IFT-7022 Natural Language Processing

Assignment 2

Ayoub. E *Oussama. A* *Marouane. M*

Presented to
Mr. Luc Lamontagne

Faculty of Science and Engineering
Laval University
Fall 2022

Task #1 – Question Classification -- MLP Networks & Word Embeddings

From the data/questions-train training file.txt and the data/questions-test.txt test file, train an MLP-like neural network models to **determine the class of a question**

For example, the question: *How tall is the Sears Building?* is the numerical quantity class *(QUANTITY)*.

Analysis A – FastText vs. Word2Vec

For this number, we decided to train 32 neural networks (MLPs) models to perform this multi-class classification task using pre-trained embeddings representation for our input words. 16 Networks were trained in an open vocabulary setting while 16 were trained in a closed vocabulary setting in order to test the accuracy of our model according to various assumption we could made when encountering unknown words in the training/test corpus. For example, in case of an open vocabulary, we decided that the embedding function will put a vector of 0s if an unknown word is seen.

Now for a specific 16 MLP configuration, 8 networks were tested on Word2Vec and 8 networks were tested using FastText as pretrained embedding functions. Those 8 networks correspond to two sets of 4 different configurations with different hidden layer size d = {50, 100, 200, 300}, one with a fusion embedding of type *average pooling* while the other with a fusion embedding of type *max pooling.* Those are essential to obtain a fixed size embedding in order to classify variable size text.

Now before presenting our results, let us see the difference between Word2Vec and FastText, which are both methods to learn word representations. Simply, a word embedding mapping W: words $\rightarrow \mathbb{R}^n$ is a parameterized function relating words in some language to high-dimensional vectors. Although FastText can be seen as an extension of Word2vec, they differ a lot as **Word2Vec uses whole words** while **FastText operates with characters** n-gram that when summed together can represent the words. Therefore, word2vec generates a vector for each word while FastText use the sum of the vectors of the n-grams for a specific word. This mean that Word2Vec learns vectors only for complete words found in the training corpus and FastText learns vectors for the n-grams that are found within each word, as well as each complete word. Hence FastText can be seen as computationally heavier than Word2Vec but can provide more information at a granular level when compared to Word2Vec.

	FastText		Word2Vec	
Pooling Type	**Average**	**Maximum**	**Average**	**Maximum**
Accuracy (hidden size d=50)	0.766	**0.718**	0.868	0.776
Accuracy (hidden size d=100)	0.778	0.706	**0.872**	**0.788**
Accuracy (hidden size d=200)	0.78	0.712	0.856	0.768
Accuracy (hidden size d=300)	**0.784**	0.71	0.866	0.786

Table 1: Efficacy of models with pretrained word embeddings FastText VS Word2Vec (via Gensim) (Closed Vocabulary) (Results on all test data)

