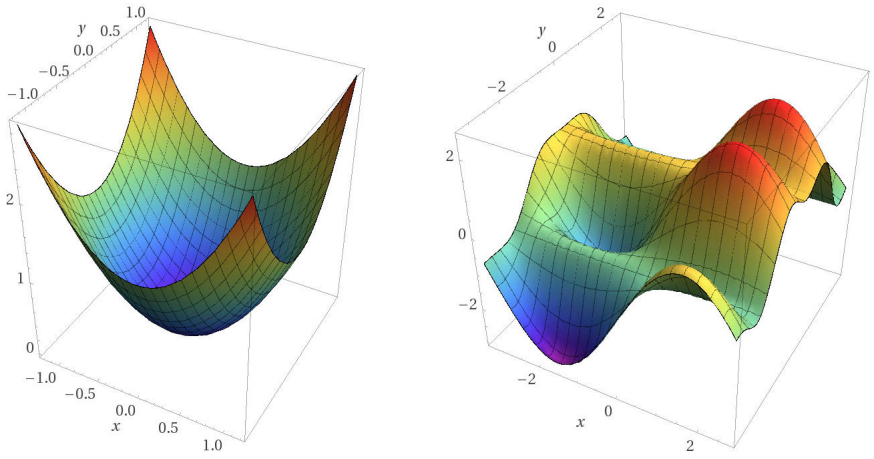


Figure 2.9: Left: The surface of a convex loss function. Right: The surface of a non-convex loss function. Figure from [74]

where α is the learning rate. The gradient defines the direction in which we should update the weights, but it does not provide any information as to the magnitude of the update step. As we saw in the description of perceptron algorithm (see Eq. 2.14), this is controlled by the learning rate. The learning rate is one of the most important hyperparameters in training ANNs. On the one hand, if the learning rate is too big, Gradient Descent may overshoot the minima and bounce back and forth on the loss surface. On the other hand, if the learning rate is too small, it may take too long to reach a local minimum, or the algorithm might get stack at a suboptimal local minimum.

An analogy that is often used to provide some intuition for the Gradient Descent algorithm is that of a blindfolded person trying to move down a valley. Imagine that every point in a valley corresponds to a setting of the parameters and the height at each point corresponds to the value of the loss function for that particular setting of the parameters. In order to reach the bottom of the valley without knowing its exact location, one could assume that the best strategy is to take steps in the direction where the slope of the ground is the steepest. This is analogous to updating the model parameters in the opposite direction of the gradient.

Backpropagation

In order to efficiently calculate the weight updates for a deep neural network, we employ the *back-propagation* algorithm [62, 46]. Backpropagation is a recursive method that uses the chain rule to compute the partial derivatives of the loss at each layer of weights. Starting with computing the gradient of the output layer and moving towards the input layer, the backpropagation algorithms avoids repeating computations by caching intermediate results.

Variants of Gradient Descent

We can categorize gradient descent variants into three groups based on the amount of data used to compute the gradient of the loss function.

- Batch gradient descent computes the loss function by averaging the loss for each datum the entire training set. For large datasets this can be very computationally expensive.
- Stochastic gradient descent (SGD) [12] updates the parameters using the gradient of the loss at a single data point at a time. Although it is faster than batch gradient descent, it can lead to noisy gradients and cause the loss function to fluctuate.
- Mini batch gradient descent falls in between the batch and the stochastic variants. The size of a batch is a learning hyperparameter that is set empirically. Larger batches provide a more

accurate estimate of the gradient, while the noisier estimate of smaller batches can help prevent overfitting. Batches can be processed in parallel, which speeds up the training process. The size of the batch is often limited by the available memory of the hardware. In practice, this is the most commonly used variant of gradient descent.

Challenges of Gradient Descent

Two common issues that arise when dealing with the optimization of a non-convex loss function [24] are local minima and saddle points. As the optimization is based on the gradient, there is a danger that the algorithm will reach a suboptimal point and not be able to escape it as the gradient becomes really small. Local minima are problematic, if they have a high loss in comparison to the global minimum. However, research suggests that saddle points are actually much more common and difficult to overcome [18], as they tend to be surrounded by plateaus of the loss surface. The noise that is introduced by stochastic gradient descent helps escape local minima and saddle points.

In addition, in the last few years a number of variants of SGD have been proposed to overcome these optimization challenges. Adagrad [20] adapts the learning rate to the parameters. This means that parameters associated with frequently occurring features will have a smaller learning rate than more rare ones. As a result, Adagrad works particularly well for sparse datasets. RMSprop is an unpublished modification of Adagrad introduced by Geoffrey Hinton. It was developed to counteract the fast decay of the learning rate by accumulating only recent gradients for the scaling of the learning rate. Lastly, Adam [40] also computes an adaptive learning rate for each parameter. On top of keeping a running average of past gradients withing a fixed-size window like RMSprop, Adam also utilizes the running average of the past squared gradients. In recent years, Adam has become the default optimizer for most applications.

Regularization

Regularization refers to a set of techniques used during training to prevent overfitting. Overfitting is a prevalent problem in deep learning. The large number of parameters, that is often in the millions, can lead to deep neural networks memorizing the training data. During training, as the model parameters are updated the loss computed over the training data diminishes. However, if the model capacity is high enough, after a certain point the parameters start adjusting to the noise in the data and the generalization error rises (see Fig. 2.10).

Most regularization strategies aim at balancing the bias-variance trade-off. In the context of model error, that means achieving a lower generalization error at the expense of increasing the training error [24]. One way to achieve that is by regulating the model capacity. However, that does not necessarily mean that the generalization gap can be closed simply by determining the appropriate number of model parameters. Since we do not have access to the underlying distribution of the data, the best strategy usually is to allow the model to have access to highly complex function families and use regularization to control the complexity. Some regularization techniques attempt to improve generalization by introducing prior knowledge or encoding a preference for simpler models. Other methods, known as ensemble methods, combine multiple models to increase the overall robustness.

In this section, we will present some common regularization techniques.

Parameter Norm Penalties

A standard approach for limiting the complexity of a deep neural network is to penalize the norm of the model parameters. This is achieved by adding to the cost function $J(\boldsymbol{w})$ another term $\Omega(\boldsymbol{w})$ known as the regularization term. The overall loss function then becomes:

$$J'(\boldsymbol{w}) = J(\boldsymbol{w}) + \lambda\Omega(w) \tag{2.26}$$

where λ is a constant in the range $[0, 1]$ that controls the strength of the regularization. The two main norm penalties are the following:

- L_1 *regularization*:

$$\Omega(\boldsymbol{w}) = \|\boldsymbol{w}\|_1 = \sum_i |w_i| \tag{2.27}$$

The update rule of gradient descent defined in Eq. 2.25 becomes:

$$\boldsymbol{w}_{i+1} = \boldsymbol{w}_i - \alpha(\boldsymbol{\nabla} J(\boldsymbol{w}_i) + \lambda \operatorname{sign}(\boldsymbol{w}_i)) \tag{2.28}$$

L_1 regularization encourages the network to become sparse. At each iteration, we subtract a constant factor from all weights with a sign equal to $\operatorname{sign}(w_i)$. This can eventually driving some of the network weights to zero. This property acts as a feature selection mechanism.

- L_2 *regularization*, also known as *weight decay*:

$$\Omega(\boldsymbol{w}) = \frac{1}{2} \|\boldsymbol{w}\|_2^2 \tag{2.29}$$

The update rule of gradient descent defined in Eq. 2.25 becomes:

$$\boldsymbol{w}_{i+1} = \boldsymbol{w}_i - \alpha(\boldsymbol{\nabla} J(\boldsymbol{w}_i) + \lambda \boldsymbol{w}_i) \tag{2.30}$$

At each iteration we subtract from the weights an additional term that is proportional to their magnitude. This causes the weights to decay over time, hence the name "weight decay". Large weights are penalized more severely than small ones. Consequently, L_2 regularization encourages the weights of the network to remain small.

Sometimes L_1 and L_2 regularizers are combined. This is known as *elastic net* regularization. The contribution of each regularizer is controlled by the hyperparameter $\lambda \in [0, 1]$ as follows:

$$\Omega(\boldsymbol{w}) = (1 - \lambda) \|\boldsymbol{w}\|_1 + \frac{\lambda}{2} \|\boldsymbol{w}\|_2^2 \tag{2.31}$$

Early Stopping

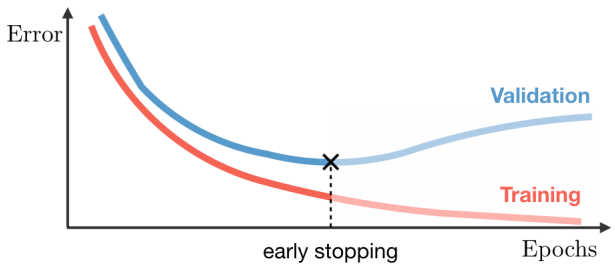


Figure 2.10: Overfitting occurs when the training error continues to decrease, while the generalization error rises. This is combatted by applying early stopping. Figure from [4]

Early stopping is a very useful technique for preventing overfitting. As mentioned above, after a certain number of iterations the model starts to learn the noise in the training data. It is therefore important to monitor at each epoch the performance of our model on the validation set. Early stopping terminates the training process as soon as the validation loss stops decreasing. In practice, the validation loss may start decreasing again after a few epochs. We usually allow the model to continue training for a predefined number of epochs, which is often referred to as *patience*. In essence, early stopping is tuning the hyperparameter number of epochs.

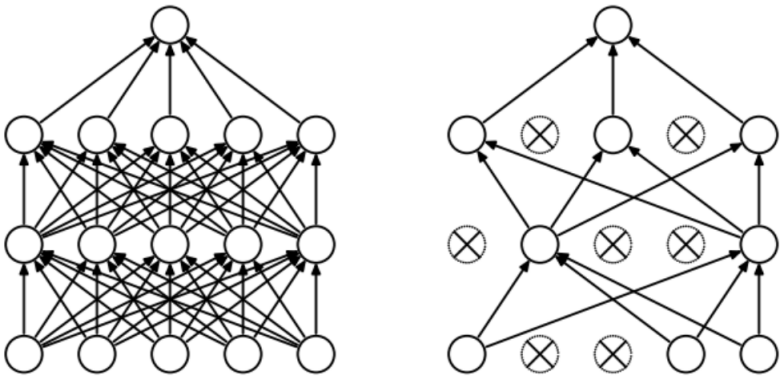


Figure 2.11: The effect of dropout during training. Figure from [64]

Data Augmentation

One straightforward way to reduce the risk of overfitting is to use more training data. Collecting large amounts of labeled training data can be costly. Instead, one can create synthetic data from the already available dataset. This strategy is called *data augmentation*. The data augmentation methods that can be applied to a training dataset in order to create new examples depend on the specific domain. Data augmentation has been utilized successfully for image classification. It is common to use small transforms of the original images that preserve the label of each sample. Such transforms can include flipping, rotating, rescaling and cropping of the image. For the natural-language domain data augmentation is less common. Nonetheless, one way to augment text data is synonym replacement. In general, data augmentation can also be achieved injecting small random noise to the data. This it helps improve the robustness of neural networks.

Dropout

Dropout [64] is one of the most frequently used regularization techniques for deep learning. It is a simple yet effective method. During training each unit is dropped with with probability $p > 0$. Being "dropped" refers to a unit being ignored during both the forward and the backward pass of an iteration (see Fig. 2.11). At test time, all units are employed, but they have to be scaled by a factor $1/p$ to account for the missing activations during training. Dropout prevents units from forming co-dependencies amongst each other.

Suppose we have a model with N units, each of which can be dropped. Training a neural network with dropout can be seen as training a collection of 2^N subnetworks that share some of their parameters. As a result, dropout can be seen as a form of ensembling an exponential number models without the requirement of training and evaluating multiple ones.

2.3.4 Recurrent Neural Networks

Recurrent neural networks (RNNs) are a type of neural networks that process sequences of input data. Unlike feedforward networks that operate under the assumption that all training instances are independent, RNNs produce their output taking into consideration information presented previously presented to them. [24]. As it is shown in Figure 2.12, RNNs have loops that allow the information to be passed from one step to the next. Formally, given an input sequence $\boldsymbol{x}_1, \boldsymbol{x}_2, ...\boldsymbol{x}_t$, the hidden state of an RNN unit with parameters θ is computed as:

$$\begin{aligned} \boldsymbol{h}(t) &= g_h(\boldsymbol{W}_{hh}\boldsymbol{h}(t-1) + \boldsymbol{W}_{xh}\boldsymbol{x}(t) + \boldsymbol{b}) \\ \boldsymbol{y}(t) &= g_y(\boldsymbol{W}_{hy}\boldsymbol{h}(t) + \boldsymbol{c}) \end{aligned} \tag{2.32}$$

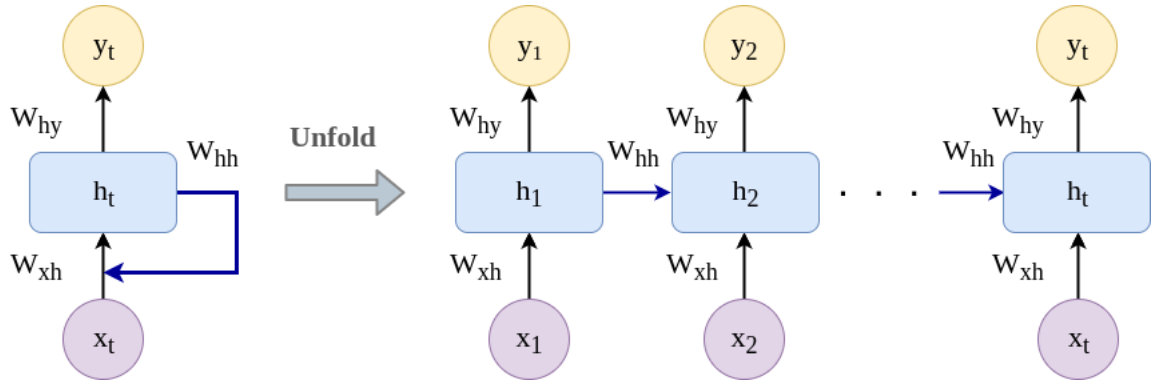


Figure 2.12: *Left:* A recurrent neural network with one input and one output. *Right:* A recurrent neural network unfolded for a sequence of length t.

where g_h and g_y are activation functions. The weight matrices $\boldsymbol{W}_{xh}$ and $\boldsymbol{W}_{hh}$ can be regarded as filters that determine the importance of the input and the previous hidden state respectively. At each step the output is computed using the same weight matrices and biases. Parameter sharing across time steps improves generalization in two ways. It allows the processing of sequences of variable lengths and and the extraction of information that can occur at different positions within a sequence (e.g. "I went to the office on Monday.", "On Monday I went to the office.").

