LSTM and CNN-LSTM Neural Networks for FOREX Price Forecasting

Peter Gillich
University of Waterloo
pmgillic@uwaterloo.ca

Abstract

We investigated the applicability of two neural network architectures to the prediction of FOREX rates for the CAD/USD currency pair: an LSTM network, and a hybrid CNN-LSTM network, to determine what benefit feature extraction via convolutional layers might have on the predictive power of an LSTM model. We trained two versions of each architecture on input histories of 30 and 60 days of daily closing exchange rates, and found that there was no clear better. We also attempted to use these models to make medium term price trend predictions, with mixed results.

1 Introduction

Trying to predict financial markets is a problem as old as the markets themselves. Most analysts today follow two schools of thought–on the one hand, some make predictions based on 'fundamentals'–these are indicators of the overall health of a company, and are properties of the company itself. On the other hand, some analysts look to make predictions based on 'technical analysis'–which seeks to make predictions based on various statistics, such as open/close price of an asset, moving averages, and a host of others. Here, the analyst is less interested in the fundamentals of a company, and more interested in what the raw technical price data say about an asset. So-called 'chartists', of the latter camp, look for patterns in historical price charts to extrapolate trends in the future price movements of an asset.

The motivations for applying the techniques of deep leaning to the technical analysis of financial time series are obvious, given the power of neural networks for learning patterns from data. In this paper, we attempt to leverage neural networks to make short-term predictions for foreign exchange (FOREX) rates, using only technical indicators to do so.

In particular, we examine the performance of two types of neural network applied to this problem: the LSTM (Long Short Term Memory) neural network, which has been used to good effect for predicting sequence data, particularly in Natural Language Processing, as well as the CNN (Convolutional Neural Network), which has transformed the landscape of image recognition through their ability to learn features from raw data, where the manual specification of such features is difficult or impossible.

In this paper, we attempt to apply a hybrid CNN-LSTM neural network to one-dimensional time series data, in the hopes that the feature extraction capabilities of the convolutional layers of the neural network can effectively learn relevant features from the raw one-dimensional time series of closing prices, to be processed by the subsequent LSTM layers, which take into account the order of data in a sequence. We compare the performance of the CNN-LSTM to the performance of an LSTM, and find that both models have similar performance. Each neural network was supplied with a one-dimensional time series of closing prices, operating on the heuristic principle that the CNN component of the hybrid model would infer certain 'chart patterns' in the 1-D time series input, like a human chartist might, and use these as extracted features to feed to the LSTM layers.

2 Related Works

Much has been written on the subject of financial time series prediction via neural networks.

Lu et al. [1] examined the accuracy of several types of neural networks, namely MLP, RNN, LSTM, CNN and CNN-LSTM, in predicting the price of the Shanghai Composite Index, and found that the hybrid CNN-LSTM had the best accuracy among all models examined, though the authors included other information (trade volume, open/close price, etc) as input to their model. Similar work was done by Wu et al. [2], although this time considering the inputs to a hybrid model as specially formatted images encoded with several features other than just closing price on a selection of Taiwanese and American stocks. Aungiers [3] used a deep LSTM to make autoregressive price movement forecasts based on successive one-step-ahead predictions, with the aim of broadly capturing long term trends in the S&P 500, in an informal article published online.

Velay and Daniel [4] applied CNN's to classify chart patterns, the patterns mentioned in the introduction which are studied by so-called 'chartists', and found that formatting input data as actual images of charts yielded better results than a 1-D time series, but we decided to focus on 1-D time series in this work and leave the 2-D image case to future work.

Alhussein et al. [5] compared the performance of CNN-LSTM architectures on short term individual household electrical load prediction, and found that this architecture improved on prediction accuracy compared to other techniques, namely CNN and LSTM alone. While this work was not applied to financial time series, the lessons from it may be relevant to other time series prediction applications.

This paper differs slightly from the aforementioned ones, in that we seek to understand the effect of 1-D convolutional layers in improving prediction tasks relative to plain LSTM, and that we also aim to explore the applicability of these models to medium term trend prediction, rather than only one-timestep-ahead prediction.

3 Problem Formulation

We wish to train a neural network to predict the price of a FOREX pair, in this case, CAD/USD pair. That is, given a historical time series of m consecutive closing prices

$$w_t^m = \{p_t, p_{t+1}, \dots p_{t+m-1}\}$$

we wish to predict p_{t+m}, the closing price on the next day after the provided history. That is, our feature-label pairs were (w_t^m, p_{t+m}).

