ANNDL Homework 2

Nicola Farenga
10537686

Giorgio Longari
10488269

1 Introduction

The following project report refers to the second homework of the Artificial Neural Networks & Deep Learning course at Politecnico di Milano (a.y. 2021/22). The aim of the project was to develop a deep learning model to perform multivariate time series forecasting. The sections' order of the report reflects the chronological order we dealt with the concerning topic.

The code repository of the project is available at `https://github.com/farenga/anndl-hw2`.

2 Methods

2.1 Data preparation

The dataset provided was composed of 7 features and 68K timesteps. Since each feature had a different range we opted to preprocess the data by applying the following *min-max* normalization method:

$$x_{\text{scaled}} = \frac{x - \min(x)}{\max(x) - \min(x)} \tag{1}$$

As a preprocessing technique also *mean* scaling has been tried out, but its use has been then neglected due to no major improvements in the models predictions capabilities. The framework selected to tackle the forecasting task was autoregression, indeed, as can be seen in Figure 1, at inference-time the model was fed with data by progressively sliding a window of fixed size over the input data, and by sequentially adding the forecasted values to the selected input-window, recursively feeding them to the model as new inputs. The strategies employed were the following two:

1. Multi-Input / Single-Output: the window $x_1, ..., x_t$ was fed as input, and the value x_{t+1} was provided as output;
2. Multi-Input / Multi-Output: the window $x_1, ..., x_t$ was fed as input, and the values $x_{t+1}, ..., x_{t+N}$ were provided as output.

In order to provide data structured in the way defined above, we developed a proper function able to extract subsequences of fixed size from the dataset, given their length, the future window size and the stride. Before the developement of such function we tried multiple approaches, such as tensorflow-keras built-in methods,

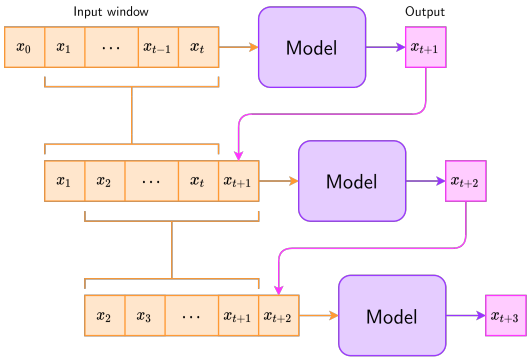

Figure 1: Autoregressive model.

but we felt much more control over the produced output by building our own function. During the developement phase we splitted the dataset leaving the last 1K datapoint for testing, then after having tested out models' performances we have retrained them on the complete dataset.

2.2 Models

The models described in the following sections are stored in the file `models.ipynb` within the main repository, where they are defined by using the keras-tensorflow syntax.

Evaluation. The following models have been evaluated by considering Root Mean Squared Error (RMSE) and Mean Absolute Error (MAE), since they were the monitored metrics in the competition.

2.2.1 Base models

Given the temporal dependency of the dataset provided and the nature of the problem we had to tackle, we have decided to firstly compare some basic models based on different recurrent units. Starting up by defining simple architectures allowed us to have a lower bound in terms of model complexity that could act as a reference point for next comparisons. In this first phase those architectures were compared in a Multi-Input / Single-Output framework. The recurrent architectures selected were the following: RNN, LSTM, stacked-LSTM and GRU. In order to make a fair comparison the different models shared the following characteristics: 256 recurrent units/layer, 50 epochs, 256 samples as batch size, 200/1 as input/output window size.

The introduction of a *learning rate* reduction schedule has been necessary since we noticed some oscillating behaviours during the training process. By comparing those architectures we can easily see that LSTMs have much better performances during training, indeed LSTMs' time per epoch was roughly 50% less than the time required by Simple RNNs, and moreover LSTMs' behaviour was much more smooth. As an example we have reported the following predictions for the *feature #6*

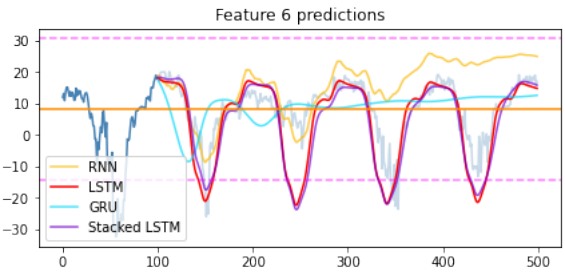

Figure 2: Base models comparison.

where can be seen that the base LSTM outperformed other architectures, indeed it converged much faster to a more reasonable prediction.

