Technical University of Denmark

Text entailment classification

Deep Learning

Authors:
Andreas K. Salk - **s144550**
Frederik Toftegaard - **s124265**

November 1, 2018

Contents

1 Introduction

Natural language processing has been studied in numerous articles and have implementations in Machine translation, Question answering etc.. In Natural language processing Deep Learning is widely used to answer these tasks. This report will try to solve one of the problems being worked on with words in deep learning, the textual entailment. With the textual entailment (TE) the goal is to take 2 sentences as an input and predict if the facts in the first sentence implies the facts in the second sentence. There are 3 different classifications when dealing with TE, the facts can be contradicting, neutral or an entailment. This article aims to create an state-of-art classifier that works with sequential data in the form of word vectors. The output of the experiment is a 3-way softmax classifier, classifying whether the data is the contradicting, neutral or an entailment.

1.1 Related work

There have been several articles working with text classification, Bowman et al. [2015] works entirely with TE and implements a bag-of-words and a Long-short-term-memory Recurrent neural network. They test both a lexicalized model and a unlexicalized model, and end with a model that has just below 80% precision on 1 million test pairs.

Conneau et al. [2018] studies how universal sentence representations with supervised data consistently outperform unsupervised methods like SkipThougth. The models tested is TE classifiers. Several models are tested with the top performer being the Bidirectional Long-short-term-memory predicting at 85% with a 4096 embedding size.

McCann et al. [2018] studies several NLP problems including TE. The model uses a Long-short-term-memory encoder and adds context vectors. The article shows that adding these context vectors on unsupervised word vectors improves perfomance. The model at best predicts at 88% on the Stanford natural language interference dataset, which is the best documented results and therefore state-of art.

1.2 Experimental setup

Throughout this project the data set used for testing and training will be the SNLI (Stanford Natural Language Inference) corpus and the Sentiment analysis (SST) provided by torchtext. The SNLI corpus is a large collection of sentence pairs that are labelled for contradiction, neutral and entailment. How this data set was developed is explained in [Bowman et al., 2015]. The training of the model will be done in Python with the PyTorch package.

2 Deep Learning models

2.1 Word2Vec - Skip gram model

The skip gram model is a neural network architecture for Word2Vec. Word2Vec uses a trick, where you train a simple neural network with a single hidden layer to perform a task, however instead of using the network for the task it was trained on, it on it uses the weights of the hidden layer. The weights for the word vectors are what this model is trying to learn [McCormick, 2016]. The task the network is performing is learning the words of our vocabulary nearby of the word we have chosen (the amount of nearby words can be changed). So in the end you will get an output of probabilities to how likely

it is you will find a vocabulary word nearby our input word. Mathematically the objective of the Skip gram model is to maximize the log probability, given a sequence of training words $w_1, w_2, w_3, ..., w_T$:

$$\frac{1}{T} \cdot \sum_{t=1}^{T} \sum_{-c<j<c, j\neq 0} logp(w_{t+j}|w_t) \tag{1}$$

Where c is the size of the training context. The basic Skip gram formulation says that $p(w_{t+j}|w_t)$ is defined by the softmax function:

$$p(w_O|w_I) = \frac{exp({v'_{w_O}}^T v_{w_I})}{\sum_{w=1}^{W} exp({v'_w}^T v_{w_I})} \tag{2}$$

Where v_w and v'_w are the vector representations of W, both input and output. W is the size of the vocabulary. These formulas can be found in [Mikolov et al.].

2.2 Bag-Of-Words

The bag of words (BoW) method is used to extract features from text and use it in modelling. The output of the BoW is the occurrences of words from a vocabulary in a text. It completely ignores the order and structure of the words, hence the name bag of words. The idea behind BoW is that if 2 (or more) sentences has the same occurrences of words, the sentences are similar. However a BoW can be more complex than just counting the occurrences of words. The complexity can change depending on the design of the vocabulary that is used in the method, or how you score the occurrences of the known words. The Skip gram and BoW can seem a bit similar, however there are some differences illustrated below:

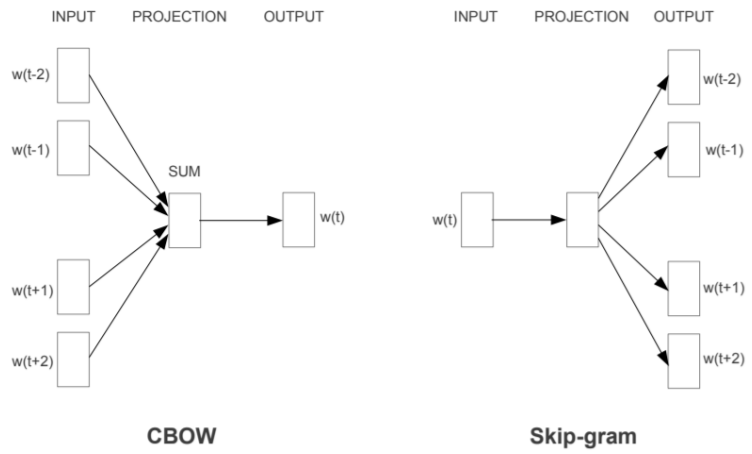

Figure 1: BoW and SG from [Chen et al.]

The input of BoW is 1 or more sentences as vectors, and the output is the occurrences of these words, if they are in the given vocabulary. While the Skip-gram model takes the input as a 1-hot vector, and the output is the weights of the surrounding words in the sentences.

The structure of the neural network classification that will be done using the BoW method will follow the architecture presented in [Bowman et al., 2015, Figure 3]. The architecture of the neural network classifier has a stack of 3 200d tanh layers, where the bottom is taking the 2 sentences as input, and the top layer is feeding the 3-way softmax classifier, that gives the output of the neural network.

2.3 Long-Short-Term-Memory Recurrent Neural Network

A Long-Short-Term-Memory Recurrent Neural Network (LSTM RNN) is done as in [Bowman et al., 2015]. The LSTM RNN from [Bowman et al., 2015] follows the mathematical model described in [Hochreiter and Schmidhuber, 1997]. The LSTM RNN is build on the BoW model, but with a recurrence in layers as described in [Nielsen].

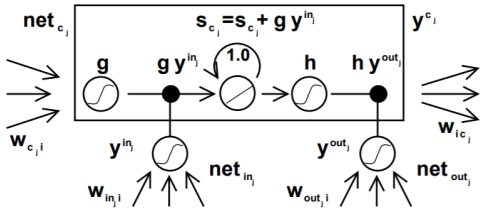

Figure 2: Architecture of the LSTM Hochreiter and Schmidhuber [1997]

The LSTM in the model follows the architecture of Figure 2. Going from a FFNN in the BoW to the LSTM RNN the model increases in robustness but also computational time. The LSTM in itself has a recurrant loop, and uses real-time recurrent learning (RTRL) with backwards propagation [Graves] which will properly handle the dynamics of the input and output gate of the LSTM. The architecture of the LSTM is explained in [Bowman et al., 2015], and take a 300d input layer (initialized with GloVe [Pennington et al., 2014]) with an additional tanh hidden layer after the LSTM RNN. The model uses L2 regularization and added dropout with a fixed dropout rate.

2.4 Biattentive Classification Network

The last model this project will examine is the biattentive classification network (BCN). This model will be built as an extension of the LSTM RNN model described previously. This section will follow the model described in [McCann et al., 2018]. Instead of feeding the BCN with standard vector with sequences of words, the input fed into the model is a GLoVe vector of words. The output from LSTM model is treated as context vectors. Therefore for classification for each input sentence w the model concatenate each vector with in GLoVe(w) with the corresponding context vector, CoVe(w).

$$\widetilde{w} = [GloVe(w); CoVe(w)] \tag{3}$$

The BCN can be used to test how well the CoVe transfers to other tasks. The BCN takes 2 converted input sentences ($w^x, w^y \rightarrow \widetilde{w^x}, \widetilde{w^y}$) and feeds it to a function f that applies a feedforward network with ReLU activation. The network is then continued by a bidirectional LSTM (BiLSTM) process that results in task specific representations, written as:

$$x = biLSTM(f(\widetilde{w}^x)), \quad y = biLSTM(f(\widetilde{w}^y)) \tag{4}$$

These are aggregated along a time axis, and this gives us the matrices X and Y. To find the representations that are interdependent the biattention mechanism is used. This biattention computes a affinity matrix $A = X \cdot Y^T$. The attention weights are extracted in a column wise normalization and then used to calculate the context summaries, C:

$$A_x = softmax(A) \qquad A_y = softmax(A^T) \tag{5}$$

$$C_x = A_x^T \cdot X \qquad C_y = A_y^T \cdot Y \tag{6}$$

This information is then integrated into the representation for each sentence with 2 seperate 1 layer biLSTM's. They operate on the link between the original representations, the differences in the context summaries and element-wise products between the original and context summaries. Mathematically this can be written as:

$$X_{|y} = \text{biLSTM}([X; X - C_y; X \odot C_y]), \quad Y_{|x} = \text{biLSTM}([X; X - C_x; Y \odot C_x]) \tag{7}$$