3.1 Data Preparation

We received a dataset from Alpine Macro, a financial analysis firm based in Montreal, QC, consisting of the daily closing prices of several different currencies, denominated in US dollars, spanning up to fifty years. For this work, we decided to focus on the US/CAD pair, and considered the daily closing prices since Jan. 1st, 1990 up until the present.

Unlike [1] and [3], who only considered train/test splits, we split the dataset split into train (70 %), validation (20 %), and test (10 %) periods in chronological order to eliminate look-ahead bias by predicting the past using information from the future. The training set covered the time period from Jan. 1st, 1990 until May 20th, 2013; the validation set, from May 23rd, 2013 until Sept. 5th, 2018; and the test set, from Sept. 6th, 2018, until Nov. 12, 2021. Each of these disjoint sets were split into overlapping "windows" w_t^m of width m, such that $w_t^m \cap w_{t+1}^m = \{p_{t+1}, p_{t+2}, \dots p_{t+m-2}\}$.

The raw data were normalized to aid training, but care had to be taken to do so in a way that would not introduce look-ahead bias. We followed the method of [6], which normalized each feature/label pair as follows:

$$(w_t^m, p_{t+m}) \mapsto \left(\frac{w_t^m - \overline{p}_t}{\sigma_t}, \frac{p_{t+m} - \overline{p}_t}{\sigma_t} \right)$$

where $\overline{p}_t$ is the mean of the elements of w_t^m, and σ_t is the standard deviation of the elements of w_t^m.

3.2 LSTM

We present a brief treatment of LSTM cells, from Zhang, Lipton, Li & Smola's *Dive into Deep Learning* [7]: LSTM, or Long Short Term Memory neural networks, operate on sequence data, as opposed to standard deep neural networks. The main idea is that, in addition to the hidden state $\boldsymbol{H}_t$ common to all recurrent neural networks, a cell memory $\boldsymbol{C}_t$ is maintained, which is moderated by three gates: an input gate $\mathbb{I}_t$, a forget gate $\boldsymbol{F}_t$, and an output gate $\boldsymbol{O}_t$, all in $\mathbb{R}^{n\times h}$.

Let $\{\boldsymbol{X}_1, \boldsymbol{X}_2, \dots \boldsymbol{X}_m\}$ be a sequence of data points in $\mathbb{R}^{n\times d}$–that is, a batch of size n of d-dimensional sequence data, with a sequence length of m. Let h be the number of hidden units in the LSTM cell. Then, with

$$\sigma : \mathbb{R} \to [0,1],\ u \mapsto \frac{1}{1+e^{-u}}$$

as the sigmoid function, we compute the input, forget, and output gates $\boldsymbol{I}_t, \boldsymbol{F}_t, \boldsymbol{O}_t \in \mathbb{R}^{n\times h}$ for the t'th time step:

$$\boldsymbol{I}_t = \sigma\left(\boldsymbol{X}_t\boldsymbol{W}_{xi} + \boldsymbol{H}_{t-1}\boldsymbol{W}_{hi} + \boldsymbol{b}_i\right) \tag{1}$$
$$\boldsymbol{F}_t = \sigma\left(\boldsymbol{X}_t\boldsymbol{W}_{xf} + \boldsymbol{H}_{t-1}\boldsymbol{W}_{hf} + \boldsymbol{b}_f\right) \tag{2}$$
$$\boldsymbol{O}_t = \sigma\left(\boldsymbol{X}_t\boldsymbol{W}_{xo} + \boldsymbol{H}_{t-1}\boldsymbol{W}_{ho} + \boldsymbol{b}_o\right) \tag{3}$$

