CE418: Final Project

Fivos Tzavellos (1956)
ftzavellos@uth.gr
Efstathia Koutougera (2249)
ekoutougera@uth.gr

University of Thessaly — September 23, 2020

Brief Code Description

The problem was to predict the next number after a sequence of 999 numbers. We solve it by turning the numbers from hexadecimal form into decimal, separating the train dataset into batches of 999 numbers and predicting the next one, then sliding the 'window' one value to the left, meaning that we take the values from the 1st one until the 999th one, then predict the 1000th, then take the values from the 2nd one until the 1000th one and predict the 1001st and so on [4]. These values are not the real values present in the training dataset that was given to us, but rather the differences between the previous value and the current one, an idea derived from here [1].
At first we tried using regression [3], but the results were never on-point, we were just approaching the value without ever getting there. So we decided to try using classification so we can get some on-point results, some real hits on the datasets.
We used one-hot encoding in order to make the classification (idea derived from here [2].

Model Architecture

We used an LSTM (Long Short Term Memory) neural network with 2 hidden layers, 128 neurons on each layer, a dropout of 20% and softmax as the activation function. We got 5958 values as output, because that was the number of classes we have (the unique differences in the dataset), and the output comes out one-hot encoded.

```
#We have 2 hidden layers, 128 neurons in each hidden layer, a dropout of 20% and softmax as our activation function

model = keras.Sequential()
model.add(LSTM(128, return_sequences=True, input_shape=(X_train.shape[1], X_train.shape[2]), dropout = 0.2 ))
model.add(LSTM(128, return_sequences=False, dropout = 0.2  ))
model.add(Dense(5958, activation = 'softmax'))
```

Training Algorithm

We use categorical cross entropy as our loss function , accuracy as a metric and adam as the optimiser.

```
start = time()
model.compile(loss='categorical_crossentropy', optimizer='adam', metrics = ['accuracy'])
print ('compilation time : ', time() - start)
```

```
compilation time :  0.023860931396484375
```

