Assignment 2

Assignment overview

In this assignment we cerate an AutoEncoder lstm network.

we use the basic model as described in the guidelines and we extend it in each task to our needs.

Basic Architecture

We created simple blocks for the encoder and decoder component which extend torch.nn.Module, a step that might be unnecessary but we find it helpful.

Encoder: the encoder block consists of an lstm, it is initialized with:

- input feature size
- hidden layer width
- number of consecutive lstm layers

it deals with the initialization of the hidden and cell states for the entire encoder, in addition its forward method deals with a typical lstm run and returns two tensors which hold the last hidden and cell states of each layer

Decoder: the decoder block is almost identical to the encoder block but the fact that it also includes a linear fully connected layer that takes the **output** of the decoder lstm and project its dimensions to the desired output dimensions, the output of the forward method is this projection or "prediction"

AutoEncoder: the autoencoder consists of an encoder block, a decoder block, a reconstruction loss criterion (MSE) and all the hyperParameters, we understand that a better practice would have been to separate the network and its training however we decided to keep the training within the network.

in its forward method the AE feeds the input through the encoder layer, recieves the context tensor (the last lstm layer's hidden cell) as output and then clone it and concatenate it to itself in order to create a sequence of identical samples to input the decoder with as the input sequence. it then passes the repeated context to the decoder component to receive the prediction and return them.

In the learning phase the network iterates epochs times over the entire data (which it receives as a pytorch DataLoader object) and then iterates over batches of data and performs optimization steps on their loss, we also hold all the losses over the iterations for future plotting.

AutoEncoder with Classifier Architecture

for the second task we were asked to classify the mnist inputs in addition to the reconstruction so we applied some changes to the network

We added a classifier layer that projects vectors of hidden_size size to vectors of num_classes size, We also added a loss criterion for the classifier in the form of cross entropy.

In the forward method we now also feed the context vector into the classifier layer to receive the probabilities for each class

In the learn phase we now accept two inputs, the data and the labels and in each iteration for each batch in addition to receive the reconstructions we also receive the class probabilities and we calculate two losses, one for reconstruction and one for classifying and we sum them and optimize on their sum

AutoEncoder with next value prediction Architecture

For the final task, our objective is to forecast the subsequent value in a time series (daily high stock prices) while also reconstructing the series itself. After an unsuccessful attempt to adapt our original autoencoder model for this new task, we decided to design a new model, termed snpAE, from the ground up. This new approach does not utilize the previously developed encoder and decoder blocks. Instead, it employs a straightforward class structure, incorporating an LSTM module for encoding and two distinct LSTM modules for both reconstruction and prediction tasks, finally we added a fully connected linear layer to project the result of the decoders to the output size.

The operation of the forward method remains largely consistent with earlier descriptions.

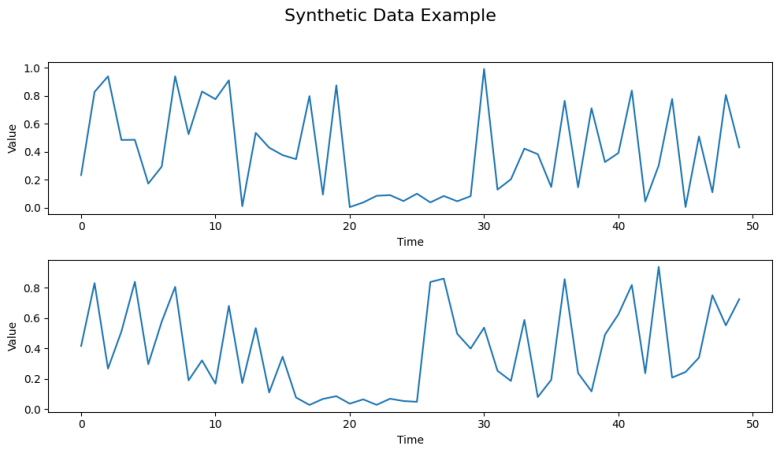
For training, we have now isolated this process into a separate function. This function processes batches of time series data, each containing 50 values, by dividing them into two segments: the first 49 values (x) and the last value (y). It then evaluates the network on x to generate both the reconstructed series and the next value prediction.