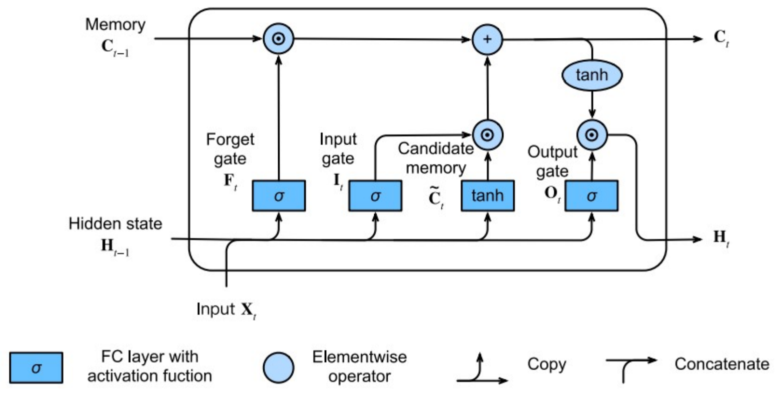
where $\boldsymbol{W}_{xi}, \boldsymbol{W}_{xf}, \boldsymbol{W}_{xo} \in \mathbb{R}^{d\times h}$, $\boldsymbol{W}_{hi}, \boldsymbol{W}_{hf}, \boldsymbol{W}_{ho} \in \mathbb{R}^{h\times h}$ are learnable weight parameters, $\boldsymbol{b}_i, \boldsymbol{b}_f, \boldsymbol{b}_o \in \mathbb{R}^{1\times h}$ are learnable biases[1], and σ is applied elementwise.

We next compute a candidate memory $\tilde{\boldsymbol{C}}_t \in \mathbb{R}^{n\times h}$:

$$\tilde{\boldsymbol{C}}_t = \tanh\left(\boldsymbol{X}_t\boldsymbol{W}_{xc} + \boldsymbol{H}_{t-1}\boldsymbol{W}_{hc} + \boldsymbol{b}_c\right) \tag{4}$$

where $\boldsymbol{W}_{xc} \in \mathbb{R}^{d\times h}$, $\boldsymbol{W}_{hc} \in \mathbb{R}^{h\times h}$, and $\boldsymbol{b}_c \in \mathbb{R}^{1\times h}$, and $\tanh : \mathbb{R} \to [-1,1]$.

Figure 1: An LSTM Cell–Zhang, Lipton, Li & Smola, *Dive Into Deep Learning* [7]



Memory $\mathbf{C}_{t-1}$
$\mathbf{C}_t$
tanh
Forget gate $\mathbf{F}_t$
Input gate $\mathbf{I}_t$
Candidate memory $\tilde{\mathbf{C}}_t$
Output gate $\mathbf{O}_t$
σ
tanh
Hidden state $\mathbf{H}_{t-1}$
$\mathbf{H}_t$
Input $\mathbf{X}_t$
FC layer with activation fuction
Elementwise operator
Copy
Concatenate

We update the cell memory for the t'th time step, where $\odot$ is the elementwise Hadamard product:

$$\boldsymbol{C}_t = \boldsymbol{F}_t \odot \boldsymbol{C}_{t-1} + \boldsymbol{I}_t \odot \tilde{\boldsymbol{C}}_t. \tag{5}$$

Having passed through σ, the entries of $\boldsymbol{F}_t \in [0,1]$, and thus regulate which part of the cell memory from the previous time step should be 'forgotten' when computing the cell memory $\boldsymbol{C}_t$. Likewise, we regulate which parts of the input to 'introduce' from the candidate cell memory $\tilde{\boldsymbol{C}}_t$.

Next, we must compute the hidden state $\boldsymbol{H}_t$:

$$\boldsymbol{H}_t = \boldsymbol{O}_t \odot \tanh(\boldsymbol{C}_t). \tag{6}$$

[1]We use broadcasting notation here–the biases are added row-wise to the matrix summands

The logic is similar to the above. If the entries of the output gate are close to zero, the hidden state $\boldsymbol{H}_t$ passed along at the next time step ignores the cell memory. If the entries are close to 1, we essentially forward the cell memory to the next time step via the hidden state $\boldsymbol{H}_t$.

The introduction of the ‘forget’, ‘input’ and ‘output’ gates alleviate the vanishing and exploding gradient problem, which is an important limitation of vanilla recurrent neural network cells, allowing an LSTM network to process much longer sequences than their earlier counterparts [7].

3.3 CNN

A convolutional neural network is designed to learn features from data. They are primarily used in image recognition, where the relevant features in an image do not change based on the spacial location in an image; for example, a cat is still a cat if its face appears in the corner of an image rather than in the center. A human learns to extract the spatially invariant ‘catness’ information regardless of scale or perspective, and a CNN seeks to emulate this.

