Learning to Reason with Adaptive Computation

Mark Neumann

A dissertation submitted in partial fulfillment

of the requirements for the degree of

Master of Science

of

University College London.

Department of Computer Science, UCL

University College London

Supervised by Pontus Stenetorp and Sebastian Riedel

September 5, 2016

Abstract

In this thesis, we aim to explore two main approaches to reduce the computational time expended whilst also improving the inturpretability of neural network models. To this end, we jointly employ several ideas introduced in the literature for adaptive computation, multi-step, data-independent reasoning over a corpus and light representations of words using attention. We demonstrate that multiple inference steps is beneficial for the task of Recognising Textual Entailment and that choosing the number of computational steps is difficult. We additionally show that in a task requiring joint analysis of multiple sentences, alignment of representations is more important than temporally dependent representations. We also show that temporal dependencies do not have to be contained within the representations, instead using recurrent neural networks to generate consistent, conditional attention over sentences. We introduce the first model involving Adaptive Computation Time which provides a performance benefit on top of a similar model without an adaptive component, achieving near state-of-the-art performance on the SNLI corpus.

Acknowledgements

This project was made considerably easier by the enthusiasm and advice of my supervisor, Pontus Stenetorp, for which I am extremely grateful. Advice from the Machine Reading group, my family, Jonathan Godwin, Chris Loy, James Gin and Laura Douglas was also much appreciated - I enjoyed working on this more because of you.

Contents

List of Figures

List of Tables

Chapter 1

Introduction

The problem of Recognising Textual Entailment, determining if a ”premise” entails a ”hypothesis”, has long been seen as a pre-requisite task for the over-arching goal of full Natural Language Understanding. RTE is a difficult problem due to the ”soft inference” required - humans utilise a massive amount of prior knowledge when inferring implications between statements. Additionally, humans are easily able to combine multiple sources of knowledge or multiple reasoned arguments in order to successfully resolve an inference.

Related to the idea of combining inferences, a fundamental aspect of human reasoning is the ability to adapt the required level of brain-power to the task. Decomposition of statements is a natural way to reason - often, only some parts of a sentence will contribute to entailment and clearly, statements with different levels of complexity will require differing numbers of parts to be combined/resolved.

Recently, artificial neural networks have become prevalent in the Machine Learning community, demonstrating state of the art performance in many tasks. However, many neural networks are computationally heavy or slow - a result of deep structures requiring sequential, large matrix multiplications. Additionally, as they are often trained using backpropagation on a deterministic computation graph, it is unusual for the number of operations executed for a particular example to differ. Given the complexity of these new approaches, methods to reduce time both during training and during inference have never been more important.

This thesis seeks to address two main problems:

- *Can we devise a method for combining inferences over a multi-sentence problem which adapts it's time complexity to the difficulty of the problem in an organic way? Can such a method provide insight into how inferences are resolved? Finally, can such a method be naturally more efficient by moderating it's computation without suffering with respect to task performance?*

- *For problems requiring the analysis of two or more sources of text, are methods which require temporally dependent processing of the sources the best way to represent words and resolve inferences? Can methods which use no temporal processing at all speed up training time without damaging performance?*

Neural models for RTE often use attention mechanisms over vector representations of words. In this thesis, we introduce several models of adaptive reasoning over computationally efficient, temporally independent representations generated over the premise and hypothesis of an RTE example.

We demonstrate the effectiveness of adaptive computation for varying the inference required for examples of different complexity and show the dual application of models designed for Machine Reading applied to textual entailment. We introduce the first model involving Adaptive Computation Time which provides a performance benefit on top of a similar model without an adaptive component.

1.1 A Motivating Example

When humans resolve entailment problems, we are adept at not only breaking down a large problem into sub-problems, but also re-combining the sub-problems in a structured way. Often in simple problems, only the first step is necessary. For instance, we can resolve the following entailment problem:

- **Premise**: An elderly gentleman is standing under an umbrella because there is a thunderstorm.

- **Hypothesis**: It is raining heavily.

by recognising that there are parts of the premise and hypothesis which can be *independently* aligned and there is redundant information in the premise. For instance, here we can simply recognise that a thunderstorm would normally include poor weather and resolve the inference correctly.

However, in more complicated cases, it can be necessary to both decompose and then reason. For instance the contradicting statements:

- **Premise:** An elderly gentleman stands near a bus stop, using an umbrella for shelter because there is a thunderstorm.
- **Hypothesis:** An old man holding a closed umbrella is sheltering from bad weather under a bus stop.

require multi-step, *temporally dependent* reasoning to resolve correctly. If we were simply to align parts of the sentences, we would conclude that an old man is present,the weather is bad and even on a second level of reasoning, that *he is sheltering from the weather*, which might lead to a positive resolution. On closer examination, the key to resolution here is the action relating entities and conditions in the scene, leading to an inference chain similar to:

- An old man, an umbrella and a bus stop are present ✓
- The weather is bad ✓
- He is sheltering from the weather ✓
- He is sheltering from the weather *using an umbrella* ✗

where the final true/false statement is built up from first observing facts about the scene and then combining and extending them. **This idea of combining distinct, low-level inferences about a Natural Language Inference problem is the specific problem addressed in this thesis.**

1.2 Thesis Outline

This thesis is divided into 3 main sections. First, we examine the history and background of Neural Networks, discussing ideas surrounding feedforward, recurrent and unsupervised networks, with particular applications, such as word representations discussed in detail. We then describe in detail the Adaptive Computation Time algorithm, giving various analogies and diagrams so the reader can understand its fundamental contribution in detail. Additionally, we examine the history of approaches to Recognising Textual Entailment and introduce the Stanford Natural Language Inference corpus.

We also briefly examine attention mechanisms used to eliminate the bottleneck formed by generating single representations of sentences and how this can aid multi-step reasoning. Finally, we describe two adaptive reasoning architectures, the Adaptive Iterative Alternating Attention model and the Adaptive Decomposable Attention model and analyse their performance on the SNLI corpus.

1.3 Summary of Results

Initially, we investigated the impact of taking multiple, temporally dependant inference steps when attempting to determine entailment. In each inference step, the premise and hypothesis are attended over using a neural attention mechanism. Below we present the difference in validation accuracy for different numbers of inference steps for the Iterative Alternating Attention model, as well as our Adaptive extension.

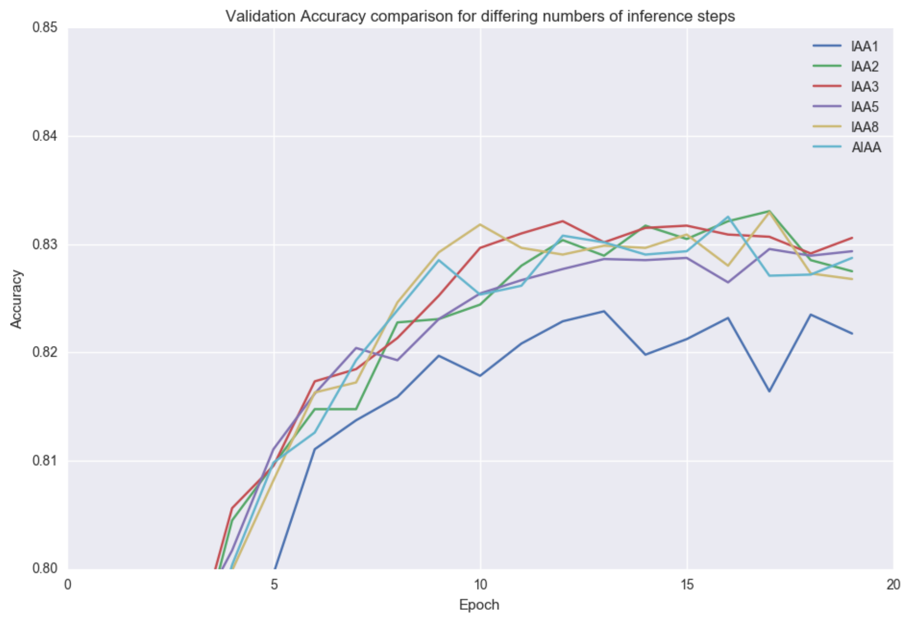


Figure 1.1: Demonstration of the accuracy gain from using multiple inference steps. Clearly, multiple steps leads to a higher accuracy, but the exact number of steps is hard to determine.

Secondly, we adapt the way we encode vector representations of words using Decomposable Attention, a fast, parallisable method for generating word-by-word alignments. We then use the same multi-step inference methods to generate attention visualisations which clearly demonstrate multi-focus sequential reasoning.

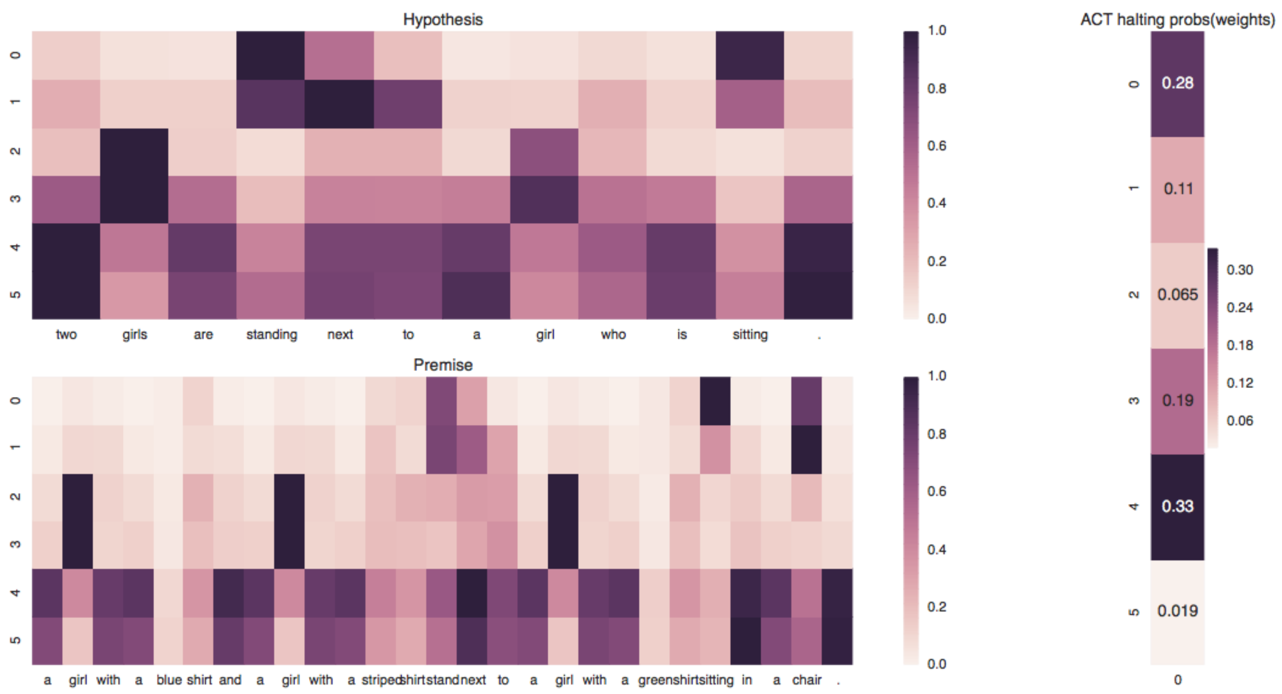


Figure 1.2: Attention weight visualisation for an example taking multiple steps. In each step(indexed on the right), first the model reads the hypothesis. The model then reads the premise, incorporating knowledge from reading the hypothesis, allowing it to compare. This forms a state for a recurrent controller, which is passed over timesteps, meaning the attention from one timestep influences the next, creating a coherent reasoning process. This example demonstrates clear multi-reference resolution.

Finally, in order to fully examine the exact impact of taking multiple inference steps, we extract the predictions the model would have made at each step, which we find to be extremely non-trivial. We observe examples where particular attention representations dramatically change the prediction, as well as examples where including more information into the inference leads to both less certain and incorrect predictions, reinforcing our claim that choosing the number of inference steps is hard and important.

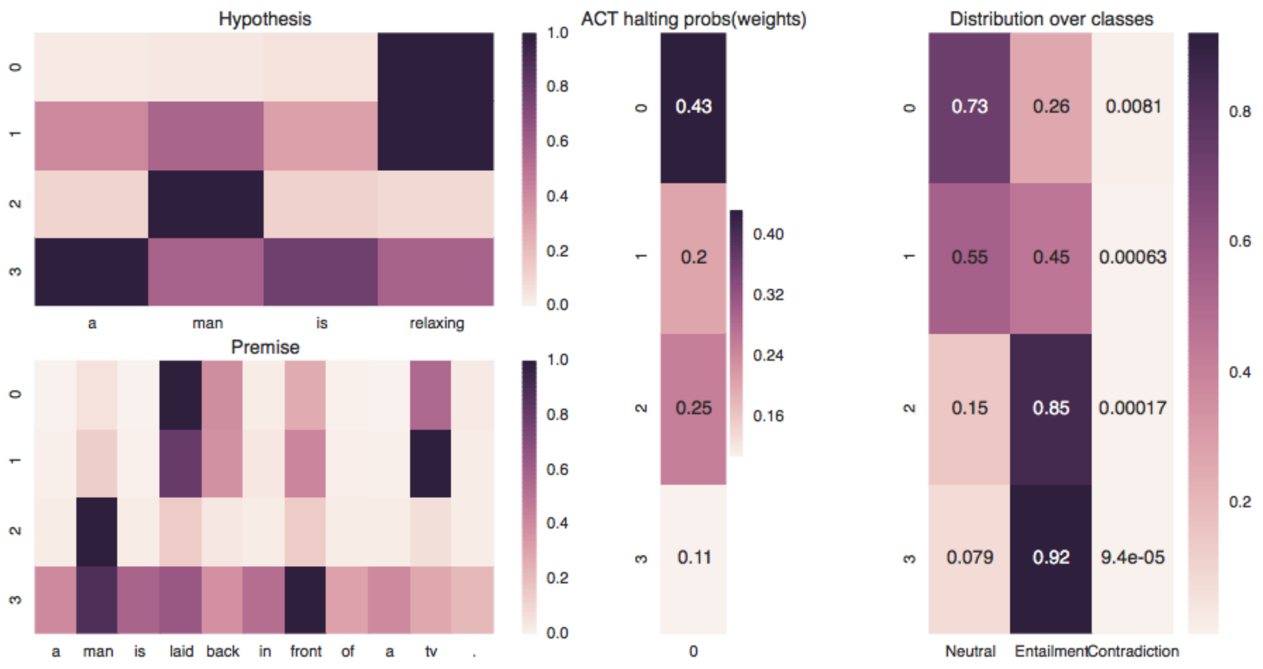


Figure 1.3: The model's prediction changes over time as it incorporates more information from different attention ”glimpses” over the hypothesis and premise.

Our work resulted in the first demonstration of a performance improvement in a model which learns one of its own hyperparameters - a 3.6% improvement. With a test accuracy of 83.8% on the SNLI corpus, our model is competitive with state of the art models, including ones which use an order of magnitude more parameters and are considerably more difficult to interpret.

1.4 Repositories

Code for the models described in the 5 section can be found at : `https://github.com/DeNeutoy/act-rte-inference`. Implementations of the model classes can be found in DAModel.py(Decomposable Attention Model (Parikh et al., 2016)), AIAAModel.py(Adaptive Iterative Alternating Attention Model) and ADAModel.py(Adaptive Decomposable Attention Model) respectively. The Stanford Natural Langugage Inference Corpus (Bowman et al., 2015) can be downloaded from `http://nlp.stanford.edu/projects/snli/`.

Chapter 2

Neural Networks

As all of the state of the art methods for Recognising Texutal Entailment involve Neural Networks, we look into the history and applications of this class of function approximators in detail. We describe different architectures, training regimes and look at some examples which are relevant to the project in detail, such as learning word representations.

This section discusses in depth the history and use of neural networks for function approximation. Neural networks have recently again come to prominence, for two main reasons: the advent of highly efficient Graphics Processing Units(GPUs) and work by many researchers, particularly Geoffrey Hinton, Yoshua Bengio and Juergen Shmidhuber on effective ways to train deep neural networks.

2.1 Background and History

Neural networks originate from the multi-layer perceptron model, first discussed in (Rosenblatt, 1958) in order to provide an alternative to formal logical systems and boolean algebra for modelling interactions in the brain, with the specific objective of modelling multiply-connected neurons and the brain's apparent plasticity. The perceptron model builds on the Linear Threshold Unit, introduced in (McCulloch, Pitts, 1990), by extending to non-binary domains and activation functions. The multi-layer perceptron models the idea that neurons in the brain are interconnected and their activation is a function of their inputs.

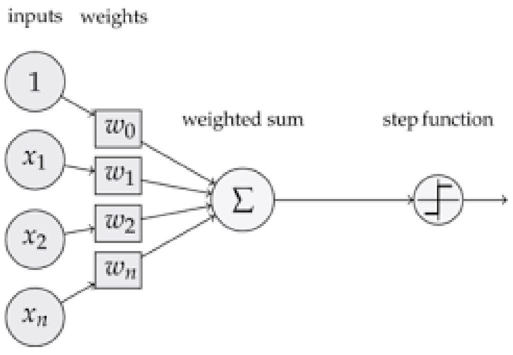


Figure 2.1: A visual demonstration of a single perceptron. A series of weights act on the inputs to the neuron, which are then passed through an activation function leading to the output.

This architecture can then be combined into multiple layers, which are all connected together. The outputs of the first layer of perceptron activations is used as the input for the second layer and so-on. These models developed as a way to enforce a structural prior on the fact that humans view many problems as being composed of a hierarchical set of sub-problems. Models such as the multi-layer perceptron, RBMs and CNNs all involve multiple layers, where the expectation is that the lower levels will model low-level features, which are then composed in to more and more complex features of the data as we progress through the layers. However, on scaling this model, considerable practical difficulties were encountered during training, rendering it of little use in practice.

Compounding this difficulty of training these multi-layered architectures was a fundamentally mis-inturpreted book, *Perceptrons* (Minsky, Papert, 1987). Poor interpretations of theorems in this book, such as a proof that a 3-layer perceptron cannot learn the XOR logical function(actually, the theorem prooves that a 3-layer perceptron cannot learn the XOR function unless one of the input neurons is connected to every neuron in the second ”hidden” layer with a non-trivial weight, demonstrating that local firing patterns were not sufficient to learn complex functions) drove the AI community away from general function approximation and towards more symbolic approaches and is generally referred to as the ”AI Winter”.

The interest in deep neural networks extends from the Universal Function Approximation theorem (Hornik et al., 1989), which states that a feedforward network with a single hidden layer and a bounded, monotonically increasing and continuous activation function can approximate any (reasonably well defined) continuous function in $\mathbb{R}^n$. However, the theorem makes no statement about how easy the parameters of such a feedforward net are to learn and exploration into deep neural architectures demonstrated empirically that adding layers makes it easier to learn better approximations to a wide range of functions and distributions.

More recently in the early 2000's, the interest in neural networks was revived, mainly due to Layerwise pre-training of multi-layer neural networks (Bengio et al., 2006) and the rediscovery of training via backpropagation(Rumelhart et al., 1988) deriving methods to make the training of neural networks effective. Additionally, the availability of large datasets and cheap, fast Graphics Processing Units for computation made it possible to train larger models, drastically increasing performance.

Further innovations in model design led to investigations into "deep" models with many hidden layers generating very promising results in the areas of Machine Vision and Speech Processing. This initial fast progress was due partly to idea transfer from signal processing, such as convolutions and fourier transforms providing a strong basis for exploration, and partly due to the existence of large datasets, such as ImageNet (Deng et al., 2009) and MNIST (Lecun et al., 1998), which were of a suitable size to facilitate training of deep neural networks. Additionally, the fast progress of regularisation techniques such as dropout (Srivastava et al., 2014), advances in first order optimisation techniques and relatively simple marginal increases gained from scaling conv-nets meant that ideas which improved the training of these networks could lead to relatively large performance gains.

