Predicting Stock Market Movement with Deep RNNs

Jason Poulos[†]

Abstract

Predicting the movement of stock market indices is a commonly studied time-series learning problem. This study applies deep recurrent neural networks (RNNs) on textual representations of daily news headlines to predict stock market movement. While previous studies rely on non-recurrent and shallow algorithms, deep RNNs can exploit the sequential nature of time-series data by sharing the same weights across multiple timesteps and can automatically relevant features from the feature set. I implement a 12-layer GRU on a benchmark dataset for predicting stock market movement only using news headlines as features and achieve 54% test set accuracy. The model is under half a percentage point shy of achieving state-of-the-art status on the benchmark dataset when area under the ROC curve (AUC) is the evaluation metric.

1. Introduction

Stock market prediction is an important time-series learning problem in financial economics. According to the weak form of the dominant efficient-market hypothesis (EMH), stock prices already reflect all past publicly available information, which means that researchers should not be able to predict stock market movement better than chance. Several studies contradict the EMH, including Schumaker and Hsinchun (2009) [16], who obtain 57.1% accuracy in predicting the direction of the future price index of S&P 500 stocks using Support Vector Machines (SVMs) trained on a feature vector containing both financial article terms and stock prices at the time of article release.[1] Better solutions to the problem would shed light on the types of information that are relevant to predicting stock market movement.

The present study applies deep recurrent neural networks (RNNs) on textual representations of daily news headlines to predict stock market movement. RNNs are a class of neural networks that take advantage of the sequential nature of time–series data by sharing the same weights across multiple timesteps [8, 10]. When folded out in time, RNNs are considered "deep" with indefinitely many layers [13]. Each layer can be interpreted as an RNN that receives the time series of the previous layer as input (Figure 1). Unlike RNNs or other learning algorithms, sufficiently deep RNNs can automatically extract relevant features from the feature set [10, 15]. Deep RNNs have achieved state-of-the-art results in missing value imputation [2, 14], character-level language modeling [13], and offline handwriting recognition [12, 11].

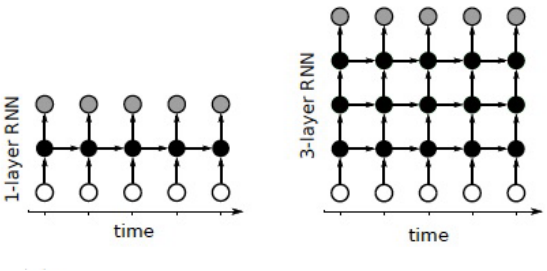


Figure 1: Architecture of (Left) standard RNN and (Right) 3-layered RNN (from Hermans and Schrauwen (2013) [13]). Arrows are connection matrices; white circles are inputs; black circles are hidden layers; and grey circles are outputs.

The focus of the present study is not to experiment using different NLP methods, but rather to determine whether deep RNNs outperform non-recurrent and shallow algorithms. I implement deep RNNs on a

[†]poulos@berkeley.edu

[1]The authors experiment on different Natural Language Processing (NLP) techniques to extract textual representations of financial news headlines.

benchmark dataset for predicting stock market movement, described in Section 2. Section 3 describes the architecture of the baseline and final models used for training and evaluation. The final model outperforms the baseline model and several non-recurrent algorithms in terms of test set accuracy measured by the area under the ROC curve (AUC), but does not do as well as an SVMs classifier (Section 4). Finally, I summarize the results and reflect on lessons learned in Section 5.

2. Data and preprocessing

I use the Daily News for Stock Market Prediction dataset from Kaggle to benchmark deep RNNs against competing models.[2] The dataset is a daily time-series containing 1,989 examples and 25 columns, each containing a news headline from Reddit WorldNews Channel (/r/worldnews). The headlines are ordinally ranked by Reddit users' votes, and only the top 25 headlines are considered for a single date. Each example is labeled "1" when the Dow Jones Industrial Average (DJIA) Adjusted Close value increased or was the same and "0" when the value decreased. The dataset covers about eight years of newspaper headlines and financial data, from 2008-08-08 to 2016-07-01.