In the one-dimensional case, we consider an input time series w as a vector in $\mathbb{R}^m$. Let $f_1, f_2, \dots, f_l$ be a set of ‘filters’ of length $0 < a \leq m$ in $\mathbb{R}^a$. Each filter is slid across w and a dot product is computed, shifting one or more entries at a time to the right. The convolution operation with the filter f_i yields an output sequence h_i of some length k, so that taken together, the output of the convolution operation with l-many filters of length a is an $l \times k$ matrix, or a new time series with more than one dimension of features. After the convolution operation, we apply a nonlinear activation function such as $\text{ReLu}(x) = \max\{x, 0\}$. Note that an element h_i^j in output sequence h_i is exposed to a-many elements of the input sequence w.

Stacking many such convolutional layers, each member of some output sequence has an ever-wider view of the original input sequence w, so that this layering allows the CNN to extract features at increasing scales, so that by the last layer, the CNN has learned to recognize high-level features. Typically, the output of the last convolutional layer is passed to a sequence of fully connected layers as in a standard deep neural network, and a prediction is made.

In order to reduce the number of parameters to be learned, and hence speed up training, we often introduce pooling layers after each convolutional layer to downsample the inputs. For example, a max-pooling layer slides a filter across an input sequence and computes the maximal element in the filter, thus reducing the size of the output, as well as having a smoothing effect on the input.

For a more detailed treatment of CNN’s, we refer the reader to [7].

3.4 Neural Network Architectures

We present the architectures of the LSTM and CNN-LSTM networks here:

Table 1: LSTM network architecture

LSTM 1	Units: 64 Dropout rate: 0.2 Activation: $\tanh$ Return sequences: True
LSTM 2	Units: 64 Dropout rate: 0.2 Activation: $\tanh$ Return sequences: True
LSTM 3	Units: 32 Dropout rate: 0.2 Activation: $\tanh$ Return sequences: False
Dropout	Dropout rate: 0.5
Dense	Units: 1

We trained each network for 500 epochs, stopping the training when the validation error did not improve over 50 consecutive epochs. We used the Adam optimizer (learning rate 0.001), and our

Table 2: CNN-LSTM network architecture

Conv1D 1	Kernel size: 3 Filters: 32 Activation: ReLu Padding: Same Strides: 1
Max Pool 1D	Pool size: 2 Padding: Same
Conv1D 2	Kernel size: 3 Filters: 16 Activation: ReLu Padding: Same Strides: 1
Max Pool 1D	Pool size: 2 Padding: Same
Conv1D 3	Kernel size: 3 Filters: 8 Activation: ReLu Padding: Same Strides: 1
Max Pool 1D	Pool size: 2 Padding: Same
LSTM 1	Units: 64 Dropout rate: 0.2 Activation: $\tanh$ Return sequences: True
LSTM 2	Units: 64 Dropout rate: 0.2 Activation: $\tanh$ Return sequences: True
LSTM 3	Units: 64 Dropout rate: 0.2 Activation: $\tanh$ Return sequences: False
Dropout	Dropout rate: 0.5
Dense	Units: 1

loss function was the mean squared error. We saved the model that had the best validation loss over all epochs of the training run.

Our choice for the parameters of the convolutional layers in Table 2 was guided by [5], who employed a common'coarse-to-fine' approach for feature extraction. The parameters for the lower LSTM layers in the CNN-LSTM network, which are identical to those of the LSTM network in Table 1, were taken from [3], to see what improvements could be made to forecast accuracy if convolutional layers were added to their LSTM architecture for S&P 500 price prediction. We employ dropout layers as a simple way to mitigate overfitting.

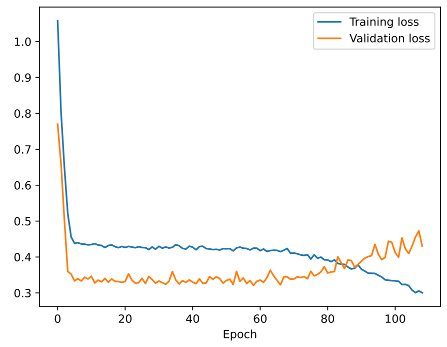
4 Main Results

We trained two versions of each architecture-one version considered a lookback history of 30 days, the other a lookback history of 60 days, and recorded the mean squared error (MSE) and mean absolute error (MAE) of the model's one-day-ahead forecasts on the test set. It was found that the CNN-LSTM architecture performed very similarly to the LSTM architecture, with the former beating the latter for a 30-day lookback period, but the latter beating the former for the 60-day lookback history.