Model Fit

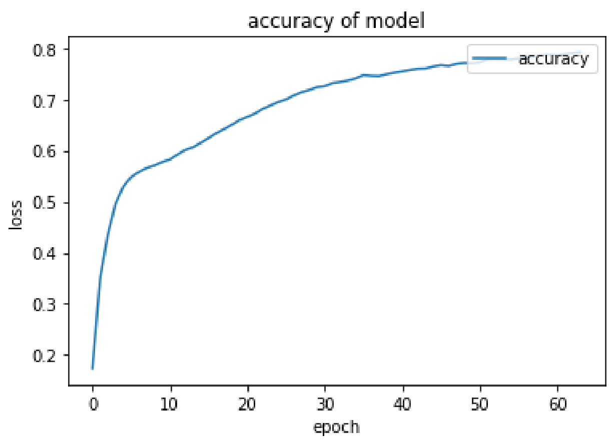
Our fit method run for 64 epochs with a batch size of 100, 20% used for validation and we set the shuffle parameter to false because we didn't want the values to get shuffled. The total time was 10048 seconds , or 2 hours and 48 minutes approximately.

```
Epoch 1/64
400/400 [==============================] - 158s 396ms/step - loss: 4.8034 - accuracy: 0.1733 - val_loss: 5.7637 - val_accuracy: 0.1735
Epoch 2/64
400/400 [==============================] - 157s 392ms/step - loss: 3.8964 - accuracy: 0.3505 - val_loss: 5.8487 - val_accuracy: 0.1504
Epoch 3/64
400/400 [==============================] - 158s 394ms/step - loss: 3.5249 - accuracy: 0.4362 - val_loss: 5.8609 - val_accuracy: 0.1713
Epoch 4/64
400/400 [==============================] - 157s 393ms/step - loss: 3.2497 - accuracy: 0.4967 - val_loss: 5.8529 - val_accuracy: 0.1562
Epoch 5/64
400/400 [==============================] - 157s 392ms/step - loss: 3.0269 - accuracy: 0.5294 - val_loss: 5.6316 - val_accuracy: 0.1599
Epoch 6/64
400/400 [==============================] - 155s 389ms/step - loss: 2.8651 - accuracy: 0.5479 - val_loss: 5.4978 - val_accuracy: 0.1645
Epoch 7/64
400/400 [==============================] - 157s 393ms/step - loss: 2.7207 - accuracy: 0.5579 - val_loss: 5.1757 - val_accuracy: 0.2022
Epoch 8/64
400/400 [==============================] - 157s 392ms/step - loss: 2.5971 - accuracy: 0.5656 - val_loss: 4.9532 - val_accuracy: 0.2319
Epoch 9/64
400/400 [==============================] - 156s 389ms/step - loss: 2.5074 - accuracy: 0.5706 - val_loss: 4.8233 - val_accuracy: 0.2808
Epoch 10/64
400/400 [==============================] - 156s 390ms/step - loss: 2.4179 - accuracy: 0.5774 - val_loss: 4.6862 - val_accuracy: 0.2895
Epoch 11/64
400/400 [==============================] - 156s 391ms/step - loss: 2.3328 - accuracy: 0.5824 - val_loss: 4.5753 - val_accuracy: 0.3190
Epoch 12/64
400/400 [==============================] - 155s 388ms/step - loss: 2.2478 - accuracy: 0.5923 - val_loss: 4.3544 - val_accuracy: 0.3541
Epoch 13/64
400/400 [==============================] - 155s 388ms/step - loss: 2.1581 - accuracy: 0.6011 - val_loss: 4.2293 - val_accuracy: 0.3910
Epoch 14/64
400/400 [==============================] - 156s 390ms/step - loss: 2.0801 - accuracy: 0.6064 - val_loss: 4.1007 - val_accuracy: 0.4148
Epoch 15/64
400/400 [==============================] - 156s 390ms/step - loss: 1.9983 - accuracy: 0.6151 - val_loss: 3.9990 - val_accuracy: 0.4468
Epoch 16/64
400/400 [==============================] - 155s 388ms/step - loss: 1.9258 - accuracy: 0.6242 - val_loss: 3.9430 - val_accuracy: 0.4845
Epoch 17/64
400/400 [==============================] - 156s 390ms/step - loss: 1.8553 - accuracy: 0.6338 - val_loss: 3.8822 - val_accuracy: 0.4990
Epoch 18/64
400/400 [==============================] - 156s 389ms/step - loss: 1.7899 - accuracy: 0.6417 - val_loss: 3.8328 - val_accuracy: 0.5233
Epoch 19/64
400/400 [==============================] - 155s 388ms/step - loss: 1.7249 - accuracy: 0.6502 - val_loss: 3.7314 - val_accuracy: 0.5598
Epoch 20/64
400/400 [==============================] - 155s 388ms/step - loss: 1.6727 - accuracy: 0.6593 - val_loss: 3.7150 - val_accuracy: 0.5759
Epoch 21/64
400/400 [==============================] - 156s 391ms/step - loss: 1.6228 - accuracy: 0.6656 - val_loss: 3.7015 - val_accuracy: 0.6068
Epoch 22/64
400/400 [==============================] - 156s 391ms/step - loss: 1.5713 - accuracy: 0.6721 - val_loss: 3.6703 - val_accuracy: 0.6222
Epoch 23/64
400/400 [==============================] - 156s 390ms/step - loss: 1.5151 - accuracy: 0.6813 - val_loss: 3.6144 - val_accuracy: 0.6286
```

```
Epoch 24/64
400/400 [==============================] - 156s 390ms/step - loss: 1.4637 - accuracy: 0.6880 - val_loss: 3.6064 - val_accuracy: 0.6406
Epoch 25/64
400/400 [==============================] - 156s 390ms/step - loss: 1.4130 - accuracy: 0.6948 - val_loss: 3.5404 - val_accuracy: 0.6618
Epoch 26/64
400/400 [==============================] - 155s 389ms/step - loss: 1.3758 - accuracy: 0.6999 - val_loss: 3.5742 - val_accuracy: 0.6681
Epoch 27/64
400/400 [==============================] - 157s 393ms/step - loss: 1.3303 - accuracy: 0.7075 - val_loss: 3.4931 - val_accuracy: 0.6797
Epoch 28/64
400/400 [==============================] - 159s 398ms/step - loss: 1.3016 - accuracy: 0.7139 - val_loss: 3.4667 - val_accuracy: 0.6911
Epoch 29/64
400/400 [==============================] - 159s 397ms/step - loss: 1.2684 - accuracy: 0.7183 - val_loss: 3.5041 - val_accuracy: 0.6912
Epoch 30/64
400/400 [==============================] - 158s 396ms/step - loss: 1.2316 - accuracy: 0.7239 - val_loss: 3.5062 - val_accuracy: 0.7061
Epoch 31/64
400/400 [==============================] - 159s 398ms/step - loss: 1.2176 - accuracy: 0.7259 - val_loss: 3.4603 - val_accuracy: 0.7196
