Artificial Neural Networks Challenge Report

Aldo Cumitini, Davide Gesualdi, Gabriele Passoni, Roberto Riva

December 22, 2023

Abstract

Given a dataset of time series and a set of categories which this series belonged to, our objective was to build a neural network capable of predicting with accuracy the next 9/18 point in time, using what we learned and experienced in classes and labs. In this document we're going to discuss about the methods we used, the experimental data we obtained, the challenges we encountered and the final results and techniques we used.

1 The Network

1.1 Recurrent Neural Networks

Recurrent neural networks (RNN) are one of the two main types in which neural networks can be divided based on the direction of the flow of information in the architecture itself. This type of networks can be distinguished by the bi-directional flow of data, opposing to the feedforward networks (FNN) which are uni-directional. This means that in FNN neurons can only pass their data to subsequent layers of the net, while in the RNN output data can be used by the same neurons in the next step of computation. Therefore RNN are able to retain some sort of previous state, which makes this type of network particularly suitable for tasks where dependencies between subsequent data can be extremely relevant, such as: speech recognition, translation or time series.

1.2 The gradient vanishing problem

Seems all good until now, we have a problem and we have a technology to solve it. Well, that would be the case if the **gradient vanishing problem** wouldn't present itself. This problem happens when gradients used in back propagation learning are so small that the weights we're training won't change anymore. But why this? Well, since we're using finite precision numbers to represent weights, if the change is not big enough our representation can't cover that and our values will be approximated to the same ones we tried to improve. This problem hits particularly harder when we're trying to preserve dependencies in the long run (which means after a consistent number of training steps).

1.3 The solution: LSTM networks

The Long Short Term Memory (LSTM) architecture is a type of RNN which aims to partially mitigate the vanishing gradient problem [1]. This can be useful when our network is dealing with problems where dependencies distant in time must be preserved, which is very important in a time series challenge (since similar patterns can be repeated in time even with long temporal distances). The atomic block of this type of network is a cell characterized by three main gates: input, output and forget gate. The first one decides what input informations are to be stored in the cell, the second what data of the current state are to be preserved and what to be discarded, the last one decides the output of the cell based on the current state (computed by input and forget gates) and the previous one. This enables the network to preserve data and reduce the gradient vanishing problem since informations can flow in the NN unchanged.

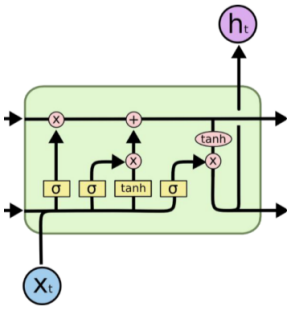

Figure 1: LSTM Cell.

1.4 The fading of transformers

Nowadays there is plenty of papers regarding this flexible architecture but, as some are pointing out [2], they're actually less accurate than simply predicting tomorrow by using today samples. Furthermore, it's a very complex structure to train properly and less efficient than a simple linear **LSTM** layer

1.5 Alternative models tested

In **Stacked LSTM seq2seq Autoencoder** model [3] we applied sequence to sequence learning, in which a RNN

model is trained to map an input sequence to an output sequence. The lengths of the input and output sequences need not necessarily be the same. The seq2seq model comprises two RNNs, specifically LSTMs, which can be conceptualized as an encoder and a decoder. The encoder part converts the given input sequence to a fixed-length vector, which acts as a summary of the input sequence. This fixed-length vector is referred to as the context vector. The context vector is then given as input to the decoder and to the final encoder state, as an initial decoder state to predict the output sequence. A repeat vector layer is involved to replicate the context vector obtained from the encoder to pass it as input to the decoder. The output received from the decoder with respect to each time step is mixed. The time distributed densely will apply a fully connected dense layer on each time step and separates the output for each timestep. The time distributed densely is a wrapper that allows applying a layer to every temporal slice of an input. By stacking LSTM's, the model's ability to comprehend more intricate representations of time-series data in hidden layers is enhanced, enabling the capture of information at various levels.

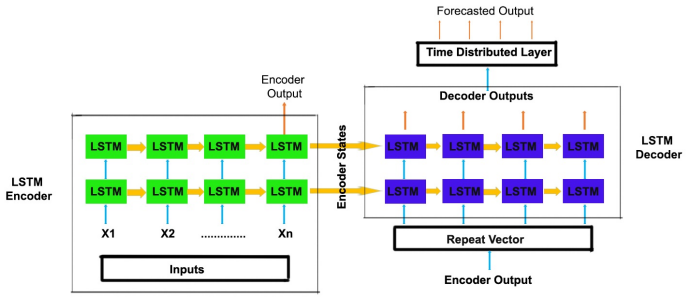

Figure 2: Stacked LSTM seq2seq Autoencoder model.

Temporal Convolutional Networks (TCNs) were initially proposed for video-based action segmentation to address the need for two separate models. The first model involves computing low-level features using a CNN, which encodes spatial-temporal information. Subsequently, these low-level features are input into a classifier that captures high-level temporal information using a RNN. TCN provides a unified approach to hierarchically capture both levels of information. TCN has the capability to process a series of any length and produce an output of the same length. In the implemented encoder-decoder model, causal convolution is involved, where a 1D fully convolutional network architecture is used. A crucial characteristic is that the output at time t is convolved only with the elements that occurred before t. [4]

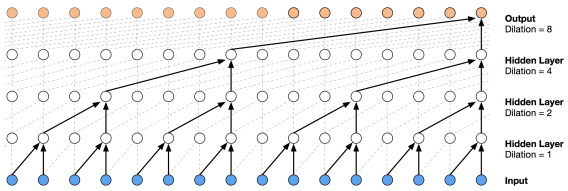

Figure 3: Stack of dilated causal convolutional layers.

model	MSE	MAE
Stacked LSTM seq2seq Autoencoder	0.0052	0.0631
TCN	0.0038	0.0594

Table 1: Private test set scores for various models on 20th time series of category F.