These advances were less rapid in Natural Language Processing due to the difficulty with training Recurrent Neural Networks (Bengio et al., 1994), designed to explicitly model non-i.i.d data and temporal dependencies. Additionally, the nature of problems in NLP means that basic elements in a neural network model which are taken for granted in a CNN, such as a fixed input size(cropped images of the same dimensions) do not necessarily apply. Additionally, CNNs act on input and output domains which are the same(RGB colour space) and importantly, continuous. Given the compositional nature of language, this continutity doesn't really apply and has been the source of much work on how to represent words, such as (Mikolov et al., 2013b). Furthermore, Natural Language Processing problems can scale in a way that explode exponentially, or are actually intractable, such as finding a sequence of words which maximises a likelihood function, as locally maximal choices of words may not correspond to globally optimal ones; this is a artifact of the non-continuous nature of natural language.

One of the first examples of deep architectures being applied to NLP was the use of CNNs for generating sentence representations using 1D - convolutions over words in a sentence (Kalchbrenner et al., 2014). Representation of sentences using a vector space model continues to pose a strong problem due to the number of possible sentences and the compositional nature of language.

However, deep neural networks are beginning to emerge in many areas of NLP as the leading models, such as Tree-structured RNNs for sentiment mining (Socher et al., 2013), sequence to sequence models for machine translation and parsing (Bahdanau et al., 2014; Vinyals et al., 2014) and Machine Reading (Sordoni et al., 2016a).

Neural networks are useful because they act as complex, non-linear feature extractors. This often made explicit in convolutional networks, as the learnt parameters correspond to ”filters”, similar to those occurring in biological vision literature. In natural language processing, these features are composed into representations which are used recurrently through processing.

2.2 Error backpropagation and calculating Loss

Backpropagation is an algorithm for efficiently calculating the derivative of a function with respect to it's variables. In the context of training a neural network, we take this function to be an error function describing the difference between the network outputs and the true value. The back propagation algorithm is then used to update the parameters(”weights”) of the neural network such that this loss function is minimised.

2.2.1 Cross-Entropy Loss

Before we describe the algorithm, we first describe the loss function that we will be using in this thesis, given the discrete nature of the task. A commonly used loss function for classification tasks is cross-entropy loss. This loss function is derived from the KL Divergence:

$$KL(p||q) = -\sum_{x} p(x) log q(x) + \sum_{x} p(x) log p(x) \tag{2.1}$$

which, although non-symmetric(and hence not a metric), describes the "distance" between two probability distributions, $q(x)$ and $p(x)$. In our context, $q(x)$ is parametrised by the distribution over the categorical classes generated from the output of a neural net via a softmax function and $p(x)$ represents the empirical distribution over the classes for the actual data(note that this is often "one-hot" - ie a particular example is classified into exactly one discrete class).

In this case, we note that the second summation in the KL divergence is only a function of the empirical distribution $p(x)$, which we have no control over. Therefore, with respect to the distribution we are generating, $q(x)$, this second term is a constant. Removing this term gives us the expression for cross-entropy loss:

$$\mathscr{L}(\theta, D) = -\sum_{D}\sum_{x} p(x) log q(x, \theta) \tag{2.2}$$

where θ are the parameters of the neural network generating the probability distribution and D is the training data, in the form of (example,label) pairs. This formulation will be used for our RTE models where the categorical classes are $\{$*entailment, neutral, negation*$\}$.

2.2.2 Backpropagation

The backpropagation algorithm uses dynamic programming to break the derivative of a function made up of the composition of many other functions of large numbers of parameters down in to a two sweeps through a computation tree where the nodes in the tree represent functions and edges represent inputs to those functions. First, the function is evaluated - the forward pass through the computation tree. The gradient of every parameter can then be computed by multiplying the gradient of each node with respect to it's inputs on a path between the loss node and the variable node. The reason this works is due to the way that the chain rule decomposes.

Suppose we have a function $F(x_1...x_T)$ made up of a computation tree with n nodes, $L_1...L_n$, with leaf nodes taking as input T variables $x_1...x_T$. The derivative of a single node with respect to the overall function F only requires the gradients further up in the tree, as demonstrated in this recursive decomposition of the chain rule:

$$d_t = \sum_{c \in child(L_n)} \frac{\partial L_n}{\partial L_c} t_c \tag{2.3}$$

where the gradient of F with respect to a node $L_t$is equal to d_t.This means that gradients can be kept track of cumulatively for each branch of the tree, meaning each node is only evaluated once. Therefore, the backpropagation algorithm has linear time complexity with respect to the number of parameters of the function, making it extremely fast, even for very complex functions, like neural networks.

2.2.3 Gradient Descent

Gradient descent is a parameter update rule which minimises a function $F(D,\theta)$, where θ are the parameters of the function and D is a set of training examples (x,y). Parameters are updated as follows:

$$\theta_t = \theta_{t-1} - \alpha \nabla F(D,\theta) \tag{2.4}$$

where α is a learning rate hyperparameter. This is guaranteed to converge to the minimum value of the function under certain conditions. Many extensions to Vanilla gradient descent exist, such as utilising second order gradient derivatives, such as Newton's method and Momentum. In this thesis, we use the ADAM(Adaptive Moment Estimation) (Kingma, Ba, 2014) optimisation method. This is an advanced optimisation technique which uses per-dimension smoothed learning rates, bias-correction and momentum and has been shown empirically to be a good optimiser for neural network architectures.

2.3 Feedforward Networks

Feedforward networks refer to neural networks which act on independent and identically distributed data and are static, in the sense that they do not have any recurrent aspect. Networks consist of ”layers of neurons” with weights on connections between the layers generating representations of the input through non-linear transformations. Although there are many types of feedforward networks, we discuss two classes of the historically important and commonly used ones below in Restricted Boltzman Machines and Autoencoders respectively.

2.3.1 Restricted Boltzman Machines

Restricted Boltzman machines (Welling et al., 2004) are a class of probabilistic latent variable graphical models consisting of only two layers - one visible and one hidden and one of the first functional shallow neural network architectures. The network architecture is exactly a fully connected bipartite graph and forms the building block for stacked RBMs and Deep Belief Networks.

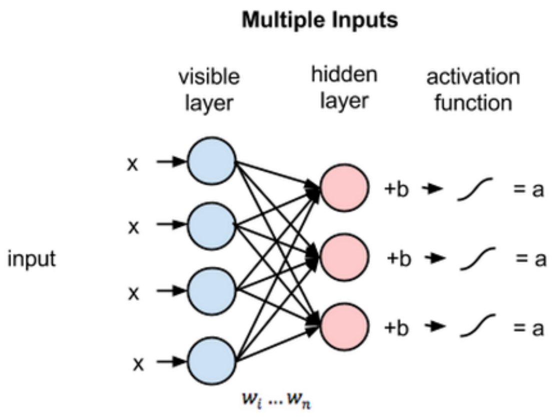


Figure 2.2: Visualisation of the structure of a Restricted Boltzman Machine. It is Restricted as there are no inter-layer connections.

RBM’s are trained in an unsupervised way, in which the aim is to generate a ”compressed” representation of the input such that if the hidden activations are then used as inputs to the same network in reverse, the square loss reconstruction error is minimised.

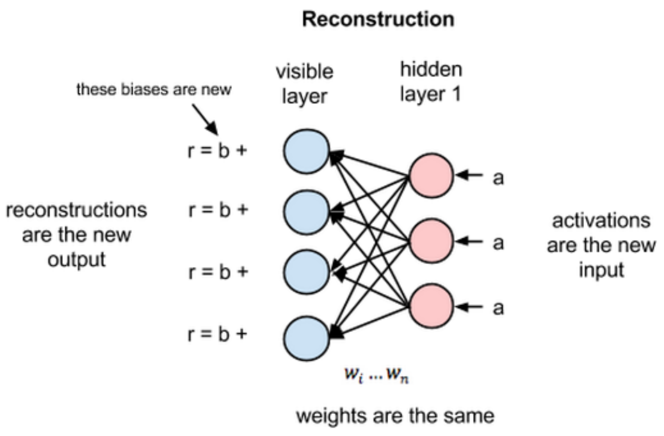


Figure 2.3: Demonstration of the outputs of a forward pass through a RBM being used to reconstruct the original input.

RBM's were originally trained using a method called Contrastive Divergence (Hinton, 2002), which involves using Markov Chain Monte Carlo sampling to approximate the gradient of the log-likelihood objective function with respect to the parameters of the model, in this case the weights on the connections between the observed and hidden layers. This gradient is difficult to compute exactly as it involves computing a Normalisation constant which is intractable for even a small number of dimensions.

There are several important points to note on RBMs which distinguish it from other reconstruction based approaches to dimensionality reduction, such as PCA. Without a non-linear function at it's output, the best solution achievable corresponds to the solution obtained via PCA, as this is provably the linear function which captures the maximal amount of variance in the input for a given dimension (Bourlard, Kamp, 1988). Additionally, a RBM must have fewer hidden nodes than training examples, as otherwise it is possible to exactly recreate the entire training set.

RBM's generalise to deeper models in the form of Deep Belief Networks. In this scenario, DBNs are formed of multiple hidden layers, where the input to the next layer is the output of the previous layer. However, these models are unsuitable for training directly via contrastive divergence as the variance of the gradient approximations becomes large due to the instability of multiple Gibbs samples. (Hinton et al., 2006) and (Bengio et al., 2006) subsequently introduced methods using tied weights which are recursively un-tied and a method for greedy layerwise training which was the first example of an efficient method to train a ”deep” neural network.

2.3.2 Autoencoders

Deep belief networks are very similar in design to autoencoders with the exception that in autoencoders, the parameters used to compress the input and the weights used to generate a reconstruction from that compressed representation are not shared.

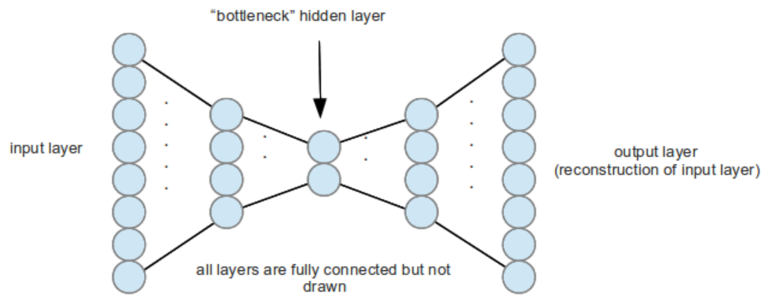

"bottleneck" hidden layer
input layer
output layer
(reconstruction of input layer)
all layers are fully connected but not drawn

Figure 2.4: Graphical representation of an autoencoder architecture. The middle set of nodes are of a smaller dimensionality, requiring the compression of the input.

Autoencoders are often used to learn ”representations” of inputs which are of a lower dimensionality than the original inputs. As with many unsupervised learning algorithms, the objective is minimise the square reconstruction error between the inputs and the recreated inputs.

It is classically required that the dimensionality of the smallest layer in an autoencoder is strictly less than that of the inputs in order to prevent the network simply learning the identity function in order to exactly recreate the inputs. However, in practice it seems like this is not the case and other representations are learnt (Bengio, 2009), although as lossy compression is generally acceptable and often in many problems, this heuristic is often employed.

Several variants of autoencoders have been proposed in order to more accurately extract useful features. Some of the more successful/notable ones include:

- **De-noising Autoencoders** use corrupted inputs whilst still attempting to reconstruct the original, un-corrupted image. This idea is based on the fact that high level representations should be robust to noise in the data, meaning overfitting to particular training set features is less likely. This type of structure is also used to de-noise images.

- **Sparse Autoencoders** use an additional regularisation term in the loss function which is minimised which encourages the weights to be small. *K-sparse autoencoders* threshold weight activations, zeroing out activations below some small epsilon value. This can result in sparser representations of the input, desirable when compression is the main objective.

- **Variational Autoencoders** (Kingma, Welling, 2013) assume that the data **X** is generated by sampling from a latent variable distribution $p(X|z)$. An approximation to the posterior distribution is then minimised.

2.4 Recurrent Neural Networks

Recurrent Neural Networks are used to model temporal dependencies in data and act over sequences of vector inputs. In the case of natural language processing, these vectors are generally vector representations of words, with the sequence representing a sentence. These types of neural networks will form the basis for generating representations of the sentences used in our task.

Given an input vector $x_t \in \mathbb{R}^N$ and the previous output $h_{t-1} \in \mathbb{R}^K$, we compute the output at time t as:

$$H = \begin{bmatrix} x_t \\ h_{t-1} \end{bmatrix} \in \mathbb{R}^{N+K} \qquad h_t = \sigma(W_l H + b_l) \tag{2.5}$$

Where W_l is a learned weight matrix and b_l a learned bias vector. Recurrent Neural Networks can be viewed either as a single network in which outputs from one timestep are used in the next, demonstrating the recurrent aspect, or unrolled for a full sequence, which is useful as it visualises all of the outputs generated from running an RNN over a sequence.

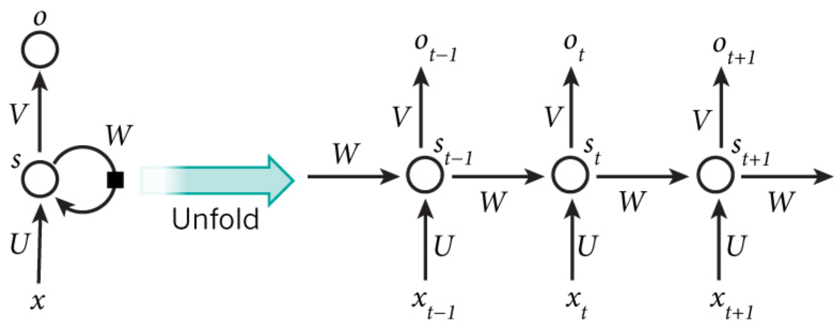


Figure 2.5: Visualisation of the two ways to inturpret an RNN cell.

Although Recurrent Neural Networks are provably Turing-Complete(i.e they can simulate a single tape Turing Machine) (T Siegelmann hava et al., 1991), in practice they are difficult to train as they suffer from ”catastrophic forgetting” induced by the exploding/vanishing gradient problem. In the next sections, we discuss specifically why this problem arises and introduce the Long Short Term Memory architecture, designed specifically to counter this problem.

2.4.1 The Exploding/Vanishing Gradient Problem

Back-Propagation Through Time can be viewed as a simple ”unrolling” of a recurrent neural network into a very deep network(as deep as the sequence length), with shared parameters at each layer. As backpropagation is effectively a recursive application of the chain rule, deep Recurrent Neural Networks suffer from catastrophic forgetting (Hochreiter et al., 2001), due to the multiplicative nature of the chain rule and the large areas of near-zero gradient in commonly used non-linear activation functions, such as the sigmoid function.

In order to clearly demonstrate the problem at hand, consider a loss function $L(W,x)$ and a vanilla recurrent neural network $F(x,s_{t-1}) = \sigma(W[x,s_{t-1}]+b)$. For a sequence of length t,

$$\frac{\partial L}{\partial W} = \sum_{i=1}^{t} \frac{\partial L}{\partial F_t} \left(\prod_{k=i+1}^{t} \frac{\partial F_k}{\partial F_{k-1}} \right) \frac{\partial F_i}{\partial W} \tag{2.6}$$

Where F_k represents the k-th application(or unrolled depth) of the recurrent neural network. In the case of $i = 1$, or the first application of F, we see that:

$$\frac{\partial L}{\partial F_t} \prod_{k=2}^{t} \frac{\partial F_k}{\partial F_{k-1}} \frac{\partial F_1}{\partial W} = \frac{\partial L}{\partial F_1} \left(W^{T^{t-1}} \prod_{k=2}^{t} diag(\sigma'[x_k, s_{k-1}]) \right) \frac{\partial F_1}{\partial W} \rightarrow 0/\infty, t \rightarrow \infty \tag{2.7}$$

where *diag* represents a vector specified as a diagonal matrix. This demonstrates that the error gradient with respect to the weight matrix W vanishes or grows exponentially fast as the sequence length grows, depending on the size of the largest eigenvalue of the W parameter.

The exploding gradient problem is fairly easily resolved - the gradients can just be capped at a certain level in order to prevent exponential growth. The vanishing gradient case was considerably more problematic and was the subject of a considerable amount of work in the early 90s in order to produce a functioning RNN, such as fast weight memories in (Schmidhuber, 1992). This problem was eventually resolved in recurrent neural networks through the introduction of gating functions, such as the LSTM (Hochreiter, Schmidhuber, 1995), which ”protect” error signals by controlling what gradients are propagated using ”forget gates”.

2.4.2 Long Short Term Memory

Recurrent networks with Long Short-Term memory (Hochreiter, Schmidhuber, 1995) have been successfully applied in domains such as machine translation (Sutskever et al., 2014), syntactic constituency parsing (Vinyals et al., 2014) and speech recognition (Graves et al., 2013). LSTMs consist of cells which selectively store information for long periods of time, using three gates to control the information which is stored. We call this a ’Vanilla’ LSTM as it does not include other innovations in structure, such as peephole connections (Gers, Schmidhuber, 2000), as often they have been demonstrated to give only incremental, or neglible improvements (Greff et al., 2015) and their removal greatly simplifies the architecture. Below we present a formulation which is optimised to perform the linearities within the gate calculations simultaneously using concatenated matrices.

Given an input vector $x_t \in \mathbb{R}^N$ at time t and the previous output of the LSTM

$h_{t-1} \in \mathbb{R}^K$, we compute the forget gate(f_t), the input gate(i_t), the output gate(o_t) and the cell input(z_t) as follows:

$$
\begin{aligned}
H &= \begin{bmatrix} x_t \\ h_{t-1} \end{bmatrix} \in \mathbb{R}^{N+K} & o_t &= \sigma(W_o H + b_o) \\
& & f_t &= \sigma(W_i H + b_i) \\
g_t &= Tanh(W_g H + b_g) & c_t &= f_t \odot c_{t-1} + i_t \odot g_t \\
i_t &= \sigma(W_i H + b_i) & h_t &= o_t \odot Tanh(c_t)
\end{aligned}
\tag{2.8}
$$

Where $W_g, W_i, W_o, W_f \in \mathbb{R}^{K \times (N+K)}$ and $b_g, b_i, b_o, b_f \in \mathbb{R}^K$ are trained weight matrices and bias vectors for N, the dimension of the input vector and K the hidden size of the LSTM. σ and $Tanh$ are element-wise activation functions representing the sigmoid and hyperbolic tangent functions respectively, with $\odot$ representing element-wise multiplication of vectors.

Note that by stacking the weight matrices for z, i, o and f gates, the linear part of the calculation can be performed efficiently in a single matrix vector product. The resulting vector in $\mathbb{R}^{4K}$ is split and the corresponding non-linearities are applied.

The LSTM architecture can be most easily understood by examining the below diagram and imagining the result when a particular gate results in an all zero output.