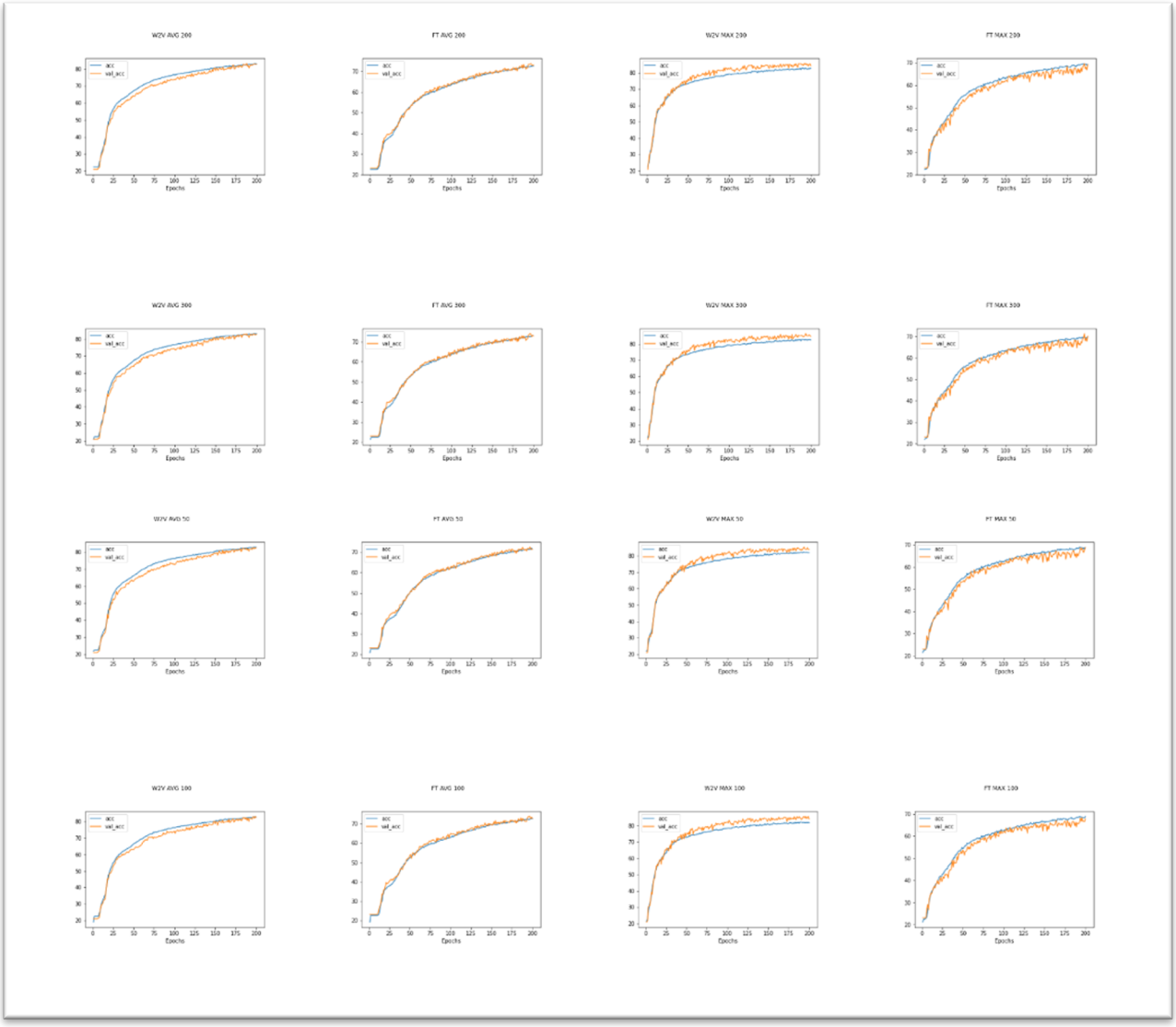

Figure 1: Training performances of the 16 MLP models in the closed vocabulary configuration

	FastText		Word2Vec	
Pooling Type	**Average**	**Maximum**	**Average**	**Maximum**
Accuracy (hidden size d=50)	0.764	0.726	**0.866**	**0.798**
Accuracy (hidden size d=100)	0.782	**0.736**	0.864	0.782
Accuracy (hidden size d=200)	0.786	0.73	0.858	0.786
Accuracy (hidden size d=300)	**0.792**	0.734	0.864	0.788

Table 2: Efficacy of models with pretrained word embeddings FastText VS Word2Vec (via Gensim) (Open Vocabulary) (Results on all test data)

