Text entailment classification

Valle, Damian
s182091@student.dtu.dk

Barkat, Sacha
s182045@student.dtu.dk

December 2018

Abstract

Natural Language Inference (NLI) is a main task in the scope of Natural Language Processing that has been around for more than a decade but in the last years has become more relevant due to the new big and high quality datasets. This datasets make it now feasible to train models such as deep neural networks which typically require a large amount of training data. The applications of NLI include question answering, semantic search, automatic text summarization, etc...
In this paper we train different models, compare them and try to interpret the results.

1 Introduction

The task discussed in this paper is entailment classification, which is a directional relation between text fragments that holds whenever the truth of one text fragment follows from another text. In this sense, $p \rightarrow h$ read as p *entails* h means that a human reading p would infer that h is most likely true.

In order to train the models that will execute the task, we rely on the **Stanford Natural Language Inference** dataset (SNLI), this corpus contains around 570K sentence pairs with three labels: *entailment, contradiction* and *neutral*.

Here we show a few examples of pairs of sentences picked from the dataset and it's corresponding entailments:

Premise: A soccer game with multiple males playing
Hypothesis: Some men are playing a sport
Judgement: Entailment

Premise: A man inspects the uniform of a figure in some East Asian country
Hypothesis: The man is sleeping
Judgement: Contradiction

Premise: An older and younger man smiling
Hypothesis: Two men are smiling and laughing at the cats playing on the floor
Judgement: Neutral

Our model aims to estimate the probability function of the entailments given two sentences as an input. We denote that probability $p(y, x)$ where y is the three class output and x consist of the two input sentences x_p and x_h. We want to find a function f that estimates that probability, that function will have a set of parameters θ that require to be learned.

2 Related Work

In order to implement three different variants of models that can solve the classification challenge we need to look at several papers: Bowman et al.[1] for the first two models (BoW and LSTM RNN) and McCann et al.[3] for the third model (BCN).

1. The first approach used by, Bill McCartney [9] is the bag of words achieving a surprisingly good result with it (75.3% test accuracy). The architecture of the model can be seen in Figure 3.

2. Building a model with an LSTM RNN Bowman et al.[1] managed to improve the baseline and also, compared to a classic RNN, the paper showed that LSTM has a more robust ability to learn long-term dependencies.

3. McCann et al.[3] managed to achieve state of the art performance on a wide variety of common NLP tasks such as sentiment analysis (SST, IMDb), question classification (TREC) and, most importantly for the purpose of this paper, entailment (SNLI) thanks to it's context vectors (CoVe). Typically in NLP we only initialize the lowest layer of deep models with pretrained word vectors, but McCann et al.[3] propose a deep LSTM encoder from an attentional sequence-to-sequence model to contextualize vectors. In all cases, models that used CoVe from their best pretrained MT-LSTM performed better than baselines that used random word vector initialization.

4. The current state of the art for this particular tasks has been achieved by Seohoon et. al.[11] using Densely-Connected Recurrent and Co-Attentive Network Ensemble with a 90.1% test accuracy.

3 Methods

3.1 Word vector representations

Our models cannot work with raw words as inputs, we need to represent them in a numerical way. The first idea that comes to mind is to use one-hot encoding over a dictionary of words but we quickly realize that this aproach is too sparse and high-dimensional. In order to get dense vectors whose dimensions are orders of magnitude smaller we use word vector representations also known as word vector embeddings. The projection to a lower dimensional space is done in such a way that words with similar meaning end up close to each other in that space. To show the advantages of such representations we compare them to the one hot encoding in Figure 1 and we show how they capture the semantics of the words and the relation to each other in Figure 2. Jeffrey Pennington et. al.[6] trained the **GloVe** model that was able to accomplish this behaviour and it's the one we are using in this paper. We'll use a matrix of dimensions $N \times d$ as a lookup table where N is the size of the vocabulary and d the output space dimension.

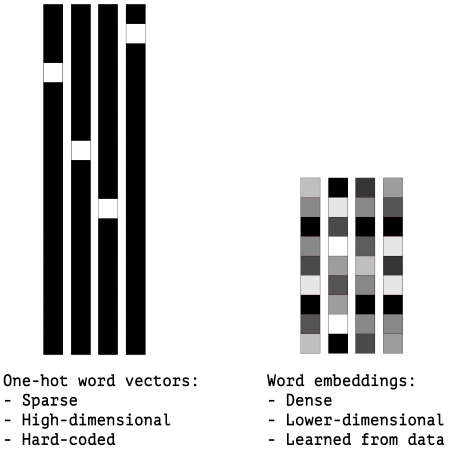

Figure 1: Embeddings comparison ??

