AN2DL Homework 2

Aleksandra Krajnovic, Iva Milojkovic, and Mariusz Wiśniewski

Team name: Three convolutioneers

Politecnico di Milano

February 23, 2022

1 Introduction

The goal of this assignment was to train a convolutional neural network able to solve a multivariate time series forecasting problem.

2 Research and Preprocessing

We started the project by researching existing approaches and solutions to similar problems. A list of initial models we wanted to attempt comprised of, but was not limited to:

- *Vanilla Feedforward Neural Network*,
- *Vanilla Convolutional Neural Network*,
- *Vanilla LSTM* [7],
- *Stacked LSTM*,
- *Convolutional LSTM*,
- *Bidirectional LSTM*,
- *Convolutional bidirectional LSTM*,
- *Seq2Seq LSTM*,
- *SCINet* [10].

We compared autoregressive and direct prediction approaches for each model, narrowing down the versions that were most applicable to our task. Before feeding the data to our chosen models, we normalized it. We attempted other preprocessing techniques, but research did not suggest any viable solutions, and the data had to stay in an ordered sequence.

3 Training the Models

During the course of Homework 1, we have learned that a better tuned and less powerful model is able to achieve better predictions than a more powerful and poorly tuned one. Therefore, we changed our approach this time, narrowing down the most interesting models and focusing on their hyperparameters. The training process was also significantly faster this time, which gave us more room to experiment.

3.1 Initial Approaches

We decided on using the mean square error (*MSE*) loss function for training and the root mean square error (*RMSE*) as a metric. As in the previous homework, we implemented the following callbacks: *EarlyStopping*, *ReduceLROnPlateau*, model checkpointing, and *Tensorboard*. All of them were provided by the *TensorFlow* framework [12].

During the first attempts, we experienced issues with models returning *NaN* values when training. We attributed that to exploding gradients and tried resolving it by applying *Dropout* layers [16] with different dropout rates as well as utilizing the gradient norm clipping strategy [14] via setting the *clipnorm* parameter, but the solution that worked best was L2 regularization.

Due to initial low scores in comparison to our colleagues, we were motivated to try different approaches and empirically moved from one solution to the other until finally making progress by using an incremental autoregressive prediction. We also experimented with the batch size and its connection with the learning rate [15].

The most promising approach occurred to be the convolutional bidirectional LSTM with the window size of 100, and the telescope value of 64.

3.2 Further Improvements

Once reaching a point where we were more confident which models are more suitable and which not, we implemented other optimizations, such as adding a *stateful* LSTM. This solution proved to be problematic, as we were only able to implement it for a batch size equal to one due to ever-changing layer sizes that could not be fed back into the model. This approach, however, did not yield any improvements.

Additionally, we replaced LSTMs with Gated Recurrent Units (*GRUs*) [2], which contributed to a slight but nonetheless significant deterioration of the results.

Subsequently, we decided to apply *batch normalization* [8] after each activation function to stabilize the training process and also exploited *Leaky ReLU* [6] in place of previous hyperbolic tangent.

All the aforementioned solutions did not improve the general results. This prompted us to look for other solutions. We decided to implement attention mechanism [17] and add it to our model. Thus, self-attention as well as multi-head attention using both dot product (Luong's attention [11]) and additive (Bahdanau's attention [1]) approach were employed. Furthermore, we tried taking advantage of residual connections [5]. Finally, we could notice significant improvements, which emerged by using multiplicative self attention.

At that point, we ended up with a model architecture that was quite performing, but hyperparameter tuning was still to be done. *KerasTuner* [13] provided by *Keras* library [3] was applied to this task. We decided on the *hyperband* [9] approach for hyperparameter optimization. Thanks to this process, we knew what number of units in each layer worked best for our task. Moreover, this made us replace the hyperbolic tangent activation function inside LSTM layers to Exponential Linear Unit (ELU) [4].