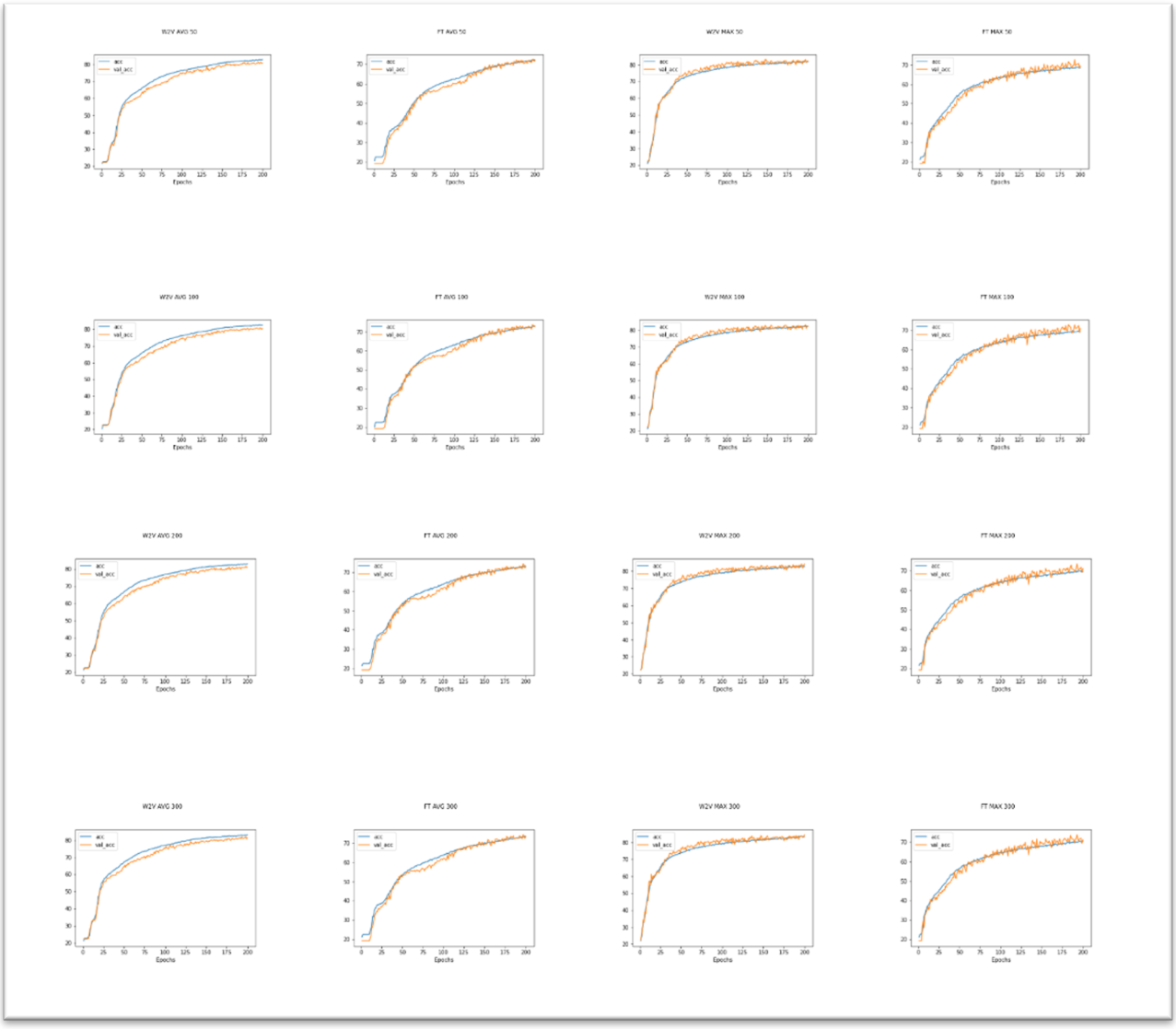

Figure 2: Training performances of the 16 MLP models in the open vocabulary configuration

Hyperparameters
Batch size = 16
Learning Rate = Default (Poutyne)
Optimizer: SGD
Activation Function: ReLU

Table 3: Common hyperparameters kept for MLPs

- **Analysis of our Results**

For our results, all models we obtained using an embedding size of d=300 and the common hyperparameters used are presented in the table 2. Now as we can see from our results, if we consider the assumption of a closed vocabulary setting, the MLPs using Word2Vec instead of FastText were definitely better when it comes to the classification accuracy on the test data. Also, the fusion embedding of type *average pooling* seems much better than the fusion embedding of type *max pooling*. For a comparison purpose, the best configuration for Word2Vec with an average pooling got 87.2% accuracy while the best configuration for FastText with an average pooling got 78.4% accuracy, which is a notable difference.

Now if we consider the assumption of an open vocabulary setting, the MLPs using Word2Vec got on average 0 or negative improvement to their accuracy while the MLPs using FastText got on average a notable boost of a few percent in their accuracy, which seems to justify the usage of FastText in a context of out of vocabulary words as the embedding vectors for a word are made from its character n grams even if the word in question doesn't appear in the training corpus.

Along with the accuracy table for the different configurations, we generated the charts for the training performances of the different models, where we can see a stable training across the epochs and no overfitting due to a stable validation accuracy. When it comes to the specific functions implemented for using the the FastText and Word2Vec embeddings, we downloaded the models on our local servers (Drive) in order to avoid difficulties when using the API calls at high frequency for requesting the models. Specific functions are available in the notebook for consultation.

Analysis B – Impact of the MLP hidden layer's size

In order to assess the impact of the MLP hidden layer size, we trained the different models across 4 different configuration of hidden layer size, which are d = {50, 100, 200, 300}. Looking at our results, it seems that our best classification accuracy across all modalities we obtained using a hidden size of d=100, which seems to make a trade-off between too much capacity for our model in the case of d=300 and the reduced capacity setting of d=50. This may be influenced by the relative difficulty of the non-linear mapping specific to this task made by our parameters and the capacity to find rapidly a suitable minimum when optimizing our objective function.

