Inferring Sentence Features from Sentence Embeddings

Jonas Vetterle

Supervised by

Dr. Sebastian Riedel

Dr. Pontus Stenetorp

This report is submitted as part requirement for

the **MSc in Machine Learning** at **University College London**.

It is substantially the result of my own work except where explicitly

indicated in the text. The report may be freely copied and distributed

provided the source is explicitly acknowledged.

Department of Computer Science

University College London

September 5, 2017

Abstract

This thesis investigates which sentence features are encoded by Recurrent Neural Networks (RNNs). We train two state-of-the-art RNN sentence encoders and compare them against a CBOW (continuous-bag-of-words) baseline, which represents sentences by averaging pre-trained word vectors. We evaluate the models on three tasks: Predicting (i) sentence length (ii) word content, and (iii) dependency tags.
Our findings are that on tasks where information about the full sentence or the order of words is not important (*e.g.* the word content task) the CBOW baseline performs on par with the RNN encoders. And on tasks where information about the full sentence or order of words is important (*e.g.* the sentence length and dependency tag tasks), the RNNs outperform the CBOW baseline.
The main contribution of this thesis is to show that dependency tags can be retrieved from a classifier which has only access to the sentence embedding, and the embeddings of two words. This provides some evidence that RNNs are capable of learning syntactic relationships between words, even if this was not part of the objective they were trained on.
Upon further inspection of sentence embeddings, we also illustrate that certain dimensions carry information about specific aspects of the sentence. We show that there are differences across models with respect to how strongly, and in how many dimensions such aspects are encoded.

Acknowledgements

I would like to say thanks to the UCL Machine Reading group for a great course on Statistical Natural Language Processing. In particular, I want to thank my supervisors Pontus Stenetorp and Sebastian Riedel for their help and guidance on writing my thesis.
Lastly I want to thank Victor Schmidt for countless fun revision sessions.

Contents

List of Figures

List of Tables

Chapter 1

Introductory Material

1.1 Motivation

The history of Machine Learning can be characterized by periods in which certain schools of thought were more popular than others. One of these schools of thought is *connectionism* which aims at creating Artificial Neural Networks (ANNs). ANNs, which have their beginnings in the 1940s, underwent a remarkable revival in the last decade. The abundance of data and ever growing computational resources made it possible to train larger, *deep*, neural networks. The concept of training deep ANNs is referred to as 'Deep Learning' and has led to a step change in performance compared to previous techniques. Today, deep ANNs are able to solve tasks which require 'human-like' capabilities. For example, in the annual ImageNet competition (Russakovsky et al., 2015), state-of-the-art models reach lower error rates on tasks like object recognition than humans. Similarly, Silver et al. (2016) developed an algorithm that was able to beat the world champion in the game of Go, an achievement that seemed to be out of reach for years to come.

Despite their success on a range of tasks, there is a reluctance to use ANNs in areas where interpretability of the model is important. Such areas include for example medical diagnoses or self-driving cars. Indeed, it may be possible for algorithms to predict with higher accuracy which patients are going to develop tumours, or to maneuver cars in such a way that there are fewer accidents. But, perhaps rightly so, we might not feel confident enough to trust these algorithms if we cannot interpret how they make decisions - even if they might make better decisions than us.

Research has therefore been conducted with the goal of understanding better what ANNs learn. For particular types of ANNs such as Convolutional Neural Networks (CNNs), which are used frequently when image or video data is involved, this has in part been achieved. In the case of images it is easy to visualize that CNNs learn simple shapes like edges and circles on the first layers, which are combined into more complex shapes in subsequent layers. Research has also been conducted into letting CNNs explain themselves why a specific classification has been made (Park et al., 2016).

If instead of images, textual data such as books or newspaper articles are involved, a different type of ANN, the Recurrent Neural Network (RNN) is often used. For example, RNNs are used in areas such as translation, document classification or sentiment analysis. While RNNs often outperform non-ANN models in the natural language domain, they suffer from the same lack of interpretability as other ANN architectures. Recent research is trying to fill this gap. One such example is the attention mechanism, which allows to visualize which words an RNNs is focusing on when performing a particular task (Rocktäschel et al., 2015). However, despite the success of RNNs on a range of tasks, a lot is still unknown about what it is they learn.

The main interest of this thesis are sentence embeddings, which are numerical representations of sentences. It is possible to obtain sentence embeddings by letting an RNN 'read' a sentence and retrieving its internal memory once it is finished. Our goal is to investigate if it is possible to infer sentence features from these sentence embeddings. Adi et al. (2016) show that it is possible to infer basic features like sentence length, word content and word order from sentence embeddings. And Linzen et al. (2016) find that RNNs are able to learn syntactic features of sentences.

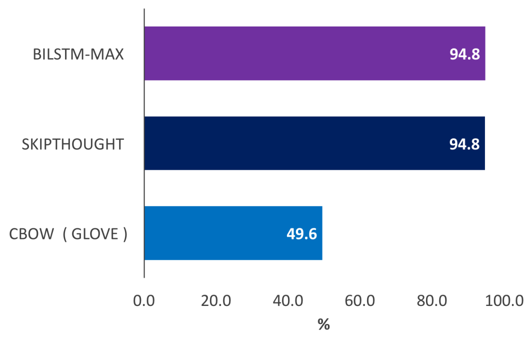
The main contribution of this thesis is to show that it is possible to retrieve syntactic features such as *dependency tags* from neural sentence embeddings.

1.2 Main results

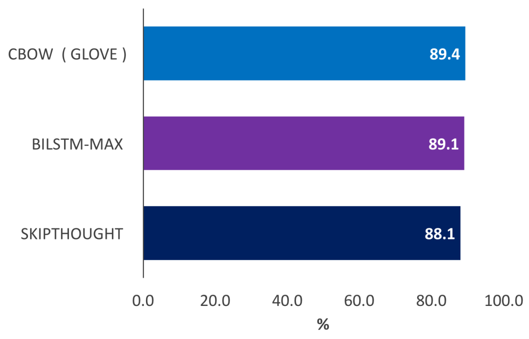
We train two state-of-the-art RNN sentence encoders: The Skip-thought (Kiros et al., 2015) and the BiLSTM-max (Conneau et al., 2017) models. Since our goal is to analyze what features of a sentence RNNs learn, we also use a simple, yet high-performing non-RNN baseline referred to as CBOW ('continuous-bag-of-words') for comparison purposes. The CBOW model represents sentences as the average of pre-trained word embeddings.

Figure 1.1: Summary of key results (weighted average F1 score)

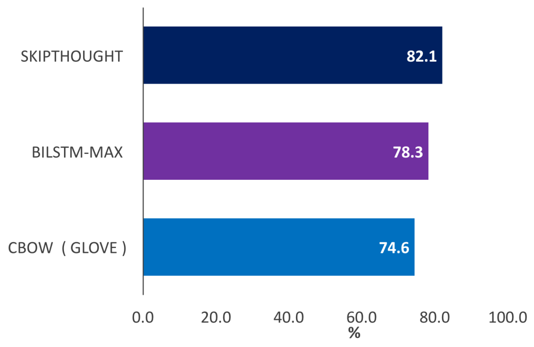
(a) Sentence length task



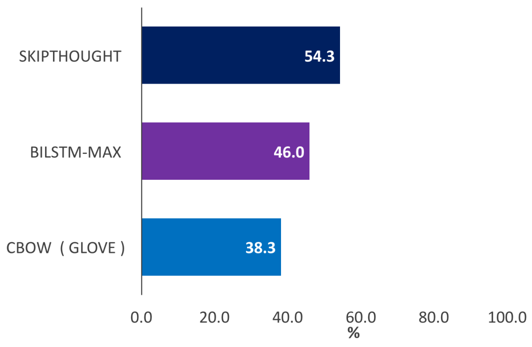
(b) Word content task



(c) Dependency tag task (inc. majority class)



(d) Dependency tag task (exc. majority class)



The key results are as follows:

1. **Sentence length task:** The highest performing model is BiLSTM-max, which achieves a weighted average F1 score of 94.8% (Figure 1.1 panel (a)).

The CBOW model only achieves 49.6%. In other words, the RNN outperforms the non-RNN model by **45.2** percentage points. This provides evidence that RNNs encode information about sentence length in sentence embeddings;

2. **Word content task:** The highest performing model is CBOW, which achieves a weighted average F1 score of 89.4% (Figure 1.1 panel (b)). This is on par with the RNN-based sentence encoders and therefore provides evidence that both RNNs and non-RNN models encode information about the presence of words in sentence embeddings;

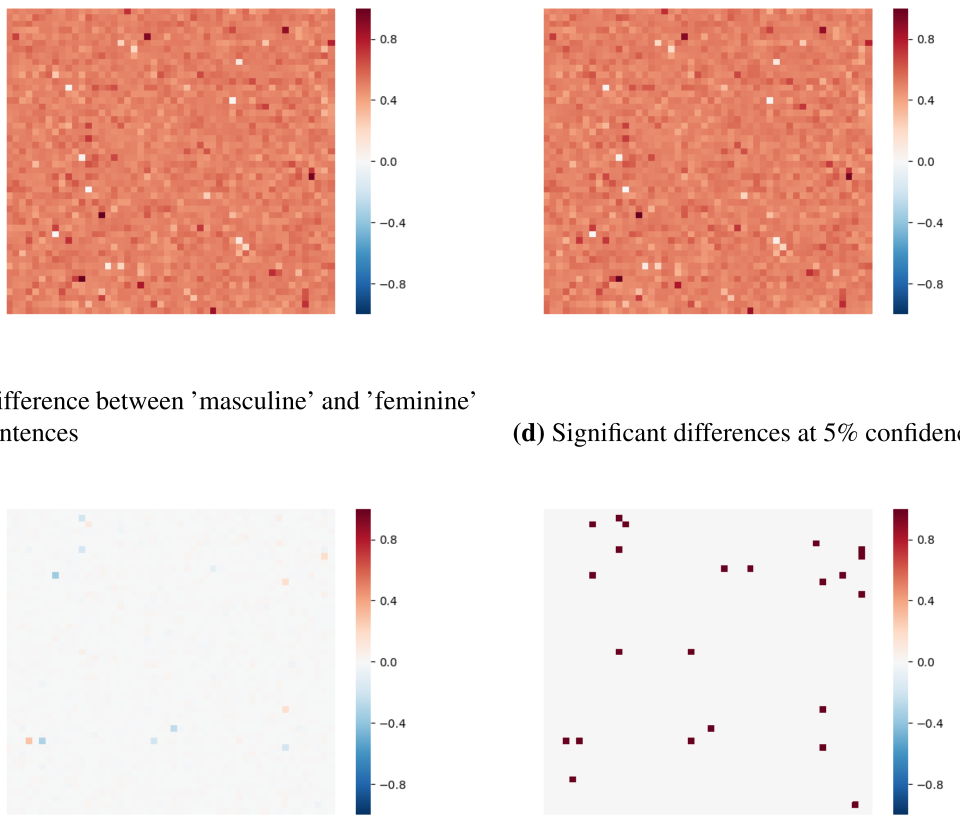
3. **Dependency tag task (inc. majority class):** The highest performing model is Skip-thought, which achieves a weighted average F1 score of 82.1% (Figure 1.1 panel (c)). The CBOW model only achieves 74.6%. In other words, the RNN outperforms the non-RNN model by **7.5** percentage points. This provides some evidence that RNNs encode information about the syntactic relationships between words in sentence embeddings; and

4. **Dependency tag task (exc. majority class):** Some words have a particular dependency tag attached to it very frequently. A naive model could just learn these *majority classes* by heart and achieve good performance. For that reason, in this task the majority class of every word is excluded. The highest performing model is Skip-thought, which achieves a weighted average F1 score of 54.3% (Figure 1.1 panel (d)). The CBOW model only achieves 38.3%. In other words, the RNN outperforms the non-RNN model by **16** percentage points. This provides further evidence that RNNs encode information about the syntactic relationships between words in sentence embeddings;

In sum, RNN sentence encoders outperform non-RNN sentence encoders on tasks in which information of the full sentence is important (*i.e.* the sentence length and dependency tag tasks). The main contribution of this thesis is to show that RNNs 'learn' about the syntactic dependencies between words. This is remarkable, because the models were not explicitly trained towards this objective.

Apart from this main contribution, we are also interested in visualizing, and quantifying, what RNNs learn. To that end, we use our models to obtain embeddings for sentence pairs belonging to three different themes. The themes are *masculine vs. feminine*, *present vs. past* and *singular vs. plural*. An example for the theme *masculine vs. feminine* would be the sentence pair ['In Hollywood, there are many actors', 'In Hollywood, there are many actresses'].

Figure 1.2: Skip-thought vector embeddings of masculine vs. feminine sentences



(a) Mean of 'Masculine' sentences (normalized) **(b)** Mean of 'Feminine' sentences (normalized)

(c) Difference between 'masculine' and 'feminine' sentences **(d)** Significant differences at 5% confidence

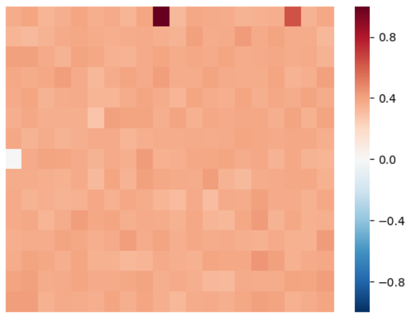
In panels (a) and (b) of Figure 1.2 we show the mean embeddings for 'masculine' and for 'feminine' sentences. Panel (c) shows the result of subtracting the mean 'feminine' embedding from the mean 'masculine' embedding. We observe that there are certain dimensions which are different across the average embeddings, highlighting which dimensions of the sentence embedding contain information about gender. In panel (d) we highlight all dimensions for which the difference between means is statistically significant.

In Figure 1.3 we repeat the same analysis for the CBOW sentence embeddings. Upon inspecting panel (c) it appears that there are no big differences across 'masculine' and 'feminine' sentences. However, panel (d) shows that there are actually many dimensions on which the differences are significant.

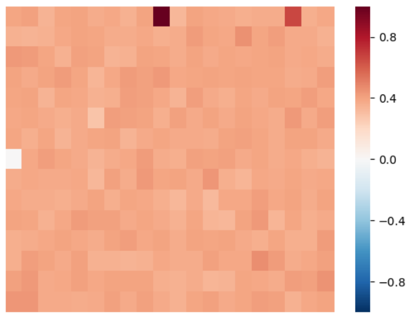
Preliminary findings suggest that the Skip-thought model has relatively fewer dimensions on which differences are significant. But on those dimensions where the differences *are* significant, they are *more* significant than in the CBOW embeddings.

Figure 1.3: Mean GloVe vector embedding of masculine vs. feminine sentences (normalized)

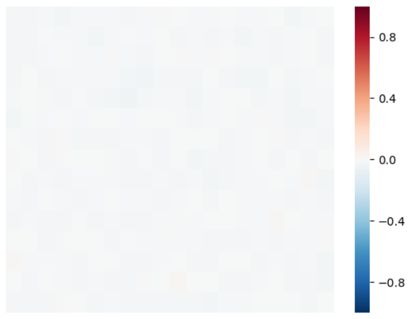
(a) ’Masculine’ sentences



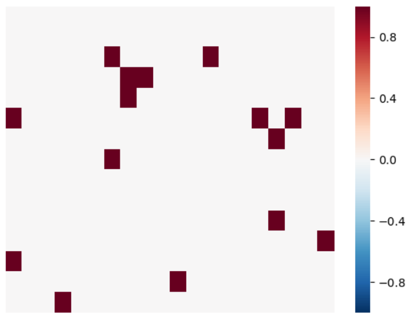
(b) ’Feminine’ sentences



(c) Difference between ’masculine’ and ’feminine’ sentences



(d) Significant differences at 5% confidence



Chapter 2

Literature review

2.1 Machine Learning

Machine Learning refers to a field of study about the design and implementation of algorithms that allow machines to learn from data with the goal of performing certain tasks. It has its theoretical foundations in Mathematics and Statistics. Another large component of it is Computer Science, as its application requires writing computer code. There are many ways in which Machine Learning can be categorized. For example, it can be broken down by the type of data that is available, or the way by which data is collected, which in turn determines what type of information can be inferred from it:

- In **Supervised Learning**, there is a collection of observations and a collection of labels corresponding to each of the observations. The availability of labels makes this type of learning supervised. The goal is to train algorithm that best predicts the label for each observation. Using the trained algorithm it is then possible to make predictions about new observations. The prediction task can be either to predict a categorical label, in which case it is called **Classification**, or numerical, in which case it is called **Regression**

- In **Unsupervised Learning**, only observations but no labels are available. Unsupervised models are therefore mainly concerned with detecting patterns in the training data. For example, grouping similar observations into clusters, or fitting a probability distribution over observations to detect improbable outliers are unsupervised algorithms.

- **Semi-supervised Learning** can be seen as a mixture between Supervised and Unsupervised Learning. That is, a supervised algorithm which uses inputs that were obtained in an unsupervised way would be considered a semi-supervised model.

One could extend the above list to include for example Reinforcement Learning, in which an agent learns from interactions with the environment, and new supervised training examples are created depending on the actions that the agent takes. But the models we use in this thesis belong strictly to the three above-mentioned groups, which is why other types of algorithms are not explained further.

2.1.1 Logistic classifier

As mentioned in Section 2.1, a common problem in Machine Learning is classification, *i.e.* to predict the class c of on observation, out of a finite and discrete set of C classes. When $C = 2$, this model is called a binary logistic classifier, first documented by Cox (1958). In the case of $C > 2$, it is commonly referred to as a multinomial logistic classifier or a softmax classifier.

What all models we trained for the purpose of this thesis have in common is that they are used for classification in one way or another. For example, to predict the next most likely word out of a vocabulary of possible words given a sequence of previous words, or to predict whether a movie review expresses a positive or a negative sentiment. For that reason, we briefly explain here the logistic classifier.

First, we introduce some notation that is used throughout this thesis. Let row vectors be denoted by lowercase bold letters such as $\mathbf{s}$ and let matrices be denoted using bold capital letters such as $\mathbf{M}$.

Let $\mathbf{M} \in \mathbb{R}^{C \times D}$ and $\mathbf{b} \in \mathbb{R}^{C}$ denote the trainable weights and bias of the logistic classifier. D refers to the dimension of the input vector $\mathbf{s}$ and C to the number of classes. The classifier then predicts the class probabilities $\mathbf{y_{pred}}$ as

$$\mathbf{y_{pred}} = \frac{\exp(\mathbf{Ms}+\mathbf{b})}{\sum_{c=1}^{C}\exp(\mathbf{M}_c\mathbf{s}+\mathbf{b}_c)} \tag{2.1}$$

Each element of $\mathbf{y_{pred}}$ takes on values between zero and one. Also the sum of all elements is equal to one by construction. This means that $\mathbf{y_{pred}}$ constitutes a valid probability distribution over classes.

2.1.2 Cross-entropy loss

So far we have seen that Machine Learning can be used to make a prediction about how to classify a certain observation: the logistic classifier mentioned in Section 2.1.1 gives the estimated probabilities of the observation belonging to each of the classes. But how to measure the performance of the classifier? It would be straightforward to say a classification is 'good' if the predicted class, *i.e.* the class with the highest estimated probability, is the same as the true class and 'bad' if it is not. But what if the classifier underestimates the probability of the true class by just a tiny amount and therefore classifies the observation incorrectly. This would also be 'bad' but not as much as if the classifier had underestimated the true probability by a large margin. We will therefore explain a performance measure which is often used in classification tasks, because it judges the accuracy of a classifier in a more nuanced way.

The amount by which a prediction deviates from the ground truth is called *loss*. It is calculated by some loss function which takes as input the estimated and the true class labels. In the case of a classification task these would be the estimated and the true class probabilities $\mathbf{y_{pred}}$ and $\mathbf{y_{true}}$.

$$L = f(\mathbf{y_{pred}}, \mathbf{y_{true}}) \tag{2.2}$$

where $\mathbf{y_{pred}}$ depends on the underlying machine learning algorithm $g(.)$ with parameters θ, and input data $\mathbf{x}$. Hence $\mathbf{y_{pred}} = g_\theta(\mathbf{x})$. The goal of training a machine learning algorithm is then to find θ which makes $\mathbf{y_{pred}}$ as similar as possible to $\mathbf{y_{true}}$, *i.e.* the θ which minimizes the loss.

A loss function commonly used in the case of classification problems is the cross-entropy loss. As we will show, this is in part because minimizing the cross-entropy loss leads to parameters which maximize the likelihood of observing the training data.

The cross-entropy loss is closely related to the concept of Kullback-Leibler (KL) divergence (Kullback and Leibler, 1951). The KL divergence is defined as

$$D_{KL}(P||Q) = \sum_i P(i) \log \frac{P(i)}{Q(i)} \tag{2.3}$$

and it measures the similarity between two probability distributions P and Q. It can be shown that $D_{KL}(P||Q) \geq 0$ and $D_{KL}(P||Q) = 0$ if and only if $P = Q$. The KL divergence can be rewritten as the difference between cross-entropy and entropy:

$$D_{KL}(P||Q) = \sum_i P(i) \log \frac{1}{Q(i)} - \sum_i P(i) \log \frac{1}{P(i)} \tag{2.4}$$

Continuing with the softmax classifier example from above, let $P = \mathbf{y_{true}}$ be the true distribution, *i.e.* a one-hot vector denoting the correct class, and $Q = \mathbf{y_{pred}}$ be the estimated distribution, *i.e.* the output of the softmax classifier. For any sentence, the cross entropy loss between the one-hot encoded class-label $\mathbf{y_{true}}$ and the predicted class probabilities $\mathbf{y_{pred}}$ is therefore calculated as

$$L_{xent} = -\mathbf{y_{true}}^T \log \mathbf{y_{pred}} \tag{2.5}$$

The KL divergence is affected by the parameters of the model only through their effect on the estimated class distribution $\mathbf{y_{pred}}$. Minimizing the cross-entropy between the estimated class distribution and the true distribution is therefore the same as minimizing the KL divergence.

Minimizing the KL divergence is also equivalent to maximizing the likelihood. To see this, note that the logarithm is a monotonic function and that maximizing the likelihood is therefore equal to maximizing the log-likelihood. But since $\mathbf{y_{true}}$ is a one-hot vector, Eq. 2.5 is just the negative log-likelihood. Consider for example a scenario in which there are 3 classes. A particular training example belongs to class 2, such that $\mathbf{y_{true}} = [0,1,0]$. Assume that the model produces class probability estimates $\mathbf{y_{pred}} = [p_1, p_2, p_3]$. The cross-entropy is therefore $-\log p_2$. This is minimized by maximizing p_2, *i.e.* the estimated likelihood that the training instance belongs to class 2.

In summary, minimizing Eq. 2.5 (the cross-entropy) with respect to the weights of the model means minimizing the KL divergence between the true and the estimated class probabilities by finding the maximum likelihood parameters.

2.1.3 A brief history of Machine Learning

Up until this point, we have explained Machine Learning algorithms in terms of their inputs, their outputs and how to evaluate their performance. Crucially what is missing is a description of how inputs are transformed into outputs, *i.e.* a description of the machine learning algorithm itself. Domingos (2015) offers a classification of algorithms by describing the five main schools of thought within Machine Learning:

- **Symbolists**, who believe that, in addition to data, some pre-existing knowledge is needed to deduct more knowledge;
- **Connectionists**, who try to imitate the learning process of the brain by adjusting the strengths of connections within a network of artificial neurons;
- **Evolutionaries**, who rather than adjusting parameters in a model, believe that a process similar to natural selection, which evolves different model structures

by combining parts of computer programs is the key to improving algorithms;

- **Bayesians**, who are concerned with how to deal with the fact that all knowledge is uncertain, how to deal with noisy, incomplete or contradictory information, and how to incorporate evidence into prior believes; and

- **Analogizers**, whose view it is that knowledge can be gained by recognizing similarities between entities and based on that deducting further similarities.

The history of Machine Learning can be broken down into phases in which certain types of algorithms were more popular than others. The beginning of Machine Learning as a field is often associated with Turing (1950), which is considered a primer of evolutionary algorithms, and the invention of the Perceptron algorithm (Rosenblatt, 1958). This is despite the fact that other theoretical discoveries such as Bayes Theorem (Bayes and Price, 1763) and Ordinary Least Squares (Legendre, 1805) were made centuries before. The perceptron algorithm also marks the beginning of connectionism, or artificial neural networks.

Shortly after, Solomonoff (1964) introduced important Bayesian concepts into machine learning. In particular, Solomonoff's theory of inductive inference, *e.g.* predicting a symbol based on a sequence of previous symbols, is used in a range of machine learning algorithms, including the models used in this thesis. With the application of the Nearest Neighbour classifier Cover and Hart (1967) introduced models that are based on finding analogies between observations and which have been used extensively in pattern recognition.