Model	RMSE	MAE
RNN	23.74	13.39
LSTM	7.76	4.43
Stacked-LSTM	11.19	6.34
GRU	29.31	14.36

Based on this first comparison we have selected the single-layer LSTM model and scaled it up to 512 units and to an input window of 400-timesteps. After training it on the whole dataset we reached a score of 3.92 (RMSE). *After reading the results of (Jozefowicz et al., 2015), we could guess that GRU poor performances were caused by a wrong learning rate schedule that probably slowed down the training process due to a local minima in the optimization procedure.*

2.2.2 Seq2Seq and Attention

Following the design characteristics of autoencoders' architecture, we developed multiple models to address the time series forecasting task by leveraging an encoder-decoder structure. In our case the encoding and decoding tasks were both performed by LSTM recurrent layers, indeed the core structure consisted of an LSTM layer with 512 units acting as an encoder, then the encoded hidden state has been wrapped by a *repeat* layer and fed to the second LSTM layer, acting as a decoder. In a second phase we stacked two LSTM layers, one of which bidirectional, instead of the single-layer decoder. The previous architecture has been then enhanced by adding *Luong Attention Mechanism* (Luong et al., 2015), allowing the model to assign weights to the encoded sequence. The attention mechanism implemented consisted of fistly computing alignemnt scores a, between the source s and the target t, that can be referred as our encoded and decoded hidden states

$$a_t(s) = \text{softmax}(\mathbf{h}_t^T \mathbf{h}_s)$$

the context vector $\mathbf{c}$ was then computed as the weighted average of the source states by using attention scores, and it has been concatenated with the source state and fed into a fully-connected layer. In terms of performances both Seq2Seq and Seq2Seq + Attention models had better results with respect to the base models, indeed we achieved the following results on test data

Model	RMSE	MAE
Seq2Seq	4.19	2.38
Seq2Seq+Attention	4.70	2.77

After the developement phase we have retrained the models on the complete dataset and we achieved our top score with the Seq2Seq + Attention model (RMSE 3.87, MAE 2.34).

2.2.3 Convolutions and LSTMs

In order to verify recurrent architectures' performances we built a non-recurrent architecture to understand the predictions capabilities of a feed-forward neural-network in a time series forecasting environment.

By building mulitiple CNN-based models we noticed good capabilites of CNNs at capturing small-scale patterns, but the main problem we faced was the unstablity due to prediction errors propagation while performing autogression. For this reason we implemented a Multi-Input/Multi-Output strategy, indeed this approach allowed us to reach results to those reached with simple recurrent architectures. In order to reach better results, we opted to scale this approach and to build a model based on both Convolutions and Long-Short Term Memories, to leverage CNN's capability of learning small-scale patterns and LSTM's ability to spot long term trends.

Inspired by DeepMind's WaveNet architecture (van den Oord et al., 2016), an autoregressive deep generative model used for audio waveforms generation, we built a model based on WaveNet's main feature: *dilated causal convolutions*. Causal convolutions ensure that the ordering in the input sequence is preserved while processing it, and their dilated version allow to increase the receptive field when stacking them without critically increasing the computational cost. The main block that gets repeated through the network is made of a stack of dilated 1-dimensional convolutional layers (by progressively doubling the number of filters and dilatation-rate), followed by batch normalization and average pooling layers. Convolutions' computational efficiency allowed us to deepen the network without

any major slow-down in terms of training time. By stacking multiple of these blocks and equipping them with residual connections, to build a structure similar to the ResNet architecture, and by allowing LSTMs states to flow between recurrent layers, we have noticed a major improvement in terms of the model prediction capabilities.