The loss L is computed once the entire input has been processed as the sum of the losses $L(t)$ at each time step:

$$L = \sum_t L(t) = -\sum_t log(\boldsymbol{y}(t)) \tag{2.33}$$

In order to optimize an RNN, the gradient needs to be propagated not only through the depth of the model but also through previous time steps. This is known as back-propagation through time [71]. When optimizing RNNs a common issue is that these long-term dependencies cause the gradients to become either very large (explode) or very small (vanish). The problem of *exploding and vanishing gradients* is mitigated by special types of RNNs, such as the Long Short-Term Memory networks (LSTMs), which we will introduce in the following section.

Some variants of RNNs include the following:

- Deep RNNs: By stacking simple RNNs on top of each other we can build deeper networks that are able to capture the hierarchy of the input data.
- Bidirectional RNNs: Bidirectional RNNs are based on the idea that the output at time t may depend on both previous and future information in the sequence. The output of each unit is a combination (sum, multiplication, concatenation) of the outputs of a forward RNN, which processes the sequence normally, and a backward RNN, which begins processing the sequence from the end. There are no connections between the hidden states of the two RNNs.
- Sequence-to-Sequence RNNs: This architecture enables the mapping between sequences of variable lengths. The encoder network maps the input sequence to a fixed-sized vector, which is fed into the decoder network to generate the output sequence. Sequence-to-sequence modeling will be discussed in further detail in Section 3.3.

Long Short-Term Memory Networks

Long short-term memory (LSTM) networks were introduced by Hochreiter and Schmidhuber [31] in order to combat the problem of exploding or vanishing gradients during the optimization of RNNs. LSTMs are able to effectively learn long-term dependencies and they have been applied successfully to a wide range of problems including speech recognition, text classification and image captioning tasks.

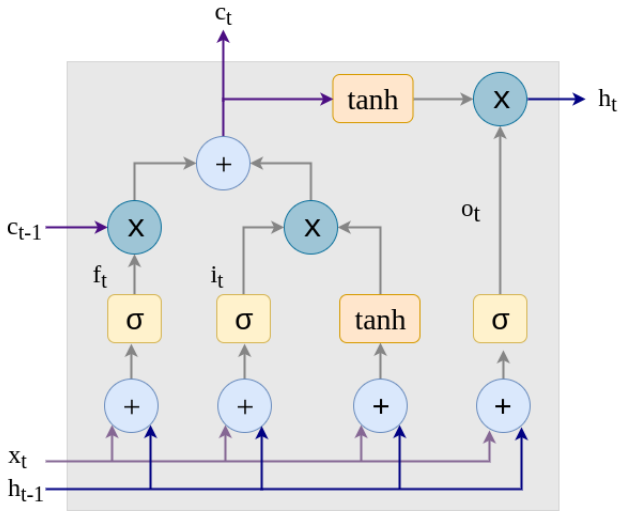


Figure 2.13: The LSTM cell.

The building block of an LSTM network is called a "cell" (see Fig. 2.13). An LSTM cell makes use of of three gating units in order to determine the flow of information, and control which information to preserve in its internal states and which to forget. In particular an LSTM cell (see Fig. 2.13)is made of:

- the *forget gate* $\boldsymbol{f}_t$, which determines the information that is going to be erased from the cell state. Information from the current input x_t and the previous hidden state h_{t-1} is passed through a sigmoid activation function 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,
- the *input gate* $\boldsymbol{i_t}$, which controls which values of the input will be carried over to the new cell state,
- the *output gate* $\boldsymbol{o}_t$ which controls how much of the cell to reveal at the output based on the current input and previous hidden state.
- the *memory cell* $\boldsymbol{c}_t$, which is updated based not only on past information but also on current input. The past information is filtered by the forget gate f_t, while the new information is filtered by the input gate i_t,
- the *hidden state* $\boldsymbol{h}_t$ by which the LSTM encodes the sequence up to the time step t.

The set of equations that define LSTM cell are :

$$\begin{aligned} \boldsymbol{f}_t &= \sigma(\boldsymbol{W}_f \boldsymbol{x}_t + \boldsymbol{U}_f \boldsymbol{h}_{t-1} + \boldsymbol{b}_f) \\ \boldsymbol{i_t} &= \sigma(\boldsymbol{W}_i \boldsymbol{x}_t + \boldsymbol{U}_i \boldsymbol{h}_{t-1} + \boldsymbol{b}_i) \\ \boldsymbol{o}_t &= \sigma(\boldsymbol{W}_o \boldsymbol{x}_t + \boldsymbol{U}_o \boldsymbol{h}_{t-1} + \boldsymbol{b}_o) \\ \boldsymbol{c}_t &= \boldsymbol{f}_t \odot \boldsymbol{c}_{t-1} + \boldsymbol{i}_t \odot \tanh(\boldsymbol{W}_g \boldsymbol{x}_t + \boldsymbol{U}_g \boldsymbol{h}_{t-1} + \boldsymbol{b}_g) \\ \boldsymbol{h}_t &= \boldsymbol{o}_t \odot \tanh(\boldsymbol{c}_t) \end{aligned} \tag{2.34}$$

where $\boldsymbol{x}_t$ is the input at time t, $\boldsymbol{h}_{t-1}$ is the output of the cell at $t-1$ and the symbol $\odot$ denotes the element-wise multiplication between the vectors.

2.3.5 Convolutional Neural Networks

Convolutional Neural Networks (CNNs) are a type of feedforward networks that process structured data using the convolutional operation. Although, drawing analogies between the function of artificial

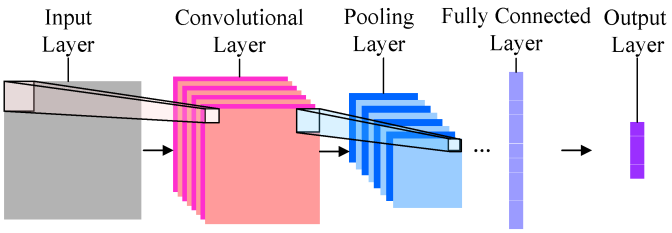


Figure 2.14: Full architecture of a CNN taken from [56]

and animal neural functions can be spurious, CNNs are a standout example of machine learning driven by neuroscientific insights. The first CNNs were inspired by the visual cortex of animals: Each of the cortial neurons responds only to stimuli coming from a restricted region, namely its *receptive field*. There are two types of cortial neuron cells, ones that detect basic shapes with a particular within a small receptive field and ones that have larger receptive fields and more spatially invariant responses. Consequently, CNNs were initially applied mostly to computer vision tasks. However, in recent years they have demonstrated competitive results in additional domains, such as speech and text. In general, CNNs can process any data that exhibit a clear grid-structure. For instance, images are typically encoded as 3-dimensional volumes of a red, a green and a blue channel (RGB encoding). Sound can be represented visually as a spectrogram and text can be viewed as a set of consecutive n-grams (sequences of n contiguous words).

A layer is characterized as convolutional, if its weights are applied to the input using the operation of convolution instead of matrix multiplication. From a mathematical viewpoint, a convolution of two functions is the integral transform that measures how much one function modifies the other, as the first one passes over the latter. Given an input function x and a weighting function w the convolution $x * w$ is defined as:

$$s(t) = x(t) * w(t) = \int x(h)w(t-h)dh \tag{2.35}$$

The weighting function is often called a *filter* or *kernel*. The output of the convolution is also referred to as a *filter/feature map*. As we have seen above, the output of a fully-connected layer with input $\boldsymbol{x}$ and weights $\boldsymbol{w}$ can be expressed as $\boldsymbol{w}^T\boldsymbol{x}$. CNNs, conversely, perform discrete convolutions of the inputs and weights. In practice, most DL libraries do not flip the kernel in favor of simplicity. The implemented function is the cross-correlation:

$$s(t) = x(t) * w(t) = \sum_h x(t+h)w(h) \tag{2.36}$$

The size of the kernels is usually much smaller than the input. This allows CNNs to learn hierarchies of patterns. The first layers of CNNs extract simple, local patterns that are gradually assembled in more abstract and complex ones moving deeper into the network.

Another key difference between fully-connected and convolutional layers is that while fully-connected layers can be broken down into two stages, the affine transformation and the activation function, convolutional layers are composed of the following three:

- Convolution stage: The first step is to apply conlvolution between the input and a set of kernels.
- Detector stage: The second step involves the use of a nonlinear activation function. A common nonlinear function used at this stage is the ReLU 2.19.
- Pooling stage: Finally, in the pooling stage the features are downsampled. Each feature in the output feature map corresponds to a summary statistic of a neighborhood of features.

The *pooling* operation is integral to the conception of CNNs as it enables successive layers to process representations of increasing fractions of the original input. In addition, pooling layers reduce the size

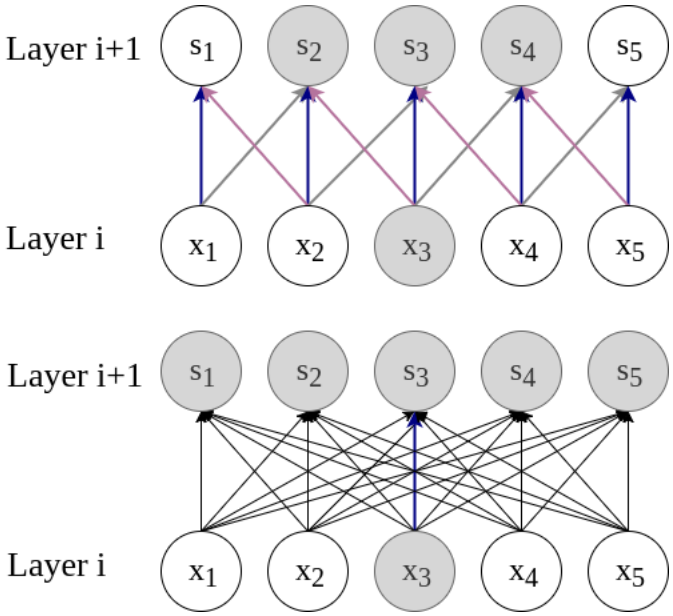


Figure 2.15: Sparse connectivity and parameter sharing in convolutional layers (top) in comparison to fully connected layers (bottom). The shading of the units is indicative of the connectivity of the units. Unit x_3 is shaded along with the units of layer $i + 1$ that are affected by the value of x_3. The bold edge is an example of parameter sharing in CNNs. In fully connected layers, each value is only used once. Figure adapted from [24]

of the output which also helps to reduce the number of model parameters. This leads to an increase of the computational efficiency and prevents overfitting. The pooling process is somehow similar to the convolution described before in terms of operating on feature neighborhoods, but instead of using a learned filter, it uses a hardcoded function. Two of the most common pooling operations are max- and average-pooling. As the names suggest, max-pooling selects the feature with the maximum value within a region, while average-pooling computes the average of a neighborhood of features. In practice, max-pooling have been found to perform better.

Pooling layers achieve invariance to small transformations of the input. If we pool over regions of the feature maps, the learned representations become translation invariant. That means that if we apply a small shift to the input, the output of the pooling will be approximately the same. This is very helpful for cases, where it is more important to know if a feature exists rather than its exact location. If we pool over features from separate feature maps, the learned representations can become invariant to transformations of the input. For example, max-pooling over image features helps grants invariance to rotations [24].

CNN display three important properties: sparse connectivity, parameter sharing and equivariant representations.

Sparse Connectivity In fully-connected layers, each output is computes as a function of all the input values. However, this leads to the number of parameters quickly blowing up when the input is very high-dimensional. CNNs overcome this problem through sparse interactions. That means that each output neuron will only be connected to a local region of the input neurons (see Fig. 2.15). The extent of the connectivity is controlled by the size of the kernel, which is a hyperparameter often referred to as the *receptive field* of the neuron. Sparse connectivity constraints the number of model parameters, which reduces memory requirements as well as the risk of overfitting.

Parameter Sharing Another benefit of using the operation of convolutions it that it enforces parameter sharing. It is motivated by the assumption that it would be desirable to extract similar features along different input regions. The kernel can be thought as a filter sliding across the input, so that each element of the kernel is applied to each element of the input. Figure 2.15 visualizes the property of parameter sharing present in convolutional layers (top graph), where the bold edge represents a

weight being used at each input node. Parameter sharing, also called *weight tying* is a technique as it significantly limits the number of model parameters.

Equivariant Representations Another effect of parameter sharing in CNNs is that the learned representations are equivariant to translation. A function f is equivariant to a function g if $f(g(x)) = g(f(x))$. That means that applying a convolutional layer to a translated input, will yield the same representation as translating the output of the convolution. This property explains why a kernel detects a particular pattern across the input. In order to detect multiple features at each layer we do not use just one kernel but a set of them.