For recommendations purposes, the configuration that seems the more performant to us in this situation is the MLP with a hidden size of d=100, using Word2Vec as an embedding function and making use of the fusion embedding of type *average pooling.* The hyperparameters used for this configuration are listed in the table 3.

Task #2 — Proverb Completion with LSTMs and Spacy's embeddings

We resume the task of the first work, which consists in completing proverbs. For this work, you need to **build an LSTM neural network that will help you complete the incomplete proverbs by adding a word in the right place.**

For example, given the incomplete sentence: *help yourself, the *** will help you* and the list of candidate words [doctor, sky, teacher, whiskey], we want your **LSTM model to return the most likely sequence,** which is *help you, the sky will help you.*

- Training of the LSTM network:

Train the recurrent LSTM network as a neural language model using the data/proverbes.txt file.

• Each proverb corresponds to 1 line of the file. No normalization of words.
• Spacy's French embeddings are used.
• You must work with a **closed vocabulary**. It's up to you to see what this entails.
• We conduct the training with a **teacher forcing strategy.**
• For details on language models with recurring networks, see section 9.3 of Jurafsky and Martin.

- Complete the proverbs:

• The instructions are similar to those of the first work. **Completing a proverb involves replacing the stars (a hidden word) with one of the words on the list of choices.**

• One of the difficulties of this problem is how to **estimate the probability of a completed proverb. It's up to you to see how this can be done with an LSTM.**

• Use the data/test_proverbes.txt test file to evaluate the performance of the LSTM model. As this file does not contain the correct answers, **it is recommended that you create another file (with the solutions) that will allow you to make an automatic evaluation of the performance of the model. Please include this new file in your discount.**

A) LSTM Training

We proceeded to train the LSTM networks using the **Teacher Forcing Strategy**, hence making use of the ground truth from a prior time step as input. In other words, at each word position t of the input, the model takes as input the correct sequence of tokens and uses them to compute a probability distribution over possible next words so as to compute the model's loss for the next token. Then we move to the next word, ignoring the predictions of our model, and using instead the correct sequence of tokens to estimate the probability of token.

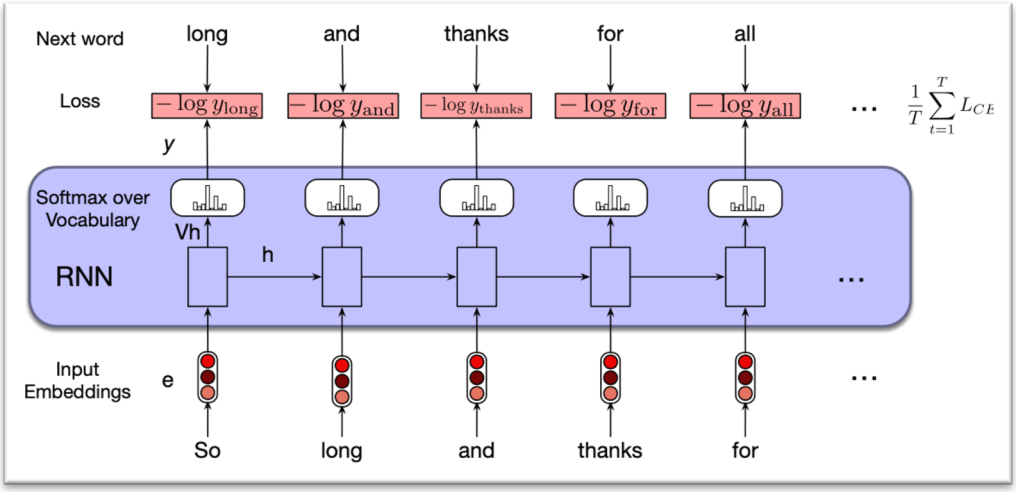

Figure 3: Forced Teaching Principle used to train a RNN (Extracted from the book by Jurafsky and Martin)

In order to train our model as a neural language model, we trained our networks to outputs the accurate next word from our training corpus. To achieve the learning, the weights in the network are adjusted to minimize the average Cross Entropy loss over the training sequence via Back Propagation through time (BPTT) and gradient descent.