Epoch 32/64
400/400 [==============================] - 159s 397ms/step - loss: 1.1782 - accuracy: 0.7318 - val_loss: 3.4643 - val_accuracy: 0.7234
Epoch 33/64
400/400 [==============================] - 157s 393ms/step - loss: 1.1694 - accuracy: 0.7340 - val_loss: 3.4711 - val_accuracy: 0.7247
Epoch 34/64
400/400 [==============================] - 158s 394ms/step - loss: 1.1435 - accuracy: 0.7368 - val_loss: 3.4905 - val_accuracy: 0.7285
Epoch 35/64
400/400 [==============================] - 157s 394ms/step - loss: 1.1167 - accuracy: 0.7409 - val_loss: 3.4899 - val_accuracy: 0.7310
Epoch 36/64
400/400 [==============================] - 157s 393ms/step - loss: 1.0918 - accuracy: 0.7473 - val_loss: 3.4951 - val_accuracy: 0.7462
Epoch 37/64
400/400 [==============================] - 158s 394ms/step - loss: 1.0861 - accuracy: 0.7459 - val_loss: 3.4717 - val_accuracy: 0.7445
Epoch 38/64
400/400 [==============================] - 158s 394ms/step - loss: 1.0809 - accuracy: 0.7453 - val_loss: 3.5465 - val_accuracy: 0.7486
Epoch 39/64
400/400 [==============================] - 159s 396ms/step - loss: 1.0672 - accuracy: 0.7491 - val_loss: 3.4988 - val_accuracy: 0.7478
Epoch 40/64
400/400 [==============================] - 158s 394ms/step - loss: 1.0495 - accuracy: 0.7520 - val_loss: 3.4924 - val_accuracy: 0.7603
Epoch 41/64
400/400 [==============================] - 156s 391ms/step - loss: 1.0350 - accuracy: 0.7546 - val_loss: 3.5165 - val_accuracy: 0.7561
Epoch 42/64
400/400 [==============================] - 158s 395ms/step - loss: 1.0153 - accuracy: 0.7573 - val_loss: 3.5521 - val_accuracy: 0.7568
Epoch 43/64
400/400 [==============================] - 159s 397ms/step - loss: 1.0029 - accuracy: 0.7597 - val_loss: 3.5261 - val_accuracy: 0.7653
Epoch 44/64
400/400 [==============================] - 158s 395ms/step - loss: 0.9926 - accuracy: 0.7602 - val_loss: 3.5437 - val_accuracy: 0.7614
Epoch 45/64
400/400 [==============================] - 157s 393ms/step - loss: 0.9763 - accuracy: 0.7641 - val_loss: 3.5605 - val_accuracy: 0.7696
Epoch 46/64
400/400 [==============================] - 157s 392ms/step - loss: 0.9606 - accuracy: 0.7669 - val_loss: 3.5471 - val_accuracy: 0.7656
Epoch 47/64
400/400 [==============================] - 156s 390ms/step - loss: 0.9566 - accuracy: 0.7652 - val_loss: 3.5458 - val_accuracy: 0.7713
Epoch 48/64
400/400 [==============================] - 156s 390ms/step - loss: 0.9497 - accuracy: 0.7693 - val_loss: 3.5388 - val_accuracy: 0.7753
Epoch 49/64
400/400 [==============================] - 157s 391ms/step - loss: 0.9341 - accuracy: 0.7712 - val_loss: 3.5683 - val_accuracy: 0.7675
Epoch 50/64
400/400 [==============================] - 156s 391ms/step - loss: 0.9396 - accuracy: 0.7703 - val_loss: 3.5979 - val_accuracy: 0.7657
Epoch 51/64
400/400 [==============================] - 156s 390ms/step - loss: 0.9286 - accuracy: 0.7722 - val_loss: 3.5960 - val_accuracy: 0.7715
Epoch 52/64
400/400 [==============================] - 156s 391ms/step - loss: 0.9081 - accuracy: 0.7780 - val_loss: 3.5891 - val_accuracy: 0.7765
Epoch 53/64
400/400 [==============================] - 157s 392ms/step - loss: 0.8991 - accuracy: 0.7785 - val_loss: 3.6039 - val_accuracy: 0.7758
Epoch 54/64
400/400 [==============================] - 156s 390ms/step - loss: 0.8968 - accuracy: 0.7781 - val_loss: 3.6081 - val_accuracy: 0.7724
Epoch 55/64
400/400 [==============================] - 156s 390ms/step - loss: 0.9026 - accuracy: 0.7776 - val_loss: 3.6385 - val_accuracy: 0.7654
Epoch 56/64
400/400 [==============================] - 156s 390ms/step - loss: 0.8876 - accuracy: 0.7800 - val_loss: 3.6299 - val_accuracy: 0.7780
Epoch 57/64
400/400 [==============================] - 157s 391ms/step - loss: 0.8753 - accuracy: 0.7814 - val_loss: 3.6549 - val_accuracy: 0.7737
Epoch 58/64
400/400 [==============================] - 156s 390ms/step - loss: 0.8658 - accuracy: 0.7836 - val_loss: 3.6736 - val_accuracy: 0.7762
Epoch 59/64
400/400 [==============================] - 156s 391ms/step - loss: 0.8543 - accuracy: 0.7858 - val_loss: 3.6532 - val_accuracy: 0.7837
Epoch 60/64
400/400 [==============================] - 156s 390ms/step - loss: 0.8473 - accuracy: 0.7872 - val_loss: 3.6780 - val_accuracy: 0.7794
Epoch 61/64
400/400 [==============================] - 156s 389ms/step - loss: 0.8533 - accuracy: 0.7859 - val_loss: 3.6423 - val_accuracy: 0.7805
Epoch 62/64
400/400 [==============================] - 156s 389ms/step - loss: 0.8408 - accuracy: 0.7892 - val_loss: 3.6933 - val_accuracy: 0.7755
Epoch 63/64
400/400 [==============================] - 159s 398ms/step - loss: 0.8237 - accuracy: 0.7906 - val_loss: 3.6278 - val_accuracy: 0.7857
Epoch 64/64
400/400 [==============================] - 158s 396ms/step - loss: 0.8267 - accuracy: 0.7923 - val_loss: 3.6879 - val_accuracy: 0.7791
```