Papert and Minsky (1969), while highlighting some of the strengths of the perceptron, criticize this algorithm by pointing out a number of limitations that were previously unknown. Their book is seen as the main reason for a halt in connectionist research in favour of symbolic algorithms. Symbolic machine learning led to called knowledge-based systems which were initially successful in the 1970s and 1980s. But interest in these systems declined when it became clear that building up a knowledge base is time- and resource-intensive and prone to errors and subjectivity (Domingos, 2015). At the same time, the pessimistic view on connectionism turned out to be ungrounded as new theoretical discoveries were made in the 1970 and 1980s. In particular, the invention of the general method for Automatic Differentiation by Linnainmaa (1976), which is the basis of the widely used Backpropagation algorithm (Rumelhart et al., 1986) meant that ANNs could be trained in a more efficient manner than previously thought possible.

Despite early successes of ANNs, a lack of training data and insufficient computational resources meant that interest in connectionist algorithms declined in the 1990s. This is in part due to the introduction of the Support Vector Machine (SVM) by Cortes and Vapnik (1995), which achieved better performance than state-of-the-art ANNs at the time. ANNs remained unpopular until Hinton et al. (2006) showed that ANNs can be trained more easily if rather than initializing parameters randomly, they are pretrained in an unsupervised manner. This discovery sparked a renewed interest in ANNs among researchers. In addition, the availability of greater amounts of training data and more computational resources have led to major breakthroughs. As a result, ANNs, and variants of it, are today one of the most widely used algorithm in Machine Learning. All models used in this thesis are ANN-based, which is why we will explain them in more detail in the next section.

2.1.4 Artificial Neural Networks (ANNs)

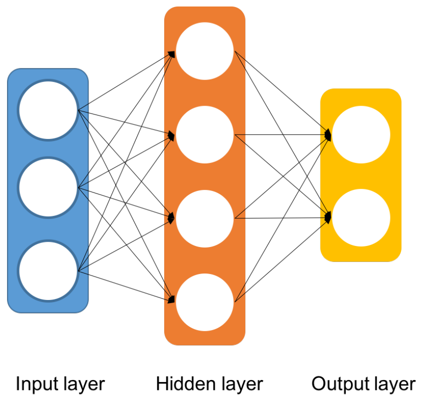
As mentioned in the previous section, an ANN is a type of Machine Learning algorithm, *i.e.* a function which maps inputs to outputs and which depends on some parameters θ.

$$g_\theta(\mathbf{x}) = \mathbf{a} \tag{2.6}$$

Depending on the particular problem at hand, the output can either be a scalar value, or a vector, which may be the input to *e.g.* a softmax classifier as explained in Section 2.1.1.

ANNs consist of a set of nodes, also called neurons or units, which are grouped together in layers. The first and last layers of a network are commonly referred to as the input and output layers, and any layers in between as hidden layers. Neurons between layers may be connected by vertices, also called weights, which are the parameters of the model. If all nodes of a layer are connected with all nodes of the next layer, this is sometimes referred to as a fully connected (FC) layer. A node is said to be activated, if the sum of nodes from the previous layer it is connected with, multiplied by their respective weights, exceed a certain threshold. The threshold is often modelled by a non-linear function such as the sigmoid function[1] or a Rectified Linear Unit (ReLU) (Nair and Hinton, 2010). Nodes and weights can be thought of as metaphors for neurons and dendrites. The process explained above then roughly resembles the way signals are propagated in the brain, which is why this model was given the name artificial neural network. Figure 2.1 depicts a neural network with one (fully-connected) hidden layer, consisting of 4 nodes.

Figure 2.1: A simple artificial neural network (ANN)



ANNs can be formed of arbitrarily many layers, with an arbitrary number of neurons in any layer. As the number of parameters increases with the number of neurons, ANNs can be thought of as a function with arbitrarily many dimensions.

[1]Defined as $S(x) = \frac{\exp(x)}{\exp(x)+1}$. This is a symmetric S-shaped function taking values between zero and one.

Depending on the choice of nonlinear functions applied after each layer, this makes ANNs very powerful non-linear function approximators (Cybenko, 1992).

2.1.5 Optimization

The fact that ANNs are potentially very high dimensional non-linear function approximators is a mixed blessing for researchers, because this makes them inherently difficult to train. Remember from Section 2.1.1 that training a classifier amounts to finding the parameters that lead to a prediction that is as close as possible to the ground truth. How close the prediction is to the ground truth is measured by a loss function. So the objective of training an ANN is finding the parameters that minimize the loss when making a prediction: $L = f(\mathbf{y_{pred}}, \mathbf{y_{true}})$. For now assume that $\mathbf{y_{pred}}$ is the output of a ANN with parameters θ. The objective function then becomes:

$$\hat{\theta} = \underset{\theta}{\operatorname{argmin}} f(g_{\theta}(\mathbf{x}), \mathbf{y_{true}}) \tag{2.7}$$

For $g(.)$ a linear model (*e.g.* linear regression) and certain loss functions (*e.g.* squared loss) there exists an analytical solution to this minimization problem. The same holds for very high dimensional linear models, even though this might be computationally expensive. However, for non-linear functions such as ANNs no analytical solution exists, which is why a lot of research has been devoted to developing good optimization algorithms.

We now discuss discuss two such algorithms that we use in this thesis: the Stochastic Gradient Descent (SGD) and Adaptive Moment Estimation (Adam) algorithms. Both are gradient-based methods and can be used in the backpropagation algorithm (Rumelhart et al., 1986) which we sketch here briefly. The backpropagation algorithm consists of a *forward pass* and a *backward pass*. The former refers to the processes of making a prediction by feeding an input into the network, following the calculations at each layer of the network, and retrieving the prediction at the output layer. Once the prediction is made and the loss is computed, as decision has to be made as to which parameters to change and by how much, in order to reduce the loss. This is referred to as the *backward pass* and is where the the following optimization methods are used.

2.1.5.1 Stochastic Gradient Descent (SGD)

SGD is a variant of Gradient Descent (GD). The basic idea of (GD) is to calculate the gradient $\nabla L_\theta(\mathbf{x})$, which is a vector containing the first derivatives of the loss with respect to all the parameters of the model. It has the interpretation of the direction of steepest ascent of the loss function. In other words, it indicates how to change the parameters if the goal was to increase the loss. Since the goal of training is to minimize the loss, parameters are therefore adjusted in the opposite direction of the gradient. To make the updates more stable, the parameters are only adjusted by a proportion ε of the gradient.

$$\theta \leftarrow \theta - \varepsilon \times \nabla L_\theta(\mathbf{x}) \tag{2.8}$$

GD calculates the gradient over all data points and is therefore exact. However for large data sets, this can be computationally expensive. Therefore, in practice SGD is used instead. SGD performs the update shown in Eq. 2.8 on a subset (a 'minibatch') of the training data. SGD is generally more unstable than GD, but on average moves into the 'right' direction if the examples of the minibatch are drawn at random. Both GD and SGD have some convergence properties under certain assumptions (Barber, 2012). Both the learning rate ε and batch size are hyperparameters which may have a substantial impact on convergence and therefore need to be chosen carefully (see Section 2.1.7).

2.1.5.2 Adaptive Moment Estimation (Adam)

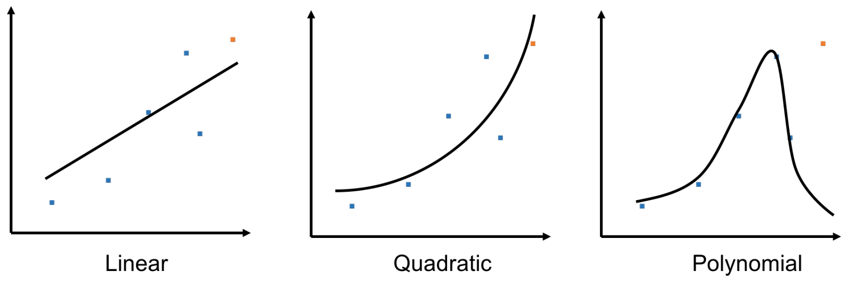
SGD has a number of shortcomings. First, while converging faster than GD, it still converges quite slowly. This is particularly the case in 'flat' regions of the loss curves, where the gradient is small. And second, it updates all parameters at the same rate. But the data may be sparse and the features may have very different frequencies. It could therefore be advantageous to update less frequently occurring features at a higher learning rate.

The Adam optimizer (Kingma and Ba, 2014) addresses exactly these two points. First, it uses a form of momentum, similar to the Momentum optimizer (Qian, 1999) or Nesterov Accelerated Gradient (NAG) (Nesterov, 1983). That is, Adam stores an exponentially decaying moving average of the mean and the variance of the gradient. This form of momentum helps the optimizer move faster through flat regions of the loss function. And secondly, similar to Adagrad (Duchi et al., 2010) and Adadelta (Zeiler, 2012), Adam has different learning rates for each parameter, allowing for larger updates for less frequently occurring features.

2.1.6 Regularization

We have seen that ANNs are high-dimensional and non-linear function approximators, which make them a powerful machine learning algorithm. But as discussed in the previous section, this comes at the cost of making them hard to optimize. There is yet another risk that comes with complex function approximators such as ANNs: overfitting.

Figure 2.2: Under- and Overfitting



Overfitting describes the situation in which the parameters of a model are adjusted such that it can explain the observations in the training data increasingly well, at the expense of its performance on an unseen data set. This can for example happen when a model is too complex for a given data set. We show this situation in Figure 2.2 where a high-order polynomial function is used to fit a quadratic data set. As can be seen, a model that is too powerful might be able to fit the training data (blue dots) very well, but does not perform well on unseen data.

The problem of overfitting arises in part due to the maximization technique. Remember that we showed in Section 2.1.2 that minimizing the cross entropy loss is equivalent to maximizing the likelihood of observing the (training) data. That is, by construction we train the model to explain the training data as well as possible. If we fit a very potent function approximator, such as an ANN, to a data set, then we risk explaining the data *too* well, such that the model fails to perform well on unseen data.

Regularization is a way to prevent overfitting even when fitting a model using maximum likelihood. We explain briefly two particular types of regularization, L2 regularization and dropout, which we will use in later chapters.

1. **L2-regularization:** This amounts to adding the euclidean norm of the weight vector to the loss function (Tikhonov and Arsenin, 1977). In other words, the loss function includes the squared values of all weight parameters, multiplied by a hyperparameter that controls how much regularization we want to apply. This way, the model gets penalized for having few very large weights and this will therefore induce more equal, smaller weights.

2. **Dropout** (Srivastava et al., 2014): This is a way of regularizing ANNs in particular. During training, there is a chance p that any neuron is 'turned off'. It therefore does not contribute to the final prediction and will also not receive an update during the backward pass. This way the model can at no point in time rely on any specific neuron and is instead forced to use a combination of neurons. During inference all neurons are 'turned on'.

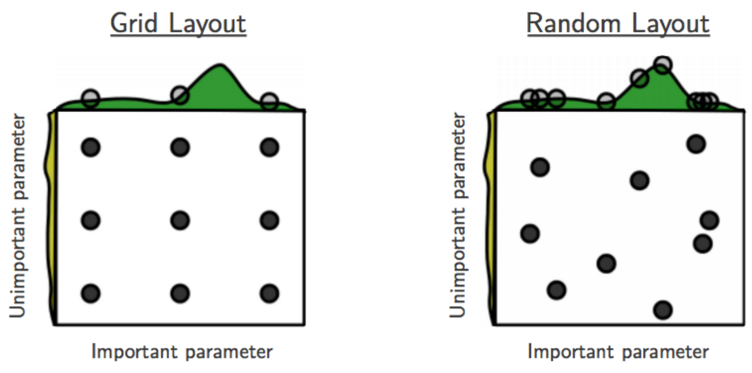
As we can see, the result in both cases is that the model will make use of a combination of parameters rather than relying on only a few large ones. This helps the model to generalize better to unseen data. Both types of regularization require choosing some hyperparameters.

2.1.7 Hyperparameter optimization

We conclude the overview of neural networks and how to optimize them with the laborious, yet important task of hyperparameter optimization. In the previous sections we have encountered a number of hyperparameters. These concern for example model architecture (the number of layers and units, type of non-linear activation functions), optimization (which optimizer to use, the learning rate, batch size, other optimizer-specific hyperparameters), regularization and other choices such as which loss function to use. Any combination of hyperparameters may lead to a different outcome and by optimizing, *i.e.* finding the best hyperparameters, the performance of models may be increased substantially.

Two commonly used ways of conducting hyperparameter optimization are grid search and random search, the latter of which is usually preferred. Bergstra and Bengio (2012) show that randomly selected values is both empirically and theoretically superior to performing a grid search. The reason for this is that using random search may explore parts of the parameter space which are not necessarily explored in the case of grid search. This idea is illustrated in Figure 2.3.

Figure 2.3: Two ways of performing hyperparameter search



Reproduced from Bergstra and Bengio (2012)

2.2 Natural Language Processing

In Section 2.1 we talked about Machine Learning in very general terms. However there is of course another way of breaking down Machine Learning: according to the domain it is applied to. Areas in which Machine Learning has been used successfully in recent years include Machine Vision, which deals with visual data such as photos or videos and Bioinformatics, where Machine Learning is used for example in drug discovery or prediction of protein structure. The topic of this thesis is Natural Language Processing (NLP), which deals with naturally occurring text, such as books, newspapers or movie reviews, and natural speech. The types of problems NLP is trying to address can be both discriminative, *e.g.* classifying spam emails, or generative *e.g.* generating captions of images.

As explained in Section 2.1.3, the history of Machine Learning is characterized by periods in which certain types of algorithms were more popular than others. This development can also be observed in the NLP domain. Broadly speaking, NLP algorithms can be divided into those that involve some kind of feature-engineering, and ANN-based approaches which do not rely on feature-engineering, but where ANNs are used to extract features. We will review the relevant literature and core concepts in turn.

But before doing that, it is worth reminding ourselves what feature engineering means in the context of NLP and why this may be important at all. Note that Machine Learning algorithms, being based on mathematical models, require a way of performing mathematical operations with their inputs. But natural text consists of strings of characters. These strings may be delineated by spaces and punctuation into words, sentences and paragraphs, but without further transforming them into a numerical input they cannot be used in Machine Learning algorithms. In other words, it is necessary to find numerical **representations** of words, or sentences, or any other unit of text.

For example, consider that we have a vocabulary of V words for each of which we want to find a numerical representation. A straight-forward way would be to assign each word an index, and then represent each word as a one-hot vector. That is, a vector consisting of zeros, apart from the entry corresponding to the word's index, where the vector takes the value one. This would lead to a V-dimensional, sparse vector representation of each word. The elements of each vector are mutually exclusive and this is referred to as a **non-distributed** representation.

There are two reasons why this is not a good way of representing words. First, for large V, computations and storage of these word vectors would be expensive. Moreover, since the vectors are sparse, this representation would also be wasteful. And second, there is a lack of geometric interpretabiltiy. Each word vector would be equally far away in a V-dimensional space. But intuitively, vectors representing similar words should be closer together than others.

A solution to these problems is to use **distributed** representations (Hinton et al., 1986). This means that a word is represented by a combination of features, rather than having one 'feature' for each word. In practice this means that words are represented by vectors whose elements are not mutually exclusive. There are two principal ways of obtaining distributed representations. They can be created manually (*i.e.* through feature engineering) or be learned by an ANN. In either case, NLP tasks are often concerned with using these representations of units of texts as inputs to classifiers.

2.2.1 Feature-based models

As mentioned in Section 2.1.3, in the 1970s, a popular type of Machine Learning algorithm were knowledge-based systems. Winograd (1972) and Welin (1979) used such systems, which required manually writing rules. In a similar spirit Hinton and Shallice (1991) create word representations by describing every word in their corpus according to whether it satisfies one of 68 features, such as 'has-legs' or 'found-near-sea'. A word could therefore be represented as a vector containing only zeros, apart from the indices corresponding to certain features, where it may take non-zero entries depending on whether it can be described by a certain feature. Other researchers (Andrews et al., 2009) let native speakers create features for words based on their meaning.

It is easy to see that this approach does not scale well. Ideally, human expert linguist would be required to annotate every word in the English language according to whether it can be described by an agreed upon set of features. What if it turns out that an important feature has been missed? The process would have to be repeated. And the same process would have to repeated for other languages too. Apart from being time- and labour-intensive this approach is also inherently subjective and prone to errors as is requires human annotators. Therefore, research has been devoted to approaches that do not rely on manual feature engineering but which instead exploit the statistical properties of a corpus.

2.2.1.1 Language Modelling

In the area of Statistical Language Modelling for example, count-based feature engineering has been used successfully for many years. Statistical Language Modelling is an NLP task concerned with assigning a probability to a sequence of words. A language model can either be used to estimate the joint probability of a sequence of words $P(w_1,...,w_T)$, or the conditional probability of a word given a certain sequence of words $P(w_T|w_1,...,w_{T-1})$. An approach that has been widely used is the n-gram model. An n-gram is a sequence of words, where n refers to the lengths of the sequence. Training an n-gram model amounts to counting the frequencies with which n-grams occur in a corpus. A language model can then for example be used to predict the next most probable word given a certain sequence of words. For example, consider the sequence ['The', 'cat', 'sat', 'on', 'the'] for which we wish to predict the most likely next word. If using a 2-gram ('bi-gram') model, we would consider all instances in our corpus in which the sequence ['the', x] occurred, where x is any other word. The most likely word is then the word x which completes the bigram ['the', x] most frequently.

One downside of the n-gram model is that a lot of domain-specific data is required, which may not exist for every task. This problem is more severe the larger

n is. That is because while uni-grams and bi-grams might occur with reasonable frequency in a corpus, the number of possible n-grams grows exponentially as n increases. The frequency with with particular n-grams are observed in the corpus therefore drops quickly for large n. The n-gram model has been extended by many scholars such as Katz (1987) and Kneser and Ney (1993) to handle the cases of missing n-grams. More recently, Brants et al. (2007) applied an n-gram model to a very large corpus consisting of trillions of words.

2.2.1.2 Text classification

Count-based models have also been used in classification tasks. One example is the Naïve Bayes (NB) classifier, which is widely used due to its simplicity and surprisingly good performance. As the name suggests, the model is based on Bayes Theorem (Bayes and Price, 1763), which expresses the conditional probability of A given B as

$$P(A|B) = \frac{P(B|A)P(A)}{P(B)} \tag{2.9}$$

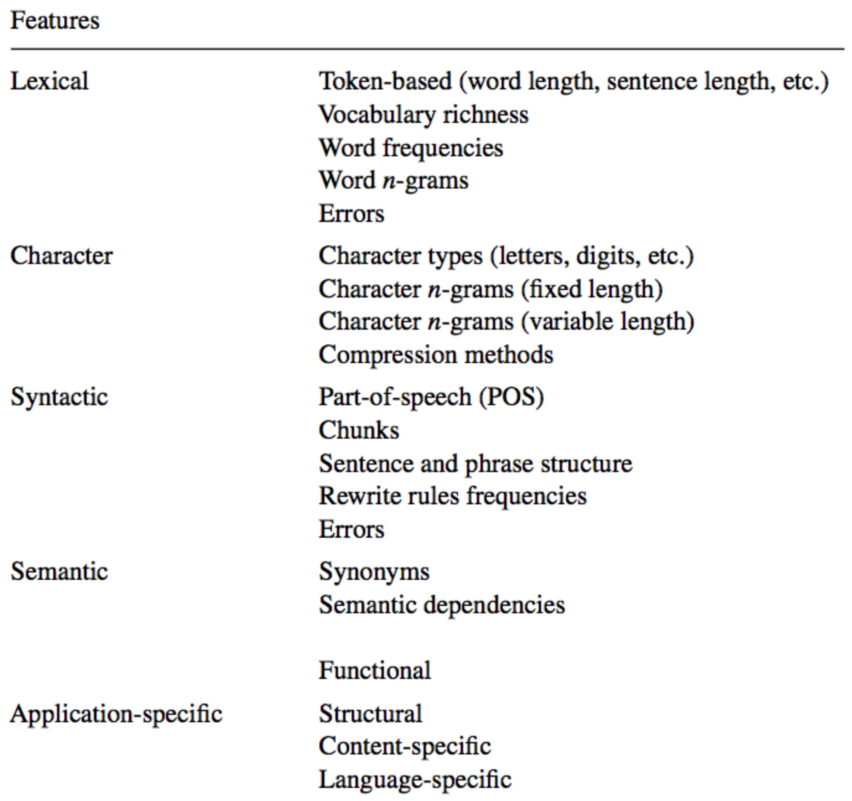
One area in which NB has been applied is spam email detection (Sahami et al., 1998). Here the task is to estimate the probability that an email is spam based on the content of the email. Using NB this would be formulated as estimating $P(spam|w_1,...,w_T)$, where $spam \in \{true, false\}$. By Bayes' Theorem this is equal to

$$P(spam|w_1,...,w_T) = \frac{P(w_1,...,w_T|spam)P(spam)}{P(w_1,...,w_T)} \tag{2.10}$$

NB is 'naïve' because it treats each word as being independent of all other words. As a result $P(w_1,...,w_T) = \prod_{t=1}^{T} P(w_t)$ and $P(w_1,...,w_T|spam) = \prod_{t=1}^{T} P(w_t|spam)$. The NB classifier can therefore be trained by simply counting the number of times each word occurs in spam emails, and how many times it occurs in non-spam emails. Because it treats a training instance just as a collection of, in this case, words, with no particular order, NB is considered a so called 'bag-of-words' approach.

Stamatatos (2009) conducted a survey of features used in authorship attribution models, which are used to predict the author of an article. Table 2.1 is a reproduction of a summary table from the original paper. The author found that the vast majority is models was based on lexical features such as n-gram counts. This is also been the basis in topic-based text classification tasks (Sebastiani, 2002).

Table 2.1: Features used in author attribution models



Features	
Lexical	Token-based (word length, sentence length, etc.)
	Vocabulary richness
	Word frequencies
	Word *n*-grams
	Errors
Character	Character types (letters, digits, etc.)
	Character *n*-grams (fixed length)
	Character *n*-grams (variable length)
	Compression methods
Syntactic	Part-of-speech (POS)
	Chunks
	Sentence and phrase structure
	Rewrite rules frequencies
	Errors
Semantic	Synonyms
	Semantic dependencies
	Functional
Application-specific	Structural
	Content-specific
	Language-specific

Reproduced from Stamatatos (2009)

While bag-of-words models are simple and fast to train, they do not take into account the order in which words occur. That is, this approach ignores potentially useful information about the context of words. Indeed, Table 2.2 shows how changing the word order can change the meaning of a sentence dramatically. For that reason researchers such as Coyotl-Morales et al. (2006), Peng et al. (2004) and Sanderson and Guenter (2006) have used word *n*-grams as features in classification tasks. That is, n-grams where $n > 1$ such that sequences of words are taken into account.

Apart from lexical features, researchers have also used syntactic features in NLP tasks. For example, Sidorov et al. (2013) found that syntactic n-grams, ’sn-grams’, are useful in classification tasks. In particular the authors find that including part-of-speech (POS) n-grams improves performance relative to a regular word n-grams baseline.

Table 2.2: Illustration how word order can change the meaning of a sentence

Only he told his wife that he trusted her.
He only told his wife that he trusted her.
He told only his wife that he trusted her.
He told his only wife that he trusted her.
He told his wife only that he trusted her.
He told his wife that only he trusted her.
He told his wife that he only trusted her.
He told his wife that he trusted only her.
He told his wife that he trusted her only.

2.2.1.3 Recognizing Textual Entailment (RTE)

Finally, feature-based methods have been used in a NLP task called Recognizing Textual Entailment (RTE). The problem is formulated as follows: there are two sentences, the *premise* and *hypothesis*. And the relationship between the two sentences can be described as either ’entailment’, ’contradiction’ or ’neutral’. For example, if the premise is ’A soccer game with multiple males playing.’ and the hypothesis is ’Some men are playing a sport.’, then arguably it can be said that the premise entails the hypothesis. This example is taken from the Stanford Natural Language Inference (SNLI) Corpus (Bowman et al., 2015) which contains 570,000 such examples. Each example is labelled by five human annotators.

Bowman et al. (2015) use six features to train a classifier on the SNLI corpus. These include (i) the BLEU (Bilingual Evaluation Understudy) score (Papineni et al., 2002) of the hypothesis with respect to the premise, (ii) the length difference, in terms of number of words, between the hypothesis and the premise, (iii) the overlap between words in the premise and hypothesis, both in absolute and relative terms, and both over all words and over just certain parts of speech (POS), (iv) an indicator for every unigram and bigram in the hypothesis, (v) an indicator for every pair of words across the premise and hypothesis which share a POS tag, an indicator feature over the two words, and (vi) an indicator for every pair of bigrams across the premise and hypothesis which share a POS tag on the second word.