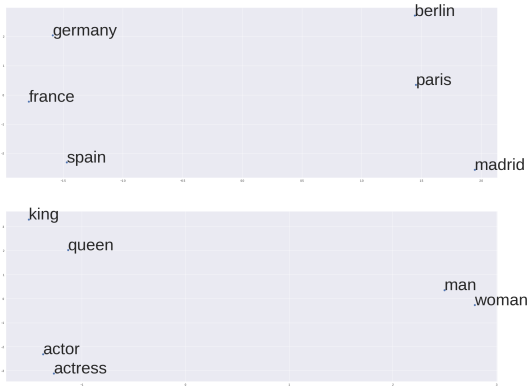

Figure 2: GloVe examples ??

3.2 Bag of Words (BoW)

In the Bag of Words model, we take all the embeddings of each word in a sentence and we average them. We use that as our latent representation of the sentence, in this case we have two latent representations, one for the premise and one for the hypothesis and we feed both of them to the classifier shown in Figure 3.

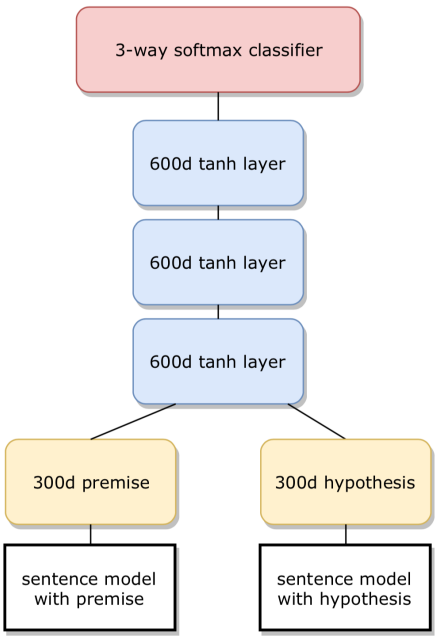

Figure 3: Classifier architecture

3.3 LSTM RNN

The problem with the Bag of Words model is that it doesn't take word order into account so here we use recurrent neural networks. We run the embeddings of the words from the premise and the hypothesis through two separate LSTM nets one for each sentence. We then use the last hidden state of each sentence as its latent representation. Those representations are the inputs to the same classifier network as the Bag of Words model shown in Figure 3.

3.4 Biattentive Classification Network (BCN)

In order to describe the BCN model, first we have to define how the Context Vectors (CoVe) are defined. Let w be a sequence of words and GloVe(w) the corresponding sequence of word vectors produced by the GloVe model, we define

$$\text{CoVe}(w) = \text{MT-LSTM}(\text{GloVe}(w)) \tag{1}$$

In our case, our output vector will be the concatenation of the two outputs

$$\tilde{w} = [\text{GloVe}(w); \text{CoVe}(w)] \tag{2}$$

we can use this output to provide context for the BCN.

The model described here is a general Biattentive Classification Network, that means it's designed to handle both single-sentence and two-sentence classification tasks. In the case of the single-sentence task we duplicate it to input both to the model.

Given our input sequences w^x and w^y, we compute their sequences of vectors $\tilde{w}^x$ and $\tilde{w}^y$. We use those as an input to a feedforward neural network modelled by f with ReLU activation to each of them and feed that to a bidirectional LSTM.

$$x = \text{biLSTM}(f(\tilde{w}^x)) \quad y = \text{biLSTM}(f(\tilde{w}^y)) \tag{3}$$

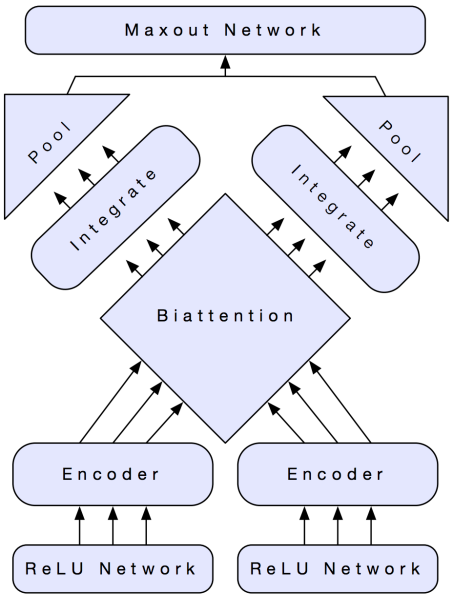

Figure 4: Topology of the BCN

We stack these sequences along the time axix to get matrixes X and Y. To compute interdependent representations we use a biattention mechanism that first computes an affinity matrix $A = XY^T$, then extracts attention weights with column-wise normalization:

$$\begin{aligned} A_x &= softmax(A) \\ A_y &= softmax(A^T) \end{aligned} \tag{4}$$

Next, it uses context summaries

$$C_x = A_x^T X \qquad C_y = A_y^T Y \tag{5}$$

to condition each sequence on the other.

