# Modeling stochastic processes by simultaneous optimization of latent representation and target variable

Jakob Jelenčič
Artificial Intelligence Laboratory
Jozef Stefan International Postgraduate School
Jozef Stefan Institute
Ljubljana, Slovenia
jakob.jelencic@ijs.si

Dunja Mladenić
Artificial Intelligence Laboratory
Jozef Stefan International Postgraduate School
Jozef Stefan Institute
Ljubljana, Slovenia
dunja.mladenic@ijs.si

## ABSTRACT

This paper proposes a novel method for modeling stochastic processes, which are known to be notoriously hard to predict accurately. State of the art methods quickly overfit and create big differences between train and test datasets. We present a method based on simultaneous optimization of latent representation and the target variable that is capable of dealing with stochastic processes and to some extent reduces the overfitting. We evaluate the method on equities and cryptocurrency datasets, specifically chosen for their chaotic and unpredictable nature. We show that with our method we significantly reduce overfitting and increase performance, compared to several commonly used machine learning algorithms: Random forest, General linear model and LSTM deep learning model.

## 1. INTRODUCTION

Time series prediction has always been an interesting challenge. Deep learning structures that are designed for time series are prone to overfitting. Especially if the underlying time series is stochastic by nature. Every young researcher's first attempt when dealing with time series, was trying to learn a time series model that will predict future prices; whether in equities, commodities, forex or cryptocurrencies. Unfortunately it is not that simple. One can easily build a near perfect model on the train dataset just to find it is completely useless on the test dataset.

We propose a novel method that is capable of effectively combatting the overfitting, especially this proves to be a difficult task when one is dealing with a problem directly applicable in practical situations. The main idea is to add noise from the same distribution as the training data and then at the same time optimize the target variable and the latent representation with the help of the autoencoder. The longer the training goes, the lower is the amplitude of noise and the less focus is on the optimization of the representation.

We have evaluated the proposed method on an equities dataset and a cryptocurrency dataset, in both cases achieving extraordinary results on the test dataset. We have also shown the importance of noise distribution and how the de-noising fails if the distributions of the data and noise do not align.

The rest of the paper is organised as follows. Section 2 describes the data we were using. In section 3 we introduce the proposed method. In section 4 we present empirical results. In section 5 we conclude by pointing out the main results and defining guidance for the future work.

## 2. DATA

The proposed method works well for stochastic processes. Equities are supposed to follow some form of stochastic process [9], either the Black-Scholes one or some more complex process with unknown formulation. In order to evaluate our method, we have collected daily data of more than 5000 equities listed on NASDAQ from 2007 on. The data is freely available on the Yahoo Finance website [2]. We transformed the data using technical analysis [10] and for test set took every instance that happened after 2019. We calculated moving average using 10 days closing price then tried to predict the direction of the change of this trendline.

The equity data turned out to be a little bit timid, not chaotic enough to demonstrate the full ability of the proposed method. This is why we also collected minute data of cryptocurrencies Ethereum and Bitcoin and used the method on them as well. Data is available on the crypto exchange Kraken [1]. We used the same transformation as for the equities, but with a bit quicker trend. This time the target variable was change in the trendline in the next 6 hours. For the test set we took every instance that has time stamp after December 2020.

The reader should note that the end goal is not to accurately predict future equity price, since that is next to impossible. As soon there is a pattern, someone will profit from it and then the pattern will change. By predicting the future trend line, one can obtain a significant confidence interval and

estimates of where the price could be, and then design for example a derivative strategy that searches for favourable risk versus rewards trades.

# 3. PROPOSED METHOD

We propose the method designed for prediction of stochastic processes. The method achieves significant results improving the metrics and loss functions on unseen data, where standard deep learning is prone to over-fit. The main advantage is reducing the gap between training data and testing data, sometimes to a degree where one sacrifices a little bit on the train side to actually have the model outperforming it on test data. This is very important in time series, where a prediction model is usually just one part of a bigger strategy and where the train over-fit is the biggest issue. For example, designing a trading strategy on over-fitted predictions, that kind of mistake can lead to huge capital losses.

The proposed method can be broken down into 3 important parts: normalization, noise addition and additional optimization of latent representation. Each part can be easily integrated into an already existing pipeline.

