Time Series Forecasting
Artificial Neural Networks and Deep Learning – A.Y. 2023/2024

Carlo Fabrizio[*], Riccardo Malpiedi[†] and Niccolò Perrone[‡]
M.Sc. Computer Science and Engineering - Mathematical Engineering, Politecnico di Milano - Milan, Italy
Email: [*]carlo.fabrizio@mail.polimi.it, [†]riccardo.malpiedi@mail.polimi.it, [‡]niccolo.perrone@mail.polimi.it
Student ID: [*]10747982, [†]10940821, [‡]10707561
Codalab Group: "Neural Net Ninjas"

1. Task Analysis

The task encompassed predicting the next 18 steps for a dataset comprising 48,000 time series categorized into six distinct categories. The aim was to minimize forecasting errors, specifically the Mean Squared Error and the Mean Absolute Error. The category distribution across the dataset was unbalanced but despite this, we did not implement any weighting as we considered it to be of limited impact. The given time series were of fixed length (2776), often padded with zeros, yet retrievable through provided valid periods.
Challenges revolved around handling this padding and categorical aspects, requiring decisions on modeling approaches. Strategies ranged from training separate models per category to considering embedding categories in models or opting for more general models. Ultimately, it was found that training the models without categorial distinctions yielded the most favorable results, striking a balance between simplicity and predictive performance.

2. Time Series Analysis

2.1. Autocorrelation Analysis

In order to assess the optimal time window to consider for the given time series, we analysed the autocorrelation function. In particular, we developed a function to automatically extract the time window given the autocorrelation function $A(k)$ of a time series and a threshold T. The latter function works as follow: scan backword $A(k)$, i.e. for k^* that goes from k_{max} to 0, until $A(k^*) \geq T$, the time window is then k^*. In order to get a better estimate of the optimal time window, we computed the autocorrelation on a denoised version of the time series, obtained by means of a moving avarage operation. We tried to cut all the sequences according to their specific time window and use a mask in order to deal with different input lengths but the results were not promising. Afterward, we examined the distribution of time windows across the entire dataset, determining that the most effective time window spanned approximately 250 time steps, so we decided to use as time window a fixed value of 200 (aligned with the test set) for our trainings.

2.2. Denoising

We tried to apply a sort of preprocessing to the time series with the aim of reducing the unpredictable white noise that intrinsically affects the time series, by applying a moving avarage operation. We developed the latter operation as a layer in the Networks (either a Conv1D layer with avarage filter or an Avarage Pooling operation, with causal padding). The denoising operation was effective only if applied to simple models as the simple vanilla with LSTM, but was bad performing if applied to more complex models like resnet style with LSTM or Transformers, probably because those models were already able to capture the noise information themselves.

3. Models

In this section all the models that we have implemented are described, starting from the easiest ones.

3.1. D-Linear and N-Linear

We tested straightforward linear models, more precisely, DLinear and Nlinear models, which have been highlighted in some papers for outperforming complex transformers across numerous benchmark datasets [3] [1]. The results were decent; however, other models demonstrated superior performance.

3.2. Vanilla LSTM

We started by constructing a straightforward Sequential network. This neural network initially employed unidirectional LSTMs, but we transitioned to Bidirectional LSTMs for their superior ability to capture bidirectional dependencies within sequential data. Dropout layers were integrated to prevent overfitting, enhancing model generalization. The final architecture consists of two Bidirectional LSTM layers and Dense layers to forecast nine steps ahead. Additionally, we experimented with a denoising filter as the initial layer, resulting in a slight performance improvement.

3.3. Resnet style with 1D-conv and LSTM

This architecture's advantage lies in its ability to capture both spatial and temporal information in the time series data effectively. This model is a variant of the ResNet architecture fused with a Bidirectional LSTM and it was built in the following way:

3.3.1. Residual Blocks. Three residual blocks consisting of a series of Convolutional layers followed by Batch Normalization and ReLU activation.

3.3.2. Shortcuts. Each block has a shortcut connection to manage vanishing gradients which takes the initial input and combines it with the output of its convolutional layers.

3.3.3. LSTMs. After the Residual Blocks, the model includes a GlobalAveragePooling1D layer to aggregate features. The output is then connected to a block of two Bidirectional LSTM layers. The first one returns sequences, and the second one only returns the final output.Finally, there is a Dense layer acting as the output layer, providing predictions for the next 9 time steps.

3.4. Informer Encoder

Inspired by the paper "Informer: Beyond Efficient Transformer for Long Sequence Time-Series Forecasting"[2], we developed a model based on the encoder structure of the Informer. More precisely, our model architecture mirrors the Informer's encoder design, alternating blocks of Convolutions, Multi-head Attentions and Maxpooling, ending with a GlobalAvaragePooling and a final Dense layer.