The goal was to optimize the weights, so the probability of the actual next word is 1, and all the other entries are 0. We perform that via the Cross-Entropy Loss function, which computes a measure of the difference between a predicted probability distribution and the correct one. Therefore, the Cross-Entropy (C-E) loss for language modeling is determined by the probability the model assigns to the correct next word. Now at time t, the C-E loss is the negative log probability the model assigns to the next word in the training sequence: $LCE(y'_t, y_t) = -\log y'_t[w_{t+1}]$.

B) Inference Process

After the training phase completed where we used a corpus of text as training material in order to have a model predict the next word at each time according to it, we needed to estimate the probability of a completed proverb using the available possible words. In order to make use of all the information available to us and not only the words preceding the prediction (as we are not using bidirectional RNNs), we decided that for each inference, we needed to generate a

different sentence for each of the possible words available and compute their joint probability P(w1, ..., wn) of occurrence using the rule of conditional probability: P (w_1, ..., w_n) = P (w_1) * P (w_2|w_1) * ... * P (w_n|w_{n-1}, ..., w_1). Each of these conditional probabilities are given by the LSTM model when selecting probabilities from our softmax layer sequentially.

For example, in the test phase, we would take the incomplete sentence "*help yourself, the *** will help you"* with its corresponding list of candidate words *[doctor, sky, teacher, whiskey]* and we would generate 4 sentences and evaluate their likelihood according to our neural language model using the role mentioned above. The sentence with the highest probability will correspond to the choice we need to make in order to select the right proverb. In this context, it should be the sequence *help you, the sky will help you.*

Also, we made sure to work in a **closed vocabulary** setting, where the test set lexicon does not contain unknown words for our LSTM model. Additionally, the Spacy's French Embedding function was used to convert the input words into a series of word embeddings sequence $X = [x_1;...;x_t;...;x_N]$.

C) Results

Model Type	Accuracy (%)
LSTM	0.6739
Unigram	0.33
Bigram	0.61
Trigram	0.98

Table 4: Accuracy results as the proportion of successful test on the 46 total tests

Hyperparameters
Batch size = 2
Learning Rate = 0.1 (Decay)
Optimizer: SGD
Activation Function: ReLU

Table 5: Hyperparameters kept the LSTM Neural Language Model

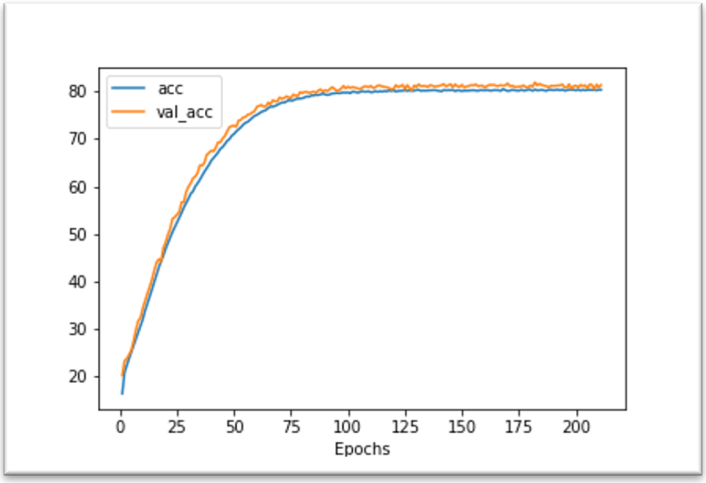

Figure 4: Training performances of the LSTM Network

D) Analysis

According to us, the LSTM model captured with a moderate level the language used in the proverb corpus, as the accuracy score (proportion of successful tests) we got was 67.39%, when trained on 220 epochs for 1h30. This may be explained by many factors. For example, for the relatively small size of our dataset and the short context windows related to this task, an LSTM network may suffer from slow learning and even overfitting for small datasets as this type of network is known for its ability of modeling long term dependencies.

When compared to results obtained from our previous work on n-grams, although the LSTM did better than bigram, it did not succeed in matching the excellent accuracy (98%) of the trigram model. This may be explained by the fact that the missing words in the corpus are really dependant on the a few words nearby, which in this case the trigram model should be very accurate while the LSTM will search much farther in the sequence length, which can generate confusion for the accurate word predicted by the model. On the other hand, n-gram models are impractical with long term dependencies, but our task here may be adapted to accurate prediction for a context size of a few words only.