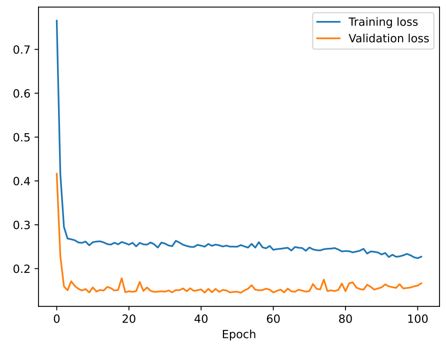
We remark that the validation loss is less than the training loss in all training runs. This is highly unusual, but we theorize it is due to the validation set being somehow 'easier' to predict than the

Table 3: Performance of LSTM and CNN-LSTM architectures on the test set

Neural Network	30-day history		60-day history	
	Test MSE	Test MAE	Test MSE	Test MAE
LSTM	0.5397	0.5132	0.1955	0.3175
CNN-LSTM	0.4708	0.4932	0.2277	0.3430

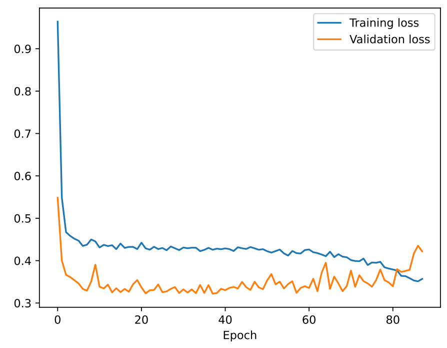


(a) LSTM training, 30 day history

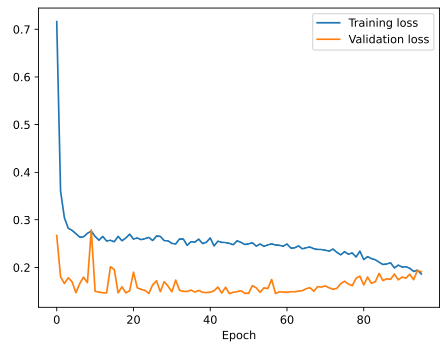


(b) LSTM training, 60 day history

Figure 2: LSTM learning curves, MSE loss



(a) CNN-LSTM training, 30 day history



(b) CNN-LSTM training, 60 day history

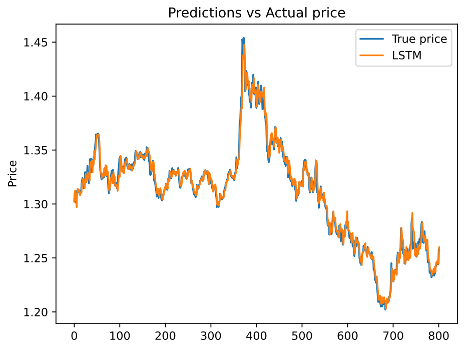
Figure 3: CNN-LSTM learning curves, MSE loss

training set was. We randomly generated synthetic random walk data to train the same architectures on, and for those runs the validation loss was higher than the train loss, as one would expect; we therefore postulate that this low validation loss is a property of the dataset, not of the models.

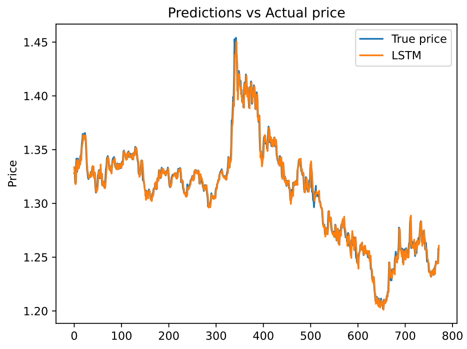
Although the results from Figures 4 and 5 look encouraging, they are not useful. Close inspection of the graphs shows that the prediction furnished by either model is almost entirely correlated with the last price found in the history window–that is, our models are essentially predicting that the price one day in the future will be the last price observed in the history.

