Transformers - Part I

Machine Learning for Natural Language Processing I
Attention! Required answers are marked in blue !

December 1, 2022

The goal of the report is to perform a name-entity recognition task. The task is trained by a seq2seq model, transformer to obtain desired results. The data is first downloaded, tokenised by pre-trained tokenisers and then fine-tuned with a transformer. Micro-F1 and Macro-F1 are reported in the end for performance evaluation on the testing dataset. The best model has micor-f1 0.9435 and macro-f1 0.4472.

1 Introduction

A transformer(Maxime, n.d.) is an extremely powerful seq2seq model in natural language processing (nlp) tasks, such as name-entity recognition. By introducing head weights, the model is extremely powerful in parallel training comparing to recursive neural network (RNN). This structure also solves the vanishing gradient problem in RNN. In addition, the model takes the proximity and ordinal properties into consideration by using the self-attention (Rasa, n.d.[a])(Rasa, n.d.[b]) and multi-head(Rasa, n.d.[c]) structures. This provides a better architecture for a better prediction on large datasets in a shorter training time.

2 Method

To train the model for name-entity recognition, the labelled data is first tokenised to adapt the pre-trained models.

2.1 Data

The data is loaded from polyglot. There are 39 languages supported in the package. In this task, it is required that only a language other than English can be used to train the model. Therefore, Polish is used for this name-entity recognition task. Accoding to the documentation of polyglot, there are three name entities, which are LOC, ORG, PER. And O is used for words that are not a name entity. The name-entity labels are then further converted into numbers for the seq2seq task later on. The corresponding label to number table is shown in Table 1. The data is not cleaned with further data cleaning steps.

Table 1: *Label to Number Conversion*

Label	Number
LOC	0
O	1
ORG	2
PER	3

To be able to train and test the models, the data is randomly split into 1000 corpora and 3000 corpora for training, and 2000 corpora for testing. Note that due to adequacy of the data, validation data is not needed and used in this task.

- LOC (Locations): cities, countries, regions, continents, neighborhoods, administrative divisions ...
- ORG (Organizations): sports teams, newspapers, banks, universities, schools, non-profits, companies, ...
- PER (Persons): politicians, scientists, artists, atheletes...

2.2 Tokenising

After splitting the data, the data is tokenised in order to be fed into the models. The tokeniser used in this task is *dkleczek/bert-base-polish-cased-v1* from HuggingFace. It provides the pretrained tokeniser and the transformer. For the pretrained tokeniser, there are cased tokeniser, which takes cased Polish sentences as inputs, and uncased tokeniser, which takes uncased

Table 2: *Model Summary*

Model	Description
Model 1	
data size	1000
embedding freeze	False
batch size	8
epoch	5
Model 2	
data size	3000
embedding freeze	False
batch size	8
epoch	5
Model 3	
data size	3000
embedding freeze	True
batch size	8
epoch	5

Polish sentences. The cased tokeniser is used for this task due to the better performance. As mentioned in the document, the cased tokeniser is trained without duplicate corpus, whole word masking, and cased characters. The uncased tokeniser cannot tokenise the corpora correctly because some Polish words cannot be easily separated without the case.

2.3 Training

After tokenisation, the pre-trained model from *dkleczek/bert-base-polish-cased-v1* is used for fine-tuning. The hyperparameters and data used in models are listed out in Table 2.

2.4 Evaluation - Micro-f1, Macro-f1

F1 score(Allwright, n.d.) is a metric of taking precision[1] and recall[2] into into consideration at the same time per class. F1-score is defined as Equation 1.

$$\begin{aligned} \text{f1-score} &= 2 \times \frac{\text{precision} \times \text{recall}}{\text{precision} + \text{recall}} \\ &= \frac{\text{TP}}{\text{TP} + \frac{1}{2}(\text{FP}+\text{FN})} \end{aligned} \tag{1}$$

Since f1-score is calculated per class, to calculate the aggregation of multi-class will become tricky. This is where micro-f1 and macro-f1 come into play. They are two different ways of aggregating multi-class f1-scores. Macro-f1 calculates the average of f1-scores of all classes as Equation 2. The equation shows that macro-f1 gives same weights to each class no matter how large the class is.

