PREDICTING CDS SPREADS WITH DEEP LEARNING ARCHITECTURE

AN APPLICATION OF TIME-SERIES FORECASTING

WEIMU LEI
leiweimu@stanford.edu

Abstract
Deep learning architectures have been widely applied to and studied in financial contexts. One of a handful exceptions is credit-risk prediction. We implement and adapt RNN-based models to predict CDS spread using a few hand-picked features. These architectures generally achieve commensurate performance, but given relative small data set, they all overfit the training data. Qualitative analysis on predicted values from trained models suggests attention mechanism does allow models to make extreme predictions that more closely follow the actual market fluctuations.

INTRODUCTION

Deep learning has garnered great attention and enthusiasm in many branches of social science, and finance is one of such subjects that have largely embraced the technology. Many breakthroughs in deep learnings have made their way into various financial applications; credit risk assessment, accounting anomaly detection, stock price prediction, and financial asset management are examples of these applications. Yet, despite the ardent interest in the risk management community, the research work on CDS (credit default swap) spread forecasting using deep learning architectures is surprisingly anemic.

CDS has become a consequential financial instrument since its invention in the mid-1990s. With $8 trillion notional value outstanding in 2018, CDS market ranks the third biggest over-the-counter derivatives market in the world[*]. Its global importance comes with heightened regulatory scrutiny, especially after the revelation of the intricacy of its involvement in the 2008 financial crisis. There is little denying, however, that, given the pivotal role of CDS in the functioning of the global financial markets, being able to gauze CDS spread movement would benefit credit risk mitigating and hedging.

This project attempts to fill the void with a recurrent neural network architecture and a handful of features inspired by empirical results and theoretical models. The core objective of this project is to investigate if an LSTM model, in the task of predicting future CDS spreads, could achieve comparable performance of the current models that don't rely on deep learning. The attention mechanism found in both A-LSTM and DA-RNN makes model to act more decisively in anticipation of a potential market swing.

RELATED WORK

CDS spread is an active research topic in both academia and industry, but as stated earlier publications on CDS spread in conjunction with A.I. is rather scant. Gündüz et al. [1] published their results in 2011 concluding that support vector regression outperformed classical models (structural model and reduced form model) in time-series analysis. Mercadier and Lardy [2] achieve promising result in CDS approximations using proposed Equity-to-Credit formula and random forest regression.

Other research works primarily dedicate to classical statistical models and determinant analysis. Work from Shaw et al. [3] suggests the fitted Nelson Siegel model on CDS spreads has predictive power.

[*] Boyarchenko, N., Costello, A. M., & Shachar, O. (2019). The Long and Short of It: The Post-Crisis Corporate CDS Market. *Federal Reserve Bank of New York Staff Reports*, No.879

Avino et al. [4] demonstrated that linear models often outperform non-linear models like Markov switching. Galil et al. [5], and Pereira et al. [6] found statistically significant and consistent association between CDS spreads and market-based variables and market liquidity. Doshi et al. [7] reaffirmed the empirical observations that much of the variation in CDS can be explained away by macro-economic and firm-specific information over time.

This project seeks to leverage these great works in the following ways:

- CDS spread determinant analysis informs us on feature selection; the features to be used as input will either have empirical evidence or theoretical foundation. This not only narrows down the scope of feature search, but also ensures selected features are relevant.
- Existing CDS spread models will serve as benchmark to evaluate our model performance.

DATASET

We hone in on 5-year CDS spreads on investment grade bond issuances from North American entities. In particular, we focus on a particular credit index – `CDX.NA.IG` published and managed by Markit. The index consists of investment grade CDSs from 125 entities in the region.

We acquired `CDX.NA.IG` pricing data (expressed in basis point) from the Bloomberg terminal (courtesy of GSB library). Since the index adjusts its constituent list every 6 months (technical term "roll"), we will use the roll-adjusted mid daily closing price for our purpose. Due to data availability, we were unable to collect `CDX.NA.IG` pricing data before September 11, 2011. As a result, our dataset spans from September 11, 2011 to October 30, 2020 with a total of 2282 data points (discussed in next section). We reserve about 15% of the dataset (or 343 data points) for testing.

