NLP Project: Natural Language Inference (NLI) Models, Developed By Fine-tuning Transformers

Or Shoham and Yaniv Roth

Department of Computer Science, IDC Herzliya

July 2021

Abstract

Natural language inference is the task of determining whether a “hypothesis” is true (entailment), false (contradiction), or undetermined (neutral) given a “premise”. In this project we tried to develop a model which uses pre-trained neural models (BERT based) that aims to achieve the best results as possible on a given dataset from Kaggle competition. In order to achieve it, we explored different Bert models, added more data (Mnli, Snli, Xnli), exmined training techniques and tried different fine-tuning NN-architectures. Our best model achieves 94.28% accuracy (top 7% in the competition). The code and the models are available at https://github.com/orsho/NLI-kaggle-competition-

1 Introduction

Natural Language Inference (NLI) task is one of the most important subsets of Natural Language Processing (NLP) which has seen a series of development in recent years. There are standard benchmark publicly available datasets like ”Stanford Natural Language Inference (SNLI) Corpus”, ”Multi-Genre NLI (MultiNLI) Corpus”, etc. which are dedicated to NLI tasks. Few state-of-the-art models trained on these datasets possess decent accuracy. In this project we will try to build model that get high score on that task. We will measure that by using Kaggle dataset and in addition we will aim to reach a high place in it’s competition.

2 What is NLI and why it’s important

Natural Language Inference which is also known as Recognizing Textual Entailment (RTE) is a task of determining whether the given “hypothesis” and “premise” logically follow (entailment) or unfollow (contradiction) or are undetermined (neutral) to each other[1]. For example, let us consider hypothesis as

“The game is played by only males” and premise as “Female players are playing the game”. The task of NLI model is to predict whether the two sentences are either entailment, contradiction, or neutral. In this case, it is a contradiction.
NLI is been used in many domains like banking, retail, finance, etc. It is widely used in cases where there is a requirement to check if generated or obtained result from the end-user follows the hypothesis. One of the use cases includes automatic auditing tasks. NLI can replace human auditing to some extent by comparing if sentences in generated document entail with the reference documents.

3 Kaggle Datasets

Kaggle has launched ”Contradictory My Dear Watson” challenge to detect contradiction and entailment in multilingual text. It has shared a training and a test datasets that contains 12120 and 5195 text pairs respectively.
This datasets contains textual pairs from 15 different languages – Arabic, Bulgarian, Chinese, German, Greek, English, Spanish, French, Hindi, Russian, Swahili, Thai, Turkish, Urdu, and Vietnamese. Sentence pairs are classified into three classes entailment (0), neutral (1), and contradiction (2).