These outputs are then aggregated by max, mean, min and self-attentive pooling along a time dimension. This was found helpful by [McCann et al., 2018]. The different pools presents different perspective on the sentences. The self-attentive pool computes weights for each step of the sentence like:

$$\beta_x = softmax(X_{|y} \cdot v_1 + d_1), \quad \beta_y = softmax(Y_{|x} \cdot v_2 + d_2) \tag{8}$$

These weights are used to get the weighted sum of each sentence:

$$x_{self} = X_{|y}^T \cdot_x \quad y_{self} = Y_{|x}^T \cdot_y \tag{9}$$

The representations of the pooled input are joined with the representation of all the input, to get one output:

$$x_{pool} = [max(X_{|y}); mean(X_{|y}); min(X_{|y}); x_{self}] \tag{10}$$

$$y_{pool} = [max(Y_{|x}]; mean(Y_{|x}); min(Y_{|x}); y_{self}] \tag{11}$$

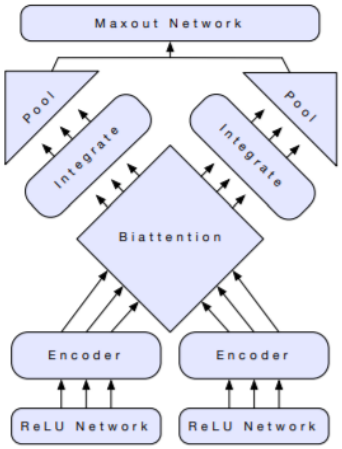

Figure 3: Architecture of the BCN network from [McCann et al., 2018]

This final representation is then feeded to a three-layer, batch-normalized maxout network, that produces the probability distribution of the 3 different classes that our project aims to predict.

References

Samuel Bowman, Christopher Potts, Gabor Angeli, and Christopher D. Malling. A large annotated corpus for learning natural language inference. URL: https://nlp.stanford.edu/pubs/snli_paper.pdf, 2015. Last read: 29-10-2018.

Kai Chen, Thomas Mikolov, Greg Corrado, and Jeffrey Dean. Efficient estimation of word representations in vector space. URL: https://arxiv.org/pdf/1301.3781.pdf. Last read: 29-10-2018.

Alexis Conneau, Douwe Kiela, Holger Schwenk adn Loïc Barrault, and Antoine Bordes. Supervised learning of universal sentence representations from natural language inference data. URL: https://arxiv.org/pdf/1705.02364.pdf, 2018. Last read: 31-10-2018.

Alex Graves. Supervised sequence labelling with recurrent neural networks. pages 20–39.

Sepp Hochreiter and Jurgen Schmidhuber. Long short-term memory. *Neural computation*, (9):1735–1780, 1997.

Bryan McCann, James Bradbury, Caiming Xiong, and Richard Socher. Learned in translation: Contextualized word vectors. URL: https://arxiv.org/pdf/1708.00107.pdf, 2018. Last read: 31-10-2018.

Chris McCormick. Word2vec tutorial - the skip-gram model. URL: http://mccormickml.com/2016/04/19/word2vec-tutorial-the-skip-gram-model, 2016. Last read: 29-10-2018.

Thomas Mikolov, Ilya Sutskever, Kai Chen, Greg Corrado, and Jeffrey Dean. Distributed representations of words and phrases and their compositionality. URL: http://papers.nips.cc/paper/5021-distributed-representations-of-words-and-phrases-and-their-compositionality.pdf. Last read: 29-10-2018.

Michael Nielsen. Neural networks and deep learning. URL: http://neuralnetworksanddeeplearning.com/chap6.html#other_approaches_to_deep_neural_nets. Last read: 31-10-2018.

Jeffrey Pennington, Richard Socher, and Christopher D. Manning. Glove: Global vectors for word representation. 2014.