FEATURES & TRANSFORMATIONS

Inspired by empirical studies and theoretical models, we includes the following financial information/economic indicators as features:

- `CDX.NA.IG` **absolute spread**
  The absolute spread is calculated as the absolute difference between `CDX.NA.IG` closing bid and ask prices. It is used to estimate CDS market illiquidity.
- `SPX` **– S&P 500 equity index of large U.S. companies**
  The index is published by S&P Global, and contains 500 company stocks. It is used as a measure for equity market performance.
- `VIX` **– CBOE volatility index of the U.S. equity market**
  The index tracks the implied volatility from S&P 500 options. The index is published by Chicago Board Options Exchange. It is pertinent to financial derivatives pricing.
- **3-month U.S. treasury bill yield**
  The yield is the return an investor realizes on a 3-month U.S. treasury bill. The data is published by U.S. Department of the Treasury. It is used as a proxy for risk-free rate.
- **parameters of the Nelson–Siegel–Svensson model on treasury yields**
  Svensson [8] extended previous function proposed by Nelson and Siegel [9] on estimating forward interest rates:

$$\begin{aligned} y(m) = \beta_0 + & \beta_1 \frac{1-\exp(-m/\tau_1)}{m/\tau_1} \\ & + \beta_2 \left(\frac{1-\exp(-m/\tau_1)}{m/\tau_1} - \exp(-m/\tau_1) \right) \\ & + \beta_3 \left(\frac{1-\exp(-m/\tau_2)}{m/\tau_2} - \exp(-m/\tau_2) \right) \end{aligned}$$

  where m is the maturity (tenor) and $y(m)$ is the corresponding yield. The extended function is then widely accepted among practitioners to model yield curve. Here we estimate parameters

$\{\beta_0, \beta_1, \beta_2, \beta_3, \tau_1, \tau_2\}$ obtained from a grid search of a series ordinary least squares estimates proposed in the original paper. These parameters together characterize the yield term structure (data also published by the U.S. Treasury).

We also performed transformations and normalizations to these features. Specifically, instead of raw prices (`CDX.NA.IG` roll-adjusted prices and `SPX` prices), we followed financial time-series analysis convention and opted for log returns:

$$r_t := \ln\left(\frac{p_t}{p_{t-1}}\right)$$

where p_t is the price at time t. Then we scaled all ten features as well as the time-series we are predicting itself (a total of eleven). We transformed dataset based on the training portion. For instance, for the j^{th} input feature of the i^{th} example in the training set:

$$\tilde{x}_j^{(i)} = \frac{x_j^{(i)} - \bar{x}_j}{s_{x_j}} \qquad \bar{x}_j = \frac{1}{m}\sum_{i=1}^{m} x_j^{(i)} \quad s_{x_j} = \sqrt{\frac{1}{m-1}\sum_{i=1}^{m}\left(x_j^{(i)} - \bar{x}_j\right)^2}$$

In prediction time, we will re-scale the new incoming data using the mean and standard deviation information from the training portion.

MODELS & TRAINING

We apply RNN-based architectures to the prediction task and investigate model performance. As a benchmark, we include classic models inspired by research. Mean Squared Error (MSE) is the primary performance metric. We also report Mean Absolute Error (MAE) as well as Mean Absolute Percentage Error (MAPE) as additional metrics. Calculations of these metrics are provided in Appendices section.

Baseline Models

Following models will help formulate the target metric we evaluate RNN-based architectures against.

- support vector regression (SVR) from Gündüz et al. [1];
- autoregressive model (AR) recommended in the publication from Avino et al. [4].

For the SVR approach, we fit two individual models: one with full set of features and one with only the time series (`CDX` log return) itself. This is because the original paper showing the efficacy of SVR-based models uses only a single-feature input. Same performance may not be replicable to multi-feature set-up. SVR algorithm attempts to find function that best fits training points outside of a ε-tube in a higher dimensional feature space induced by a kernel function.

$$f(\vec{x}) = \sum_{i=1}^{m}\left(a^{(i)\star} - a^{(i)}\right)\cdot k\left(\vec{x}^{(i)}, \vec{x}\right)$$

where $\vec{x}^{(i)}$ is the i^{th} example of the training set, $a^{(i)\star}$ and $a^{(i)}$ are corresponding Lagrange multipliers solved using Karush-Kuhn-Tucker conditions, and k is kernel function. Suggested hyper-parameters for the SVR-based models in the paper are: linear kernel function, cost $C = 10$, ε-band $= 10^{-4}$.

For the AR approach, we only use time-series as input. AR models posit that the time-series is dependent linearly with its own previous values and a random term. The use of AR model is warranted as research from Byström [10] shows positive autocorrelation in `iTraxx Europe` spread – another CDS index. Avino et al. [4] in their work only use a simple AR(1) model (number of time lags is 1). We therefore expand their effort by considering a more generalized family of AR models: autoregressive–moving-average (ARMA) models where past error term values are integrated.

$$x_t = \sum_{i=1}^{p}\phi_i x_{t-i} + \sum_{i=1}^{q}\theta_i \varepsilon_{t-i} + \varepsilon_t + C$$

where x_t and ε_t are the time-series and error term at time step t, respectively. ϕ_i and θ_i are parameters, and C is a constant. Orders $p = 10$ and $q = 3$ for both AR and MA components are determined using cross-validation. Results are available in Table A1.

