Seq to seq approach to entailment problem

Hynek Kydlíček

November 1, 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. The two sentences can be either in contradiction, neutral or entailment relation. While the description is simple, the problem is not and there are few things that need to be considered.

1. The sentences can be in different languages.
2. Entity conference is a problem. Given two sentence: 'Peter slept at home', 'Peter slept at his friends' it is not clear if both sentences refer to the same Peter
3. Events conference is a problem. Given two sentence: 'I am in library', '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 SNIL dataset which tackles these problems.

1.2 SNIL dataset

[5] SNIL dataset was created at Stanford University and is a collection of 570k sentence pairs. 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. All sentences in English.
2. The image captions grounded the sentence to specific situation and the participants had to make sentence with respect to the situation.
3. Same as above

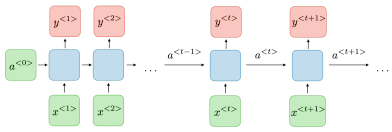
The sentences were then validated by another group of humans and only one with high inter-annotator agreement were kept. The paper [5] that introduced the dataset also introduced a SOTA model of that time with accuracy of 80.8% on test set.

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 2 prominent deep learning approaches that emerged in the last few years RNNs and Transformers.

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 1.3.1. The problem of RNNs is that they are not



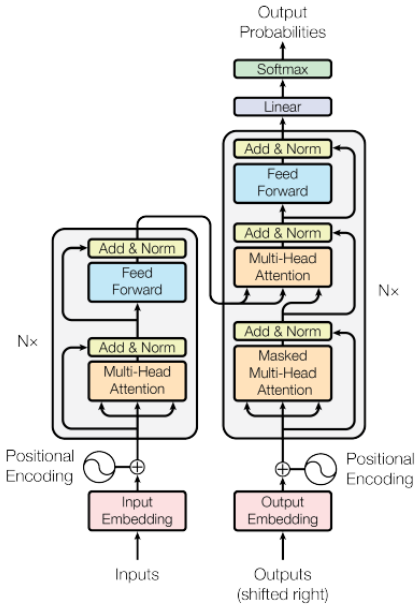
[1]

Figure 1: RNN architecture

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.[4] is a good in-depth look at LSTM cells.

1.3.2 Transformers

Transformers are newer architecture that was introduced in 2017 by [6]. The overview of architecture can be seen at 1.3.2. 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



[6]

Figure 2: Transformer architecture

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 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 [3]. Glove embeddings is a collection of word embeddings. They

exists in different variants varying in size and training corpus. We will be using the 300 dimensional 6B variant.

1.5 Research question

With all the key terms introduced we can now formulate our research question. Given the SNIL dataset we want to evaluate whether later described Transformer architecture is better than later described RNNs for the NLI problem with $p = 0.05$. We chose these models to have at most 2M trainable parameters and we also required them to use 300d 6B Glove embeddings. We will evaluate the models by measuring accuracy on the test set.

2 Models description

2.1 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 [2]. Also there is a dropout layer between each layer of multilayer-LSTM cells. The architecture of the model can be seen at 2.1.
As can be seen we used max pooling to get vector for linear layer. We haven't

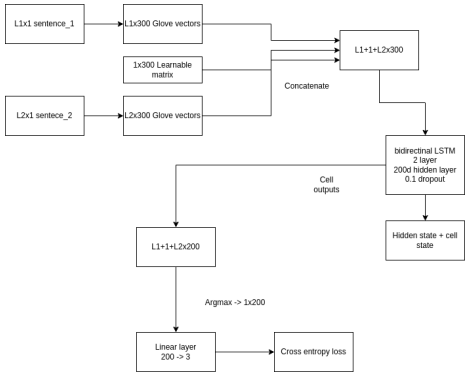


Figure 3: RNN model architecture

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.

2.2 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. 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 2.2.