We integrate the conditioning information into our representations for each sequence with two separate one-layer, bidirectional LSTMs

$$\begin{aligned} X_{|y} &= biLSTM([X; X - C_y; X \odot C_y]) \\ Y_{|x} &= biLSTM([Y; Y - C_x; Y \odot C_x]) \end{aligned} \tag{6}$$

The outputs of the bidirectional LSTMs are aggregated by pooling along the time dimension. The self-attentive pooling computes weights for each time step of the sequence.

$$\beta_x = softmax(X_{|y} v_1 + d_1) \beta_y = softmax(Y_{|x} v_2 + d_2) \tag{7}$$

and uses these weights to get weighted summations of each sequence:

$$x_{self} = X_{|y}^T \beta_x y_{self} = Y_{|x}^T \beta_y \tag{8}$$

The pooled representations are combined to get one joined representation for all inputs.

$$\begin{aligned} x_{pool} &= [max(X_{|y}); min(X_{|y}); mean(X_{|y}); x_{self}] \\ y_{pool} &= [max(Y_{|x}); min(Y_{|x}); mean(Y_{|x}); y_{self}] \end{aligned} \tag{9}$$

We feed this joined representation through a three-layer maxout network. Ian J. Goodfellow et al.[10].

4 Results

Both the BoW and LSTM models managed to get above 70% test accuracy, the evolution of the accuracy over the training can be seen in Figure 5 and Figure 6. Unfortunately we couldn't make the biattention model work properly since it only gave random outputs.

We found the best hyperparameters for both the models to be:

optimizer	Adam
epochs	10
batch size	512
learning rate	0.001
weight decay	1e-7

A slight increase in learning rate often resulted in random outputs and overfitting was small for the bag of words model and non existent for the LSTM which let us tune down the weight decay. Increasing the number of epochs didn't improve the performance since it stabilized at around 15000 iterations. Using Google Colab GPUs the time of computation for a full training loop is around 20 minutes with no significant difference between the models.

5 Experimental setup

The code implementation of the models described will be done using python torch implementation (PyTorch) on python notebooks running on Google Colab GPUs.

We use a github repository[1] to manage the version control of the code, dataset and papers.