Recurrent-Neural-Network-based Models

We narrow our focus on recurrent-neural-network (RNN) architectures because they are designed to deal with sequence problems. Time-series prediction is an appropriate use of such architectures. We experiment three different architectures, with increasing model capacity.

- single layer long short-term memory model (LSTM);
- two-layer long short-term memory model with attention mechanism (A-LSTM);
- dual-stage attention-based recurrent-neural-network (DA-RNN) proposed by Qin et al. [11].

With LSTM, the dimension of hidden unit is set at 64; with A-LSTM the dimension of hidden unit at each layer is 16 and 32, respectively; with DA-RNN, both encoder and decoder have 128 dimensional hidden unit (as specified in the publication). In all models, first layer standardizes inputs; subsequent layers are followed by batch normalization; and the last layer is a fully connected layer that also applies inverse transformation.

We employ walk-forward cross-validation scheme to find optimal look-back size and learning rate. Specifically, we split the entire training set into k folds, with each fold consisting of a portion of training sub-section followed by a development sub-section. The first fold starts from the beginning, and then we "roll" each fold forward, adding new examples to the previous fold. This approach ensures temporal order of training sub-section and development sub-section. We train the model for 100 epochs for each fold, and evaluates on the development data. Cross-validation error for a specific model is aggregated by taking the average of k MSE errors. We perform a grid search on look-back size ranging from 1 to 10, and random learning rate values sampled between 1e-3 and 1e-5 (on a logarithm scale). Both cross-validation splits of the full data set and hyper-parameters grid search results are presented in Appendices section.

We use MSE as loss function because we strive to minimize the difference between predicted return and true return. In training time, we pre-process original input and obtain windowed input. That is, we pair up examples from l previous steps with the current step CDS spread log return (l is the look-back window size). We then shuffle the pairs and pick B pairs to form a training batch (B is the batch size). An entire epoch is concluded when all pairs have been fed into the model. Adam is the choice of optimizer. As we are exploring different learning rates in cross validation, we keep $\beta_1 = 0.9$, $\beta_2 = 0.999$, and decay rate $= 0$ fixed. In testing time, we also first generate windowed inputs, but they go through the trained model sequentially. In all cases, we keep batch size to be $B = 64$.

RESULTS & DISCUSSION

The following table summarizes the performance of benchmark models on test set. The first value in each pair is measured directly on log returns, whereas the second value is measured on recovered index price based predicted return and true previous price.

model / *metric*	MSE	MAE	MAPE
single-feature SVR	<u>0.00058583</u> / 7.84729490	0.01812129 / 2.05785338	14.86450801 / 0.01843948
multi-feature SVR	0.00060360 / 8.28526058	0.01846152 / 2.10569444	15.14359846 / 0.01886816
ARMA	0.00058982 / 7.89520958	0.01818067 / 2.06378594	14.91321619 / 0.01849264

Table 1: performance of baseline models

All of the benchmark models have comparable performance, and single-feature SVR model offers a thin edge over the remaining benchmark models. This comes as a bit of a surprise considering the model only uses time series at a single point to make a prediction. It suggests additional features and more previous steps don't translate into more accurate prediction in SVR and AR models.

Upon further inspection (plots presented in Appendices section), we observe that both single-feature SVR and ARMA don't attune to actual fluctuations and trends, as they consistently output a value within a small region around the average return. This is especially pronounced in ARMA – the predicted sequence is essentially a flat line until toward the very end. Multi-feature SVR, on the other hand, provides more realistic predictions through out the testing period. The model seems to be able to make more varied predictions and track the underlying trend; but it reports the lowest MSE value because it initially understated return values, and failed to output more extreme values when a big swing is present.

