”Question Answering”
Natural Language Processing Assignment

MCSZN, Natural Language Processing

Abstract— **This report covers what we accomplished over the assignment dedicated to this Natural Language Processing course. Our purpose was to structure a coherent project and build an algorithm/model that could predict whether a message on forum board would be the best answer to a given question. The data we were provided consists in 750,000 lines of data in a csv file. Our report goes as follows:**

1) **Introduction**
2) **Prior Work**
3) **Data Pre-processing**
4) **Models and experiments**
5) **Conclusion**

We will first cover Natural Language Processing state of the art and the way we went through the project.

I. Introduction

A. Overview

Natural Language Processing is a branch of Artificial Intelligence which consists in computing designs that should enable algorithmic understanding of languages written or spoken by human beings. To this end, we use a specific subbranch of Artificial Intelligence: Machine Learning. A typical common attribute that unites different Machine Learning techniques is that the model we set up should be optimizing itself while feeding it with data. This is why the ”Big Data” revolution is getting so much credit. Big Data enable large scale machine learning programs to evolve and make better products. Precisely, an area that has won a considerate amount of attention due to great results on largely agreed hard artificial intelligence tasks is Deep Learning. Deep Learning can also be split into different specialties.

B. Neural Networks

We will further explain two different subgroup of deep-learning models but it is essential that the reader knows about the term Neural Networks (NN). Neural Networks are the continuation of the ”perceptron”. A basic perceptron α computes the following.

$$\alpha = \Sigma([W \times X] + \beta) \tag{1}$$

$$\alpha = \sigma(\alpha) \tag{2}$$

Where W and β are randomly generated vectors, X is the vector of inputs and sigma is the activation function chosen to output the summation and multiplication of W (weights) X (inputs) and β (biases). There are multiple activation functions that can be used but the most common are:

$$\tanh(x) = \frac{e^x - e^{-x}}{e^x + e^{-x}} \tag{3}$$

$$\sigma(x) = \frac{1}{1 + e^{-x}} \tag{4}$$

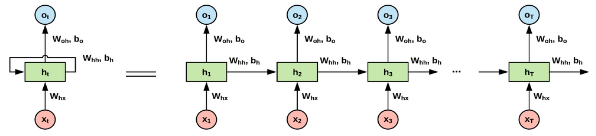
$$ReLU(x) = x \iff x \geq 0 \tag{5}$$

$$ReLU(x) = 0 \iff x < 0 \tag{6}$$

Neural Networks consist in an accumulation of ”layers” of these perceptron. Deep NN then simply mean that there are many layers for even better results on larger and more complex datasets. However, as one can see, Deep Neural Networks that are designed this way do not take in account order in the data and don’t have a general sense of context.

C. Recurrent Neural Networks

This can be a problem when analyzing time series data or any data that may have a temporal component. To tackle this problem NLP practitioners use Neural Networks that are call ”Recurrent Neural Networks” or RNN for short. RNNs consist in a cells that send information horizontally and vertically and can as well process the same input multiple times thus making it recurrent. Introducing this flow of information between inputs is a way to give context and pass an order definition. However, the problem with such systems is that the information is distributed to the cell's neighbor and they tend to forget words or events that are not far from each other.



Long Short Term Memory cells let data scientists create models that keep some amount of context and remember words that were at the beginning of the sentence. Our model will learn using gradient descent which is a typical optimization technique for Deep NN. Other techniques can be used such as Ordinary Differential Equations Networks but we won't be using them. Gradient Descent consists in updating the weights and biases following which way they influence the network to output a better prediction using partial derivatives.