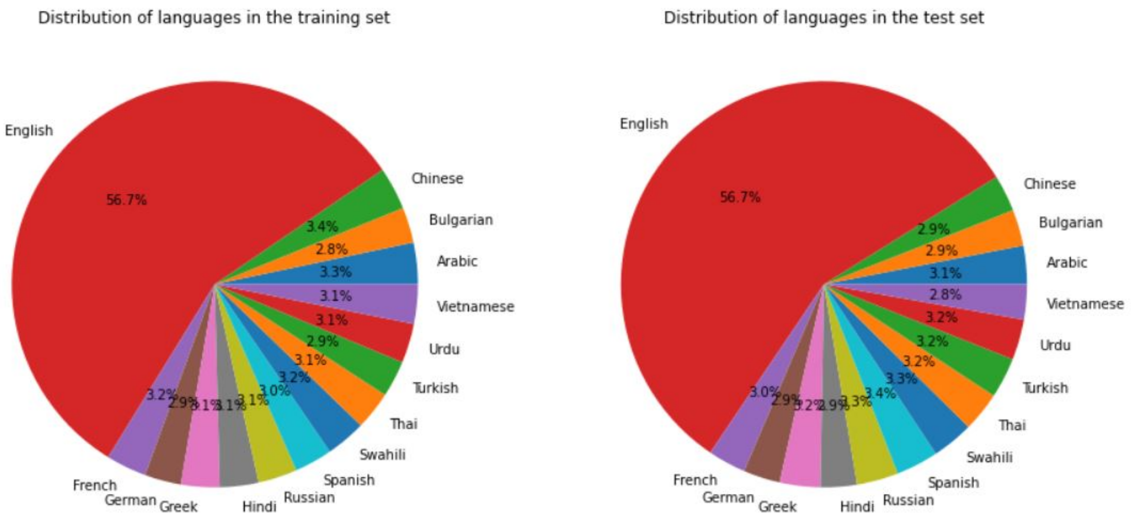


Figure 1: languages distribution of the train and test datasets

4 Brief summary of the process and experiments

Before we dive deeper into all our experiments, we will briefly summarize the process we performed and the key changes we made between each step until we found the model that brings the best results. The next table diagram present the process:

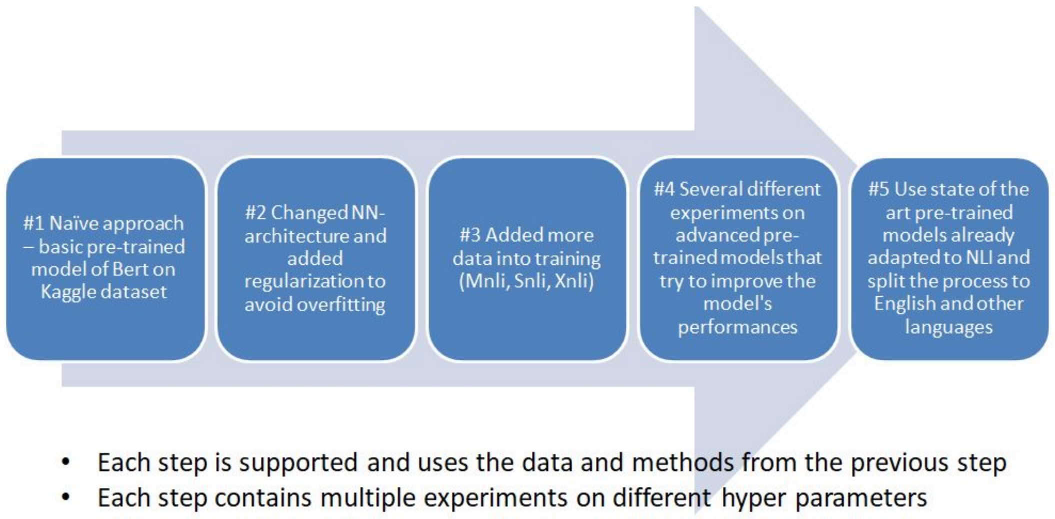

#1 Naïve approach – basic pre-trained model of Bert on Kaggle dataset
#2 Changed NN-architecture and added regularization to avoid overfitting
#3 Added more data into training (Mnli, Snli, Xnli)
#4 Several different experiments on advanced pre-trained models that try to improve the model's performances
#5 Use state of the art pre-trained models already adapted to NLI and split the process to English and other languages
Each step is supported and uses the data and methods from the previous step
Each step contains multiple experiments on different hyper parameters

Figure 2: summary of the process and experiments

5 Models and experiments

5.1 Experiment 1 - Naive approach

On our first model we decided to use naive approach in order to better understand the data and the task we are dealing with. We decided to use the pre-trained model named ”bert-base-multilingual-cased” which is a generic Bert model that got trained on 104 different languages. We trained our model on the Kaggle train dataset and fine tuning the last layer of the model. The table below present the accuracy we have got for different hyper parameters:

batch size\learning rate	lr=0.0001	lr=0.00001
size=8	54%	51%
size=32	55%	49%

* Our most important measure is accuracy since that how Kaggle competition evaluate the model performance but we used F1-score as our secondary measurement to improve our next models.