The learning task specified by the dataset donor is to predict whether the DJIA closed higher or lower given the headlines. More formally, we want to output a prediction of $\mathbf{y}^{(t)}$ given a sequence that contains information from the past, $\mathbf{x}^{(1)}, \ldots, \mathbf{x}^{(t-1)}$, and the present input $\mathbf{x}^{(t)}$.

I preprocess the data as follows. First, implement the roughly 80% to 20% train-test split specified by the dataset donor. In order to preserve the sequential nature of the data, the split is not random; rather, the first six years of data are used as the training set and the last two years is reserved for the test set.[3] Second, I concatenate all 25 headlines of a day into a single string. Third, I use the short for term frequency-inverse document frequency (tf-idf) weighting scheme to extract a feature vector.[4] The feature extraction select tokens of two or more alphanumeric characters, ignoring punctuation and stopwords common to the NLTK data package [4]. I use the training set fit of the tf-idf feature extractor on the test set, to ensure the test data are unseen before test time.

3. Baseline and final models

The baseline model, visualized in Figure 2, stacks three Gated Recurrent Unit (GRU) layers [6]. The first two GRU layers return their full output sequences, while the third GRU layer drops the temporal dimension and returns the last step in the output sequence. Dropout is applied to the hidden units of each GRU layer. The model includes a masking layer to handle varying input lengths.

The final model shares the same architecture as the baseline model, except it has 12 stacked GRU layers instead of 3 (Figure 6 in the Appendix). The number of hidden layers is treated as a hyperparameter selected on the basis of validation set performance, described in Section 4.1. Like the baseline model, the final model optimizes the logloss objective function with RMSProp, an adaptive stochastic gradient method known to work best for recurrent models [18]. Both models are rendered "stateful", which means that the states for the samples of each batch are reused as initial states for samples in the next batch, allowing the model to process longer sequences.[5]

I implement both models using the `Keras` neural networks library [5], which is written in Python and runs on top of Theano [3], and allows for adding many recurrent GRU layers.[6]

4. Training and evaluation

In this section, I describe the process of choosing optimal hyperparameters for the baseline and final models, present model evaluation metrics, and benchmark the models against competing models.

4.1. Hyperparameter selection

When choosing hyperparameters during training, I reserve the last 20% of the training set for validation and select the best model in terms of classification accuracy on the validation set. For the baseline model, I experiment with the following hyperparameters during training:

- fraction of the input units to drop at each update during training time, {0.25, 0.5, 0.75};

[2] `https://www.kaggle.com/aaron7sun/stocknews`.

[3] Specifically, data from 2008-08-08 to 2014-12-31 forms the training set and data from 2015-01-02 to 2016-07-01 is the test set.

[4] tf-idf is a numerical value that captures how important a word is to a single days' headlines. The value increases proportionally to the word frequency in the single days' headlines, but is offset by the frequency of the word in the entire collection of days' headlines.

[5] In stateful models, all batches need to have the same number of samples. I use a batch size of 14 because it divides evenly into the training and test sets.

[6] I train the final model on an Amazon Elastic Compute Cloud (Amazon EC2) instance based on a custom Intel 2.4 GHz Xeon E5-2676 v3 Haswell processor with AVX2 (8 GiB of memory, 2 vCPU, EBS-only, 64-bit platform).