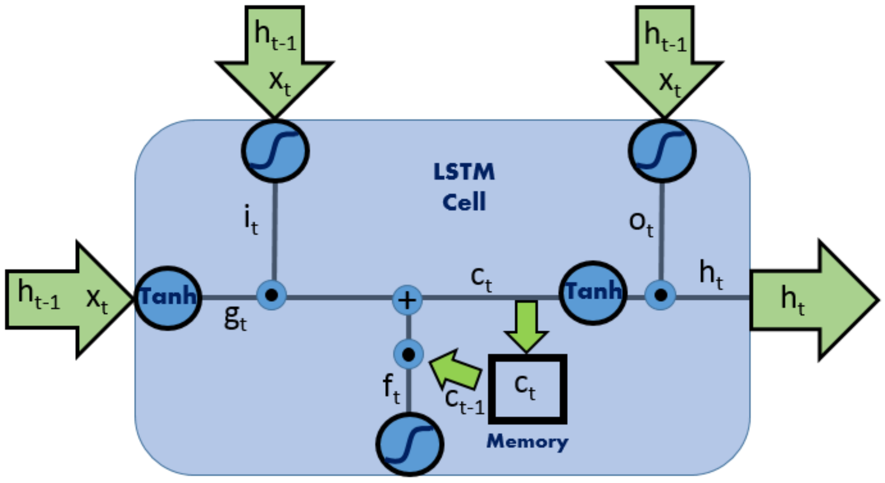


Figure 2.6: A structured visualisation of the gates used in a LSTM cell. First, we apply a non-linearity to the input, resulting in g_t. Next, we multiply by the input gate - if the result of this gate was a zero vector, none of the current input is used in the computation, or ”written” to the memory. Next, we access the cell memory using the forget gate, f_t. Again, if this was a zero vector, none of the memory cell would be used. Finally, we control how much of the current memory and state vector combination we output using the output gate, o_t.

2.4.3 Gated Recurrent Units

Although LSTM units were the first architecture proposed to correct the issue of propagating long term dependencies through recurrent neural networks, the Gated Recurrent Unit (Chung et al., 2014) introduced a simpler formulation, by merging the output and forget gates of the LSTM model.

Given an input vector $x_t \in \mathbb{R}^N$ at time t and the previous output of the GRU $h_{t-1} \in \mathbb{R}^K$, we compute the cell input(z_t), memory reset gate(f_t) and non-linear combination of these as the new hidden state(h_t) as follows:

$$\begin{aligned} H &= \begin{bmatrix} x_t \\ h_{t-1} \end{bmatrix} \in \mathbb{R}^{N+K} & r_t &= \sigma(W_r H + b_r) \\ & & \tilde{h}_t &= Tanh(W_h[r_t \odot h_{t-1}, x_t]) \\ z_t &= \sigma(W_z H + b_z) & h_t &= (1 - z_t) \odot h_{t-1} + z_t \odot \tilde{h}_t \end{aligned} \tag{2.9}$$

Where $W_z, W_r, W_h \in \mathbb{R}^{K \times (N+K)}$ and $b_z, b_r, b_h \in \mathbb{R}^K$ are trained weight matrices and bias vectors for N, the dimension of the input vector and K the hidden size of the GRU. Functions are as described previously.

Throughout the experimentation in this thesis, we use the GRU over the LSTM, due to the simplifications arising from not having a separate memory vector c_t to carry through recurrent computations. All current comparative literature on low level recurrent cell structures (Józefowicz et al., 2015; Greff et al., 2015) demonstrates empirically that the only design feature having a significant impact on performance is the presence and initialisation of a forget/reset gate and hence this distinction will not be remarked upon in further analysis.

2.4.4 Bi-Directional RNNs

Bi-directional RNNs (Schuster, Paliwal, 1997) are a simple extension to RNN architectures introduced in order to address the fact that in many applications, feeding the word representations of a sentence into an RNN from left to right is an arbitrary choice. The outputs and states for a bidirectional RNN are formed simply by concatenating the representations from two recurrent models run independently over the input, with one receiving inputs in the forwards direction and the other in the backwards direction. Bidirectional RNNs have provided more informative representations for solving a variety of tasks, as demonstrated by their use in state of the art models for POS tagging (Wang et al., 2015) and Machine Reading (Sordoni et al., 2016b).

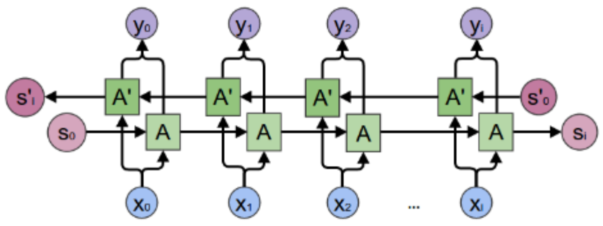


Figure 2.7: Diagram demonstrating the dual processing of a sequence using a bi-directional RNN. (Colah, 2015)

Although bidirectional RNNs have a demonstrated ability to carry more information in their latent representations, it should be noted that the dimensionality of the hidden state for any downstream tasks is necessarily doubled as a result of the state concatenation, which can result in a serious slowdown in training time and increase in complexity. Also, it is worth noting that bidirectional RNNs require a static, pre-defined input and hence cannot be used in an online setting.

2.5 Unsupervised Word Representations

Word representations describe the process of embedding a vocabulary of words into a high dimensional vector space, with one vector per word, whilst ensuring that the representations retain semantic and syntactic information. Capturing semantic structure is a difficult task given that word use and meaning is constantly evolving and is heavily dependent on many linguistic subtleties, such as irony, co-reference and metaphor. Despite this, techniques relying the fundamental premise of statistical linguistics, the *distributional hypothesis*, have proved successful. The distributional hypothesis states that *"linguistic items with similar distributions have similar meanings"* (Harris, 1954).

2.5.1 Global Methods

Before the introduction of local methods enabling much more efficient training, the complexity of standard methods involving Singular Value Decomposition and Non-Negative Matrix Factorisation(Probabilistic Latent Semantic Analysis) scaled with the size of the corpus and vocabulary. Latent Semantic Analysis (Deerwester et al., 1990) involves factorising a co-occurrence matrix of a vocabulary and n-word contexts. Once this factorisation is obtained, low-dimensional representations of words can be extracted by taking the first k eigenvectors corresponding to the k largest eigenvalues. Mathematically, this method is guaranteed to minimise the reconstruction error(a simple result derived from PCA), as the singular values represent the directions of largest variance when the co-occurrence matrix is modelled using the new basis.

However, both these methods suffer from computationally intensive training, as the time complexity of SVD is $O(n^3)$ and large vocabularies or training datasets (this increases the number of observed contexts) are required to obtain results which encode accurate approximations of distributional similarity. Additionally, the probabilistic version of this algorithm often assumes a Gaussian prior on word frequency, but words in a vocabulary more often take on a Zipfian(power-law) distribution. Similarly, Latent Dirichlet Allocation (Blei et al., 2001), a hierarchical 3 - layered Bayesian mixture model, is more commonly used in topic modelling and hence the low-dimensional representations of the vocabulary are not necessarily optimised towards having a global semantic consistency, but more towards enabling accurate categorisation of a finite number of document topics.

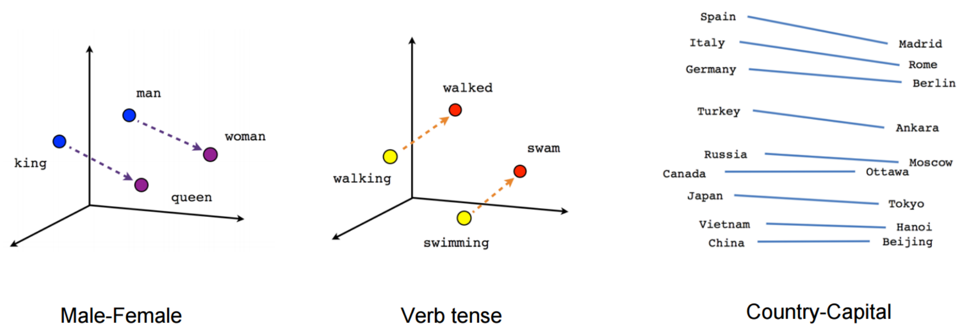


Figure 2.8: Visualisation of the linear structures present in vector representations of words learnt using word-context entropy minimisation algorithms. These linear structures provide interpretable results from simple algebraic manipulations of the representations.

2.5.2 Local Context/Prediction Methods

In contrast to global methods, local context and prediction methods utilise a sliding "context window" to formulate a prediction problem of learning $P(w|c)$ where w is a word in the vocabulary and c is an observed context window. Bengio's Neural Probabilistic Language Model (Bengio et al., 2003) was the first way of learning representations using such a method. After initialising a h-dimensional random vector per word in the vocabulary, a context representation is formed using a single feed-forward neural network to sequentially process the context. This context representation is then fed into a softmax function to generate a probability distribution over the entire vocabulary, in order to predict the next word. Training uses error back-propagation (Rumelhart et al., 1988) to incrementally minimise the cross-entropy loss, equivalent to Maximum Likelihood Estimation.

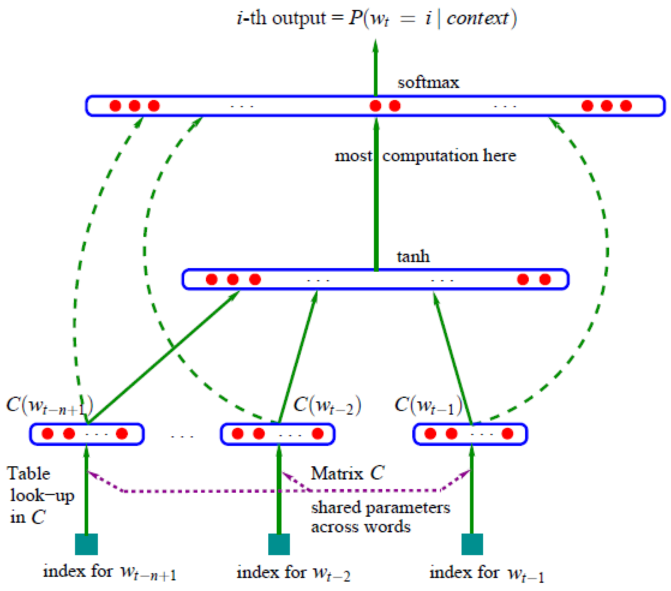


Figure 2.9: Model architecture from (Bengio et al., 2003). Note that this is not a Recurrent Neural Network - hidden representations generated from a feed-forward network are simply concatenated, with a non-linear function applied before the application of the softmax function.

Although the continued development of computational resources and optimisation techniques such as sampled/tree-structured softmax functions have made training neural models tractable in recent years, we would ideally like to generate word representations trained on billions of words. Given the generation of (context, word) pairs for a supervised training objective can be done in an effectively unsupervised manner and huge amounts of data is readily available, the bottleneck for training these word representations is one of computational resource/model complexity rather than unavailability of training sets.

To this end, we will now examine in detail the skipgram architecture (Mikolov et al., 2013b,a) which utilise a log-bi-linear model, along with negative sampling, a technique derived from Noise-Contrastive Estimation (Gutmann, Hyvärinen, 2010) to derive an approximation to the Maximum Likelihood objective which can be optimised efficiently.

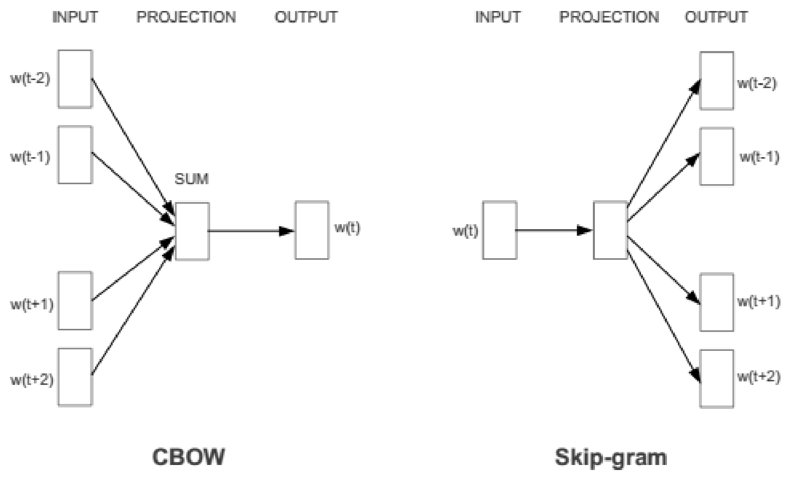


Figure 2.10: Visualisation of the CBOW and Skipgram architectures proposed in (Mikolov et al., 2013b,a)

Given a corpus of text, we are concerned with estimating the probability $p(w|c)$ for words w and their contexts, c.

$$\arg\max_{\theta} \left\{ \prod_{(w,c) \in D} p(w|c;\theta) \right\} \tag{2.10}$$

where D is the set of all (word, context) pairs extracted from the corpus, given some pre-defined word window size, normally 4.

A standard way to parameterise the conditional probability in neural network literature is via the softmax function:

$$p(w|c;\theta) = \frac{e^{v_c \cdot v_w}}{\sum_{\hat{c} \in C} e^{v_{\hat{c}} \cdot v_w}} \tag{2.11}$$

where $v_c, v_w \in \mathbb{R}^d$ are distinct vector representations of each word $w \in V$ and context $c \in C$. The collection of these vectors form the set of learnable parameters θ, resulting in $|V| \times |C| \times d$ parameters.

Note that this objective function is expensive to optimise due to the sum over all contexts for every conditional distribution over words. This can be approximated using hierarchical or tree-structured softmax functions, where a subset of the contexts are used to approximate the normalisation factor in the denominator of the softmax. Instead, (Mikolov et al., 2013b) used a version of NCE, negative sampling to optimise a different objective.

The key observation used in negative sampling is that given a large enough corpus, if a context c is *randomly sampled* from all possible contexts C, it is likely to not make sense. This can be used to define a function related to the joint probability, $p(w,c)$, which jointly maximises the probability $p(w|c)$ of pairs from the corpus and minimises the probability $p(w|\hat{c})$ when $\hat{c}$ is a randomly sampled context.

$$p(Observed = 1|w,c;\theta) = \sigma(v_c \cdot v_w) \qquad p(Observed = 0|w,c;\theta) = 1 - \sigma(v_c \cdot v_w) = \sigma(-v_c \cdot v_w) \tag{2.12}$$

where σ is the sigmoid function and a pair (w,c) is observed if it appears naturally in the training data. This allows us to define the objective(additionally converting to log space):

$$\arg\max_{\theta} \left\{ \sum_{(w,c)\in D} log(\sigma(v_c \cdot v_w)) + \sum_{(w,c)\sim(U(V),U(C))} log(\sigma(-v_c \cdot v_w)) \right\} \tag{2.13}$$

where $(w,c) \sim (U(V),U(C))$ denotes sampling a word and context uniformly and independently from the empirical distributions over the word vocabulary V and set of all contexts C respectively.

Note that negative sampling here is essential to removing the ability to find a trivial solution - without the joint necessity to minimise the probability of the negative samples, we could simply set the parameters to any large, very positive value and quickly the sigmoid function will converge to 1, saturating the sigmoid function and minimising the negative log likelihood. Therefore, the use of negative sampling is key to forcing the objective to also minimise the probability of unlikely (w,c) pairs.

Additionally note that this objective function differs slightly to the original presented in (Mikolov et al., 2013b) as they present the objective for a single (w,c) instance, as well as making use of optimisations effectively unrelated to the overall objective, including temperature sampling for drawing negative examples, word sub-sampling, dynamic window sizes and a hyperparameter k denoting the ratio of negative samples drawn.

Interestingly, it has been proven that both the Skipgram and CBOW models are actually implicitly factorising a word-context matrix of Pairwise Mutual Information (Church, Hanks, 1990) shifted by a global constant when the dimensionality of the word vectors used is large enough to allow perfect reconstruction (Levy, Goldberg, 2014). For low dimensional word representations which cannot exactly reconstruct the word-context PMI matrix, this shifted PMI metric becomes additionally weighted by the frequency of (w,c) occurrences.

Exact solutions to the decomposition of a similar matrix factorisation problem via SVD show poorer performance than using either of Mikalov's model, demonstrating that the better representations from Skipgram/CBOW extend directly from weighted loss dependent on word frequency - a particular problem with empirical PMI is its instability under low observation counts, so this is clearly a sensible alteration to the objective function. This was an unusual conclusion, as it was thought that the shallow neural architectures proposed by Mikalov were a departure from the standard matrix factorisation approaches to constructing word embeddings, such as LSA.

2.5.3 Compositionality and issues

Skipgram and other log-bilinear models provide extremely good vector representations of single words. However, as we learn a vector representation per word, this approach does not scale to learning sentence representations, due to the exponential increase in the number of possible sentences.

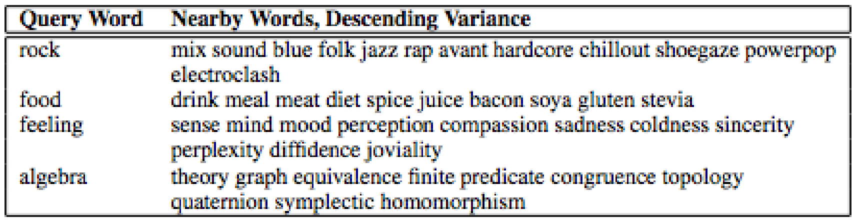
Therefore, it would be convenient if word embeddings could retain some form of compositionality - i.e representations of sentences bearing the same meaning which are composed of functions of their respective word embeddings are near each other in a sentence embedding space. Indeed, even when the composition function is relatively naive, such as taking sums of the word vectors of a sentence as its representation, this can generate reasonable results, particularly for short sentences expressing fairly simple concepts.

However, this element-wise composition falls down due to the observation that linguistic rules and meaning rarely fit in to a precisely defined linear framework. Transitivity and associativity are widely assumed to imply a correct logical structure, but even the simplest of linguistic structures do not satisfy this requirement - take for example a sentence which can be ambiguously parsed, or the fact that adjective negations are not symmetric.The way such a sentence is composed is not only dependent on word order, but also on the surrounding context.

An additional problem with context based representations is that they do not resolve linguistic phenomona such as antonymy and hypernymy. As the contexts of words which are antonyms(such as ”hot” and ”cold”) are often similar, the vectors for the resulting words will be near each other in the vector space. For some purposes, this is actually desirable, but it can often cause systems to struggle with fine grained inference involving grouped concepts, such as colours. Several attempts have been made to address this problem, using techniques including generating multiple vectors for different sense-disambiguations of a single word (Huang et al., 2012).

Although word representations appear to be consistent with the distributional hypothesis, they do not retain more fine grained semantic information such as hyponymy or hypernymy. As a result, word-embeddings are relatively poor at predicting these types of relations and it is clear that this would be the case, given that we have already discussed the problem of representing transitive relations, which includes both hypernymy and hyponymy(if a cat is a mammal and a mammal is an animal, a cat must also be an animal).

Attempts have been made to improve the ability of word representations to include ”soft” structural constraints, such as modelling word embeddings not as fixed point vectors, but as densities, using k-dimensional multivariate Gaussian distributions (Vilnis, McCallum, 2014). This method allows the modelling of hyponymy relations by using the size of a spherical covariance as a proxy for the size of a word’s semantic field(breadth of meaning). Using an energy minimisation approach to training, they demonstrate that distributions representing hyponyms of a word lie completely within it's spherical covariance.



Query Word	Nearby Words, Descending Variance
rock	mix sound blue folk jazz rap avant hardcore chillout shoegaze powerpop electroclash
food	drink meal meat diet spice juice bacon soya gluten stevia
feeling	sense mind mood perception compassion sadness coldness sincerity perplexity diffidence joviality
algebra	theory graph equivalence finite predicate congruence topology quaternion symplectic homomorphism

Figure 2.11: Examples of nearest neighbours sorted by decreasing spherical covariance from (Vilnis, McCallum, 2014). Note that more specific words have smaller covariances.

