Sequential Modeling - Exercise 2

Dan Ben Ami, ID: 316333079
Nir Mualem, ID: 205467780

December 2022

1 Introduction and Data Processing

In this work, we considered two kinds of datasets, a music dataset for classification and an exchange rate dataset for regression. The firsr dataset, named JSB Chorales, is a polyphonic music dataset consisting of the entire corpus of 382 four-part harmonized chorales by J. S. Bach. Each input is a sequence of elements. Each element is an 88-bit binary code that corresponds to the 88 keys on a piano, with 1 indicating a key that is pressed at a given time. The data is divided into 3 subsets: the training data set - contains 229 series, the validation data set - contains 76 series and the test data set - contains 77 series. The second, named Exchange, records the daily exchange rates of eight different countries ranging from 1990 to 2010. We divided the data into two sets: the training set containing 75% of the data and the test set containing the remaining 25%.

2 Sequential Forecasting

In this assignment we considered the sequential forecasting problem in context of both classification (JSB Chorales dataset) and regression (Exchange dataset) settings. In both settings, the main idea is to predict the next item, x_{t+1} for each time series of items $(x_1, ..., x_t)$, that is, to find a stationary function $f(x_1, ..., x_t) = \tilde{x}_{t+1}$ for which $MSE(\tilde{x}_{t+1}, x_{t+1})$ (for the Exchange data set) or the $NLL(\tilde{x}_{t+1}, x_{t+1})$ (for the JSB data set) is minimal. The main difference between the settings is that in the classification problem x is a binary vector with a size of 88, compared to the regression where the x is real scalar.

On each of the data sets, in order to find the most appropriate f function, we compared 4 different models: ARIMA, RNN, TCN, Transformer. In order to train each of the models, we divided the training series into chunks of 20 time steps, and on each of the chunks we let the model predict a single time step ahead and compared it to the ground truth item in the series to compute the loss. On each of the data sets, each of the models (except ARIMA which in order to fit it does not require epochs) was trained for 100 epochs, when at the end of each epoch we saved the loss for the training data set and the test data set.

2.1 ARIMA

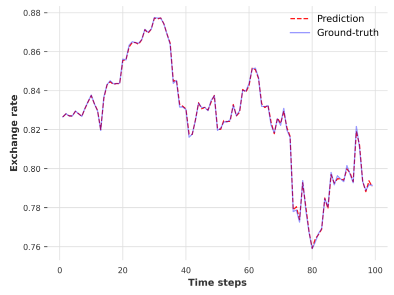
An auto regressive integrated moving average (ARIMA), is a statistical analysis model for the forecasting problem. The model receives 3 hyper parameters: p, d, q, where p is the auto regressive order (The number of steps back), d is the differencing order (i.e., the number of times the data values are replaced by the difference between the data values and the previous values), and q is the order of the moving average (the size of the moving average window). The model equation is given by:

$$\left(1 - \sum_{i=1}^{p} \phi_i L^i\right)(1 - L)^d Y_t = c + (1 + \sum_{i=1}^{q} \theta_i L^i\Big)\epsilon_t \tag{1}$$

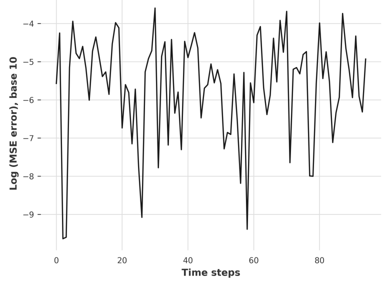
where L is lag operator as described in class, ϕ, θ are the model parameters and $\epsilon_t, t \in \mathbb{N}$ are generally assumed to be white noise variables, i.e., independent, identically distributed standard Gaussian variables.

With the Exchange dataset Since ARIMA model can only take a univariate time series, in this model particularly, we trained the model only on the 'OT' column.

with the JSB Chorales dataset Since the Arima model can be trained on a single time series, we concatenated between all the time series in the training data, where between every two series we padded with 20 zeros. For the same reason as before, since the Arima model is only intended for univariate time series, in this data set we had to define 88 different models, that is, for each note on the piano we trained a different forecasting model.



