Artificial Intelligence II - Homework 3

Ioanna Oikonomou

January 2023

Contents

1 Introduction

In this report I am going to explain my solution to the third assignment which requires creating a recurrent neural network (RNN) using Pytorch.

2 Assignment

2.1 Data Preprocessing - Glove Embeddings

Data preprocessing and data embedding is absolutely the same as in the second project. You can check the equivalent report for more information and explanations.

2.2 RNN Definition

In my implementation a bidirectional RNN is defined which supports both LSTM and GRU type cells. Additionally, it supports gradient clipping, skip connections and dropout. All of these hyperparameters are going to be discussed in the next section.

2.3 Hyperparameters

2.3.1 Number of layers

In the example that was shown in class 3 layers were used but for my model 4 layers gave optimal results.

2.3.2 Hidden size

The hidden size was left 64, as in the example that was shown in class. Around 50-70 were the best choices.

2.3.3 Cell types

Both LSTM and GRU type of cells had similar results with no remarkable differences. Also, both types gave better results when combined with gradient clipping and skip connections.

2.3.4 Gradient Clipping

When gradient clipping is used, the gradient vector has a norm of a maximum size. For me the max norm is 3. In this way, even if the loss is too big, the gradient stays normal. When not using gradient clipping, the accuracy is slightly worse.

2.3.5 Skip connections

In my implementation, skip connections can be used and one every two layers is skipped. That means that instead of the output of every layer, being the input to the next one, the output of one layer is stored and added to the output of the next one. This sum is, then, the input to the third layer.

Skip connections are vital in my implementation and have a huge impact on

the accuracy of the model. When run the model without skip connections, the accuracy is close to 0.5 and the loss close to 0.7.

2.3.6 Dropout probability

Dropout with probability around 0.5-0.8 was pretty effective. It helped increase the model's accuracy and reduce overfitting. In my model, I'm using probability = 0.8 as it gave the best results.

2.3.7 Loss function

Cross Entropy Loss function is used as it is suggested in the instructions of the project.

2.3.8 Optimizer

Adam optimizer is used as it is suggested in the instructions of the project.

2.3.9 Learning rate

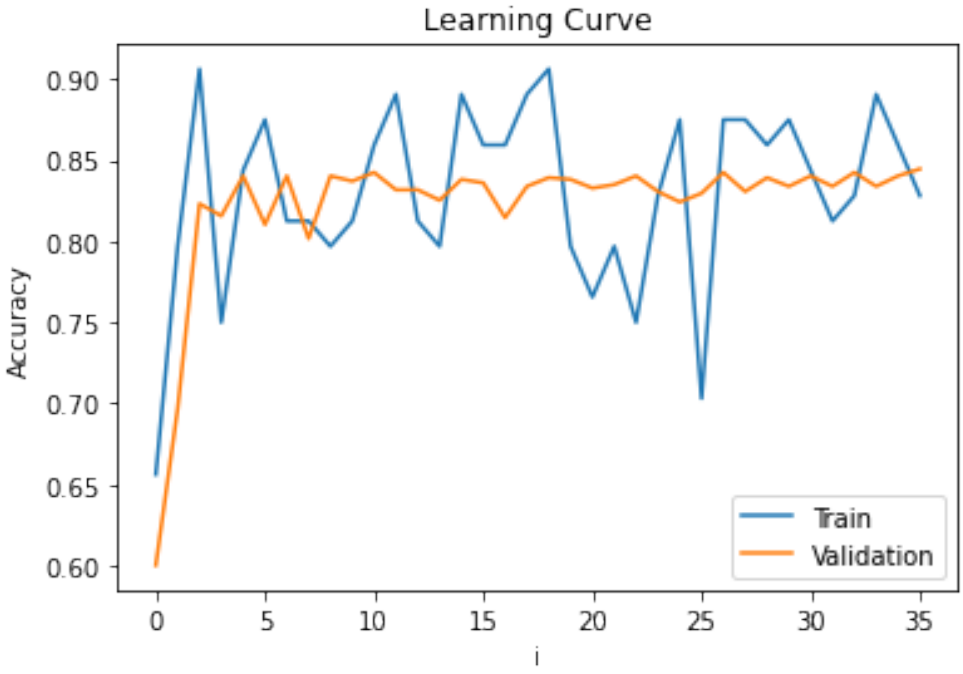
Learning rate is 0.01 as in the example that was shown in class. 0.001 was also tried but made the model too slow without remarkable differences in the results.