We initially ran both LSTM and A-LSTM for $10,000$ epochs, and the results indicate the models have overfit the training set (training and testing errors presented in Table A4). Even with a model as simple as LSTM with a single layer of LSTM cell, the model is already quite expressive, and has the potential to fit the training data even better if the training went longer. To mitigate the overfitting issue, we increase the drop-out rate from 0.1 tp 0.3, and use cross-validation to determine the optimal early stopping position for both models (cross-validation plots presented in Appendices section). Results are summarized in the following table.

metric / *model*	MSE	MAE	MAPE
	epochs = 10,000 *drop-out = 0.1*		
LSTM	0.00085408 / 11.71654362	0.02229393 / 2.53554940	17.85809587 / 0.02275542
A-LSTM	0.00086257 / 12.23533121	0.02175617 / 2.49177435	17.42733023 / 0.02236256
	drop-out = 0.3		
LSTM *epochs = 160*	0.00063501 / 8.72387561	0.01884446 / 2.14714567	15.09496577 / 0.01926967
A-LSTM *epochs = 300*	<u>0.00059720</u> / 8.07059335	0.01827941 / 2.07873858	14.64234425 / 0.01865575

Table 2: performance of LSTM and A-LSTM

We see with shorter training and more aggressive drop-out rate, testing errors are generally on par with training errors. From plotted predicted values, variability of the predictions has gone down dramatically; however, overfitted models are clearly better at predicting extreme values. Between two models, marked regions suggest A-LSTM is even more capable of making abrupt changes when the market is indeed volatile.

metric / *model*	MSE	MAE	MAPE
DA-RNN *T = 5*	0.00081609 / 11.81883050	0.02101380 / 2.41791116	17.23718096 / 0.02166580
DA-RNN *T = 10*	<u>0.00068488</u> / 9.30567002	0.01998492 / 2.28297271	17.50756201 / 0.02040664
DA-RNN *T = 15*	0.00074998 / 10.28764195	0.02060551 / 2.34955174	15.59782370 / 0.02094186

Table 3: performance of DA-RNN

As for DA-RNN, we experiment with varying sizes of look-back window. Although all three models also suffer from overfitting, they seems to have learned when to output extreme predictions and when to stay tranquil. Compared with both LSTM and A-LSTM models, DA-RNN models appear to be more adaptive in the sense that these models tend to jiggle less during calm periods and react aptly during volatile periods.

CONCLUSION & FUTURE WORK

RNN-based models generally have comparable or slightly worse performance than benchmark models. Predictions from RNN-based models show drastically increased predicting capacity when compared to benchmark models; but it is apparent that RNN-based models don't generalize well on test data. As flexible and expressive as they are, the presence of high-variance situation suggests more data or/and less noisy input is necessary in order to further improve performance.

There are a few interesting aspects that this topic can be further studied.

- Improved data quality by using de-noise algorithms. Financial data tend to be noisy. Qiu et al. [12] apply wavelet transformation to pre-process stock prices before dispatching it to the attention based RNN architecture. The same technique should be applicable to CDS spread data as well.
- More feature-rich input data:

  - additional hand-engineered features include: **default probability** is directly related to CDS spread, and can be estimated using the Merton model [13]; **CDS spread term structure** could also be a relevant feature, and can be parameterized using Nelson–Siegel–Svensson model discussed earlier.

  - implementation of feature-extraction architectures: grafting another convolutional-neural-network architecture to obtain better feature representations is shown to have a better stock market prediction precision (Chen and Zhang et al. [14], Yang et al. [15], Chen and Lin et al. [16]).
- A systematic way to acquire more historical data. We expect these RNN architectures could achieve better results given more training data. Getting financial data, however, is not always straightforward. It may be worth to try to synthesize data in order to adequately train these RNN-based models.
- Incorporating level shift detection using deep learning architectures. Classical econometric models can be more effectively utilized if abrupt level shift can be identified.

Thanks Avoy for the much-needed encouragement and support in times of difficulties :)

REFERENCES

[1] Y. Gündüz and M. Uhrig-Homburg, “Predicting credit default swap prices with financial and pure data-driven approaches,” *Quantitative Finance*, vol. 11, no. 12, pp. 1709–1727, 2011.

[2] M. Mercadier and J.-P. Lardy, “Credit spread approximation and improvement using random forest regression,” *European Journal of Operational Research*, vol. 277, no. 1, pp. 351–365, 2019.

[3] F. Shaw, F. Murphy, and F. O’Brien, “The forecasting efficiency of the dynamic Nelson Siegel model on credit default swaps,” *Research in International Business and Finance*, vol. 30, pp. 348–368, 2014.

[4] D. Avino and O. Nneji, “Are CDS spreads predictable? An analysis of linear and non-linear forecasting models,” *International Review of Financial Analysis*, vol. 34, pp. 262–274, 2014.

[5] K. Galil, O. M. Shapir, D. Amiram, and U. Ben-Zion, “The determinants of CDS spreads,” *Journal of Banking & Finance*, vol. 41, pp. 271–282, 2014.

[6] J. Pereira, G. Sorwar, and M. Nurullah, “What drives corporate CDS spreads? A comparison across US, UK and EU firms,” *Journal of International Financial Markets, Institutions & Money*, vol. 56, pp. 188–200, 2018.