Other, more strict methods to force structure into vector representations of concepts, not always words, have proven successful. Particularly, the idea of preserving orderings, or partial orderings explicitly (Vendrov et al., 2015), rather than trying to model the distributional hypothesis has been successful in predicting hypernymy and image caption retrieval. Additionally, in (Demeester et al., 2016), they demonstrate how logical rules in a knowledge base can be represented via an ordering in latent space, ensuring that inferences used to complete the knowledge base are consistent with a set of prior logical rules. This demonstrates that purely similarity based embeddings are not the only way to extract meaning from latent vector spaces.

Overall, unsupervised word embeddings are one of the most successful instances of transfer learning - vector representations of words in a corpus form the starting point of a large number of NLP tasks and are widely considered to help with over-fitting(if they are not treated as parameters during training).

2.6 Attention Mechanisms

Attention mechanisms originate in machine vision, first introduced to emulate human eye tracking for image generation using Restricted Boltzman Machines (Larochelle, Hinton, 2010). Attention began to gain momentum in NLP after the introduction of sequence-to-sequence models, primarily used in machine translation to address the alignment problem (Sutskever et al., 2014). The fundamental bottleneck in sequence to sequence models is the idea that the input is encoded,

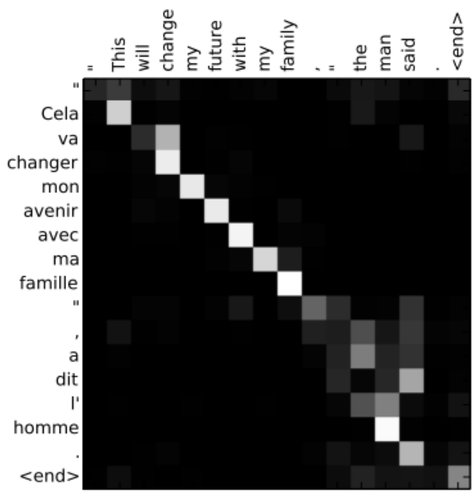


Figure 2.12: Demonstration of the alignments generated by the attention mechanism in (Bahdanau et al., 2014). The weighted sum of the source sentence using these weights is used as an input to the decoder as it generates words in the target sentence as output, following the "sequence to sequence" learning pattern introduced in (Sutskever et al., 2014).

using a RNN, into a single vector representation, which is used to generate a conditional probability distribution over a vocabulary, sequentially generating a translation. Given the complexity involved in translation, this single vector must encode not only individual word meanings, but relations between words and other semantic information.

Attention was introduced in order to resolve this bottleneck, enabling representations used for generating distributions over words at decoding time to be generated by a non-linear combination of the current decoder state and a weighted sum of the encoded input representations (Bahdanau et al., 2014). Attention has been successfully applied to sequence problems in many domains, including parsing, machine reading and translation (Bahdanau et al., 2014; Vinyals et al., 2014; Hermann et al., 2015), generating state of the art results.

One additional factor in the popularity of attention mechanisms is their contribution to the inturpretability of neural models. The distributions over inputs tend to show "natural and intuitive" attention whilst generating outputs, often resolving multi-word synonyms or co-location.

2.6.1 Description

There are several expositions of attention mechanisms, using different combinations of non-linear functions or pre-processing the input to the softmax in some way, such as via a feed-forward network. In principle, all of these attention mechanisms share an underlying idea, which is creating a conditional distribution over a sequence of inputs, given a query. Below we give a description which generalises all of these mechanisms to a weighted sum of state representations given a probability distribution generated by a context function.

Given a sequence of input representations $\{h_1....h_n\}$, such as the outputs from an RNN model over a premise sentence in entailment, or the source sentence in a translation problem, we compute the attention representation c_i given a query vector q_i (such as the final hidden state from encoding the hypothesis in RTE, or the current state representation in a translation decoder RNN) as follows:

$$c = \sum_{j=1}^{n} \alpha_j h_j \qquad \alpha_j = \frac{e^{a_j}}{\sum_{k=1}^{n} e^{a_k}} \tag{2.14}$$

Where a_{ij} is the result of a context function. The context function varies depending on the application, but is normally a feed-forward network, taking as a query q_i and the input representation h_j to produce the scalar valued a_{ij}. In various applications, the query q_i may be a hidden state of a RNN Decoder, or a sentence representation.

This arbitrary context function demonstrates why attention is so flexible and applicable in many domains. Attention is also naturally applicable to domains which are tree structured in nature, such as document summarisation and text classification. Attention can be hierarchically employed on a word, sentence and document level independently, producing multi-level attention based representations of documents producing state of the art performance (Yang et al., 2016).

2.6.2 Word-by-word attention

Word by word attention extends the idea of allowing a representation to be generated by attending over a previous input to include this attention mechanism at each step of the second input. This is the standard approach taken in Machine Translation (Bahdanau et al., 2014).

$$c_i = \sum_{j=1}^{n} \alpha_{ij} h_i \qquad \alpha_{ij} = \frac{e^{a_{ij}}}{\sum_{k=1}^{n} e^{a_{ik}}} \tag{2.15}$$

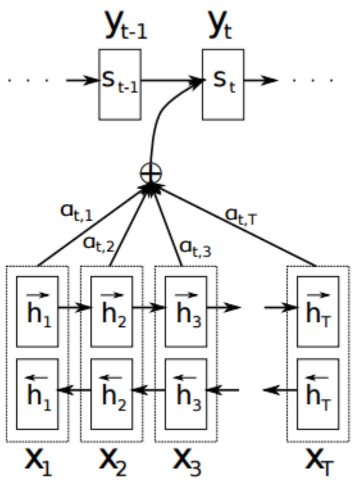


Figure 2.13: Diagram detailing the aggregation method used to attend over the source sentence for Machine Translation (Bahdanau et al., 2014).

Additionally, allowing a model the freedom to attend over an input in a fine-grained fashion has qualitatively been demonstrated to assist the resolution of inferences based on deeper semantic relations, e.g "snow" is found "outside" (Rocktäschel et al., 2015). This is particularly relevant for recognising entailment.

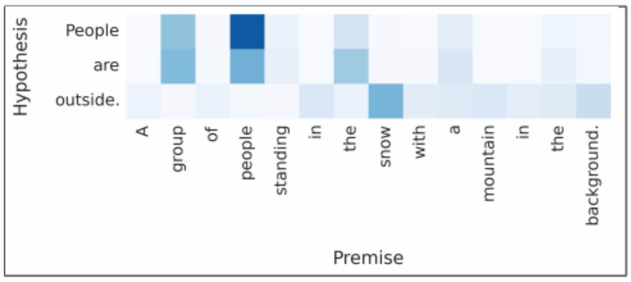


Figure 2.14: Word-by-word attention masks whilst computing the hypothesis representation for RTE (Rocktäschel et al., 2015).

2.6.3 Attention as accessible memory

Attention can be viewed as an expansion on the problem of long-term memory in recurrent neural networks. However, it differs in the sense that it relies on information not carried through the RNN hidden state, and for this reason, it has become an extremely promising area of research. Static and dynamic attention-addressable memory, including both content and location based access, have been detailed in architectures such as Neural Turing Machines and Memory Networks (Graves et al., 2014; Weston et al., 2014), where the specific distinction between these models and previous inference techniques involving external knowledge bases is that they are end-to-end differentiable, so they can be optimised using gradient descent.

Overall, attention has driven large improvements in many areas. However, it does have flaws; in computationally least efficient case(word-by-word attention) it requires $\mathcal{O}(n^2m)$ computations for an input representation $\{h_1...h_n\}$ to generate m output tokens, as in machine translation. Above, we have addressed an effective resolution to this issue, but the case still holds that human attention allows us to expend *less, not more*, memory/time to resolve inferences, which is efficient when the input length grows to be extremely large, such as answering questions from a document.

This bottleneck is partly due to the requirement that the function producing the distribution over the inputs is differentiable. Other methods, which form attention representations detached from the size of the input (Mnih et al., 2014) (prevalent in CV, where the input space is often larger), are not differentiable and are trained using REINFORCE and other algorithms from direct policy-gradient optimisation literature (Sutton et al., 2000).

Chapter 3

Adaptive Computation

In this section, we will demonstrate that many of the methods described in the above section have extensions which build on these models in an adaptive way which in some way allows: more expressive representations, more efficient training, improved generalisation or a combination of the above. We aim to demonstrate why adaptive computation methods are a useful tool and argue that their application should become more widespread.

The idea of adaptive latent variables is very common. Almost every conceived probabilistic model has a partnered infinite dimensional equivalent - mixtures of gaussians become gaussian processes, Latent Dirichlet Allocation becomes Hierarchical Dirichlet Processes. All these ideas were pioneered due to the desire to not restrict a probabilistic model unnecessarily - for instance in topic modelling, the number of topics is a latent variable in HDP and can be optimised within the Maximum Likelihood objective, whereas in LDA, it must be set beforehand.

This desire to incorporate hyperparameter optimisation within a model is the basic drive behind the application of adaptive computation.In this section, we will cover Adaptive Computation Time, a generalisation of a RNN to take multiple timesteps, both demonstrate how adaptive computation can bring inturpretability as well as performance to a neural network based model.

3.1 Adaptive Computation Time

This section introduces Adaptive Computation Time for Recurrent Neural Networks (Graves, 2016). The contribution made by this paper is fundamental to this thesis, as we employ a similar adaptive mechanism in an RNN in addition to attention over two input sequences for classification. The ACT algorithm builds on and combines the idea of a ”self-delimiting neuron” (Schmidhuber, 2012), a binary halting function, with the ”neural stack” architecture in (Grefenstette et al., 2015), a ”soft” version of the push and pop operations on a standard stack data structure.

Adaptive computation is a unique neural network architecture in the sense it has transformed a hyper-parameter attribute, the running-time/depth of the network, into a regulariser added to a loss function and trained with gradient descent methods. ACT addresses the fundamental problem that many machine learning algorithms cannot adapt the amount of time spent on a problem depending on the complexity of a task.

In (Graves, 2016), ACT is evaluated on several toy tasks, including parity checking, addition and sorting, and a more standard language modelling task. In the toy example cases, the problems are biased to give a significant advantage to processing a single input using some recurrent state, as ACT does during a single step. As expected, ACT demonstrates considerable advantages on these problems, as they are designed to show. However, on the language modelling task, performance is not particularly competitive. We argue that there *are* problems which are practically applicable which have a structure in which the use of ACT does confer an advantage, demonstrating the use of this algorithm is not constrained to artificial examples.

In order to determine the number of intermediate steps taken, define:

$$
\begin{aligned}
h_t^n &= \sigma(W_p s_t^n + b_p) & N(t) &= min\left\{ n : \sum_{i=1}^{n} p_t^i \geq 1 - \varepsilon \right\} \\
p_t^n &= \begin{cases} R(t) & n = N(t) \\ h_t^n & otherwise \end{cases} & R(t) &= 1 - \sum_{i=1}^{N(t)-1} p_t^i
\end{aligned}
\tag{3.1}
$$

Where p_t^n is the halting probability at outer timestep t, inner recurrence timestep n, $N(t)$ is the inner recurrence timestep at which the accumulated halting probabilities $p_t^{1 \ldots n}$ reach $1 - \varepsilon$ and $R(t)$ is the remainder at timestep $N(t)$. $W_p \in \mathbb{R}^d, b_p \in \mathbb{R}$ are learnt parameters and σ is the element-wise sigmoid function.

Given the above, we define the next cell output and state to be weighted sums of the intermediate states:

$$s_t = \sum_{i=1}^{N(t)} p_t^i s_t^i \qquad y_t = \sum_{i=1}^{N(t)} p_t^i y_t^i \tag{3.2}$$

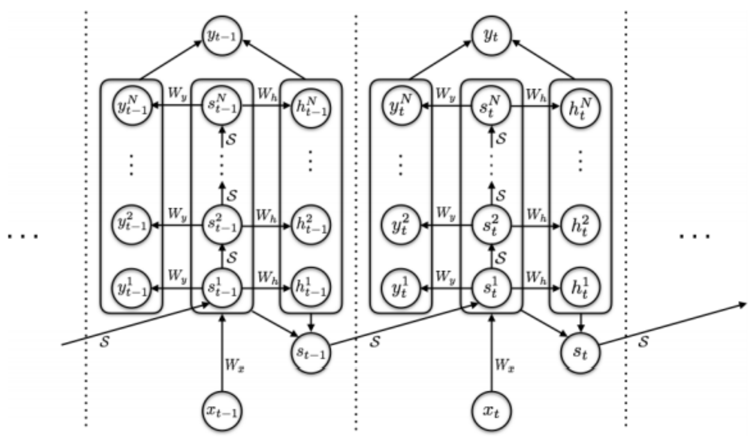


Figure 3.1: Visual depiction of a single ACT step (Graves, 2016). For each input x_t and hidden state s_{t-1} we compute additional RNN outputs $y_t^{1 \ldots N}$ as well as generating halting probabilities $h_t^{1 \ldots N}$ until their sum reaches $1 - \varepsilon$. The actual cell output and state are weighted sums of the intermediate outputs and states respectively.

The final addition to this model is how to regularise the number of computational steps taken. This is done by adding a regulariser, or the *Ponder Cost* to the loss function, made up of a combination of the remainder function and the iteration counter.

$$\mathscr{P}(x) = \sum_{i=1}^{T} N(t) + R(t) \tag{3.3}$$

Note that the actual function which we want to control by adding to the loss function is $N(t)$. Including the $R(t)$ function means we are adding an upper bound to $N(t)$ to the loss function as a trade-off for the function being continuous(and with a non-zero gradient) away from the points where N is incremented. Effectively, this means that the dis-continuities can be ignored at these points by assuming that $N(t)$ is constant, in the same manner used for the RELU activation function (Nair, Hinton, 2010).

This function is then added to the loss function:

$$\tilde{\mathscr{L}}(\theta,X) = \mathscr{L}(\theta,X) + \alpha\mathscr{P}(x) \tag{3.4}$$

where $\mathscr{L}$ is the loss function of the neural network(in our case cross-entropy loss over some number of labelled (x,y) pairs) and α is a timetrade-off parameter; by using a larger value, we encourage the function to take less time over minimising the actual loss.

3.1.1 Similarity to While Loops

Adaptive computation can be seen as a differentiable implementation of a while loop, where the stopping condition for the loop is the same as the stopping condition in ACT. This is useful as it clarifies the idea that we can perform any other useful operations within the body of this while loop, generalising away from the context of a single recurrent neural network and allowing us to have larger architectures, such as modules including attention or extraneous inputs, executing for an input-conditional length of time.

Algorithm 1 ACT as a while loop

```
1: procedure INNER ACT STEP
2:     while $p \leq 1 - \varepsilon$ do
3:         $y_t^n, s_t^n = InnerCell(x_n, s_t^{n-1})$
4:         $p_t^i = \sigma(W_p s_t^n + b_p)$
5:         $p = p + p_t^i$
6:     end while
7: $s_t = \sum_{i=1}^{N(t)} p_t^i s_t^i$
8: $y_t = \sum_{i=1}^{N(t)} p_t^i y_t^i$
```

Note that this algorithmic implementation is for a *single* ACT timestep.

3.1.2 Are mean-field approximations in latent space a legitimate assumption?

A major assumption made in (Graves, 2016) is that using mean field approximations to the expectation over the halting probabilities is an accurate approximation to sampling from the distribution generated by $p_t^{1 \ldots N(t)}$. He argues that this is satisfactory as vector representations of words have ”been observed to behave in linear ways” (Mikolov et al., 2013b). This view is reinforced by the fact that this distributivity is not solely a feature of the unsupervised skip-gram model pioneered by Mikalov (Mikolov et al., 2013a), but actually applies to a much wider range of word vectors produced by a number of different approaches (Levy et al., 2015). additionally commenting that techniques for neural network optimisation, such as dropout, often enforce considerably more aggressive and potentially disruptive constraints than the assumption of linearity.

3.1.3 Summary

Overall, we have demonstrated that adaptive computation methods can provide tangible benefits to a variety of neural network models, including hyperparameter optimisation, unsupervised and recurrent models. Adaptive computation is a promising avenue for research, as being able to adapt to your inputs and surroundings is a trait not normally associated with static, pre-defined machine learning models.

In the next section, we describe a problem that we believe can benefit from adaptive computation.

Chapter 4

Recognising Textual Entailment

Recognising Textual entailment is the problem of determining whether a hypothesis can be inferred from a premise. RTE is therefore related to formal logic, with the distinction being that natural language as a formal vocabulary space is certainly not well defined. Natural language inference relies on semantic knowledge, linguistic features and fundamentally, language understanding, rather than formal reasoning structures, such as theorem proving and first order logic.

Additionally, the departure from logical reasoning is compounded by the fact that a positive entailment does not necessarily imply strict logical consequence. For instance, consider:

- Premise: A man is holding a ladder steady whilst his friend is painting a roof.
- Hypothesis: A person is standing on a ladder painting a roof.

Although it is not a strict logical consequence that a person would need to be on a ladder to paint a roof, in the context of a spoken human conversation, it would be assumed that the person referred to would be on the ladder. Humans use surrounding context to inform logical conclusions - here, we immediately infer that a man would not need to hold a ladder steady if there was not someone standing on the ladder and hence resolve the inference positively. It is this kind of informality and assumptive reasoning that distinguishes the task from formal logical inference.

4.1 Why is RTE an interesting problem?

Recognising textual entailment forms a fundamental part of many downstream NLP tasks, such as summarisation, sentiment analysis and particularly question answering. Additionally, the breadth and depth of language features required to accurately complete RTE tasks is large. Without representations conferring at least some of the following properties, any attempt at RTE is unlikely to be particularly successful:

- Object and event co-reference
- Background knowledge (”parks” are ”outside spaces”)
- Quantification of adjectives(”very nice” $>$ ”quite nice”)
- Lexical/scope ambiguity (Who’s coat is being taken, when referring to a group)
- Single word/phrase hypernymy/hyponymy
- In subtle cases, stress on sentence structure through phrasal verbs etc.

These facilities are also intimately tied up in the idea of feature representations and why they are difficult to construct; a good model for sentence representation would necessarily need to include many of the above features in order to accurately represent the meaning of a sentence.

4.2 History of RTE Approaches

A very basic initial approach to RTE was creating representations of premise and hypothesis sentences using bag of words approaches. A probabilistic model, originally introduced in (Glickman, Dagan, 2005) ignores all syntactic structure, word order and dependency information and simply assumes that $P(h|p)$, the probability that the premise supports the hypothesis, decomposes in the following way:

$$\begin{aligned} P(h|p) &= \prod_j P(h_j|P) \\ &= \prod_j \max_i P(h_j|P_i) \end{aligned} \tag{4.1}$$

where the authors assume that the contribution to the final entailment classification for a given word in the hypothesis is mainly generated from a single word in the premise. This allows the focus of the problem to move from a sentence based entailment metric to a lexical per-word scoring function. Several different similarity metrics have been attempted here, such as vector space similarity models and distributional similarity metrics. This method of matching words with their "support" from the premise draws many similarities with Statistical Machine Translation alignment algorithms and is very similar to the baseline models presented by Och and Ney in (Och, Ney, 2003).

Various augmentations to the basic model, such as weighting alignments by some global count-based metric like TF-IDF, or using the similarity metric to generate multiple overlapping features (Jijkoun, Rijke, 2005), all improve the flexibility of this model to account for distributional statistics and other basics, such as the presence of negations. However, models of these type will never be able to take into account word order, non-local word features or background knowledge.

Developments along this vein include MANLI (MacCartney et al., 2008), which uses a phrase-based alignment and a averaged perceptron scoring function over lexical features. However, the entailment problem requires considerably more than prediction derived from a good alignment of the premise and hypothesis; linguistic constructs, such as antonymy, modality, implicative verbs and negations play a large role in determining entailment as well. Proper resolution of these linguistic features requires syntactic structure to be incorporated into the model.