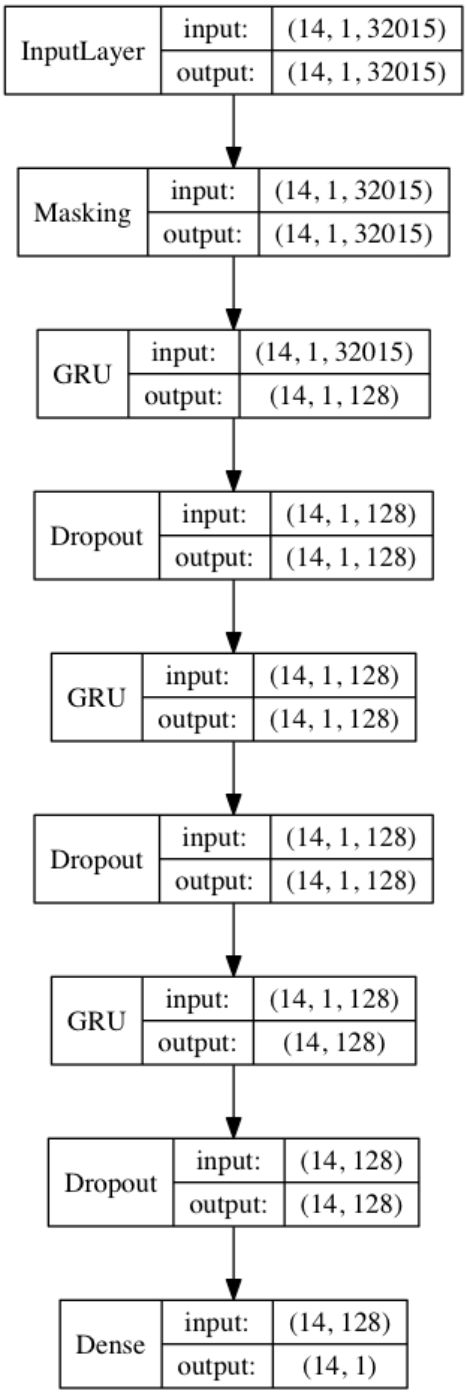


Figure 2: Architecture of baseline 3-layer GRU. The first GRU layer takes the input (batch size, # timesteps, # features) and the last two GRU layers take the input (batch size, # timesteps, # hidden nodes). The first two GRU layers return their full output sequences (batch size, # timesteps, # hidden units) and the last GRU layer converts the input sequence into a single vector (batch size, # hidden units).

- number of hidden nodes in each GRU layer, {32, 64,128};
- activation function for the fully-connected layer, {softmax, tanh, sigmoid}; and
- weight initialization function, {uniform, Gaussian, Xavier}.

The validated baseline model has for each layer 128 hidden units with weights initialized with Xavier initialization [9], a dropout of 0.25 applied to the hidden units of each layer, and a sigmoid activation function for the fully-connected layer. The model achieves a validation set accuracy of 63.5%.

I carry over the same hyperparameters used in the baseline model to the final model, and add GRU layers until validation set accuracy stops improving. The final model with 12 GRU layers achieves a validation set accuracy of about 65%. Performing a manual search with a strategy of coordinate descent [1], I try all of the following hyperparameter changes, none of which successfully increase validation accuracy:

- using an advanced activation function, i.e., exponential linear units (ELUs) [7];
- increasing the number of hidden units;
- increasing the dropout rate; and
- experimenting with adaptive learning rates.

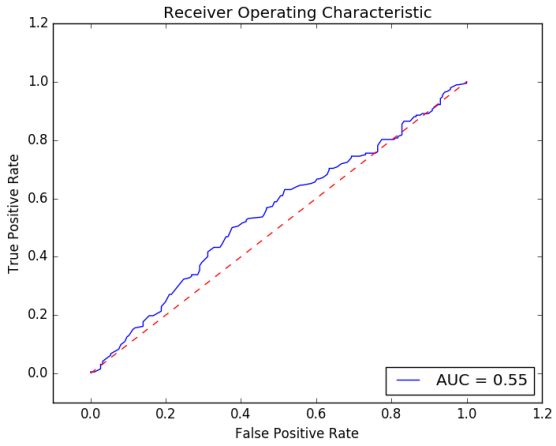
Incorporating ELUs into the model did not affect the speed of learning or increase accuracy, as expected. Increasing the model complexity led to underfitting and slower training times. Higher dropout rates did not appear to help improve accuracy. Lastly, experimenting with different drop-based learning rate schedules and decaying the learning rate over each training epoch often led to worse classification accuracies than keeping constant the initial learning rate of 10e-3.[7]

