Seq to seq approach to entailment problem

Hynek Kydlíček

November 13, 2022

Abstract

In this paper we will be looking at Natural language inference problem. Precisely we will evaluate seq to seq approaches to this problem.

1 Introduction

1.1 Natural language inference

Natural language interface problem also called entailment problem is a problem of determining whether a given sentence is entailed by another sentence. One ordered pair (S1, S2) of sentences can be either contradiction, neutral or entailment.

1. A pair is entailment if second is implied by first sentence.
   Example:

   S1: 'My brother's name is Hynek'

   S2: 'I have a brother'

2. A pair is contradiction if second sentence in contradiction with first sentence.
   Example:

   S1: 'I have one brother, his name is Hynek'

   S2: 'I have a brother called Peter'

3. A pair is neutral if pair is neither entailment nor contradiction.
   Example

   S1: 'I have a brother'

   S2: 'I have a sister'

While the description is simple, the problem is not and there are few things that need to be considered.

1. Different languages problem: The sentences can be in different languages.
2. Entity coreference problem: Given two sentence: S1: ‘Peter slept at home’, S2: ‘Peter slept at his friends’ it is not clear if both sentences refer to the same Peter
3. Events coreference problem: Given two sentence: S1: ‘I am in library’, S2: ‘I am in school’ it is not clear whether both events are happening at the same time.

Due to these problems we decided to use dataset which tackles these problems.

1.2 SNIL dataset

SNIL dataset was created at Stanford University and is a collection of 570k sentence pairs (“The Stanford Natural Language Processing Group”, n.d.). It was introduced in 2016 and is one of the most popular dataset for NLI problem. The sentences are in English and are annotated by humans. The collection of data was done by crowdsourcing with Amazon Mechanical Turk. The crowdsourcers were given a image caption from Flickr which served as a premise and they had to write a sentence which would be either in contradiction, neutral or entailment relation with the premise. The mentioned problems were solved by

1. Different languages problem: All sentences in English.
2. Entity coreference problem: The image captions grounded the sentence to specific situation and the participants had to make sentence with respect to the situation.
3. Event coreference problem: Same as above

The sentences were then validated by another group of humans and only ones with high inter-annotator agreement were kept. The paper by Bowman et al., 2015 that introduced the dataset also introduced a SOTA model of that time with accuracy of 80.8% on test set.

1.3 Seq to seq problem and approaches

Seq to seq problem is a problem of translating one sequence to another. The most typical example is translating from one language to another. The problem is usually solved by using encoder-decoder architecture. The first sequence is first encoded into a context vector and the context vector decoded into the second sequence. There are two prominent deep learning approaches that emerged in the last few years RNNs and Transformers.

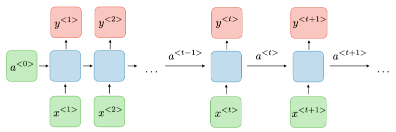

Figure 1: RNN architecture (“CS 230 - Recurrent Neural Networks Cheatsheet”, n.d.)

1.3.1 RNNs

RNNs are a type of neural network that can be used to solve seq to seq problem. From overview they look like Figure 1.

The problem of RNNs is that they are not able to capture long term dependencies. They are also not able to parallelize the computation because they depend on previous context vector. There is also a problem of vanishing gradient which makes it hard to train RNNs. The vanishing gradient is partially solved by using LSTM and GRU cells. The LSTM cells are very similar to standard RNN cell but have one more vector they pass between each other called cell state. This helps with gradient propagation. Staudemeyer and Morris, 2019 is a good in-depth look at LSTM cells.