[7] H. Doshi, J. Ericsson, K. Jacobs, and S. M. Turnbull, “Pricing credit default wwaps with observable covariates,” *The Review of Financial Studies*, vol. 26, no. 8, pp. 2049–2094, 2013.

[8] L. E. Svensson, “Estimating and interpreting forward interest rates: Sweden 1992 - 1994,” working paper, National Bureau of Economic Research, 1994.

[9] C. R. Nelson and A. F. Siegel, “Parsimonious modeling of yield curves,” *The Journal of Business*, vol. 60, no. 4, pp. 473–489, 1987.

[10] H. Byström, “CreditGrades and the iTraxx CDS Index market,” *Financial Analysts Journal*, vol. 62, no. 6, pp. 65–76, 2006.

[11] Y. Qin, D. Song, H. Chen, W. Cheng, G. Jiang, and G. Cottrell, “A dual-stage attention-based recurrent neural network for time series prediction,” International Joint Conference on Artificial Intelligence (IJCAI), 2017.

[12] J. Qiu, B. Wang, and C. Zhou, “Forecasting stock prices with long-short term memory neural network based on attention mechanism,” *PLOS ONE*, vol. 15, no. 1, pp. 1–15, 2020.

[13] M. Tudela and G. Young, “A Merton-model approach to assessing the default risk of UK public companies,” working paper, Bank of England, 2003.

[14] C. Chen, P. Zhang, Y. Liu, and J. Liu, “Financial quantitative investment using convolutional neural network and deep learning technology,” *Neurocomputing*, vol. 390, no. 1, pp. 384–390, 2020.

[15] C. Yang, J. Zhai, and G. Tao, “Deep learning for price movement prediction using convolutional neural network and long short-term memory,” *Mathematical Problems in Engineering*, 2020.

[16] Y. Chen, W. Lin, and J. Z. Wang, “A dual-attention-based stock price trend prediction model with dual features,” *IEEE Access*, vol. 7, pp. 148047–148058, 2019.

APPENDICES

Metrics

- Mean Squared Error (MSE): $\frac{1}{m}\sum_{i=1}^{m}\left(y^{(i)}-\hat{y}^{(i)}\right)^2$
- Mean Absolute Error (MAE): $\frac{1}{m}\sum_{i=1}^{m}\left|y^{(i)}-\hat{y}^{(i)}\right|$
- Mean Absolute Percentage Error (MAPE): $\frac{(1/m)\sum_{i=1}^{m}\left|y^{(i)}-\hat{y}^{(i)}\right|}{(1/m)\left|\sum_{i=1}^{m}y^{(i)}\right|}$

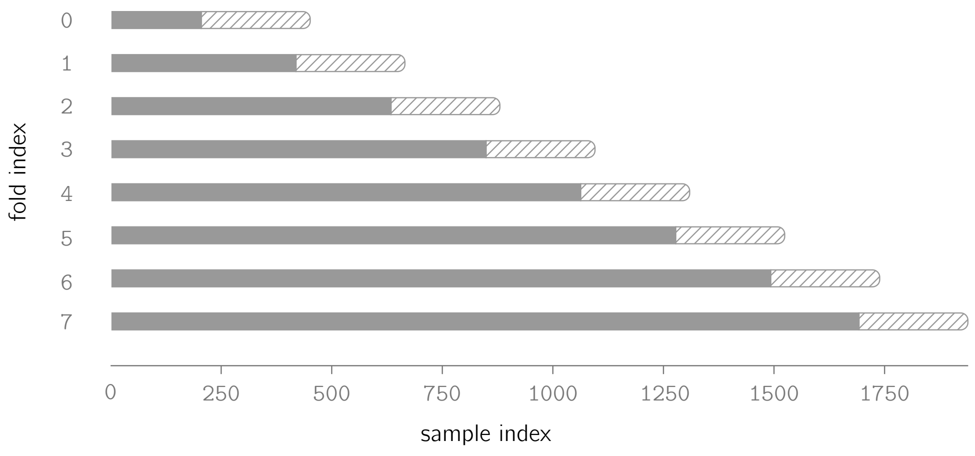
ARMA Cross-Validation Errors

Average MSE reported on k development sets using walk-forward cross-validation scheme.