We can observe that that model get to over-fitting very fast since the train data contains approximately 10K samples. The next graph shows the result for the best hyper parameters we have got:

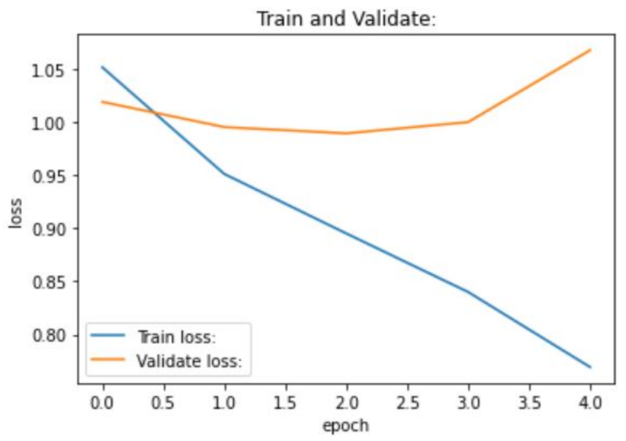


Figure 3: train loss vs validation loss

The main ways we considered to avoid that is:
1. Cross-validation
2. Add more data
3. Regularization
4. Early stopping
Clearly option number 4 isn't relevant in the case above since we get to over-fitting even before epoch number 1. We decided to try first regularization (next experiment) which suppose to help us later when we will add more complexity and more data to our model.

5.2 Experiment 2 - Regularization on Naive approach

On our second model, we used 'bert-base-multilingual-cased' pretrained model like experiment 1. We added to that model 2 more fully connected classification layers to predict the labels. The classification layers takes as input the last hidden vector of the [CLS] token after the transformers layers. The classification layers using dropout to prevent over-fitting of the model. In addition we used the AdamW[2] optimizer that yields better training loss and that the models generalize much better than models trained with Adam.

The classification layers are:
1. ReLU and Dropout on Linear layer 768 $\rightarrow$ 512
2. Linear layer 512 $\rightarrow$ 3 (the number of labels)

Accuracy for the model after 4 Epochs on batch size 32:

dropout value\learning rate	lr=0.0001
dropout=0.1	54.0%
dropout=0.4	54.2%
dropout=0.8	56.6%

On the next Graph, we can observe again that the model is over-fitting very quickly. The main cause for it, is probably an outcome of training so many parameters for such a small data. The uses of extra layers and dropout didn't solve it and only large dropout(0.8) achieved slightly better result from the previous experiment and help to postponed the over-fit.

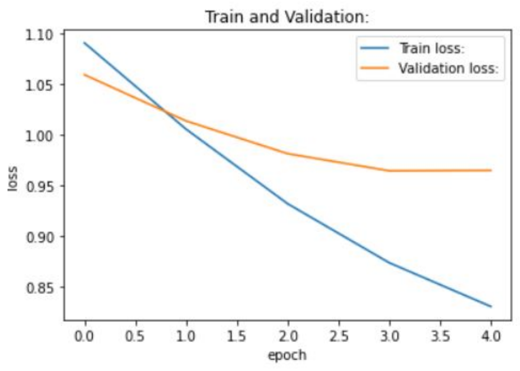


Figure 4: train loss vs validation loss of Ex.2

In the next models we will use bigger data sets in order to prevent the over-fitting.

5.3 Experiment 3 - Adding more data

In order to overcome the over-fitting problem we needed to add more data to the training process. NLI task is equipped with dedicated datasets. In our experiments we used:

1. The Stanford Natural Language Inference Corpus (SNLI)[3] - collection of 570k human-written English sentence pairs manually labeled for balanced classification with the label.
2. The Multi-Genre Natural Language Inference (MultiNLI) corpus[4] - one of the largest corpora available for the task of NLI, contains 433k english examples.
3. The Cross-lingual Natural Language Inference (XNLI) corpus[5] - crowd-sourced collection of 5,000 test and 2,500 dev pairs for the MultiNLI corpus.

The pairs are annotated with textual entailment and translated into 14 languages (112,500 examples total).