The feature-based baseline in Bowman et al. (2015) is of significance, because it was introduced together with the SNLI corpus, which has since become a standard data set for training NLP models. This presents a way of directly comparing a feature-based model with ANN-based models. Consider that the model based on the features described above reaches an accuracy of 78.2% on the test set, while state of the art ANNs reach 88.8% (Wang et al., 2017). This concludes the overview of feature-based models in the context of NLP and provides a natural transition to ANN-based models.

2.2.2 ANN-based methods

We have explained in 2.2 that regardless of whether features are manually created or obtained using an ANN, it is necessary to create a **representation** of a string of text, before it can be used in any Machine Learning algorithm. We are now going to motivate the use of ANNs by pointing out the advantages of representations resulting from ANNs relative to feature engineering.

2.2.2.1 Word embeddings

Assume that we want to find distributed representations for a vocabulary consisting of V words. A feature-engineer would look at every word in turn and think about which features best describe it. Consider the first word in the vocabulary was *tree* and assume it is best described by a number of features such as $\{$*'object': True, 'Plant': True,...*$\}$. Now assume the next word in the vocabulary was *health* and again it was possible to describe it by some features like {*'object': False, 'affected-by-lifestyle': True,...*}. In sum, every word would be described by a d-dimensional, potentially sparse, vector consisting of zeros, apart from the indices corresponding to the *n* features which describe it. *d* is the number of features required to uniquely represent each word in the vocabulary. It becomes clear that this exercise would take a long time to complete for large vocabularies, such as for example all words in the English language.

ANN-based approaches do away with this problem. Rather than relying on distributed representations which are manually created by human experts ahead of training, the representations are automatically created by the ANN during training. This is achieved by, rather than being considered fixed inputs, feature vectors are treated as trainable parameters of the model, just like any other parameters. This way, the ANN will adjust the word vectors such that they best explain the observed data.

ANN-based models that yield word embeddings can be categorized as

- **Local Models**: These models are based on local statistics around words. This means that words are considered together with the context in which they appear, *i.e.* a fixed size window of surrounding words. Two such models are the skip-gram and continuous-bag-of-words models introduced by Mikolov et al. (2013a). The former model is trained to predict context word given a certain word, while the latter is trained to predict a word given its context; and

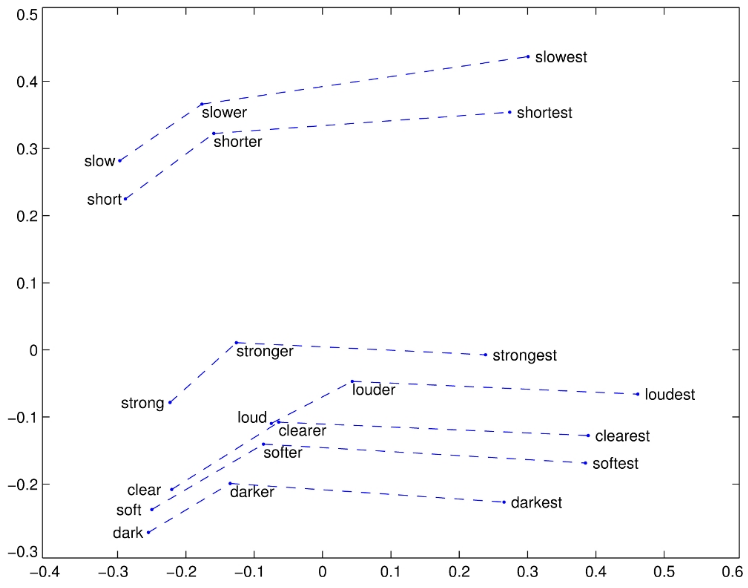
- **Global Models**: These models are trained by exploiting global corpus statistics. The starting point are so-called co-occurrence matrices, which measure how frequently words occur in the context of other words across the entire corpus. The approach is then to use matrix-factorization to find vector representations of words. Early models such as Latent Semantic Analysis (Deerwester et al., 1990) and Latent Dirichlet Allocation (Blei et al., 2003) have shown to perform poorly at preserving linear relationships between words and to be computationally expensive on large data sets (Mikolov et al., 2013a). GloVe (Global Vectors) introduced by Pennington et al. (2014) solves these problems and is discussed in more detail in Section 3.1.

The result of the above models are dense word vectors with typically much fewer dimensions than the number of words in the vocabulary. Moreover, state-of-the-art models such as skip-gram, continuous-bag-of-words and GloVe are trained in such a way that words which carry a similar meaning, *i.e.* words that are used in similar contexts, are close together in vector space. This means that basic operations such as adding or subtracting word vectors leads to meaningful results. A classic example is that when starting with vector(King), adding vector(Man) and subtracting vector(Woman) results in a vector that whose nearest neighbour is that of the word ”Queen” (Mikolov et al., 2013c). These linear relationships between word vectors can also be shown graphically (see Figure 2.4). As we will see in Chapter 3, pre-trained word vectors can be used as inputs in other NLP models.

2.2.2.2 Sentence embeddings

Just like the interest in feature-engineering has declined in favour of distributed representations of words, a similar shift occurred in relation to sentence representations. Early attempts were based on using word representations to construct sentence representations. Mitchell and Lapata (2010) introduce the concept of compositional models, which are defined as a general function f of two word vectors $\mathbf{u}$ and $\mathbf{v}$. The types of functions f considered by the authors give rise to additive and multiplicative models. Blacoe and Lapata (2012) explore different ways of creating sentence representations from word embeddings including compositional methods and a deep recursive autoencoder. The authors find that simply adding word representations, albeit not taking into account word order, outperforms more complex methods on certain tasks. Paperno et al. (2014) find that compositional

Figure 2.4: Visualization of linear relationships between GloVe word vectors



Reproduced from https://nlp.stanford.edu/projects/glove/

models based on distributional semantics are able to outperform simple additive and multiplicative compositional models.

2.2.2.3 Recurrent Neural Networks

At the same time, researchers have tried to use ANN-based models rather than compositional models to obtain sentence representations. For example Socher et al. (2011a) and Socher et al. (2011b) use recursive neural networks based on parse structure. That is, the network is trained using a list of words, and a binary tree representing their syntactic structure as input. Zhao et al. (2015) propose a recursive neural network model based on non-syntactic hierarchical structure called AdaSent. Kalchbrenner et al. (2014), Kim (2014) and He et al. (2015), among others, use convolutional neural networks to encode sentences.

Arguably one of the most popular ANN variants in NLP today are Recurrent Neural Networks (RNNs). The most basic form of an RNN is very similar to ANNs as described in Section 2.1.4. The main differences between ANNs and RNNs are that:

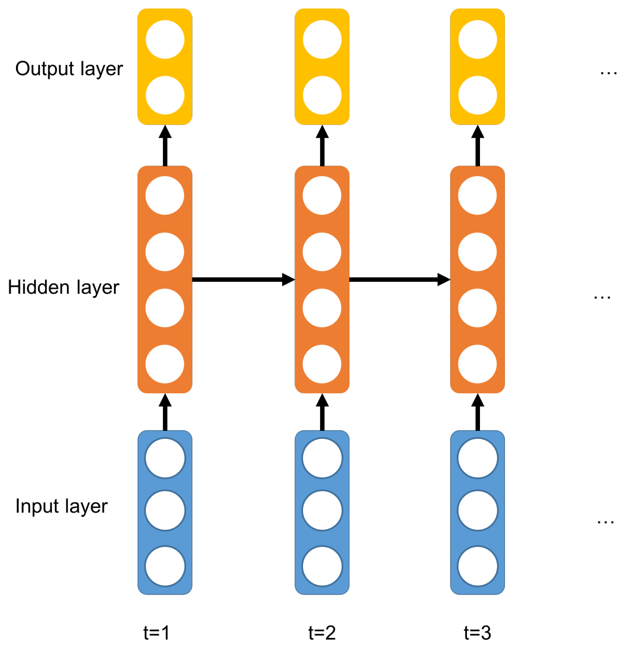
1. RNNs are commonly used when input data is sequential.

2. RNNs consists of a variable number of layers, in which the weights at each layer are identical. The number of layers depends on the length of the sequence.

3. In RNNs, the hidden layers are not only connected to the input layer, but also to the hidden layer of the previous input.

This is illustrated in Figure 2.5 where the arrows between two layers mean that all nodes between the two layers are connected. A simple RNN can then be expressed

Figure 2.5: A simple recurrent neural network (RNN)



mathematically as

$$\text{Hidden layer: } \mathbf{h_t} = a(\mathbf{U}\mathbf{w_t} + \mathbf{V}\mathbf{h_{t-1}}) \tag{2.11}$$

$$\text{Output layer: } \mathbf{y_t} = g(\mathbf{h_t}) \tag{2.12}$$

where $t = 1, ..., T$ indexes the words as they occur in a sequence, $\mathbf{w_t} \in \mathbb{R}^d$ is the word vector input to layer t and $\mathbf{h_t} \in \mathbb{R}^D$ is the hidden state in layer t. The matrices

$\mathbf{U} \in \mathbb{R}^{D \times d}$ and $\mathbf{U} \in \mathbb{R}^{D \times D}$ contain learnable parameters, *i.e.* the weights. $a(.)$ is a nonlinear function such as the sigmoid function. $g(.)$ can be either a non-linear or a linear function such as the identity function. In some applications it is not the output layer, but, for reasons which are explained in Section 2.2.3, the last hidden state that is of interest. The last hidden state is often the input to further fully connected layers, or a classifier, such as the logistic classifier (Section 2.1.1).

The RNN explained here is rarely used in practice. That is because it suffers from a problem called vanishing/exploding gradients which may occur due to repeated matrix multiplications. This is a problem especially for long sequences. Therefore, in this thesis we use RNN variants which partially solve this problem. In particular, Section 3.2 introduces the Gated Recurrent Unit (GRU) (Chung et al., 2014) and Section 3.3 the Long-Short-Term Memory (LSTM) (Hochreiter and Schmidhuber, 1997) models.

Attempts to use recurrent neural networks to encode sentences include Liu et al. (2015) who use an LSTM in which the hidden state is partitioned to model texts of different lengths, and Tai et al. (2015a) who introduce tree-based LSTMs. While these are supervised models, and hence require labelled data sets, there have been attempts to obtain sentence embeddings unsupervisedly. Examples include Le and Mikolov (2014) and Kiros et al. (2015) whose models perform well on a range downstream tasks such as sentiment analysis. Conneau et al. (2017) achieve current state-of-the-art results with a bidirectional LSTM and using pre-trained GloVe word vectors.

2.2.3 Understanding what RNNs learn

The hidden state $\mathbf{h_t}$ is sometimes referred to as the memory of the RNN. Mathematically, it is a non-linear combination of a sequence of word vectors. Intuitively, it can be thought of as a vector representation of the sentence, a type of summary, up until the t-th word. Upon processing the full sequence of T words, the hidden state can therefore be seen as a vector representation of the full sentence. It is this vector representation of sentences which is the main interest of this thesis.

Recent research has been focused on understanding better what information about a sentence is encoded by RNNs (Adi et al., 2016). The authors use sentence embeddings resulting from three models to predict basic sentence features (sentence length, word-content and word-order). The models considered are an LSTM autoencoder, a CBOW base line produced by averaging word2vec word embeddings (Mikolov et al., 2013a), and a CBOW models using Skip-thought sentence embeddings (Kiros et al., 2015). Adi et al. (2016) find that all models outperform the random baseline, with the autoencoder achieving the best performance on most tasks. Furthermore, the LSTM autoencoder outperforms the CBOW baseline in most experiments. This provides some evidence that RNNs encode additional information compared to simple baselines that rely on averaging word embeddings.

Linzen et al. (2016) investigate to what extent RNNs are capable of learning the syntactic structure of sentences. In particular, the authors investigate the performance of LSTMs on a number of prediction tasks. One example is the Number Prediction Task, in which the model has to complete sentences like ”The keys to the cabinet ___”. That is, the model has to make a binary choice between the singular and singular of the following verb. The authors find that LSTMs achieve very low error rates on these, and more difficult tasks, indicating that LSTMs are in principle able to learn syntactic concepts.

The LSTM in Linzen et al. (2016) is trained on a data set containing training examples like the one described above. It is therefore a supervised approach, and the model is evaluated on the same task as it was trained on. The authors also conduct unsupervised experiments. That is, they train an LSTM Language Model with the objective of predicting the next word given any sequence. This means that it is not specifically trained to just make a decision between the singular or plural form of a word, but it has to predict the most likely word in the entire vocabulary. The authors find that in this setting error rates increase, and conclude that RNNs can learn about the syntactic structure of sentences, but some supervision might be required to do so.

This thesis builds on the above papers. In particular, we use the same approach as Adi et al. (2016), *i.e.* we predict sentence features from sentence embeddings. But we extend the analysis to investigate whether RNNs learn about syntactic features as Linzen et al. (2016). We conduct this analysis using state-of-the-art sentence encoders. In particular, we train *universal* sentence encoders, *i.e.* models that are trained on data sets containing sentences from a wide range of different contexts. As we will see, this will lead to sentence encoders that perform well on a range of downstream tasks, which have a different objective as the tasks on which the sentence encoders were trained on.

Chapter 3

Training universal sentence encoders

The discussion of relevant literature in Chapter 2 has shown how over time NLP researchers have moved from using hand-crafted feature vectors and count based methods to distributed representations that are learned by ANNs. These representations have shown to be useful inputs in other NLP models. However, a disadvantage of learned feature vectors is that they are not as easily interpretable as manually created feature vectors. After all, they were created by an algorithm which does not necessarily assign features in the same way as humans would. We concluded the literature review with a new branch of research, which is directed towards understanding what information, *i.e.* which features, of a sentence are encoded by RNNs. This thesis is going to contribute to this area of research. As a first step, this requires obtaining sentence embeddings. In this chapter we describe three models that we use to that end:

- **CBOW**, a continuous-bag-of-words approach using pre-trained GloVe word embeddings (Pennington et al., 2014);
- **Skip-thought** (Kiros et al., 2015), an unsupervised model that we train on the BookCorpus (Zhu et al., 2015); and
- **max-BiLSTM** (Conneau et al., 2017), a supervised model that we train on the SNLI corpus (Bowman et al., 2015).

All three models are trained on large corpora, containing text from a range of different contexts. They are therefore referred to as *universal* sentence encoders. As we will see in Section 3.4, sentence embeddings resulting from universal sentence encoders are useful when transferred to a range of different downstream tasks.

3.1 CBOW

As mentioned in Section 2.2.2.2, one of the earlier attempts to create sentence embeddings are so called compositional methods. These are based on combining word embeddings using relatively simple algebraic operations such as addition or multiplication. As the goal of this thesis is to understand what RNNs learn, we use a compositional method as a baseline for comparison purposes. That is, by exploring the difference between this baseline and an RNN encoder, we will be able to assess the incremental performance due to the RNN. As we will see in Chapter 4, this baseline will perform on par with RNN-based methods on tasks where the sequential nature of sentences does not play a big role. But we will also see that the CBOW baseline will perform worse than RNN-based models where information about the full sentence is important.

Like in Adi et al. (2016), in this thesis the term CBOW (continuous-bag-of-words) means that sentences are treated as a collection of words in no particular order. More precisely, it refers to the method of creating sentence representations by taking the mean of the vector representations of the words that occur in the sentence. It should therefore not be confused with the *training method* CBOW, which is used in the word2vec algorithm (Mikolov et al., 2013a) and which is a way of obtaining word embeddings by predicting a word on the basis of its surrounding words.

The CBOW model in this thesis relies on pre-trained word embeddings called GloVe (Pennington et al., 2014). The main reason for this choice is that Pennington et al. (2014) show that GloVe vectors outperform vectors obtained from other algorithms such as word2vec (Mikolov et al., 2013a). The name GloVe, Global Vectors, derives from the fact that the model takes into account global corpus statistics, and does not only rely on a small window of surrounding words like the word2vec model (Mikolov et al., 2013a). The main difference of GloVe relative to local models such as word2vec is that the latter is essentially a language model. Its objective is to predict the most likely word given its surrounding words, or vice versa. The fact that this yields word embeddings which contain preserve relationships between words is almost a by-product of the algorithm. GloVe vectors on the other hand are trained specifically with that goal in mind. As we will see in Eq. 3.7, the model's objective is to find word embeddings which minimize the difference between the dot-product of two word embeddings and the logarithm of their co-occurrence count. Put differently, if words occur frequently in the same context, then they must have a similar meaning and their word embeddings should therefore be similar.

In what follows we are going to paraphrase the approach of Pennington et al. (2014), using the same notation. Global corpus statistics are captured by a matrix $\mathbf{X}$ which contains word-word co-occurrence counts. That is, $\mathbf{X}_{ij}$ refers to the number of times the word j occurred in the context of word i in the corpus. $\mathbf{X}_i = \sum_k \mathbf{X}_{ik}$ then refers to the total number of times any word occurred in the context of word i. Using this notation, $P_{ij} = P(j|i) = \mathbf{X}_{ij}/\mathbf{X}_i$ is the conditional probability of observing word j in the context of word i.

The starting point of Pennington et al. (2014) is the observation that for any three words w_i, w_j and w_k, the ratio $P_{ik}/P_{jk} > 1$ if w_k occurs more often in the context of w_i than it does in the context of w_j. The authors propose a general model F describing the relationship between any three words in the form of

$$F(\mathbf{w_i}, \mathbf{w_j}, \widetilde{\mathbf{w}}_\mathbf{k}) = \frac{P_{ik}}{P_{jk}} \tag{3.1}$$

where $\mathbf{w_i}, \mathbf{w_j} \in \mathbb{R}^d$ are word vectors and $\widetilde{\mathbf{w}}_\mathbf{k} \in \mathbb{R}^d$ refers to a context word. The authors then constrain the function space of F to include only a subset of linear relationships between word vectors:

$$F((\mathbf{w_i} - \mathbf{w_j})^T \widetilde{\mathbf{w}}_\mathbf{k}) = \frac{P_{ik}}{P_{jk}} \tag{3.2}$$

The distinction between a word and a context word is arbitrary as any word could occur in the context of another word and vice versa. The model should therefore be invariant to exchanging $\mathbf{w} \leftrightarrow \widetilde{\mathbf{w}}$ and $\mathbf{X} \leftrightarrow \mathbf{X}^T$. The authors ensure that this is the case in two steps.

First, they require that F is a homomorphism between the groups $(\mathbb{R}, +)$ and $(\mathbb{R}_{>0}, \times)$, that is

$$F((\mathbf{w_i} - \mathbf{w_j})^T \widetilde{\mathbf{w}}_\mathbf{k}) = \frac{F(\mathbf{w_i}^T \widetilde{\mathbf{w}}_\mathbf{k})}{F(\mathbf{w_j}^T \widetilde{\mathbf{w}}_\mathbf{k})} \tag{3.3}$$

Combining Eqn. (3.2) and Eqn. (3.3) it holds that

$$F(\mathbf{w_i}^T \widetilde{\mathbf{w}}_\mathbf{k}) = P_{ik} = \frac{\mathbf{X}_{ik}}{\mathbf{X}_i} \tag{3.4}$$

And a solution to Eqn. (3.3) is given by $F(.) = \exp(.)$, or

$$\mathbf{w_i}^T \widetilde{\mathbf{w}}_\mathbf{k} = \log P_{ik} = \log \mathbf{X}_{ik} - \log \mathbf{X}_i \tag{3.5}$$

Second, the authors add bias terms for both words $\mathbf{w_i}$ and $\widetilde{\mathbf{w}}_\mathbf{k}$. The term $\log \mathbf{X}_i$ is independent of k and can therefore be fully absorbed by $\mathbf{b_i}$.

$$\mathbf{w_i}^T \widetilde{\mathbf{w}}_\mathbf{k} + \mathbf{b_i} + \widetilde{\mathbf{b}}_\mathbf{k} = \log \mathbf{X}_{ik} \tag{3.6}$$

Eqn. (3.6) is ill-defined when the argument of the logarithm approaches zero. This can be resolved by adding 1 to the argument. Another drawback of this model is that all co-occurrences are weighted equally, whereas it might be desirable to put less weight on less frequent observations as they contain less information and are more noisy. The authors solve this problem by introducing a weighted least squares model

$$J = \sum_{i,j=1}^{V} f\left(\mathbf{X}_{ij}\right)\left(\mathbf{w_i}^T\widetilde{\mathbf{w}}_\mathbf{j} + \mathbf{b_i} + \widetilde{\mathbf{b}}_\mathbf{j} - \log \mathbf{X}_{ij}\right)^2 \tag{3.7}$$

where $f\left(\mathbf{X}_{ij}\right)$ is a weighting function defined as

$$f(x) = \begin{cases} (x/x_{\max})^\alpha & \text{if } x < x_{max} \\ 1 & \text{otherwise} \end{cases} \tag{3.8}$$

which was found to perform well with $\alpha = 3/4$. Pennington et al. (2014) train this model on different corpora and with various word vector dimensions. The resulting word vectors are publicly available.[1] For the purpose of this thesis, we use 300-dimensional word vectors that were trained on 840 billion tokens of online text from Common Crawl[2] are used.

As mentioned in Section 2.2.2.2, there are many compositional functions that could be used to obtain sentence embeddings. Simply averaging is one possibility which has been used in other contexts - for example by Wieting et al. (2015), Hill et al. (2016) and Adi et al. (2016). We therefore follow this approach too and compute sentence embeddings as the average of word embeddings:

$$\mathbf{s} = \frac{1}{T}\sum_{t=1}^{T} \mathbf{w_t} \tag{3.9}$$

3.2 Skip-thought

One flaw of the CBOW model discussed in the previous section is that it does not take into account the sequential nature of sentences. Instead, it just takes the average of word vectors that occur in the sentence. As shown in Table 2.2, word order has a substantial impact on the meaning of a sentence. And as discussed in Section 2.2.2.3, RNNs were designed to deal with sequential data. Our hypothesis is therefore that RNNs should in principle be able to create better word embeddings

[1] https://nlp.stanford.edu/projects/glove/
[2] http://commoncrawl.org

when evaluated on tasks in which the order of words matters. This is indeed one of our main findings in Chapter 4.

In this section we are going to introduce a state-of-the-art sentence encoder: the Skip-thought model. The Skip-thought model constitutes an unsupervised approach to training a sentence encoder. Unsupervised models have the advantage of not requiring labelled training data, which might be costly to obtain. However, the disadvantage is that they are often more difficult to train, and require a lot more data than unsupervised models.

The model architecture of the Skip-thought model is inspired by the skip-gram model (Mikolov et al., 2013a), which aims to predict the context of a word. The skip-thought model takes this idea to the sentence level and tries to predict the previous and following sentences of any given sentence.

3.2.1 Model architecture

In the original paper, Kiros et al. (2015) use an RNN encoder with Gated Recurrent Units (GRU) (Chung et al., 2014), and two RNN decoders with GRU cells that are conditioned on the input sentence. We explain how GRU cells work in Eq. 3.10-3.13. The conditioning is achieved as follows. First the encoder is used to obtain a sentence representation, given by the last hidden state of the encoder. And second, a linear transformation of the sentence representation is added to the GRU activations of the decoders at each time step.

In this thesis, the approach with respect to conditioning the decoders on the encoder output is slightly different. This is because there is no *a priori* reason why the sentence representation of the encoder should be included in the decoder activations at every time step. Instead, we set the hidden state of the decoders equal to the final hidden state of the encoder. This is computationally slightly more efficient and leads to comparable results. It has also become a standard way of conditioning decoders in encoder-decoder models and has been used in alternative implementations[3]. In order to to rule out any ambiguities, we describe the encoder and decoders in detail below.

Let a training instance be a 3-tuple of ordered sentences $\{s_1, s_2, s_3\}$ as they occur in the corpus. s_2 will the the input to the encoder, and s_1 and s_3 the inputs to the two decoders. Let $\mathbf{E} \in \mathbb{R}^{vocab\_size \times d}$ denote a matrix of learnable parameters which are shared by the encoder and decoders, and which contain the d-dimensional word vectors for every token in the vocabulary.

Encoder: Given a sequence of T words $\{w_1, ..., w_T\}$ constituting sentence s_2, let $\{\mathbf{w_1}, ..., \mathbf{w_T}\}$ be the corresponding sequence of embedded words, where each $\mathbf{w_t} \in \mathbb{R}^d$ is a d-dimensional vector. At any time step t, the encoder takes as input $\mathbf{w_t}$ and performs the set of matrix operations shown in Eq. 3.10 - Eq. 3.13[4]. This has the effect of updating the hidden cell state $\mathbf{h_t} \in \mathbb{R}^D$ where D is the number of units of the encoder cell. The hidden state $\mathbf{h_t}$ can be thought of as a summary of the sequence of words observed until time t. Upon processing the last input $\mathbf{w_T}$, $\mathbf{h_T}$ is then a representation of the full sentence. Both $\mathbf{h_T}$ and $\mathbf{s_2}$ refer to the encoded sentence s_2 and are used interchangeably.

