Spring 2019 Home Exam: Natural Language Inference Encoders

Skjalg Gustav Eriksen
Language Technology Group, Department of Informatics, University of Oslo
`skjale@ifi.uio.no`

Abstract

Natural language inference is a natural language task about determine entailment between two sentences. This is done as a classification task, but the sentences can have varying length. Therefore there is a need to make a fixed-length-representation of the sentences. In this paper I explore different encoder architectures.

1 Introduction

This paper is the final home exam for IN5550 at UiO. I have chosen task 2 about natural language inference (NLI). The Goal of NLI is to determine the entailment between two sentences, in other words does the sentence 1 entail sentence 2. This is done as a classification task with 3 classes; entailment, contradiction and neutral. Since the two sentences can be of varying length there is a need to make a fixed-length-representation of the two sentences. The fixed-length-representations can then be passed down to the classifier. The general structure is showed in figure 1, where we first encode the sentences with one encoder and then pass these representations down to a multi-layer perceptron (MLP). This follows the Generic NLI training scheme (Conneau et al., 2017, Figure 1).

1.1 Sections

The remainder of this paper is organized in five sections. Section 2 sets up the general structure of the classifier. Section 3 briefly discuss a paper before taking a deeper look at Encoder architectures. Section 4 starts by testing the baseline models for different encoder architectures. Section 5 experiments with hyper-parameters. Section 6 concludes the paper.

2 Multi-layer perceptron (MLP) as a classifier

My Classifier will have a encoder model as a parameter, which it will use on the sentences and encode them into representation u and v. these are then concatenated together in 3 ways

$$[u, v, |u - v|, u * v]$$

this is then passed into a single layer MLP[1].

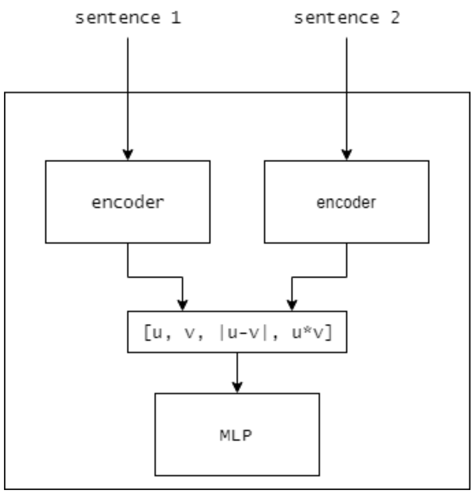

Figure 1: model structure

classifier baseline parameters

Parameter	value
number of hidden layers	1
hidden layer size	512
activation	relu
dropout	0

Table 1: baseline hyper-parameters for the MLP classifier

[1] A single layer MLP is a class of feedforward neural network (Goldberg, 2017, Chapter 4.2) where you have linear input layer, one non-linear activation and a linear output layer.

3 Fixed length representation

To turn a sentence into a fixed length representation you need to encode the sentence using a encoder. In the paper (Conneau et al., 2017, Section 3.2) they discuss four different sentence encoder architectures; LSTM and GRU, BiLSTM with max pooling, self-attentive sentence embedding. I will take a deeper look at these architectures in the following sections.

3.1 LSTM and GRU

A simple sentence encoder can be a long short-term memory (LSTM) or Gated recurrent unit (GRU) where you take the last hidden sate and using that as a representation of the sentence (Conneau et al., 2017, Section 3.2.1). The GRU and LSTM are different types of recurrent neural network architectures designed to remedy the vanishing gradient problem (Goldberg, 2017, Chapter 15.3). The GRU consists of an update gate and a reset gate, these will update or reset the GRU depending on the input. The LSTM consists of an update gate and reset gate like the GRU, however LSTM has an additional gate for the output. LSTM also splits its hidden states into ”memory cells” and working memory (Goldberg, 2017, Chapter 15.3.1). [2]

Parameter	value
hidden state size	1024

Table 2: baseline hyper-parameters for the LSTM and GRU encoder

3.2 Hierarchical convolutional neural network (ConvNet)

ConvNet is described in (Conneau et al., 2017, Section 3.2.4). ConvNet is a convolutional neural network with four convolutional layers, each with kernel size 3. Each layer does a convolution over the output of the previous layer. As the final sentence representation each layer will be max-pooled and concatenated. This will give a hierarchical representation of the sentence with four different layers of abstraction concatenated together as the final output. Figure 2 is a simple figure showing this structure.

