Implementation of an SNLI Based Deep Learning Model from the Literature

Itay Cohen

Submitted as an assignment report for the DL course, BIU, 2021

1 Introduction

In this assignment we were asked to reproduce results of an SNLI-based paper [1]. Stanford's website contains several papers which attempt to correctly classify samples in the SNLI dataset. The paper I chose to implement is "A Decomposable Attention Model for Natural Language Inference" by Parikh et al. [2]. I used the PyTorch package to reproduce its results.

2 The SNLI Corpus

Natural Language Inference (NLI), also known as Recognizing Textual Entailment (RTE), is the task of determining the inference relation between two (short, ordered) texts: entailment, contradiction, or neutral (MacCartney and Manning 2008 [3]).

The Stanford Natural Language Inference (SNLI) corpus (version 1.0) is a collection of 570k human-written English sentence pairs manually labeled for balanced classification with the labels entailment, contradiction, and neutral. Its purpose is to serve both as a benchmark for evaluating representational systems for text, especially including those induced by representation-learning methods, as well as a resource for developing NLP models of any kind. The following paper introduces the corpus in detail [4].

3 Paper Description

I chose to implement this specific paper for a few reasons. First, I was intrigued by the concept of attention, and I wanted so see how effective it is when used on a natural language dataset. Second, due to a lack of computing resources, I searched for a model with relatively low amount of parameters. The model in the paper contains 500k parameters, so I was able to train it using CPU resources only. Finally, the model description in the paper was very clear, so it seemed reasonable to reconstruct the network's computation graph.

3.1 Methods Used in the Paper

We present the vanilla version of the architecture used in the paper. The main concept is to create a soft alignment matrix using neural attention, and then use this alignment to create subproblems

that can be solved independently. Finally, the solutions to the subproblems are aggregated to produce the final classification. This approach does not consider the word order in a given sentence at all, and yet it achieved state of the art results back in 2016. The approach consists of three steps: attend, compare and aggregate. Feed forward neural networks of the same structure are used in each of the steps. This neural network has one hidden layer with ReLu activations and dropout for each layer.

Let $a = (a_1, ..., a_{l_a})$ and $b = (b_1, ..., b_{l_b})$ be two input sentences of lengths l_a and l_b, where each a_i and b_j are GloVe word embeddings of dimesion $d = 300$. The training set consists of pairs of sentences, and a label y for each pair.

3.1.1 Attend Step

Each word of a is aligned with a subphrase of b. A subphrase is a linear combination of embedding vectors of b. The weights of the linear combination are taken from the weight matrix e, and then normalized. This matrix is calculated in the following way:

$$e_{ij} := F(a_i)^T F(b_j) \tag{1}$$

Where F is a feed forward neural network of the form described above. We denote the aligned subphrase of a_i by β_i. Later on, each of b is aligned with a subphrase of a, which is obtained in a similar way, using the same weight matrix e. We denote the aligned subphrase of b_i by α_i.

3.1.2 Compare Step

Next, we concatenate each word with its aligned subphrase, and pass the concatenated vector through another feed forward neural network of the same architecture. We denote this neural network by G.

$$v_{1i} := G([a_i, \beta_i]) \qquad \forall i \in [1, ..., l_a] \tag{2}$$

$$v_{2j} := G([a_j, \beta_j]) \qquad \forall j \in [1, ..., l_b] \tag{3}$$

3.1.3 Aggregate Step

We sum each set of comparison vectors we calculated in the last step $\{v_{1i}\}_{i=1}^{l_a}$, $\{v_{2j}\}_{j=1}^{l_b}$.

$$v_1 = \sum_{i=1}^{l_a} v_{1i} \qquad , \qquad v_2 = \sum_{j=1}^{l_b} v_{2j} \tag{4}$$

Then, we concatenate $v1$ and $v2$ and pass them through another feed forward neural network of the same architecture. We finally feed the result through an additional linear layer with an output of three neurons.

$$\hat{y} = H([v_1, v_2]) \tag{5}$$

$\hat{y}$ represents the unnormalized scores for each class. $\hat{y}$ is next normalized by a log-softmax layer. The loss function for training was chosen to be cross entropy loss.

4 Results Comparison

I did not manage to replicate the paper's results, but got close. Train accuracy and test accuracy are above 80% after 250 epochs. The full performance comparison is below.

Method	Train Accuracy	Dev Accuracy	Test Accuracy
Paper results (vanilla approach)	0.895	0.869	0.863
My results	0.819	0.792	0.806