The following set of equations constitute a GRU cell and show how at any time point t a non-linear combination of the current word $\mathbf{w_t}$ and the previous hidden state $\mathbf{h_{t-1}}$ is used to update the hidden state $\mathbf{h_t}$. The matrices $\mathbf{U_x} \in \mathbb{R}^{D \times d}$ and $\mathbf{V_x} \in \mathbb{R}^{D \times D}$ where $\mathbf{x} \in \{z, r, h\}$ according to the gate in which they are used.

$$\text{Update gate: } \mathbf{z_t} = \sigma(\mathbf{U_z w_t} + \mathbf{V_z h_{t-1}}) \tag{3.10}$$

$$\text{Reset gate: } \mathbf{r_t} = \sigma(\mathbf{U_r w_t} + \mathbf{V_r h_{t-1}}) \tag{3.11}$$

[3]https://github.com/tensorflow/models/tree/master/skip_thoughts

[4]All vectors and matrices belonging to the encoder should carry the superscript *enc* to distinguish them from the parameters used in the decoders *pre* and *post*. This is not shown to make the notation lighter.

$$\text{Hidden state candidate: } \mathbf{h_t^*} = tanh(\mathbf{U_h w_t} + \mathbf{V_h}(\mathbf{r_t} \circ \mathbf{h_{t-1}})) \tag{3.12}$$

$$\text{Hidden state update: } \mathbf{h_t} = (\mathbf{1} - \mathbf{z_t}) \circ \mathbf{h_{t-1}} + \mathbf{z_t} \circ \mathbf{h_t^*} \tag{3.13}$$

Decoders: In this thesis, the decoder GRU cells have the same structure as the encoder cells described above. Each decoder, which we refer to as *pre*coder and *post*coder, has its own set of trainable parameters **U** and **V**. At the first time step, the hidden states of the decoders are set equal to $\mathbf{s_2}$. This way, the decoders are conditioned on the vector representation of the middle sentence of the sequence. At each time step, the decoders receive the ground truth as input, *e.g.* the *pre*coder receives the word vector of the t-th word in s_1, and output a probability distribution over the next word in the sequence $P(w_{t+1}|s_2, w_1, ..., w_t)$. This is achieved by projecting the hidden state of the decoder onto the vocabulary using a matrix of trainable parameters $\mathbf{P} \in \mathbb{R}^{vocab\_size \times D}$ and normalizing the resulting vector such that it represents a valid probability distribution over all words in the vocabulary. **P** is shared between both decoders.

In addition, we apply layer-normalization to both encoder and the decoders, as this has shown to improve the training time and downstream performance of the skip-thought model, and RNNs in general (Ba et al., 2016). Layer normalization amounts to normalizing the inputs of the non-linearities, *i.e.* the sigmoid- and tanh-functions, at each layer, *i.e.* at each time step t. For example, let $\mathbf{a_t} = \mathbf{U^z w_t} + \mathbf{V^z h_{t-1}}$ be the input of the sigmoid function of the update gate (Eq. 3.10). That is, $\mathbf{z_t} = \sigma(\mathbf{a_t})$. Layer normalization instead uses the input

$$\mathbf{a_t'} = \frac{\mathbf{g}}{\sigma_t} \circ (\mathbf{a_t} - \mu_t) + \mathbf{b} \tag{3.14}$$

where **b** and **g** are bias and gain parameters of the same dimension as $\mathbf{h_t}$, μ_t is the mean of the H hidden units in a layer

$$\mu_t = \frac{1}{H}\sum_{i=1}^{H} a_t^i \tag{3.15}$$

and σ_t is the standard deviation across the H hidden units in a layer

$$\sigma_t = \sqrt{\frac{1}{H}\sum_{i=1}^{H}(a_t^i - \mu_t)^2} \tag{3.16}$$

3.2.2 Corpus

The goal of the Skip-thought model is to obtain universal sentence embeddings. That is, sentence embeddings which perform well on a wide range of downstream tasks, not necessarily related to the domain on which the model was trained on. Therefore it is important to train it on a data set that demonstrates as much variation as possible. If the model was trained on a very specific data set, then it would not be able to generalize well to other contexts. With permission of the authors, we therefore use same corpus as in the original skip-thought paper (Kiros et al., 2015), namely the BookCorpus (Zhu et al., 2015). This is a large collection of novels written by unpublished authors as per 2015. It could be argued that this data set is not well suited for training a universal sentence encoder, because it only includes one specific type of naturally occurring text: Books. On the other hand, the data set is relatively comprehensive, as it contains books from 16 different genres. The most common genres are: *Romance* (2,865 books), *Fantasy* (1,479), *Science fiction* (786) and *Teen* (430). As we will see in Section 3.4, the model indeed performs well on a wide range of different downstream tasks.

We show deescriptive statistics of the BookCorpus in Table 3.1. While the mean sentence length is 13, there are a few outliers. Therefore, as in Kiros et al. (2015), we exclude sentences that are longer than 30 tokens for computational reasons.

Table 3.1: BookCorpus descriptive statistics

Reproduced from Kiros et al. (2015)

# books	# sentences	# words	# unique words	mean # words per sentence
11,038	74,004,228	984,846,357	1,316,420	13

As in Kiros et al. (2015), We construct a vocabulary consisting of the 20,000 most frequent words in the corpus. We replace words that are not in the vocabulary are with an out-of-vocabulary token. The reason for this is that at every time step, the two decoders perform a projection of the hidden state onto the vocabulary. This would involve multiplying the hidden state with a weight matrix of size $[D, vocab\_size]$ which in this case is computationally expensive as $vocab\_size$ is greater than 1.3m. While using a restricted vocabulary makes training the model a lot faster, it potentially reduces its performance on downstream tasks, as it may encounter many unknown words. For that reason we perform a vocabulary expansion performed after training (see Section 3.2.5).

3.2.3 Hyperparameters

In this thesis, we use largely the same hyperparameters as Kiros et al. (2015). That is, we use a single layer GRU model with hidden size $D = 2400$, and word embeddings of dimension $d = 620$. The word embeddings **E** are initialised using a random normal distribution. In the GRU cells, the matrices **V** are initialised using an orthonormal initializer (Saxe et al., 2013). Matrices **U**, and the projection matrix from the hidden size of the decoder onto the vocabulary **P** are initialised uniformly on the interval $[-0.1, 0.1]$. Gradients are clipped if their norms exceed 5. As loss function we use the mean cross-entropy (Section 2.1.2) over all training examples in the minibatch.

The model is trained using the Adam optimizer (Kingma and Ba, 2014) with learning rate 0.0008 and a learning rate decay of 0.5 after 400,000 batches. The batch size is 128. These hyperparameters correspond to the ones used in a different implementation of the Skip-thought model.[5] The model architecture is illustrated

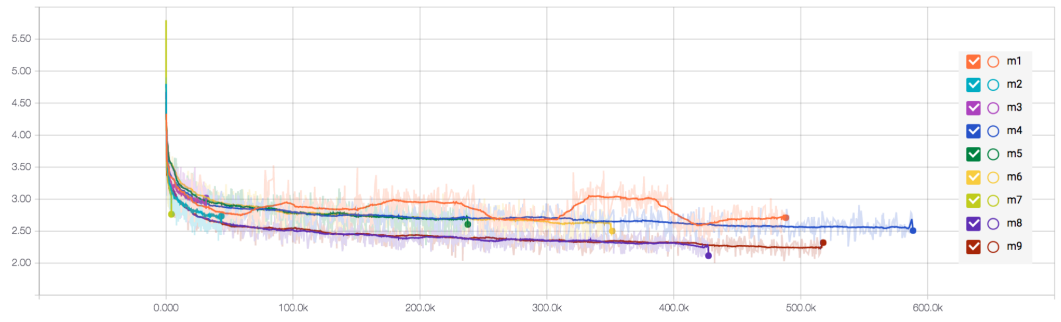
[5]https://github.com/tensorflow/models/tree/master/skip_thoughts

in Appendix A.

3.2.4 Training

We train a number of Skip-thought models with different hyperparameter settings. That is, learning rates of 0.008 or 0.0008, gradient clipping at norm 5 or 10, dropout of 0.2 or no dropout, and learning rate decay of 0.5 or 0.2. The highest performing model is the one explained in the previous section, *i.e.* model m1 in Figure 3.1. This is somewhat surprising as it is not the one that was trained the longest, it did not reach the lowest loss, and its loss did not decline as smoothly as for the other models. The explanation for the last observation is possibly that the training data was shuffled in terms of books, not in terms of training instances. Nevertheless, this model reached the highest score on the downstream evaluation tasks discussed in Section 3.4.

Figure 3.1: Skip-thought training curves



During training, every 1,000 batches the model generates the pre- and proceeding sentences given an input sentence. That is, the encoder creates a sentence representation, from which the decoders decode sample sentences. Unlike during training, when at each time step the decoders receive the ground truth word as input, here the decoders take as input the previously sampled word and continue until the END token is generated. See Table 3.2 for an example of this procedure after around 200,000 training steps.

Table 3.2: Comparison between original sentences, and sentences generated of a Skip-thought model after 200,000 training steps

Original sequence:
were you ever coming back inside ? ”
“ in a while . ”
he extended his arm and she crawled in close beside him .

Predicted sequence:
“ i hired killers , are n’t you ? ”
“ in a while . ”
he moved closer , and the heaviness in her voice vanished .

3.2.5 Vocabulary Expansion

Due to computational reasons, we restrict the vocabulary to include only the 20,000 most frequent words during training (See Section 3.2.2). While this reduces the time it takes to train the model, it potentially also decreases the model’s performance on downstream tasks, because it might encounter many unknown words and hence fail to encode sentences properly. To alleviate this problem, a vocabulary expansion is performed.

The idea is based on Mikolov et al. (2013b), who use vocabulary expansion to translate missing words by learning a linear mapping between the vector spaces of two languages. Similarly, in this thesis a linear mapping is learned between the word vectors of the Skip-thought model and the word vectors from a much larger vocabulary. As in Kiros et al. (2015), we use open source word2vec[6] word vectors for this purpose. As a first step this involves fitting a linear regression model between common words in the Skip-thought and the word2vec vocabulary. That is, the parameters $\mathbf{W} \in \mathbb{R}^{300\times 620}$ are found that minimize

$$||\mathbf{X} - \mathbf{Y}\mathbf{W}||^2 \tag{3.17}$$

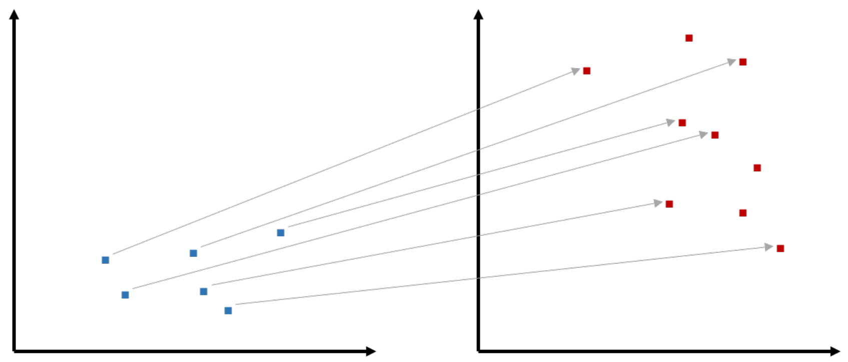
where $\mathbf{X} \in \mathbb{R}^{num\_words\times 620}$ are the 620-dimensional Skip-thought word vectors, $\mathbf{Y} \in \mathbb{R}^{num\_words\times 300}$ are the 300-dimensional word2vec word vectors and *num_words*

[6]https://code.google.com/archive/p/word2vec/

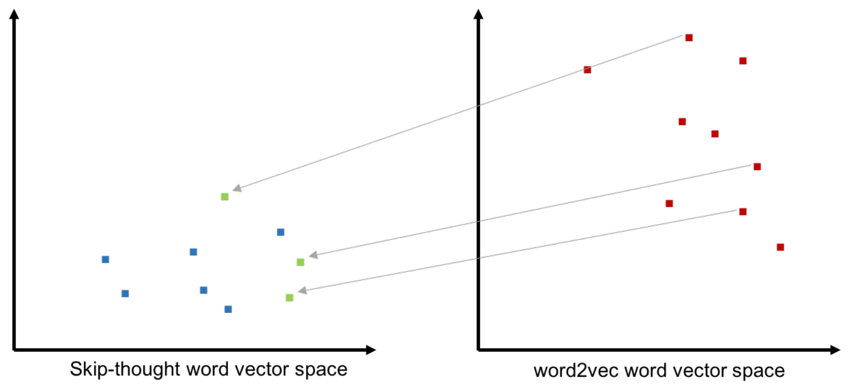
refers to the number of common words in both vocabularies. Using the estimated parameters **W** it is then possible to project missing word vectors from the word2vec space to the Skip-thought vector space and hence increase the size of the vocabulary from 20,000 to 930,911 words. This procedure is illustrated in Figure 3.2.

Figure 3.2: Vocabulary expansion

(a) Finding a mapping between skip-thought and word2vec word vector spaces, based on common words



(b) Using mapping to project missing word vectors from word2vec to skip-thought word vector space



3.3 BiLSTM-max

3.3.1 Model architecture

The BiLSTM-max model (Conneau et al., 2017) is a semi-supervised approach to training a sentence encoder, as it is used on an annotated data set, but uses word vectors as inputs that were obtained in an unsupervised manner. The BiLSTM-max model is trained on a task known as recognising textual entailment (RTE). In this task, the model takes as input a pair of sentences referred to as *premise* and *hypothesis*. The model then makes a prediction of whether the two sentences are (i) contradicting each other, (ii) not related, or whether (iii) the premise entails the hypothesis.

The sentence encoder of the BiLSTM-max model is based on a Long Short Term Memory (LSTM) cell (Hochreiter and Schmidhuber, 1997). The LSTM predates the GRU cell and is slightly more complex in that it has three gates and a cell state, in addition to the hidden state. Both GRU and LSTM perform similarly well on various tasks as shown by Chung et al. (2014) and Greff et al. (2015). Like the GRU cell, the LSTM takes as input a sequence $\{\mathbf{w_1}, ..., \mathbf{x_T}\}$ of word embeddings and performs the matrix operations shown below. That is, it updates its hidden state $\mathbf{h_t}$, which is referred to as the representation of the sequence at time step t. Let $\overrightarrow{\mathbf{h}} = [\overrightarrow{\mathbf{h_1}}, ..., \overrightarrow{\mathbf{h_T}}]$ denote the collection of hidden states produced by an LSTM after having processed a sentence of length T in correct order, *i.e.*, from left to right.

$$\text{Forget gate:} \mathbf{f_t} = \sigma(\mathbf{U^f w_t} + \mathbf{V^f h_{t-1}} + \mathbf{b^f}) \tag{3.18}$$

$$\text{Input gate: } \mathbf{i_t} = \sigma(\mathbf{U^i w_t} + \mathbf{V^i h_{t-1}} + \mathbf{b^i}) \tag{3.19}$$

$$\text{Output gate: } \mathbf{o_t} = \sigma(\mathbf{U^o w_t} + \mathbf{V^o h_{t-1}} + \mathbf{b^o}) \tag{3.20}$$

$$\text{Hidden state candidate: } \mathbf{j_t} = tanh(\mathbf{U^j w_t} + \mathbf{V^j h_{t-1}}) \tag{3.21}$$

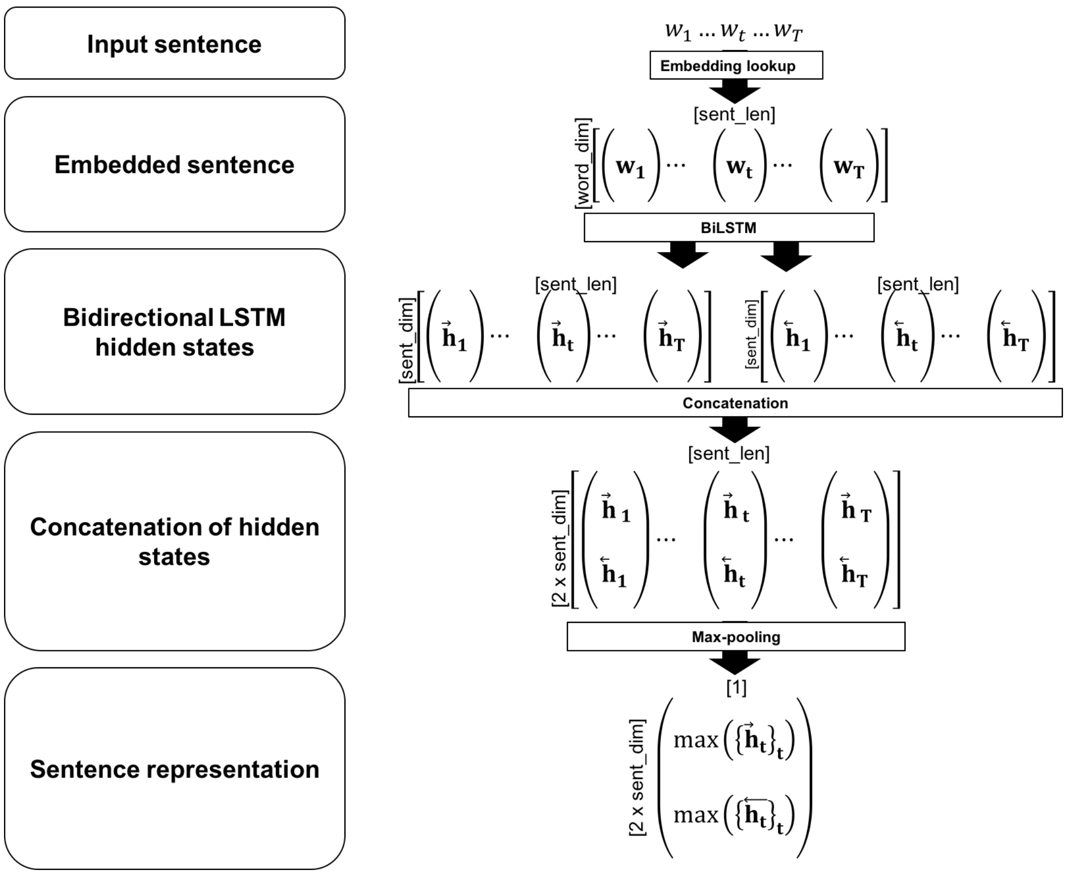
$$\text{Cell state update: } \mathbf{c_t} = \mathbf{f_t} \circ \mathbf{c_{t-1}} + \mathbf{i_t} \circ \mathbf{j_t} \tag{3.22}$$

$$\text{Hidden state update: } \mathbf{h_t} = \mathbf{o_t} \circ tanh(\mathbf{c_t}) \tag{3.23}$$

In the BiLSTM-max model, an extention of the LSTM, a bidirectional LSTM (Schuster and Paliwal, 1997), is used. A bidirectional LSTM effectively consists of two LSTMs; one which takes as input a sequence in correct order, $\mathbf{w_1}, ..., \mathbf{w_T}$, and one that takes as input the same sequence in reverse order $\mathbf{w_T}, ..., \mathbf{w_1}$. The two LSTMs hence produce hidden state representations $\overrightarrow{\mathbf{h}} = [\overrightarrow{\mathbf{h_1}}, ..., \overrightarrow{\mathbf{h_T}}]$ and $\overleftarrow{\mathbf{h}} = [\overleftarrow{\mathbf{h_1}}, ..., \overleftarrow{\mathbf{h_T}}]$, respectively. Let $\mathbf{h_t} = [\overrightarrow{\mathbf{h_t}}, \overleftarrow{\mathbf{h_t}}]$ denote the concatenation the hidden states of the two LSTMs at any time step t.

Depending on the number of tokens in a sentence, $\{\mathbf{h_t}\}_t$ will be of varying length. Like in Conneau et al. (2017), a fixed length representation of sentence is obtained by taking the maximum value over each dimension of the hidden units. This is referred to as max pooling. An illustration of the full procedure is shown in Figure 3.3.

Figure 3.3: Procedural steps of creating a sentence embedding using BiLSTM-max



The above steps are carried out on both the premise and the hypothesis, resulting in corresponding sentence representations denoted as $\mathbf{u}$ and $\mathbf{v}$. For the final classification, we follow the approach of Conneau et al. (2017) and create a feature vector $[\mathbf{u}, \mathbf{v}, \mathbf{u} \cdot \mathbf{v}, |\mathbf{u} - \mathbf{v}|]$, *i.e.* the concatenation of u and v, their element-wise product and their absolute difference. With this feature vector as input, a single-layer

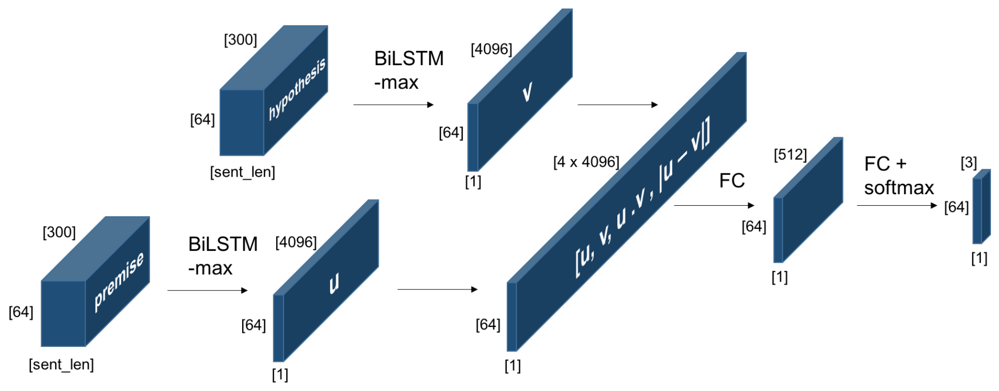
ANN (see Section 2.1.4) whose output is fed into a softmax layer (see Section 2.1.1) is trained by minimising the cross entropy (see Section 2.1.2) between the softmax output and the one-hot representation of the corresponding label.

3.3.2 Hyperparameters

For the purpose of this thesis, we use the same hyperparameter settings as in Conneau et al. (2017). That is, the hidden state of the LSTMs consist of 2048 units such that the resulting sentence representations are 4096-dimensional (2048 for each direction). For the word-embeddings, open source GloVe vectors trained on Common Crawl[7] 840B with 300 dimensions[8] are used, and held fixed throughout training. The ANN in the classification layer consists of a single hidden layer with 512 units. The model architecture resulting from this choice of hyperparameters is illustrated in Figure 3.4.

Like Conneau et al. (2017), we train the model using stochastic gradient descent (SGD) with a batch size of 64 and an initial learning rate of 0.1. Learning rate decay is applied as follows: after every epoch, the learning rate either decreases by a factor of 0.99 if the accuracy on the dev set did not improve, or by a factor of 0.2 if it improved.

Figure 3.4: BiLSTM-max model architecture



[7]http://commoncrawl.org
[8]https://nlp.stanford.edu/projects/glove/

3.3.3 Corpus

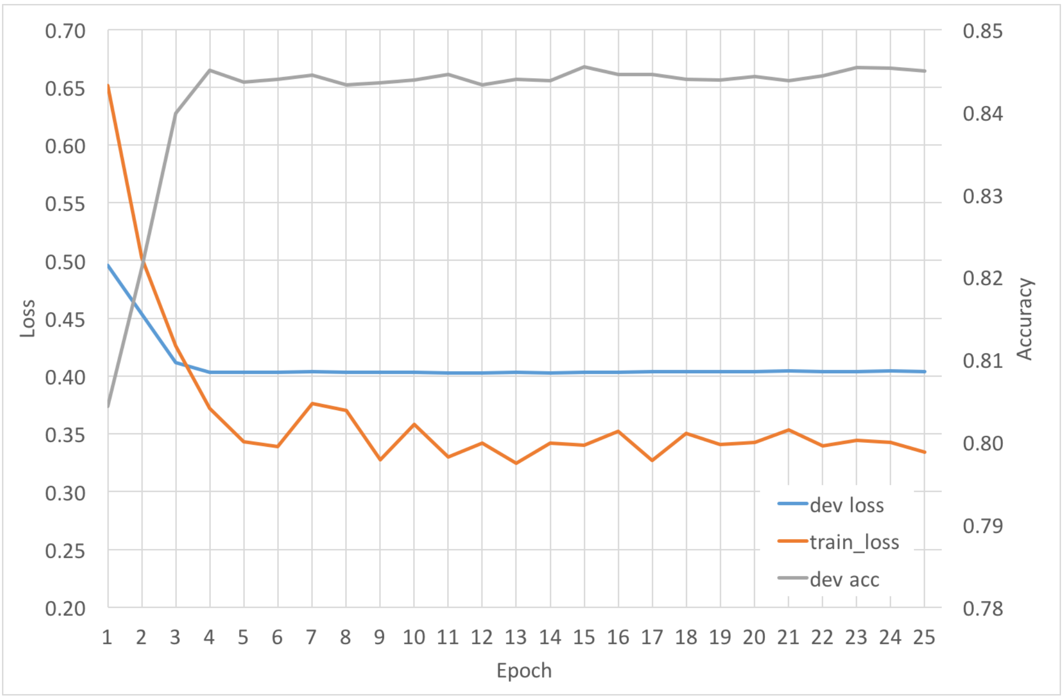
We train the model on the SNLI corpus (Bowman et al., 2015), which is a standard corpus used for RTE tasks. As explained in Section 2.2.1.3, this is a collection of 570,000 human-written English sentence pairs which are human-annotated with the labels $contradiction$, $neutral$ and $entailment$. As mentioned in the beginning of this chapter, our goal is to train *universal* sentence encoders. That is, sentence encoders which are trained on a large and varied corpus, such that they can generalize to other contexts and perform well when evaluated on transfer tasks. We note that the SNLI corpus fulfills this requirement. The premises of the corpus are largely based on image captions retrieved from the Flickr30k corpus (Young et al., 2014). This corpus consists of image captions describing photos of everyday activities, scenes and events and is therefore likely to provide the necessary breadth of contexts.