2.3.6 Attention Mechanisms

In recent years, attention has been established as a valuable tool for deep learning models. Originally, attention was used in Neural Machine Translation to dynamically attend over the representation of the input sequence and predict the next word in the translated sequence [8]. Since then attention mechanisms have been applied to multiple tasks and domains yielding state-of-the-art results. This success motivated the design of the Transformer [69], a type of network for processing sequences based entirely on self-attention.

Attention can be interpreted as focusing on the most relevant elements of the input by computing weights of importance for their representations. For instance, these elements could be regions in an image or words in a sentence. The importance of each element is typically measured by their alignment with a query vector q.

Formally, given a set of N input vectors $\boldsymbol{x}_1, \ldots, \boldsymbol{x}_n$ and a query vector $\boldsymbol{q}$ the attention module uses an alignment function to score the relevance of each $\boldsymbol{x}_i$ to the query $\boldsymbol{q}$. The alignment scores s_i are then normalized using the softmax function to produce attention weights $\alpha_1, \ldots, \alpha_n$ that sum to 1. The final representation $\hat{\boldsymbol{x}}$ is represented as the average of the inputs $\boldsymbol{x}_i$ weighted by α_i.

$$s_i = \text{align}(\boldsymbol{q}, \boldsymbol{x}_i) \tag{2.37}$$

$$a_i = \text{softmax}(s_i) = \frac{e^{s_i}}{\sum_j e^{s_j}} \tag{2.38}$$

$$\hat{\boldsymbol{x}} = \sum_i a_i \boldsymbol{x}_i \tag{2.39}$$

The score s_i for each vector x_i is a scalar computed with an alignment score function. Table 2.1 summarizes a collection of popular alignment score functions.

Name	Alignment Score Function
General [50]	$\boldsymbol{s}^\top \boldsymbol{W_a} \boldsymbol{x}_i$
Additive [8]	$\mathbf{v}_a^\top \tanh(\boldsymbol{W}_a[\boldsymbol{s}; \boldsymbol{x}_i])$
Dot-Product [50]	$\boldsymbol{s}^\top \boldsymbol{x}_i$
Scaled dot-product [69]	$\frac{\boldsymbol{s}^\top \boldsymbol{x}_i}{\sqrt{n}}$
Content-based [28]	$\text{cosine}(\boldsymbol{s}, \boldsymbol{x}_i)$

Table 2.1: Types of alignment score functions that take as input the representations $\boldsymbol{x}_i \in \mathbb{R}^n$ and the query $\boldsymbol{q} \in \mathbb{R}^d$.

Chapter 3

Multimodal Learning

Visual question answering is the task of predicting an answer to a natural-language question regarding an image. To tackle this task, the first step is to extract representations from both the textual and visual inputs. In this section, we will present the methods used to represent textual and visual information in the scope of this thesis and we will introduce the concept of sequence-to-sequence modeling.

3.1 Natural Language Processing

Natural language processing (NLP) is a subfield of AI, that specializes in automating the analysis, understanding and generation of natural -i.e human- language. The objective of NLP is to enable the interaction between humans and machines in natural language. However, this has been proven to be a rather demanding goal, as human language is abstract, complex and highly diverse. The meaning of a sentence can vary according to the interpretation of ambiguous words, syntax or tone. For example, the word "python" can refer either the reptile or the programming language [13], while the phrase "That's just what I needed today!" can express either a negative or positive sentiment depending on whether or not it is used sarcastically. In recent years, NLP has seen impressive progress across all core tasks, but especially considering the learning of word representations. This progress has been marked by the replacement of high-dimensional sparse vectors with low-dimensional distributed representations.

Word Embeddings

The simplest way to represent a word as a vector would be to encode it by its index in the vocabulary. This can be expressed as a vector with dimension equal to the vocabulary size which is marked with a '1' at the position corresponding to the word index and is filled with '0's everywhere else. This is called a *one-hot encoding*. One-hot encodings have several disadvantages. First of all, the size of these vectors scales linearly with the size of the vocabulary. This is associated with a problem known as the *curse of dimensionality* [9], which refers to overfitting occurring when the feature space becomes sparse. Moreover, one-hot vectors encode no information about the similarity of words. For instance, given that one-hot vectors represent words as independent unit vectors, the word "dog" would be considered equally dissimilar to the words "cat" and "car". Nowadays, these issues have been resolved by the use of *word embeddings*, which map words to vectors in a low-dimensional, continuous space. This allows words the semantic relationship of words to be captured by their relative positioning in the embedding space.

Most models for learning word embeddings are motivated by the *Distributional Hypothesis* [21], which states that words that appear in similar contexts have a similar semantic meaning. The context of a word typically refers to the set of words that occur within a window around it. Learning word embeddings is typically posed as the unsupervised task of predicting a word based on its context. So for example, one expects to find the words "dog" and "cat" in similar contexts as they both refer to four-legged, popular pets. However, the words "dog" and "car" would co-occur significantly more rarely in the same context. As a result, the embeddings of "dog" and "cat" will be closer to each other than to the embedding of "car".

The Word2Vec model was proposed in 2013 by Mikolov et al. [53] and it has since become one of the most widely used models for learning word embeddings. It is a computationally efficient method

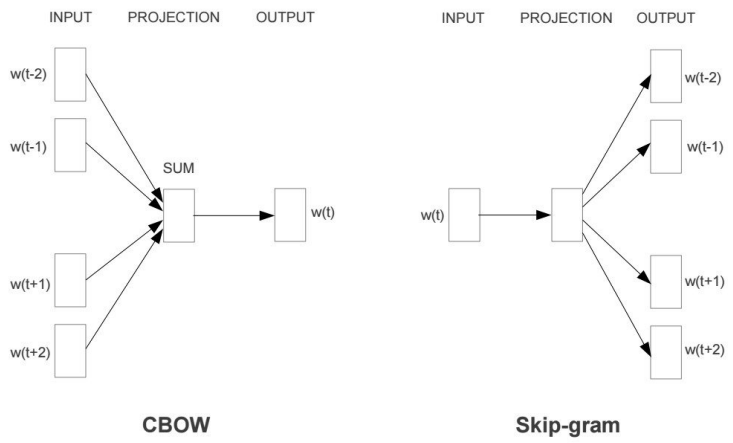


Figure 3.1: The two training models of Word2Vec. Figure from [53]

based on training a shallow neural network. It come in two versions: the Continuous Bag-of-Words model (CBOW) and the Skip-Gram model. The CBOW model learns to predict the current word from its context, while the Skip-Gram model learn to predict context words from the current word. The context used during training is determined by the size of the window. Large windows tend to capture more semantic similarities, while smaller windows tend to capture more syntactic similarities.

Word2Vec embeddings have been found to preserve semantic relationships which allow basic vector arithmetic. For example, the vector operation $king - man + woman$ yields an embedding close to the embedding of $queen$. However, one limitation of Word2Vec is that it takes into account co-occurencies only within a window around each word. Global Vectors for Word Representation (GloVe) [58] overcomes this problem by combining global co-occurrence statistics and local context window methods. GloVe computes a co-occurrence counts matrix and then learns a factorization of this matrix such that the low-dimensional representations will preserve linear relations between words just like Word2Vec.

Word embeddings are trained on large, unannotated corpora and can provide meaningful and dense representations of words. It is a common practice to use pretrained word embeddings instead of one-hot vectors as inputs for downstream NLP tasks. This helps reduce the number of trainable parameters, which is especially important for supervised tasks where only a small number of annotated data are available. If the amount of available data is large enough, one can use the pretrained embeddings to initialize the input layer and allow it to be fine-tuned for a specific task during training.

Sentence Representations

The Bag of Words (BOW) model is arguably the simplest algorithm for computing a sentence representation. In BOW, a sentence is represented as an aggregation -typically the average- of its words disregarding completely the order of the words within the sentence. CBOW is a variation of this model, where words are represented by their embeddings. Although very simplistic, CBOW is a strong baseline for many text classification tasks.

Following the success of word embedding models, there have been efforts to train models that encode entire sentences in fixed-size, low-dimensional vectors. The motivation is that these sentence embeddings will be robust enough to be utilized as off-the-shelf text features. Skip-thought [42] vectors have been an attempt at this direction. The Skip-thought model emulates the Skip-gram Word2Vec model. However, instead of predicting the context words, Skip-thoughts predict the surrounding sentences of the current one.

Nonetheless, the standard approach for most NLP tasks is to train a recurrent neural network end-to-end. Suppose we have a sentence $w_1, \dots w_T$ that we want to classify. At time step t the input

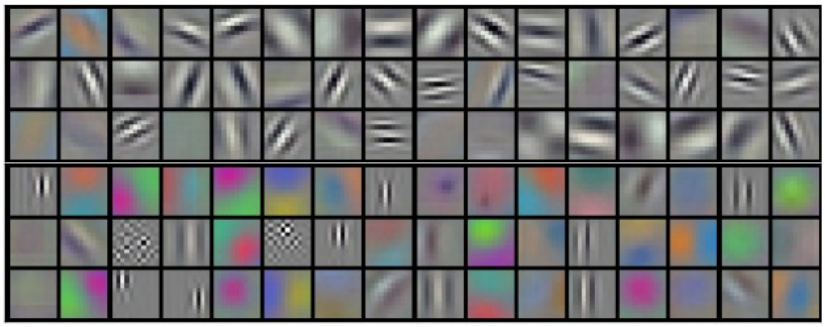


Figure 3.2: Visualization of 96 kernels of size $11 \times 11 \times 3$ from the first convolutional layer [44]. The first layer extracts low-level information about edges of different orientations and colors.

word w_t passes through the embedding layer and the word embedding e_t is then fed to the RNN. The sentence is represented by the hidden state of the RNN at the last time step of the sequence. Finally, the sentence representation is fed to a classifier implemented as a series of fully-connected layers. In order to encode both future and past information at each time step, one can use a bidirectional rnn and combine the hidden states through concatenation. Moreover, one can take the average or the max of the hidden states at each time step, or add a self-attention layer on top of the RNN.

3.2 Convolutional Neural Networks for Images

Images are usually represented as 3-dimensional volumes $H \times W \times C$, where H is the image height, W is the width and C is the number of channels. RBG images have three channels defining the red, green, and blue color components of each individual pixel. As one can expect, these representations are very high dimensional. Some of the most widely used CNN architectures accept images of size 224×224. That means that if we were to process an image of this size using a fully-connected network, the number of weights for a single neuron in the input layer would be $224 * 224 * 3 = 150.528$. This explains why the properties of sparse connectivity and parameter sharing of CNNs are imperative for Computer Vision tasks.

Transfer learning has been applied with remarkable success to computer vision problems. Training deep CNNs from scratch requires a lot of resources and it can take up to several weeks. As a result, people tend to make use of networks pretrained on a large dataset. Two common pretraining tasks are image classification and object detection. Although similar, these two tasks are distinct. *Image classification* aims at determining correctly the label of an image, while *object detection* aims at identifying the labels as well as the location of multiple objects in an image. Pretrained CNNs are used in one of the following scenarios:

- Fixed feature extraction: The image features are extracted by taking the output of an intermediate layer of the pretrained CNN. In order to obtain a global image representation, it is sufficient to remove the output layer that produces scores for candidate labels and use the rest of the CNN as the feature extractor. Taking the output of earlier layers results in a feature map, i.e. a set of features representing regions of the input image.

- Fine-tuning: If the dataset of the target task is large or the data differ substantially from the original ones, it is beneficial to fine-tune the pretrained model. In this scenario, the weights of the pretrained model are used as an initialization. During training, these weights are usually updated using a smaller learning rate than randomly initialized ones.

As we described in Section 2.3.5, the feature extraction part of CNNs is composed of stacked convolutional, activation and pooling layers that transform one volume of activations to another. CNNs are thus able to learn hierarchies of patterns in the data. Kernels detect simple and generic features

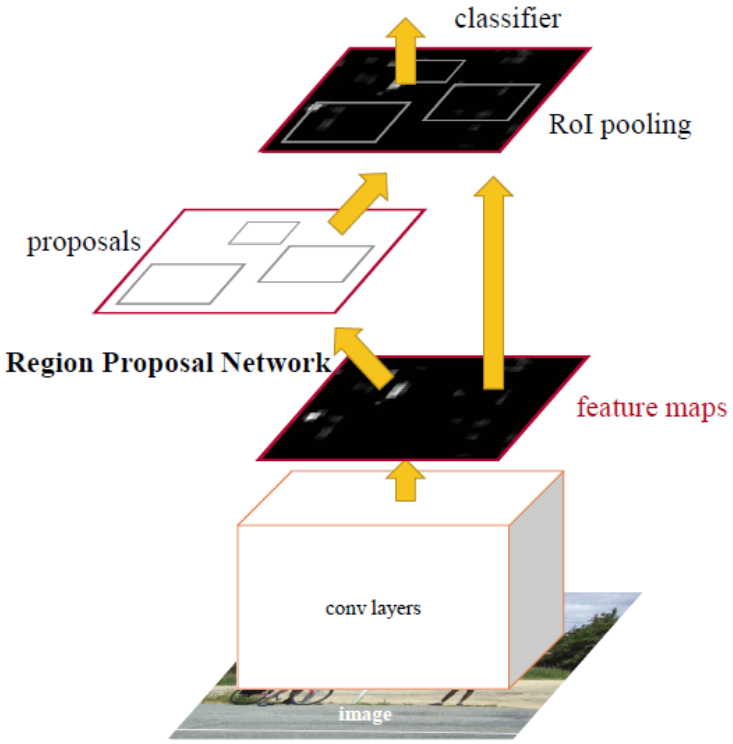


Figure 3.3: Overview of the Faster R-CNN architecture. Image from [60]

in the earlier layers, which get more and more complex as the layers get deeper. Figure 3.2 shows an example of kernel weights if the first layer of a CNN trained for object classification [44]. We can see that the first layer recognizes low-level features such as edges and colors.