In order to absolve this lack of structure, many RTE models moved to a form of graph-based alignment over syntactic dependency trees (Haghighi et al., 2005;

Salvo Braz de et al., 2005). Possible alignments are scored using some feature based scoring function and the highest scoring alignment is used as a proxy for entailment. However, as finding the largest maximal matching of two graphs is NP-complete, these approaches require simplification in order to find a solution in polynomial time. Many consider only individual node scores initially and incorporate edge scores between nodes incrementally using local search to find solutions.

The other popular structured approach to natural language inference is transforming premise and hypothesis pairs to first order logic formulas in order to use formal theorem proving approaches (Akhmatova, Aliod, 2005). Particularly, this approach has demonstrated good results when combined with external knowledge bases and deeper formal logic theory, such as the theory of models. (Bos, Markert, 2005) demonstrated state of the art performance(in 2005) on (Dagan et al., 2005) the using an approach which generates models(logical systems of axioms) where the inference is false, which requires a smaller search space.

However, logical and graph based systems suffer fundamentally from the fact that if we were able to represent natural language statements in terms of first order logic, they would be perfectly solvable given enough external knowledge. That these approaches are not actually very successful points to the fact that representing natural language via formal structures is difficult and below we present 3 possible reasons, discussed in (MacCartney et al., 2006):

- Graph matching does not provide a 1 - 1 mapping. Modal operators which effect entailment, such as ”could, should” and even simple modifying adjectives, do not effect a parsing structure but *do* alter possible entailments which could be drawn from a premise containing such words.

- Using local search to match parse trees does not account for the fact that many important elements of a sentence for determining entailment, such as negations and only take effect when viewed from a global perspective. Consider the case ”The dog was running fast” implies ”A dog was running”. However, the case: ”No dogs were running fast” does not entail ”No dogs were running”.

- Graph matching may suit the positive entailment problem, but minimising the cost of matching two parses makes determining the negative alignment case difficult.

These points are separate to the over-arching fact that without human level background knowledge and sometimes even with it, a sentence can be ambiguously parsed. For example, humans require background knowledge that ”vultures are birds” and ”circling is something that birds do in the sky” to tell that in the sentence ”He noticed a vulture was circling whilst crossing the road”, a man has not observed a vulture crossing the road whilst circling, but has in fact noticed a vulture in the sky whilst *he* was crossing the road. Sentences with dangling modifiers are frequently used in the English language and often not noticed, as given the context, we can resolve parses which to a machine with no background knowledge, would seem be extremely difficult to deduce.

Considering the above, we decided to evaluate neural models with an adaptive reasoning component on Natural Language Inference because:

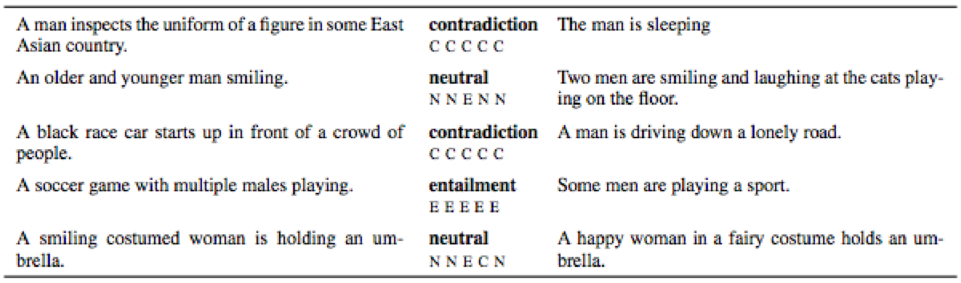
- Neural networks have demonstrated state of the art performance on many NLP tasks, including RTE.
- Neural models have the ability to learn ”soft” inference rules which may go some way to negating the problems discussed with graph matching methods.
- Due to the complex, multi-premise inference sometimes required to determine entailment, we hoped that the adaptive number of reasoning steps our model took may provide benefit over and above another task which cannot be broken down into smaller steps.

4.3 Stanford Natural Language Inference Corpus

The Stanford NLI Corpus (Bowman et al., 2015) is the first RTE dataset of considerable size and quality, two orders of magnitude larger than any other RTE corpus previously released. It was created using Amazon Turk and image labels from the

Flickr 30k dataset (Young et al., 2014) by asking Turkers to provide hypotheses based on an image, where the image caption was used as the premise.

This approach generated 570k (hypothesis, premise) pairs of reasonably high quality. The authors note that spelling errors are infrequent and there is a high prevalence of pairs which require some background knowledge.



A man inspects the uniform of a figure in some East Asian country.	**contradiction** C C C C C	The man is sleeping
An older and younger man smiling.	**neutral** N N E N N	Two men are smiling and laughing at the cats playing on the floor.
A black race car starts up in front of a crowd of people.	**contradiction** C C C C C	A man is driving down a lonely road.
A soccer game with multiple males playing.	**entailment** E E E E E	Some men are playing a sport.
A smiling costumed woman is holding an umbrella.	**neutral** N N E C N	A happy woman in a fairy costume holds an umbrella.

Figure 4.1: Examples from the Stanford Natural Language Inference Corpus (Bowman et al., 2015).

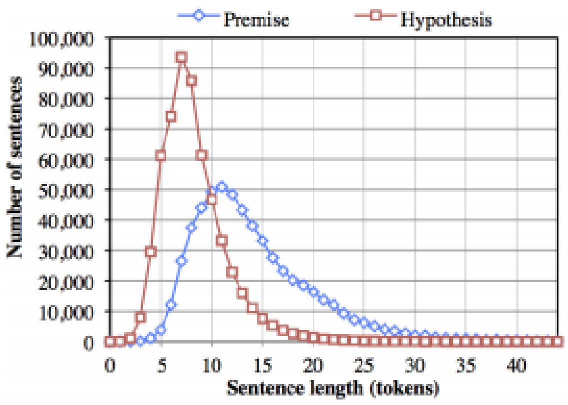


Figure 4.2: Graph showing the distribution of hypothesis and premise sentence lengths from the Stanford Natural Language Inference Corpus. The difference in distributions arises from the method of collection, although a majority of sentences are syntactically complete. (Bowman et al., 2015)

Previous corpora for specifically for Natural Language Inference include Sentences Involving Compositional Knowledge(SICK) (Marelli et al., 2014) and the Framework for Compositional Semantics (Cooper et al., 1996). Both of these resources are considerably smaller in size, 10k and 300 examples respectively. They are hand curated, although SICK includes heuristically generated examples.

A few other relevant corpora exist which are of a suitable size, such as the Paraphrase Database (Ganitkevitch et al., 2013) Entailment Graphs (Levy et al., 2014). However, in the first case, the data is overly primitive, consisting of simple adjective removal and other very basic syntactic transformations. Additionally, the example lengths are often short, meaning it is unsuitable for use with neural models designed to encode information over a longer time period.

In the second case, the data is more suited to the intersection of Information Retrieval, Knowledge Base Completion and Textual Entailment. The premise of Entailment Graphs is to generate inferences from structured data in pre-existing knowledge bases, such as (pain, symptom of, arthritis) implies (arthritis, causes, pain). This dataset is automatically generated and partially automatically labelled. Additionally, there are many restrictions on the types of Entailment representable in the KB tuple format. For instance, resolving hypernymy/hyponymy often requires context(”cures” implies ”killed off” only in a medical/bacterial context), which is not possible given that the extracted tuples have no context whatsoever.

For the above reasons, we decided to utilise the SNLI corpus as the main basis of evaluation for our models.

4.4 Co-reference Resolution

A key observation when working on Textual Entailment data is the decision made regarding co-reference. Without assuming events and objects in (premise, hypothesis) pairs refer to the same thing, it becomes very difficult to obtain *any* RTE pairs which fulfil the negative label.

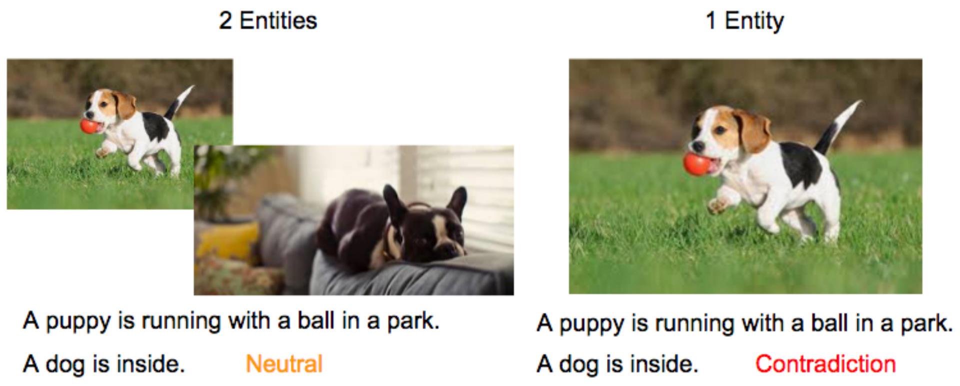

2 Entities
1 Entity
A puppy is running with a ball in a park.
A dog is inside.
Neutral
A puppy is running with a ball in a park.
A dog is inside.
Contradiction

Figure 4.3: Visual demonstration of the co-reference problem in RTE. If we do not consider objects and entities to persist over the premise and hypothesis, almost all examples with the ”contradiction” label will become neutral. Only globally contradictory statements would persist, which is directly counter-intuitive to the way humans infer entailment - via context and co-reference assumptions.

In fact, without this constraint, the problem becomes not at all interesting or relevant to computational approaches to determine entailment. All natural language processing tasks which require textual inference, such as Question Answering, Sentiment Analysis or Summarisation necessarily require this approach and therefore it is natural to take this assumption as given. For instance, Chen et al. (Chen et al., 2016) recently demonstrated an approximate upper bound on the accuracy obtainable on the CNN/Daily Mail datasets (Hermann et al., 2015), due to irresolvable co-references, demonstrating its importance when considering entity matching/question answering tasks. Helpfully, this is also the approach taken in the Stanford Natural Language Inference Corpus, described in the previous section.

Chapter 5

Methods

In this section we describe the models implemented in Tensorflow (Abadi et al., 2015) and evaluated on the Stanford Natural Language Inference corpus (Bowman et al., 2015). We implemented 3 models, two of which use an adaptive reasoning component.

5.1 Adaptive Iterative Attention

The first model we discuss is an extension of the Iterative Alternating Attention model employed in (Sordoni et al., 2016a) for Machine Reading. The original motivation behind this model was to bypass the bottleneck generating a single document representation over large documents in machine reading, as this can be prohibitively restrictive when the document grows in size. Instead, the model incorporates an "inference" step, in which the query and the document are iteratively attended over in order to generate a representation for classification. Our contribution is to generalise this inference step to run for an input conditional number of steps using adaptive computation.

Below we describe the altered model in more detail - the alterations include a modification to the output layer in order to take in to account the cross-entropy loss function(rather than the pointer loss used in MR), a minor difference in the computation of the attention mechanism and the new addition of the adaptive nature of the inference GRU.

Hypothesis and Premise Encoding: Given a premise $p_1....p_n$ and hypothesis

$h_1 ... h_m$, each word is represented by a vector $x \in \mathbb{R}^d$ stored in a word embedding matrix $X \in \mathbb{R}^{|V| \times d}$. We then process the sequence of word vectors using two distinct GRU encoders (Chung et al., 2014), generating two sequences of outputs, $\tilde{p}_1 .... \tilde{p}_n$ and $\tilde{d}_1 .... \tilde{d}_n$ respectively. Note that in the original formulation, bi-directional GRUs are utilised, however, due to computational restrictions we do not.

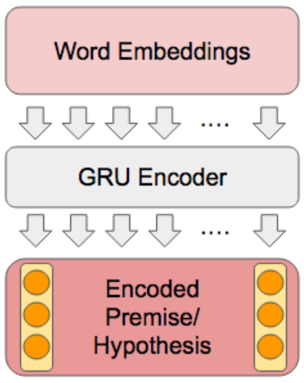


Figure 5.1: Demonstration of using a GRU to encode representations of the premise and hypothesis. In this thesis we use seperate encoders for the hypothesis and premise.

Alternating Iterative Attention: Now we have a temporally dependant representation of both the hypothesis and the premise, we iteratively generate attention representations of both. At inference iteration t, we generate an attention representation of the hypothesis:

$$q_{it} = Softmax_{1...m}(\tilde{h}_i^T (W_h s_{t-1} + b_h)) \qquad q_t = \sum_i q_{it} \tilde{h}_i \tag{5.1}$$

Where q_{it} are the attention weights, $W_h \in \mathbb{R}^{d \times s}$ where s is the dimensionality of the *inference GRU state*. This attention representation of the hypothesis is then used to generate an attention mask over the premise:

$$d_{it} = Softmax_{1...n}(\tilde{p}_i^T (W_p [s_{t-1}, q_t] + b_p)) \qquad d_t = \sum_i d_{it} \tilde{p}_i \tag{5.2}$$

Where d_{it} are the attention weights, $W_p \in \mathbb{R}^{d \times (d+s)}$ where s is the dimensionality of the *inference GRU state* and $[x, y]$ denotes the concatenation of vectors. Note that this attention representation of the premise is ”conditioned” on the attention representation of the hypothesis at the current timestep, as well as the previous state of the inference GRU.

Gating Mechanism: Although the attention representations could now be concatenated into an input for the inference GRU, one last addition is used in (Sordoni et al., 2016a) in order to allow attention representations to be forgotten/not used. This is important as the future attended locations are dependent on the locations attended at this timestep, but if they are not informative, we would prefer to ignore them. For this reason, we include:

$$\mathbf{r_t} = G_p([s_{t-1}, d_t, q_t, d_t \cdot q_t]) \qquad \mathbf{s_t} = G_h([s_{t-1}, d_t, q_t, d_t \cdot q_t]) \tag{5.3}$$

Where G_p, G_h are 2 layer, feedforward networks $f : \mathbb{R}^{s+3d} \to \mathbb{R}^d$. The use of what is effectively a polynomial feature in the $q_t \cdot d_t$ is not entirely explainable, but is in theory meant to make it easier to take hypothesis-premise word alignments into consideration, although it is not clear why this is required or even desirable. The generated attention representations are multiplied element-wise with the result of the gating function and concatenated($[r_t \cdot d_t, s_t \cdot q_t] \in \mathbb{R}^{2d}$), forming the input at time t to the inference GRU.

In (Sordoni et al., 2016a), the number of steps this inference GRU is run for is a hyperparameter of the model. Instead, we learn the number of inference steps to take using a single step of the Adaptive Computation Time algorithm, described above, where the input into the halting layer is the output y_t of the inference GRU.

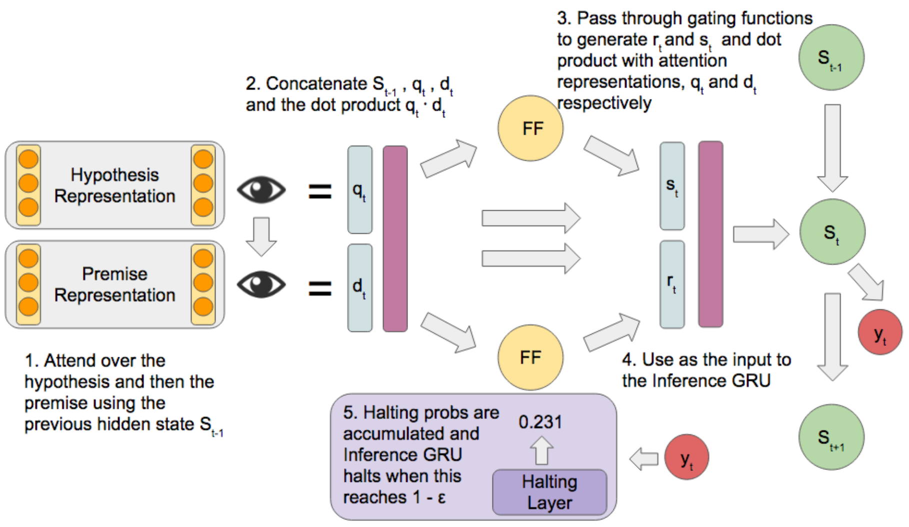

3. Pass through gating functions to generate r_t and s_t and dot product with attention representations, q_t and d_t respectively
2. Concatenate S_t-1 , q_t , d_t and the dot product q_t · d_t
S_t-1
FF
Hypothesis Representation
q_t
s_t
S_t
Premise Representation
d_t
r_t
y_t
1. Attend over the hypothesis and then the premise using the previous hidden state S_t-1
FF
4. Use as the input to the Inference GRU
5. Halting probs are accumulated and Inference GRU halts when this reaches 1 - ε
0.231
Halting Layer
y_t
S_t+1

Figure 5.2: Visualisation of a single step of the *inference GRU*, with step-by-step explanations.

5.2 Comparison model: Decomposable Attention

In order to compare the new model against a state of the art benchmark, we implemented the model described in the paper ”A Decomposable Attention Model for Natural Language Inference” (Parikh et al., 2016). This model is an elegant yet frustrating approach to RTE on the SNLI corpus as it employs a novel attention mechanism which is computationally efficient and parallisable, but which acts on pure word representations, with no recurrent aspect used in the model at all.

In this approach, the attention representation purely generates an alignment as the word vectors are temporally independent - it is simply generated by the dot product of the result of a single feedforward network.

The un-normalised alignment weight given hypothesis and premise representations $h_1 ... h_m$ and $p_1 ... p_n$ are defined as:

$$e_{ij} = F(h_i)^T F(p_j) \tag{5.4}$$

Where $F : \mathbb{R}^d \rightarrow \mathbb{R}^d$ is a feedforward network with RELU activation functions. The vector representations of the hypothesis and premise are then generated by taking the softmax over the respective dimensions of the E matrix:

$$\beta_j = \sum_{i=1}^{m} Softmax_{1...m}(e_{ij}) \cdot h_i \qquad \alpha_i = \sum_{j=1}^{n} Softmax_{1...n}(e_{ij}) \cdot p_j \tag{5.5}$$

Ie here β_j is the phrase in the hypothesis which is softly aligned to word p_j in the premise and vice versa for α_i. Note that this attention representation is of a finer grain than previously used, as we are generating an alignment for each word in the hypothesis and premise.

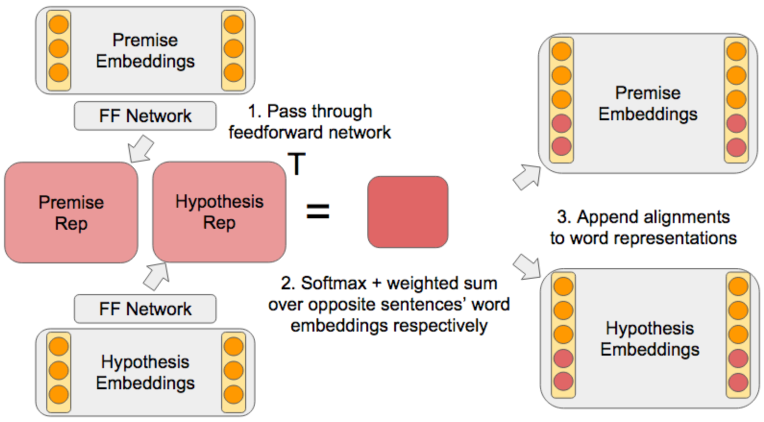

Premise Embeddings
FF Network
1. Pass through feedforward network
Premise Rep
Hypothesis Rep
T
=
FF Network
Hypothesis Embeddings
2. Softmax + weighted sum over opposite sentences' word embeddings respectively
Premise Embeddings
3. Append alignments to word representations
Hypothesis Embeddings

Figure 5.3: Visualisation of the alignment process used in Decomposable attention. Note that a matrix multiplication of both the encoded hypothesis and premise computes the element-wise product of all vectors efficiently.