After training our model on the above datasets we discovered that there is no over-fitting and the added data solved the problem. We also found that despite that SNLI contains a lot of examples it didn't improve our model so much and only consumed a lot of computational resources, so we decided to use only MNLI and XNLI as our extra datasets.

The next table shows the results (average precision, recall, F1 for all the classes together and total accuracy) for different learning rate and different number of layers we unfroze on the training process (batch size 32, 5 epochs, optimizer=adamW):

unfrozen layers/ learning rate	**lr = 1e-4**				**lr = 1e-5**			
	precision	**recall**	**F1**	**accuracy**	**precision**	**recall**	**F1**	**accuracy**
1 layer	67.06	66.31	66.26	66.41	70.33	69.04	69.23	69.37
2 layers	74.15	72.87	72.88	73	71.35	71.14	71.15	71.2
3 layers	74.24	72.32	72.29	73.15	73.71	73.18	73.26	73.25

From the table above we can observe:
1. there is significantly better results when we trained the last 2 layers instead of only 1, but from 2 layers to 3 the changes are insignificant.
2. Learning rate of 1e-5 got better result when we unfroze 1 layer but lr of 1e-4 got better result when we unfroze 2 layers. Note that lr below 1e-5 had progressed very slowly and above 1e-4 didn't do any learning.
3. There isn't significant different between the average precision and recall. We did observed some differences when we separated the result of each class individually (Further explanations on that in the next experiment).

Since those results don't come close to the Kaggle competition top results, even after running more epochs on the dataset, we decided to try additional methods which discussed in the following experiments.

5.4 Experiment 4 - Different experiments on advanced pre-trained models

Experiment 4 including several experiments that try to improve the model's performances. In all of the next experiments, we used the 'XLM-roberta-large' pre-trained model[6] with the classification layers from Ex.2. XLM-roberta is a multilingual model that outperforms multilingual BERT by using a lot more data for training the model.

5.4.1 Experiment 4.1 - XLM-roberta-large

In this experiment we used data of 500,000 instances from MNLI, XNLI and Kaggle datasets. "XLM-roberta-large" model has 24 layers of transformers. We froze all the layers parameters except the last two.

Accuracy for the model after 4 Epochs:

batch size\learning rate	lr=0.0001
size=32	82.5%

5.4.2 Experiment 4.2 - Mean of hidden states vectors

In this experiment we have tried a variation of the last model. Instead of passing the classifier, the hidden vector that represents the [CLS] token, we have sent the mean of all the hidden states vectors. Using this method, we are feeding the classifier with an input expressing the whole sentences. Therefore, it may result with better outcome.

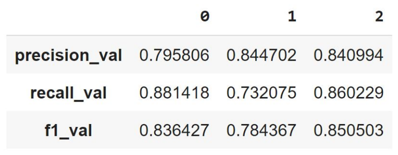
Accuracy for the model after 4 Epochs:

batch size\learning rate	lr=0.0001
size=32	81.45%

As we can observe, there is no improvement by using this method. The main reason, as we can understand by BERT paper, is that the hidden vector of the [CLS] token trained to give an expression for the all sentences by itself. This vector, is stronger than the mean vector because it uses weighted arithmetic mean while training. Therefore, it will have stronger ability of understanding the context of the sentences.

5.4.3 Experiment 4.3 - Binary classifier

Let us observe the performances of the model by analyzing the precision, recall and F1 values:



	0	1	2
precision_val	0.795806	0.844702	0.840994
recall_val	0.881418	0.732075	0.860229
f1_val	0.836427	0.784367	0.850503

Figure 5: analyze experiment 4.1 results

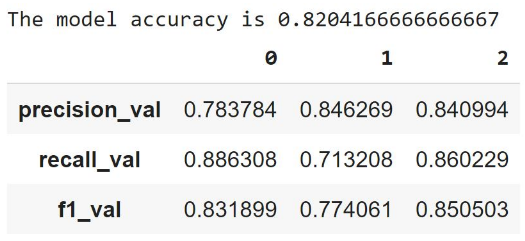
By analyze the F1 values, we can understand that the model is doing better job with predicting contradiction (label 2), than predicting the other labels - entailment (label 0) and neutral (label 1).