[2]I have dubbed this encoder 'lastStateEncoder' in my code.

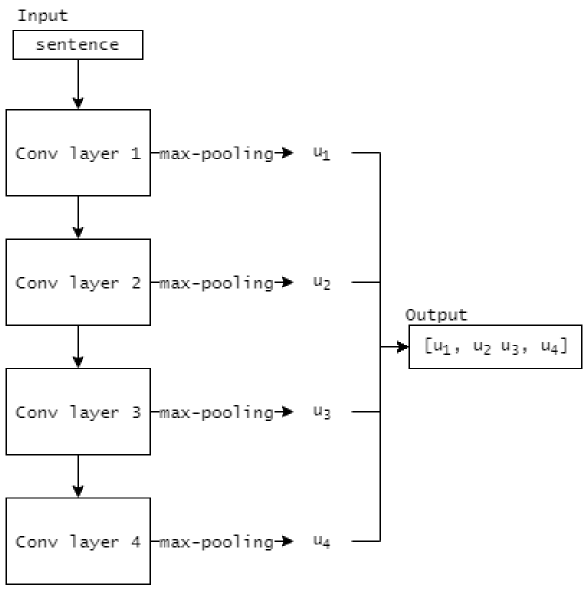

Figure 2: ConvNet encoder structure

Parameter	value
convolutional layers	4
kernel size	3
padding	1
filter size	1024
pooling	max

Table 3: baseline hyper-parameters for the convNet encoder

3.3 Bi-directional LSTM (BiLSTM) with max-pooling

In (Conneau et al., 2017, Section 3.2.2) Conneau et al. describes a BiLSTM with max-pooling or mean-pooling over all the hidden states to make a sentence representation. The sentence will be sent through the BiLSTM and for each word there are two hidden states, one for each direction. These two hidden states $\overrightarrow{h_t}$ and $\overleftarrow{h_t}$ will be concatenated together to make up a hidden state h_t and the final representation will be a max pooling over all the concatenated hidden states $H = [h_1, h_2, ...h_n]$. The resulting vector representation will have the shape $(1, 2d)$ where d is the hidden state size of the BiLSTM. This structure is shown in figure 3.

Parameter	value
hidden state size	1024
Bidirectional	true
pooling	max

Table 4: baseline hyper-parameters for the BiLSTM with max pooling encoder

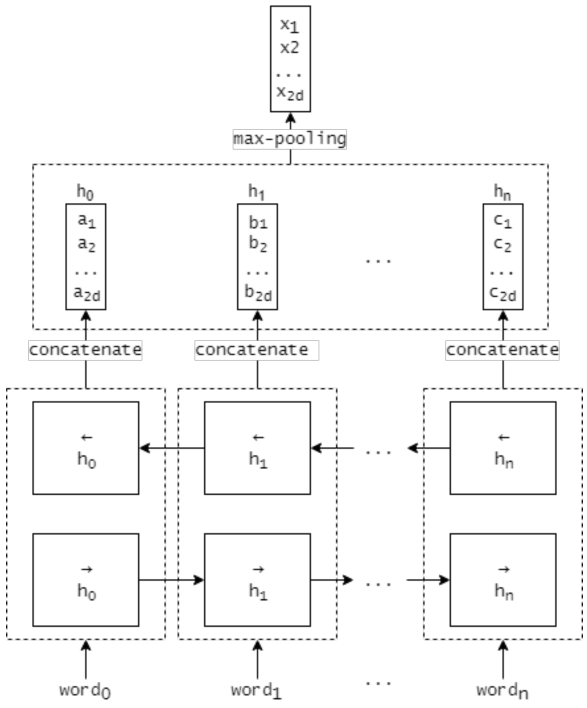

Figure 3: BiLSTM with max-pooling encoder structure

3.4 Self-Attentive Sentence Embedding

So far the encoders mentioned are non-attentive, they do not consider what words in the sentences is important. The self-attentive sentence embedding uses a simple from of attention using a MLP on the hidden states from a BiLSTM to make an attention distribution.