CNN architectures for object detection differ slightly from regular CNNs in order to account for the localization of objects in the image. An object detection system must output a tuple for each detected object: the class scores and the bounding box coordinates. Since each image contains an arbitrary number of objects, most models determine a group of region proposals and then use a classifier to compute the probability of an object being present in that region. The regions proposals are generated by applying multiple anchor boxes at different positions of the image. Anchor boxes are basically bounding boxes of different shapes and sizes. One of the most efficient models for object detection is the Faster R-CNN [60], . The steps of this model are the following:

- An input image is passed through a convolutional layers to obtain the feature map of the image.
- A Region Proposal Network (RPN) is applied on the feature map. The RPN uses the features of each anchor box to compute an "objectness" score that expresses the likelihood of an object being present in the region specified by the anchor.
- The regions proposed by the RPN have different sizes. As a result, the feature maps corresponding to these regions also have varying sizes. To obtain representations of a fix size for all regions, a Region of Interest (RoI) pooling layer is applied to the feature maps. Unlike a Max-Pooling layer, ROI pooling splits the feature map into a fixed number of regions, and keeps the maximum value in every region. Therefore, the size of the output is the same regardless of the size of input.
- Finally, the features of the proposals are passed to a classifier that outputs class label probabilities and to a linear regression layer that refines the coordinates of the bounding box.

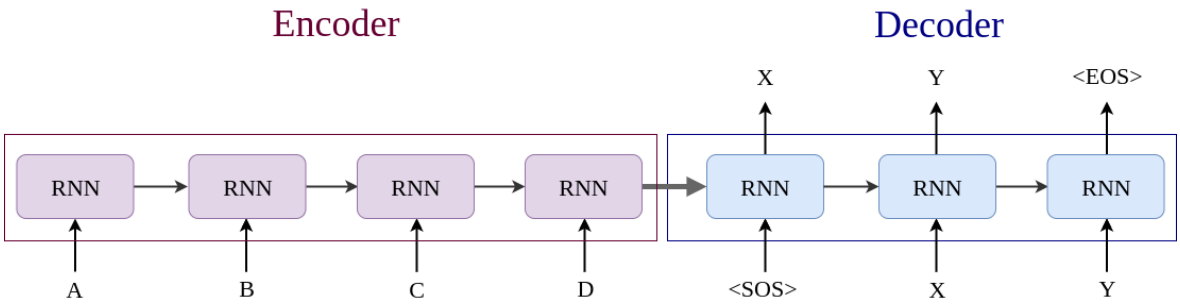


Figure 3.4: The sequence-to-sequence architecture. The model takes as input the sequence "ABC" and generates the sequence "XY". The first input to the decoder is the start-of-sequence token <*SOS*> and the generation halts after outputting the end-of-sequence token <*EOS*>. Image adapted from [65]

3.3 Sequence-to-Sequence Models

Sequence-to-sequence (seq2seq) learning is a framework for mapping input to output sequences of different lengths. The challenge of a target sequence not having the same length as the inputs particularly common in Machine Translation. For example, the sentence "How are you doing?" in English is translated to "πώς είσαι?" in Greek. The English sequence is made up of 4 words, while the Greek one is made up of just 2. It is obvious that there does not exist a one-to-one mapping from the sequence of one language to another. It is thus impossible to use a regular RNN to translate each word from the source language to a word in the target language. This motivated Sutskever et al. [65] to develop the seq2seq neural architecture. Since then seq2seq models have been applied to multiple tasks such as speech recognition [15] and video captioning [70].

The general seq2seq approach is straightforward. The network has two parts – an encoder and a decoder (see Fig. 3.4). The encoder maps the input sequence to a fixed dimensional vector c which is utilized by the decoder this vector to generate the output. Formally, the task of generating the output $y_1, \dots y_{T'}$ sequence given the input sequence $x_1, \dots x_{T'}$ can be expressed as:

$$P(y_1, \dots y_{T'} | x_1, \dots x_T) = \prod_{t=1}^{T'} P(y_t | c, y_1, \dots y_{t-1}) \tag{3.1}$$

Since both the encoder and the decoder process sequences, they are usually implemented using RNNs. In more detail, the seq2seq approach involves the following stages:

- **Encoder**: The input sequence $x_1, \dots x_T$ is processed by the encoder RNN. The last hidden state h_T^e serves as the fixed dimensional representation of the input (i.e. $c = h_T^e$) and it is used to initialize the hidden state of the decoder RNN. This vector is often referred to as the "*context*" or "*conditioning*" of the decoder, because its role is to summarize the information of the input based on which the decoder will generate the output.

- **Decoder**: The hidden state of the decoder is initialized with the context vector c. Following that, the decoder takes as input a start-of-sequence token <*SOS*> and starts generating words one by one. At each time step t, the next word y_t of the output sequence is predicted given the last predicted word and the decoder hidden state. The decoder continues to predict words until an end of sequence token <*EOS*> is predicted.

A shortcoming of this approach is that the information of the input is accessed only through the context vector. Trying to encode variable-length sequence into a fixed-size representation creates a kind of information bottleneck. A solution to this issue has been provided by the use of an attention mechanism, that dynamically combines the outputs of the encoder for each input word during the decoding stage. That means that at each time step the decoder has access to all encoder states and can

focus on the most relevant parts of the input. Finding a solution to this problem was the final advance that made neural MT competitive with previous approaches.

Training a seq2sec model a few particularities that distinguish it from a regular RNN. One of them concerns the formulation of the loss function. As we have mentioned before, cross-entropy is a popular loss function for classification problems. In practice, sequence generation is implemented as performing a series of classifications for each input. As a result, the loss J of a model with parameters $\boldsymbol{w}$ for a single data sample is computed as the sum of the cross-entropy losses between the target $\bar{y}_t$ and the predicted y_t words:

$$J(\boldsymbol{w}) = -log \prod_{t=1}^{T'} P(\bar{y}_t = y_t | c, y_1, \dots y_{t-1}) = -\sum_{t=1}^{T'} log P(\bar{y}s_t = y_t | c, y_1, \dots y_{t-1}) \tag{3.2}$$

Another useful technique for training seq2seq models is *teacher forcing*. Using incorrectly predicted words as inputs of the decoder can cause instability during training. Instead, teacher forcing leads to faster convergence by forcing the decoder takes the ground truth as input at each time step.

Chapter 4

Sequence-to-Sequence Modeling for Visual Question Answering

4.1 Motivation

A VQA system takes as input an image and a free-form, natural-language question regarding the image and outputs a natural-language answer. Generally, the questions are open-ended, meaning they cannot be answered with just "yes" or "no". In this thesis, we focus on free-form answer generation. Since the thematic of visual questions is unconstrained, their level of complexity varies widely and answering can require diverse types of reasoning. To illustrate that point, let's consider a few examples. Visual questions require object detection capabilities to be able to locate the objects referenced in the question. In addition to that, answering the question "What color of shirt is the man on the left wearing?" involves spatial reasoning to identify the man and attribute classification to recognize the color. To respond to questions similar to "How many people are in the image?" calls for solving a visual counting problem. Lastly, the inquiry "What is the model of this vehicle?" requires commonsense knowledge that cannot be extracted directly from the image. Creating a VQA system that can answer arbitrary questions is understandably very challenging.

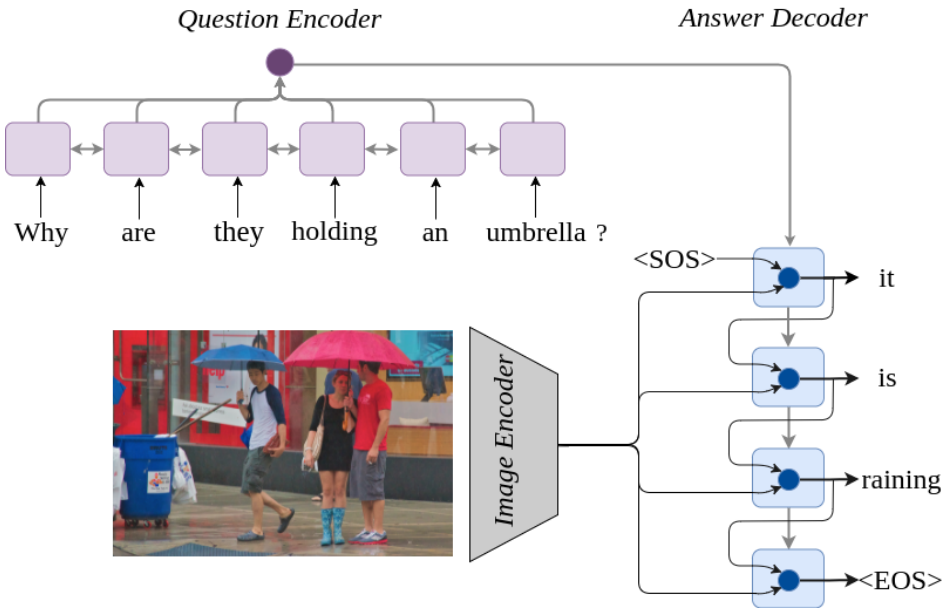


Figure 4.1: Overview of the proposed model.

Most current approaches formulate VQA as a classification task. Although the ground truths are not usually complete sentences, they range from single words to short phrases. The set of possible classes is selected as the answers that occur most frequently in the training data. This implies that answers that are indeed short phrases are treated as separate classes from the words that they comprise. For instance, under this setting answers "black and white", "black" and "white" are viewed as

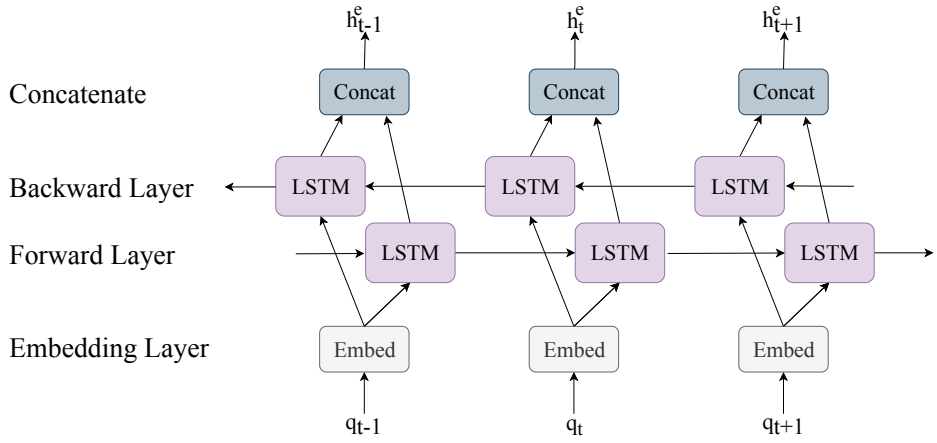


Figure 4.2: BiLSTM of the Question Encoder.

completely independent classes. This contradicts our intuition that the meaning of a phrase emerges from the syntax and the semantics of individual words. An object that is "black and white" shares a common attribute with both black and white objects. As a result, we hypothesize that learning to recognize the color of a "black and white" object can benefit from learning the concepts of the two individual colors from other samples in the dataset. Moreover, predicting answers at a phrase-level limits the expressiveness of proposed models to closed-world settings. The space of possible answers is strictly defined in advance and no novel answers can be composed from the existing model knowledge. For instance, in the case of answering questions about the color of objects, every combination of colors that has not been added to the set of possible classes will be prohibited. Consequently, aiming at a high coverage of answers under the classification setting leads to a fast expansion of the number of possible answers.

Motivated by these observations, we formulate the VQA problem as a sequence generation task. The answer is generated by a seq2seq model (Sec. 3.3), conditioned on both the question and the image representation. We use Faster RCNN features (Sec. 3.2) to extract the image representation and an LSTM (Sec. 2.3.4) to encode the question. The answer is generated by a decoder LSTM. Our system performs grounded answer generation by attending to the image features at each decoding step. We show empirically in our ablation study that grounding has a large impact on model performance. Our results are competitive with the state-of-the-art, especially for questions that are not closed-form. This showcases the feasibility of free-form answer generation for open-ended VQA. In the following section we will describe in detail the modules of the grounded seq2seq model for VQA.

4.2 Model Description

Figure 4.1 depicts an abstract scheme of the proposed grounded seq2seq architecture. The overall system is made up of three main modules: the question encoder, the image encoder and the answer decoder. Given an input pair of a question Q and an image I, the system generates an answer Y as following:

- The question $Q = [q_1, \dots q_N]$ is encoded using a single-layer, bi-directional LSTM (BiLSTM) encoder (see Fig 4.2). Initially, the words q_t in the question sequence with length N are represented as one-hot vectors, where the index of the "1" corresponds to the index of the word in the input vocabulary V_q. The one-hot vector of each word is passed through an embedding layer, that has been initialized with GloVe embeddings [57] and is fine-tuned during training. The resulting embeddings are then fed to the BiLSTM, which produces a new hidden state for each direction.

  The hidden states from the forward $\vec{\boldsymbol{h}}^e_t$ and the backward pass $\overleftarrow{\boldsymbol{h}}^e_t$ are concatenated to obtain the

BiLSTM output $\boldsymbol{h}_t^e$ at each time-step t :

$$\boldsymbol{h}_t^e = [\vec{\boldsymbol{h}}_t^e; \overleftarrow{\boldsymbol{h}}_t^e] \quad \text{for } t \in 1 \dots N \tag{4.1}$$

Suppose that $\vec{\boldsymbol{h}}_t^e$ and $\overleftarrow{\boldsymbol{h}}_t^e$ are k-dimensional vectors. The combined state $\boldsymbol{h}_t^e$ will then be a d-dimensional vector, where $d = 2k$. We stack the states $\boldsymbol{h}_t^e$ for $t[\in 1, \dots N]$ in order to obtain the matrix $\boldsymbol{H} \in \mathbf{R}^{N \times d}$.