The data set comes split in a train, dev and test set with 550,000, 10,000 and 10,000 examples each. As in Conneau et al. (2017), words that are not found in the pre-trained GloVe embeddings are simply not taken into account. This is the case for fewer than 1% of words in the corpus.

3.3.4 Training

We record the loss on the training set after each batch. Figure 3.5 shows the average training loss over all batches at the end of the epoch (*train_loss*). It also shows the average loss on the dev set (*dev_loss*) as well as the average accuracy on the dev set (*dev_acc*), where accuracy is defined as the percentage of instances classified correctly, after each epoch. The model is trained for 25 epochs and the best model is selected on the basis of the highest dev accuracy, which occurred at epoch 15.

Figure 3.5: BiLSTM-max training curve



3.4 Evaluation

The process of monitoring the loss or accuracy throughout training is sometimes referred to as *intrinsic* evaluation. That is, the model's performance is evaluated on a sub-task, which is related to the real task of interest, and which is computationally cheaper to assess than the performance on the real task. In the case of the skip-thought model, this sub-task was to to generate the surrounding sentences of any given sentence. And in the case of the BiLSTM-max model, the sub-task is to predict whether the relationship between any pair of sentences is best described as *neutral*, *contradiction* or *entailment*.

But in both the skip-thought and the BiLSTM-max models, the main goal is to train a sentence encoder which is able to generate universal sentence representations. In other words, an encoder which is able to capture the semantics of a sentence in a wide range of different contexts, rather than just to perform well on a particular sub-task. In the case of the skip-thought model, the encoder is trained to capture the meaning of a sentence as closely as possible. This is because the better the encoder does at this task, arguably the better the decoders can perform their task, which subsequently decreases the training loss. And similarly, the encoder in the BiLSTM-max model is trained to capture the meaning of a sentence as well as possible, because the better it performs this task, the easier it becomes for the final classifier to judge what is the relationship between two sentences.

3.4.1 Evaluation tasks

There is a set of common *extrinsic* evaluation tasks, which have been used in the context of sentence encoders such as Kiros et al. (2015), Hill et al. (2016) and Conneau et al. (2017). These evaluation tasks allow to test to what extent a sentence encoder is able to produce universal sentence representations. That is, sentence representations that are useful on downstream tasks that are different to the one used to train the model. We give a brief overview of the data sets involved in the evaluation tasks:

Movie review sentiment (MR) (Pang and Lee, 2005). This is a data set consisting of sentences coming from either positive or negative movie reviews. There are 5341 positive and 5337 negative instances in the data set. An example for a positive instance is ”Guaranteed to move anyone who ever shook, rattled, or rolled.”. An example of a negative instance is ”While the performances are often engaging, this loose collection of largely improvised numbers would probably have worked better as a one-hour tv documentary.”

Product reviews (CR) (Hu and Liu, 2004). This is a dataset consisting of sentences coming from positive and negative product reviews. There are 2406 positive and 1366 negative instances. An example of a positive instance is ”Update: The finish is more ”mirror” than silver – and I like it!”. An example of a negative instance is ”I’ve tried leaving questions on their site never to have them answered”

Subjectivity classification (SUBJ) (Pang and Lee, 2004). This data set consists of sentences coming from either subjective or objective movie reviews. There are 5,015 subjective and 5,000 objective instances. An example of a subjective instance is ”Plotless collection of moronic stunts is by far the word movie of the year.”. And an example of an objective instance is ”Ronnie must then deal with his new life, while having to deal with missing his ex-wife that he left back home.”

Opinion polarity (MPQA) (Wiebe et al., 2005). A data set consisting of phrases from newspaper articles annotated for positive and negative sentiment. There are 3311 positive and 7292 negative instances. An example of a positive instance is ”are also being encouraged”. An example of a negative instance is ”We look at a bleak future”.

Question type classification (TREC) (Voorhees, 2002). This is a data set consisting of 6,000 questions that can belong to one of six different classes. The classes are abbreviation, entitiy, description, human, location and numeric. An example of a question belong to the class description is ”How can I find a list of celebrities’ real names?”

Paraphrase identification (MSRP) (Dolan et al., 2004). This is a data set consisting of 4,076 sentences pairs and labels according to whether they are paraphrases of one another or not.
An example of a positive instance is the sentence pair [”Amrozi accused his brother, whom he called ”the witness’, of deliberately distorting his evidence.”, ”Referring to him as only ”the witness”, Amrozi accused his brother of deliberately distorting his evidence.”].
An example of negative instance is [”Yucaopa owned Dominick’s before selling the chain to Safeway in 1998 for $ 2.5 billion”, ”Yucaipa bought Dominick’s in 1995 for $ 693 million and sold it to Safeway for $ 1.8 billion in 1998.”]

Semantic relatedness and entailment (SICK-R, SICK-E) (Tai et al., 2015b). This is a data set consisting of 10,000 sentence pairs and two types of human-annotated labels. The first label is a score on a scale from 1-5 measuring related the two sentences are. The second label is a standard RTE label, like in the SNLI data set. That is, it takes the values 'contradiction', 'entailment' or 'neutral'. An example of a training instance are the two sentences "A few men in a competition are running outside" and "A few men in a competition are running indoors" which come with the a relatedness score of 3.9 and the RTE label 'contradiction'.

Semantic Textual Similarity (STS14) (Agirre et al., 2014). This is also a data set used for recognizing semantic relatedness between two sentences. It consists of 4,500 sentence pairs and with human annotated similarity scores between 0-5. One example is the sentence pair Liquid ammonia leak kills 15 in Shanghai and Liquid ammonia leak kills at least 15 in Shanghai which comes with a label of 4.6.

3.4.2 The SentEval suite

The above downstream tasks are implemented in the SentEval suite[9] by Conneau et al. (2017), which we use for the purpose of this thesis. The binary and multiclass classification tasks (MR, CR, SUBJ, MPQA and TREC) are implemented as a logistic classifier (as in Section 2.1.1) on top of the sentence embeddings generated by the models described in this chapter. The SICK-E task uses a logistic classifier on top of the concatenation of the two sentence embeddings. The SICK-R and STS14 tasks follow the approach by Tai et al. (2015b) and estimate a probability distribution over relatedness scores based a joint representation of both sentences. The SentEval suite uses L2-regularization which is fine-tuned using grid-search on the dev set.

The SentEval suite reports the following metrics:

[9]https://github.com/facebookresearch/SentEval

- Binary and multi-class classification tasks: Accuracy, *i.e.* the share of correctly predicted classes;
- MRPC: Accuracy and F1 score (explained in more detail in Section 4.1);
- SICK-R: Pearson correlation coefficient (Pearson, 1895);
- SICK-E: Accuracy; and
- STS14: Pearson and Spearman correlation coefficients (Spearman, 1904).

3.4.3 Results

Table 3.3 shows the results of evaluating the models trained in this thesis using the SentEval suite. We also include the comparable models from the original papers, and untrained versions of our Skip-thought and BiLSTM-max models. Untrained in this context means that all parameters of the model are initialized as described as in Sections 3.2.3 and 3.3.2. The exception are the word embeddings which the models take as input, and which are either the Skip-thought word embeddings resulting from the trained model, or the pre-trained GloVe embeddings, respectively. The untrained baseline hence gives some indication of the incremental performance of the RNN cells.

The key observations are:

1. Both the Skip-thought and the BiLSTM-max models outperform the CBOW model. This is in line with Kiros et al. (2015) and Conneau et al. (2017) and means that the sequential information of sentences which is important for downstream tasks such as sentiment analysis, is to some extent captured by the RNN-based sentence encoders;
2. The above is further evidence by comparing the trained Skip-thought and BiLSTM-max variants with their untrained counterparts;
3. The drop in performance of the trained relative to the untrained version is more severe for the Skip-thought model. This could be partly due to the fact

that Skip-thought word vectors are just a by-product of training the Skip-thought model, whereas the GloVe come from a model that was designed to lead to good word embeddings.

4. Our Skip-thought model slightly outperforms the version in the original paper. A potential explanation for this is that we use layer normalization, which has been shown to lead to better performing models (Ba et al., 2016). Our version of the BiLSTM-max model performs worse than the version in the original paper, with the exception of the SICK-R and SICK-E tasks.

Table 3.3: Downstream evaluation tasks

Model	MR	CR	SUBJ	MPQA	SST	TREC	MRPC	SICK-R	SICK-E	STS14
Models trained in this thesis										
CBOW	74.3	77.3	89.8	84.2	78.8	81.4	71.8/80.2	79.6	78.8	0.46/0.47
Skip-thought	78.9	83.3	93.5	87.8	84.2	90.0	73.6/81.5	84.3	81.43	0.35/0.35
BiLSTM-max	75.3	81.1	90.8	85.2	79.5	86.2	73.9/82.0	88.4	86.4	0.57/0.60
Comparable models in original papers										
Skip-thought†	75.5	79.3	92.1	86.9	-	91.4	73.0/81.9	84.8	-	-
BiLSTM-max‡	79.9	84.6	92.1	89.8	83.3	88.7	75.1/82.3	88.5	86.3	0.66/0.64
Other baselines										
Skip-thought (untrained)	65.75	69.03	77.21	84.33	67.5	79.4	69.86/80.29	63.9	68.9	0.34/0.35
BiLSTM-max (untrained)	74.04	77.4	90.12	84.64	79.8	85.8	72.6/81.4	85.2	82.0	0.47/0.48

†uni-skip model in Kiros et al. (2015)

‡BiLSTM-Max (on SNLI) in Conneau et al. (2017)

Chapter 4

Inferring sentence features from sentence embeddings

Recent research by Adi et al. (2016) aims at understanding what information about sentences is encoded by RNNs. In their paper, the authors try to infer basic features of a sentence from its sentence embedding: (i) the length of a sentence, (ii) which words occur in a sentence, and (iii) in which order words occur. The sentence length task is formulated as a multiclass classification with 8 classes corresponding to binned sentence lengths. The word-content task is formulated as a binary classification task to determine whether or not a word is present in a sentence. And the word-order task is also formulated as a binary classification task to determine which word in a given word pair occurs first in a given sentence. The authors find that an RNN-based sentence encoder, an LSTM autoencoder, outperforms a non-RNN baseline, a CBOW model similar to the one described in Section 3.1, in most settings.

As mentioned in Section 2.2.3, Linzen et al. (2016) find that LSTMs are in principle capable of learning the syntactic structure of sentences. The authors show that in a supervised setting, *i.e.* where the model has the explicit objective to learn about syntactic features, this works with higher accuracy than in an unsupervised setting.

In this chapter we are going to build on the above mentioned papers. We are going to use the sentence encoders that we trained as described in Chapter 3 and use a similar methodology as Adi et al. (2016). That is, we are going to infer sentence features from the sentence embeddings obtained by the models we trained. But we are going to extend the analysis by inferring syntactic features as in Linzen et al. (2016). The approach taken in this thesis is novel in two ways. First, we introduce a new task: to predict dependency tags from sentence embeddings. And second, we use state-of-the-art sentence encoders.

4.1 Experimental set up

We carry out three main experiments in this chapter: predicting (i) sentence lengths, (ii) word content and (iii) dependency tags. Each of the experiments is conducted using the models described in Chapter 3. That is, we conduct each experiment using sentence embeddings from the CBOW, Skip-thought and BiLSTM-max model. This results in sentence embeddings $\mathbf{s} \in \mathbb{R}^D$, where $D \in \{300, 2400, 4096\}$ for CBOW, skip-thought and BiLSTM-max, respectively. In some of the experiments, which are explained in more detail in the following sections, word embeddings are concatenated to the sentence embbeddings before being fed as input to the final classifier.

In the sentence length and word content experiments we train a logistic classifier as explained in Section 2.1.1. Adi et al. (2016) train a neural network on those tasks, but we achieve comparable with our approach. The classifier is trained by minimizing the mean cross-entropy loss (see Section 2.1.2) across all sentences in a training batch. We use the Adam optimizer (Kingma and Ba, 2014) and tried out combinations of different batch sizes (32, 64, 128) and learning rates (0.01, 0.001, 0.0001). Upon running initial experiments we found that the best results were achieved with batch size 64 an learning rate 0.0001.

In the dependency tag task, we tried out using both a logistic classifier and a neural network classifier with a single hidden layer with 100 units as explained in

Section 2.1.4. The neural network classifier performed significantly better so we chose this over a simple logistic classifier. We also found that using a small amount of dropout (Srivastava et al., 2014) (dropout probability = 0.1) improved the results.

Since we are mostly interested in the relative performance of different models rather than achieving the highest possible accuracy, this concludes the hyperparameter optimization. Hyperparameter search beyond this would be recommended in a larger scale study.

We measure the performance of the classifiers both in terms of accuracy (the percentage of correctly classified instances) and the weighted average F1-score. That is, we calculate F1 score for each class and then weigh it by the number of instances of each class. The F1 score can be interpreted as the mean of *precision* and *recall*. It is a special case of the F_β score (Rijsbergen, 1979), which is a weighted average between *precision* and *recall*. There is no reason why either *precision* or *recall* should be overemphasized in this case, which is why we use the F1 score in this thesis. It is defined as

$$F_1 = 2 \cdot \frac{\text{precision} \cdot \text{recall}}{\text{precision} + \text{recall}} \tag{4.1}$$

where precision $= \frac{\text{true positives}}{\text{true positives}+\text{false positives}}$ and recall $= \frac{\text{true positives}}{\text{true positives}+\text{false negatives}}$. In unbalanced data sets the F1 score may be preferred over simpler measures such as accuracy (*i.e.* precision). That is because a classifier could reach a high accuracy by just predicting the majority class for each test instance, without having learned to discriminate between classes at all. In balanced data sets, the two measures are usually quite similar.

We evaluate the models on the dev set after each epoch and conclude training when the weighted average F1 score does not improve further. The results we show in this Chapter are obtained by evaluating the models on the test set.

4.2 Corpus

All experiments in this section make use of the SNLI corpus (Bowman et al., 2015), as introduced in Section 2.2.1.3. As the experiments do not rely on pairs of sentences, we only use the hypotheses of the corpus. We keep the original split in train, dev and test set. That is, the train, dev and test sets consist of 500,000, 10,000 and 10,000 sentences, respectively.

4.3 Tasks

4.3.1 Sentence length

As in Adi et al. (2016), we group sentences together based on their length. We use the bins: [0-4], [5-6], [7-8], [9-10], [11-12], [13-60]. These deviate from the ones in Adi et al. (2016), because a different corpus is used. As shown in Table 4.1 this choice of bins leads to a reasonably balanced data set with the majority class accounting for around 31.5% of the data.

Table 4.1: Distribution of sentence lengths

	0-4	5-6	7-8	9-10	11-12	13-60
train	6.8%	23.7%	31.5%	18.9%	9.8%	9.2%
dev	6.4%	23.1%	31.3%	19.1%	10.2%	9.9%
test	6.2%	23.6%	31.6%	19.2%	9.4%	9.9%

We train a logistic classifier as explained in Section 2.1.1 with just the sentence embedding $\mathbf{s} \in \mathbb{R}^D$ as input and binned sentence lengths as labels. After each epoch, we evaluate the model performance in terms of the weighted average F1 score across all bins on the dev set. Training is concluded after 20 epochs and the best model is selected on the basis of dev set performance.

4.3.2 Word content

In this task we train a binary classifier to predict whether or not a word occurs in a sentence, on the basis of the corresponding sentence and word embeddings alone. That is, we create a feature vector by concatenating a sentence embedding and a word embedding, which is then the input into a logistic classifier as described in Section 2.1.1.

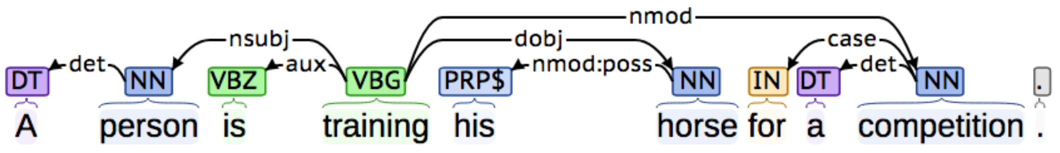
We create a positive and a negative training instance for every sentence in our data set. The positive instances are simply created by taking a word that occurs in a sentence at random. The negative examples have to be created more carefully. A naive approach would be to simply select a word for each sentence which does not occur in that sentence. But this would be problematic, as the classifier would to some degree be able to just remember which words are positive and which words are negative examples. A solution to this problem is to create negative examples by selecting a word which does not occur in the given sentence, but which is among the set of positive examples for other sentences. This way, the classifier is forced to learn whether a word occurs in a sentence by not only taking into account the word embedding, but also the sentence embedding.

4.3.3 Dependency tree tags

In this task the goal is to train a classifier which predicts the dependency tag between two words, based on the embedding of the sentence in which they occur, and their respective word embeddings.

A labelled data set is created using a dependency parser. Dependency parsing refers to the process of extracting the grammatical relationships between words in a sentence. Dependencies are directed relations and can be expressed as 3-tuples consisting of *child*, *parent* and a *tag* i.e. the type of relation. They can be illustrated in the form of dependency trees, where the *child* is commonly the word at the arrow head. Figure 4.1, based on Stenetorp et al. (2012), shows the dependency tree for the sentence ”A person is training his horse for a competition”.

Figure 4.1: Dependency tree



In this thesis, the Stanford neural network dependency parser (Chen and Manning, 2014) is used to create dependency tags for each of the sentences in the corpus. For the example sentence above, the parser creates the tuples shown in Table 4.2.

Table 4.2: Training examples created by dependency parser

['A', 'person', 'det']
['person', 'training', 'nsubj']
['is', 'training', 'aux']
['his', 'horse', 'nmod:poss']
['horse', 'training', 'dobj']
['for', 'competition', 'case']
['a', 'competition', 'det']
['competition', 'training', 'nmod]

The tags *ROOT* and *PUNCT* are excluded as they do not carry interesting semantic information. For a sentence of length T, there are usually in the order of T dependency tags, out of a total of $T(T-1)$ possible tags. In other words, it is far more likely that there is no dependency between any randomly picked pair of words than that there is a dependency. A good classifier should not only be able to predict the dependency tag between words for which it is known that there is a dependency between them, but it should also be able to predict whether there is any dependency between the words at all. For that reason, a *None* class is created, in addition to the dependency tags resulting from the dependency parser. It is computationally expensive to create all possible negative examples, *i.e.* all word pairs that do not have a dependency relation, for all sentences in the corpus. Therefore three experiments are conducted in ascending difficulty: for every sentence in the corpus, either 3, 20 or 50 negative examples are created. For the example sentence above, this could lead to random negative samples such as ['a', 'horse', 'None'], ['for', 'his', 'None'] and ['is', 'competition', 'None'].

A neural network classifier with a single hidden layer with 100 units (Section 2.1.4) is trained to predict the dependency tags between two words w_1 and w_2 that occur in a sentence s. The input to the classifier varies according to which model is used. For the CBOW baseline, the input is a concatenation of the average of all word vectors that occur in the sentence, $\mathbf{s} \in \mathbb{R}^{300}$, and the GloVe word vectors $\mathbf{w_1}, \mathbf{w_2} \in \mathbb{R}^{300}$ corresponding to *child* and *parent*, resulting in a 900-dimensional input vector. For the skip-thought model, the input is a concatenation of the skip-thought sentence embedding $\mathbf{s} \in \mathbb{R}^{2400}$ and the two skip-thought word vectors $\mathbf{w_1}, \mathbf{w_2} \in \mathbb{R}^{620}$. The input is therefore 3640-dimensional. And for the BiLSTM-max model, the input is a concatenation of the BiLSTM-max sentence embedding $\mathbf{s} \in \mathbb{R}^{4800}$ and the GloVe word vectors $\mathbf{w_1}, \mathbf{w_2} \in \mathbb{R}^{300}$ resulting in a 5400-dimensional vector. Training is concluded if the weighted average F1 score does not increase any further, which happens after around 8 epochs.

4.4 Results

4.4.1 Sentence Length

Figure 4.2 illustrates the performance of the different models in the sentence length prediction task in terms of the weighted F1 score on the test set. Table 4.3 provides some further detail.

Figure 4.2: Sentence length prediction results (weighted F1 score)

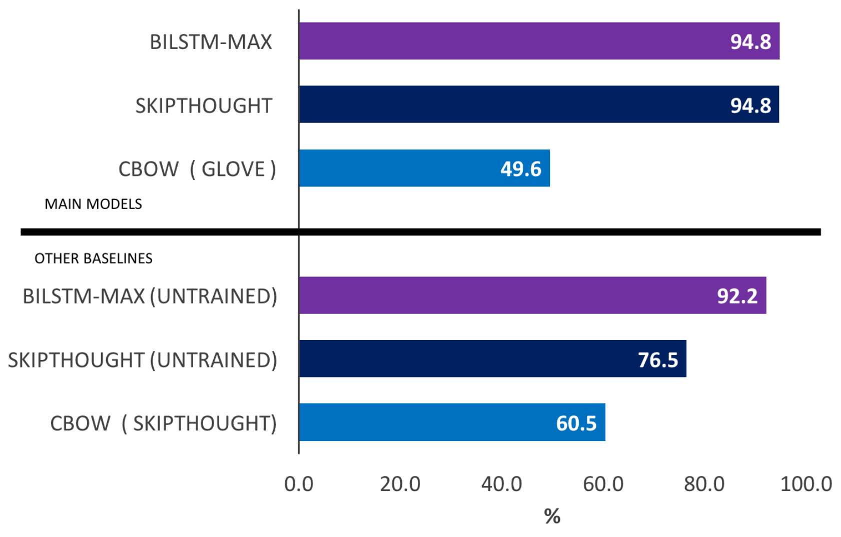


Table 4.3: Sentence length prediction results

Model	Dimensions	Weighted F1	Accuracy
Main models			
CBOW	300	49.6	52.4
Skip-thought	2400	94.8	95.4
BiLSTM-max	4096	94.8	95.4
Other baselines			
CBOW (skipthought)	620	60.5	61.3
Skip-thought (untrained)	2400	76.5	77.3
BiLSTM-max (untrained)	4096	92.2	92.9

The key observations are:

1. As expected, the CBOW model performs considerably worse than the RNN-based models, because it does not take into account the order of words. But it performs better than untrained models, or a classifier that would just predict the majority class which would reach an accuracy of around 4.1. This is in line with Adi et al. (2016) whose explanation for this is that there is a correlation between sentence length and the norm of the sentence representation resulting from averaging word vectors.

2. The Skip-thought and BiLSTM-max models perform on par, despite being trained on different corpora and having different numbers of dimensions

3. The CBOW model using the average of Skip-thought vectors performs better than the CBOW model using GloVe vectors. A likely explanation is that the Skip-thought word vectors are higher-dimensional and therefore carry more information about words.

4. Both the untrained Skip-thought and BiLSTM-max models perform worse than their trained counterparts. The untrained BiLSTM-max performs only slightly worse than the trained model. This is in line with Conneau et al. (2017) who find a similar result when evaluating the models on the downstream evaluation tasks explained in Section 3.4. Interestingly, the untrained Skip-thought model performs considerably worse, despite the word vectors it takes as input are higher dimensional.

4.4.2 Word content

Figure 4.3 illustrates the performance of the different models in the word content prediction task in terms of the weighted F1 score on the test set. Table 4.4 provides some further detail.