Starting with the BiLSTM, Like explained in section 3.3 you get a hidden state matrix $H = [h_1, h_2, ...h_n]$, but instead of doing a max-pooling over H we want to make a sentence embedding.

H is an n-by-2d matrix where n is the number of words in the sentence and d is the hidden state size of the BiLSTM. With the equation from (Lin et al., 2017, Equation 6) we define attention matrix A as:

$$A = softmax(W_{s2}tanh(W_{s1}H^T))$$

Lets go through the formula step by step:

$$W_{s1}H^T$$

W_{s1} is a weight matrix with dim d_a-by-2d, and since H^T is a 2d-by-n this results in a d_a-by-n matrix. Where d_a is arbitrary chosen hyper-parameter.

$$W_{s2}tanh(W_{s1}H^T)$$

W_{s2} is a r_{hops}-by-d_a, r_{hops} is a hyper-parameters deciding the number of attention hops or attention distributions the sentence embedding is going to contain. After operation this we will have a r_{hops}-by-n matrix.

$$softmax(W_{s2}tanh(W_{s1}H^T))$$

The last softmax will make r_{hops} attention distributions over the n words in the input sentence. The final sentence embedding M will then be the hidden states H multiplied by the attention distributions.

$$M = AH$$

The final embedding will have a dimension r_{hops}-by-2d which will is a fixed size representation.

Parameter	**value**
hidden state size	1024
(r_{hops}) attention hops	4
(d_a) attention dimension	256
penalty coefficient	0

Table 5: baseline hyper-parameters for the self-attentive sentence Embedding encoder

4 Baseline Models Comparison

The NLI5550[3] training dataset provided for the project is huge, containing 550 150 examples. Since this project has a limited time period I need to cut the training data down to a manageable size. I'm going to use 10% of the dataset, this amounts to 55 015 examples. Even when limiting the dataset training is very time consuming so each model will get to train for 5 epochs and at each epoch the model will be evaluated against the dev dataset. Normally you would train each model a couple of times to get a solid performance test, but I will only train each model once and treat it as an estimate. With these tests I hope to get a grasp over which encoders work so it can inform and motivate further testing. Table 6 lists the hyper-parameters for training.

Parameter	**value**
batch size	32
learning rate	0.001
epochs	5
embedding	glove.6B.100d

Table 6: general hyper-parameters

[3]https://github.uio.no/in5550/2019/blob/master/tasks/nli/nli5550.zip

4.1 Results

Figure 4 is a plot that shows the baseline accuracy on dev. Due to size limits, Ive shortened

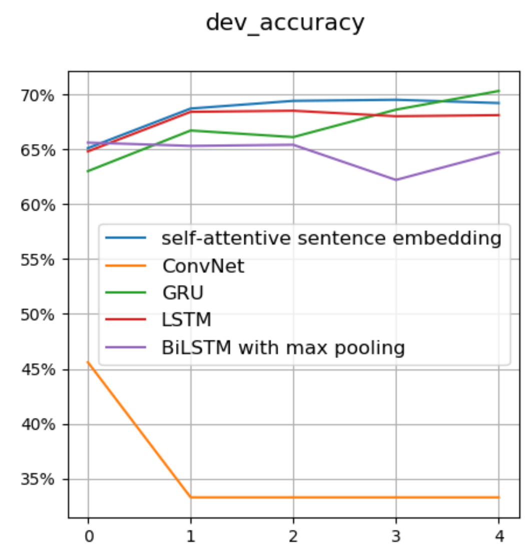

Figure 4: Plot of dev accuracy in y-axis and epoch in x-axis for baseline models

self-attentive sentence embedding to attention, and BiLSTM with max-pooling to just BiLSTM in Table 7. For some reason ConvNet preformed really

model	dev accuracy	macro F1-score
LSTM	0.685	0.5175
GRU	**0.703**	**0.5306**
BiLSTM	0.656	0.4956
convNet	0.456	0.3222
attention	0.695	0.5251

Table 7: Best baseline result for each model

bad and lost all its predictive power after the first epoch. This might be due to the filter sizes being too large. Surprisingly the GRU encoder preformed unexpectedly the best of all the encoders. It might be the more simplistic gates giving it an edge with less complexity. The attention network did preform pretty good and not too far under the GRU encoder. BiLSTM did worse than the LSTM, this could be because the max-pooling lose too much contextual information about sentence whereas LSTM will have similar sequential context for both sentences.