II. Prior Work

A. SQuAD

SQuAD is the Stanford Question Answers Dataset. It consists in questions that are linked to Wikipedia articles which answer can be found in the article. In this dataset the article will be called the "context". We must use the context along with the question to return an answer. It is worth knowing that during the beginning of the project our group focused on the actual SQuAD dataset but latter was provided with a custom dataset. This allowed us to dive into the subject of Question Answering and write multiple scripts that would make further development faster and easier.
Our first idea was to start with the basics: importing the data, transform it for an algorithms. The two main ways of providing an answer are: giving the two numbers that correspond to the place where one can find the start and end of the question provided to the model. The second way is to make the model output the words making up the answer in the right order.

B. SQuAD experiments

We first read the data with pandas and then focused on going through every step of the modern Natural Language Processing process. Our first goal was to be able to convert words to individual ids and so we implemented our own "Count Vectorizer". We then created a Vocab Object that can store different types of information. In this object we can translate words to ids and ids back to words. We can also retrieve the vocabulary from the dataset and use it further for the embedding. Embedding is a technique that allows data scientists to mathematically process words and project them into multidimensional matrices were their position gives them meaning. For example the word pairs: "King/Queen" and "Man/Woman" will have the same mathematical mapping. To go from masculine to feminine one will only need the apply some particular mathematical operation and it will gender equivalent words. We decided to use transfer learning for this task knowing that the intelligence of a brand new model hardly can compete with models that have already been trained on generalized tasks. The BERT model seemed like an adequate choice for our Task. BERT or Bidirectional Encoder Representations from Transformers is a model that was trained by Google using an unbelievably high amount of data and the following learning process:

1) Take sentence
2) Mask some words
3) Try to predict the whole sentence

This process allowed BERT to have general knowledge about how sentences work and what meaning do words hold. We could then theoretically use BERT Embedding and than add an LSTM network to predict which words compose the answer. This however came at a large cost: development time.

C. SQuAD training

In order to finally train our SQuAD model we transformed our pandas Data Frame objects into PyTorch tensors, padded them. We used the BERT pre-trained weights translated into PyTorch by "HuggingFace" we finally created a Sequence to Sequence model that uses tokens as input and return tokens corresponding to the words we should be the ones to output. However, at this very crucial training period we were provided with the actual dataset and we decided to go on to the real project.

III. Data Pre-processing

A. *Exploratory Data Analysis*

As we saw it earlier, our data consists in a csv file containing 750,000 lines. It consists in data extracted from internet Forums. There are 18 fields. Lots of meta data can be extracted such as ”dates”, ”ids”, ”votes” and ”views”. But the core of the data consists in two fields that should influence a third one. These two fields are ”messages”, and ”titre” or ”title” in English. They are supposed to influence the ”is best answer” binary field. So our goal will be to make a model that classifies whether a message is the best answer to a given title/question.

B. *Pre-processing*

The first step is to read the data in order to be able to process it. One usually uses the python package ”pandas” in order to extract csv contained information with the ”pandas.read-csv()” function. However there are some real-world issues we encountered when trying to read the csv file. Actually, the csv reader had some trouble extracting information since some comas were included in the message field and the title field. This problem was tackled using a special function that would extract the data we need a block and store it in a json format that would be easier to work with. We developed these steps on scripts that would be small but effective returning the transformed data in a new json file. Having learned from our previous project we decide to use the Flair project in order to be able to focus on the model rather the transfer learning part. The Flair Team has made it very easy to use the BERT Embedding and our team took advantage of it. Eventually we also found it easier to simply train the model ”stochasticaly” (one example by one) rather than making up batches of variable sizes thus having to pad manually each input in the batch. One can find the specific ”input pre-processing” function in our utils.py file in the QnA folder.