[1] `https://github.com/DamianValle/deep-entailment`

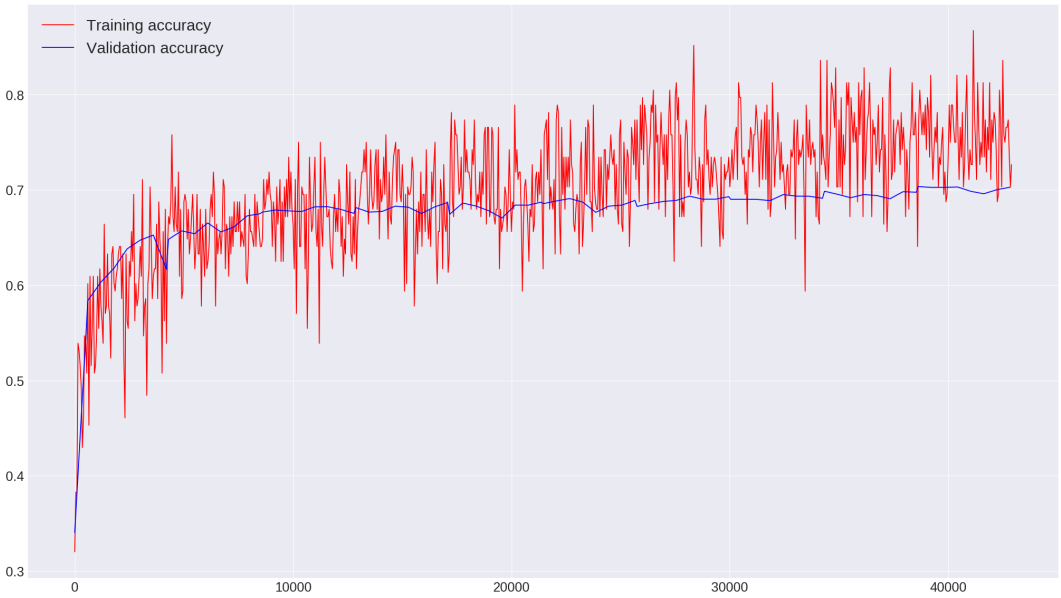

Figure 5: Bag of words model accuracy over training iterations

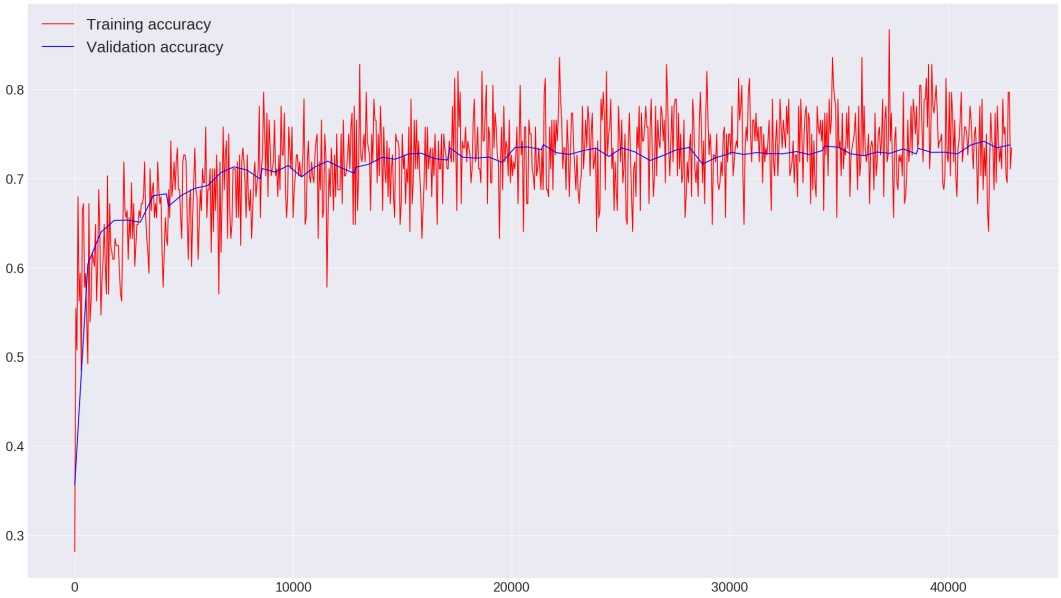

Figure 6: LSTM RNN model accuracy over training iterations

References

[1] Samuel R. Bowman, Gabor Angeli, Christopher Potts and Christopher D. Manning. *A large annotated corpus for learning natural language inference.* Standford Natural Language Processing Group, 2015.

[2] Samuel R. Bowman, Jon Gauthier, Abhinav Rastogi, Raghav Gupta, Christopher D. Manning, Christopher Potts. *A Fast Unified Model for Parsing and Sentence Understanding.* Standford Linguistics

[3] Bryan McCann, James Bradbury, Caiming Xiong and Richard Socher. *Learned in Translation: Contextualized Word Vectors.* Neural Information Processing Systems, Long Beach, CA, 2017

[4] Shuohang Wang and Jing Jiang. *Learning Natural Language Inference with LSTM.* Information Systems, Singapore Management University 2016

[5] Tomas Mikolov, Kai Chen, Greg Corrado and Jeffrey Dean. *Efficient estimation of Word Representations in Vector Space.* Google Inc., Mountain View, CA, 2013

[6] Jeffrey Pennington, Richard Socher and Christopher D. Manning *GloVe: Global Vectors for Word Representation.*

[7] Chris McCormick. *Word2Vec: The Skip-Gram Model.* `http://mccormickml.com/2016/04/19/word2vec-tutorial-the-skip-gram-model`

[8] Sepp Hochreiter and Jurgen Schmidhuber. *Long Short Term Memory.* Technische Universitat Munchen.

[9] MacCartney, Bill. *Natural language inference.* `https://search.proquest.com/docview/305018371`

[10] Ian J. Goodfellow, David Warde-Farley, Mehdi Mirza, Aaron Courville and Yoshua Bengio. *Maxout Networks.* Departement d'Informatique et de Recherche Operationelle, Universite de Montreal

[11] Seonhoon Kim, Inho Kang and Nojun Kwak. *Semantic Sentence Matching with Densely-connected Recurrent and Co-attentive Information.* Seoul National University

[12] Adam Poliak, Jason Naradowsky, Aparajita Haldar, Rachel Rudinger, and Benjamin Van Durme. *Hypothesis Only Baselines in Natural Language Inference.*