In this experiment we created and trained a binary classifier to predict between the labels - entailment and neutral. we performed the same model as Ex.4.1, but we will accept the prediction for only label 2. We will send again all the instances that classified with labels 0/1 to the trained binary classifier.

Accuracy for the binary classifier after 4 Epochs training:

batch size\learning rate	lr=0.0001
size=32	86.5%

The combined model performance:



```
The model accuracy is 0.8204166666666667
```

	0	1	2
precision_val	0.783784	0.846269	0.840994
recall_val	0.886308	0.713208	0.860229
f1_val	0.831899	0.774061	0.850503

Figure 6: binary model results

As we can observe in figure 6, there is no accuracy improvement, but only minor movements of the precision and recall values.

5.4.4 Experiment 4.4 - Reduction to MaskedLM task

Another way to improve our performances is by using reduction to another known task. In this experiment, we used Bert for MaskedLM model. The database used to train this model is MNLI with a twist. We concatenated another sentence to every instance, by this way - Premise + Hypothesis + 'The relation between the two sentences is entailment/neutral/contradiction'. After replacing the last token to [MASK] - we trained the model and got the next results for the kaggle validation set in the English language only:

batch size\learning rate	lr=0.0001
size=32	76.8%

While the XLM model from experiment 4.1 got 80.8% on the same validation set.

Now, we tried to combine to two methods by analyzing the probabilities of the predicted labels gathered by XLM model. If, for a given sentence pairs, the label that got the highest probability is close (by threshold we defined) to the probability of the second best label, we will test this instance again in the MaskedLM model, and use that prediction as the final one. Bottom line - we use a different task in cases where our main model gives a very close prediction between two labels

We've got the next results for the different threshold values:

Threshold	Number of instances	Accuracy success for those instances in XLM model only	Final accuracy after combining the methods
0.01	7	28.5%	80.8%
0.05	32	34.3%	80.62%
0.075	47	42.5%	80.58%
0.1	57	40.3%	80.62%
0.2	114	43.8%	80.33%

The results of this experiment as well didn't achieve any improvement.

5.5 Experiment 5 - Final model

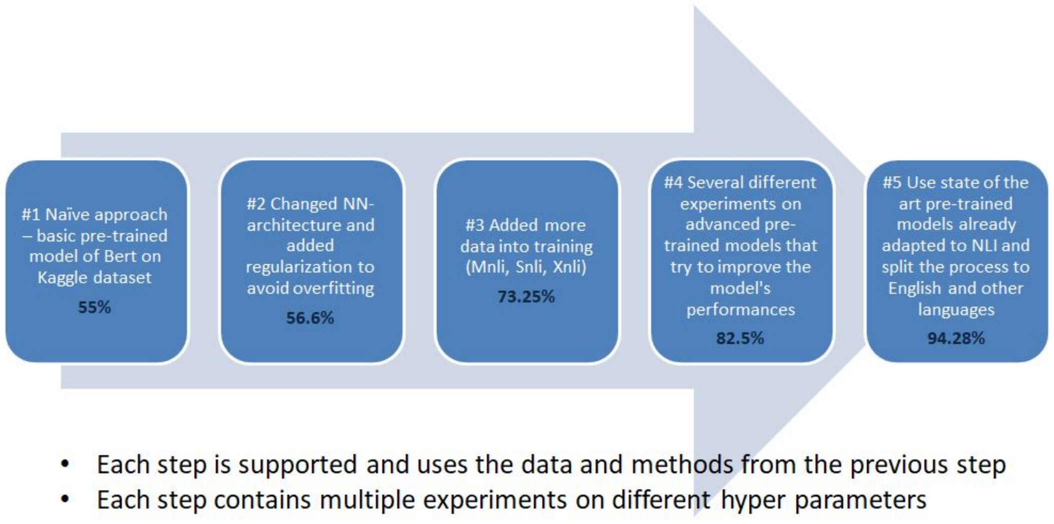
Our best model which gave the best result on Kaggle competition (accuracy on Kaggle test data) was achieved by performing as following:
We divided the train and test data to 2 datasets - English dataset and other languages dataset. For each dataset we performed different actions.