## 3.1 Empirical normalization

Normalization plays an important role in deep learning models. It was shown that normalization significantly speeds up the gradient descent, almost independently of where normalization takes place. It can be weight normalization [11] during the actual optimization, or it can be the batch normalization [8], or just normalization of the whole input data [7]. In the proposed method it is important that the 3 dimensional input data comes from the same distribution as the generated noise. Since it is fairly straightforward to sample data from a 3 dimensional normal distribution, we normalize input data using an empirical cumulative distribution function [12] and empirical copula [4] [5]. We align all central moments of the unknown distribution to the ones from centered and standardised normal distribution. The normalization takes place before the data is reshaped to 3 dimensional tensor.

## 3.2 Noise addition

Introduction of the noise is not new in unsupervised learning and it was shown that it has a positive effect [14]. Adding noise to input data and then forcing the model to learn how to ignore it has a lot of success in generative adversarial networks [3], where convergence can be very tricky to achieve. We transformed that idea and embedded it into supervised learning procedure. The noise addition is described in Algorithm 1.

In Algorithm 1 we will use the following abbreviations.

- $X = [bs, ts, np]$ stands for the input tensor with 3 dimensions; batch size, time steps and number of features used for predictions.
- $\alpha$, $\beta$ are parameters that control how fast noise will decrease during the training procedure. They should be between 0 and 1, where lower value correspond to a faster decrease in the amplitude of the added noise.
- $mvn$ stands for function sampling from a two dimensional correlated Gaussian distribution, where $\Sigma$ is the covariance. $matmul$ stands for matrix multiplication.

**Algorithm 1** Noise definition

Inputs: $X$, $\alpha$, $\beta$, $epoch$
$Y = [ts, ts, np]$ ▷ Array for holding Cholesky decompositions of time correlation matrices.
**for** $t \in \{1, \ldots, np\}$ **do**
  $\Sigma_t = cov(X[, , t])$
  $Y[, , t] = chol(\Sigma_t)$ ▷ In practice the closest positive definite matrix of $\Sigma_t$ is computed before the Cholesky decomposition.
**end for**
$Z = [bs, ts, np]$ ▷ Array for holding noise samples.
**for** $i \in \{1, \ldots, ts\}$ **do**
  $\Sigma_i = cov(X[, i, ])$
  $Z[, i, ] = mvn(bs, \Sigma_i)$
**end for**
**for** $j \in \{1, \ldots, np\}$ **do**
  $Z[, , j] = matmul(Z[, , j], Y[, , j])$ ▷ Correcting initially independent noise samples with respect to time.
**end for**
**for** $w \in \{1, \ldots, ts\}$ **do**
  $Z[, w, ] = Z[, w, ] * ((\beta^{ts-w} \cdot \alpha^{epoch}) \cdot sd)$ ▷ Decrease the noise during the training procedure.
**end for**
$R = X + Z$
Return $R$.

## 3.3 Optimization of latent representation

The most common issue with deep learning optimization is falling into a local optimum and being unable to move past it [13]. We introduce autoencoder part into the optimization procedure in order to force the model to shift from going directly to local optimum to learning the latent representation first. We expect that this combined with the addition of noise, will force the model first to learn how to ignore the noise that we added and the noise that is already in the data by nature of the stochastic process [15]. We optimized the model using the Adam optimizer [6]. The loss function used in optimization is defined like:

$$L = L_Y + W_{ae} \cdot decay^{epoch} \cdot L_{ae},$$

where $L_Y$ stands for the supervised loss function which will depend on the problem while $L_{ae}$ stands for the loss between encoded output and input data. Decay weight is decreasing the longer the training goes on.

# 4. RESULTS

We have divided the results section into 2 parts: unsupervised and supervised. In the first we demonstrate why the noise distribution is important. For the unsupervised part, due to hardware constraints, we have only used the cryptocurrency dataset since we deemed it more demanding than the equity one. In the second, we demonstrate how the our method increases test metric on both datasets.

## 4.1 Unsupervised learning results

In order to test the efficiency of distributed noise versus just random noise, we created 3 models. The baseline model was a deep learning model with 3 stacked LSTM layers, encoded layer, then again 3 stacked LSTM for decoded output. We have used Adam as optimizer. As loss function we used mean-squared error. We have stopped the learning after there was no improvement for 25 epochs on the validation set. The validation set was randomly taken out of the train set. Parameters $\alpha$ and $\beta$ were both set to 0.99 and $sd$ was initially set to 1.25. The noise decreases with learning procedure. Interestingly keeping noise constant did not achieve any results.