The training function calculates the loss using two mean squared error (MSE) criteria, one for each output (reconstruction and prediction), before combining these losses and applying optimization techniques to refine the model.

Task 1 - Synthetic Data reconstruction

In this part we were asked to generate a random dataset that contains 10000 sequences of 50 random numbers in the range [0,1], we were also instructed to randomly decrease a portion of 10 consecutive values by a factor of 10.

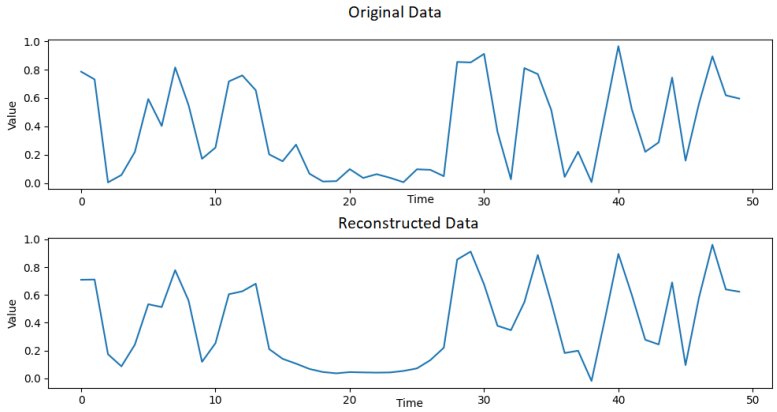
We share two examples of the data we generated:



We trained the basic network, whose architecture we described earlier, on the dataset, we performed a grid search over the hyper parameters, We made sure, as instructed, that the network doesn't learn the identity function by not checking hidden sizes of up to 50, and received this as the best combination:

```
Best Hyperparameters:
{'learning_rate': 0.005, 'hidden_size': 45, 'num_layers': 1, 'batch_size': 32}
```

The resulting reconstruction:



Task 2 - Mnist reconstruction and classification

Reconstruction

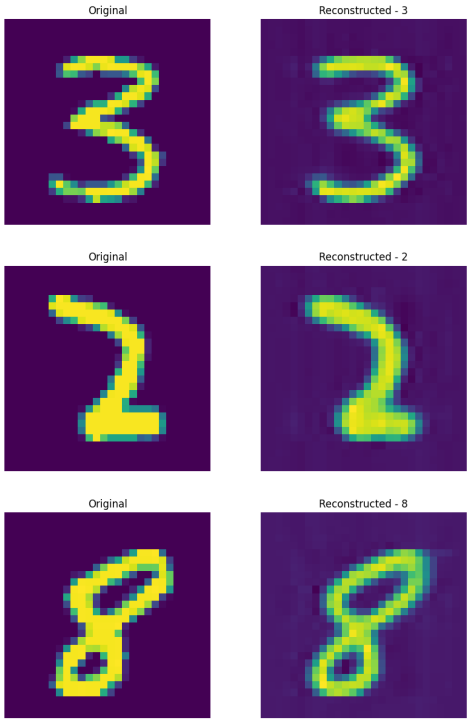
Next we were asked to use our network to reconstruct Mnist dataset.

first, we were to input each image as a sequence of 28 samples of 28 features.

for That we used the AE as previously described (later on we changed the architecture so the code in lstm_ae_mnist.py now holds the version of the network with the classifier but the results of the reconstruction stay similar), We tweaked with the hyper-parameters and got the best results with:

- hidden size = 27
- layer number = 2
- epochs = 300
- learning rate = 0.01
- grad clip = 1
- batch size = 64

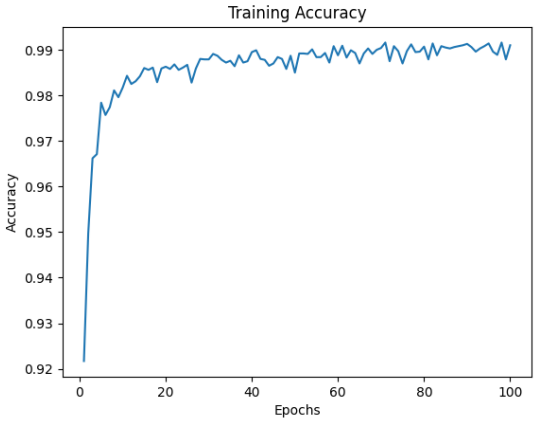
here are some reconstructions:

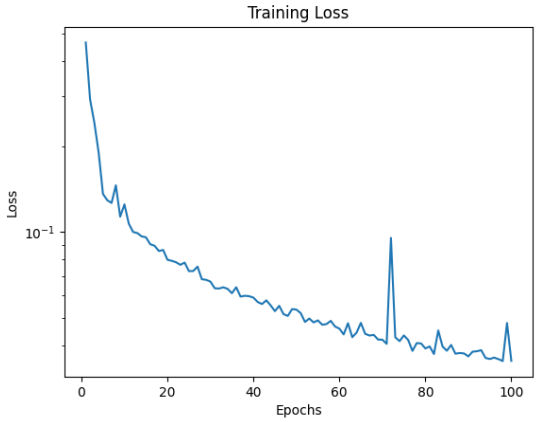


Classification

For this part we were requested to train the network to classify Mnist images in addition to reconstruct the signal, for this we used the second architecture we described, AE with Classifier, we fed the classifying layer the context of each signal (the encoder last hidden layer) and we optimized on the sum of the losses of the classifying and the reconstruction, We had to run the network for many iterations in order to get both a decent reconstruction and a high enough accuracy rate, we used google colab GPUs

Here are the graphs that describe loss vs. epochs and accuracy vs. epochs:

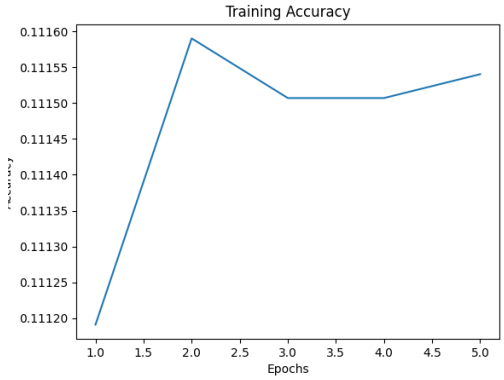




Pixel by Pixel sequencing

for this part we were asked to train the network on a different view of the input, as a sequence of 784 pixels. This part was very hard for the computer so we ran fewer epochs and the results were respectively poor, that is due to the fact that LSTMs and RNNs in general struggle with long sequences of data due to exploding/vanishing gradients.

here are the accuracy rates:



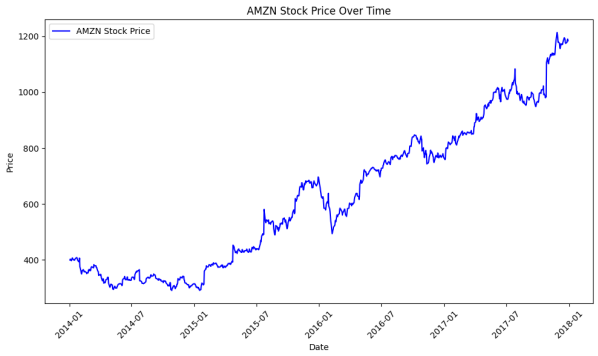
we can clearly see there is no improvement after 5 epochs and the network classifies at random

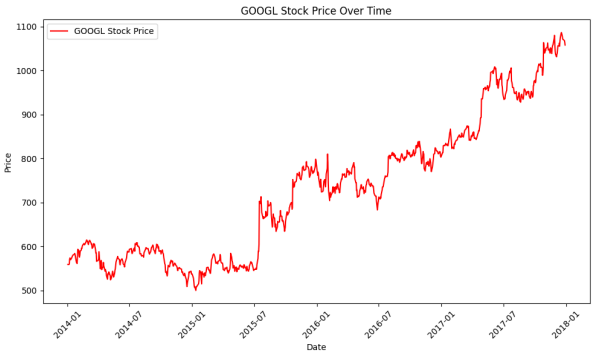
Task 3 - S&P500 reconstruction and next-day prediction

Finally we were instructed to modify our net to allow reconstruction and later prediction of stock high price data. Since LSTM networks struggle with long data sequences, we cut each stock data to subsequences of 50 . in addition we performed normalization of each subsequence based on its max and min value.