The role of the question encoder is to map variable length input sequences to a fixed-size representations. The question representation $\boldsymbol{e}$ is computed as the convex sum of $\boldsymbol{h}_t^e$ using weights a_t^e, that are learned from a self-attention mechanism [47]:

$$\boldsymbol{a}^e = \text{softmax}(\boldsymbol{w}_{a2}\text{tanh}(\boldsymbol{W}_{a1}\boldsymbol{H}^T)) \tag{4.2}$$

$$\boldsymbol{e} = \sum_{t=1}^{N} a_t^e \boldsymbol{h}_t^e \tag{4.3}$$

where $\boldsymbol{W}_{a1} \in \mathbf{R}^{d \times d}$ and $\boldsymbol{w}_{a2} \in \mathbf{R}^{1 \times d}$ are the self-attention parameters. These weights a_t^e signify the contribution of each hidden state $\boldsymbol{h}_t^e$ to the final question representation $\boldsymbol{e}$.

- The image I is represented as a set of R feature vectors $\{\boldsymbol{i}_1, ...\boldsymbol{i}_R\}$ extracted from a pretrained object detection network. Specifically, we use features extracted from a Faster RCNN network trained on the Visual Genome dataset [43], which have been made publicly available by Anderson et al. [5]. These features encode bounding-box regions in an image, thus providing localized information about objects. In essence, representing each image in the dataset as a collection of object features acts as a form of bottom-up attention that allows us to encode only salient image regions.

  A Faster RCNN network can output an arbitrary number of bounding boxes for each image. Following [5], we keep for all images the 36 bounding boxes with the highest confidence. In brief, an input image I is represented by a feature map $R = 36$ feature vectors $\boldsymbol{i}_r \in \mathbf{R}^{2048}$ corresponding to salient regions in the image.

  We apply $L2$ normalization to the input feature vectors I so that all the inputs are at a comparable range and pass the normalized features through a fully-connected layer with a tanh non-linearity to obtain the visual representation $V \in \mathbf{R}^{R \times d}$:

$$\boldsymbol{V} = \text{tanh}(\frac{\boldsymbol{I}}{\|\boldsymbol{I}\|}\boldsymbol{W}_v) \tag{4.4}$$

  where $\boldsymbol{W}_v \in \mathbf{R}^{2048 \times d}$ is the matrix that projects the image features to the dimension of the question representation.

- The answer $Y = [y_1, ...y_L]$ is generated by a single-layer, uni-directional LSTM with size equal to d. The decoder LSTM is conditioned on the question by initializing its hidden state with question representation $\boldsymbol{e}$:

$$\boldsymbol{h}_0^d = \boldsymbol{e} \tag{4.5}$$

  The answer generation follows a greedy decoding process. At each time step $t \in [1, ...L]$, the decoder takes as input the previous word y_{t-1} and updates its states. The current hidden state $\boldsymbol{h}_t^d$ is used to attend to the image features $\boldsymbol{V}$. This produces the visual context vector $\bar{\boldsymbol{v}}_t$ as described in Eq. 4.8. The motivation behind applying spatial attention at each decoding step is to allow the model to attend to different regions in order to predict the next word. We combine the visual context vector $\bar{\boldsymbol{v}}_t$ and the hidden state $\boldsymbol{h}_t^d$ using the Hadamard product $\circ$ to obtain the fused vector $\boldsymbol{f}_t$:

$$\boldsymbol{f}_t = \bar{\boldsymbol{v}}_t \circ \boldsymbol{h}_t^d \tag{4.6}$$

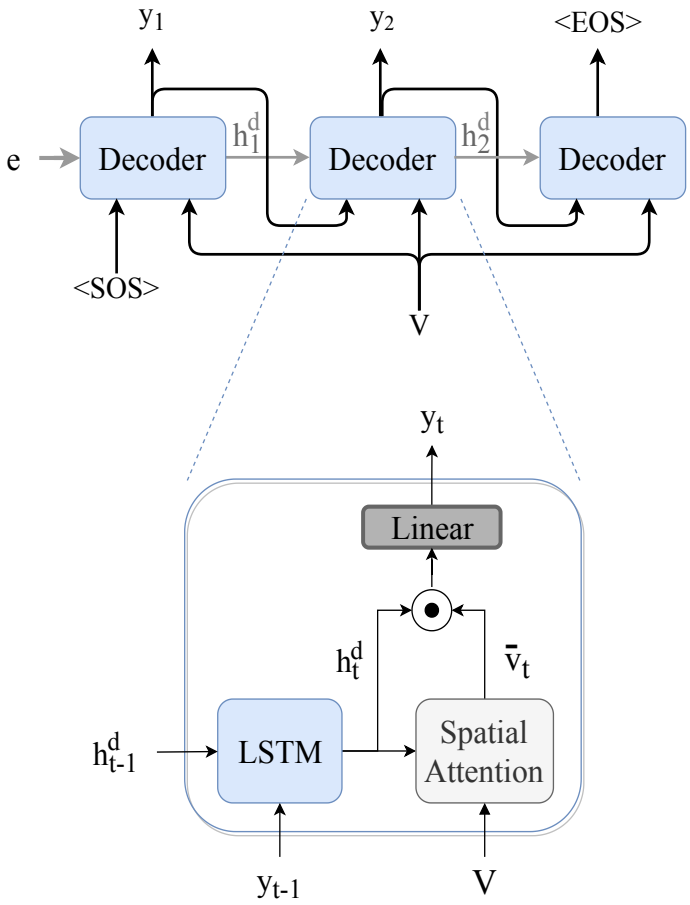


Figure 4.3: Answer decoder.

The next word y_t is computed as the word with the highest probability from the conditional probability distribution over the answer vocabulary V_a:

$$P(y_t|\boldsymbol{e}, y_1, ...y_{t-1}, \boldsymbol{f}_t) = \text{softmax}(\boldsymbol{W}_d \boldsymbol{f}_t) \tag{4.7}$$

where $\boldsymbol{W}_d \in \mathbf{R}^{|V_a| \times d}$ is a learned weight matrix. The decoding halts when the <*EOS*> token is predicted. In practice, we merge the questions V_q and answers V_a vocabularies into a common vocabulary V and use the same embedding layer for the question encoder and the answer decoder in order to learn a joint textual representations of the words in V.

Spatial Attention Mechanism

We will now go into further detail about the spatial attention mechanism, that is used to attend over the image features during the decoding. Fig. 4.4 illustrates an overview of the spatial attention mechanism.

At each time step t, the image is summarized into a single d-dimensional vector $\bar{\boldsymbol{v}}_t$ using a spatial attention mechanism, that allows the network to focus on the most relevant parts of the image. The attention mechanism we utilize follows the definition of the scaled dot-product attention introduced in [68]. The relevance of each feature vector is determined in relation to a *query vector*, which in our case is guided by the decoder hidden state $\boldsymbol{h}_t^d \in \mathbf{R}^d$.

In order to compute the attended image representation $\bar{\boldsymbol{v}}_t$, we first take two linear projections $\boldsymbol{V}_1$ and $\boldsymbol{V}_2$ of the visual representation $\boldsymbol{V}$. $\boldsymbol{V}_1$ is used to compute the affinity between each region feature vector and the query vector $\boldsymbol{h}_t^d$. The affinity scores are converted to attention weights $\boldsymbol{a}_t^d$ by going through a softmax layer. The attention weights $\boldsymbol{a}_t^d$ are lastly applied to the image features V_2 producing

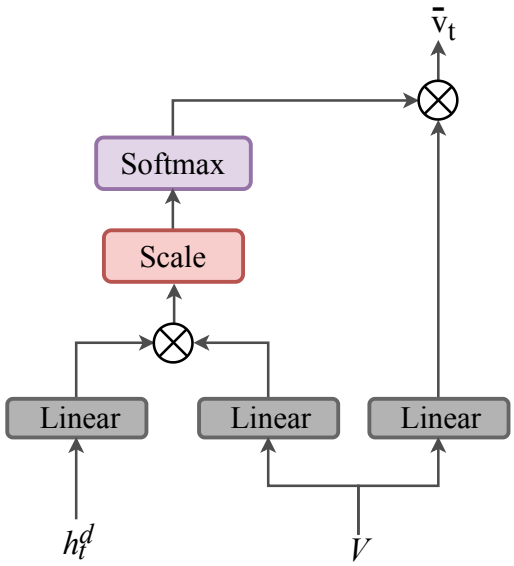


Figure 4.4: Overview of the spatial attention mechanism.

the representation $\bar{v}_t$ as follows:

$$\boldsymbol{a}_t^d = \text{softmax}(\frac{\boldsymbol{h}_t^d \boldsymbol{V}_1^T}{\sqrt{d}}) \tag{4.8}$$

$$\bar{\boldsymbol{v}} = \boldsymbol{a}_t^d \boldsymbol{V}_2 \tag{4.9}$$

The factor $1/\sqrt{d}$ is used to scale dot product values in order to prevent vanishing softmax gradients. As mentioned above, by attending over the image regions at each step, the decoder has the opportunity to adjust the attention weights depending on the previous words as well as to avoid long-term dependencies to the visual input.

Chapter 5

Experimental Setup, Evaluation and Results

In this chapter, we describe the experimental procedure and present our evaluation results. First, we provide some information about the dataset and previous related work on the task. Then, we specify the chosen parameters of the proposed Grounded Seq2Seq model and continue with the evaluation. We do so by comparing it to state-of-the-art models and examining the results of an ablation study done to investigate the contribution of each components to the model performance. Finally, we present visualization examples of our model.

5.1 Dataset

We evaluate our model on the second version of the *Visual Question Answering under Changing Priors* (VQA-CP v2) dataset [3]. This dataset constitutes a variant of the VQA v2 dataset [27] that is currently the most widely used dataset for VQA. The VQA v2 dataset consists of 443K train, 214K validation and 453K test samples of image-question pairs. Each image-question pair is accompanied with 10 answers, that have been collected from 10 individual human annotators. The dataset contains multiple questions for every single image, with the total number of different images adding up to 200K.

A reasonable concern regarding VQA models is whether answer prediction is actually grounded in the visual input. Recent work [27, 2, 14, 34] has repeatedly highlighted that language biases in VQA datasets allow models to guess the correct answer even when disregarding the visual input. This is attributed to language being a simpler signal for learning, which encourages the learning of incidental statistics in the questions and answers. As a result, a model achieving good performance is not necessarily addressing the VQA problem, which can pose a setback for real progress. The VQA v2 dataset attempts to suppress language priors by collecting for each question 2 complementary images that result into different answers. For example, for the question "What is the person doing?" there exist 2 samples in the dataset, one depicting a person surfing and one depicting a person skateboarding. Despite this effort, biases in the distribution of the questions or answers persist. As seen in Table 5.1, the language-only model still manages to answer correctly 43% of the test questions and remarkably to achieve 67.95% accuracy in the "Yes/No" category. The VQA-CP v2 variant was proposed as a remedy to this issue. It was created by reorganizing the splits of VQA v2 dataset so that the distribution of answers for each question type differs between training and evaluation sets. The samples from the train and the validation sets of the VQA v2 dataset are repartitioned in the train and test sets of the VQA-CP v2. The mismatch of the training and evaluation answer distributions really test how well the model can reason over the visual input to deduce the correct answer. As it can been seen in Table 5.1, VQA models show significant performance degradation when evaluated on VQA-CP v2.

Next, we enumerate some key details about the dataset:

- VQA-CP v2 *train* consists of 121K images, 438K questions and 4.4M answers, while VQA-CP v2 *test* consists of 98K images, 220K questions and 2.2M answers.
- The training questions have a vocabulary of size 14.5K, while the training answers have a vocabulary of size 37K. The joint vocabulary of the questions and the answers has a size of 40K words.

Method	Dataset	Accuracy			
		Overall	Yes/No	Number	Other
language-only [7]	VQA v2	43.01	67.95	30.97	27.20
	VQA-CP v2	15.95	35.09	11.63	07.11
language+image [7]	VQA v2	51.61	73.06	34.41	39.85
	VQA-CP v2	19.73	34.25	11.39	14.41
language+image+attention [73]	VQA v2	52.02	68.89	34.55	43.80
	VQA-CP v2	24.96	38.35	11.14	21.74

Table 5.1: Comparison of the performance of existing VQA models on VQA-CP test split to their performance on the original VQA validation split. [3]

- In addition to reporting the overall accuracy, it is common to break down model performance in 3 categories: "Yes/No", "Number" and "Other". As the names suggest, the "Yes/No" category contains all closed questions. "Number" samples refer to all questions answered with a number. Although the majority of "Number" questions are counting questions, there can also include questions of digit recognition (e.g. "What is the number on the bus?"). All remaining data samples are grouped under the term "Other". Concerning the percentages of each answer category, approximately 42% of the answers belong to "Yes/No", 12% belong to "Number" and 46% belong to "Other".
- Figure 5.1 depicts the length distributions of training questions and answers. The maximum question length is 25 words. Although the maximum answer length reaches 24 words, the majority of answers consist of just one word.