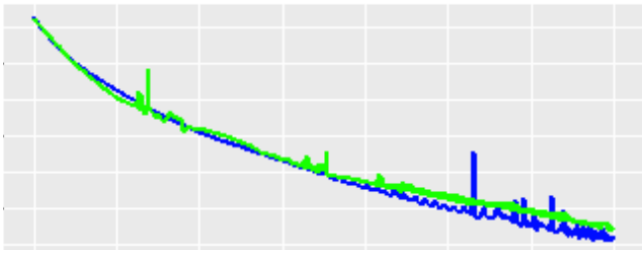

**Figure 1: Test loss of autoencoder model with random noise (green) versus no noise (blue).**

Initially we have tested baseline model versus de-noising model but with uncorrelated noise. In the Figure 1 is plotted the de-noising test loss function in green colour and the baseline test loss function in blue. Training was stopped relatively early compared to Figure 2 and it is also obvious that de-noising test loss is even worse than that of the classic autoencoder.

In the second example we switched from uncorrelated noise to the noise with same distribution as input data. As is apparent on Figure 2, where again we have de-noising test loss plotted with green and classic test loss with blue, the de-noising autoencoder achieved lower test loss than the classic one.

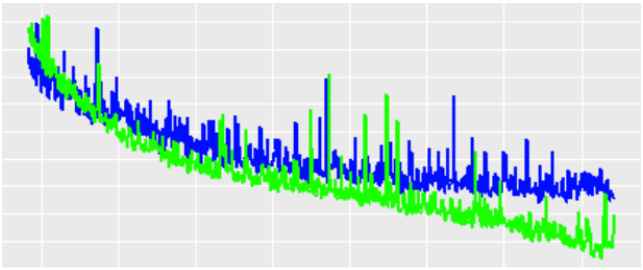

**Figure 2: Test loss of autoencoder model with correlated noise (green) versus no noise (blue).**

What we expected is that then the train and validation losses will be worse than with the classic autoencoder. Surprisingly, that was not the case. With the de-noising autoencoder using noise with the same distribution as the input data, both train and validation losses were better than with classic one. This result is definitely worth further investigation and experimentation.

## 4.2 Supervised learning results

In the previous section we have shown that the distribution of the noise matters. In this section we will show that noise combined with optimization of latent representation significantly improves metrics on unseen data. Similarly as before, $\alpha$ and $\beta$ were both set to 0.99 and $sd$ was initially set to 1.25. From our experience this setting achieves the best results, but further exploration needs to be done. $W_{ae}$ was initially set to 5 and $decay$ to 0.95.

Since we now operate in a supervised environment, we can compare our models to the majority class. But to really demonstrate the effectiveness of the method, we chose to compare the following models:

- Majority class, which serves as a sanity check.
- Random Forest with 500 trees.
- Generalized linear model.
- Deep learning model with 3 stacked LSTM layers.
- Deep learning model with 3 stacked LSTM layers and optimization of latent representation.
- Deep learning model with 3 stacked LSTM layers and correlated noise addition.
- Finally, deep learning model with 3 stacked LSTM layers and correlated noise addition and optimization of latent representation.

All 4 of the deep learning models are identical, all are optimized with Adam and categorical cross entropy was used as a loss function for the supervised part and mean squared error for the autoencoder part. Initially we have only tested the models on equities data, but it turned out that the equities were not chaotic enough. By that we mean that especially with deep learning models the difference between train and test loss was not so big that it would be problematic. From previous work experience we know that overfit is a big issue in cryptocurrency dataset, so then we decided to test that dataset in a supervised setting as well. All models were trained three times on each dataset and the results in Table 1 and Table 2 are the averages of the 3 runs.

In Table 1 we show the results from the equity dataset. Our method managed to improve test accuracy (from 0.673 to 0.682) without decreasing train accuracy (0.681). Maintaining test accuracy and keeping it comparable to test one is important if one needs to build additional strategy upon predictions. Just noise addition slightly improved the results (from 0.673 to 0.675), while just the optimization of the latent distribution does not improve anything.