2.4 Results

2.4.1 Learning curves

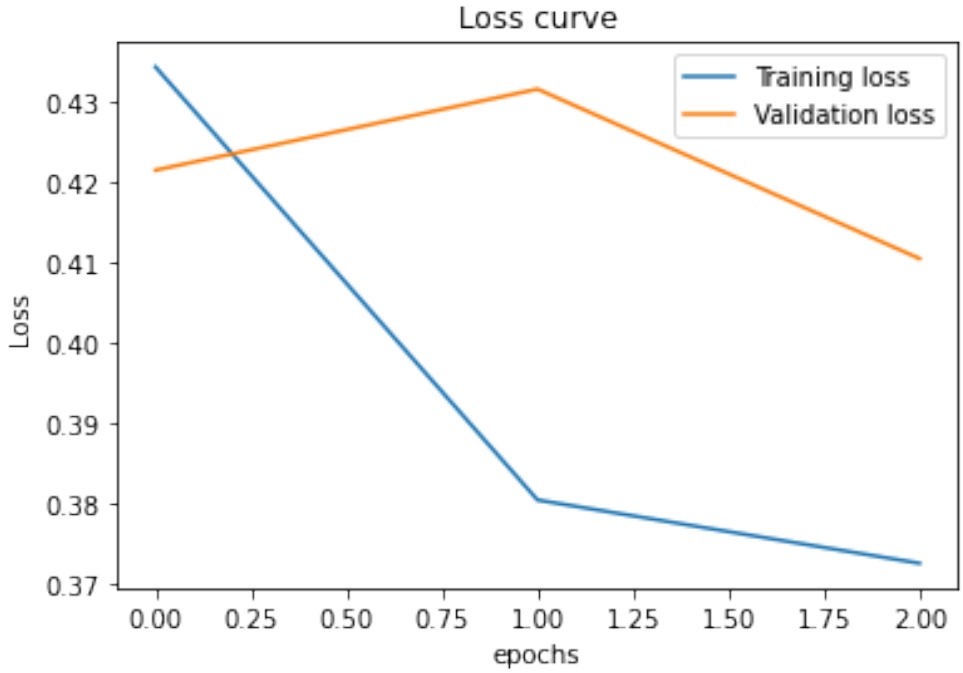
I made two kinds of learning curves, one based on the accuracy and one based on the loss. The results for the range of 3 epochs are depicted below:

Learning curve based on accuracy:



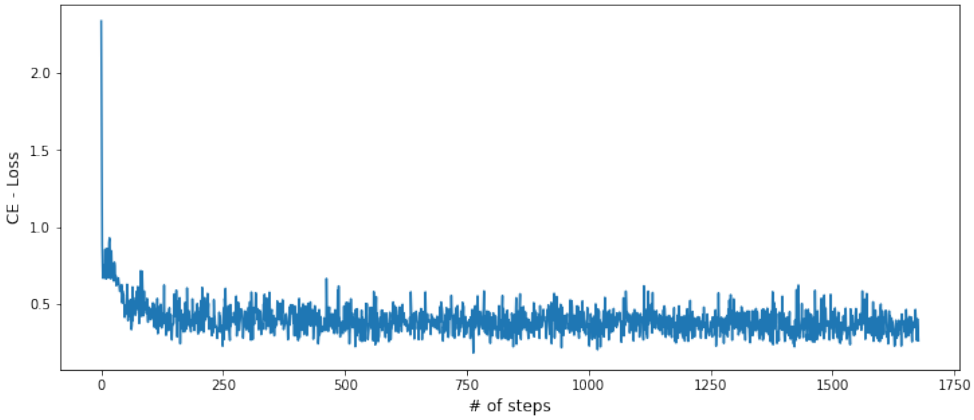
In this graph we can clearly see that the model is not overfitting or underfitting and the accuracy is close to 0.85 for the validation set.

Learning curve based on loss:



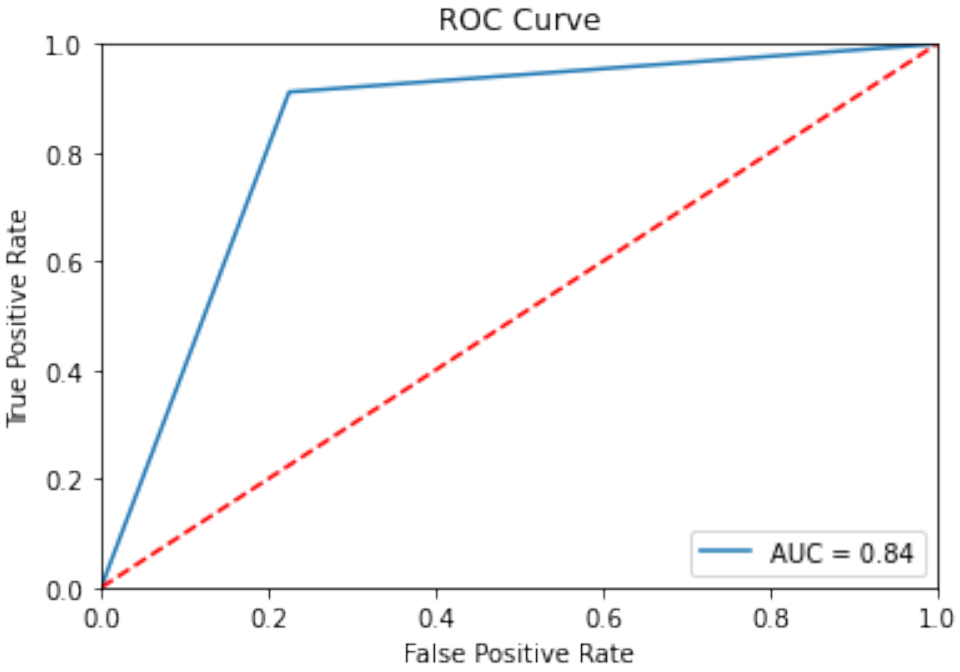
Here we can see that the loss for the validation set is more stable than for the training set.