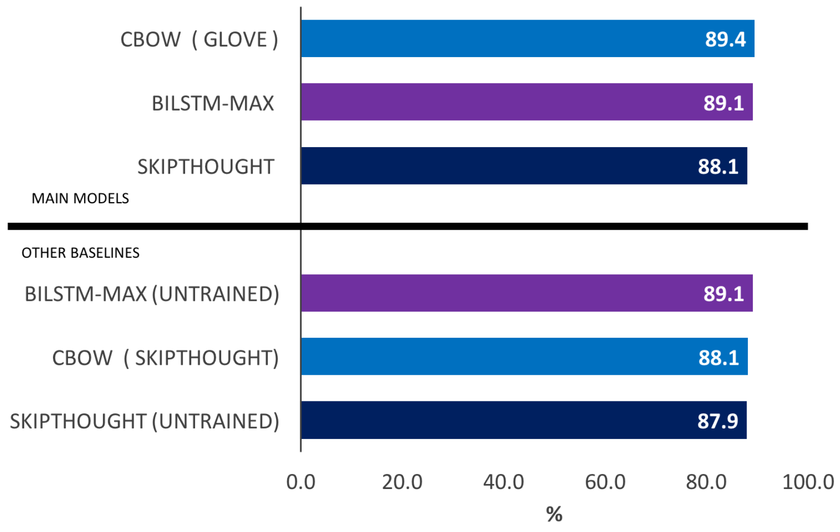
Table 4.4: Word content prediction results

Model	Dimensions	Weighted F1	Accuracy
Main models			
CBOW	300 + 300	89.4	90.1
Skip-thought	2400 + 620	88.1	88.7
BiLSTM-max	4096 + 300	89.1	89.8
Other baselines			
CBOW (skipthought)	620 + 620	88.1	88.7
Skip-thought (untrained)	2400 + 620	87.9	88.5
BiLSTM-max (untrained)	4096 + 300	89.1	89.8

The key observations are:

1. While the CBOW model was performing relatively poorly on the sentence length prediction task, it is performing well on the word content task. In fact, it is the highest performing model. A possible explanation for this is that the

Figure 4.3: Word content prediction results (weighted F1 score)



CBOW sentence embeddings contain no information other than the word embeddings, while the other two models contain information about word order etc. As the task is purely to predict whether a word is present in the sentence, this might give the CBOW model an edge, as it contains less ’irrelevant’ information for this particular task.

2. The untrained version of the BiLSTM-max model performs on par with its trained version. And the untrained Skip-thought model only performs marginally worse than its trained counterpart. This means that the sequential nature of the RNNs does not add much to the word content task, which seems intuitive.

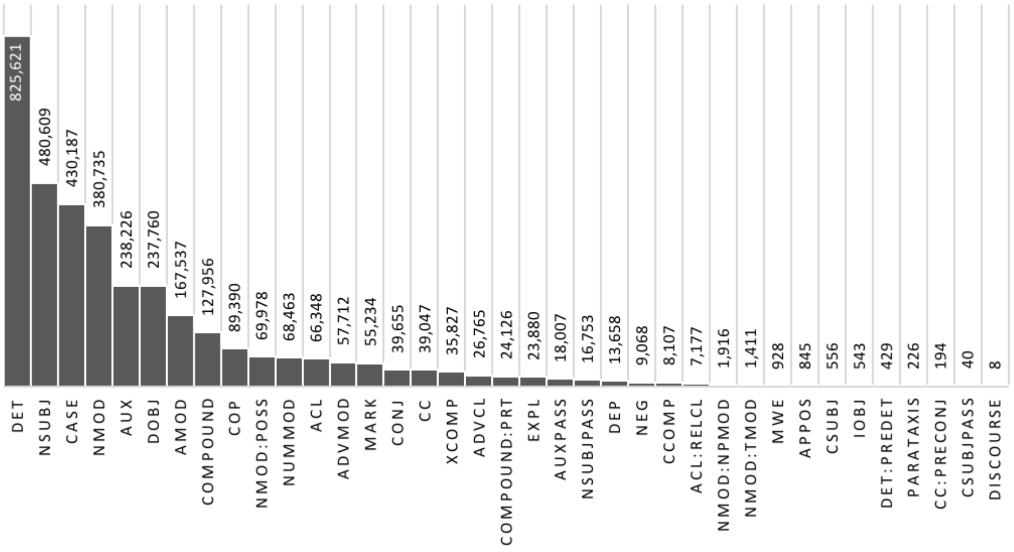
3. We find that CBOW models generally perform well on the word content task. Also, the higher dimensional CBOW model (the one using Skip-thought vectors) performs worse. Both these findings are in line with Adi et al. (2016)

4.4.3 Dependency tags

The distribution of dependency tags is highly non-uniform as can be seen in Figure 4.4. Furthermore, for most words in the vocabulary there is a majority class, *i.e.* a tag which is disproportionally often attached to that word. For example, out of the

4,866 times that the word *a* occurs as a *child* in the corpus, it is associated with the tag *DET* 4,860 times. For that reason, the trained classifiers are evaluated in two ways. First, on a test set including the majority class (i.e the most common tag) for each word. And second, on a test set excluding the majority class for each word. In both cases, the classifier is trained on a data set including the majority class.

Figure 4.4: Distribution of dependency tags



The results, *i.e.* the weighted average F1 score for each experiment, are shown in Table 4.5. All figures are excluding the F1 scores for the None class. This is because they would otherwise dominate the weighted average as they are by construction the most frequent label by a large margin. The key observations are:

1. Performance is better when taking into account the majority class (the most frequent dependency tag for any word) than when the majority class is excluded. This is expected on unbalanced data sets such as the one at hand.

2. Performance is better the fewer negative instances, *i.e.* word pairs without a dependency tag, are created for any sentence. This is expected, as creating more negative instances makes the data set more unbalanced.

3. Both the Skip-thought and the BiLSTM-max models perform substantially better than the CBOW baseline and their untrained counterparts. In particular, when the majority class for every word in included, the Skip-thought model performs **7.5** percentage points better than the GloVe baseline and **2.3** percentage points better than the untrained Skip-thought model (Figure 4.5). When the majority class is excluded, the respective incremental performance is **16.0** percentage points and **4.5** percentage points (Figure 4.6). This provides some evidence that the RNN-based models encode information in the sentence embeddings that is useful for predicting dependency tags between words.

Figure 4.5: Dependency tag prediction results including majority class (weighted F1 score)

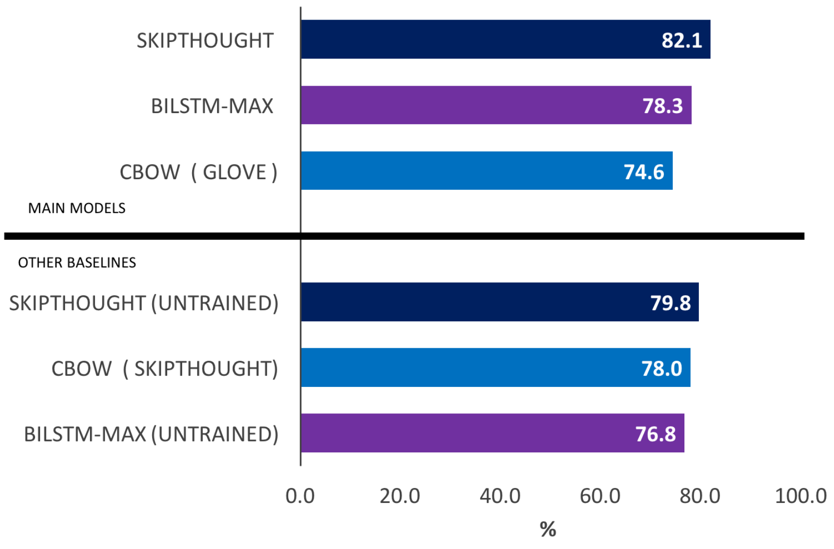


Figure 4.6: Dependency tag prediction results excluding majority class (weighted F1 score)

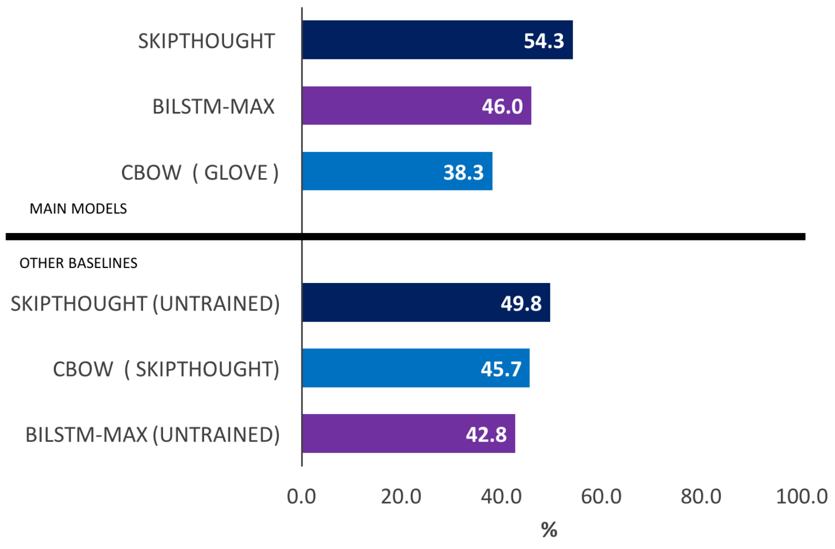


Table 4.5: Dependency tag prediction results (weighted F1 score)

Sentence embeddings	Word embeddings	Dimensions	All classes			No majority class		
			3†	20†	50†	3†	20†	50†
Main models								
CBOW	GloVe	300 + 2×300	0.8264	0.6655	0.5482	0.4828	0.3036	0.2049
Skipthought	Skipthought	2400 + 2×620	0.8710	0.7351	0.6371	0.5987	0.4096	0.3182
BiLSTM-max	GloVe	4096 + 2×300	0.8339	0.6815	0.5748	0.5254	0.3343	0.2628
Other baselines								
CBOW	Skipthought	620 + 2×620	0.8576	0.7096	0.5931	0.5574	0.3703	0.2695
Skipthought (untrained)	Skipthought	2400 + 2×620	0.8590	0.7121	0.6212	0.5606	0.3696	0.2904
BiLSTM-max (untrained)	GloVe	4096 + 2×300	0.8332	0.6719	0.5714	0.5062	0.3107	0.2535

† Refers to the max number of negative examples creates for each sentence.

All results shown here represent the F1 score for each dependency tag, weighted by their frequency. This excludes the *NONE* class, which by construction occurs very frequently, is predicted with very high frequency and high accuracy and which would therefore artificially increase the results shown.

4.5 Further analysis of dependency tags

In this section we investigate further the differences across models on the dependency tag prediction task. We focus on the experiment including the majority class (*i.e.* the most frequent label for every word) and in which up to 50 negative examples are created for each sentence were generated.

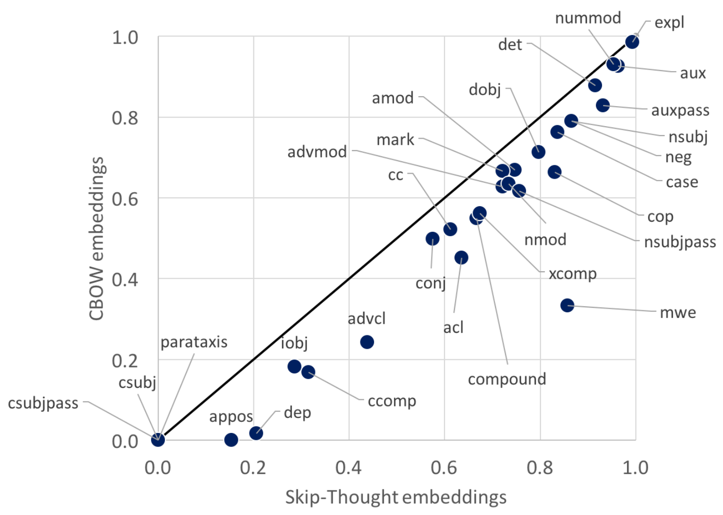
4.5.1 Analysis by class label

In Figure 4.7 we plot the performance of the Skip-Thought and BiLSTM-max models relative to the CBOW baseline, broken down by dependency tag. As can be seen, the Skip-thought model outperforms the baseline on all tags. The BiLSTM-max model outperforms the baseline on all tags apart from the tags 'conj' and 'neg'. 'conj' refers to conjunction, and an example for this label would be the words 'big' and 'honest' in the sentence 'Bill is big and honest'. 'neg' refers to negation and an example would be the words 'scientist' and 'not' in the sentence 'Bill is not a scientist'. In the case of the Skip-thought there is one extreme outlier, the tag 'mwe', on which the Skip-thought model performs substantially better than the baseline. This refers to multi-word-expression, for example the words 'well' and 'as' in the sentence 'I like dogs as well as cats'.

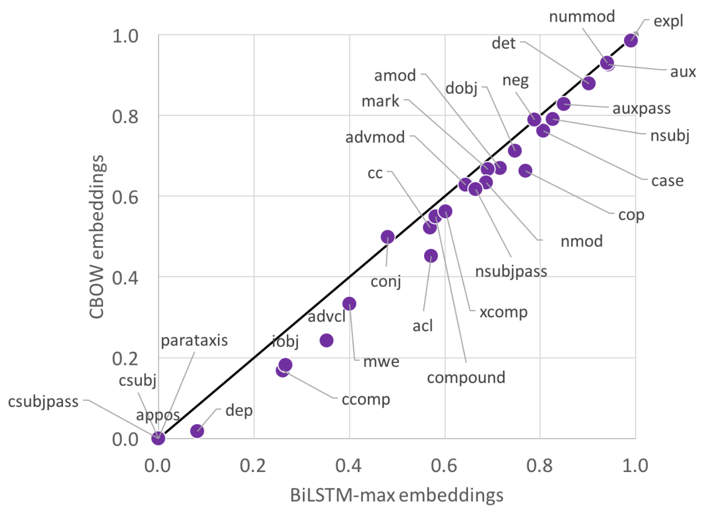
We test whether the results shown in Figure 4.7 are driven by the number of observations of each class. To that end we plot the Skip-thought F1 score relative to the number of observations in Figure 4.8. As can be seen, there is some positive correlation. In particular, most of the classes for which the Skip-thought model predicts an F1 score above 0.6 occur more than 1,000 times on the test set. And the classes for which the model has a F1 score of less than 0.2 occur only around 10 times in the test set. Note that the proportions of classes in train, dev and test set are very similar, as the labels are generated by the same process. Again, we see that the tag 'mwe' is an outlier.

Figure 4.7: RNN-based vs. CBOW weighted F1, including majority class, 50 negative samples

(a) Skipthought vs. CBOW



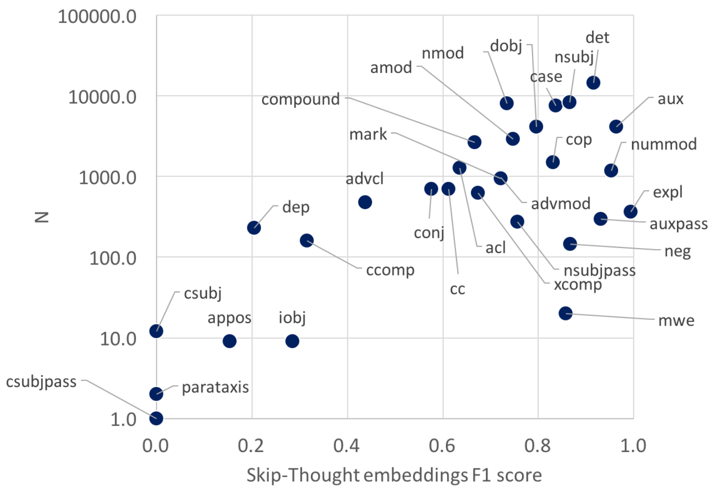
(b) BiLSTM-max vs. CBOW



4.5.2 Additional information encoded by RNN-based models

We have seen that RNN-based sentence encoders seem to encode more information about a sentence than a simple CBOW baseline. As a last step, we will investigate if it is possible to quantify and visualize what additional information RNNs encode relative to a CBOW baseline.

Figure 4.8: Skip-thought F1 score on dependency tag prediction vs. number of observations in each class



To this end, we prepare three sets of 40 sentences pairs each. Within each set, the sentences of any pair are the same, except they differ according to the theme of the set. The themes are *Masculine vs. Feminine*, *Singular vs. Plural* and *Present vs. Past*. For example, a sentence pair in the set *Masculine vs. Feminine* would be ’He walked down the alley’ and ’She walked down the alley’. We ensure that there is enough variation in the words that express the polarity. So rather than using ’he’ and ’she’ in every example we also use sentence pairs like ’The host welcomes the guests’ and ’The hostess welcomes the guests’. A full list of sentences used is shown in Appendix B.

Our analysis for any set of sentences involves the following steps:

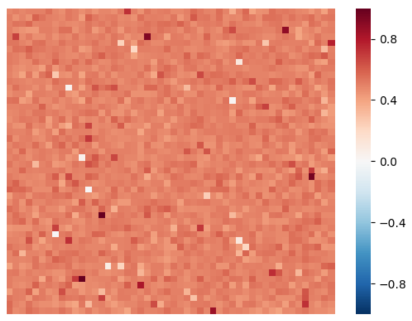
1. Encode all sentences in the set using one of our models;

2. Calculate the mean embedding for the two types of sentences (*e.g.* masculine and feminine). We normalize the values to be between zero and one. See panels (a) and (b) of Figure 4.9;

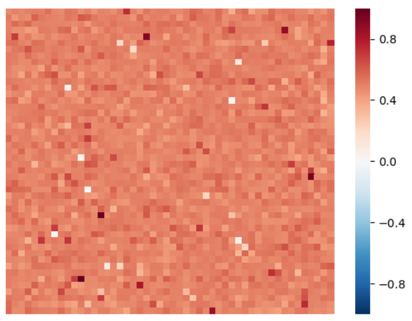
3. We take the difference between the two mean embeddings. This give us some sense about how different the sentence embeddings are and at which dimen-

Figure 4.9: Skip-thought vector embedding of masculine vs. feminine sentences

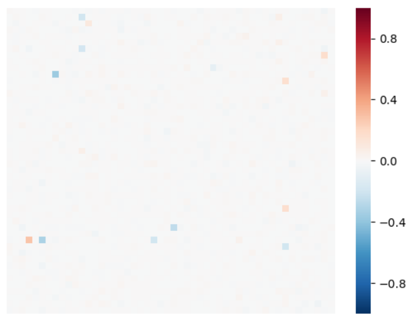
(a) Mean of 'Masculine' sentences (normalized)



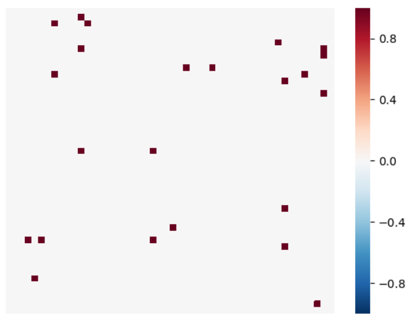
(b) Mean of 'Feminine' sentences (normalized)



(c) Difference between 'masculine' and 'feminine' sentences



(d) Significant differences at 5% confidence



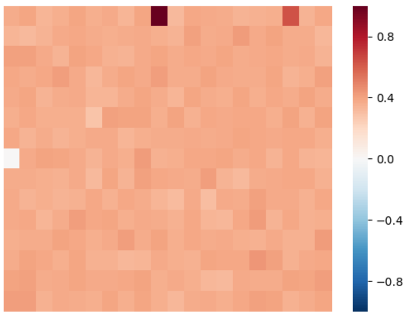
sions. See panel (c) of Figure 4.9;

4. In a more quantitative approach, We calculate whether the differences at any dimension are statistically significant. That is, we test whether the hypothesis that the difference between means is different from 0 can be rejected with 95% confidence, separately for every dimension. Let $\mu_{x,masculine}$ and $\sigma_{x,masculine}$ be the mean and standard deviation of the masculine sentence embeddings along dimension x. Remember, there are 300, 2400 and 4096 dimensions in the CBOW, Skip-thought and BiLSTM-max models respectively. Let the standard error ε_x be defined as $\varepsilon_x = \sqrt{\frac{\sigma^2_{x,masculine}}{n_{masculine}} + \frac{\sigma^2_{x,feminine}}{n_{feminine}}}$ where the number of observations $n_{masculine} = n_{feminine} = 40$. We calculate the critical t-value t^* corresponding to a two-sided confidence interval of 5% using 39 degrees

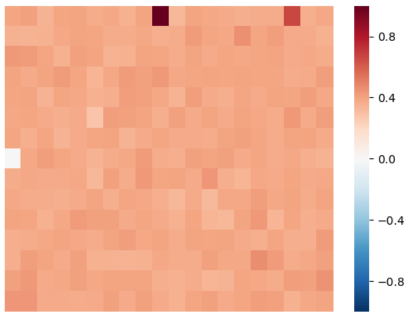
of freedom. We then illustrate the dimensions at which the absolute value of the t-statistic $t = \frac{\mu_{x,masculine} - \mu_{x,feminine}}{\varepsilon_x}$ exceeds the critical value in panel (d) in Figure 4.9.

Figure 4.10: CBOW vector embedding of masculine vs. feminine sentences

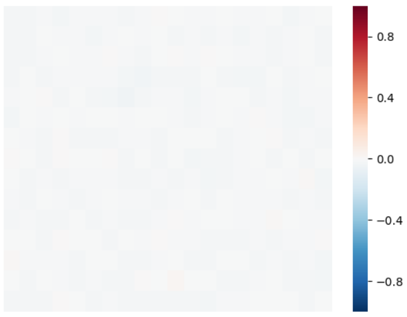
(a) Mean of 'Masculine' sentences (normalized)



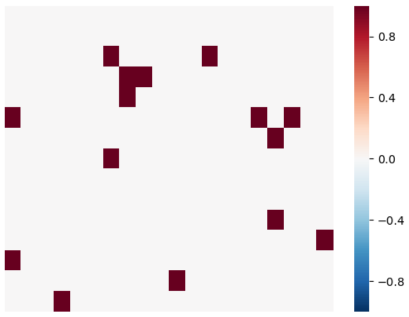
(b) Mean of 'Feminine' sentences (normalized)



(c) Difference between 'masculine' and 'feminine' sentences



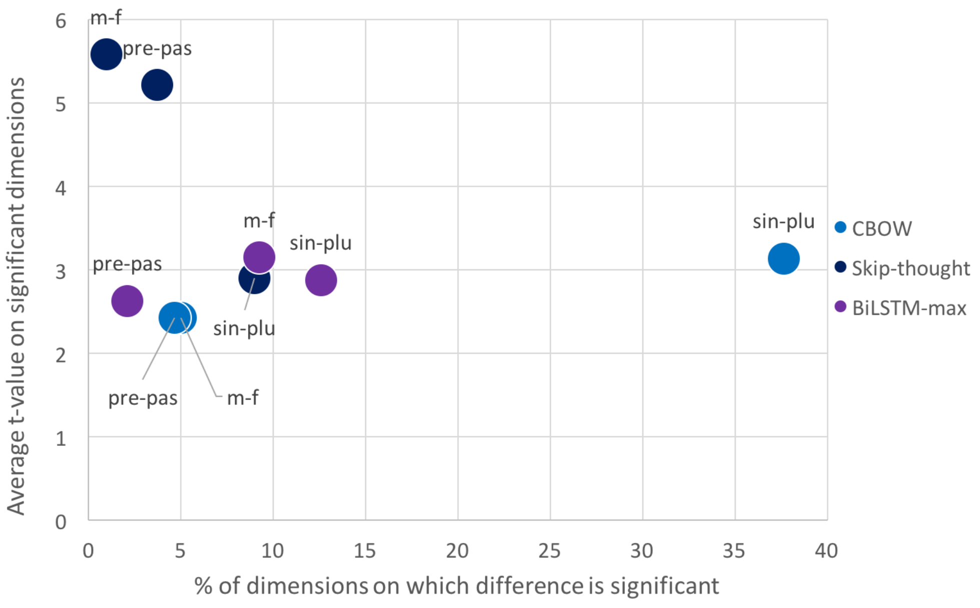
(d) Significant differences at 5% confidence



We show the results of this analysis on the *masculine vs. feminine* data set using the Skip-thought model in Figure 4.9 and using the CBOW baseline in Figure 4.10. Figures for other data sets and all models can be found in Appendix C. Upon inspection of panels (c) the Skip-thought model seems to encode gender information more strongly than the CBOW model. But it would be a false conclusion that the CBOW does not encode this information at all. Indeed, panel (d) suggests that the differences between mean 'masculine' and 'feminine' sentence embeddings are also significant on some dimensions in the CBOW embeddings.

We substantiate this claim by analyzing the the mean t-value on dimensions for which the difference is statistically significant. In other words, we compare the mean t-values of the dimensions highlighted in panels (d).