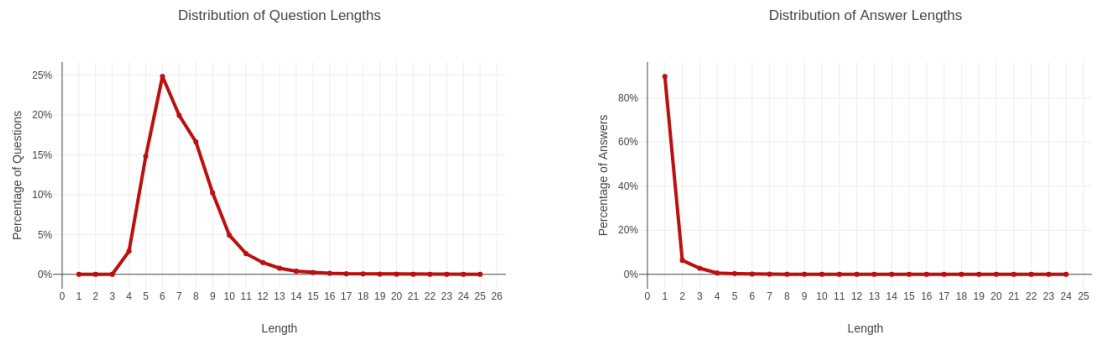


Figure 5.1: Distributions of sequence lengths in training questions and answers.

5.2 Related Work

Following the release of large scale VQA datasets [7, 27], many approaches to tackle the task have been proposed. Most methods follow a similar pipeline: Image representations are extracted from a pretrained CNN based model, while questions are encoded using RNNs. The image and question features are then combined using different attention and fusion mechanisms to improve cross-modal grounding. The joint representation is then fed to a classifier in order to predict the answer.

The introduction of attention mechanisms in the VQA pipeline led to a significant performance boost. Notable work on attention includes the Stacked Attention Network (SAN) [73] that utilizes two successive layers of spatial attention in order to extract more fine-grained information. Grounded VQA model (GVQA) [3] is a hybrid architecture built upon SAN that distinguishes between "Yes/No" and the rest of the questions. "Yes/No" questions are treated as a binary visual verification task,

while the remainders are processed by combining predicted visual concepts and candidate answers. In [49], question features are extracted on word, phrase and sentence level. These features are then used to attend over both the image and the question by use of an affinity matrix. Anderson et al. [5] apply spatial attention on top of a set of regional features extracted from a Faster RCNN. The spatial attention weights are calculated by projecting the concatenated image and question vectors from a gated tanh layer. More recently, Ramakrishnan et al. [59] introduce an adversarial training scheme to train a model with the same architecture as [5] that is , nonetheless, less dependent on language biases. Specifically, they train a language-only model as an adversary and adding an entropy term to the loss function that estimates the information gain after considering the image.

Fusion of image and question features is often performed with simple operations such as the Hadamard product or concatenation [7]. However, more intricate fusion approaches have also been investigated. These approaches focus on efficient approximations of the outer-product between image and question features using compact bilinear pooling [23], low rank tensor approximation [39] or Tucker decomposition [10]. Andreas et al. [6] propose dynamically assembling a network from a set of pretrained modules to adapt the attention and fusion scheme on the question input.

5.3 Experimental Setup

Model Hyperparameters

Table 5.2 summarizes the main hyperparameters of the Grounded Seq2Seq model. We use 300-dimensional GloVe embeddings to initialize the embedding layer with a vocabulary size of $10K$ words. We employ weight tying for the embedding layers of the encoder and the decoder in order to reduce the number of model parameters and enforce the semantic alignment of questions and answers. We use a BiLSTM with 256 units for the encoder, an LSTM with 512 units for the decoder and batch size 128. We set the maximum length of the questions equal to 23. Due to the short length of the answers in the dataset, we keep the maximum length of the predicted answers to 5.

Embeddings Initialization		300-d GloVe
Vocabulary Size		10K words
Question Encoder	LSTM size	512
	Max length	23
Answer Decoder	LSTM size	1024
	Max length	5

Table 5.2: Grounded Seq2Seq Parameters.

Training Setup

- During training we retain a third of the training data for validation and monitor the validation accuracy to apply early stopping with patience of 5 epochs,
- we apply dropout with probability 0.2 after each layer,
- we apply teacher forcing, meaning we feed the correct previous word to the answer generator,
- we use the Adam optimizer with learning rate 0.001 and
- we perform data augmentation for *'other'* answers: At each epoch, we select at random one of the ten human provided answers as the target for each sample. Thus, we present our model with paraphrased answers for each question, which acts as a form of regularization.

For the model implementation we used the PyTorch library [55]. The training process takes approximately 10 hours on a GeForce GTX TITAN X GPU.

Model	Overall	Yes/No	Number	Other
d-LSTM Q [7]	15.95	35.09	11.63	7.11
d-LSTM Q + I [48]	19.73	34.25	11.39	14.41
SAN [73]	24.96	38.35	11.14	21.74
GVQA [3]	31.30	57.99	13.68	22.14
MCB [23]	36.33	41.01	11.96	40.57
UpDn [5]	39.74	42.27	11.93	**46.05**
UpDn+Adv [59]	**41.17**	**65.49**	**15.48**	35.48
Grounded Seq2Seq	**36.42**	**40.29**	**12.07**	**41.08**

Table 5.3: Comparison with baselines and state-of-the-art on the VQA-CP v2 dataset.

5.4 Results & Discussion

Evaluation Metric

We evaluate our model using the standard metric for the VQA dataset [7]. The hard accuracy, which accepts a predicted answer as correct only if it exactly matches the ground truth answer, is unsuitable for the particular dataset. The inter-annotator agreement is around 80%, which means that it would be impossible for an algorithm to achieve 100% accuracy. Moreover, as not all errors are equal, this metric can be overly strict. For instance, answering the question "What's in the sky?" with the answer "airplanes" instead of the singular "airplane" or its synonym "plane" would be penalized the same as choosing a completely unrelated answer such as "penguin".

A simple way to alleviate this issue is by taking advantage of the multiplicity of answers included in the dataset. As mentioned above, each question is accompanied by 10 answers provided by different annotators. The variation in the ground truth answers accounts to some degree for the real-world scenario, where questions can have multiple acceptable answers. A predicted answer a receives perfect score $s(a)$ if at least three human annotators have provided the same answer:

$$score(a) = \min(\frac{\mathbf{1}(a = a_i)}{3}, 1) \tag{5.1}$$

where $\mathbf{1}$ is the indicator function.

This evaluation is still imperfect. The set of possible answers is still limited in expressiveness. Consequently, the model can sometimes predict an answer that is reasonable to a human, but still receives a zero score. On the other hand, the VQA accuracy metric can in some cases give an inflated performance score. In some cases, the ground truths contain opposite answers, such as "left" and "right" or "yes" and "no". Even if the majority of the annotators answered "yes" but one to three of them answered "no", predicting the minority answer of "no" will still receive a positive score. Examples of these cases will be shown in Sec. 5.5.

Comparison to Baselines

Table 5.3 reports the performance of our model on the VQA-CP v2 dataset in comparison with baseline and state-of-the-art models. We consider the two first models as baselines, namely the "d-LSTM" and "d-LSTM Q + I". "d-LSTM" refers to the language-only model that encodes the question using a deep LSTM with 2 layers and disregards the image completely. "d-LSTM Q + I" uses the same layout to encode the question plus a pretrained CNN model to extract a global image feature vector. The question and image feature vectors are concatenated and fed to a classifier. As we can see, our model significantly outperforms both baselines. This performance gap is evidently attributed to the visual representations and the addition of a the cross-model attention mechanism. The global image feature vector, although adequate for image classification, is too coarse a representation to capture the information needed to answer specific questions.

Comparison to State-of-the-Art

When comparing the proposed Grounded Seq2Seq with sota models, we observe that our model yields competitive results, particularly in the "Other" category. Although "Other" is a very broad category, this result is supported by the motivation of our work. We hypothesize that because these are the questions that accept longer answers, they can benefit to a greater extent from a sequence generation setting. Note that no single model achieves best performance across all categories. We notice that the models *GVQA* and *UpDn+Adv* yield good results for the "Yes/No" questions outperforming the rest by a large margin. However, they achieve moderate results in the "Other" category. Compared to the baselines, sota models achieve the least performance improvement in "Number" questions. Counting requires a distinct kind of reasoning, which has stimulated a line of work focusing exclusively on this task [67, 75]. Another factor that seems to be particular significant for the overall model performance is the quality of visual features. Specifically, Faster RCNN features consistently outperform features from object recognition tasks. Models *SAN*, *GVQA* and *MCB* utilize regional features from a model trained on image classification, while *UpDn*, *UpDn+Adv* and the proposed model leverage the bottom-up attention that an object detection model such as Faster RCNN provide.

Ablation Study

We perform an ablation study to isolate components of our model and evaluate their effect on the performance of the proposed model. This study includes experiments where one or both modalities are missing (replaced by random vectors) to assess the contribution of the textual and visual modality as well as the random chance performance for each question type. Table 5.4 summarizes the examined variants of the Grounded Seq2Seq model and Table 5.5 presents the results of our experiments.

Model	Description
A	Replace the image and question representations with random vectors
B	Take only the image features as input
C	Take only the questions as input
D	Use a fused question and image representation only to initialize the decoder hidden state
E	Replace the spatial attention mechanism with that from *UpDn*
F	Use the final state of the question encoder without applying the self-attention
G	Use concatenation for the fusion instead of the Hadamard product
H	Use MUTAN fusion instead of the Hadamard product

Table 5.4: Characteristics of the models examined during the ablation study.

Model	Overall	Yes/No	Number	Other
(A): Random input	16.84	56.34	0.38	0.65
(B): Image input only	17.44	57.90	0.39	0.91
(C): Question input only	18.66	38.29	10.25	10.38
(D): Question and Image decoder initialization	35.51	38.57	11.10	40.30
(E): UpDn Attention Mechanism	33.20	39.56	11.54	35.45
(F): Question encoder without self-attention	35.92	40.83	11.93	40.03
(G): Concatenation fusion	35.42	39.47	12.30	39.35
(H): MUTAN fusion	35.67	40.00	11.46	39.70
Grounded Seq2Seq	**36.42**	**40.29**	**12.07**	**41.08**

Table 5.5: Ablation results.

The experiments with completely random or limited input (*A, B, C*) provide some intuition on the nature of the dataset. First, we observe that providing only image input yields a performance similar to random, while providing only question input yields better performance in the "Other" and "number" categories. This is consistent with observations in the literature about language bias in

VQA systems [3]. The performance of models (*A, B*) in "Yes/No" questions seems surprising at a first glance, but it results from the model always predicting *'no'* as the answer, which is actually the most common answer in the training data. Despite the shift of the prior distribution between the training and evaluation data, these models still achieve an accuracy of over 50% due to the evaluation metric rewarding minority answers even when the contradict the majority. It is clear that to achieve good performance in the "Other" category, incorporation of both modalities is important. For the "Number" category, most of the information is encoded in the question and visual grounding does not contribute much.

Models (*C*) and (*F*), investigate the chosen setup for the question encoder. Compared to the language-only baseline (see Table 5.3), the language-only version of our model achieves $+3\%$ absolute improvement. This suggests that the use of a bidirectional LSTM with self-attention provides a better representation of the input question. The significance of the self-attention mechanism is also confirmed by the reduced performance of model (*F*).

When comparing experiments (*D*) and (*E*) with Grounded Seq2Seq, we see that attending to the image features using the scaled dot-product attention mechanism at each decoding step has a positive impact on model performance. In particular, our results demonstrate that the selected attention mechanism is conducive to better visual grounding.

Finally, the experimentation with different fusion methods shows that the Hadamard product based fusion outperforms both concatenation (*G*) and MUTAN (*H*) fusion by 1.00% and 0.75% respectively. The slightly inferior performance of model (*G*) can be by the fact that concatenation increases the size of the feature vector while failing to capture the interactions that occur between the inputs. MUTAN fusion, on the other hand, is a factorized bilinear pooling method, which increases significantly the model complexity. Since we used the parameters from the original paper [10], the performance model (*H*) would probably benefit from further hyperparameter tuning.

5.5 Examples

In this section, we provide a few examples showcasing outputs of the Grounded Seq2Seq model, in order gain some insight about the cases in which our model succeeds or fails. Figure 5.2 demonstrates the model accuracy on certain question categories. From that we can infer that the model does well on questions related to recognizing activity ("what sport", "what is the person"), object class ("what kind/type/room/animal") or visual attributes ("what color"). However, it performs poorly on questions that require common sense reasoning or real-world knowledge ("why", "what brand", "what time").

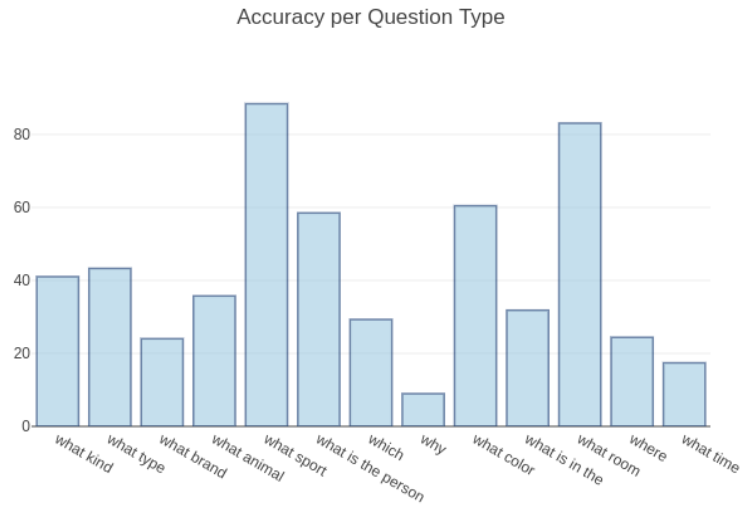


Figure 5.2: Model performance on specific question types.

Going through the predicted answers for the samples of the test set, we observe that not all correct predictions are rewarded equally and not all errors are equally bad. We select three representative examples for each of the following four cases: predictions that get a perfect score, predictions that get a partial score, predictions that get a zero score but pass as plausible and incorrect predictions. For each sample, along with the question-image pair and the predicted answer, we provide all available ground truths. For each unique ground truth we specify in parentheses the number of its occurrences.