Now we have these per-word alignment representations, we use feedforward networks to generate single representations for the premise and hypothesis individually:

$$\mathbf{p} = \sum_{j=1}^{n} G([\beta_j, p_j]) \qquad \mathbf{h} = \sum_{i=1}^{m} G([\alpha_i, h_i]) \tag{5.6}$$

Where $G : \mathbb{R}^{2d} \rightarrow \mathbb{R}^d$ is a feedforward network taking as input a word in either the hypothesis/premise and it's respective alignment. These two representations $[\mathbf{h}, \mathbf{p}]$ are then concatenated and fed through a final softmax layer in order to generate a probability distribution over the classes {entailment, neutral, contradiction}.

This model achieves 86.3% test accuracy on the SNLI corpus and until recently, was the state of the art. Additionally, it uses an order of magnitude fewer parameters than most other approaches.

5.3 Alterations and other models

We quickly realised that the critical computational bottleneck of the above model is the use of GRU/LSTM encoders for the premise and hypothesis. In order to attempt to alleviate this problem, we substituted the GRU encoders for the premise and hypothesis for the feedforward attention mechanism described above from (Parikh et al., 2016), which acts on unadulterated pre-trained word vector representations, removing the need to generate GRU/LSTM representations of the premise and hypothesis.

Adaptive Decomposable Attention: In this approach, the attention representation purely generates an alignment as the word vectors are temporally independent - it is simply generated by the dot product of the result of a single feedforward network. We then used this representation of the words in the hypothesis and premise as the input to the *inference GRU* module using adaptive computation to determine the number of steps in exactly the same way as previously. As was demonstrated in (Parikh et al., 2016), for tasks involving reasoning over pairs of sentences, cross sentence alignment rather than temporal consistency is more important. Therefore, we hypothesised that allowing the Inference GRU module to conduct inference steps over these pre-aligned representations could provide benefits.

Chapter 6

Experiments

In this section, we describe results and insights from running the models described in the Methods section on the Stanford Natural Language Inference Corpus (Bowman et al., 2015).

Experiments were run in two stages due to computational limitations in terms of the availability of GPU resources. The first step involved running a grid search on the UCL CPU cluster over selected hyperparameters for 10 epochs per model. We explored the following hyperparameter ranges:

- Learning Rate: $\{0.001, 0.0001\}$
- Dropout: $\{ 0.4, 0.2 \}$
- Word Representation Size: $\{126,200,256\}$
- Step Penalty $\{0.01,0.005,0.001,0.0005,0.0001,0.00005,0.00001\}$

After running these models in parallel on different CPUs, the hyperparameter setting with the best accuracy on the validation data is selected an trained for another 10 epochs. Only these models are evaluated on the test set. All models were trained using the ADAM optimiser (Kingma, Ba, 2014) with 0.9 first momentum coefficient of 0.9 and second momentum coefficient of 0.999, the default configuration suggested by the authors. Word embeddings are initialised using GloVe pre-trained vector representations (Pennington et al., 2014) and words without a pre-trained representation are initialised to vectors drawn from a standard Normal distribution

$\mathbb{N}(0, 0.05)$. Word embeddings are updated during training. Following the approach taken in the original SNLI paper (Bowman et al., 2015), we discard any examples with no gold label, leaving us with 549,367, 9,842 and 9,824 for training, validation and testing respectively.

Additional hyperparameters were set and not optimised as follows:

- Encoder GRU Size: 128
- Inference GRU Size: 256
- Max Gradient Clipping: 5
- Embedding Regularization: 0.0001
- Batch Size: 32
- Max Adaptive Steps: 20 (Adaptive Models only. This limit was never reached during training and therefore did not effect the learning process.)
- Epsilon for halting probability accumulation: 0.01

As the Decomposable Attention model (Parikh et al., 2016)(DA) was originally evaluated on the SNLI corpus, we used the hyperparameters described in the paper and did not do any hyperparameter search.

One additional point to bear in mind when comparing the models below is the difference between our re-implementation of the Decomposable Attention paper and their original training scheme. After failing to reproduce their results with the same hyperparameters, I contacted them to obtain more details regarding their distributed training schedule - they use 10 asynchronous threads during optimisation, with the total number of passes over the full set of training data equalling 3200.

Given that our training schedule was effectively limited to around 20 epochs due to time and resource constraints(admittedly, only using a single threaded implementation, but multi-threading optimisation was not within the scope of this thesis), we are comparing these models given the resources and time available. Therefore, the test accuracy obtained by our implementation is considerably lower, although it is clear from the training and validation plots 6.1b that the DA model has not completely converged. This is taken into consideration when comparing models.

Below, we present the training and validation accuracy plots for 3 models:

- Adaptive Iterative Alternating Attention - the best model from a grid search trained over 20 epochs.
- Adaptive Decomposable Attention model - the best model from a grid search trained over 20 epochs
- Decomposable Attention (Parikh et al., 2016) with hyperparameters stated in the paper

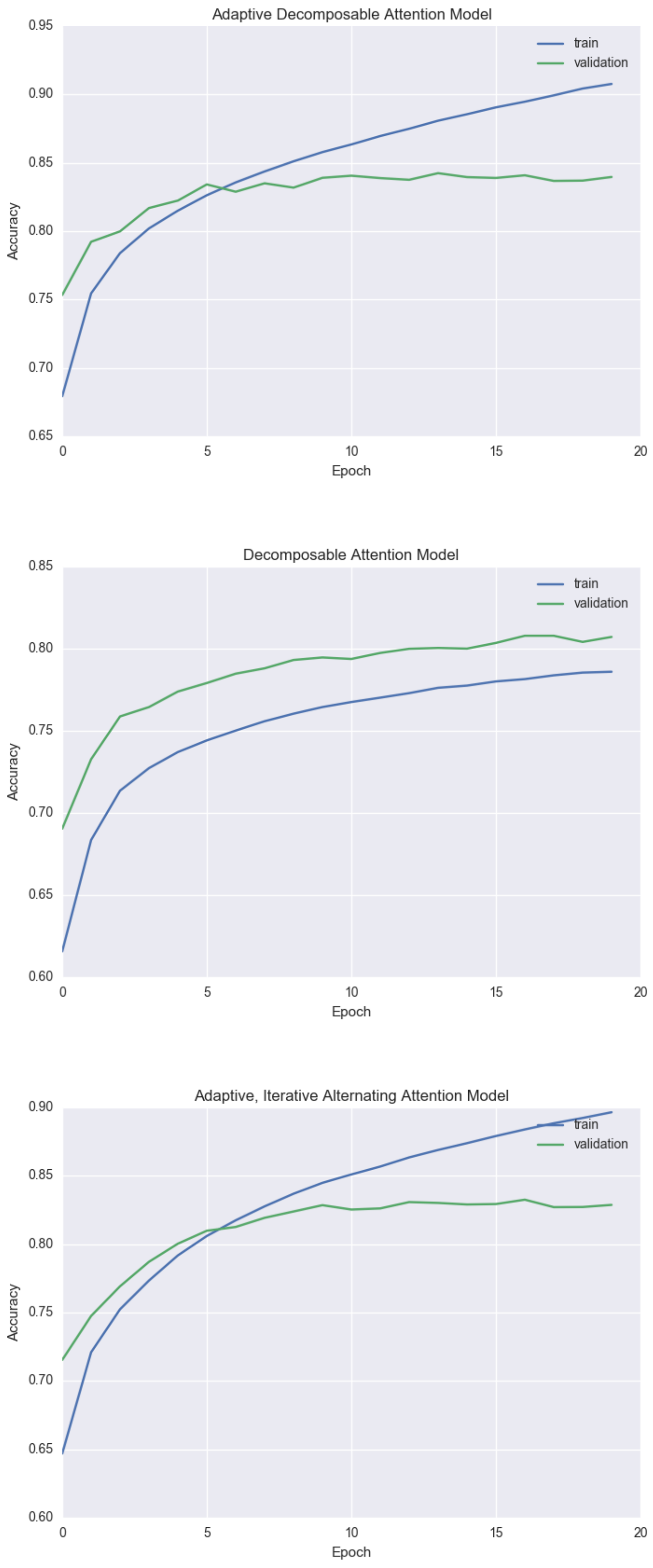


Figure 6.1: Training and Validation accuracy plots for the Decomposable Attention, Adaptive Decomposable Attention and Adaptive IAA models. Parameters for the DA model are taken from the original paper where training was over 3200 epochs and this is reflected in the validation plot for this model.

6.1 TensorFlow

All models were implemented using Tensorflow (Abadi et al., 2015). Tensorflow is an open source library for numerical computation, specifically supporting graph-based data-flow graphs and efficient backpropagation. Tensorflow is particularly useful because it automatically maps a computational graph onto available devices, enabling very easy GPU usage.

Additionally, as Tensorflow compiles computational flow graphs and runs them using heavily optimised kernels for linear algebra computation(BLAS) and has an underlying core built in c++, it is reasonably fast to train. Using batch sizes of 32, a single epoch of our fairly complicated architecture takes around 4 hours to complete a single iteration over the entire number of training examples(500,000) on a CPU, or about 20 minutes on a GPU.

Relative to other gradient-based computation libraries, such as Theano (Theano Development Team, 2016), Tensorflow provides a very well balanced interface to deep neural networks, allowing both granularity (we implemented our own attention mechanisms and Adaptive Computation modules) and abstraction, such as providing pre-defined optimiser classes for frequently used optimisation techniques.

6.2 Adaptive Iterative Alternating Attention Model

Clearly above, the Adaptive reasoning component does not detract from the overall validation accuracy and is competitive with state of the art approaches given training restrictions. However, we are more interested in whether multi-hop inference has a positive impact on performance. Below, we showing the accuracy for the same IAA model with different numbers of fixed inference steps(the number in the legend) compared to the performance of the IAA model with the adaptive component. From the below graph, it is clear that there is a difference between models using a single inference hop and ones employing more than this. However, performance clearly does not scale linearly with respect to the number of inference hops taken by the model given the similar results for 1-8 hops. This provides an even stronger argument that non-adaptive tuning of this parameter is a bad idea - we do not want to be forcing a model to use more inference steps than necessary if it has no impact on performance.

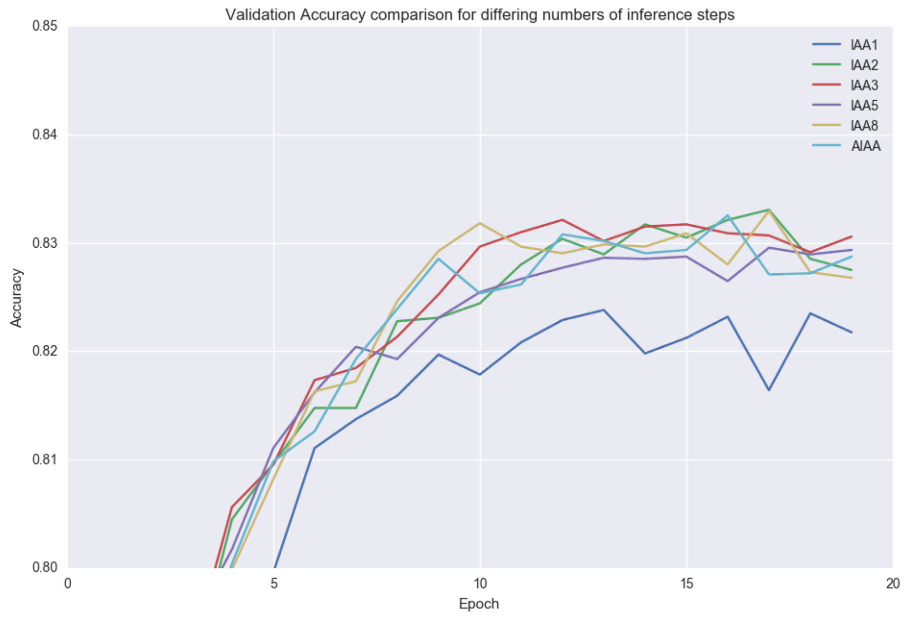
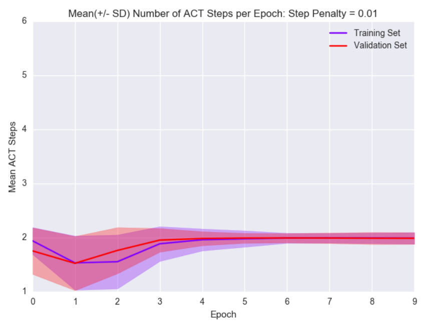
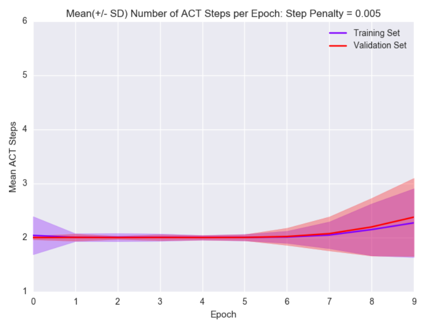


Figure 6.2: Demonstration of the accuracy gain from using multiple inference steps. Clearly, multiple steps leads to a higher accuracy, but the exact number of steps is hard to determine.

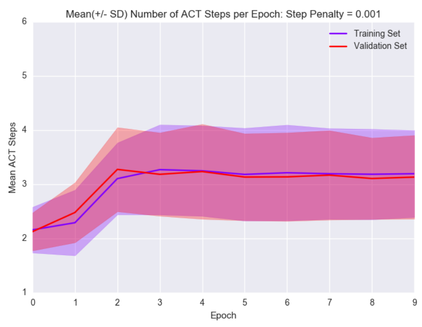
Given the above graph demonstrates that multiple inference hops may be a good idea, we now demonstrate the restriction of using Adaptive Computation Time to learn this - the model is extremely sensitive to the step penalty hyper parameter. Below we present the mean(+/- 1 SD) number of inference steps taken during training when evaluated on the training and validation sets. Clearly, there is a stark difference between the 0.01 and 0.005 values and the rest. It is particularly interesting to note that the mean number of steps taken by the model is in some sense naturally bounded, as no real increase in the mean is observed past a step penalty equal to 0.0005.



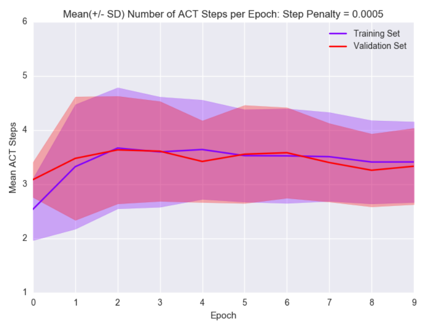
(a) 0.01



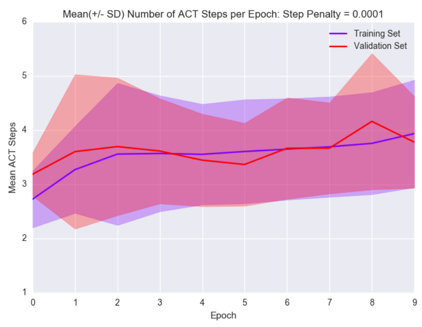
(b) 0.005



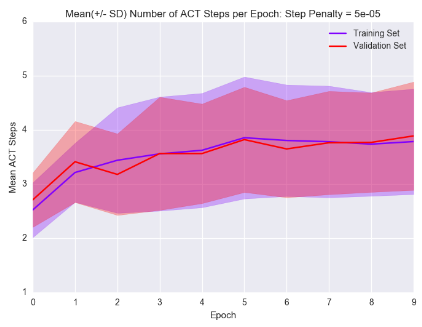
(c) 0.001



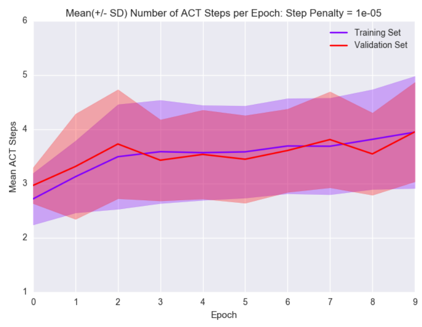
(d) 0.0005



(c) 0.0001



(d) 0.00005



(e) 0.00001

Figure 6.3: Graphs showing the mean and standard deviation of the ACT Steps during training on the training and validation steps, with decreasing Step Penalties reading left to right, top to bottom. Clearly, the model is very sensitive to this parameter up to a point, but settles after this point with the main difference being increases in variance.

6.3 Adaptive Decomposable Attention Model

Demonstrating the difficulty of determining the step parameter, below we give the mean(+/- SD) number of steps taken by the Adaptive Decomposable Attention model:

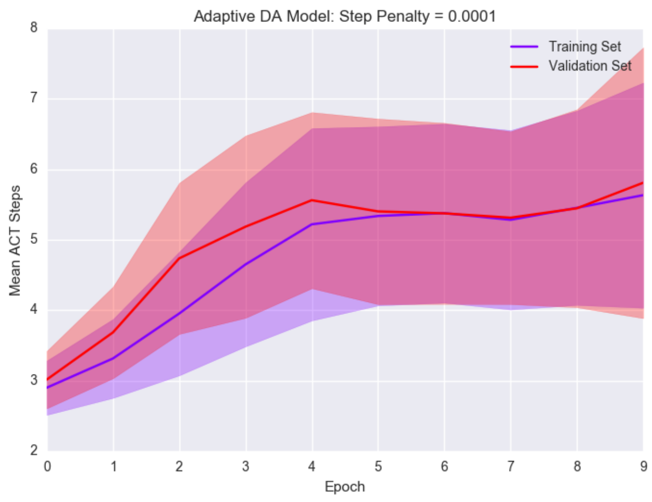


Figure 6.4: Clearly, the step penalty applied to the ADA model has a considerably different impact on performance - with this model, the number of adaptive steps taken is considerably higher but not at a cost of performance.

Below, we present the pairwise joint distribution over the number of inference steps taken on the test set and the averaged length of the Hypothesis and Premise. This distribution goes slightly against our expectations, as we had presumed that longer premise/hypothesis statements would lead to more inference steps. However, this is clearly not the case and this assumption that complexity is related to sentence length is possibly quite naive, especially considering how the dataset was generated. For instance, a Hypothesis generated by removing adjectives from a long Premise is not particularly difficult to deduce and equally, it is certainly possible to generate a short Premise,Hypothesis pair which requires deep background knowledge.

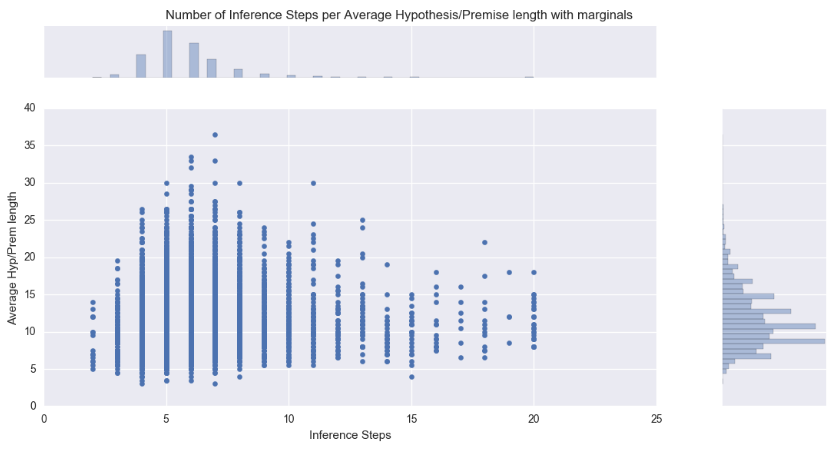


Figure 6.5: Joint and marginal distributions of the number of inference steps taken with respect to the averaged Premise and Hypothesis lengths.