1.3.2 Transformers

Transformers are newer architecture that was introduced in 2017 by Vaswani et al., 2017. The overview of architecture can be seen at Figure 2. The only part we will explain here is the multi-head attention as it’s the most important one. If you are interested in more in-depth look at transformers please look at the paper. In input of multi-head attention we have n vectors of size d. The vectors are translated into their 3 new representations Keys, Values and Queries. One Query is then dot-producted by every Key. The result is then softmaxed to normalize and then multiplied by appropriate Values. Then we sum results to get the vector of size d. We do this for every Query. This yields the same output size as input size. We can think of this as each input attending to all other inputs to find the most important ones (They will have high dot-product and thus they will impact the resulting vector the most). The attention can be done multiple time for one layer. This is why it’s called multi-head.

The encoder thus uses several layers of multi-head attention at the end producing k Keys and Values. The decoder also uses the multi-head attention with Queries,Keys and Values produced by input vectors but it also uses one where Keys and Values are ones produced by encoder and Queries are the ones produced by decoder.

1.4 Transfer learning

Transfer learning refers to technique of training a model on one task and then reusing the same model and it weights on another task. This can mean replacing

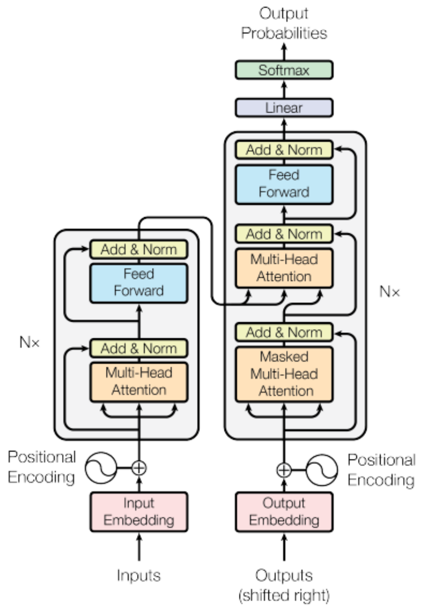

Figure 2: Transformer architecture (Vaswani et al., 2017)

the last layer of a model with a new one and training it on the new task while freezing the other layers or it can simply mean using the pretrained embeddings for the words and using the embeddings with the model In our case we will be using the second approach. We will train all our models with the Glove embeddings as defined in Pennington et al., 2014. Glove embeddings is a collection of word embeddings. They exist in different variants varying in size and training corpus. We will be using the 300 dimensional 6B variant.

2 Methodology

2.1 Research question

With all the key terms introduced we can now formulate our research question.

> **We want to evaluate which of our trained models (RNN and Transformer) will have better accuracy on SNIL test dataset.**

We chose these models to have at most 2M trainable parameters and we also required them to use 300d 6B Glove embeddings. We decided to use pretrained embedding because we wanted to focus on the architecture and not the embeddings. We also hopped that the embeddings would overall increase the accuracy of trained models as they carry implicit information. The 2M parameter limit was chosen to guarantee reasonable training time.

2.2 Hypothesis

Let's formulate the hypothesis:

> **We predict that the Transformer model will have higher accuracy than the RNN model on SNIL test dataset.**

To prove that we we will try to reject the null hypothesis, which can be stated as:

> **The accuracy of the Transformer model will be less or equal than the accuracy of the RNN model on SNIL test dataset.**

We chose the p-value of 0.05 as the threshold for rejecting the null hypothesis. We will use paired-bootstrap test to reject the null hypothesis.

3 Models description

In this section we will describe the RNN and Transformer models we trained.

3.1 RNN

For RNNs we decided to use simple LSTM multilayer cells. The multilayer version of LSTM cell is similar to the standard LSTM cell but it uses multiple LSTM cells at once. The input to next layers of cells is output from previous one. It was first introduced by Graves et al., 2013. Also there is a dropout layer between each layer of multilayer-LSTM cells. The architecture of the model can be seen at Figure 3.