Correct Answers

First, we present a few examples for which the model achieves a perfect score of 1.

What is the dog playing with?

Predicted Answer: teddy bear

Ground Truths:

bear	(4)
teddy bear	(3)
stuffed bear	(2)
toy	(1)

Which landmark is this?

Predicted Answer: washington monument

Ground Truths:

washington monument	(7)
pentagon	(1)
sears tower	(1)
tall pyramid	(1)

Where is the man in this photo?

Predicted Answer: on horse

Ground Truths:

on horse	(3)
on horseback	(1)
horse	(1)
field	(1)
woods	(1)
riding	(1)
outside	(1)

Table 5.6: Examples of questions and generated answers that are assigned an accuracy score of 1.

Almost Correct Answers

Next, we look into outputs that are assigned a partial score, meaning the cases where the predicted answers match only 1-2 out of the 10 provided ground truths. We chose these examples to showcase instances for which our model is assigned a partial score, even though it succeeds in predicting semantically correct answers. In general, we observe that predicted answers can fall under this category, if they exhibit a different level of granularity compared to the ground truths. For instance, even though "playing baseball" is considered a correct answer for the second example, "batting" would indeed be a more precise answer.

	*What is the color of the cup?*		
	Predicted Answer:	orange	
	Ground Truths:	red	(9)
		orange	(1)
	*What is he doing?*		
	Predicted Answer:	playing baseball	
	Ground Truths:	batting	(6)
		playing baseball	(2)
		swinging	(1)
		baseball bat	(1)
	*What color is the train?*		
	Predicted Answer:	yellow and blue	
	Ground Truths:	blue and yellow	(6)
		yellow and blue	(2)
		blue yellow	(1)
		blue/yellow	(1)

Table 5.7: Examples of questions and generated answers that are assigned a partial accuracy score.

Almost Incorrect Answers

We observe that in some cases the selected evaluation metric is in disagreement with human evaluation. The following examples demonstrate that plausible answers will be automatically evaluated as wrong, if they do not exactly match at least one of the ground truths provided by the human annotators.

Where is the riding crop?

Predicted Answer: in front of horse

Ground Truths:		
	on horse	(2)
	in boy's hand	(2)
	right hand	(1)
	hand	(1)
	on ground	(1)
	foreground	(1)
	around rider's waist	(1)
	on horse	(1)

Why is the man in the air?

Predicted Answer: playing frisbee

Ground Truths:		
	jumping	(3)
	catching frisbee	(2)
	to catch frisbee	(1)
	he's jumping	(1)
	he jumped	(1)
	catching	(1)
	catching frisby	(1)

What is the color of the train?

Predicted Answer: red and black

Ground Truths:		
	red	(8)
	orange	(2)

Table 5.8: Examples of questions and generated answers that are assigned an accuracy score of 0, but are plausible.

Incorrect Answers

Lastly, we consider a few examples of incorrect predictions. What these examples indicate is that our model fails to predict the correct answer in cases of longer questions and more complex scenes. Even without looking at the images, we can infer that the outputs of the first and third examples are erroneous. The second example illustrates a case of a plausible assumption from a language viewpoint that is, however, not grounded in the visual input.

What is the name of the store on the left hand side of the screen that is written in English?

Predicted Answer: no parking

Ground Truths: sasa (10)

What eating utensils are on the table?

Predicted Answer: fork and knife

Ground Truths:

forks and spoons	(3)
spoon and fork	(2)
fork and spoons	(1)
spoons forks	(1)
spoon fork	(1)
fork spoon	(1)
8	(1)

What type of herb is on the left?

Predicted Answer: orange

Ground Truths:

parsley	(7)
basil	(3)

Table 5.9: Examples of questions and generated answers that are assigned an accuracy score of 0.

Chapter 6

Conclusion

6.1 Conclusions

In this thesis, we propose a multimodal sequence-to-sequence model for the task of visual question answering. VQA has benefited greatly from progress in natural language processing and computer vision, with more and more sophisticated methods contributing towards improved overall performance. State-of-the-art models approach open-ended VQA as a classification task over the most common answers in the dataset. However, the restriction of potential answers narrows the expressiveness of the models and limits their application to closed-world settings. Motivated by this observation, we decided to investigate the feasibility of adapting a popular sequence generation framework to the task of VQA. Specifically, we propose the Grounded Seq2Seq model that generates answers conditioned on the image-question input pairs. Our model follows a similar pipeline to that of established methods, but avoids treating compound answers as separate classes from the individual words that comprise them. Instead, it enables the generation of answers from a large vocabulary that is shared among questions and answers.

The comparison with baseline models and the accompanying ablation study help us identify the design choices that contribute most to the performance of the proposed model. First, our model makes use of prior knowledge from powerful pretrained models both for encoding the image as a collection of region-specific features and for the initialization of the word embedding layer. Moreover, the ablation results suggest that the use of attention is critical for model performance. We make use of two attention mechanisms: a self-attention mechanism that is employed for the computation of a fixed-sized question representation, and a cross-modal attention mechanism that allows the decoder to focus on the most relevant visual information.

During the answer generation phase, the decoder attends over the image regions at each decoding step using a scaled dot-product attention mechanism. Both the type of attention mechanism and its recurrent application seem to contribute significantly to the model's accuracy. Our experiments indicate that the selected spatial attention mechanism yields results superior to those in existing VQA approaches. Additionally, the attention feedback loop seems to enable the grounding of the generated answer to the given image. We attribute the performance of the proposed model to certain training details, as well. In order to reduce the memorization of language biases, we use a data augmentation technique. By randomly sampling the target answer from the given set of possible ground truths, we reduce the model's generalization gap. Lastly, we apply teacher forcing, a method that speeds convergence and improves training stability.

We evaluate the proposed Grounded Seq2Seq model on the VQA-CP v2 dataset and achieve results comparable to those of state-of-the-art models. Although VQA models are far from achieving human-level performance, the outcome of our experiments challenges the need to confine possible answers to pre-defined independent classes. In summary, the use of sequence generation models shows potential for open-ended visual question answering and acts as a step in the direction of utilizing VQA systems in real-world scenarios.

6.2 Future Work

Combining natural language with computer vision constitutes a complex problem, as the moderate performance of current models on the VQA task corroborates. A shift in the distribution of answers between the training and the test models reveals that visual grounding of language is still an open problem. One limitation of our work is that it is applied on a dataset with short answers. This setting limits the potential of sequence generation models to take advantage of the compositional structure of language. As a result, we weren't able to exhaustively test our hypothesis that a sequence generation model can learn semantic relations of answer words in order to generalize to new answers. We plan to experiment with the recently released GQA dataset [33], that is more appropriate for the proposed model as it provides both short and long answers. This dataset would enable us to examine the model's ability to generate novel answers containing words seen in training instances.

Following the work of Ramakrishnan et al. [59], we are interested in leveraging adversarial examples [25] in order to increase the robustness of our answer generation model. To be more exact, we hypothesize that we can improve the grounding ability of our model by generating samples consisting of random combinations of questions and images assigned a ground truth in one of the following manners. First, we can retain the original ground truth of the question in an attempt to reduce the dependency on language biases. Second, we could replace answer words by randomly words from the vocabulary that belong to the same category (such as number, sport, color etc.). Potentially correct answers can be avoided by eliminating from the pool of possible words those that appear in ground truths for other questions about the selected image. In reverse, we can employ semantically wrong answers that appear, nonetheless, as ground truths for the selected image given other questions in the dataset.

Furthermore, prior research in VQA has demonstrated that using more powerful pretrained vision models for feature extraction improves significantly the model performance. Although we use pretrained word embeddings to initialize the embedding layer, it is expected that superior sentence representations could be extracted from pretrained language models. In general, language models trained on large corpora are able to capture contextual relations between words. We assume that transfer learning would be especially beneficial for questions that require common sense reasoning. Consequently, a future direction could involve transferring knowledge from state-of-the-art Transformer networks [68]. In future experiments, we will fine-tune a Bidirectional Encoder Representations from Transformer (BERT, [19]), a recently released sequence model that has achieved performance improvements across a wide variety of downstream NLP tasks.

Bibliography

[1] Underfitting and Overfitting. https://vitalflux.com/wp-content/uploads/2015/02/fittings.jpg, 2015. [Online; accessed 29-April-2019].

[2] Aishwarya Agrawal, Dhruv Batra, and Devi Parikh. Analyzing the behavior of visual question answering models. In *Proceedings of the Conference on Empirical Methods in Natural Language Processing*, pages 1955–1960, 2016. doi: 10.18653/v1/D16-1203. URL http://aclweb.org/anthology/D16-1203.

[3] Aishwarya Agrawal, Dhruv Batra, Devi Parikh, and Aniruddha Kembhavi. Don't just assume; look and answer: Overcoming priors for visual question answering. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pages 4971–4980, 2018.

[4] Afshine Amidi and Shervine Amidi. Deep Learning Tips and Tricks cheatsheet. https://stanford.edu/~shervine/teaching/cs-230/cheatsheet-deep-learning-tips-and-tricks, 2017. [Online; accessed 11-May-2019].

[5] Peter Anderson, Xiaodong He, Chris Buehler, Damien Teney, Mark Johnson, Stephen Gould, and Lei Zhang. Bottom-up and top-down attention for image captioning and visual question answering. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pages 6077–6086, 2018.

[6] Jacob Andreas, Marcus Rohrbach, Trevor Darrell, and Dan Klein. Neural module networks. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pages 30–48, June 2016.

[7] Stanislaw Antol, Aishwarya Agrawal, Jiasen Lu, Margaret Mitchell, Dhruv Batra, C. Lawrence Zitnick, and Devi Parikh. VQA: Visual Question Answering. In *International Conference on Computer Vision (ICCV)*, 2015.

[8] Dzmitry Bahdanau, Kyunghyun Cho, and Yoshua Bengio. Neural machine translation by jointly learning to align and translate. *arXiv preprint arXiv:1409.0473*, 2014.

[9] Richard Bellman. *Adaptive control process: a guided tour*. Princeton University Press, 1961.

[10] Hedi Ben-Younes, Remi Cadene, Matthieu Cord, and Nicolas Thome. Mutan: Multimodal tucker fusion for visual question answering. In *Proceedings of the IEEE International Conference on Computer Vision*, pages 2612–2620, 2017.

[11] Christopher M. Bishop. *Pattern recognition and machine learning, 5th Edition*. Information science and statistics. Springer, 2007. ISBN 9780387310732. URL http://www.worldcat.org/oclc/71008143.

[12] Léon Bottou. Large-scale machine learning with stochastic gradient descent. In *Proceedings of COMPSTAT'2010*, pages 177–186. Springer, 2010.

[13] Eleftheria Briakou, Nikos Athanasiou, and Alexandros Potamianos. Cross-topic distributional semantic representations via unsupervised mappings. *CoRR*, abs/1904.05674, 2019. URL http://arxiv.org/abs/1904.05674.

[14] Wei-Lun Chao, Hexiang Hu, and Fei Sha. Being negative but constructively: Lessons learnt from creating better visual question answering datasets. In *Proceedings of the Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies*, pages 431–441, 2018. URL http://aclweb.org/anthology/N18-1040.

[15] Chung-Cheng Chiu, Tara N. Sainath, Yonghui Wu, Rohit Prabhavalkar, Patrick Nguyen, Zhifeng Chen, Anjuli Kannan, Ron J. Weiss, Kanishka Rao, Katya Gonina, Navdeep Jaitly, Bo Li, Jan Chorowski, and Michiel Bacchiani. State-of-the-art speech recognition with sequence-to-sequence models. *CoRR*, abs/1712.01769, 2017. URL http://arxiv.org/abs/1712.01769.

[16] George Cybenko. Approximation by superpositions of a sigmoidal function. *MCSS*, 2:303–314, 1989.

[17] Abhishek Das, Satwik Kottur, Khushi Gupta, Avi Singh, Deshraj Yadav, José M. F. Moura, Devi Parikh, and Dhruv Batra. Visual dialog. *CoRR*, abs/1611.08669, 2016. URL http://arxiv.org/abs/1611.08669.

[18] Yann Dauphin, Razvan Pascanu, Çaglar Gülçehre, Kyunghyun Cho, Surya Ganguli, and Yoshua Bengio. Identifying and attacking the saddle point problem in high-dimensional non-convex optimization. *CoRR*, abs/1406.2572, 2014. URL http://arxiv.org/abs/1406.2572.

[19] Jacob Devlin, Ming-Wei Chang, Kenton Lee, and Kristina Toutanova. BERT: pre-training of deep bidirectional transformers for language understanding. *CoRR*, abs/1810.04805, 2018. URL http://arxiv.org/abs/1810.04805.

[20] John Duchi, Elad Hazan, and Yoram Singer. Adaptive subgradient methods for online learning and stochastic optimization. *Journal of Machine Learning Research*, 12(Jul):2121–2159, 2011.

[21] J. R. Firth. A synopsis of linguistic theory 1930-55. 1952-59:1–32, 1957.

[22] Scott Fortmann-Roe. Understanding the Bias-Variance Tradeoff. http://scott.fortmann-roe.com/docs/BiasVariance.html, 2012. [Online; accessed 29-April-2019].

[23] Akira Fukui, Dong Huk Park, Daylen Yang, Anna Rohrbach, Trevor Darrell, and Marcus Rohrbach. Multimodal compact bilinear pooling for visual question answering and visual grounding. In *Proceedings of the 2016 Conference on Empirical Methods in Natural Language Processing*, pages 457–468, 2016. URL http://aclweb.org/anthology/D16-1044.

[24] Ian Goodfellow, Yoshua Bengio, and Aaron Courville. *Deep Learning*. MIT Press, 2016. http://www.deeplearningbook.org.