4.2. Evaluation

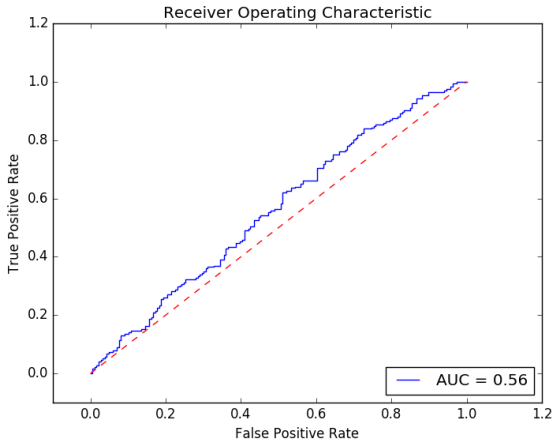
I fit the best baseline model to the full training set and measure test set performance in terms of both classification accuracy and AUC, the latter metric suggested by the dataset donor. The baseline model, trained with 25 epochs, achieves 52.3% test set accuracy and an AUC of 55%. The ROC curve for the baseline model is plotted in Figure 3 (a).

I allow the final model to train longer (160 epochs) and check-point the model so that model weights are

[7]This initial learning rate and other RMSprop parameters is recommended by Tieleman and Hinton (2012) [18].

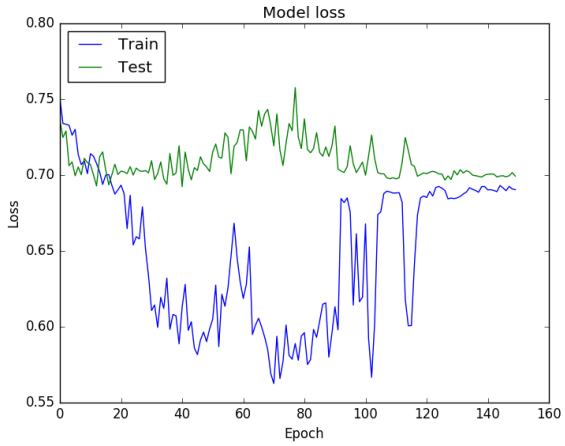


(a) Baseline model

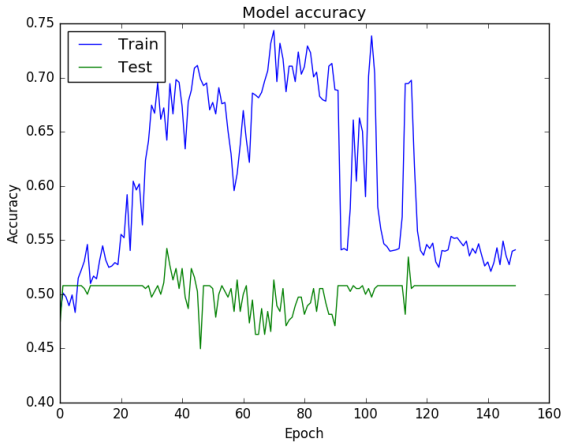


(b) Final model

Figure 3: ROC/AUC for (a) baseline and (b) final models.



(a) Loss



(b) Accuracy

Figure 4: Final model performance in terms of (a) loss and (b) accuracy.

saved when there is an improvement in classification accuracy on the test set. Figures 4 (a) and 4 (b) show the evolution of training the model over time, in terms of loss and accuracy, respectively. The model begins to massively overfit after the 20^{th} epoch and stabilizes after the 100^{th} epoch. I recover the model weights just before 40^{th} epoch, when test set accuracy is the highest (Figure 4 (b)). The best final model modestly outperforms the baseline with a 54.2% accuracy and 55.9% AUC, the latter metric represented in Figure 3 (b).

Figure 5 benchmarks the performance of final and baseline models against competing models published by Kaggle users.[8] Not all models are directly comparable because different NLP methods are used to train the models. The state-of-the-art Radial Basis Function (RBF) kernel SVMs classifier trained with default settings (56.3% AUC), however, uses virtually similar preprocessing methods as those in the present study.[9]

5. Discussion

The 12-layer GRU predicts the movement of the DJIA on the basis of textual representations of important news headlines with 54% accuracy. This result contradicts the weak form of the EMH, which postulates that researchers should not be able to predict

[8] `https://www.kaggle.com/aaron7sun/stocknews/kernels`.

