LaTeX Template for ECE418 Final Project Report

Athanasios Papanikolaou 2337
atpapanikolaou@e-ce.uth.gr

September 23, 2020

Abstract

The world, in the last years, is all about efficiency, speed, faster and stronger computing power. One issue that is studied in the hope of perfection is predictions, all sorts of predictions, which can help in all kinds of topics and problems. One particular is predicting memory addresses like in the caches, in order to make every process accessing the cache a lot faster due to the prediction of the next address that the system needs to read or write to. In this project I tackle this prediction problem.

1 Introduction

In the frame of this final project, as explained in the abstract as well, my aim is to create a neural network capable of predicting the next address in a sequence of addresses. In simpler terms, given a numerical dataset, I try to predict the next number of the sequence, specifically in batches of 999 addresses.

2 Dataset

The dataset for training consists of 400000 addresses in hexadecimal form, as well as 50000 extra addresses (1 missing every 999 addresses) that is used for the prediction of these 50 addresses after training the model. As it will be explained later, I had to cut the first dataset down to a smaller size, specifically of 41000 samples due to ram overuse and crashes (almost 10% of the original dataset). In another more powerful and with more ram computer, a bigger dataset than this could be run in order to create better accuracy.

3 The beginning of experimentation and final result

3.1 The first attempt

The goal from the start was to create a LSTM model, one of the most suitable neural network techniques for problems of this kind. At first I started with just putting the scaled dataset (between 0 and 1 with MinMax Scaler), after turning it into decimal from hexadecimal, in batches in the X train and 1 address after the batch in y train (the dataset splits into these two), put it inside the lstm model, which consisted of an experimental number of hidden layers and neurons and one output layer of one neuron that predicted the outcome. This kind of thinking although failed. In the research and thinking I made I found out that a regression method like the one I described would fail by default. The aim is the accuracy, the ability to specifically predict the exact address I want, something that regression cannot provide. That is the reason that the project shifted to being one of the classification approach.

3.2 The training algorithm and predictions

The process is similar but changes in the details. First of all putting the scaled dataset as it is gave no results so, thanks to some excellent paperwork that

is listed below, I changed the dataset to consist only the difference between consecutive addresses because of its consistency of having numbers really close to each other in terms of value. In that way my dataset became full of 4s, -4s, 8s etc. The next big change is that I do not scale it anymore to (0,1) but rather onehot code y train, creating a matrix, this is where the classification part begins. Onehot coding it means that the rows remain the same but the columns become as many as the classes I want to predict are, with a 1 at the column the number belongs and 0 to all the other columns (classes). Of course this meant changes to the model of LSTM, different hidden layers, different shaped arrays and of course an output layer of not one neuron but of as many as my classes were. After making the training algorithm everything else is just using it and calculating the hit rate. I predict the values of the y test using X test (using the same coding method of onehot and difference transformation as I did with X/y train), reverse them from the onehot coding, reverse them from the difference transformation and then compare them to the actual values of the dataset, thus creating the famous hit rate.

3.3 Last part

In the last part of the program I do the same techniques again but this time for predicting 50 addresses in a new dataset. Nothing changes, I use the trained model with the transformed data and create 50 addresses, 49 in my situation to be exact thanks to a problem that could not be tackled, and then fill the empty cells in the excel document with these results.

4 The LSTM model in detail

The LSTM model is rather pretty simple with only two hidden layers. First of all, the first 'LSTM' hidden layer, that takes the data from X dataset, consists of 64 neurons, returns sequences, for the next hidden layer, and has a dropout of 0.2, something that is quite useful for large datasets like ours. The next hidden layer is exactly the same, with the difference that it does not return sequences of course. The output 'Dense' layer on the other hand, as explained before, has as many neurons as the classes we have, exactly 4271 with an activation function of 'softmax'. The model runs for only 32 epochs. It has been proven by testing that more neurons in each layer, additional layers and of course more epochs would result in better accuracy, all within a limit of course, too many epochs can result to no more training and just wasting time, and more neurons and layers could overfit the program and just produce wrong predictions. Unfortunately, training in my computer takes hours even for the 'simpler version' of the model and my computing power is limited (a lot of times the run just stops and crashes if it takes half a day of training) and that's why I had to resort to use only the numbers mentioned above. Additionally, the compilation of the model was done with a loss of 'categorical crossentropy', the optimizer 'adam' and printing the accuracy of the predictions. Lastly the fit of the model was done of course with the usage of X and onehot coded y. The batch size should be a number perfectly dividing the number of samples and 100 seems as a nice choice, a lot more or a lot less create problems or makes the whole training endless in terms of hours. Validation split is considered useful in this scenario so I use them as well and finally turn shuffle off as it would mess up with the sequence.

5 Related work and citation

Using the knowledge from the subject Machine Learning from Mr Ilias Xoustis and and the excellent paperwork of these sites:

- `https://arxiv.org/pdf/1803.02329.pdf`
- `https://towardsdatascience.com/lstm-for-google-stock-price-prediction-e35f5cc84165`
- `https://machinelearningmastery.com/time-series-forecasting-long-short-term-memory-network-python/?fbclid=IwAR0pxEAXWWG0lGXJeyPLc1BIMrcf9aSFHr-wDutQj8FoiRmo57I0HoSHvds`

- https://machinelearningmastery.com/how-to-use-an-encoder-decoder-lstm-to-echo-sequences-of-random-integers/
- https://machinelearningmastery.com/reshape-input-data-long-short-term-memory-networks-keras/

I got the information and knowledge needed to solve such a problem, from how to create such a model to how to shape the data in the way that would create the needed output. Of course it goes without saying that a lot of smaller questions about the algorithmic or programming nature of this problem were solved by sites like stackoverflow and many more, suggesting ideas about probable solutions, different ways of tackling an error etc. One such example is one-hot encoding, something unfamiliar to me until now. Some of the mentioned sites explained beautifully the concept of this coding as well as the decoding part of it.