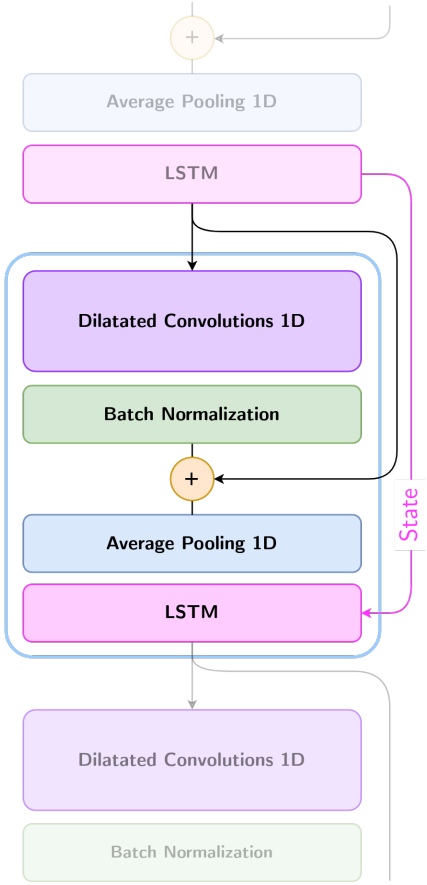

Figure 3: dilated CNN + LSTM Block.

The architecture has been tested out with multiple settings, by replacing *max pooling* with *average pooling*, and by scaling the input and output windows.

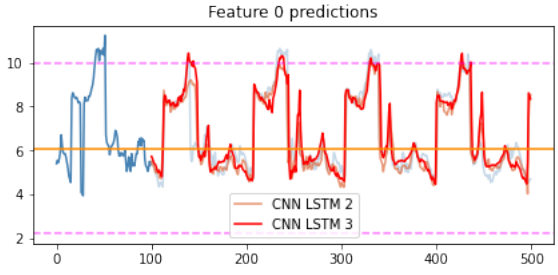

Figure 4: CNN+LSTM: Max vs Average Pooling.

The performances of the selected setup (input/output sequence length = 400/4 + average pooling) were comparable to those of previously built models, with a score of 3.99 (RMSE).

2.3 Hyperparameters Tuning

During the developement phase we tested out how tuning different hyperparameters would result in terms of overall performances. Multiple tests have been performed both manully and by adopting keras-tuner. We focused our attention on the following hyperparameters: input-output sequence length, learning rate, number of recurrent units.

(1) After testing the base models with multiple input window sizes we concluded that being in the range $200 \div 400$ allowed to capture the periodicity of the time-series while mantaing a reasonable computational load. While for the output window size, we saw that dealing with a size of 4 was optimal in the CNN+LSTM Multi-Input/Multi-Output case.

(2) The learning rate has been initally setted as a fixed value (1e-3) but, expecially while dealing with the CNN+LSTM network, we had to adopt a learning rate reduction schedule, to prevent non-monotone behaviours of the loss function.

(3) For what concerns the number of units per LSTM layer, we tested out multiple setups and we saw that scaling the number of units allows to achieve better results, but after certain values the increase in model performances is negligible with respect to the increase in computational cost, so for this reason we have selected 512 units as a upper bound.

3 Conclusions

Having evaluated a variety of architectures, we have concluded that recurrent architectures are well suited for dealing with tasks regarding time series. Implementing Seq2Seq architecture equipped with Attention mechanism allowed us to understand their effectiveness, and by combining RNNs with CNNs we showed the capabilities of models that are not based on such mechanisms.

References

[Jozefowicz et al. 2015] Jozefowicz, R., Zaremba, W. & Sutskever, I. *An empirical exploration of recurrent network architectures.*.

[Luong et al. 2015] Luong, M., Pham, H. & Manning, C. *Effective Approaches to Attention-based Neural Machine Translation*.

[van den Oord et al. 2016] van den Oord, A., Dieleman, S., Zen, H., Simonyan, K., Vinyals, O., Graves, A., Kalchbrenner, N., Senior, A. & Kavukcuoglu, K. *WaveNet: A Generative Model for Raw Audio*.