6.4 Qualitative analysis

In this section, we provide several visualisations of the attention weights generated by both the AIAA and ADA models when evaluated on the test set of the SNLI corpus. These images show the attention weights over the hypothesis and premise, as well as the halting probabilities generated by the ACT loop. First, the model generates attention weights over the hypothesis, seen in the first block of attention weights at the top of the image. Secondly, the model generates attention weights over the premise, seen in the bottom block. These then form the state of the *inference GRU*, as previously described. The index on attention weights correspond to the ACT step in which the weights were generated and they are not independent, as the *inference GRU* state is passed through each timestep. On the right hand side, the the visualisation of the halting probabilities show how much weight is assigned to each attention representation in the overall final state. This is a probability distribution and sums to 1.

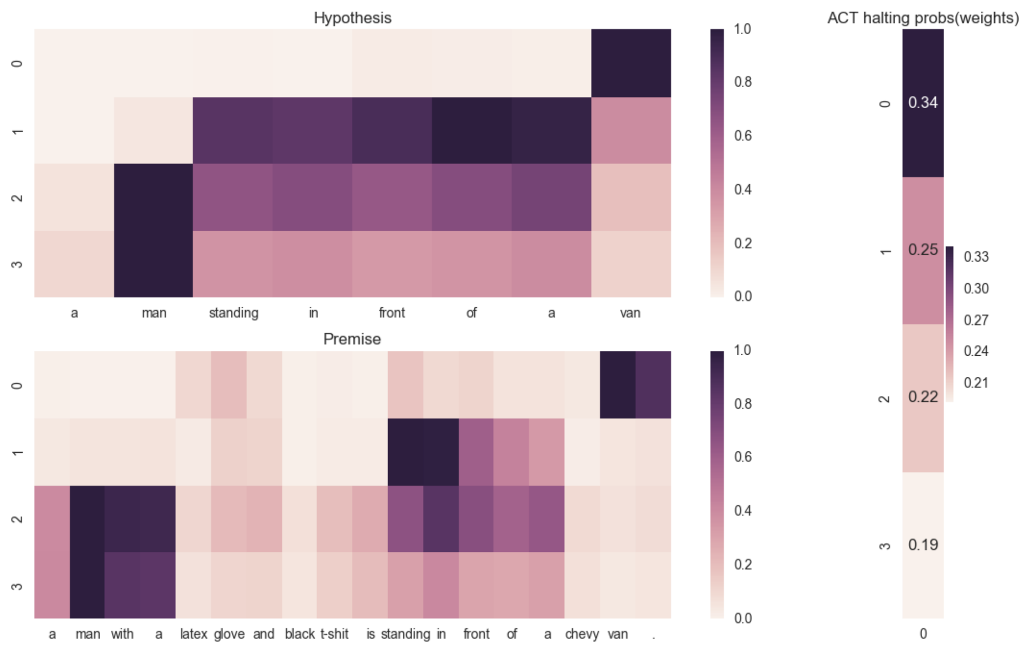


Figure 6.6: Visualisation of the attention weights produced at each inference step which are used to form the states fed into the inference GRU. At each time-step, the Hypothesis is attended over using the inference GRU state as the query. This representation is then appended to the query and used to generate an attention mask over the Premise.

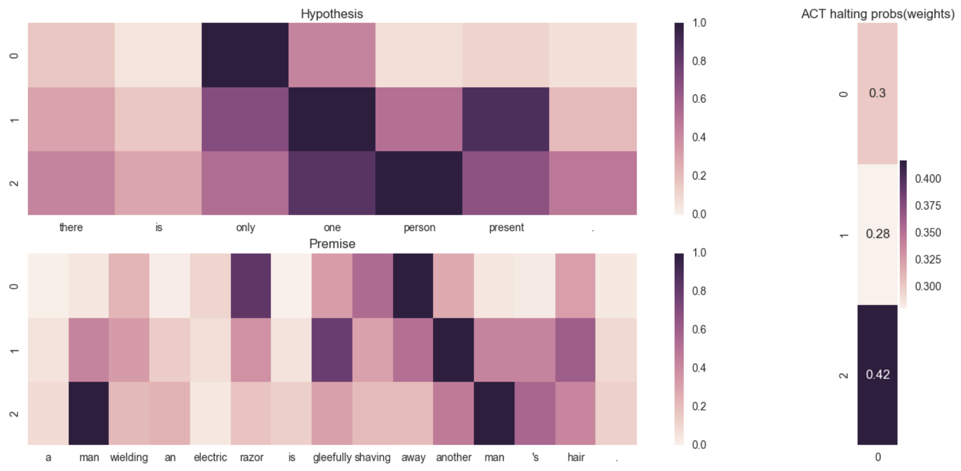


Figure 6.7: Attention visualisation for the Adaptive DA model. Note that although similar to the previous visualisation, the vector representations of words include both the word representation and a representation of the individual word's alignment in the opposite sentence.

Above, we have provided two attention visualisations demonstrating the way the model uses it's access to the premise and hypothesis representations. In both models, we observed multi-step reasoning, in the sense that the attention weights change over the course of the ACT steps and attend over different parts of the sentence. The ADA model utilised considerably more reasoning steps - although the optimal step penalty parameter settings for the two models differed (ADA: 0.0001, AIAA: 0.01), even relative to the AIAA model with the same step penalty, it used considerably more across the board. This further reinforces the argument that this parameter is difficult to set correctly.

The sequential resolution in the second two steps of "one ... present" and "another" and then the alignment matching "one person" to both the references to the men in the premise demonstrates a scenario where a single attention mask may have missed the required information required.

In addition to the analysis provided above, there were several qualitative points to be raised after analysing the attention visualisations which can be found in the appendix A. We found the following points to be particularly of note:

- The initial attention on the hypothesis seems to always focus on the end of the sentence. We hypothesise that this is due to the nature of the memory retention in a GRU cell and that this end representation gives a good initial "summary" of the sentence. This would make sense of how the attention is computed, as the initial "query" vector on the first step is all zeros, so the attention is computed purely on the basis of the sentence encoding.

- The model often seems to fixate on certain words, but still generate more steps via ACT. It is possible that this arises from the way in which the halting probability is generated, using a single linear layer with a sigmoid activation function. An interesting extension would be to use a deeper network to generate this, as it is quite important in the context of the model.

- The attention clearly shows multiple foci in reasoning over examples. The model very rarely takes a single step, even in the cases of high step regularisation.

- There are several instances of a large ACT weight on the final loop in the inference GRU(such as in 6.7), demonstrating that when the model has found the key attention representation which provides the necessary insight, it can learn to halt immediately.

After seeing the effects of the number of inference steps taken by the IAA model, we decided to do a similar analysis for the ADA model. As the inference GRU output for every step that it takes, we decided to extract the output at every step and evaluate it in the same way we would normally have done with the final representation, using the softmax function to produce a distribution over three classes. In this way, we can see how the model's opinion develops over time and often, these are non-trivial.

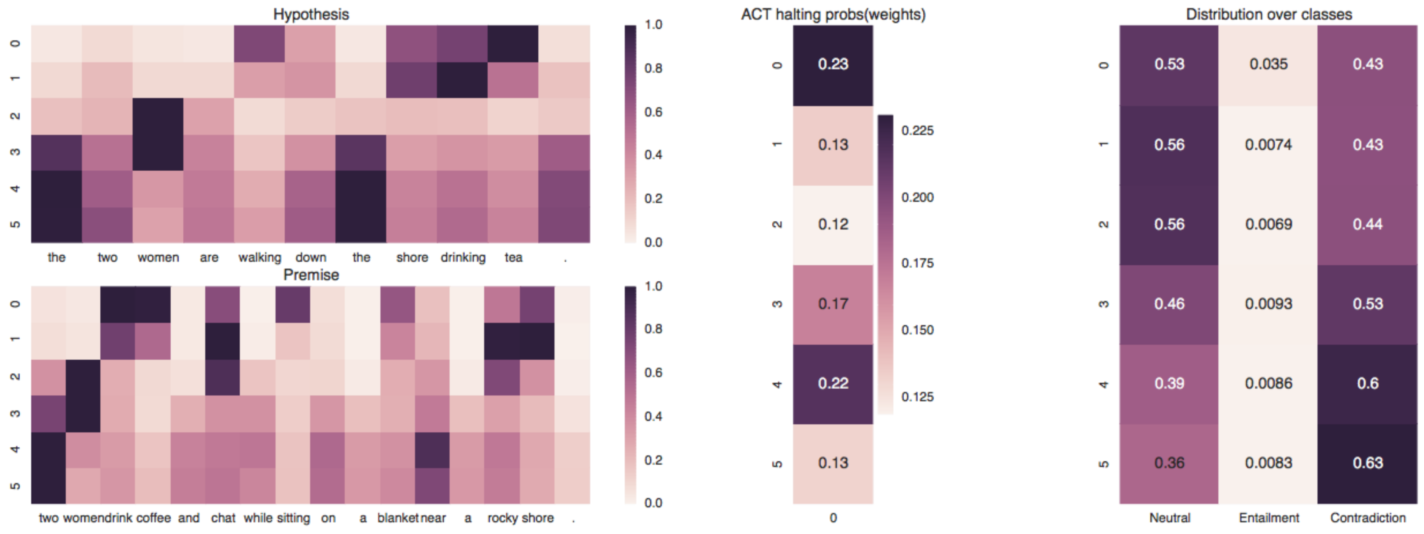


Figure 6.8: Demonstration of the model's predictions after collecting varying amounts of information from the premise and hypothesis via the attention mechanism. Note that in this case, at least 3 inference GRU steps are required to solve the inference correctly.

6.5 Relative Performance

Below, we compare results from the models presented in this thesis to other state of the art models in the field.

Model	Test Accuracy	Parameters
Logistic Regression w Lexical features (Bowman et al., 2015)	78.2%	n/a
Baseline LSTM (Bowman et al., 2015)	77.6%	220k
1024D "Skip Thought" GRU (Vendrov et al., 2015)	81.4%	15m
SPINN-PI Recursive NN (Bowman et al., 2016)	83.2%	3.7m
100D LSTM w word-by-word attention (Rocktäschel et al., 2015)	83.5%	250k
200D Decomposable Attention (Parikh et al., 2016)	86.3%	380k
300D Full tree matching NTI-SLSTM-LSTM (Munkhdalai, Yu, 2016)	87.3%	3.3m
200D Decomposable Attention (our implementation) (Parikh et al., 2016)	80.2%	380k
Adaptive IAA (ours)	82.2%	1.25m
Adaptive DA (ours)	83.8%	990k

Table 6.1: Table showing relative performance on the Test set of the SNLI corpus, in addition to the number of non-word embedding parameters used in the respective models.

Initially, we compared our models to baseline models proposed in the original SNLI paper. These include a logistic regression model acting on features extracted from the hypothesis and premise, including sentence length, word overlap, bigrams, trigrams, POS tags and cross uni/bigrams. Our models outperform this benchmark by 4% and 5.6% respectively on the test set. This is not a surprising conclusion, given the complexity of the task and the availability of training data will tend to favour a neural network model. Our models also outperform the baseline LSTM model by a considerable distance. However, this model is fairly naive and simply uses the final state of two LSTMs run over the premise and hypothesis as inputs to feedforward networks, using a softmax function to generate 3-class probabilities. This model contains very little pairwise comparison of the two sentences, which is demonstrably key in this task, given the performance benefit demonstrated in the models above which use attention (Rocktäschel et al., 2015; Munkhdalai, Yu, 2016).

Additionally, the Adaptive DA model outperforms the word-by-word approach taken by (Rocktäschel et al., 2015) and the method proposed by (Bowman et al., 2016) using Recursive Neural Networks and an external memory component. This perhaps demonstrates that the pre-alignment, followed by attention over these pre-aligned representations is a good way of pre-conditioning sentences in tasks involving the comparison of sentences. We also note that Bowman's method acts purely on sentence representations constructed without examining the other sentence and therefore addresses a related, but slightly tangential problem of informative sentence representations. The Adaptive DA model outperforms our re-implementation of the original Decomposable Attention paper by 3.6 percentage points, a substantial improvement which would outperform current state of the art models if we could achieve the same relative improvement on their reported statistics.

Chapter 7

Evaluation and Further Research

In this thesis we have introduced Natural Language Inference models which can adapt their computation to examples. Taking inspiration from Machine Reading, the Adaptive Computation Time algorithm and light attention representations, we have built a model from principled foundations and evaluated its performance on a Recognising Textual Entailment task. We have demonstrated that alignment via temporally independent attention can produce powerful representations and that these representations can be used to reason with in the same way as those generated by a Recurrent Neural Network.

Additionally, we believe this is the first application of ACT which achieves a result competitive with state of the art approaches on a large scale NLP task, whilst simultaneously demonstrating how ACT can provide novel insight into multi-step reasoning in the context of a NLI system.

Although in this thesis we have demonstrated the effectiveness of multi-step reasoning, we believe that this approach could be applied to recent models which make use of external memory, such as Memory Networks (Weston et al., 2014). Using ACT as an addressing system for multiple memory read/write heads is an interesting direction to take this work in, particularly when applied to a problem whose solution is actually *dependent* on taking multiple steps, perhaps in the case of accessing a knowledge base for Question Answering.

Appendix A

Appendix of additional figures

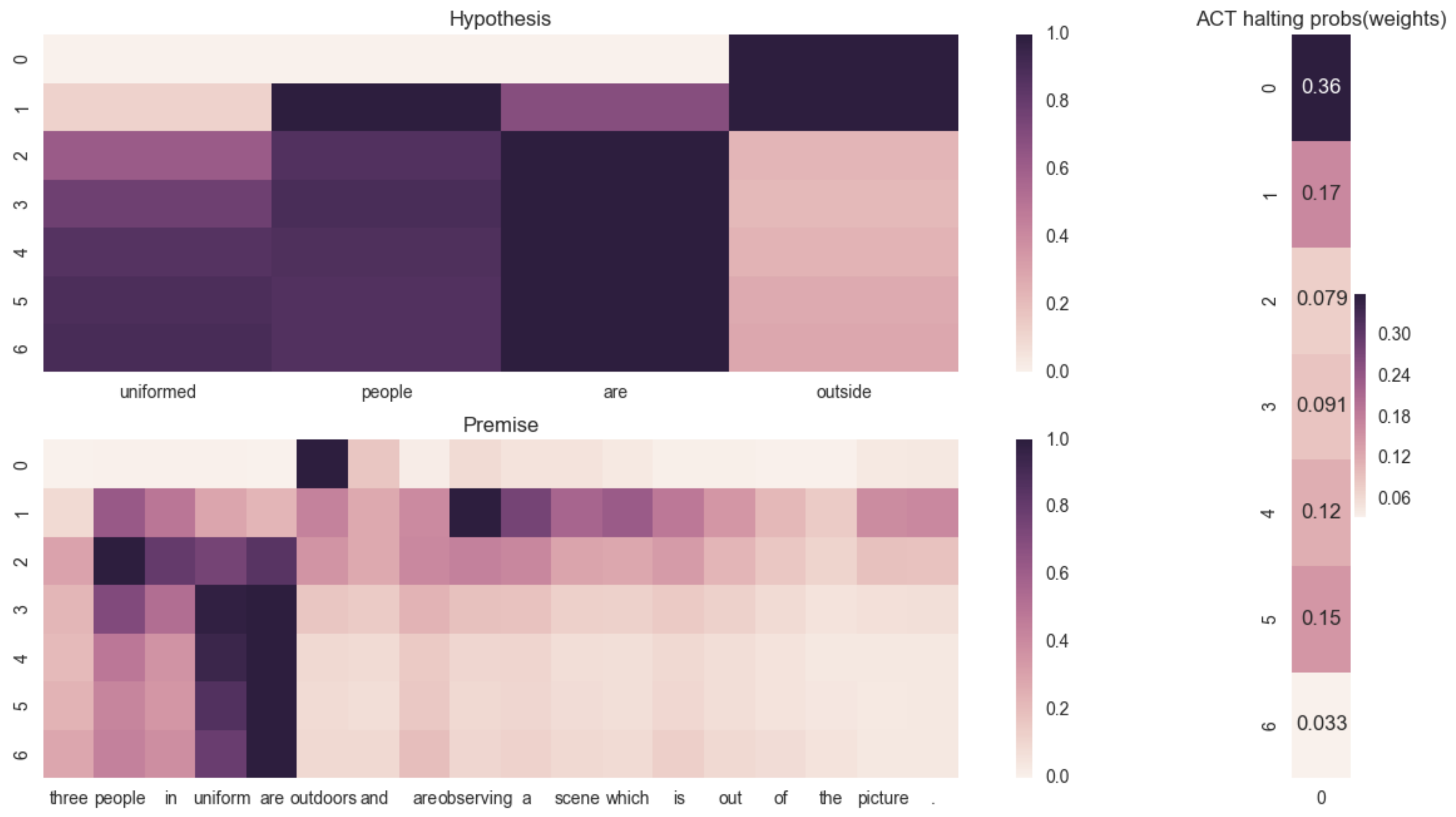


Figure A.1: Clear demonstration of simple synonym resolution in the IAA model.

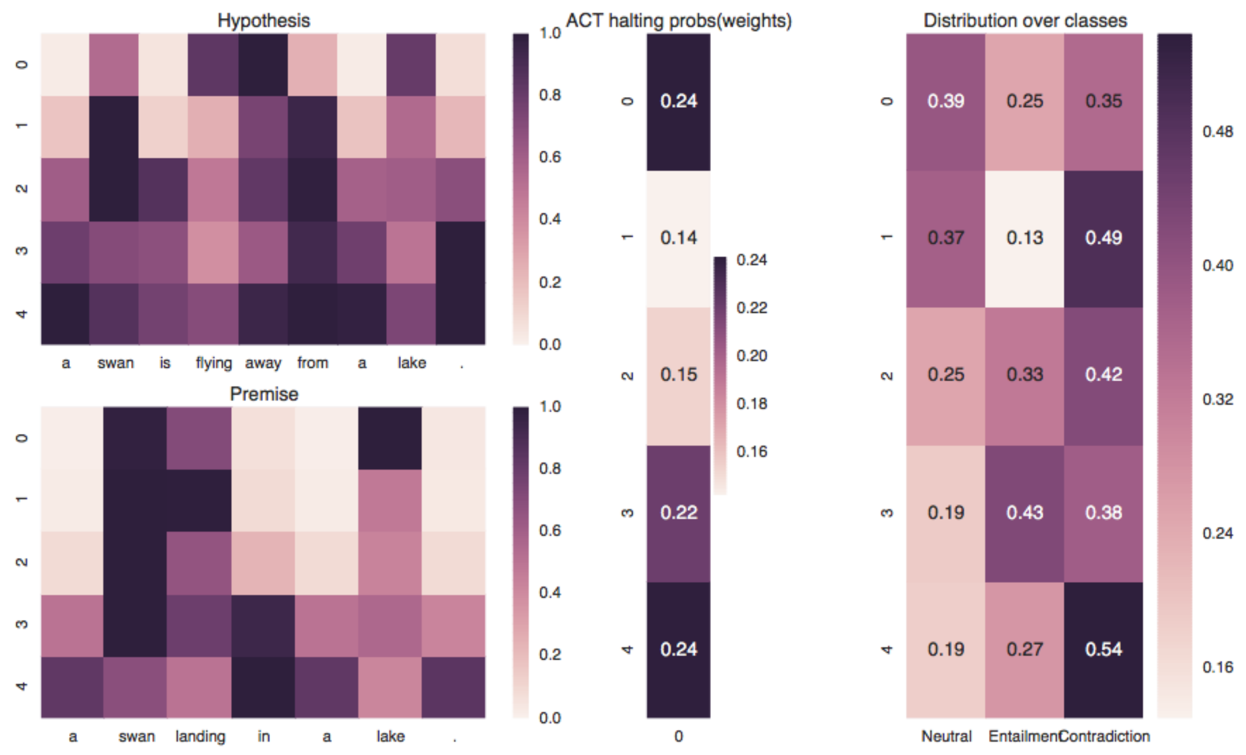


Figure A.2: ADA example showing an instance where specific information has drastically changed the decision, even when the premise and hypothesis have a lot of similar words. Here, the focus on ”in” being different from ”away from” is key to correctly resolving the inference.