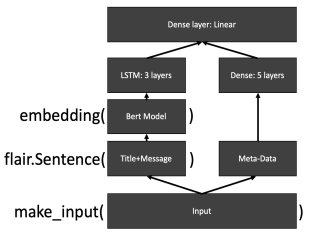
IV. Models and experiments

A. *The Concat.v1 model*

Our first approach was to single handily let BERT do what BERT does the best: Embedding. Our way of pre-processing the data would be simple: first concatenate all the data and the meta-data together, enclosing them into specific tags. e.g: ”[sot]Why should I delete system32?[eot][sos]For security reasons![eos][sov]420[eov]...”
With SOT being the tag for ”Start of Title”, EOS being ”End of Sentence” and SOV being ”Start of Votes” etc...
The problem with this method is that it required a lot of training time in order to get a learning curve that seemed to make sense. Concatenating did not turn out to be a good way of doing things. Another problem we encountered was that the data was extremely biased towards the label being 0. Our training data consisted in 96% of 0 labeled messages and 4% of 1 labeled messages. Our method to counter this effects were not to use SMOTE but simply setting 96% of chances of rejecting the training step if the message is 0 labeled. This eventually helped us in having a balanced dataset.

B. *The Hybrid.v2 model*

Since the concatenated model was not working our team decided to create a Hybrid model that could be able to process both meta-data and the sentences. Our thoughts on this was to create a hybrid binary classification model. In this model the Sentence processing part is given the BERT Embedded sentences and the Meta-Data processing one is treated by a shallow plain vanilla neural network. Each output is sent to a final layer were both choice is weighted and the neural network takes its final guess on the classification.

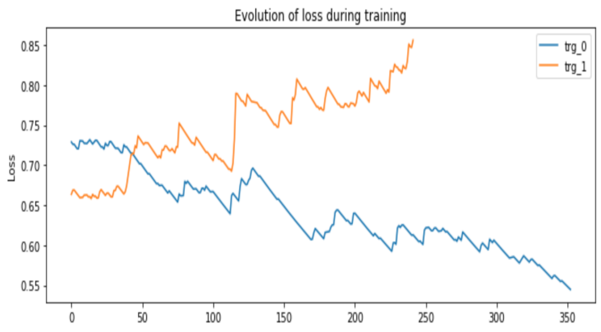


This model allowed us reaching a good training curve where accuracy for both labels. However one could wisely argue that it still needs some optimization in terms of number of layers and how we could take advantage of mini-batch to avoid slow stochastic training and finally reach good results fast. We use a Stochactic Gradient Descent optimizer with Nesterov Momentum but this could also be improved by performing learning rate decay and constantly checking for overfitting by evaluating our model on each optimization step.

V. Results

A. Learning

We trained our model using Google Colab GPUs in order to at least get some results out of our model experimentations. We were not able to use it long enough so as to obtain good results since Google colab only allows training for several minutes. We required hours of training. However we are content with our results that show the network is taking its firsts steps into learning what makes a forum message the best message to that forum. It is not easy for a human so it clearly is not easy for a robot. Here is a sneak peak of the learning curve we obtain.



The strokes that we can see correspond to different forum questions. When going to a new question the neural network has a hard time understanding what is going on but quickly adapts to the question and learns from it.

B. Further work

Further work for our project would be to actually train the model enough and evaluate it with mini-batch gradient descent and early-stopping. We would then be able to fine tune hyper-parameters such as the learning rate as well as other parameters such as the probability to train the model given its label. Eventually one could also decide on changing the actual architecture of the hybrid model so that it catches as efficiently as possible information from the dataset. We worked on a super file that enables the user to launch the model for training or predicting and lets the user decide whether to use a GPU and other important parameters for training. We also helped the user by making a setup.py file that downloads all the dependencies automatically when running. One can find all the code we used during this project in our github.

VI. CONCLUSIONS

This Natural Language Processing project was a big time investment but it helped us go through all the most important step of dealing with NLP tasks. We went from retrieving the data the right way to pre-processing it, creating models and iterating when they don't work. We used a hosted environment for training and learned how basic scikit-learn NLP utils function actually work by making our own classes. We also did some research around the newest NLP models and used transfer Learning to improve significantly the hypothetical results we could have gotten if we had some GPUs and servers at our disposal.

We would like to kindly thank Professor Ta who was the initiator of that project and without whom we wouldn't have focused so much on the basics of NLP.