[9] The present study removes stopwords and splits before extracting a feature vector, using the tf-idf training fit on the test set. The preprocessing used by the state-of-the-art model does not remove stopwords and splits after fitting the tf-idf, which compromises the validation.

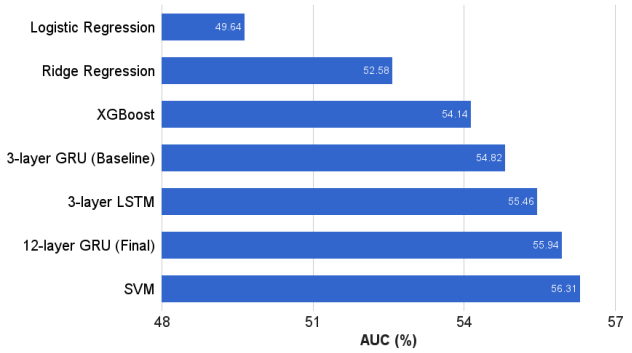


Figure 5: Comparison of published models' test performance on Kaggle data.

stock market movement any better than chance. The 12-layer GRU bests the baseline model, a 3-layer GRU, by 2% in terms of classification accuracy and about 1% in terms of AUC. In terms of AUC, the 12-layer GRU is under half a percentage point shy of achieving state-of-the-art status, which is claimed by a simple untuned RBF kernel SVMs using nearly identical preprocessing methods. SVMs have achieved very good results in stock market movement prediction problems, e.g., Schumaker and Hsinchun (2009) [16], and are likely better-suited for classification problems on numerical representations of text data. However, SVMs do not exploit the sequential nature of the data and cannot learn its own features, while deep RNNs have these capabilities.

These results come with several surprises and lessons learned. First, feature extraction is more important in terms of increasing model performance than model selection or hyperparameter selection for this particular learning task. The goal of the present study is to evaluate the performance of deep RNNs for time-series prediction — not to experiment with different NLP methods. Still, using more advanced NLP models for preprocessing likely matters more for improving performance than incorporating neural network bells-and-whistles. Second, the risk of overfitting the 12-layer GRU was very high, due to the fact that the training set size is small relative to the model complexity. Higher dropout rates did not appear to improve validation set accuracy, which also may due to the size of training set [17]. Third, while adding more hidden layers improved validation accuracy, increasing the number of hidden units in the 12-layer GRU slowed down training and led to underfitting. Finally, the built-in advantages of deep RNNs do not appear to have significantly improved the results. It is possible that stock market movement is more of a stochastic process and does not rely on prior histories, or simply that there are not enough examples to learn from.

Future work might experiment with larger datasets to determine whether deep RNNs can better exploit the sequential nature of data with more examples. Moreover, the problem of overfitting may be handled more effectively with dropout when the training set size is larger. It would also be useful to compare deep RNNs with non-recurrent deep neural networks, which may shed light on whether exploiting the time-series is important to the learning task.

5.1. Team Contributions

Jason Poulos is responsible for all aspects of this project. The code and data used for this project is stored in the Github repo drnns-prediction (`https://github.com/jvpoulos/drnns-prediction`).

References

[1] Y. Bengio. Practical recommendations for gradient-based training of deep architectures. In *Neural Networks: Tricks of the Trade*, pages 437–478. Springer, 2012.

[2] Y. Bengio and F. Gingras. Recurrent neural networks for missing or asynchronous data. *Advances in neural information processing systems*, pages 395–401, 1996.

[3] J. Bergstra, O. Breuleux, F. Bastien, P. Lamblin, R. Pascanu, G. Desjardins, J. Turian, D. Warde-Farley, and Y. Bengio. Theano: A cpu and gpu math compiler in python. In *Proc. 9th Python in Science Conf*, pages 1–7, 2010.

[4] S. Bird, E. Klein, and E. Loper. *Natural language processing with Python.* " O'Reilly Media, Inc.", 2009.