Furthermore, we kept experimenting with various batch sizes, window sizes, strides and prediction telescopes, which eventually lead us to minimize the *RMSE* on the test set down to 3.819.

4 Results and Remarks

The final model we chose for the time series prediction was an ensemble of models that performed best for each prediction feature. They are mostly based on the convolutional bidirectional LSTM with self-attention, but their parameters (window length, stride, number of units in the LSTM layer, prediction telescope) varied. None of the models could predict all the variables reliably, but they managed to generalize better as an ensemble. The final *RMSE* score we achieved was 3.679.

Future work might include training each of the ensemble models to fit a specific feature separately, and only then combining them, also exploring which voting mechanism works best for the ensemble. Overall, we are pleased with the slow but steady improvements in our predictions during the course of the challenge.

5 References

[1] Dzmitry Bahdanau, Kyunghyun Cho, and Yoshua Bengio. *Neural Machine Translation by Jointly Learning to Align and Translate*. 2016. arXiv: 1409.0473 [cs.CL].

[2] Kyunghyun Cho et al. *Learning Phrase Representations using RNN Encoder-Decoder for Statistical Machine Translation*. 2014. arXiv: 1406.1078 [cs.CL].

[3] François Chollet et al. *Keras*. https://keras.io. 2015.

[4] Djork-Arné Clevert, Thomas Unterthiner, and Sepp Hochreiter. *Fast and Accurate Deep Network Learning by Exponential Linear Units (ELUs)*. 2016. arXiv: 1511.07289 [cs.LG].

[5] Kaiming He et al. *Deep Residual Learning for Image Recognition*. 2015. arXiv: 1512.03385 [cs.CV].

[6] Kaiming He et al. *Delving Deep into Rectifiers: Surpassing Human-Level Performance on ImageNet Classification*. 2015. arXiv: 1502.01852 [cs.CV].

[7] Sepp Hochreiter and Jürgen Schmidhuber. "Long short-term memory". In: *Neural computation* 9.8 (1997), pp. 1735–1780.

[8] Sergey Ioffe and Christian Szegedy. *Batch Normalization: Accelerating Deep Network Training by Reducing Internal Covariate Shift*. 2015. arXiv: 1502.03167 [cs.LG].

[9] Lisha Li et al. *Hyperband: A Novel Bandit-Based Approach to Hyperparameter Optimization*. 2018. arXiv: 1603.06560 [cs.LG].

[10] Minhao Liu et al. *Time Series is a Special Sequence: Forecasting with Sample Convolution and Interaction*. 2021. arXiv: 2106.09305 [cs.LG].

[11] Minh-Thang Luong, Hieu Pham, and Christopher D. Manning. *Effective Approaches to Attention-based Neural Machine Translation*. 2015. arXiv: 1508.04025 [cs.CL].

[12] Martín Abadi et al. *TensorFlow: Large-Scale Machine Learning on Heterogeneous Systems*. Software available from tensorflow.org. 2015. URL: https://www.tensorflow.org/.

[13] Tom O'Malley et al. *KerasTuner*. https://github.com/keras-team/keras-tuner. 2019.

[14] Razvan Pascanu, Tomas Mikolov, and Yoshua Bengio. *On the difficulty of training Recurrent Neural Networks*. 2013. arXiv: 1211.5063 [cs.LG].

[15] Samuel L. Smith, Pieter-Jan Kindermans, and Quoc V. Le. "Don't Decay the Learning Rate, Increase the Batch Size". In: *CoRR* abs/1711.00489 (2017).

[16] Nitish Srivastava et al. "Dropout: A Simple Way to Prevent Neural Networks from Overfitting". In: *Journal of Machine Learning Research* 15.56 (2014), pp. 1929–1958. URL: http://jmlr.org/papers/v15/srivastava14a.html.

[17] Ashish Vaswani et al. *Attention Is All You Need*. 2017. arXiv: 1706.03762 [cs.CL].