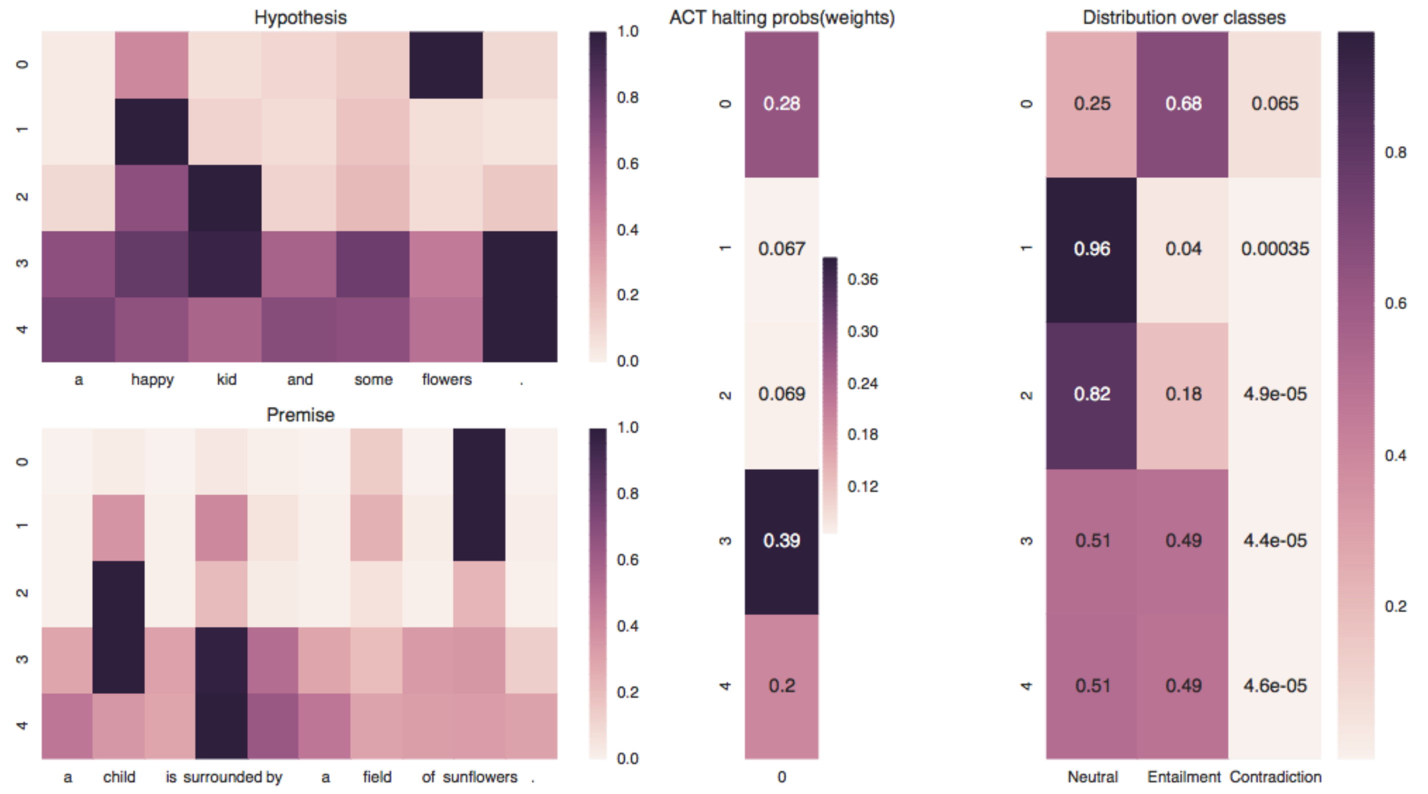


Figure A.3: ADA example demonstrating that more steps does not always help to correctly resolve inferences. Here we see the model becoming less sure in the later stages.

Bibliography

Abadi Martín, Agarwal Ashish, Barham Paul, Brevdo Eugene, Chen Zhifeng, Citro Craig, Corrado Greg S., Davis Andy, Dean Jeffrey, Devin Matthieu, Ghemawat Sanjay, Goodfellow Ian, Harp Andrew, Irving Geoffrey, Isard Michael, Jia Yangqing, Jozefowicz Rafal, Kaiser Lukasz, Kudlur Manjunath, Levenberg Josh, Mané Dan, Monga Rajat, Moore Sherry, Murray Derek, Olah Chris, Schuster Mike, Shlens Jonathon, Steiner Benoit, Sutskever Ilya, Talwar Kunal, Tucker Paul, Vanhoucke Vincent, Vasudevan Vijay, Viégas Fernanda, Vinyals Oriol, Warden Pete, Wattenberg Martin, Wicke Martin, Yu Yuan, Zheng Xiaoqiang. TensorFlow: Large-Scale Machine Learning on Heterogeneous Systems. 2015. Software available from tensorflow.org.

Akhmatova Elena, Aliod Diego Mollá. Recognizing Textual Entailment Via Atomic Propositions // MLCW. 2005.

Bahdanau Dzmitry, Cho Kyunghyun, Bengio Yoshua. Neural Machine Translation by Jointly Learning to Align and Translate // CoRR. 2014. abs/1409.0473.

Bengio Yoshua. Learning Deep Architectures for AI // Foundations and Trends in Machine Learning. 2009. 2. 1–127.

Bengio Yoshua, Ducharme Rejean, Vincent Pascal. A Neural probabilistic language model // Journal of Machine Learning Research. 2003. 3. 1137–1155.

Bengio Yoshua, Lamblin Pascal, Popovici Dan, Larochelle Hugo. Greedy Layer-Wise Training of Deep Networks // NIPS. 2006.

Bengio Yoshua, Simard Patrice Y., Frasconi Paolo. Learning long-term dependencies with gradient descent is difficult // IEEE Trans. Neural Networks. 1994. 5. 157–166.

Blei David M., Ng Andrew Y., Jordan Michael I. Latent Dirichlet Allocation // Journal of Machine Learning Research. 2001. 3. 993–1022.

Recognising Textual Entailment With Logical Inference. // . 2005.

Bourlard H, Kamp Y. Auto-association by multilayer perceptrons and singular value decomposition. // Biological cybernetics. 1988. 59 4-5. 291–4.

Bowman Samuel R., Angeli Gabor, Potts Christopher, Manning Christopher D. A large annotated corpus for learning natural language inference // Proceedings of the 2015 Conference on Empirical Methods in Natural Language Processing (EMNLP). 2015.

Bowman Samuel R., Gauthier Jon, Rastogi Abhinav, Gupta Raghav, Manning Christopher D., Potts Christopher. A Fast Unified Model for Parsing and Sentence Understanding // CoRR. 2016. abs/1603.06021.

Chen Danqi, Bolton Jason, Manning Christopher D. A Thorough Examination of the CNN/Daily Mail Reading Comprehension Task // Association for Computational Linguistics (ACL). 2016.

Chung Junyoung, Gülçehre Çaglar, Cho KyungHyun, Bengio Yoshua. Empirical Evaluation of Gated Recurrent Neural Networks on Sequence Modeling // CoRR. 2014. abs/1412.3555.

Church Kenneth Ward, Hanks Patrick. Word Association Norms, Mutual Information, and Lexicography // Comput. Linguist. III 1990. 16, 1. 22–29.

Colah Christopher. Neural networks, types and functional programming. 2015. [Online; accessed 19-July-2016].

Cooper Robin, Crouch Richard, Eijck Jan van, Fox Chris, Genabith Josef van, Jaspars Jan, Kamp Hans, Pinkal Manfred, Milward David, Poesio Massimo, Pulman Stephen, Briscoe Ted, Maier Holger, Konrad Karsten. Using the Framework. 1996. FraCaS deliverable D16, 136 pages, also available by anonymous ftp from `ftp://ftp.cogsci.ed.ac.uk/pub/FRACAS/del16.ps.gz`.

Dagan Ido, Glickman Oren, Magnini Bernardo. The PASCAL Recognising Textual Entailment Challenge // MLCW. 2005.

Deerwester Scott, Dumais Susan T., Furnas George W., Landauer Thomas K., Harshman Richard. Indexing by latent semantic analysis // JOURNAL OF THE AMERICAN SOCIETY FOR INFORMATION SCIENCE. 1990. 41, 6. 391–407.

Demeester Thomas, Rocktäschel Tim, Riedel Sebastian. Lifted Rule Injection for Relation Embeddings // CoRR. 2016. abs/1606.08359.

Deng Jia, Dong Wei, Socher Richard, Li Li-Jia, Li Kai, Fei-Fei Li. ImageNet: A large-scale hierarchical image database // CVPR. 2009.

Ganitkevitch Juri, Durme Benjamin Van, Callison-Burch Chris. PPDB: The Paraphrase Database // NAACL. 2013.

Gers Felix A, Schmidhuber Jürgen. Recurrent nets that time and count // Neural Networks, 2000. IJCNN 2000, Proceedings of the IEEE-INNS-ENNS International Joint Conference on. 3. 2000. 189–194.

A Probabilistic Setting And Lexical Coocurrence Model For Textual Entailment. // . 2005.

Graves Alan, Mohamed Abdel-rahman, Hinton Geoffrey. Speech recognition with deep recurrent neural networks // Acoustics, Speech and Signal Processing (ICASSP), 2013 IEEE International Conference on. 2013. 6645–6649.

Graves Alex. Adaptive Computation Time for Recurrent Neural Networks // CoRR. 2016. abs/1603.08983.

Graves Alex, Wayne Greg, Danihelka Ivo. Neural Turing Machines // CoRR. 2014. abs/1410.5401.

Grefenstette Edward, Hermann Karl Moritz, Suleyman Mustafa, Blunsom Phil. Learning to Transduce with Unbounded Memory // CoRR. 2015. abs/1506.02516.

Greff Klaus, Srivastava Rupesh Kumar, Koutník Jan, Steunebrink Bas R., Schmidhuber Jürgen. LSTM: A Search Space Odyssey // CoRR. 2015. abs/1503.04069.

Gutmann Michael, Hyvärinen Aapo. Noise-contrastive estimation: A new estimation principle for unnormalized statistical models // JMLR. 2010.

Haghighi Aria, Ng Andrew Y., Manning Christopher D. Robust Textual Inference via Graph Matching // NAACL. 2005.

Harris Zellig. Distributional structure // Word. 1954. 10, 23. 146–162.

Hermann Karl Moritz, Kociský Tomás, Grefenstette Edward, Espeholt Lasse, Kay Will, Suleyman Mustafa, Blunsom Phil. Teaching Machines to Read and Comprehend // CoRR. 2015. abs/1506.03340.

Hinton Geoffrey E. Training Products of Experts by Minimizing Contrastive Divergence // Neural Computation. 2002. 14. 1771–1800.

Hinton Geoffrey E., Osindero Simon, Teh Yee Whye. A Fast Learning Algorithm for Deep Belief Nets // Neural Computation. 2006. 18. 1527–1554.

Hochreiter Sepp, Bengio Yoshua, Frasconi Paolo, Schmidhuber Jrgen. Gradient Flow in Recurrent Nets: the Difficulty of Learning Long-Term Dependencies. 2001.

Hochreiter Sepp, Schmidhuber Jrgen. Long Short-term Memory. 1995.

Hornik Kurt, Stinchcombe Maxwell B., White Halbert. Multilayer feedforward networks are universal approximators // Neural Networks. 1989. 2. 359–366.

Huang Eric H., Socher Richard, Manning Christopher D., Ng Andrew Y. Improving Word Representations via Global Context and Multiple Word Prototypes // ACL. 2012.

Recognizing Textual Entailment Is Lexical Similarity Enough? // . 2005.

Józefowicz Rafal, Zaremba Wojciech, Sutskever Ilya. An Empirical Exploration of Recurrent Network Architectures // ICML. 2015.

Kalchbrenner Nal, Grefenstette Edward, Blunsom Phil. A Convolutional Neural Network for Modelling Sentences // ACL. 2014.

Kingma Diederik P., Ba Jimmy. Adam: A Method for Stochastic Optimization // CoRR. 2014. abs/1412.6980.

Kingma Diederik P., Welling Max. Auto-Encoding Variational Bayes // CoRR. 2013. abs/1312.6114.

Larochelle Hugo, Hinton Geoffrey E. Learning to combine foveal glimpses with a third-order Boltzmann machine // Advances in Neural Information Processing Systems 23. 2010. 1243–1251.

Gradient-based Learning Applied to Document Recognition. // . 1998.

Levy Omer, Dagan Ido, Goldberger Jacob. Focused Entailment Graphs for Open IE Propositions // CONLL. 2014.

Levy Omer, Goldberg Yoav. Neural Word Embedding as Implicit Matrix Factorization // NIPS. 2014.

Levy Omer, Goldberg Yoav, Dagan Ido. Improving Distributional Similarity with Lessons Learned from Word Embeddings // Transactions of the Association for Computational Linguistics. 2015. 3. 211–225.

MacCartney Bill, Galley Michel, Manning Christopher D. A Phrase-Based Alignment Model for Natural Language Inference // EMNLP. 2008.

MacCartney Bill, Grenager Trond, Marneffe Marie-Catherine de, Cer Daniel M., Manning Christopher D. Learning to recognize features of valid textual entailments // NAACL. 2006.

SemEval-2014 Task 1: Evaluation of Compositional Distributional Semantic Models on Full Sentences through Semantic Relatedness and Textual Entailment. // . 2014.

McCulloch W S, Pitts W. A logical calculus of the ideas immanent in nervous activity. 1943. // Bulletin of mathematical biology. 1990. 52 1-2. 99–115; discussion 73–97.

Mikolov Tomas, Chen Kai, Corrado Greg, Dean Jeffrey. Efficient Estimation of Word Representations in Vector Space // CoRR. 2013a. abs/1301.3781.

Mikolov Tomas, Sutskever Ilya, Chen Kai, Corrado Greg S, Dean Jeff. Distributed Representations of Words and Phrases and their Compositionality // Advances in Neural Information Processing Systems 26. 2013b. 3111–3119.

Minsky Marvin, Papert Seymour. Perceptrons - an introduction to computational geometry // DAGLIB. 1987.

Mnih Volodymyr, Heess Nicolas, Graves Alex, Kavukcuoglu Koray. Recurrent Models of Visual Attention // CoRR. 2014. abs/1406.6247.

Munkhdalai Tsendsuren, Yu Hong. Neural Tree Indexers for Text Understanding // CoRR. 2016. abs/1607.04492.

Nair Vinod, Hinton Geoffrey E. Rectified Linear Units Improve Restricted Boltzmann Machines // ICML. 2010.

Och Franz Josef, Ney Hermann. A Systematic Comparison of Various Statistical Alignment Models // Computational Linguistics. 2003. 29. 19–51.

Parikh Ankur P., Tackstrom Oscar, Das Dipanjan, Uszkoreit Jakob. A Decomposable Attention Model for Natural Language Inference // CoRR. 2016. abs/1606.01933.

Pennington Jeffrey, Socher Richard, Manning Christopher D. Glove: Global Vectors for Word Representation // EMNLP. 2014.

Rocktäschel Tim, Grefenstette Edward, Hermann Karl Moritz, Kociský Tomás, Blunsom Phil. Reasoning about Entailment with Neural Attention // CoRR. 2015. abs/1509.06664.

The Perceptron: a Probalistic Model for Information Storage and Organization in the Brain. // . 1958.

Neurocomputing: Foundations of Research. // . Cambridge, MA, USA: MIT Press, 1988. Learning Representations by Back-propagating Errors, 696–699.

Salvo Braz Rodrigo de, Girju Roxana, Punyakanok Vasin, Roth Dan, Sammons Mark. An Inference Model for Semantic Entailment in Natural Language // IJCAI. 2005.

Schmidhuber Jurgen. Learning to Control Fast-Weight Memories: An Alternative to Dynamic Recurrent Networks // Neural Computation. 1992. 4. 131–139.

Schmidhuber Jürgen. Self-Delimiting Neural Networks // CoRR. 2012. abs/1210.0118.

Schuster M., Paliwal K.K. Bidirectional Recurrent Neural Networks // Trans. Sig. Proc. XI 1997. 45, 11. 2673–2681.

Recursive Deep Models for Semantic Compositionality Over a Sentiment Treebank. // . 2013.

Sordoni Alessandro, Bachman Phillip, Bengio Yoshua. Iterative Alternating Neural Attention for Machine Reading // CoRR. 2016a. abs/1606.02245.

Sordoni Alessandro, Bachman Phillip, Bengio Yoshua. Iterative Alternating Neural Attention for Machine Reading // CoRR. 2016b. abs/1606.02245.

Srivastava Nitish, Hinton Geoffrey E., Krizhevsky Alex, Sutskever Ilya, Salakhutdinov Ruslan. Dropout: a simple way to prevent neural networks from overfitting // Journal of Machine Learning Research. 2014. 15. 1929–1958.

Sutskever Ilya, Vinyals Oriol, Le Quoc V. Sequence to Sequence Learning with Neural Networks // CoRR. 2014. abs/1409.3215.

Sutton Richard S., McAllester David A., Singh Satinder P., Mansour Yishay. Policy Gradient Methods for Reinforcement Learning with Function Approximation // Advances in Neural Information Processing Systems 12 (NIPS 1999). 2000. 1057–1063.

T Siegelmann hava, , Sontag Eduardo D. Turing Computability with Neural Nets // MLCW. 1991.

Theano Development Team . Theano: A Python framework for fast computation of mathematical expressions // arXiv e-prints. V 2016. abs/1605.02688.

Vendrov Ivan, Kiros Ryan, Fidler Sanja, Urtasun Raquel. Order-Embeddings of Images and Language // CoRR. 2015. abs/1511.06361.

Vilnis Luke, McCallum Andrew. Word Representations via Gaussian Embedding // CoRR. 2014. abs/1412.6623.

Vinyals Oriol, Kaiser Lukasz, Koo Terry, Petrov Slav, Sutskever Ilya, Hinton Geoffrey E. Grammar as a Foreign Language // CoRR. 2014. abs/1412.7449.

Wang Peilu, Qian Yao, Soong Frank K., He Lei, Zhao Hai. Part-of-Speech Tagging with Bidirectional Long Short-Term Memory Recurrent Neural Network // CoRR. 2015. abs/1510.06168.

Welling Max, Rosen-Zvi Michal, Hinton Geoffrey E. Exponential Family Harmoniums with an Application to Information Retrieval // NIPS. 2004.

Weston Jason, Chopra Sumit, Bordes Antoine. Memory Networks // CoRR. 2014. abs/1410.3916.

Yang Zichao, Yang Diyi, Dyer Chris, He Xiaodong, Smola Alex, Hovy Eduard. Hierarchical attention networks for document classification // Proceedings of the 2016 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies. 2016.

Young Peter, Lai Alice, Hodosh Micah, Hockenmaier Julia. From image descriptions to visual denotations: New similarity metrics for semantic inference over event descriptions // TACL. 2014. 2. 67–78.