(a) A prediction example on one time series from the series in the test data set



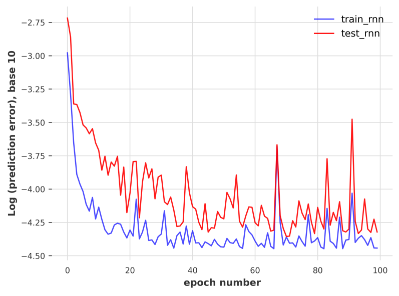
(b) log of the MSE error for prediction on one time series from the test data set

Figure 1: ARIMA model prediction results after training.

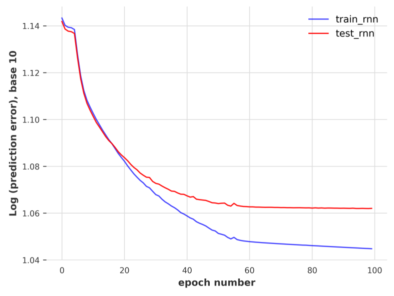
In figure 1 the results of the model after training on the Exchange data set are shown. The MSE error on all the predictions in the series in the Exchange test set is $2.219 \cdot 10^{-5}$, in addition, the NLL error in all the predictions in the series in the JSB test data set is 9.210. As we can see in the graphs, the ARIMA model is able to reasonably predict the substitution data set, but this claim is not true for the JSB data set where it consistently predicts 0 for all notes on the piano at each time step, because it receives as input only a single note and cannot consider the correlation between the characters.

2.2 RNN

Recurrent Neural Network (RNN) is a class of artificial neural networks where the output of certain neurons are connected back to more posterior neurons and create cycles. In this way the network can "remember", and issue an output that also depends on previous inputs at earlier time steps. We used a vanilla RNN model with 2 rnn layers and hidden state in size of 100. Unlike ARIMA, since RNN can receive as input, time series in which each item is a vector, this model could also rely on previous information from other columns in the Exchange data set, while in the JSB data set, the model could consider other piano notes in order to predict When a specific note will be played again. That is, this model received as input whole vectors at each time step and at each prediction it predicted a whole vector (length 8 or 88, depending on which data set).



(a) Log (base 10) of the MSE error of the train and test subsets in the Exchange data set



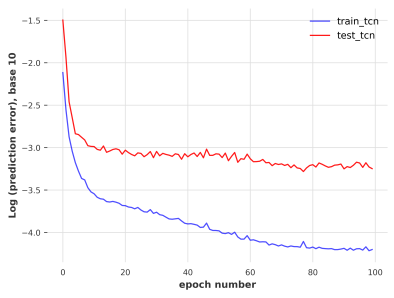
(b) Log (base 10) of the NLL error of the train and test subsets in the JSB Chorales data set

Figure 2: RNN model error per epoch on the train and test subsets.

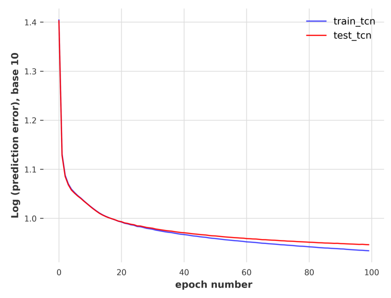
Figure 2 shows the errors of the RNN models that were trained on each of the data sets (Exchange and JSB) in each of the different epochs in which they were trained. As shown in the graphs, the RNN achieves good results and converges in about 60 epochs without entering into significant overfitting.

2.3 TCN

Temporal Convolutional Network (TCN) [1] is is a class of artificial neural networks where, compared to normal Convolutional networks (CNN) which are mainly intended for images, are intended for sequential data and perform forecasting. The time series is entered into the network as a vector/tensor padded with zeros according to the size of the kernel in the first layer. In each layer, the kernel size actually constitutes the amount of backward time steps that each item in the output series depends on from the input series to this layer. Using this method, in addition to skip connection, normalization and dropout, the TCN network was created. Similar to RNN, this model also predicts an entire vector at each time step and therefore can also take into account the correlations between the different features of each input vector when predicting the next vector.



(a) Log (base 10) of the MSE error of the train and test subsets in the Exchange data set



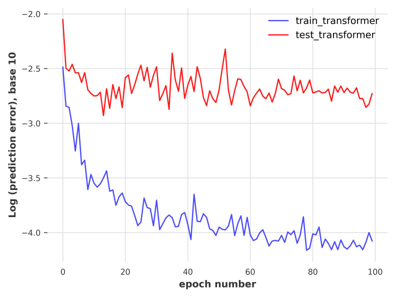
(b) Log (base 10) of the NLL error of the train and test subsets in the JSB Chorales data set

Figure 3: TCN model error per epoch on the train and test subsets.