5 Experiments

In this section I will do some experiments with hyper-parameters, changing one parameter at a time and compare it to the original baseline.

5.1 Attention penalization

In section 3.4 I described an attention mechanism that can have multiple ’hops’ of attention. To encourage diversity of attention between the attention hops (Lin et al., 2017) proposes a penalization that punishes attention hops that are similar. Penalty P is given by:

$$P = ||(AA_T - I)||_F$$

Here A is the r_{hops}-by-n attention matrix, I is A’s the identity matrix, and $||\ ||_F$ is frobenius norm of a matrix. P is multiplied by a coefficient then added to and minimized with the loss during training. P will be higher when the attention hops are similar then go down as they diverge.

5.1.1 baseline with and without penalty

As shown in Figure 5 the regularization did not have a positive effect, though this is not surprising as (Lin et al., 2017, Section 4.4.1) did not have improvements when using penalization on the SNLI dataset.

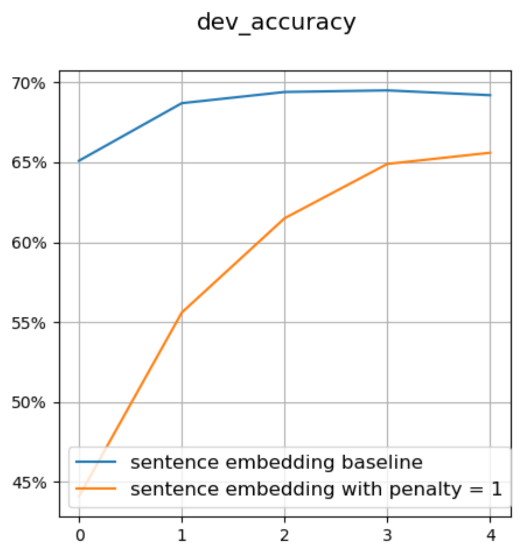

Figure 5: Plot of dev accuracy in y-axis and epoch in x-axis for sentence embedding with penalty coefficient=1

model	dev accuracy	macro F1-score
baseline	0.695	0.5251
penalized	0.656	0.4943

Table 8: Results for penalization

5.1.2 Visualizing the attention hops

Since the effects were not positive I decided to find out how exactly the attention penalization had played out. Using bar plot I plot each attention hop's attention distribution over the sentences. Each attention hop is given their own color. In Figure 6 shows the baseline encoder and Figure 7 shows the encoder with penalty.

Looking at the attention distribution for the baseline model it does seem to make a useful distribution. Attention hop 0 highlights 'A' and 'girl' in both sentences, while features that are not important have low attention. In the attention distribution for Figure 7 its pretty evident that the penalty has impaired the encoders ability to learn useful attention features, and it has a lot of attention on words that mean little. However the penalty itself does look effective at making the attention hops focus on different words or aspects of the sentence.

With a penalty coefficient of 1 the network is prevented from learning useful attention. This likely occurs because the penalty is added to the loss and since the penalty can be comparatively big it takes precedence over loss generated from learning attention. Therefore using a small penalty coefficient will most likely provide better results.

Since the baseline has a small number of attention hops to begin with having redundant attention does not seem to be a problem. The penalty should be a lot more effective when dealing with a large number of attention hops.

5.2 GRU self-Attention

With the results from Figure 4 showing GRU coming out on top and out preforming the LSTM significantly (given these are just estimates). There might be an advantage with using a GRU for textual entailment. so I decided to test a encoder the same architecture described in section 5 but instead of using a BiLSTM, use a BiGRU.

5.2.1 Results

Surprisingly table 9 shows the BiGRU encoder with self-attention preformed on equal grounds with the normal GRU encoder. They did have a different learning curve and the self-attention BiGRU reached a peak earlier than the baseline GRU encoder. I'm guessing the BiGRU is not fit for the attention scheme I'm using or my attention parameters are not optimized enough for the attention to make a difference.