Performance

The final accuracy was 0.79.



And we got a hit rate of 78% on the test/train dataset, which seems rather impressive (hit rate as in count of the times our predicted value matched the real value we have on the dataset).

```
#The number below is the hit rate
```

```
78.11956521739131
```

Getting our results

We load the dataset provided to us and we now change the way we 'slide the window'. Instead of doing what we did before for the training dataset, we now load the differences of the values as such: 1st to 999th, predict the 1000th (it's blank on the Excel), then 1001st to 1999th, predict the 2000th and so on. That means that the 50th value we want to find will not be possible to attain since the last element of the array needs a next element to make its difference. That's why we arbitrarily choose -4 as the final difference (since it's one of the most frequently occurring differences) and we append in on the dataset. That gives us a rather odd final value prediction but we couldn't think of some other way to do it. We use the same procedure to derive the results as in the training part of our code.

Inspiration

The links mentioned below on the bibliography, along with the research papers provided to us for the project, plus some talks with people that were working on the project to exchange ideas, and finally the work for the final project in the course of Machine Learning by Mr Houstis.

References

[1] Milad Hashemi Kevin Swersky Jamie A. Smith Grant Ayers Heiner Litz Jichuan Chang Christos Kozyrakis Parthasarathy Ranganathan
Learning Memory Access Patterns.
https://arxiv.org/pdf/1803.02329.pdf

[2] Jason Brownlee
How to use an Encoder-Decoder LSTM to Echo Sequences of Random Integers.
Link

[3] Jason Brownlee
Time Series Forecasting with the Long Short-Term Memory Network in Python .
Link

[4] randerson112358
Stock Price Prediction Using Python & Machine Learning.
Link