$$\text{macro-f1} = \frac{\text{sum(f1-score)}}{\text{number of classes}} \tag{2}$$

On the other hand, micro-f1 is defined by Equation 3. This equation is the same as the one for f1-score (Equation 1). However, the TP, FP and FN stands for the sum of the metrices for all classes.

$$\text{micro-f1} = \frac{\text{TP}}{\text{TP} + \frac{1}{2}(\text{FP}+\text{FN})} \tag{3}$$

This shows that f1-score views each observation points equally important. This might cause a bias of measurement with imbalanced datasets. With f1-score, larger classes with more observation points will have a larger impact on the f1-score.

For the task, both macro-f1 and micro-f1 will be used for the evaluation on the testing dataset. A comparison and discussion of the two metrices will be explained more in section 3.

[1]According to Layman definition, percision means, of all the positive predictions I made, how many of them are truly positive?

[2]According to Layman definition, percision means, of all the actual positive examples out there, how many of them did I correctly predict to be positive?

Table 3: *Result Summary*

Data Size	Embedding Freeze	micro-f1	macro-f1
1000	False	0.9299	0.3700
3000	False	0.9435	0.4472
3000	True	0.6209	0.1931

3 Results & Discussion

The results of the training summary is shown in Table 3. Micro-f1 and macro-f1 are two metrices used for the model evaluation. For both metrices, model 2 (3000 corpura without embedding freezing) performs the best.

3.1 Differences in Macro-f1 & Micro-f1

From Table 3, it is obvious that micro-f1 is always larger than macro-f1. This might be caused by the imbalanced testing dataset. Micro-f1 shows that the performance is quite well, but might just because the model performs really well on the class that has the most corpora. Judging from the low performance from macro-f1, it is highly likely that the model does not perform well on classes with few corpora.

3.2 Impact of Data Size

Comparing model 1 and model 2, they differ only in data size in the training data. The performance does

not differ a lot with model 2 performs slightly better than model 1. It is obvious that the more quality data the merrier. With 3000 corpora, the model is able to capture and learn more useful information, and thus, leading to better results.

3.3 Impact of Embedding Freezing

Model 2 and model 3 have all the conditions and hyper-parameters the same, except for whether the embeddings are freezed or not(Koren, n.d.). From the result, model 2 performs significantly better than model 3. This might because the embeddings used does not take the contents of the corpora in the training dataset into consideration. The connection (distance of words in the embedding vector space) is not correctly initialised for the task since the pre-trained model is used at the initialisation point of the task. Model 2 fixes the connection by slightly updating the embedding vectors via back-propagation. This might be the reason why model 2 performs better than model 3.

3.4 Model Initialising Warnings

While initialising BertForTokenClassification from the pre-trained model, *dkleczek/bert-base-polish-cased-v1*, the following warning shows up in the console.

```
Some weights of
BertForTokenClassification were not
initialized from the model checkpoint
at dkleczek/bert-base-polish-cased-v1
and are newly initialized:
['classifier.bias',
'classifier.weight'] You should
probably TRAIN this model on a
down-stream task to be able to use it
for predictions and inference.
```

As suggested by the warning, the weights and biases are not initialised. This is because the pre-trained model is used and the weights and biases are already uploaded/used in this case. So this can only be used for the down-stream task[3].

Bibliography

Allwright, Stephen (n.d.). *Micro vs Macro F1 score, what's the difference?* URL: https://stephenallwright.com/micro-vs-macro-f1-score/#:~:text=The%20key%20difference%20between%20micro,the%20two%20metrics%20is%20interpretation.. (accessed: 01.12.2022).

Koren, Vitaliy (n.d.). *Do we need to freeze embeddings when fine-tuning our LM?* URL: https://korenv20.medium.com/do-we-need-to-freeze-embeddings-when-fine-tuning-our-lm-c8bccf4ffeba. (accessed: 01.12.2022).

Maxime (n.d.). *What is a Transformer?* URL: https://medium.com/inside-machine-learning/what-is-a-transformer-d07dd1fbec04. (accessed: 01.12.2022).