2 Dataset

2.1 Data transformations

Data transformation is a crucial process that involves mathematical equations to modify data, enabling it to conform to specific statistical assumptions, such as normality, homogeneity, and linearity. This transformations enhance the data's quality and utility, facilitating the extraction of meaningful insights and reducing noise that might impede accurate analysis. We tried different data transformation techniques starting from data scaled between 0 and 1. **MinMaxScaler** rescales the dataset such that all values are in the range [0, 1] after computing the minimum and maximum value. However it does not reduce the effect of outliers, but it linearily scales them down into a fixed range, where the largest occuring data point corresponds to the maximum value and the smallest one corresponds to the minimum value. Unlike the previous scaler, the centering and scaling statistics of **RobustScaler** are based on percentiles and are therefore not influenced by a small number of very large marginal outliers. This Scaler removes the median and scales the data according to the quantile range (defaults to IQR: Interquartile Range). The IQR is the range between the 1st quartile (25th quantile) and the 3rd quartile (75th quantile). Outliers can skew a probability distribution and make data scaling using standardization, calculated by subtracting the mean value and dividing by the standard deviation, difficult as the calculated mean and standard deviation will be skewed by the presence of the outliers. One approach to standardizing input variables in the presence of outliers is to ignore the outliers from the calculation of the mean and standard deviation, then use the calculated values to scale the variable. This can be achieved by calculating the median (50th percentile) and the 25th and 75th percentiles. The values of each variable then have their me-

dian subtracted and are divided by the interquartile range (IQR) which is the difference between the 75th and 25th percentiles.

$$value = (value–median)/(p75–p25)$$

The resulting variable has a zero mean and median and a standard deviation of 1, although not skewed by outliers and the outliers are still present with the same relative relationships to other values [5]. **QuantileTransformer** applies a non-linear transformation such that the probability density function of each feature will be mapped to a uniform or Gaussian distribution. In this case, all the data, including outliers, will be mapped to a uniform distribution with the range [0, 1], making outliers indistinguishable from inliers. But contrary to RobustScaler, QuantileTransformer will also automatically collapse any outlier by setting them to the a priori defined range boundaries (0 and 1). This can result in saturation artifacts for extreme values. We implemented these transformations by importing the corresponding method from sklearn.preprocessing [6]. In out tests QuantileTransformer outperforms the alternatives.

Data transformation	MSE	MAE
Original dataset	0.0223	0.1294
MinMaxScaler	0.0049	0.0594
RobustScaler	0.0054	0.0661
QuantileTransformer	**0.0053**	0.0637

Table 2: Private test set scores for Stacked LSTM seq2seq Autoencoder model on 20th time series of category F.

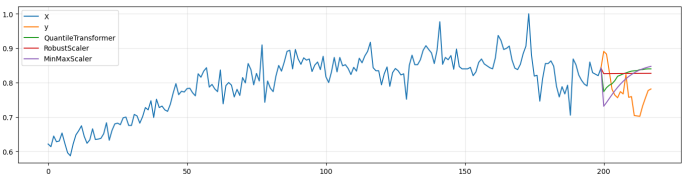

Figure 4: Prediction of last 18 timesteps of 20th time series of category F.

3 Contributions

3.1 Authors

- **Aldo Cumitini**: Tested different stacked LSTM models, parameters tuning , Latex expert, contributed writing the report.
- **Davide Gesualdi**: Tested LSTM models, Stacked LSTM seq2seq Autoencoder, TCN, data transformation techniques and contributed writing the report.
- **Gabriele Passoni**: Tested bi-directional LSTM network, tested concatenation of pre-trained forecasters, tested MinMaxScaler, contributed writing the report.
- **Roberto Riva**: Tested EfficientNetV2L and VGG19, dataset cleaning expert, contributed writing the report.

3.2 Materials

- Github
- Google Drive

3.3 Acknowledgements

We extend our heartfelt gratitude to ChatGPT for its invaluable assistance throughout this project. Its contributions were instrumental in refining our code and enhancing the quality of our manuscript by meticulously correcting grammatical errors in the English language.

References

[1] Nir Arbel. *How LSTM networks solve the problem of vanishing gradients*. https://medium.datadriveninvestor.com/how-do-lstm-networks-solve-the-problem-of-vanishing-gradients-a6784971a577. 2018.

[2] Matteo Matteucci Riccardo Ughi Eugenio Lomurno. *Two Steps Forward and One Behind: Rethinking Time Series Forecasting with Deep Learning*. https://arxiv.org/abs/2304.04553. 2023.

[3] Suggula Jagadeesh. *Multivariate Multi-step Time Series Forecasting using Stacked LSTM sequence to sequence Autoencoder in Tensorflow 2.0 / Keras*. https://www.analyticsvidhya.com/blog/2020/10/multivariate-multi-step-time-series-forecasting-using-stacked-lstm-sequence-to-sequence-autoencoder-in-tensorflow-2-0-keras/. 2023.

[4] Dr Barak Or. *Temporal Convolutional Networks, The Next Revolution for Time-Series?* https://medium.com/metaor-artificial-intelligence/temporal-convolutional-networks-the-next-revolution-for-time-series-8990af826567. 2020.

[5] Jason Brownlee. *How to Scale Data With Outliers for Machine Learning*. https://machinelearningmastery.com/robust-scaler-transforms-for-machine-learning/. 2020.

[6] Lars Buitinck et al. “API design for machine learning software: experiences from the scikit-learn project”. In: *ECML PKDD Workshop: Languages for Data Mining and Machine Learning*. 2013, pp. 108–122.