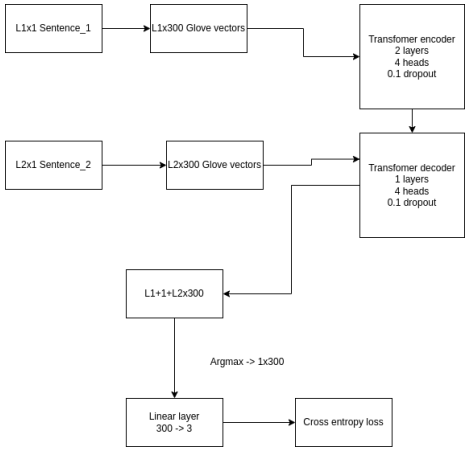


Figure 4: Transformer model architecture

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

2.3 Notes on training

We used Cross-entropy loss for both models. We used Adam optimizer with learning rate of 0.001 for both models. We used gradient clipping with value of 0.5 for both models. We used 40 epochs for Transformer model and 20 epochs for RNN model. 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/`.

3 Results

The results of the models can be seen at 1. The loss curves can be seen at 3. The accuracy on training set can be seen at 3. From the results we can immediately see that our hypothesis was wrong. Just to make sure we run paired bootstrap

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

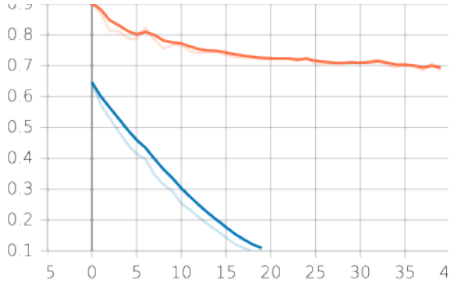


Figure 5: Loss curves of the models

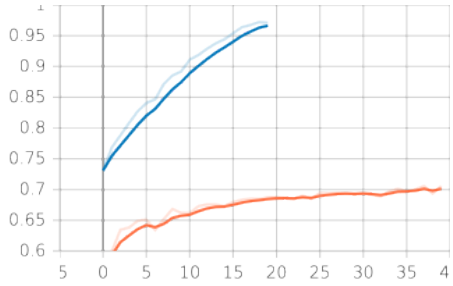


Figure 6: Accuracy curves of the models

test with 50 samples 500 times. The p-value was 1.0, so we cannot reject the null hypothesis.

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

5 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

[1] *CS 230 - Recurrent Neural Networks Cheatsheet*. URL: https://stanford.edu/~shervine/teaching/cs-230/cheatsheet-recurrent-neural-networks# (visited on 10/30/2022).

[2] Alex Graves, Abdel-rahman Mohamed, and Geoffrey Hinton. *Speech Recognition with Deep Recurrent Neural Networks*. Mar. 22, 2013. arXiv: 1303.5778 [cs]. URL: http://arxiv.org/abs/1303.5778 (visited on 10/31/2022).

[3] Jeffrey Pennington, Richard Socher, and Christopher Manning. "GloVe: Global Vectors for Word Representation". In: *Proceedings of the 2014 Conference on Empirical Methods in Natural Language Processing (EMNLP)*. EMNLP 2014. Doha, Qatar: Association for Computational Linguistics, Oct. 2014, pp. 1532–1543. DOI: 10.3115/v1/D14-1162. URL: https://aclanthology.org/D14-1162 (visited on 10/31/2022).

[4] Ralf C. Staudemeyer and Eric Rothstein Morris. *Understanding LSTM – a Tutorial into Long Short-Term Memory Recurrent Neural Networks*. Sept. 12, 2019. arXiv: 1909.09586 [cs]. URL: http://arxiv.org/abs/1909.09586 (visited on 10/30/2022).

[5] *The Stanford Natural Language Processing Group*. URL: https://nlp.stanford.edu/projects/snli/ (visited on 10/27/2022).

[6] Ashish Vaswani et al. *Attention Is All You Need*. Dec. 5, 2017. DOI: 10.48550/arXiv.1706.03762. arXiv: 1706.03762 [cs]. URL: http://arxiv.org/abs/1706.03762 (visited on 10/30/2022).