Learning curve based on loss for a range of 1700 steps is shown below:



2.4.2 ROC Curve

The ROC curve that resulted from my model is the following:



We can see that the fit of the roc curve is pretty good and the AUC is also high enough.

2.5 Sklearn metrics

The classification report of my model is shown below. Here we can see the sizes of metrics like recall precision etc.

	precision	recall	f1-score	support
0.0	0.89	0.78	0.83	458
1.0	0.81	0.91	0.86	475
accuracy			0.84	933
macro avg	0.85	0.84	0.84	933
weighted avg	0.85	0.84	0.84	933

2.6 Comparison with previous projects

2.6.1 First project

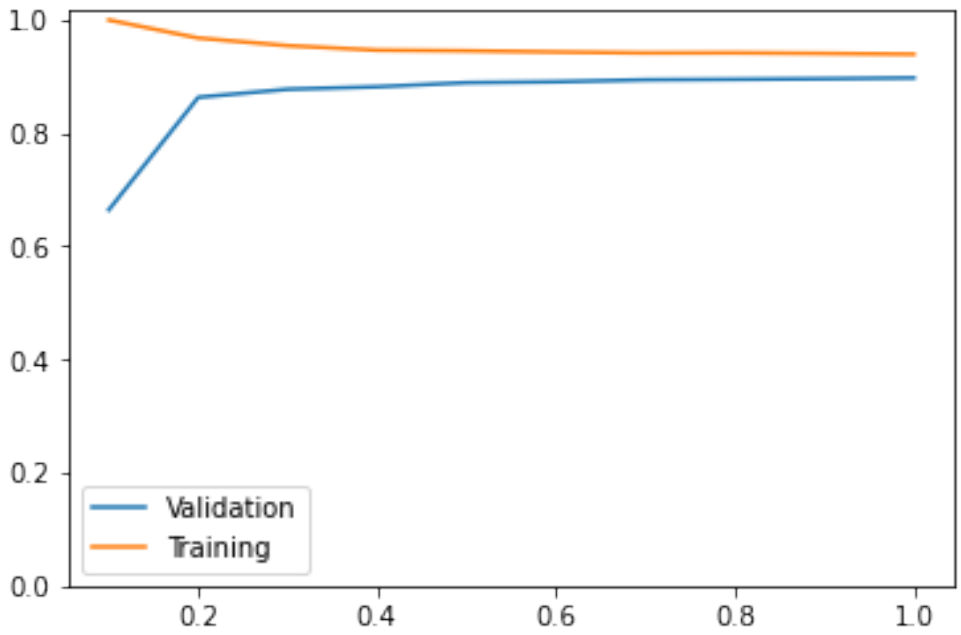
In the first project we were called to make a sentiment classifier using logistic regression. The equivalent classification report is shown below:

	precision	recall	f1-score	support
False	0.89	0.91	0.90	4410
True	0.91	0.89	0.90	4523
accuracy			0.90	8933
macro avg	0.90	0.90	0.90	8933
weighted avg	0.90	0.90	0.90	8933

accuracy = 90.14%

It is clear that the accuracy in the first project is better than the one of the RNN.