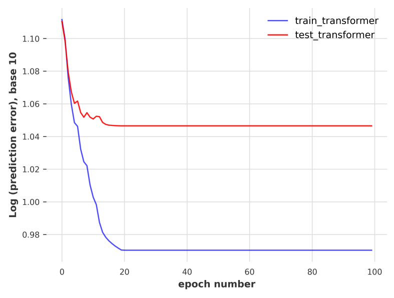
Shown in figure 3 is the MSE error for the Exchange data set and the NLL error for the JSB data set at the end of each of the 100 epochs in which the TCN models were trained. As shown in the graphs, the TCN model achieves good results, but in the Exchange data set, unlike the JSB data set, it enters overfitting very quickly.

2.4 Transformer

Transformer is a neural network architecture, follows an encoder-decoder structure but unlike the previous architectures does not rely on recurrence and convolutions in order to generate an output. The role of the encoder is to receive the entire input at once, along with positional encoding, and output to the decoder a continuous vector that represents the input series. The role of the decoder is to receive the representative vector together with the previous output it produced and to generate a new output item in the output series, ”At each step the model is auto-regressive, consuming the previously generated symbols as additional input when generating the next.” [2].



(a) Log (base 10) of the MSE error of the train and test subsets in the Exchange data set



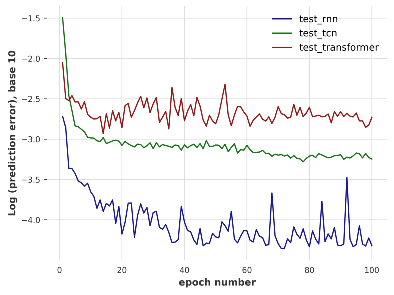
(b) Log (base 10) of the NLL error of the train and test subsets in the JSB Chorales data set

Figure 4: Transformer model error per epoch on the train and test subsets.

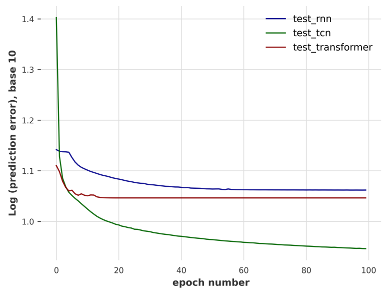
Shown in figure 4 is the MSE error for the Exchange data set and the NLL error for the JSB data set at the end of each of the 100 epochs in which the Transformer models were trained. As shown in the graphs, the Transformer model achieves good results but converges very quickly and enters overfitting within the shortest number of epochs among all models.

3 Models Comparison and Conclusions

As shown in figure 5, the type of task (classification or regression), as well as the amount of samples in the training data set, has a great influence on the performance of each model. In addition, we tried several hyper-parameters for each model, in this work we presented the models using the hyper-parameters with the empirically best results that were achieved. However, different choices which we did not try might result a better performance in both of the tasks.
For the Exchange data set, the RNN model seems to achieve the best performance, as opposed to the JSB data set, on which the TCN model seems to achieve the best results.



(a) Log (base 10) of the MSE error of the test subsets in the Exchange data set



(b) Log (base 10) of the NLL error of the test subsets in the JSB Chorales data set

Figure 5: models error comparison per epoch on the test subsets.

Model / result	ARIMA	RNN	TCN	Transformer
Final MSE on Exchange test set	$2.219 \cdot 10^{-5}$	$4.411 \cdot 10^{-5}$	$5.230 \cdot 10^{-4}$	$1.174 \cdot 10^{-3}$
Final NLL on JSB test set	9.210	11.53	8.838	11.13

Table 3 shows the error values of the final models for each of the data sets, we emphasize that the ARIMA model was trained differently from the other models as described before, so the comparison with it is not completely objective.

References

[1] Shaojie Bai, Zico Kolter, and Vladlen Koltun. “An Empirical Evaluation of Generic Convolutional and Recurrent Networks for Sequence Modeling”. In: *arXiv:1803.01271* (2018).

[2] Ashish Vaswani et al. “Attention is all you need”. In: *Advances in neural information processing systems* 30 (2017).