As can be seen we used max pooling to get vector for linear layer. We haven't

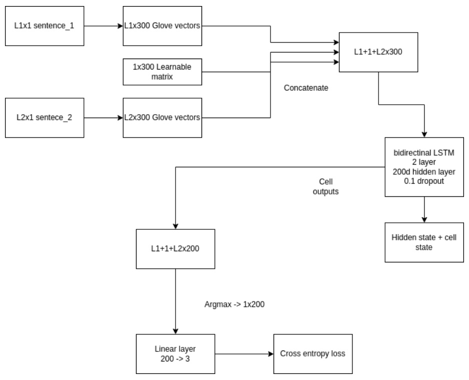

Figure 3: RNN model architecture

tried other ways but it might be interesting to see, how other methods would evaluate (for examples sum). Also note that we used outputs of LSTM to feed into linear layer and not the hidden state. We tried using the hidden state but we got worse results. Total number of trainable parameters of this model is 1.8M.

3.2 Transformer

For transformer we tried many different architectures and we had really huge problems to get it trained. The biggest problem was that it was not overfitting even on one batch after 200 epochs. That is a problem as it shows model's inability to learn just hundreds (one batch) of samples. We tried to use just encoder with similar input and output processing as with RNNs but it didn't overfit even after 1k epochs. At the end we used the encoder decoder architecture. While it was also overfitting very slowly at least it was overfitting. The final architecture can be seen at Figure 4.

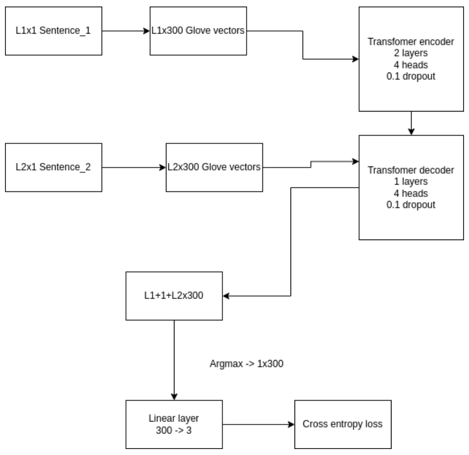

Figure 4: Transformer model architecture

Again we used max pooling to get vector for linear layer. Total number of trainable parameters is 1.9M.

3.3 Notes on training

1. **We used Cross-entropy loss for both models.**
   We chose this loss as it is the most common one for classification tasks.

2. **We used Adam optimizer with learning rate of 0.001 for both models.**
   We tried using different $(10^{-4}, 10^{-2}, 10^{-1})$ learning rates but we got worse loss after one epoch of training on test set. However we acknowledge that we should have used more sophisticated methods like (Smith, 2017).

3. **We used gradient clipping with value of 0.5 for both models.**
   This was added mainly to prevent exploding gradients in RNNs.

4. **We used 40 epochs for Transformer model and 20 epochs for RNN model.**
   These values were chosen to achieve reasonable training times.

The training was done on K80 GPU with 12GB of memory on Google Colab (Because it's impossible to use school computers). All the code is available at guess.ipynb and is written in PyTorch. The trained model weights are available at models/. Tensorboard logs are available at logs/.

4 Results

The results of the models can be seen at Table 1. It's important to note that

Model	RNN	Transformer
Parameters	1.8M	1.9M
Accuracy (Train)	0.96	0.7
Accuracy (Test)	0.78	0.69

Table 1: Results of the models

the dataset is well-balanced so the accuracy is a good metric to use. While the accuracy is way better than random chance model which would have 0.3 accuracy, we were hoping to get better results for both models. The accuracy of the RNN is more than 20 percent points higher than the accuracy of the Transformer on train dataset. As we said we had problems with inability to learn of the Transformer model and this is just another proof of that. The results on test dataset look a bit better for the Transformer model but it's still significant difference of 9 percent points. Overall difference is very dramatic on both datasets. The loss curves can be seen at Figure 5. The accuracy on training set can be seen at Figure 6. We tried to reject null-hypothesis with paired-bootstrap test and we got p-value of 1.0. Thus we cannot reject the null-hypothesis and we can't say that our hypothesis is true with 95% confidence.