[25] Ian J Goodfellow, Jonathon Shlens, and Christian Szegedy. Explaining and harnessing adversarial examples. *arXiv preprint arXiv:1412.6572*, 2014.

[26] Alison Gopnik and Andrew N. Meltzoff. Semantic and cognitive development in 15- to 21-month-old children. *Journal of Child Language*, 11(3):495–513, 1984. doi: 10.1017/S0305000900005912.

[27] Yash Goyal, Tejas Khot, Douglas Summers-Stay, Dhruv Batra, and Devi Parikh. Making the V in VQA matter: Elevating the role of image understanding in visual question answering. *CoRR*, abs/1612.00837, 2016. URL http://arxiv.org/abs/1612.00837.

[28] Alex Graves, Greg Wayne, and Ivo Danihelka. Neural turing machines. *CoRR*, abs/1410.5401, 2014. URL http://arxiv.org/abs/1410.5401.

[29] Stevan Harnad. The symbol grounding problem. *Physica D: Nonlinear Phenomena*, 42(1): 335 – 346, 1990. ISSN 0167-2789. doi: https://doi.org/10.1016/0167-2789(90)90087-6. URL http://www.sciencedirect.com/science/article/pii/0167278990900876.

[30] Zellig S. Harris. Distributional structure. WORD, 10(2-3):146–162, 1954. doi: 10.1080/00437956.1954.11659520. URL https://doi.org/10.1080/00437956.1954.11659520.

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

[32] Md. Zakir Hossain, Ferdous Sohel, Mohd Fairuz Shiratuddin, and Hamid Laga. A comprehensive survey of deep learning for image captioning. *CoRR*, abs/1810.04020, 2018. URL http://arxiv.org/abs/1810.04020.

[33] Drew A. Hudson and Christopher D. Manning. GQA: a new dataset for compositional question answering over real-world images. *CoRR*, abs/1902.09506, 2019. URL http://arxiv.org/abs/1902.09506.

[34] Allan Jabri, Armand Joulin, and Laurens van der Maaten. Revisiting visual question answering baselines. In *Proceedings of the European Conference on Computer Vision*, pages 727–739, 2016.

[35] Gareth James, Daniela Witten, Trevor Hastie, and Robert Tibshirani. *An Introduction to Statistical Learning: With Applications in R*. Springer Publishing Company, Incorporated, 2014. ISBN 1461471370, 9781461471370.

[36] J Judd. Neural network design and the complexity of learning [microform] /. 05 2019.

[37] Andrej Karpathy. CS231n Convolutional Neural Networks for Visual Recognition. http://cs231n.github.io/neural-networks-1, 2017. [Online; accessed 3-May-2019].

[38] Douwe Kiela, Alexis Conneau, Allan Jabri, and Maximilian Nickel. Learning visually grounded sentence representations. *CoRR*, abs/1707.06320, 2017. URL http://arxiv.org/abs/1707.06320.

[39] Jin-Hwa Kim, Kyoung Woon On, Woosang Lim, Jeonghee Kim, JungWoo Ha, and Byoung-Tak Zhang. Hadamard product for low-rank bilinear pooling. In *Proceedings of the International Conference on Learning Representations*, 2017.

[40] Diederik P Kingma and Jimmy Ba. Adam: A method for stochastic optimization. *arXiv preprint arXiv:1412.6980*, 2014.

[41] Jamie Kiros, William Chan, and Geoffrey Hinton. Illustrative language understanding: Large-scale visual grounding with image search. In *Proceedings of the 56th Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pages 922–933, Melbourne, Australia, July 2018. Association for Computational Linguistics. URL https://www.aclweb.org/anthology/P18-1085.

[42] Ryan Kiros, Yukun Zhu, Ruslan Salakhutdinov, Richard S. Zemel, Antonio Torralba, Raquel Urtasun, and Sanja Fidler. Skip-thought vectors. *CoRR*, abs/1506.06726, 2015. URL http://arxiv.org/abs/1506.06726.

[43] Ranjay Krishna, Yuke Zhu, Oliver Groth, Justin Johnson, Kenji Hata, Joshua Kravitz, Stephanie Chen, Yannis Kalantidis, Li-Jia Li, David A. Shamma, Michael S. Bernstein, and Li Fei-Fei. Visual genome: Connecting language and vision using crowdsourced dense image annotations. *International Journal of Computer Vision*, 123:32–73, 2017. URL https://doi.org/10.1007/s11263-016-0981-7.

[44] Alex Krizhevsky, Ilya Sutskever, and Geoffrey E Hinton. Imagenet classification with deep convolutional neural networks. In F. Pereira, C. J. C. Burges, L. Bottou, and K. Q. Weinberger, editors, *Advances in Neural Information Processing Systems 25*,

pages 1097–1105. Curran Associates, Inc., 2012. URL http://papers.nips.cc/paper/4824-imagenet-classification-with-deep-convolutional-neural-networks.pdf.

[45] Barbara Landau, Linda Smith, and Susan Jones. Object perception and object naming in early development. *Trends in Cognitive Sciences*, 2:19–24, 01 1998. doi: 10.1016/S1364-6613(97)01111-X.

[46] Yann LeCun, Léon Bottou, Yoshua Bengio, Patrick Haffner, et al. Gradient-based learning applied to document recognition. *Proceedings of the IEEE*, 86(11):2278–2324, 1998.

[47] Zhouhan Lin, Minwei Feng, Cícero Nogueira dos Santos, Mo Yu, Bing Xiang, Bowen Zhou, and Yoshua Bengio. A structured self-attentive sentence embedding. *CoRR*, abs/1703.03130, 2017. URL http://arxiv.org/abs/1703.03130.

[48] Jiasen Lu, Xiao Lin, Dhruv Batra, and Devi Parikh. Deeper LSTM and normalized CNN visual question answering model. https://github.com/VT-vision-lab/VQA_LSTM_CNN, 2015.

[49] Jiasen Lu, Jianwei Yang, Dhruv Batra, and Devi Parikh. Hierarchical question-image co-attention for visual question answering. In *Proceedings of the Conference on Neural Information Processing Systems*, pages 289–297, 2016. URL http://dl.acm.org/citation.cfm?id=3157096.3157129.

[50] Minh-Thang Luong, Hieu Pham, and Christopher D. Manning. Effective approaches to attention-based neural machine translation. *CoRR*, abs/1508.04025, 2015. URL http://arxiv.org/abs/1508.04025.

[51] Arthur M Glenberg and David A Robertson. Symbol grounding and meaning: A comparison of high-dimensional and embodied theories of meaning. *Journal of Memory and Language*, 43:379–401, 10 2000. doi: 10.1006/jmla.2000.2714.

[52] Andrew L. Maas. Rectifier nonlinearities improve neural network acoustic models. 2013.

[53] Tomas Mikolov, Kai Chen, G.s Corrado, and Jeffrey Dean. Efficient estimation of word representations in vector space. *Proceedings of Workshop at ICLR*, 2013, 01 2013.

[54] Michael A. Nielsen. Neural networks and deep learning, 2018. URL http://neuralnetworksanddeeplearning.com/.

[55] Adam Paszke, Sam Gross, Soumith Chintala, Gregory Chanan, Edward Yang, Zachary DeVito, Zeming Lin, Alban Desmaison, Luca Antiga, and Adam Lerer. Automatic differentiation in pytorch. 2017.

[56] Min Peng, Chongyang Wang, T Chen, Guangyuan Liu, and Xiaolan Fu. Dual temporal scale convolutional neural network for micro-expression recognition. *Frontiers in Psychology*, 10 2017.

[57] Jeffrey Pennington, Richard Socher, and Christopher D. Manning. Glove: Global vectors for word representation. In *Proceedings of the Conference on Empirical Methods in Natural Language Processing*, pages 1532–1543, 2014. URL http://www.aclweb.org/anthology/D14-1162.

[58] Jeffrey Pennington, Richard Socher, and Christopher D. Manning. Glove: Global vectors for word representation. In *Empirical Methods in Natural Language Processing (EMNLP)*, pages 1532–1543, 2014. URL http://www.aclweb.org/anthology/D14-1162.

[59] Sainandan Ramakrishnan, Aishwarya Agrawal, and Stefan Lee. Overcoming language priors in visual question answering with adversarial regularization. In *Proceedings of the Conference on Neural Information Processing Systems*, pages 1548–1558, 2018.

[60] Shaoqing Ren, Kaiming He, Ross B. Girshick, and Jian Sun. Faster R-CNN: towards real-time object detection with region proposal networks. *CoRR*, abs/1506.01497, 2015. URL http://arxiv.org/abs/1506.01497.

[61] F. Rosenblatt. *Principles of neurodynamics: perceptrons and the theory of brain mechanisms*. Report (Cornell Aeronautical Laboratory). Spartan Books, 1962. URL https://books.google.ca/books?id=7FhRAAAAMAAJ.

[62] David E Rumelhart, Geoffrey E Hinton, and Ronald J Williams. Learning internal representations by error propagation. Technical report, California Univ San Diego La Jolla Inst for Cognitive Science, 1985.

[63] Carina Silberer and Mirella Lapata. Learning grounded meaning representations with autoencoders. In *Proceedings of the 52nd Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pages 721–732, Baltimore, Maryland, June 2014. Association for Computational Linguistics. doi: 10.3115/v1/P14-1068. URL https://www.aclweb.org/anthology/P14-1068.

[64] Nitish Srivastava, Geoffrey E. Hinton, Alex Krizhevsky, Ilya Sutskever, and Ruslan Salakhutdinov. Dropout: A simple way to prevent neural networks from overfitting. *Journal of Machine Learning Research*, 15(1):1929–1958, 2014.

[65] Ilya Sutskever, Oriol Vinyals, and Quoc V Le. Sequence to sequence learning with neural networks. In Z. Ghahramani, M. Welling, C. Cortes, N. D. Lawrence, and K. Q. Weinberger, editors, *Advances in Neural Information Processing Systems 27*, pages 3104–3112. Curran Associates, Inc., 2014. URL http://papers.nips.cc/paper/5346-sequence-to-sequence-learning-with-neural-networks.pdf.

[66] Sergios Theodoridis. *Machine Learning: A Bayesian and Optimization Perspective*. Academic Press, Inc., Orlando, FL, USA, 1st edition, 2015. ISBN 0128015225, 9780128015223.

[67] Alexander Trott, Caiming Xiong, and Richard Socher. Interpretable counting for visual question answering. *CoRR*, abs/1712.08697, 2017. URL http://arxiv.org/abs/1712.08697.

[68] Ashish Vaswani, Noam Shazeer, Niki Parmar, Jakob Uszkoreit, Llion Jones, Aidan N Gomez, Lukasz Kaiser, and Illia Polosukhin. Attention is all you need. In *Proceedings of the Conference on Neural Information Processing Systems*, pages 5998–6008. 2017. URL http://papers.nips.cc/paper/7181-attention-is-all-you-need.pdf.

[69] Ashish Vaswani, Noam Shazeer, Niki Parmar, Jakob Uszkoreit, Llion Jones, Aidan N Gomez, Łukasz Kaiser, and Illia Polosukhin. Attention is all you need. In *Advances in neural information processing systems*, pages 5998–6008, 2017.

[70] Subhashini Venugopalan, Marcus Rohrbach, Jeff Donahue, Raymond J. Mooney, Trevor Darrell, and Kate Saenko. Sequence to sequence - video to text. *CoRR*, abs/1505.00487, 2015. URL http://arxiv.org/abs/1505.00487.

[71] Paul J Werbos et al. Backpropagation through time: what it does and how to do it. *Proceedings of the IEEE*, 78(10):1550–1560, 1990.

[72] David H. Wolpert. The lack of a priori distinctions between learning algorithms. *Neural Computation*, 8(7):1341–1390, 1996. doi: 10.1162/neco.1996.8.7.1341.

[73] Zichao Yang, Xiaodong He, Jianfeng Gao, Li Deng, and Alex Smola. Stacked attention networks for image question answering. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pages 21–29, 2016.

[74] Reza Zadeh. The hard thing about deep learning. https://www.oreilly.com/ideas/the-hard-thing-about-deep-learning, 2016. [Online; accessed 3-May-2019].

[75] Yan Zhang, Jonathon S. Hare, and Adam Prügel-Bennett. Learning to count objects in natural images for visual question answering. *CoRR*, abs/1802.05766, 2018. URL http://arxiv.org/abs/1802.05766.

Appendix A

Abbreviations

(AI): Artificial Intelligence
(ANN): Artificial Neural Network
(BiLSTM): Bi-directional LSTM
(BOW): Bag-of-Words
(CNN): Convolutional Neural Network
(CV): Computer Vision
(DL): Deep Learning
(DNNs): Deep Neural Networks
(GPU): Graphical Processor Unit
(GD): Gradient Descent
(VQA): Visual Question Answering
(GVQA): Grounded Visual Question Answering model, [33]
(LR): Logistic Regression classifier
(LSTM): Long Short-Term Memory unit
(MCB): Multimodal Compact Bilinear Pooling, [23]
(ML): Machine Learning
(MT): Machine Translation
(MUTAN): Multimodal Tucker Fusion for Visual Question Answering, [10]
(NBOW): Neural Bag-of-Words
(NLP): Natural Language Processing
(RNNs): Recurrent Neural Networks
(RoI): Region of Interest
(RPN): Region Proposal Network
(R-CNN): Region-based Convolutional Neural Network
(SAN): Stacked Attention Network, [73]
(SGD): Stochastic Gradient Descent
(SVM): Support Vector Machine classifier
(Seq2Seq): Sequence-to-Sequence model
(VQA): Visual Question Answering
(VQA-CP): Visual Question Answering under Changing Priors dataset, [33]
(UpDn): Up-Down model, [5]