5 Experiments

We wished to investigate, on an admittedly informal basis, the predictive power of our models, especially to see if they could identify medium term trends in the price of the CAD/USD pair. To this end, we used autoregression to make a 14-day forecast given a price history of 30 or 60 days, for each model described above. A history of 30 (60) days was fed as input to each model, and the predicted price was appended to the provided history, before shifting the history window ahead by

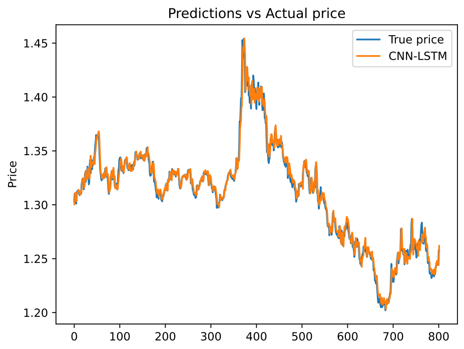


(a) LSTM predictions, 30 day history

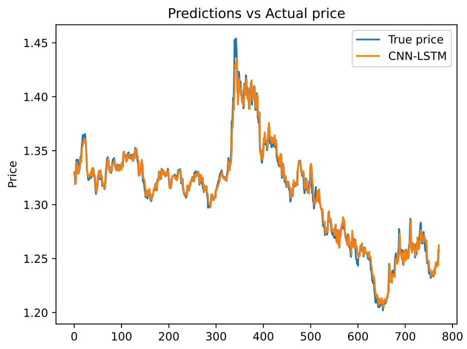


(b) LSTM predictions, 60 day history

Figure 4: LSTM one-day-ahead price predictions



(a) CNN-LSTM predictions, 30 day history



(b) CNN-LSTM predictions, 60 day history

Figure 5: CNN-LSTM one-day-ahead price predictions

one day. This was repeated until the model had predicted 14 days, after which the history window was shifted and the process was restarted, in the same spirit as [3]. We present the results from the models trained on a 60-day history.

6 Conclusion

Two neural network architectures for forecasting FOREX rates were investigated–LSTM, and a hybrid CNN-LSTM, to see what effect the convolutional layers had on the models' predictive power. Each model had similar performance, with no clear better–although this could be a matter of finding the right hyperparameters for the convolutional layer. It was found moreover that a longer input price history seemed to give more accurate models. Visually inspecting Figures 6 and 7, we find the medium term forecasts range from approximately correct to disastrous, with no clear preference to which; and that in one-day-ahead prediction, each model predicts that the price won't change much from the end of the provided history.