Rasa (n.d.[a]). *Rasa Algorithm Whiteboard - Transformers & Attention 1: Self Attention*. URL: https://www.youtube.com/watch?v=yGTUuEx3GkA&list=PL75e0qA87dlG-za8eLI6t0_Pbxafk-cxb&index=10. (accessed: 01.12.2022).

– (n.d.[b]). *Rasa Algorithm Whiteboard - Transformers & Attention 2: Keys, Values, Queries*. URL: https://www.youtube.com/watch?v=tIvKXrEDMhk&list=PL75e0qA87dlG-za8eLI6t0_Pbxafk-cxb&index=11. (accessed: 01.12.2022).

– (n.d.[c]). *Rasa Algorithm Whiteboard - Transformers & Attention 3: Multi Head Attention*. URL: https://www.youtube.com/watch?v=23XUv0T9L5c&list=PL75e0qA87dlG-za8eLI6t0_Pbxafk-cxb&index=12. (accessed: 01.12.2022).

[3] Usually the pretrained model trained on a more general dataset. The pretrained models are then be used in fine-tuning stages for more specific task usages. The fine-tuning stages are called down-stream tasks.

How Angry Are You? - Part II

Machine Learning for Natural Language Processing I
Attention! Required answers are marked in blue !

December 4, 2022

The goal of the report is apply transformers on a regression task to identify the level of anger in the tweets. In this report, labeled data with anger intensity with tweets are first cleaned, tokenised, and then trained with transformers. On top of the transformer, a regression is used to adapt the quantitative label type.

1 Introduction

To understand the emotion behind tweets, various machine learning techniques can be used. Among all models, transformers(Transformers, n.d.) have a great chance of accomplishing this task. Transformers are able to capture the proximity properties of tweets and identifies the emotions behind them.

2 Method

The data is first preprocessed into tokenised tweets. The tokenised data is then sent to pretrained transformers for fine-tuning. The fine-tuning is done with *flaml* (Fast and Lightweight AutoML) for training through hyperparameter combinations.

2.1 Data

The data of the task consists of tweets with their corresponding intensity score angrer. Naturally, the intensity score is continuous between 0 and 1, corresponding to not angry at all and furious. The data is separated into training, validation and testing datasets.

2.2 Data Cleaning

Data Cleaning for the task is composed of the following steps:

- lowercasing
- removing html tags
- replacing emojis
- replacing urls, and users
- replacing swear words
- removing punctuation and numbers
- lemmatisation

Except for *replacing urls, and users*, *replacing emojis*, *replacing swear words*, other steps are self-explanatory and will not be further explained here in the report.

Replacing urls and users means to find out urls and users in the tweets, and replace them with the words *URL* and *USER* respectively.

Replacing emojis means to replace emojis with textual descriptions. Completely erasing emojis resuls in information lost. However, replacing the emojis with descriptions helps provide useful expressions to the tweets.

Replacing swear words means to identify swear words[1] in the tweets and replace them with the word "angry". This step along with the step *replacing urls and users* helps elimiate information that are not that useful such as the name of the user or the swear words, but somehow preserves the whole meaning and structure of the tweets.

After data cleaning, the tweets are tokenised by the pretrained tokeniser with *distilbert-base-uncased* model.

2.3 Training

Models used for this task is transformer for regression. As mentioned in Part I of the report, transformers are seq2seq models. With the sequential output, classification or regression can be built on top of that. In this task, obviously, on top of the transformer, a regression

[1]The swear words are identified by the python package, *better_profanity*.

Table 1: *Model 1-3 Summary*

Hyperparameter	Detail
Model 1	
batch size	16
epoch	3
learning rate	0.00005
weight decay	0
adam β_1	0.9
adam β_2	0.999
Model 2	
batch size	64
epoch	20
learning rate	0.00002
weight decay	0.01
adam β_1	0.9
adam β_2	0.999
Model 3	
batch size	64
epoch	10
learning rate	0.0001
weight decay	0.01
adam β_1	0.8
adam β_2	0.888

is used.