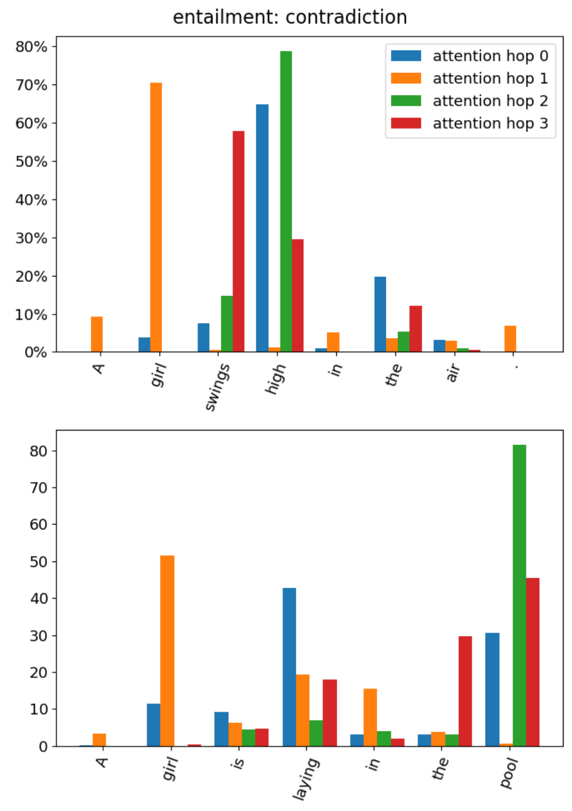

Figure 6: Plot of the attention distribution in matrix A for baseline sentence embedding encoder on example 100 in dev dataset. The y-axis shows how much percent attention each word on the x-axis is getting.

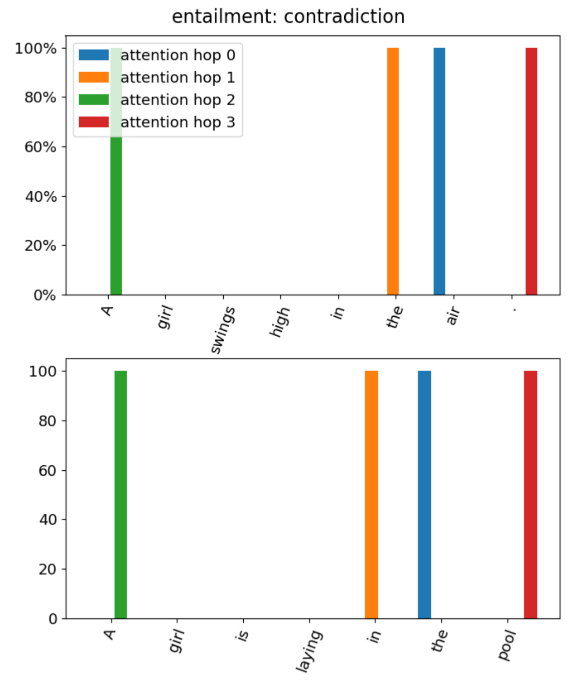

Figure 7: Plot of the attention distribution in matrix A for sentence embedding encoder on example 100 with penalty coefficient = 1. The y-axis shows how much percent attention each word on the x-axis is getting.

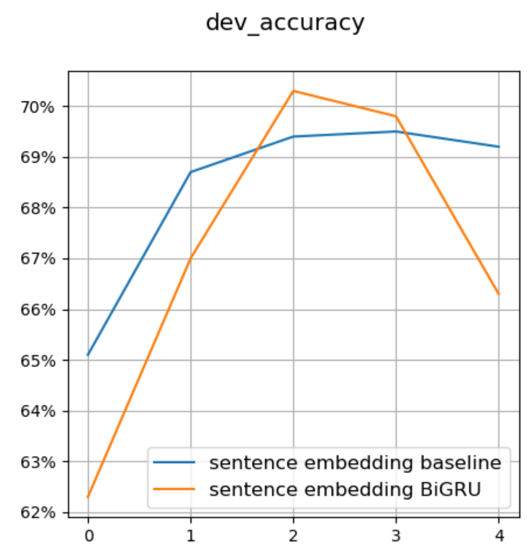

Figure 8: Plot of dev accuracy in y-axis and epoch in x-axis for sentence embedding with BiGRU and baseline

model	dev accuracy	macro F1-score
baseline	0.695	0.5251
BiGRU	0.703	0.5306

Table 9: Results for baseline attention embedding and attention embedding with BiGRU

6 Conclusion

In this paper I have gotten familiar with four different encoders; GRU, LSTM, BiLSTM with max pooling and Self-attentive sentence embedding. The GRU encoder had the best baseline performance. I have explored attention penalization and seen that can be very efficient at splitting attention hops away from each other but having a too high penalization hinders the network in learning to give useful features in the sentence attention. I'm convinced the self-attentive sentence embedding will be better than the GRU encoder with more parameter optimization.