**Table 1: Supervised results on equity dataset.**

| Method | Train Accuracy | Test Accuracy |
|---|---|---|
| Majority | 0.513 | 0.537 |
| Random Forest | 0.649 | 0.655 |
| GLM | 0.664 | 0.655 |
| LSTM | 0.681 | 0.673 |
| latent LSTM | 0.633 | 0.673 |
| noise LSTM | 0.681 | 0.675 |
| latent noise LSTM | 0.681 | 0.682 |

In Table 2 we show results from the cryptocurrency dataset. Similar as on the equity dataset, our method behaves as intended on the cryptocurrency dataset as well. We can see reduced overfitting that is apparent in the normal LSTM model. With those results we can conclude that the proof of concept works, but for additional claims we will need more testing and deeper parameter analysis.

**Table 2: Supervised results on cryptocurrency dataset.**

| Method | Train Accuracy | Test Accuracy |
|---|---|---|
| Majority | 0.512 | 0.556 |
| Random Forest | 0.689 | 0.692 |
| GLM | 0.682 | 0.695 |
| LSTM | 0.754 | 0.696 |
| latent LSTM | 0.736 | 0.683 |
| noise LSTM | 0.697 | 0.695 |
| latent noise LSTM | 0.706 | 0.714 |

It is interesting to point out that with the proposed method the test loss on cryptocurrency dataset was 0.552, while train loss was 0.592. While 0.552 was the best loss any deep learning model achieved, that wide difference indicates that we could improve our model even further by fine tuning the parameters.

# 5. CONCLUSIONS AND FUTURE WORK

In this work we have introduced and demonstrated how the addition of noise and simultaneous optimization of latent representation and target variable reduce overfitting on time series data. In the unsupervised case we have shown that the distribution of the noise matters and the input data must align to achieve maximum effect from the noise addition.

In the future work we have to estimate the effect of the newly introduced parameters on method's convergence. At the same time we need to explore how the method behaves when embedded into larger models, transformers for example. We also need to evaluate the method in datasets that are by nature stochastic but do not come from the financial domain. Finally, we need to evaluate our method on a dataset that is not stochastic.

# 6. ACKNOWLEDGMENTS

This work was supported by the Slovenian Research Agency. We also wish to thank prof. dr. Ljupčo Todorovski for his help, especially with unsupervised results.

# 7. REFERENCES

[1] *Kraken exchange*. `https://www.kraken.com/`.

[2] *Yahoo Finance*. `https://finance.yahoo.com/`.

[3] A. Creswell, T. White, V. Dumoulin, K. Arulkumaran, B. Sengupta, and A. A. Bharath. Generative adversarial networks: An overview. *IEEE Signal Processing Magazine*, 35(1):53–65, 2018.

[4] P. Jaworski, F. Durante, W. K. Hardle, and T. Rychlik. *Copula theory and its applications*, volume 198. Springer, 2010.

[5] H. Joe. *Dependence Modeling with Copulas*. CRC Press, 2014.

[6] D. Kingma and J. Ba. *Adam: A Method for Stochastic Optimization*. 2014. `https://arxiv.org/abs/1412.6980`.

[7] K. Y. Levy. The power of normalization: Faster evasion of saddle points. *arXiv preprint arXiv:1611.04831*, 2016.

[8] M. Liu, W. Wu, Z. Gu, Z. Yu, F. Qi, and Y. Li. Deep learning based on batch normalization for p300 signal detection. *Neurocomputing*, 275:288–297, 2018.

[9] R. C. Merton. Option pricing when underlying stock returns are discontinuous. *Journal of financial economics*, 3(1-2):125–144, 1976.

[10] J. J. Murphy. *Technical Analysis of the Financial Markets: A Comprehensive Guide to Trading Methods and Applications*. New York Institute of Finance Series. New York Institute of Finance, 1999.

[11] T. Salimans and D. P. Kingma. Weight normalization: A simple reparameterization to accelerate training of deep neural networks. *Advances in neural information processing systems*, 29:901–909, 2016.

[12] B. W. Turnbull. The empirical distribution function with arbitrarily grouped, censored and truncated data. *Journal of the Royal Statistical Society: Series B (Methodological)*, 38(3):290–295, 1976.

[13] R. Vidal, J. Bruna, R. Giryes, and S. Soatto. Mathematics of deep learning. *arXiv preprint arXiv:1712.04741*, 2017.

[14] P. Vincent, H. Larochelle, Y. Bengio, and P.-A. Manzagol. Extracting and composing robust features with denoising autoencoders. In *Proceedings of the 25th international conference on Machine learning*, pages 1096–1103, 2008.

[15] N. Wax. *Selected papers on noise and stochastic processes*. Courier Dover Publications, 1954.