	$p = 1$	$p = 2$	$p = 3$	$p = 4$	$p = 5$	$p = 6$	$p = 7$	$p = 8$	$p = 9$	$p = \mathbf{10}$
$q = 0$	0.00050862	0.00050837	0.00050839	0.00050827	0.00050814	0.0005082	0.00050776	0.00050796	0.00050793	0.00050736
$q = 1$	0.0005086	0.00050839	0.00050852	0.00050809	0.00050816	0.00050819	0.00050733	0.00050844	0.00050784	0.00050735
$q = 2$	0.00050831	0.00050878	0.00050873	0.00050727	0.00050737	0.00050774	0.00050735	0.00050759	0.00050752	0.00050773
$q = 3$	0.00050753	0.00050783	0.00050808	0.00050701	0.0005073	0.00050754	0.00050745	0.00050805	0.00050726	**0.00050654**
$q = 4$	0.00050744	0.00050776	0.00050831	0.00050777	0.00050731	0.00050761	0.00050684	0.00050712	0.00050796	0.00050786
$q = 5$	0.00051016	0.00050766	0.00050766	0.00050755	0.00050793	0.00050843	0.00050766	0.00050738	0.00050778	0.00050849

Table A1: cross-validation error for autoregressive models

Cross-Validation Splits



LSTM Cross-Validation Errors

Average MSE reported on k development sets using walk-forward cross-validation scheme.

learning rate *look-back size*	0.000011	0.000013	0.000027	0.000061	0.000781
1	0.00059165	0.00060355	0.00056671	0.0006032	0.00054593
2	0.00055475	0.00069688	0.00053107	0.00060834	0.00053722
3	0.00060508	0.00055021	0.00060129	0.00056344	0.00055699
4	0.00106016	0.0006011	0.00066083	0.00055371	0.00054687
5	0.00056635	0.00055871	0.00060154	0.0005622	0.00054527
6	0.00057914	0.00065929	0.00073333	0.00056988	0.00054511
7	0.00062257	0.00072196	0.00058697	0.0005468	0.00053886
8	0.00055407	0.00060648	0.00056296	0.00055276	0.00055247
9	0.0006091	0.00075999	0.00055355	0.00053829	0.00054544
10	0.00056587	0.00064079	0.00059	0.00061336	0.00053738

Table A2: cross-validation error for LTSM

A-LSTM Cross-Validation Errors

Average MSE reported on k development sets using walk-forward cross-validation scheme.

learning rate *look-back size*	0.000010	0.000015	0.000019	0.000023	0.000026
1	0.00057696	0.00058125	0.00060381	0.00063551	0.00060676
2	0.0008208	0.00058056	0.00058939	0.00056776	0.00057868
3	0.00082091	0.00080905	0.00062418	0.00058965	0.00061544
4	0.00060402	0.00063641	0.00063859	0.00060534	0.00063807
5	0.00071563	0.00060468	0.00065161	0.00062438	0.00058957
6	0.00067399	0.0006129	0.00060147	0.00058851	0.00057928
7	0.00067239	0.00080909	0.0006233	0.00065332	0.00061012
8	0.00064888	0.00060962	0.00063191	0.00060425	0.00059756
9	0.00062033	0.00066734	0.00078467	0.00061823	0.00061248
10	0.00070796	0.00085071	0.00070938	0.00071565	0.00062601

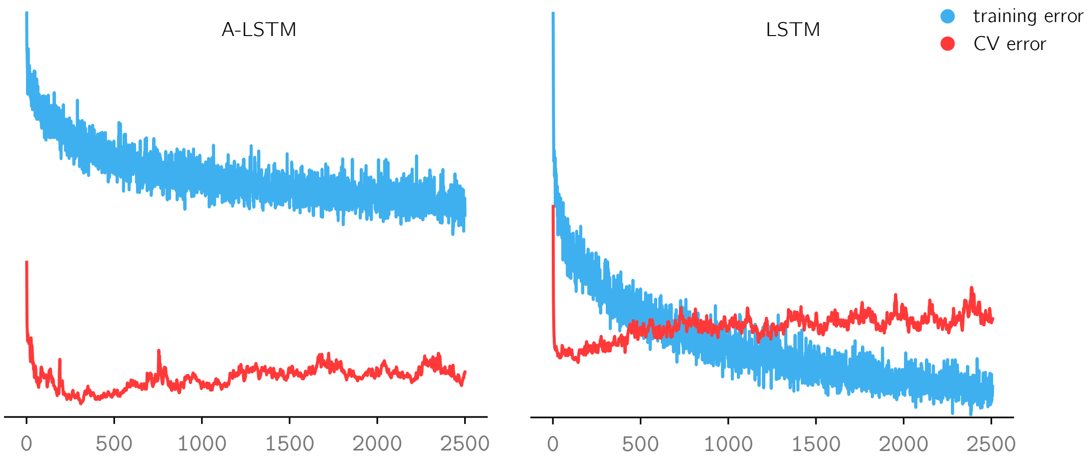
Table A3: cross-validation error for A-LTSM