Figure 4.11: Scatter plot between % of dimensions on which difference between



m-f: masculine-feminine, pre-pas: present-past, sin-plu: singular-plural

There are different ways of interpreting the results in Figure 4.11. For a more systematic analysis, this should be repeated for more than just three themes. However we can make the following observations:

1. For the Skip-thought model, the share of dimensions for which the difference is significant is low relative to the BiLSTM-max and CBOW models. However, the dimensions that are significant, are so very strongly. Put differently, in the Skip-thought sentence embeddings, relatively few dimensions matter, but they matter a lot. This could be a reason why the Skip-thought model performs better than the BiLSTM-max model in this thesis.

2. When encoding sentences in which the subject is singular vs. when it is plural, we observe that the resulting sentence embeddings vary significantly on more dimensions, relative to the other sets of sentences.

Chapter 5

Conclusions

The aim of this thesis was to investigate what Recurrent Neural Networks (RNNs) learn when they 'read' a sentence. To this end, we trained two state-of-the-art RNN sentence encoders: the Skip-thought (Kiros et al., 2015) and the BiLSTM-max (Conneau et al., 2017) models. For comparison purposes, we used a CBOW (continuous-bag-of-words) baseline, which encodes sentences by averaging pre-trained GloVe (Pennington et al., 2014) word embeddings.

Our research builds in particular on work by Adi et al. (2016) and Linzen et al. (2016). The former infer sentence features such as (i) sentence length) (ii) word content and (iii) word order from sentence embeddings. And the latter show that LSTMs are capable of learning syntactic features of sentences. In this thesis, we use our trained models to obtain sentence embeddings, and infer sentence length and word content from them. We also introduce a new task focused on syntactic features: to predict dependency tags between words on the basis of a sentence embedding and two word embeddings.

5.1 Summary of key results

The key results of this thesis can be summarized as follows:

1. **Sentence length task:** The highest performing model is BiLSTM-max, which achieves a weighted average F1 score of 94.8%. The CBOW model only achieves 49.6%. In other words, the RNN outperforms the non-RNN

model by **45.2** percentage points. This provides evidence that RNNs encode information about sentence length in sentence embeddings;

2. **Word content task:** The highest performing model is CBOW, which achieves a weighted average F1 score of 89.4%. This is on par with the RNN-based sentence encoders and therefore provides evidence that both RNNs and non-RNN models encode information about the presence of words in sentence embeddings;

3. **Dependency tag task (inc. majority class):** The highest performing model is Skip-thought, which achieves a weighted average F1 score of 82.1%. The CBOW model only achieves 74.6%. In other words, the RNN outperforms the non-RNN model by **7.5** percentage points. This provides some evidence that RNNs encode information about the syntactic relationships between words in sentence embeddings; and

4. **Dependency tag task (exc. majority class):** Some words have a particular dependency tag attached to it very frequently. A naive model could just learn these *majority classes* by heart and achieve good performance. For that reason, in this task the majority class of every word is excluded. The highest performing model is Skip-thought, which achieves a weighted average F1 score of 54.3%. The CBOW model only achieves 38.3%. In other words, the RNN outperforms the non-RNN model by **16** percentage points. This provides further evidence that RNNs encode information about the syntactic relationships between words in sentence embeddings;

The main contribution of this thesis is to show that RNN sentence encoders perform better than a non-RNN baseline on the dependency tag prediction task. This provides some evidence that RNNs are able to learn about the syntactic relationships between words. This is remarkable, because this was not part of the objective they were initially trained on.

We also visualize sentence embeddings obtained from different models. This illustrates that certain dimensions carry information about specific aspects of a sentence. We provide some evidence that there are differences across models with respect to how strongly, and in how many dimensions such aspects are encoded. This offers a potential explanation for why embeddings obtained from RNNs perform better on certain classification tasks than embeddings obtained from non-RNN methods.

5.2 Critique

There are some limitations and points of criticism to our approach:

1. A more thorough hyperparameter optimization, *i.e.* a random search of parameters, should be undertaken. The reasons why we only conducted a grid search are that (i) the results presented in Section 4 are in line with those of Adi et al. (2016) and (ii) the goal of this thesis was not to find the highest possible accuracy, but rather to highlight the differences between different models.

2. The data set that we use for our experiments in Section 4 is the SNLI corpus (Bowman et al., 2015). The average sentence length is around 7 words and the sentences are mostly grammatically very simple. The results of our experiments should therefore be validated on a more difficult corpus, such as for example the BookCorpus (Zhu et al., 2015), on which our Skip-thought model was trained.

3. The word content experiments in Section 4.3.2 are carried out with a balanced data set. This means that the models are evaluated under the assumption that any randomly picked word is equally likely to be part of the sentence or not. This is the same approach as was used by Adi et al. (2016). But in reality, the probability that a randomly picked word is not part of any given sentence is of course much higher. Therefore this experiment would ideally be conducted in a similar way as our dependency tag experiments in Section 4.3.3. Here we ensure that the model is evaluated under more realistic conditions, *i.e.* on an unbalanced data set.

4. The dependency tag experiments in Section 4.3.3 are carried out by generating up to 50 negative samples for each sentence. This number is chosen because the average sentence length is around 7 words. So for the average sentence, most of the possible negative samples are created. A more thorough approach would be to indeed create all possible negative samples for all sentences.

5. Lastly, in Section 4.5.2 we provide some evidence that there are differences in terms of how RNNs encode particular sentence features. This evidence is not conclusive as our results rely on a small data set and on arbitrarily chosen categories. The results should therefore be verified systematically on a larger data set.

5.3 Further Research

The finding that RNNs encode syntactic relationships between words is reaffirming, as this task requires taking into account the sequential nature of natural language which is the very objective of RNNs. There are several ways in which the results of this thesis could be taken forward:

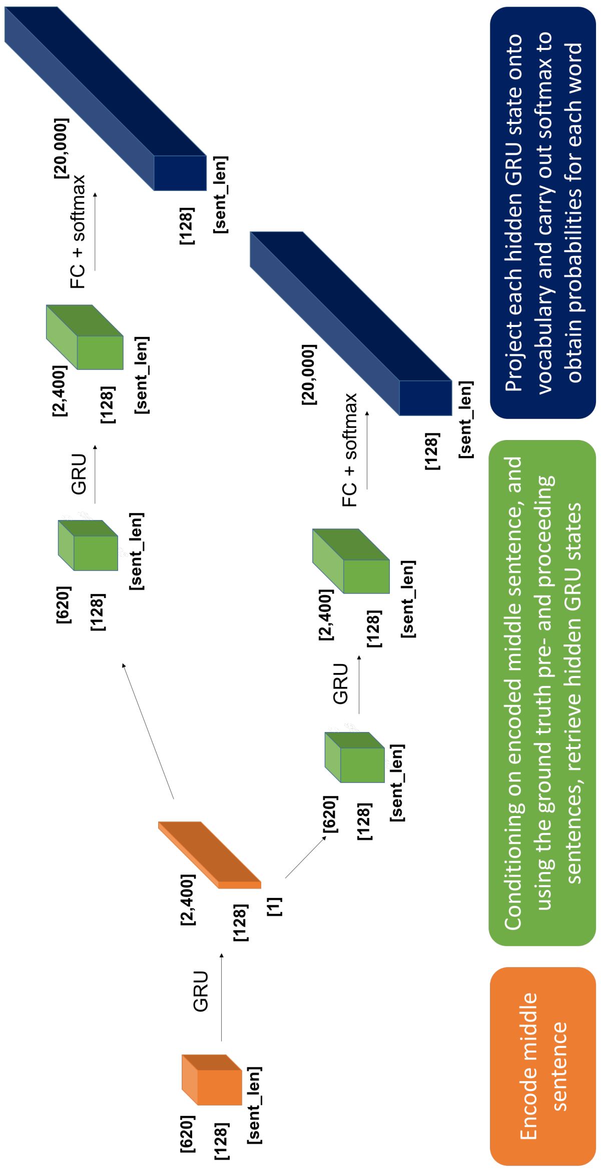
1. This thesis only investigates whether direct syntactic relationships between words , *i.e.* between a *parent* and a *child*, can be inferred from sentence embeddings. A further area of research is to investigate if also longer ranging syntactic relationships, *i.e.* relationships between words which are indirectly connected in a dependency parse tree, can also be retrieved from sentence embeddings.

2. This thesis shows that features such as dependency tags can be retrieved from sentence embeddings. An interesting question is whether these features coincide with features which have been successfully used in models that rely on feature-engineering. In other words, do RNNs encode sentence features which humans also found to be important? One way of achieving this would be to train a baseline using hand-crafted features, and to investigate whether

the weights of the most expressive features correspond to the features which can be retrieved from neural sentence embeddings.

Appendix A

Skipthought model architecture


[620]
[128]
[sent_len]
GRU
[2,400]
[128]
[1]
[620]
[128]
[sent_len]
GRU
[2,400]
[128]
[sent_len]
FC + softmax
[20,000]
[128]
[sent_len]
[620]
[128]
[sent_len]
GRU
[2,400]
[128]
[sent_len]
FC + softmax
[20,000]
[128]
[sent_len]
Encode middle sentence
Conditioning on encoded middle sentence, and using the ground truth pre- and proceeding sentences, retrieve hidden GRU states
Project each hidden GRU state onto vocabulary and carry out softmax to obtain probabilities for each word

Appendix B

Sentence pairs used for investigation of sentence embeddings

’Masculine’ sentences

’He walked down the alley’, ’The cook sharpened his knife’, ’Paul rings the door bell’, ’Adam eats a sandwich for lunch’, ’Rock music is not his favourite’, ’Once in a while, a man comes to buy flowers’ , ’The school boy forgot his books’, ’A man is sitting near a bike and is writing a note ’, ’Two men are taking a break from a trip on a snowy road’, ’A boy is standing outside the water ’, ’A few men in a competition are running outside’, ’A man is playing the flute’, ’Some dangerous men with knives are throwing a bike against a tree ’, ’A man is climbing a rope’, ’There is no man drawing’, ’A guy is cutting some ginger’, ’Chris is a keen swimmer’, ’Mr May is a poor guy’, ’The king found a way to the tower’, ’The Duke eats chicken’, ’The Little Baron is a movie’, ’The Emperor watches the sun set’, ’Everyone deserves a good brother’, ’My big brother is a student’, ’Male students study in the library’, ’The librarian writes with his pen’, ’The children and their uncles are going to the kindergarden’, ’Uncles are brothers of your parents’, ’The parents have a son’, ’The son has a cat’, ’Gentlemen are smoking cigars’, ’Dave drinks a coffee in the cafe’, ’The prince is melking cows’, ’Stewards are working on a plane’, ’The host welcomes the guests’, ’The waiter brings the bill’, ’In some schools the headmaster is not very popular’, ’In order to secure safety, policemen are waiting outside’, ’The chairman introduces new colleagues’, ’In Hollywood, there are many actors’

’Feminine’ sentences

’She walked down the alley’, ’The cook sharpened her knife’, ’Paula rings the door bell’, ’Eva eats a sandwich for lunch’, ’Rock music is not her favourite’, ’Once in a while, a woman comes to buy flowers’ , ’The school girl forgot her books’, ’A woman is sitting near a bike and is writing a note ’, ’Two women are taking a break from a trip on a snowy road’, ’A girl is standing outside the water ’, ’A few women in a competition are running outside’, ’A woman is playing the flute’, ’Some dangerous women with knives are throwing a bike against a tree ’, ’A woman is climbing a rope’, ’There is no woman drawing’, ’A girl is cutting some ginger’, ’Kate is a keen swimmer’, ’Mrs May is a poor girl’, ’The queen found a way to the tower’, ’The Duchess eats chicken’, ’The Little Baroness is a movie’, ’The Empress watches the sun set’, ’Everyone deserves a good sister’, ’My big sister is a student’, ’Female students study in the library’, ’The librarian writes with her pen’, ’The children and their aunts are going to the kindergarden’, ’Aunts are sisters of your parents’, ’The parents have a daughter’, ’The daughter has a cat’, ’Ladies are smoking cigars’, ’Dana drinks a coffee in the cafe’, ’The princess is melking cows’, ’Stewardesses are working on a plane’, ’The hostess welcomes the guests’, ’The waitress brings the bill’, ’In some schools the headmistress is not very popular’, ’In order to secure safety, policewomen are waiting outside’, ’The chairwoman introduces new colleagues’, ’In Hollywood, there are many actresses’

’Present’ sentences

’He walks down the alley’, ’The cook sharpens his knife’, ’Paul rings the door bell’, ’Adam eats a sandwich for lunch’, ’Rock music is not his favourite’, ’Once in a while, a man comes to buy flowers’ , ’The school boy forgets his books’, ’A man is sitting near a bike and is writing a note ’, ’Two men are taking a break from a trip on a snowy road’, ’A boy is standing outside the water ’, ’A few men in a competition are running outside’, ’A man is playing flute’, ’Some dangerous women with knives are throwing a bike against a tree ’, ’A woman is climbing a rope', 'There is no woman drawing', 'A girl is cutting some ginger', 'Kate is a keen swimmer', 'Mrs May is a poor girl', 'The queen finds a way to the tower', 'The Duchess eats chicken', 'The Little Baroness is a movie', 'The Empress watches the sun set', 'The police man prepares his uniform', 'The car drives down the street', 'President Trump is lying', 'Ranger Smith lights the fire', 'The camp fire glows in the dark', 'Dark rooms have no light', 'Light weight boxers fight', 'Fighting turtles are popular', 'Popular singers sell many CDs', 'Music industry bosses earn a lot of money', 'Money does not make you happy', 'Happy hippies dance in trance', 'Trance artists use synthetic instruments', 'Instruments lie on the floor in the concert hall', 'Hall publishes economic articles', 'Articles in the newspaper contain useful information', 'Information leaks mean problems for the agents', 'Chemical agents react with others'

'Past' sentences

'He walked down the alley', 'The cook sharpened his knife', 'Paul rang the door bell', 'Adam ate a sandwich for lunch', 'Rock music was not his favourite', 'Once in a while, a man came to buy flowers' , 'The school boy forgot his books', 'A man was sitting near a bike and is writing a note ', 'Two men were taking a break from a trip on a snowy road', 'A boy was standing outside the water ', 'A few men in a competition were running outside', 'A man was playing flute', 'Some dangerous women with knives were throwing a bike against a tree ', 'A woman was climbing a rope', 'There was no woman drawing', 'A girl was cutting some ginger', 'Kate was a keen swimmer', 'Mrs May was a poor girl', 'The queen found a way to the tower', 'The Duchess ate chicken', 'The Little Baroness was a movie', 'The Empress watched the sun set', 'The police man prepared his uniform', 'The car drove down the street', 'President Trump was lying', 'Ranger Smith lit the fire', 'The camp fire glew in the dark', 'Dark rooms had no light', 'Light weight boxers fought', 'Fighting turtles were popular', 'Popular singers sold many CDs', 'Music industry bosses earned a lot of money', 'Money did not make you happy', 'Happy hippies danced in trance', 'Trance artists used synthetic instruments', 'Instruments lay on the floor in the concert hall', 'Hall published economic articles', 'Articles in the newspaper contained useful information', 'Information leaked mean problems for the agents', 'Chemical agents reacted with others'

'Singular' sentences

'He walked down the alley', 'The cook sharpened his knife', 'Paul rings the door bell', 'Adam eats a sandwich for lunch', 'Rock music is not his favourite', 'Once in a while, a man comes to buy flowers' , 'The school boy forgot his books', 'A man is sitting near a bike and is writing a note ', 'One man is taking a break from a trip on a snowy road', 'A boy is standing outside the water ', 'One man in a competition is running outside', 'A man is playing the flute', 'A dangerous man with knives is throwing a bike against a tree ', 'A man is climbing a rope', 'There is no man drawing', 'A guy is cutting some ginger', 'Chris is a keen swimmer', 'Mr May is a poor guy', 'The king found a way to the tower', 'The Duke eats chicken', 'The Little Baron is a movie', 'The Emperor watches the sun set', 'Everyone deserves a good brother', 'My big brother is a student', 'A male student studies in the library', 'The librarian writes with his pen', 'The children and their uncle are going to the kindergarden', 'An uncle is a brother of your parent', 'The parent has a son', 'The son has a cat', 'A Gentleman are smoking cigars', 'Dave drinks a coffee in the cafe', 'The prince is melking cows', 'A steward is working on a plane', 'The host welcomes the guests', 'The waiter brings the bill', 'In some schools the headmaster is not very popular', 'In order to secure safety, a policeman is waiting outside', 'The chairman introduces a new colleague', 'In Hollywood, there is an actor'

'Plural' sentences

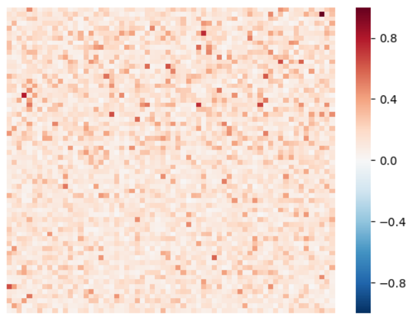
'They walked down the alley', 'The cooks sharpened their knives', 'Paul and Paula ring the door bell', 'Adam and Eve eat sandwiches for lunch', 'Rock music is not their favourite', 'Once in a while, two men come to buy flowers' , 'The school boys forgot their books', 'Two men are sitting near a bike and are writing notes ', 'Two men are taking a break from a trip on a snowy road', 'Some boys are standing outside the water ', 'A few men in a competition are running outside', 'Five men are playing the flute', 'Some dangerous men with knives are throwing a bike against a tree ', 'A man is climbing a rope', 'There are no men drawing', 'The guys are cutting some ginger', 'Chris and Ken are keen swimmers', 'Mr May and Mr April are poor guys', 'The kings found a way to the tower', 'The Dukes eat chicken', 'The Little Barons is a movie', 'The Emperors watch the sun set', 'Everyone deserves good brothers', 'My big brothers are students', 'Male students study in the library', 'The librarians write with their pens', 'The children and their uncles are going to the kindergarden', 'Uncles are brothers of your parents', 'The parents have sons', 'The sos have cats', 'Gentlemen are smoking cigars', 'Dave and David drink a coffee in the cafe', 'The princes are melking cows', 'Stewards are working on a plane', 'The hosts welcome the guests', 'The waiters bring the bills', 'In some schools the headmasters are not very popular', 'In order to secure safety, policemen are waiting outside', 'The chairmen introduce new colleagues', 'In Hollywood, there are many actors'

Appendix C

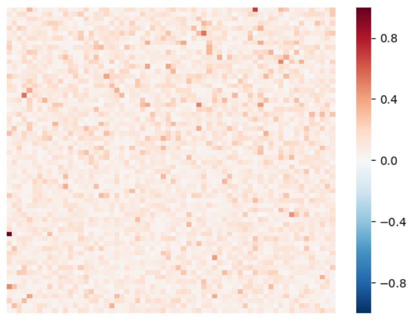
Visualizing average sentence embeddings

Figure C.1: BiLSTM-max vector embedding of male vs. female sentences

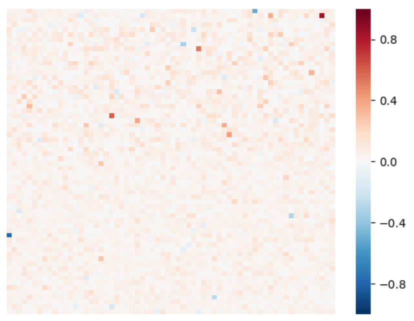
(a) Mean of ’Male’ sentences (normalized)



(b) Mean of ’Female’ sentences (normalized)



(c) Difference between ’male’ and ’female’ sentences



(d) Significant differences at 5% confidence

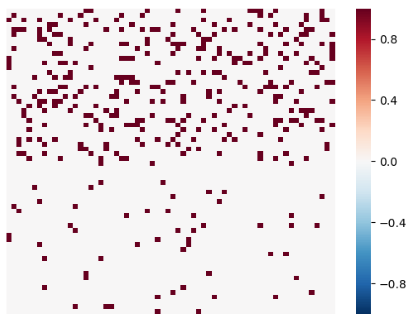
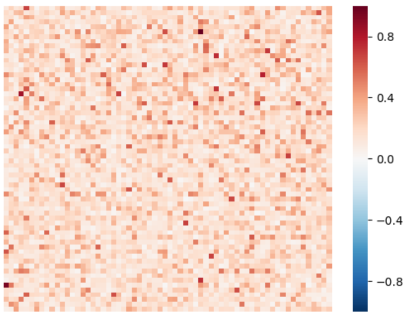
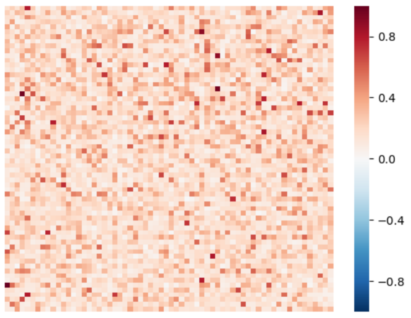


Figure C.2: BiLSTM-max vector embedding of present vs. past sentences

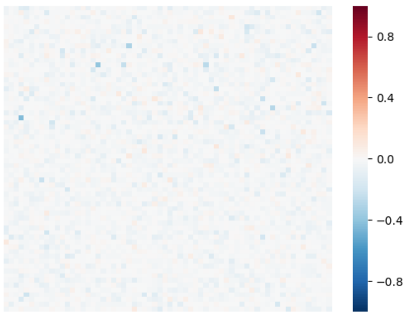
(a) Mean of ’Present’ sentences (normalized)



(b) Mean of ’Past’ sentences (normalized)



(c) Difference between ’present’ and ’past’ sentences



(d) Significant differences at 5% confidence

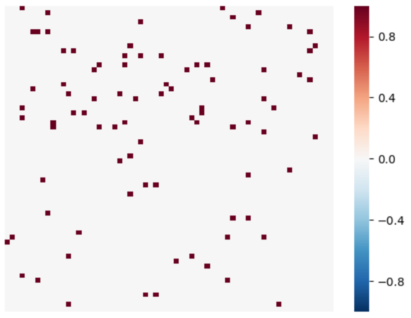
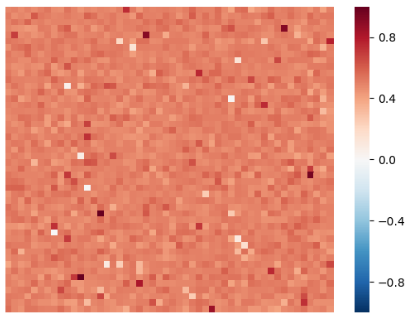
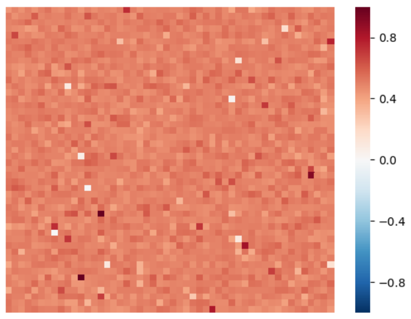


Figure C.3: Skip-thought vector embedding of present vs. past sentences

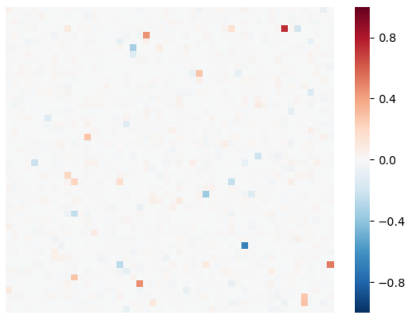
(a) Mean of ’Present’ sentences (normalized)



(b) Mean of ’Past’ sentences (normalized)



(c) Difference between ’present’ and ’past’ sentences



(d) Significant differences at 5% confidence

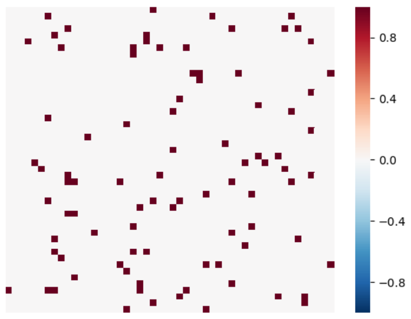
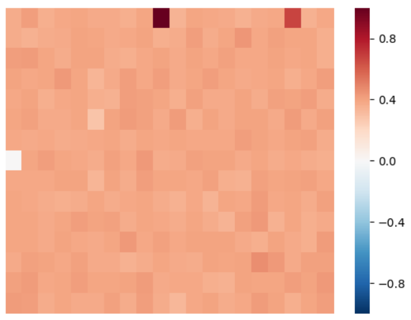
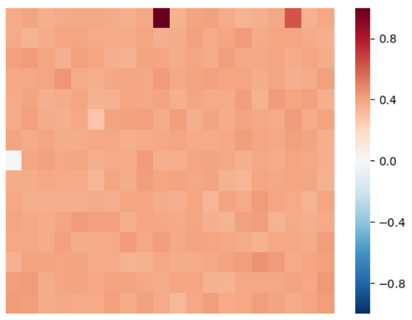


