TP deep learning for NLP

13 décembre 2024

Sam Vallet

Résumé

This report aims to present the implementation of two models : an MLP model using n-grams and an LSTM model, for word prediction. The goal is to compare the effectiveness of these two models. The purpose of this lab is not to achieve the best performance, but rather to implement these two models in order to understand their workings.

1 Introduction

Training models to predict the next words in a text is a common approach to obtaining models with a strong understanding of language. We will implement this using an MLP model and an LSTM model on a dataset from Google. This report will discuss the choice of hyperparameters and the performance of these two models, keeping in mind that the LSTM model is designed to have a better overall understanding of the text compared to the MLP with n-grams

2 Data Preprocessing

We perform several data preprocessing operations before starting the training of our models, which we will detail in this section.

Tokenization We begin by creating a dictionary that records all the words in the training dataset, `word_dic`. We convert all words to lowercase to avoid repetitions such as "Hello" and "hello". We also remove punctuation and add three tokens : "<UNK>" to represent any word not present in our dictionary, "<bos>" to mark the beginning, and "<eos>" to mark the end of sentences. We then create a function `Tokenizer` that takes string-type sentences as input and returns a list containing the ID of each word in the dictionary.

n-gram To prepare the n-gram, we create the function `generate_ngrams_with_padding`, which takes as input a sequence [word1, word2, word3] and, for n=2, produces the following tuples : [[word1], word2], [[word2], word3], where the idea is to provide the first word of the tuple to the model, and from that, it should predict the second word. However, with this setup, our model never has to guess word1. To correct this, we add n-1 occurrences of the <bos> token, so in our case the sequence would be : [[bos], word1], [[word1], word2], [[word2], word3], [[word3], eos].

target_pair For the LSTM model, we do not use n-grams. Instead, we implement a different approach. We create the function `target_pairs`, which, given a sequence [word1, word2, word3, word4] and a parameter n=2, returns : [[bos, bos], word1], [[bos, word1], word2], [[word1, word2], word3],...

3 Models

In this section, we will discuss our two language models and the choice of hyperparameters.

3.1 MLP n-gram

For the architecture of our MLP model, we begin with an embedding layer followed by a linear layer. We apply the ReLU activation function after the first linear layer, with dimensions (input, hidden). Then, we move on to an output linear layer with dimensions (hidden, 1). The input layer size is `dim_embedding` * n, where n is the n-gram size. We apply the ReLU activation function, and we will also test the model without the activation function.

Since training is quite slow on my machine, we are using relatively small dimensions for our model. The embedding dimension is set to 16, and the hidden layer dimension is 32. We will train for three epochs, with each epoch taking approximately five minutes to complete.

The choice of the number of epochs takes into account the long compilation time and the fact that we use a separate validation set that is not part of the training data. The results we obtained for the validation set after three epochs are as follows : epoch 1 : 7.16, epoch 2 : 7.06, epoch 3 : 7.00. There is not much difference between epoch 2 and epoch 3, and the results for epoch 3 are slightly worse. Therefore, we decided to keep 3 epochs.

3.2 LSTM

For our LSTM model, the architecture starts with an embedding layer, followed by the LSTM layer, and then a linear layer to produce the output dimension. We apply dropout before passing through the final linear layer. Initially, we will test using Random Dropout and later explore using Variational Dropout.

For the dimensions, we will use the same values as the MLP model to effectively compare these two models. We will also train for three epochs and use a batch size of 128. The reasoning behind this choice is that the results do not change significantly when using a batch size of 64, and the training time does not improve when increasing the batch size to 254. Therefore, we select the optimal batch size for model performance and execution time. For the both models, the loss function used for training is Cross-Entropy, and the optimizer is Adam.

4 Results

In this section, we will discuss the various results obtained.

Evaluation Metrics To evaluate our models, we use two metrics : perplexity, which measures the "surprise" of a model with respect to the predicted word, and cross-entropy loss. Perplexity is calculated using the `Perplexity` class. Both metrics are computed on a test dataset that is not part of the training set.

Learning Rate In our first experiment, we tested different learning rates.

	Loss	Perplexity
LSTM lr = 0.001	6.12	454
LSTM lr = 0.01	**6.06**	**428**
MLP lr = 0.001	7.02	1129
MLP lr = 0.01	8.14	3452

The best results are obtained with the LSTM model using a learning rate of 0.01. For the MLP model, the best learning rate is 0.001. The relatively high perplexity values can be explained by the small dimensions of the weight matrices in our models.

n-gram For the MLP model, we examine the effect of the value of n for the n-gram by testing $n = 2$ and $n = 3$ for both training and testing.

	Loss	Perplexity
MLP n = 2	**6.96**	**1054**
MLP n = 3	7.02	1129

The best values are obtained with $n = 2$.

Activation Function We tested the use of the ReLU activation function before the output, compared to not using ReLU.

	Loss	Perplexity
MLP ReLU	6.96	1054
MLP	6.33	563
LSTM ReLU	8.23	3753
LSTM	**6.06**	**428**

Both models perform better without ReLU. Using this activation function significantly increases their perplexity.

Dropout Finally, we compare the use of random dropout and variational dropout for the LSTM model.

	Loss	Perplexity
LSTM Random	6.06	428
LSTM Variational	**6.04**	**421**

The use of variational dropout slightly improves performance compared to random dropout.

5 Conclusion

To conclude, we observed that the LSTM model achieves better results than our MLP model using n-grams. This can be attributed to the LSTM's ability to better understand text by retaining contextual information over longer sequences.