Maxpooling layers play a pivotal role in this model, introducing a "distilling" operation that selectively emphasizes dominant features within each sequence. By distilling the most salient information while reducing spatial dimensions, this step enhances the model's capacity to focus on strong temporal patterns.

In addition to that, the model takes advantage of a positional sinusoidal encoding, that is added to the input, to better represent the time dependence.

3.5. Transformer

We decided to make some experiments with transformers[1] and we managed to get good results with the architecture described in this section.

3.5.1. Positional encoding. Since transformers don't inherently understand the order of the sequence, as they process inputs in parallel, we used positional encoding to provide information about the position of the elements of the input sequence.

3.5.2. Transformer encoder and decoder.

We used 3 encoder and decoder blocks. The encoder processes the input sequence and captures contextual information for each token and the decoder generates the output sequence based on the encoded information. The encoder contains a multi-head self-attention mechanism followed by a feedforward neural network, while the decoder includes multi-head self-attention, multi-head attention over the encoder's output and a feedforward neural network.

3.5.3. Last layers.

We used a Global Average Pooling to reduce the temporal dimension of the output sequence followed by the output layer (a simple dense) to get the final sequence.

3.6. ResNet-style with attention

The proposed model aims to expoits once again the ResNet-inspired residual connections for effective feature extraction and the integration of attention mechanisms to enhance sequence comprehension by focusing on relevant parts of the sequence.

3.6.1. Convolutional Layers.

The model begins with two convolutional layers, each followed by batch normalization. These layers apply filters to the input sequence, extracting important features using ReLU activation.

3.6.2. Self-Attention Layer.

A multi-head self-attention mechanism is employed, utilizing eight heads and a key dimension of 32.

3.6.3. Residual Connections.

A residual connection is established from a previous convolutional layer to the output of the self-attention layer.

3.6.4. Additional Convolutional Layers.

Following the residual connection, two more convolutional layers with batch normalization are applied to further refine the learned representations.

3.6.5. Flatten and Dense Layers.

The model concludes with flattening the output and passing it through a dense layer with linear activation to predict nine steps ahead in the sequence.

4. Ensemble

We tried to combine our model predictions by doing the average of the predicted sequences and we managed to improve our results. We weighted our model contributions in different ways and after some attempts we got the best configuration that is described in the following section.

5. Best Models

Our best model for phase 1 (MSE of 0.0044) is an ensemble of ResNet-style with LSTMs, ResNet-style with Attention, Informer and Transformer with equal contribution for each model. In phase 2 we got our best result (MSE of 0.0091) with an ensemble of the same models weighted, respectively: 0.4, 0.25, 0.175, 0.175.

During the second phase, we encountered a challenge: predicting 18 time steps instead of 9. Initially, we attempted to train our models using identical hyperparameters to directly forecast all 18 time steps in one go. However, this approach didn't yield promising results, likely due to the intricate interconnections between the time steps to predict and other hyperparameters.

As an alternative, we opted to use the same models employed in the first phase. Initially, these models forecasted the subsequent 9 time steps, and then we concatenated these predictions with the input to predict the following 9 time steps.

6. Results Phase 1

	Validation Set	Test Set
	MSE	MSE
Vanilla LSTM	0.0057	0.0064
Vanilla LSTM denoising	0.0055	0.0063
Resnet LSTM	0.0055	**0.0047**
Informer	0.0052	<u>0.5</u>
Resnet LSTM Attention	0.0050	0.0052
Transformer	0.55	0.53
Nlinear	0.0063	?
Dlinear	0.0062	?

Figure 1: Comparison of models' performances.

7. Contributions

All of us contributed on the ensemble part, testing different configurations of the weights for the models.

- Carlo Fabrizio: developed the data analysis part for the time windows and denoising; developed Dlinear and Nlinear models; developed the Informer style model.
- Niccolò Perrone: built the vanilla models; implemented the Resnet-style approach; tested several architectures including attention mechanism.
- Riccardo Malpiedi: made experiments with transformers and performed explainability of models.

References

[1] Lei Zhang Ailing Zeng, Muxi Chen and Qiang Xu. Are transformers effective for time series forecasting?, 2022.

[2] Jieqi Peng Shuai Zhang Jianxin Li Hui Xiong Wancai Zhang Haoyi Zhou, Shanghang Zhang. Informer: Beyond efficient transformer for long sequence time-series forecasting, 2021.

[3] Eugenio Lomurno Riccardo Ughi and Matteo Matteucci. Two steps forward and one behind: Rethinking time series forecasting with deep learning, 2023.