Table 1: Results Comparison Between the Two Models

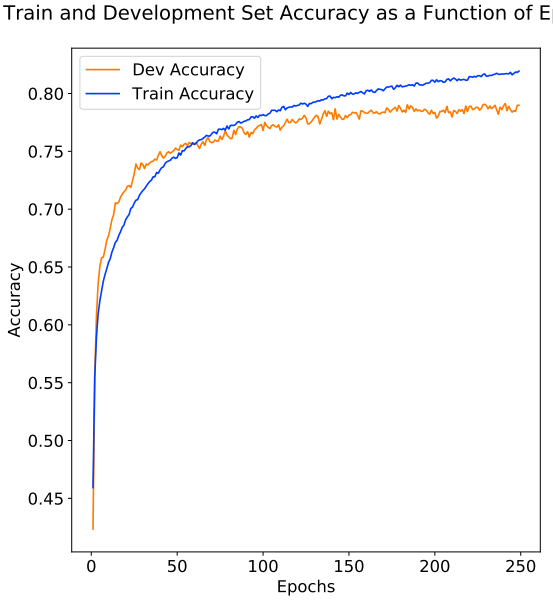

Figure 1: Train and Dev Accuracies over Epochs

5 Results Reproduction Details

5.1 Data Preprocessing

I extracted only the raw sentence pairs and their labels from the dataset, and ignored anything else. I saved the extracted data as text files, which were relatively lightweight comparing to the original dataset. The pairs were sorted with respect to the first sentence length, and the second sentence length. Then, I padded the sentences so all of them were long as the longest sentence found in the dataset. Later on, the pairs were splitted into batches, so each batch had a uniform length for the first sentences of each pair, and a uniform length for the second sentences of each pair (those lengths may be different from each other). Each batch contained 32 pairs at most, but a few of

them were shorter, if there was insufficient amount of sentences for a given length. At first I tried to train the model on the padded batches, but then noticed that the performance improves when the batches' padding is truncated, so each batch contains the sentences with no residual data.

Finally, I matched each sentence with its relevant GloVe word embeddings (Pennington et al., 2014 [5]). I used the 42B.300d version. At first I tried to load the entire embeddings file as a matrix. Since the projection matrix should be trained, the model contained too many parameters and there were memory issues. Consequently, I then loaded only the relevant embedding vectors, according to the dataset's vocabulary. This decision significantly reduced the amount of parameters in my model.

5.2 Computation Graph

The next step was designing the network. The whole network is primarily based on three identical feed-forward neural networks. When I designed the feed-forward neural network, I initially did not include the first dropout layer before the first ReLu layer. Not being satisfied with the test set accuracy, I searched for an existing implementation and figured out that one dropout layer is missing. It slightly improved the results.

Originally, I added a single linear layer (projection matrix) that is being trained. However, when defining it I did not explicitly mentioned that it should not include bias (the default value for this property is true). Removing the bias led to an improvement as well.

5.3 Training

The model was trained for 250 epochs, the development accuracy was calculated 3 times for each epoch. I used Adagrad as an optimizer, similarly to the way it was used in the paper. However, when I tried to configure it with an initial value accumulator it did not affect the results, so ultimately I did not include this modification. When I had to choose a loss function, I initially tried cross entropy loss with softmax layer on the outputs of the final layer. Then I realized that the paper described a bit different loss architecture: log-softmax layer on the outputs combined with a cross entropy loss. This change also had a positive impact on the results.

I first tried to train the model on my laptop, but then realized it would take too long to train it every time. Ultimately, the model was trained on a Google cloud C2 virtual machine with 16GB of RAM.

5.4 Hyperparameters

The following hyperparameters were used to reproduce the paper's results:

Hyperparameter	Value
Batch Size	32
Number of Out Of Vocabulary Embeddings	100
GloVe Embeddings Projection Dimension	300
Weight Initialization Std	0.1
Learning Rate	0.05
Hidden FNN Layer Size	300
Dropout	0.2
Number of Training Epochs	250

Table 2: Hyperparameters List

6 Suggestions

- The embedding dimension of each pair of sentences is relatively low (400) in the end of the aggregation step. It is possible that the current embeddings do not capture well enough the essence of each pair of sentences. It should be considered so increase this dimension.
- Using a richer pre-trained word embeddings that cover a larger portion of the dataset's vocabulary might have improved the results.

7 Code

The code for this project can be found below [6].

References

[1] (2021) The stanford natural language inference (snli) corpus. [Online]. Available: https://nlp.stanford.edu/projects/snli/

[2] A. Parikh, O. Täckström, D. Das, and J. Uszkoreit, "A decomposable attention model for natural language inference," in *Proceedings of the 2016 Conference on Empirical Methods in Natural Language Processing*. Austin, Texas: Association for Computational Linguistics, Nov. 2016, pp. 2249–2255. [Online]. Available: https://aclanthology.org/D16-1244

[3] B. MacCartney and C. D. Manning, "Modeling semantic containment and exclusion in natural language inference," in *Proceedings of the 22nd International Conference on Computational Linguistics (Coling 2008)*. Manchester, UK: Coling 2008 Organizing Committee, Aug. 2008, pp. 521–528. [Online]. Available: https://aclanthology.org/C08-1066

[4] S. R. Bowman, G. Angeli, C. Potts, and C. D. Manning, "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, 2015.

[5] J. Pennington, R. Socher, and C. Manning, "GloVe: Global vectors for word representation," in *Proceedings of the 2014 Conference on Empirical Methods in Natural Language Processing*

(EMNLP). Doha, Qatar: Association for Computational Linguistics, Oct. 2014, pp. 1532–1543. [Online]. Available: https://aclanthology.org/D14-1162

[6] (2021) Snli paper implementation. [Online]. Available: https://github.com/itay99988/snli-paper-implementation