LSTM and A-LSTM Errors (MSE)

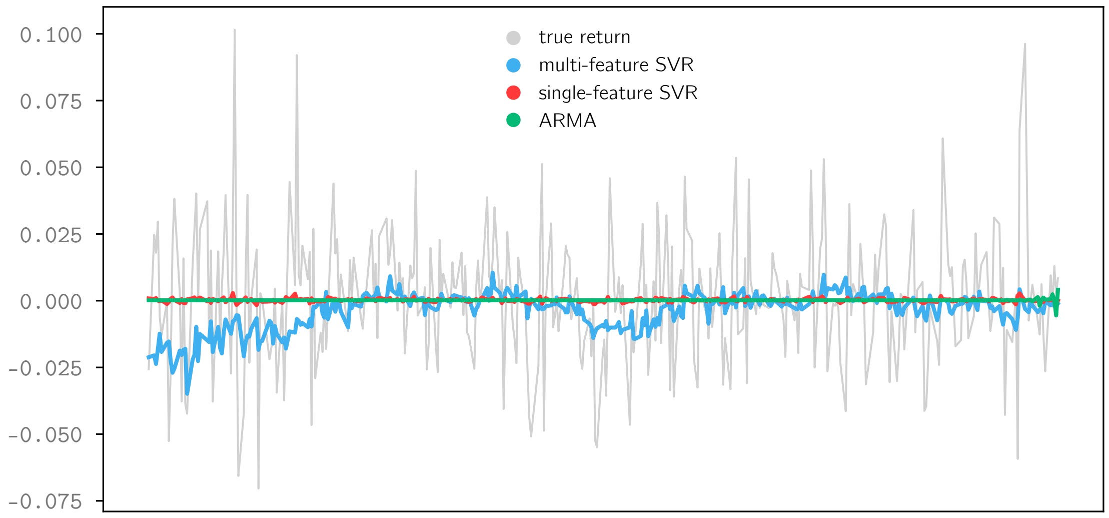
errors *model*	training error	test error
	epochs = 10,000 drop-out = 0.1	
LSTM	0.00011466 / 1.31766348	0.00085408 / 11.71654362
A-LSTM	0.00026994 / 2.17702236	0.00086257 / 12.23533122
	drop-out = 0.3	
LSTM *epochs = 160*	0.00061787 / 3.96046418	0.00063501 / 8.72387561
A-LSTM *epochs = 300*	0.00063171 / 3.94699284	0.00059720 / 8.07059335