6 Results and Discussion

In the end of the day the program actually works but it is sadly very slow, around 5 hours for the 'simpler version' of the model and if I did not end the 'more complicated one' with double the numbers of neurons per layer and double epochs it would take half a day, if it did not crash of course. However it is highly efficient for our level of coding, compared to some research and programs I found while researching that take this problem to another level. In more details the following picture shows the time the training algorithm run in order to get trained and the final hit rate is precisely 85.51388888888889%. Making the predictions of the split test dataset for the comparison to create the hit rate took some seconds at max, the same goes for the second set of predictions, the ones for the excel document. The second set of predictions is both inside the code and in the excel document. Also, apart from the time for training the model, the time to compile was some milliseconds. Lastly, inside the code there are extra comments about what every single cell does and how I handle some problems. Thank you for viewing my work!

```
Epoch 1/32
319/319 [==============================] - 555s 2s/step - loss: 5.2799 - accuracy: 0.1470
Epoch 2/32
319/319 [==============================] - 611s 2s/step - loss: 4.7641 - accuracy: 0.1682
Epoch 3/32
319/319 [==============================] - 608s 2s/step - loss: 4.6433 - accuracy: 0.2122
Epoch 4/32
319/319 [==============================] - 610s 2s/step - loss: 4.5133 - accuracy: 0.2310
Epoch 5/32
319/319 [==============================] - 589s 2s/step - loss: 4.3903 - accuracy: 0.2500
Epoch 6/32
319/319 [==============================] - 590s 2s/step - loss: 4.2692 - accuracy: 0.2798
Epoch 7/32
319/319 [==============================] - 591s 2s/step - loss: 4.1439 - accuracy: 0.3101
Epoch 8/32
319/319 [==============================] - 592s 2s/step - loss: 4.0229 - accuracy: 0.3396
Epoch 9/32
319/319 [==============================] - 595s 2s/step - loss: 3.9081 - accuracy: 0.3615
Epoch 10/32
319/319 [==============================] - 595s 2s/step - loss: 3.7614 - accuracy: 0.3974
Epoch 11/32
319/319 [==============================] - 598s 2s/step - loss: 3.6572 - accuracy: 0.4214
Epoch 12/32
319/319 [==============================] - 600s 2s/step - loss: 3.5483 - accuracy: 0.4364
Epoch 13/32
319/319 [==============================] - 602s 2s/step - loss: 3.4319 - accuracy: 0.4544
Epoch 14/32
319/319 [==============================] - 600s 2s/step - loss: 3.3266 - accuracy: 0.4678
Epoch 15/32
319/319 [==============================] - 594s 2s/step - loss: 3.2221 - accuracy: 0.4783
Epoch 16/32
319/319 [==============================] - 592s 2s/step - loss: 3.1331 - accuracy: 0.4826
Epoch 17/32
319/319 [==============================] - 594s 2s/step - loss: 3.0338 - accuracy: 0.4916
Epoch 18/32
319/319 [==============================] - 599s 2s/step - loss: 2.9647 - accuracy: 0.4939
Epoch 19/32
319/319 [==============================] - 593s 2s/step - loss: 2.8756 - accuracy: 0.4992
Epoch 20/32
319/319 [==============================] - 596s 2s/step - loss: 2.8221 - accuracy: 0.5030
Epoch 21/32
319/319 [==============================] - 597s 2s/step - loss: 2.7228 - accuracy: 0.5107
Epoch 22/32
319/319 [==============================] - 598s 2s/step - loss: 2.6518 - accuracy: 0.5155
Epoch 23/32
319/319 [==============================] - 594s 2s/step - loss: 2.5841 - accuracy: 0.5230
Epoch 24/32
319/319 [==============================] - 599s 2s/step - loss: 2.5314 - accuracy: 0.5272
Epoch 25/32
319/319 [==============================] - 601s 2s/step - loss: 2.4768 - accuracy: 0.5320
Epoch 26/32
319/319 [==============================] - 598s 2s/step - loss: 2.4162 - accuracy: 0.5366
Epoch 27/32
319/319 [==============================] - 600s 2s/step - loss: 2.3545 - accuracy: 0.5409
Epoch 28/32
319/319 [==============================] - 599s 2s/step - loss: 2.2988 - accuracy: 0.5502
Epoch 29/32
319/319 [==============================] - 601s 2s/step - loss: 2.2662 - accuracy: 0.5515
Epoch 30/32
319/319 [==============================] - 601s 2s/step - loss: 2.2021 - accuracy: 0.5605
Epoch 31/32
319/319 [==============================] - 602s 2s/step - loss: 2.1602 - accuracy: 0.5650
Epoch 32/32
319/319 [==============================] - 607s 2s/step - loss: 2.1135 - accuracy: 0.5694
```