Daily Max Examples

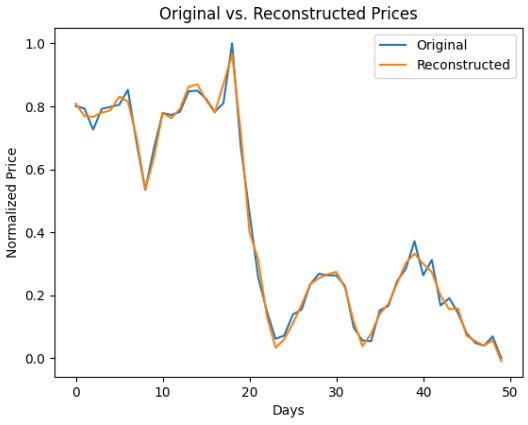
Here are plots that depict the daily max value of the full AMZN and GOOGL stocks:





Reconstruction

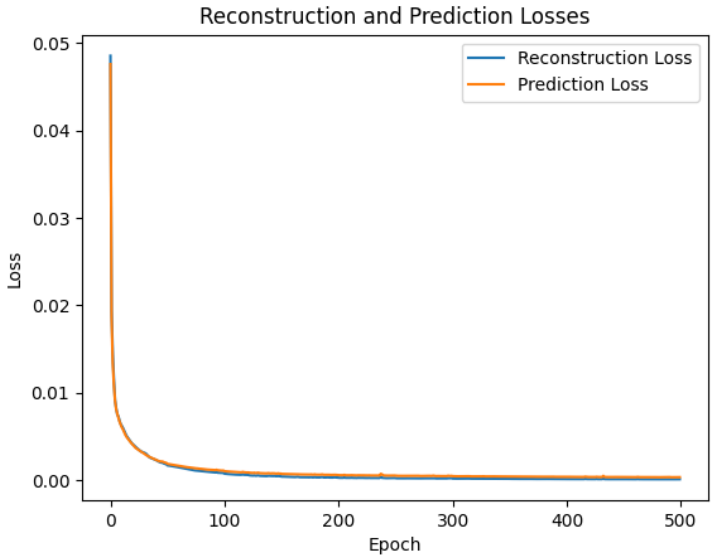
In this part we were asked to train our LSTM AE to reconstruct stock price data. we use the network as described in the overview, here are 3 results of the reconstruction, in the title we present the hyper-parameters we used.



Next Value Prediction

As outlined in the overview, the dataset for this segment comprised two sequences of 49 values each: x, which serves as the training data ranging from 0 to 49, and y, which represents the target predictions spanning values 1 to 50.

To predict the subsequent value, we employed a distinct decoder leveraging the context vector. The loss was determined by computing the cross-entropy loss between the predictor's output and the $y$values.



Additionally, we present a plot comparing the reconstructed and predicted signal to the original following the modification. After this change, the network's ability to reconstruct experienced a slight decline.

In the plot comparing original versus predicted prices, we adjusted the signals to ensure alignment over the same days, with the original values indexed from 1 to 50 and the predicted values from 0 to 49. Despite a minor discrepancy, the graphs align closely with the reconstruction graph, which we deemed acceptable.

Multi-Step Prediction

Lastly, we conducted a test on our predictor by attempting to reconstruct a full signal from a sliced sequence. We selected a random stock and sampled its first 100 values, then sliced this sample in half to serve as the input for our network. We iteratively passed this input through the network, appending the output's last value to the resulting signal each time.

While there was notable alignment with the first half of the signal, the accuracy of the predictions declined as we extended the signal. This decrease in accuracy can be linked to our approach for generating the predicted signal:

Initially, we produced the entire predicted half sequence and appended it to the original's first half. Then, for each iteration, we fed the last 50 values of the evolving signal into the network to generate a new prediction, but only the last predicted value was added to our resultant signal.

Here is a plot showing the results of a random stock