5 Discussion

We were very surprised by the results. We expected the Transformer to perform much better, because of the attention mechanism. Clearly there was a problem with learning as we mention that we found it very hard to overfit on single batch and it was overall unstable. It might be solved by using different optimizer or scheduler. We don't think we could find a better loss function for this problem.

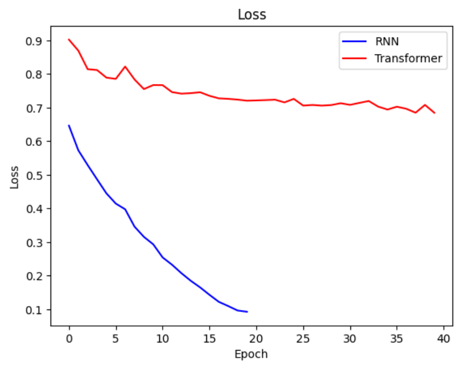

Figure 5: Loss curves of the models

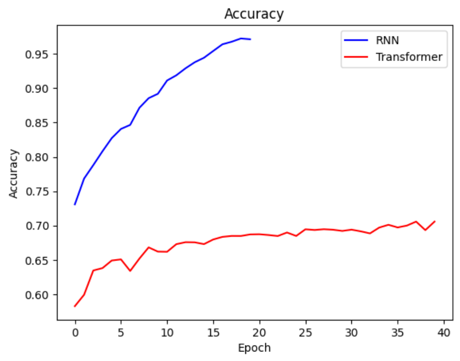

Figure 6: Accuracy curves of the models

It's also possible that the glove embeddings are not the best choice for this problem.

Note: I talked about this with might bc. thesis supervisor and he said that I should have used leraning scheduler as the transformer need high learning rate at the beginning and then it should be decreased. Also he said that I should have used different optimizer as Adam is not the best choice for transformer.

6 Conclusion

We have evaluated the Transformer and RNN architecture on the NLI problem. While we expected the Transformer to perform much better, it didn't. However we could clearly see that it was generalizing way better. So it's possible that if we would substantially increase the number of parameters it would perform better. This would be very interesting to research in future. Another interesting thing would be to not use pretrained embedding but to train them. Last but not least we could try to fine-tune a language model for this task.

References

Bowman, S. R., Angeli, G., Potts, C., & Manning, C. D. (2015, August 21). A large annotated corpus for learning natural language inference. https://doi.org/10.48550/arXiv.1508.05326

CS 230 - Recurrent Neural Networks Cheatsheet. (n.d.). Retrieved October 30, 2022, from https://stanford.edu/~shervine/teaching/cs-230/cheatsheet-recurrent-neural-networks#

Graves, A., Mohamed, A.-r., & Hinton, G. (2013, March 22). Speech Recognition with Deep Recurrent Neural Networks. Retrieved October 31, 2022, from http://arxiv.org/abs/1303.5778

Pennington, J., Socher, R., & Manning, C. (2014). GloVe: Global Vectors for Word Representation. *Proceedings of the 2014 Conference on Empirical Methods in Natural Language Processing (EMNLP)*, 1532–1543. https://doi.org/10.3115/v1/D14-1162

Smith, L. N. (2017). Cyclical Learning Rates for Training Neural Networks. *2017 IEEE Winter Conference on Applications of Computer Vision (WACV)*, 464–472. https://doi.org/10.1109/WACV.2017.58

The Stanford Natural Language Processing Group. (n.d.). Retrieved October 27, 2022, from https://nlp.stanford.edu/projects/snli/

Staudemeyer, R. C., & Morris, E. R. (2019, September 12). Understanding LSTM – a tutorial into Long Short-Term Memory Recurrent Neural Networks. Retrieved October 30, 2022, from http://arxiv.org/abs/1909.09586

Vaswani, A., Shazeer, N., Parmar, N., Uszkoreit, J., Jones, L., Gomez, A. N., Kaiser, L., & Polosukhin, I. (2017, December 5). Attention Is All You Need. https://doi.org/10.48550/arXiv.1706.03762