For English we chose the pre-trained model 'microsoft/deberta-v2-xlarge-mnli'[7] which trained on large amount of English data and received the best result on GLUE benchmark tasks. it contain 900M parameters and 24 layers. For that model we added 2 more classification layers built in experiment 2 and trained that on English sentences from Kaggle and XNLI datasets. We got 91.91% accuracy.

For the other languages we chose the pre-trained model 'joeddav/xlm-roberta-large-xnli'. This model is intended to be used for zero-shot text classification, especially in languages other than English. We decided not to train this model since we got great result without any training (97.6% accuracy).

If we combine the prediction for English and the prediction for other languages we got 94.28 accuracy on Kaggle which is top 7% percentile on the competition.

6 Summary of experimental results - best accuracy achieved in each step



#1 Naïve approach – basic pre-trained model of Bert on Kaggle dataset
55%
#2 Changed NN-architecture and added regularization to avoid overfitting
56.6%
#3 Added more data into training (Mnli, Snli, Xnli)
73.25%
#4 Several different experiments on advanced pre-trained models that try to improve the model's performances
82.5%
#5 Use state of the art pre-trained models already adapted to NLI and split the process to English and other languages
94.28%
• Each step is supported and uses the data and methods from the previous step
• Each step contains multiple experiments on different hyper parameters

Figure 7: summary of the process and experiments with the best accuracy achieved in each step

7 Conclusion

In this project we learned a lot about NLP and in particular about the NLI task. As we dived deeper we realized how many different options and methods there are for performing the task. It is still possible to improve the results we have got by training in a greater number of epochs, further improving the data, using cloud capabilities such as AWS, and exploring additional papers that can be integrated into the model we have built. Although, After many experiments, we think we got pretty good results that reached a high place in the competition combined with better understanding of the field.

References

[1] Marie-Catherine de Marneffe Nanjiang Jiang. *Evaluating BERT for natural language inference: A case study on the CommitmentBank*. 2019. ISBN 9781417642595. URL `https://aclanthology.org/D19-1630.pdf`.

[2] Frank Hutter Ilya Loshchilov. *Decoupled weight decay regularization*. 2019. ISBN 9781417642595. URL `https://arxiv.org/pdf/1711.05101.pdf`.

[3] Christopher Potts Samuel R. Bowman, Gabor Angeli and Christopher 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)*. 2015. ISBN 9781417642595. URL `https://nlp.stanford.edu/projects/snli/`.

[4] Sam Bowman Adina Williams, Nikita Nangia. *introduces the Multi-Genre Natural Language Inference (MultiNLI) corpus, a dataset designed for use in the development and evaluation of machine learning models for sentence understanding*. 2017. ISBN 9781417642595. URL `https://arxiv.org/abs/1704.05426`.

[5] Ruty Rinott Holger Schwenk Ves Stoyanov Alexis Conneau, Guillaume Lample. *XNLI: Evaluating Cross-lingual Sentence Representations*. 2018. ISBN 9781417642595. URL `https://arxiv.org/pdf/1809.05053.pdf`.

[6] Kartikay Khandelwal Alexis Conneau. *Unsupervised Cross-lingual Representation Learning at Scale*. 2020. ISBN 9781417642595. URL `https://arxiv.org/pdf/1911.02116.pdf`.

[7] Pengcheng He, Xiaodong Liu, Jianfeng Gao, and Weizhu Chen. Deberta: Decoding-enhanced bert with disentangled attention. In *International Conference on Learning Representations*, 2021. URL `https://openreview.net/forum?id=XPZIaotutsD`.