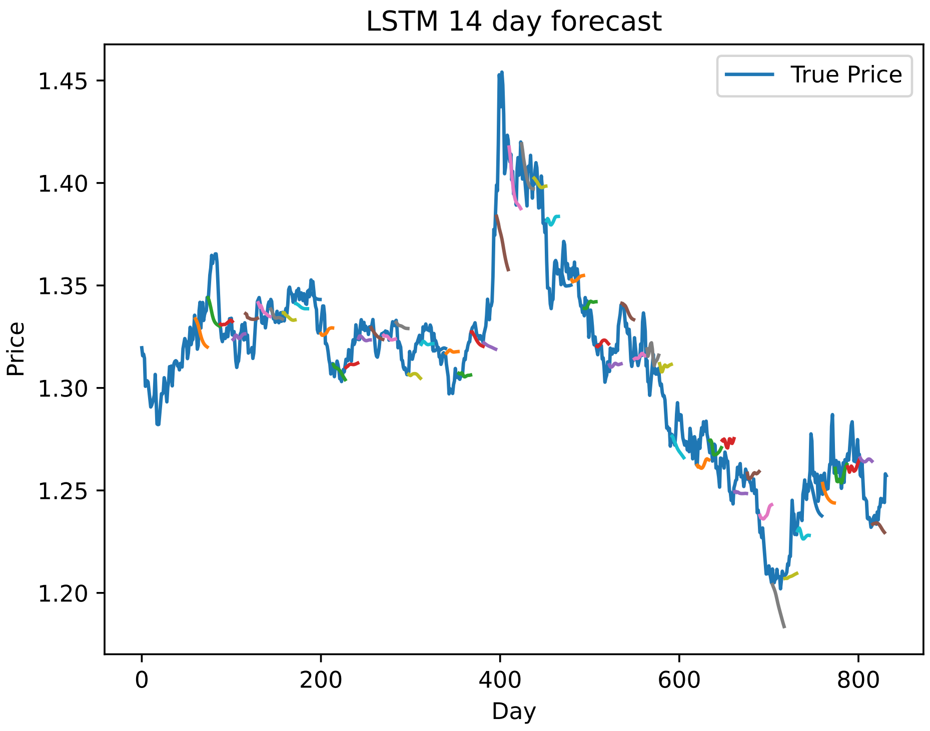


Figure 6: LSTM 14-day trend prediction, 60 day history

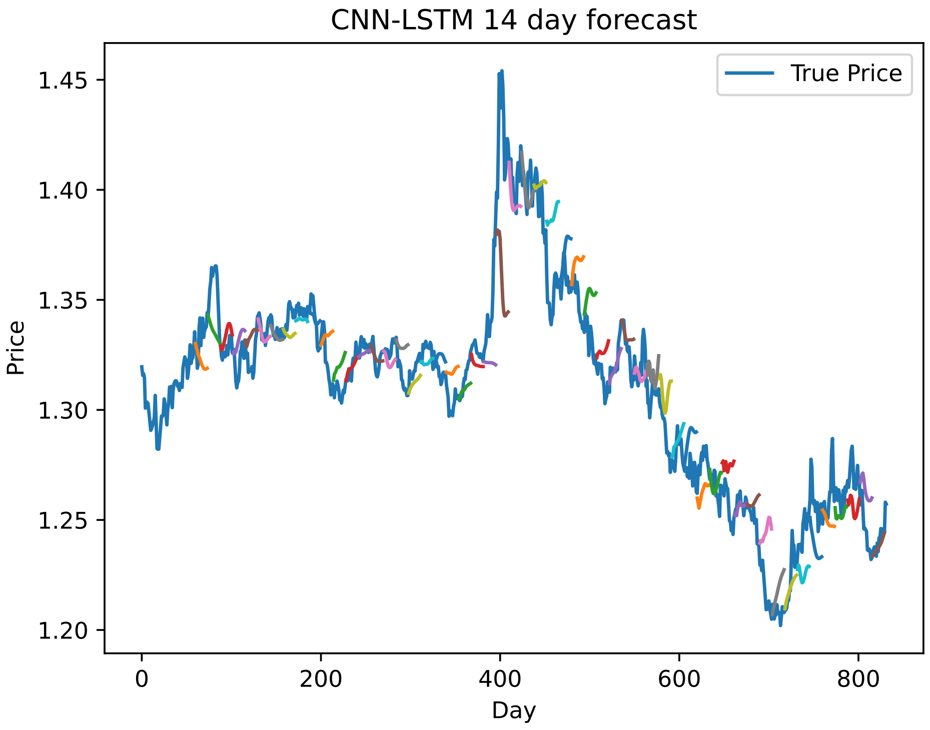


Figure 7: CNN-LSTM 14-day trend prediction, 60 day history

Acknowledgement

We would like to warmly thank Henry Wu and Andrew Gillich at Alpine Macro, based in Montreal, QC, for sharing their datasets with us, and for their many helpful suggestions. We also thank Prof. Yu for his insight, guidance, and patience.

References

[1] Wenjie Lu, Jiazheng Li, Yifan Li, Aijun Sun, and Jingyang Wang. "A CNN-LSTM-based model to forecast stock prices". *Complexity*, vol. 2020 (2020) (cit. on p. 2).

[2] Jimmy Ming Tai Wu, Zhongcui Li, Norbert Herencsar, Bay Vo, and Jerry Chun Wei Lin. "A graph-based CNN-LSTM stock price prediction algorithm with leading indicators". *Multimedia Systems* (2021) (cit. on p. 2).

[3] Jakob Aungiers. "Time Series Prediction Using LSTM Deep Neural Networks". *AltumIntelligence.com* (Sept. 2018) (cit. on pp. 2, 5, 7).

[4] Marc Velay and Fabrice Daniel. *Stock Chart Pattern recognition with Deep Learning*. 2018 (cit. on p. 2).

[5] Musaed Alhussein, Khursheed Aurangzeb, and Syed Irtaza Haider. "Hybrid CNN-LSTM Model for Short-Term Individual Household Load Forecasting". *IEEE Access*, vol. 8 (2020), pp. 180544–180557 (cit. on pp. 2, 5).

[6] Dingming Wu, Xiaolong Wang, Jingyong Su, Buzhou Tang, and Shaocong Wu. "A labeling method for financial time series prediction based on trends". *Entropy*, vol. 22 (10 Oct. 2020), pp. 1–25 (cit. on p. 2).

[7] Aston Zhang, Zachary C Lipton, Mu Li, and Alexander J Smola. *Dive into Deep Learning Release 0.16.1*. 2021 (cit. on pp. 3, 4).