The learning curve of that model is shown below:

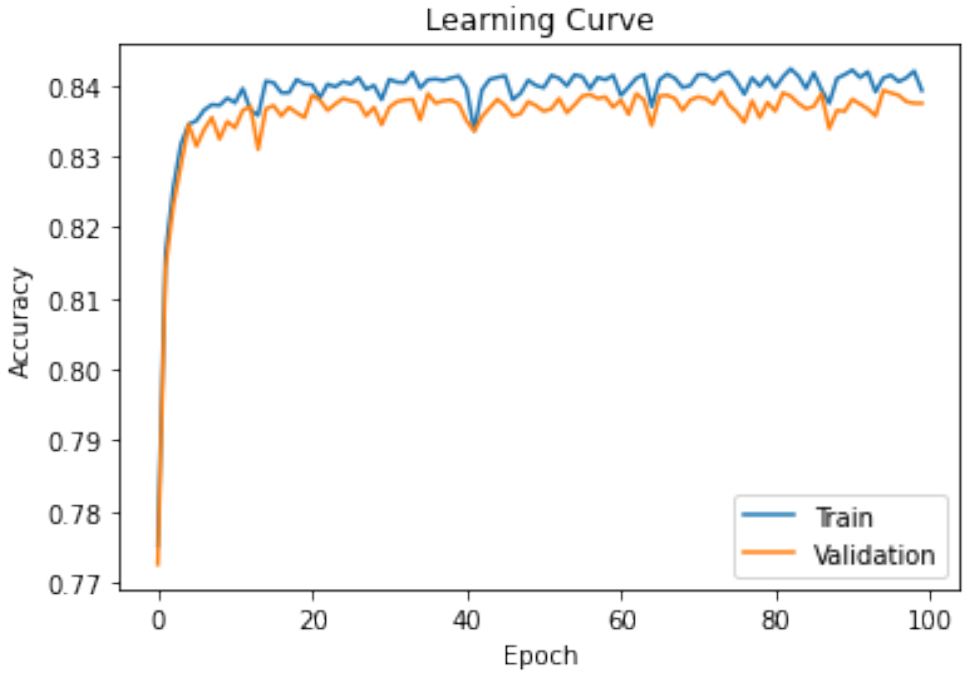


We can see that the shape of the learning curve is completely different than the one of the RNN and there is a slight overfitting here which we don't deal with in project 3.

2.6.2 Second Project

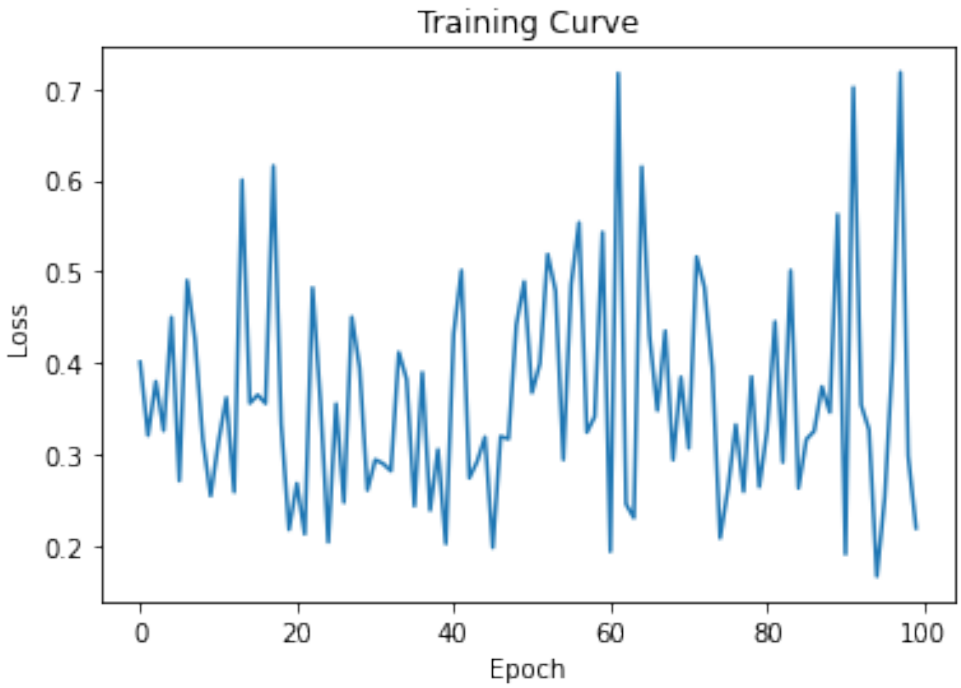
In the second project we were called to make a sentiment classifier using a Feed Forward Neural Network.

The equivalent learning curve is shown below:



Here we can see that the model is not overfitting or underfitting and the accuracy is basically the same as the one of the RNN. Additionally, the shape of the graph is more similar to the one we have in the third project than the one of the first project.

The loss curve of the FFNN model is shown below:



We can see that this loss curve has a different shape than the one of the RNN, but the size of the loss is approximately the same.

2.7 Testing

The last part of my code is a section you can use to test with the test set. I'm not sure this is what you needed in order to test it. I hope it helps :)

2.8 Sources

-Pytorch
-Lecture's slides from eclass
-Stack Overflow