Figure C.4: CBOW vector embedding of present vs. past sentences

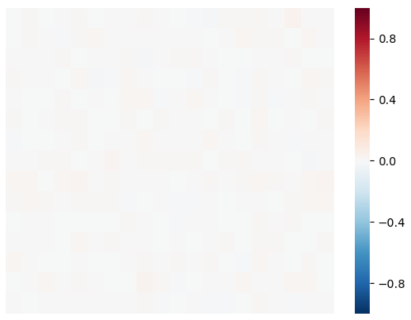
(a) Mean of 'Present' sentences (normalized)



(b) Mean of 'Past' sentences (normalized)



(c) Difference between 'present' and 'past' sentences



(d) Significant differences at 5% confidence

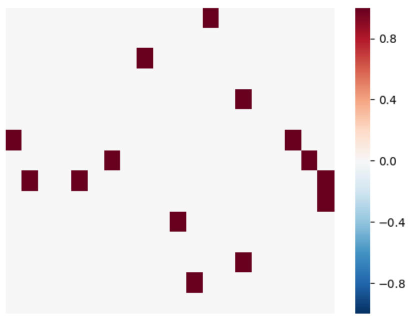
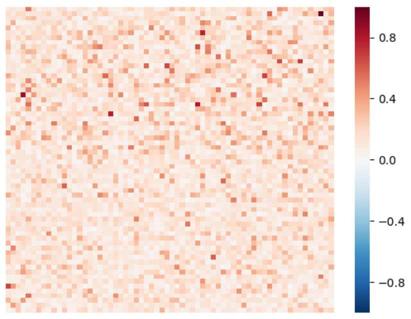
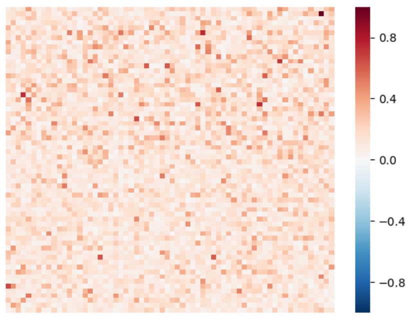


Figure C.5: BiLSTM-max vector embedding of singular vs. plural

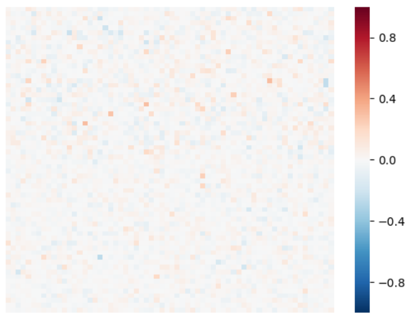
(a) Mean of 'Singular' sentences (normalized)



(b) Mean of 'Plural' sentences (normalized)



(c) Difference between 'singular' and 'plural' sentences



(d) Significant differences at 5% confidence

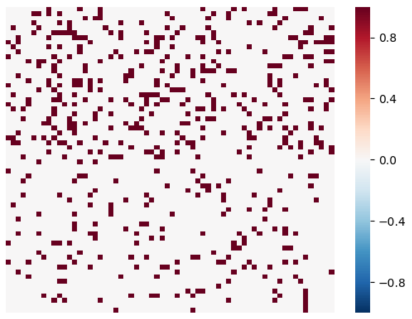
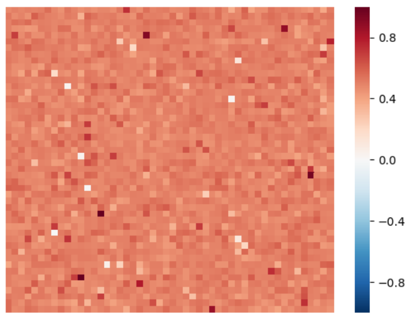
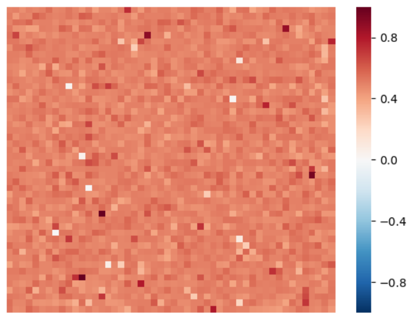


Figure C.6: Skip-thought vector embedding of singular vs. plural

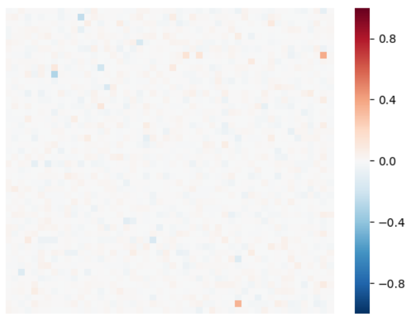
(a) Mean of 'Singular' sentences (normalized)



(b) Mean of 'Plural' sentences (normalized)



(c) Difference between 'singular' and 'plural' sentences



(d) Significant differences at 5% confidence

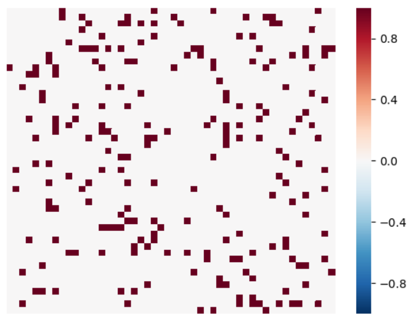
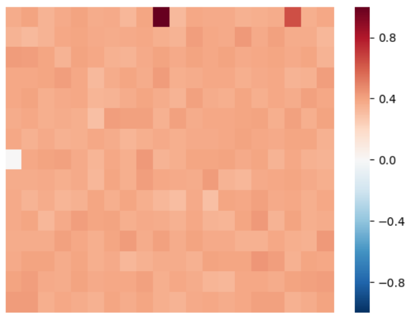
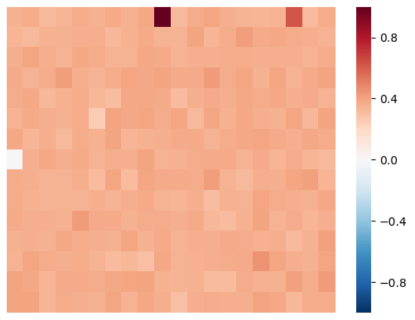


Figure C.7: CBOW vector embedding of singular vs. plural

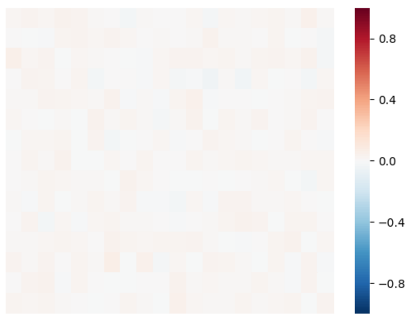
(a) Mean of 'Singular' sentences (normalized)



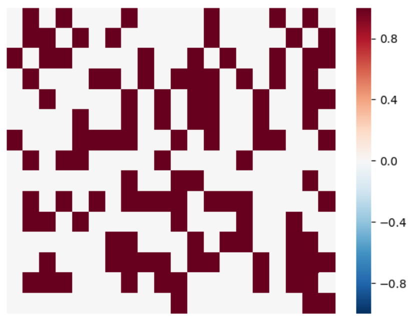
(b) Mean of 'Plural' sentences (normalized)



(c) Difference between 'singular' and 'plural' sentences



(d) Significant differences at 5% confidence



Appendix D

Colophon

The code used in this thesis is mainly based on Python 3.5 and Tensorflow 1.0. It can be found in the following GitHub repositories:

1. Skip-thought model (Section 3.2): https://github.com/jonvet/skipthought
2. BiLSTM-max model (Section 3.3): https://github.com/jonvet/InferSent
3. Classification and visualization (Chapter 4): https://github.com/jonvet/thesis

The SentEval suite used in Section 3.4 is based on Python 2.7 and PyTorch ¿=0.2 and can be found at https://github.com/facebookresearch/SentEval

Bibliography

Adi, Y., Kermany, E., Belinkov, Y., Lavi, O., and Goldberg, Y. (2016). Fine-grained analysis of sentence embeddings using auxiliary prediction tasks. *CoRR*, abs/1608.04207.

Agirre, E., Banea, C., Cardie, C., Cer, D. M., Diab, M. T., Gonzalez-Agirre, A., Guo, W., Mihalcea, R., Rigau, G., and Wiebe, J. (2014). Semeval-2014 task 10: Multilingual semantic textual similarity. In *SemEval@COLING*.

Andrews, M., Vigliocco, G., and Vinson, D. (2009). Integrating experiential and distributional data to learn semantic representations. *Psychological Review*, pages 463–498.

Ba, J., Kiros, R., and Hinton, G. E. (2016). Layer normalization. *CoRR*, abs/1607.06450.

Barber, D. (2012). *Bayesian Reasoning and Machine Learning*. Cambridge University Press.

Bayes, M. and Price, M. (1763). An essay towards solving a problem in the doctrine of chances. by the late rev. mr. bayes, f. r. s. communicated by mr. price, in a letter to john canton, a. m. f. r. s. *Philosophical Transactions (1683-1775)*, 53:370–418.

Bergstra, J. and Bengio, Y. (2012). Random search for hyper-parameter optimization. *J. Mach. Learn. Res.*, 13:281–305.

Blacoe, W. and Lapata, M. (2012). *A Comparison of Vector-based Representations for Semantic Composition*, pages 546–556. Association for Computational Linguistics.

Blei, D. M., Ng, A. Y., and Jordan, M. I. (2003). Latent dirichlet allocation. *Journal of Machine Learning Research*, 3:993–1022.

Bowman, S. R., Angeli, G., Potts, C., and Manning, C. D. (2015). A large annotated corpus for learning natural language inference. In *Proceedings of the 2015 Conference on Empirical Methods in Natural Language Processing (EMNLP)*. Association for Computational Linguistics.

Brants, T., Popat, A. C., Xu, P., Och, F. J., and Dean, J. (2007). Large language models in machine translation. In *In Proceedings of the Joint Conference on Empirical Methods in Natural Language Processing and Computational Natural Language Learning*, pages 858–867.

Chen, D. and Manning, C. D. (2014). A fast and accurate dependency parser using neural networks. Proceedings of EMNLP 2014.

Chung, J., Gülçehre, Ç., Cho, K., and Bengio, Y. (2014). Empirical evaluation of gated recurrent neural networks on sequence modeling. *CoRR*, abs/1412.3555.

Conneau, A., Kiela, D., Schwenk, H., Barrault, L., and Bordes, A. (2017). Supervised learning of universal sentence representations from natural language inference data. *CoRR*, abs/1705.02364.

Cortes, C. and Vapnik, V. (1995). Support-vector networks. In *Machine Learning*, pages 273–297.

Cover, T. and Hart, P. (1967). Nearest neighbor pattern classification. *IEEE Transactions on Information Theory*, 13(1):21–27.

Cox, D. R. (1958). The regression analysis of binary sequences. *Journal of the Royal Statistical Society. Series B (Methodological)*, 20(2):215–242.

Coyotl-Morales, R. M., Villaseñor-Pineda, L., Montes-y Gómez, M., and Rosso, P. (2006). *Authorship Attribution Using Word Sequences*, pages 844–853. Springer Berlin Heidelberg, Berlin, Heidelberg.

Cybenko, G. (1992). Approximation by superpositions of a sigmoidal function. *Mathematics of Control, Signals and Systems*, 5(4):455–455.

Deerwester, S., Dumais, S. T., Furnas, G. W., Landauer, T. K., and Harshman, R. (1990). Indexing by latent semantic analysis. *Journal of the American Society for Information Science*, 41(6):391–407.

Dolan, B., Quirk, C., and Brockett, C. (2004). Unsupervised construction of large paraphrase corpora: Exploiting massively parallel news sources. In *In Proceedings of 20th International Conference on Computational Linguistics (COLING*.

Domingos, P. (2015). *he Master Algorithm: How the Quest for the Ultimate Learning Machine Will Remake Our World*. New York: Basic Books.

Duchi, J., Hazan, E., and Singer, Y. (2010). Y.: Adaptive subgradient methods for online learning and stochastic optimization. Technical report.

Greff, K., Srivastava, R. K., Koutník, J., Steunebrink, B. R., and Schmidhuber, J. (2015). LSTM: A search space odyssey. *CoRR*, abs/1503.04069.

He, H., Gimpel, K., and Lin, J. J. (2015). Multi-perspective sentence similarity modeling with convolutional neural networks. In *EMNLP*.

Hill, F., Cho, K., and Korhonen, A. (2016). Learning distributed representations of sentences from unlabelled data. *CoRR*, abs/1602.03483.

Hinton, G. E., McClelland, J. L., and Rumelhart, D. E. (1986). Parallel distributed processing: Explorations in the microstructure of cognition, vol. 1. chapter Distributed Representations, pages 77–109. MIT Press, Cambridge, MA, USA.

Hinton, G. E., Osindero, S., and Teh, Y.-W. (2006). A fast learning algorithm for deep belief nets. *Neural computation*, 18(7):1527–1554.

Hinton, G. E. and Shallice, T. (1991). Lesioning an attractor network: Investigations of acquired dyslexia. *Psychological Review*, 98(1):74–95.

Hochreiter, S. and Schmidhuber, J. (1997). Long short-term memory. *Neural Comput.*, 9(8):1735–1780.

Hu, M. and Liu, B. (2004). Mining and summarizing customer reviews. In *Proceedings of the Tenth ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, KDD '04, pages 168–177, New York, NY, USA. ACM.

Kalchbrenner, N., Grefenstette, E., and Blunsom, P. (2014). A convolutional neural network for modelling sentences. *CoRR*, abs/1404.2188.

Katz, S. M. (1987). Estimation of probabilities from sparse data for the language model component of a speech recognizer. In *IEEE Transactions on Acoustics, Speech and Signal Processing*, pages 400–401.

Kim, Y. (2014). Convolutional neural networks for sentence classification. *CoRR*, abs/1408.5882.

Kingma, D. P. and Ba, J. (2014). Adam: A method for stochastic optimization. *CoRR*, abs/1412.6980.

Kiros, R., Zhu, Y., Salakhutdinov, R., Zemel, R. S., Torralba, A., Urtasun, R., and Fidler, S. (2015). Skip-thought vectors. *CoRR*, abs/1506.06726.

Kneser, R. and Ney, H. (1993). Improved clustering techniques for class-based statistical language modelling. In *Third European Conference on Speech Communication and Technology, EUROSPEECH 1993, Berlin, Germany, September 22-25, 1993*.

Kullback, S. and Leibler, R. A. (1951). On information and sufficiency. *Ann. Math. Statist.*, 22(1):79–86.

Le, Q. and Mikolov, T. (2014). Distributed representations of sentences and documents. In Xing, E. P. and Jebara, T., editors, *Proceedings of the 31st International Conference on Machine Learning*, volume 32 of *Proceedings of Machine Learning Research*, pages 1188–1196, Bejing, China. PMLR.

Legendre, A.-M. (1805). Appendice sur la methode des moindres quarres , annexe a l'ouvrage nouvelles methodes pour la determination des orbites des cometes. pages 72–80.

Linnainmaa, S. (1976). Taylor expansion of the accumulated rounding error. *BIT Numerical Mathematics*, 16(2):146–160.

Linzen, T., Dupoux, E., and Goldberg, Y. (2016). Assessing the ability of lstms to learn syntax-sensitive dependencies. *CoRR*, abs/1611.01368.

Liu, P., Qiu, X., Chen, X., Wu, S., and Huang, X. (2015). Multi-timescale long short-term memory neural network for modelling sentences and documents. In *EMNLP*.

Mikolov, T., Chen, K., Corrado, G., and Dean, J. (2013a). Efficient estimation of word representations in vector space. *CoRR*, abs/1301.3781.

Mikolov, T., Le, Q. V., and Sutskever, I. (2013b). Exploiting similarities among languages for machine translation. *CoRR*, abs/1309.4168.

Mikolov, T., Yih, W.-t., and Zweig, G. (2013c). Linguistic regularities in continuous space word representations. In *HLT-NAACL*, pages 746–751.

Mitchell, J. and Lapata, M. (2010). Composition in distributional models of semantics. *Cognitive Science*, 34(8):1388–1429.

Nair, V. and Hinton, G. E. (2010). Rectified linear units improve restricted boltzmann machines. In Frnkranz, J. and Joachims, T., editors, *Proceedings of the 27th International Conference on Machine Learning (ICML-10)*, pages 807–814. Omnipress.

Nesterov, Y. (1983). A method of solving a convex programming problem with convergence rate o (1/k2). In *Soviet Mathematics Doklady*, volume 27, pages 372–376.

Pang, B. and Lee, L. (2004). A sentimental education: Sentiment analysis using subjectivity summarization based on minimum cuts. In *In Proceedings of the ACL*, pages 271–278.

Pang, B. and Lee, L. (2005). Seeing stars: Exploiting class relationships for sentiment categorization with respect to rating scales. In *In Proc. 43st ACL*, pages 115–124.

Paperno, D., Pham, N. T., and Baroni, M. (2014). A practical and linguistically-motivated approach to compositional distributional semantics. In *Proceedings of the 52nd Annual Meeting of the Association for Computational Linguistics, ACL 2014, June 22-27, 2014, Baltimore, MD, USA, Volume 1: Long Papers*, pages 90–99.

Papert, M. M. S. and Minsky, M. (1969). Perceptrons: an introduction to computational geometry. *Expanded Edition*.

Papineni, K., Roukos, S., Ward, T., and jing Zhu, W. (2002). Bleu: a method for automatic evaluation of machine translation. pages 311–318.

Park, D. H., Hendricks, L. A., Akata, Z., Schiele, B., Darrell, T., and Rohrbach, M. (2016). Attentive explanations: Justifying decisions and pointing to the evidence. *CoRR*, abs/1612.04757.

Pearson, K. (1895). Notes on regression and inheritance in the case of two parents. In *Proceedings of the Royal Society of London, Philosophical Transactions of the Royal Society*, volume 58, pages 240–242. Royal Society of London.

Peng, F., Schuurmans, D., and Wang, S. (2004). Augmenting naive bayes classifiers with statistical language models. *Information Retrieval*, 7(3):317–345.

Pennington, J., Socher, R., and Manning, C. D. (2014). Glove: Global vectors for word representation. In *Empirical Methods in Natural Language Processing (EMNLP)*, pages 1532–1543.

Qian, N. (1999). On the momentum term in gradient descent learning algorithms. *Neural Networks*, 12(1):145 – 151.

Rijsbergen, C. J. V. (1979). *Information Retrieval*. Butterworth-Heinemann, Newton, MA, USA, 2nd edition.

Rocktäschel, T., Grefenstette, E., Hermann, K. M., Kociský, T., and Blunsom, P. (2015). Reasoning about entailment with neural attention. *CoRR*, abs/1509.06664.

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

Rumelhart, D. E., Hinton, G. E., and Williams, R. J. (1986). Learning representations by back-propagating errors. *Nature*, 323:533–536.

Russakovsky, O., Deng, J., Su, H., Krause, J., Satheesh, S., Ma, S., Huang, Z., Karpathy, A., Khosla, A., Bernstein, M., Berg, A. C., and Fei-Fei, L. (2015). ImageNet Large Scale Visual Recognition Challenge. *International Journal of Computer Vision (IJCV)*, 115(3):211–252.

Sahami, M., Dumais, S., Heckerman, D., and Horvitz, E. (1998). A bayesian approach to filtering junk e-mail. Technical report, PAPERS FROM THE 1998 WORKSHOP, AAAI.

Sanderson, C. and Guenter, S. (2006). Short text authorship attribution via sequence kernels, markov chains and author unmasking: An investigation. In *Proceedings of the 2006 Conference on Empirical Methods in Natural Language Processing*, EMNLP '06, pages 482–491, Stroudsburg, PA, USA. Association for Computational Linguistics.

Saxe, A. M., McClelland, J. L., and Ganguli, S. (2013). Exact solutions to the nonlinear dynamics of learning in deep linear neural networks. *CoRR*, abs/1312.6120.

Schuster, M. and Paliwal, K. K. (1997). Bidirectional recurrent neural networks. *IEEE Trans. Signal Processing*, 45(11):2673–2681.

Sebastiani, F. (2002). Machine learning in automated text categorization. *ACM Comput. Surv.*, 34(1):1–47.

Sidorov, G., Velasquez, F., Stamatatos, E., Gelbukh, E., and Chanona-hernndez, L. (2013). L.: Syntactic dependency-based n-grams as classification features. In *MICAI 2012, Part II. LNCS (LNAI*, pages 1–11. Springer.

Silver, D., Huang, A., Maddison, C. J., Guez, A., Sifre, L., van den Driessche, G., Schrittwieser, J., Antonoglou, I., Panneershelvam, V., Lanctot, M., Dieleman, S., Grewe, D., Nham, J., Kalchbrenner, N., Sutskever, I., Lillicrap, T., Leach, M., Kavukcuoglu, K., Graepel, T., and Hassabis, D. (2016). Mastering the game of go with deep neural networks and tree search. *Nature*, 529:484–503.

Socher, R., Huang, E. H., Pennin, J., Manning, C. D., and Ng, A. Y. (2011a). Dynamic pooling and unfolding recursive autoencoders for paraphrase detection. In Shawe-taylor, J., Zemel, R., Bartlett, P., Pereira, F., and Weinberger, K., editors, *Advances in Neural Information Processing Systems 24*, pages 801–809.

Socher, R., Pennington, J., Huang, E. H., Ng, A. Y., and Manning, C. D. (2011b). Semi-Supervised Recursive Autoencoders for Predicting Sentiment Distributions. In *Proceedings of the 2011 Conference on Empirical Methods in Natural Language Processing (EMNLP)*.

Solomonoff, R. J. (1964). A formal theory of inductive inference, part l. *I and IIquot;; Information and Control*, 7:1–22.

Spearman, C. (1904). The proof and measurement of association between two things. *American Journal of Psychology*, 15:88–103.

Srivastava, N., Hinton, G., Krizhevsky, A., Sutskever, I., and Salakhutdinov, R. (2014). Dropout: A simple way to prevent neural networks from overfitting. *Journal of Machine Learning Research*, 15:1929–1958.

Stamatatos, E. (2009). A Survey of Modern Authorship Attribution Methods. *Journal of the American Society for Information Science and Technology*, 60(3):538–556.

Stenetorp, P., Pyysalo, S., Topi, G., Ohta, T., Sophia, A., and Tsujii, J. (2012). brat: a web-based tool for nlp-assisted text annotation. In *Proceedings of the Demonstrations Session at EACL*.

Tai, K. S., Socher, R., and Manning, C. D. (2015a). Improved semantic representations from tree-structured long short-term memory networks. *CoRR*, abs/1503.00075.

Tai, K. S., Socher, R., and Manning, C. D. (2015b). Improved semantic representations from tree-structured long short-term memory networks. In *IN PROC. ACL*.

Tikhonov, A. N. and Arsenin, V. I. (1977). *Solutions of ill-posed problems / Andrey N. Tikhonov and Vasiliy Y. Arsenin ; translation editor, Fritz John*. Winston ; distributed solely by Halsted Press Washington : New York.

Turing, A. M. (1950). Computing machinery and intelligence. *Mind*, 59(236):433–460.

Voorhees, E. M. (2002). Overview of the trec 2002 question answering track. In *In Proceedings of the Eleventh Text REtrieval Conference (TREC*, pages 115–123.

Wang, Z., Hamza, W., and Florian, R. (2017). Bilateral multi-perspective matching for natural language sentences. *CoRR*, abs/1702.03814.

Welin, C. W. (1979). Scripts, plans, goals and understanding, an inquiry into human knowledge structures. *Journal of Pragmatics*, 3(2):211 – 217.

Wiebe, J., Wilson, T., and Cardie, C. (2005). Annotating expressions of opinions and emotions in language. *Language Resources and Evaluation*, 39:165–210.

Wieting, J., Bansal, M., Gimpel, K., and Livescu, K. (2015). Towards universal paraphrastic sentence embeddings. *CoRR*, abs/1511.08198.

Winograd, T. (1972). Procedures as a representation for data in a computer program for understanding natural language. *Cognitive Psychology*, 3.

Young, P., Lai, A., Hodosh, M., and Hockenmaier, J. (2014). From image descriptions to visual denotations: New similarity metrics for semantic inference over event descriptions. Technical report.

Zeiler, M. D. (2012). ADADELTA: an adaptive learning rate method. *CoRR*, abs/1212.5701.

Zhao, H., Lu, Z., and Poupart, P. (2015). Self-adaptive hierarchical sentence model. *CoRR*, abs/1504.05070.

Zhu, Y., Kiros, R., Zemel, R., Salakhutdinov, R., Urtasun, R., Torralba, A., and Fidler, S. (2015). Aligning books and movies: Towards story-like visual explanations by watching movies and reading books. In *arXiv preprint arXiv:1506.06724*.