[5] F. Chollet. Keras. `https://github.com/fchollet/keras`, 2015.

[6] J. Chung, C. Gulcehre, K. Cho, and Y. Bengio. Empirical evaluation of gated recurrent neural networks on sequence modeling. *arXiv preprint arXiv:1412.3555*, 2014.

[7] D.-A. Clevert, T. Unterthiner, and S. Hochreiter. Fast and accurate deep network learning by exponential linear units (elus). *arXiv preprint arXiv:1511.07289*, 2015.

[8] S. El Hihi and Y. Bengio. Hierarchical recurrent neural networks for long-term dependencies. In *NIPS*, volume 400, page 409. Citeseer, 1995.

[9] X. Glorot and Y. Bengio. Understanding the difficulty of training deep feedforward neural networks. In *Aistats*, volume 9, pages 249–256, 2010.

[10] I. Goodfellow, Y. Bengio, and A. Courville. Deep learning. Book in preparation for MIT Press, 2016.

[11] A. Graves. Neural networks. In *Supervised Sequence Labelling with Recurrent Neural Networks*, pages 15–35. Springer, 2012.

[12] A. Graves and J. Schmidhuber. Offline handwriting recognition with multidimensional recurrent neural networks. In *Advances in neural information processing systems*, pages 545–552, 2009.

[13] M. Hermans and B. Schrauwen. Training and analysing deep recurrent neural networks. In *Advances in Neural Information Processing Systems*, pages 190–198, 2013.

[14] S. Parveen and P. Green. Speech recognition with missing data using recurrent neural nets. In *Advances in Neural Information Processing Systems*, pages 1189–1195, 2001.

[15] S. C. Prasad and P. Prasad. Deep recurrent neural networks for time series prediction. *arXiv preprint arXiv:1407.5949*, 2014.

[16] R. P. Schumaker and H. Chen. Textual analysis of stock market prediction using breaking financial news: The azfin text system. *ACM Transactions on Information Systems (TOIS)*, 27(2):12, 2009.

[17] N. Srivastava, G. E. Hinton, A. Krizhevsky, I. Sutskever, and R. Salakhutdinov. Dropout: a simple way to prevent neural networks from overfitting. *Journal of Machine Learning Research*, 15(1):1929–1958, 2014.

[18] T. Tieleman and G. Hinton. Lecture 6.5-rmsprop: Divide the gradient by a running average of its recent magnitude. *COURSERA: Neural Networks for Machine Learning*, 4(2), 2012.

Appendices

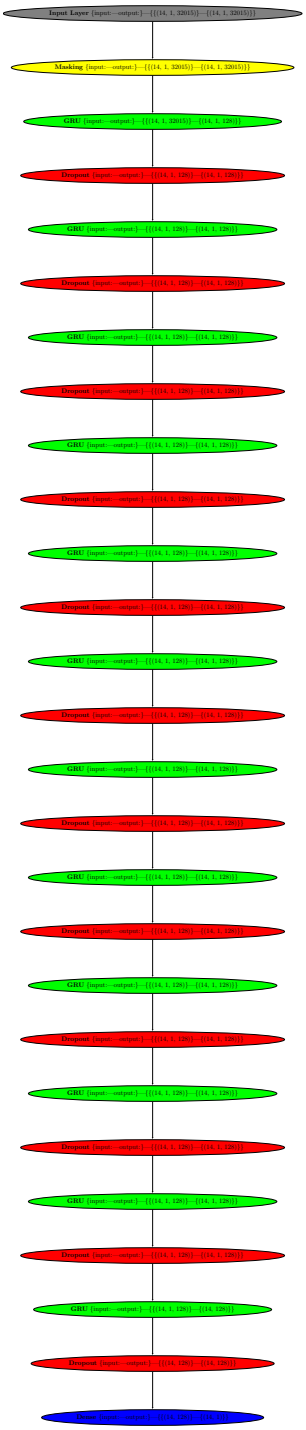


Figure 6: Architecture of baseline 12-layer GRU, consisting of dropout applied to each GRU layer; a Masking layer that handles varying input lengths; an initial layer and a fully-connected layer.