The first, second and third models are *distilbert-base-uncased* from HuggingFace with different hyperparameters. The hyperparameters are set through *TrainingArguments* from HuggingFace. The summary of the three models is shown in Table 4. The hyperparameters not listed in the table are default values. Details of the default values are listed in the website.

The fourth, fifth, sixth and seventh model is trained using *flaml*(HuggungFace, n.d.[c]) (Fast Library for Automated Machine Learning & Tuning) tool integrated in *Ray*(HuggungFace, n.d.[a])(HuggungFace, n.d.[b]). It is used for automatically fine-tuning hyperparameters and performing customised training process. The model used is still *distilbert-base-uncased* but with a wider range of hyperparameters being trained. The range of the hyperparameters are listed in Table 2. Model fourth and fifth are ray searched without the weight decay. And the best combination of hyperparameters is listed in Table 3.

3 Results & Discussion

The performances of each model is listed in **??**. The best performing model is model 7 . Therefore, model 7 is tested on the testing dataset for pearson r evaluation. The final person r for the task reaches 0.7619. The most different used for model 7 is the objective func-

Table 2: *Hyperparameters Tuning Range*

Hyperparameter	Range
epoch	[1, 64]
learning rate	[1e-6, 1e-4]
adam ϵ	[1e-9, 1e-7]
adam β_1	[0.8, 0.99]
adam β_2	[9999e-4, 98e-2]
weight decay	[0, 0.3]

Table 3: *Hyperparameters Tuning Range*

Hyperparameter	Range
Model 4	
epoch	15
batch size	64
learning rate	4.28e-6
adam ϵ	6.90e-8
adam β_1	0.84
adam β_2	0.99
weight decay	0
objective function	r2
Model 5	
epoch	50
batch size	8
learning rate	4.28e-6
adam ϵ	6.90e-8
adam β_1	0.84
adam β_2	0.99
weight decay	0
objective function	r2
Model 6	
epoch	50
learning rate	1.86e-5
adam ϵ	2.49e-8
adam β_1	0.81
adam β_2	0.99
weight decay	0.29
objective function	r2
Model 7	
epoch	50
learning rate	1.86e-5
adam ϵ	2.49e-8
adam β_1	0.81
adam β_2	0.99
weight decay	0.29
objective function	pearson r

Table 4: *Model 1-3 Summary*

Hyperparameter	Detail
Model 1	
validation loss	0.0188
Rmse	0.1371
Model 2	
validation loss	0.0185
Rmse	0.1360
Model 3	
training loss	0.0021
validation loss	0.0186
Rmse	0.1364
Model 4	
validation loss	0.0196
Rmse	0.1399
Model 5	
validation loss	0.0203
Rmse	0.1424
Model 6	
validation loss	0.4005
Rmse	0.6328
Model 7	
validation loss	0.0107
Rmse	0.1034

tion, which is setted as pearson r, while other models have the objective functions as squared-r. The two objective functions are quite different to model training. Pearson r(Laerd, n.d.) tends to catch the correlation between the predictions and the actual data, whereas squared-r captures the quality of the model training and is the square of pearson r. The reason for the better performance of setting the objective function to pearson r might because the squared r does not show negative signs, causing the loss lowering to have problems.

Bibliography

HuggungFace (n.d.[a]). *Hyperparameter Search with Transformers and Ray Tune*. URL: https://huggingface.co/blog/ray-tune. (accessed: 01.12.2022).

– (n.d.[b]). *Running Tune experiments with BlendSearch and CFO*. URL: https://docs.ray.io/en/latest/tune/examples/flaml_example.html. (accessed: 01.12.2022).

– (n.d.[c]). *Trainer*. URL: https://huggingface.co/docs/transformers/main_classes/trainer. (accessed: 01.12.2022).

Laerd (n.d.). *Pearson Product-Moment Correlation*. URL: https://statistics.laerd.com/statistical-guides/pearson-correlation-coefficient-statistical-guide.php. (accessed: 01.12.2022).

Transformers, Simple (n.d.). *Regression*. URL: https://simpletransformers.ai/docs/regression/. (accessed: 01.12.2022).