Table A4: training and test errors for LTSM and A-LTSM

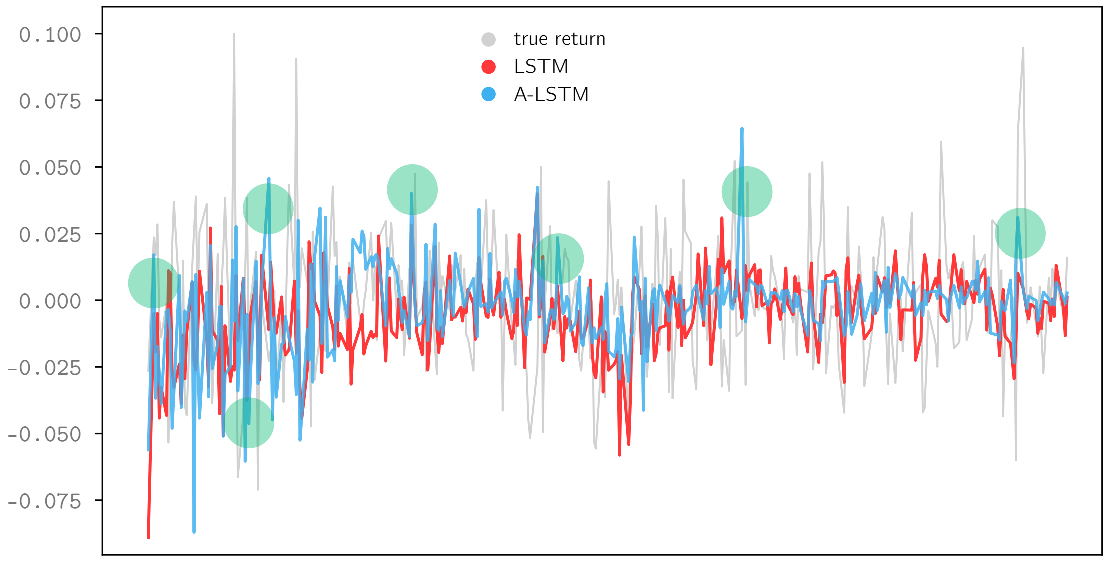
Errors vs. Epochs



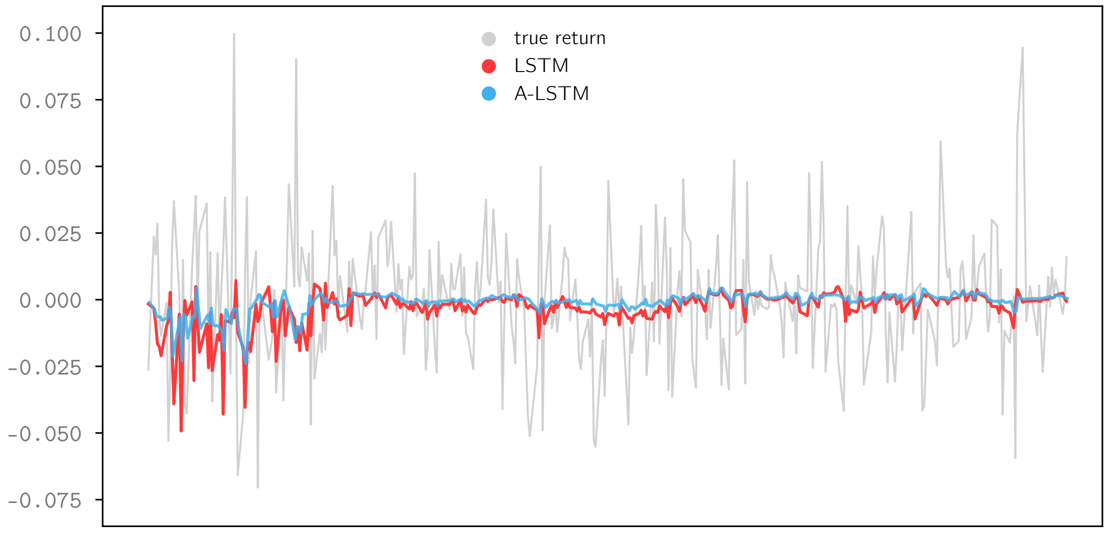
Predictions of Benchmark Models



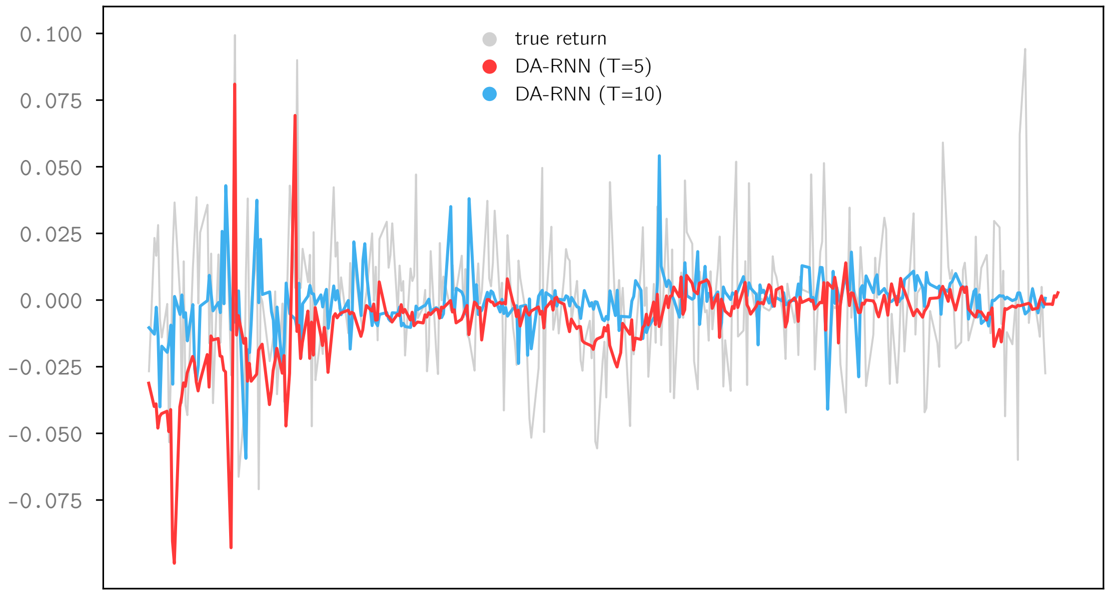
Predictions of LSTM and A-LSTM (long training)



Predictions of LSTM and A-LSTM (short training)



Predictions of DA-RNN


● true return
● DA-RNN (T=5)
● DA-RNN (T=10)