6.1 More plots

Some hyper-parameter search without follow up.

References

Alexis Conneau, Douwe Kiela, Holger Schwenk, Loic Barrault, and Antoine Bordes. 2017. Supervised learning of universal sentence representations from natural language inference data. *arXiv preprint arXiv:1705.02364*.

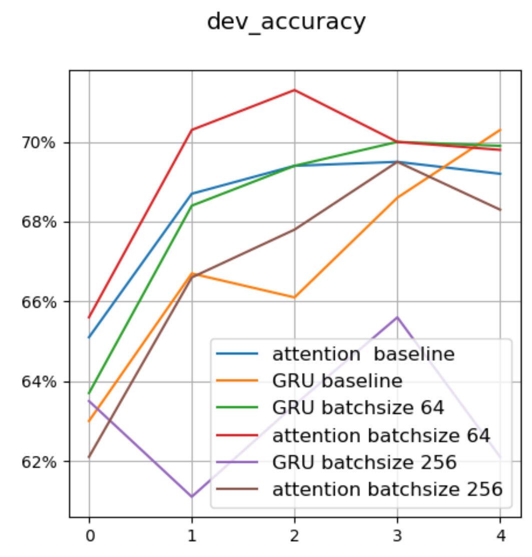

Figure 9: Plot of dev accuracy in y-axis and epoch in x-axis for varying batch sizes with self-attentive sentence embedding (attention) and GRU

model	dev-accuracy	macro F1-score
attention baseline	0.695	0.5251
GRU baseline	0.703	0.5306
attention batchsize 64	**0.713**	**0.5393**
GRU batchsize 64	0.700	0.5271
attention batchsize 256	0.695	0.5251
GRU batchsize 256	0.656	0.4921

Table 10: varying batch sizes for self-attentive sentence embedding (attention) and GRU

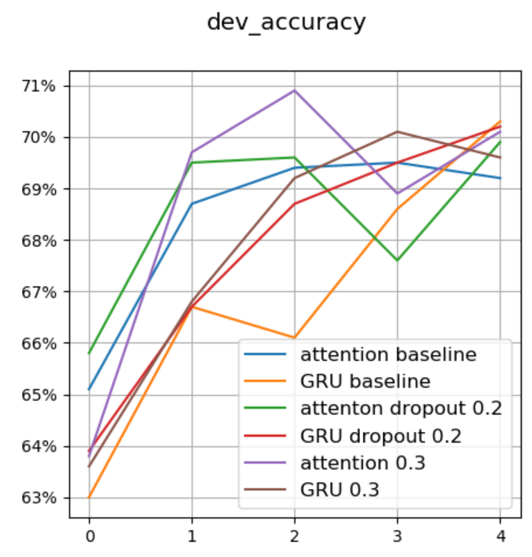

Figure 10: Plot of dev accuracy in y-axis and epoch in x-axis for varying dropout values with self-attentive sentence embedding (attention) and GRU

model	**dev accuracy**	**macro F1-score**
attention baseline	0.695	0.525089
GRU baseline	0.703	0.530564
attention dropout 0.2	0.699	0.527405
GRU dropout 0.2	0.702	0.531259
attention dropout 0.3	**0.709**	**0.534944**
GRU dropout 0.3	0.701	0.530944

Table 11: varying dropout values for self-attentive sentence embedding (attention) and GRU

Yoav Goldberg. 2017. Neural network methods for natural language processing.

Zhouhan Lin, Minwei Feng, Cícero Nogueira dos Santos, Mo Yu, Bing Xiang, Bowen Zhou, and Yoshua Bengio. 2017. A structured self-attentive sentence embedding. *CoRR*, abs/1703.03130.