NTNU
Kunnskap for en bedre verden

Department of Computer Science

TDT4173 - Assignment 3

An Empirical Evaluation of Convolutional and Recurrent Networks for Stock Time Series

Group:
Deep_Learning_Group 16

Authors:
Haakon Hafsahl Svane (haakohsv)
Lei Cheng (leic)

November 26, 2020

Preface

The aim of the project is to classify stock time series. Both convolutional and recurrent networks are applied. After analyzing multiple evaluation metrics, temporal convolutional network (TCN) is proved to be the best among all three models. In addition, normalization is proved to help training models.

GitHub

The full source code for the project, as well as all the raw data, preprocessed data and results are found on the project github here: https://github.com/HaakonSvane/Icarus. The repository contains multiple README's for all of the subpackages created as well as a master README in the root directory.

Website

A supplementary website was created as well.
This is found here: https://haakonsvane.github.io/Icarus. The purpose of the website is to give additional information about the data sourcing, labeling process and the preprocessing stage. In addition to this, a *further work* section is presented. The section introduces new ideas for applying neural networks on the dataset as well as discussing shortcomings of this project.

List of definitions

Technical terms and terminology in finance can be confusing. Below is a list of explanations and definitions for terminology that will be used in the project.

Open
The price of a stock at the start of a trading period.

Close
The price of a stock at the end of a trading period. This is often the value used in stock price plots.

High
The highest price recorded during the trading period.

Low
The lowest price recorded during the trading period.

Volume
The number of stocks traded during the trading period.

Trading hours
Normal trading hours. The times vary between different exchanges. The project follows standard NYSE/NASDAQ times (09:30 - 16:00).

RSI
(Relative strength index) A popular technical indicator often used in stock trading. It is a momentum oscillator describing the momentum of price changes. ranges from 0 to 100.

Adjusted prices
Adjusted stock prices take split and dividend events into account. These events often drastically affect the price, resulting in large spikes had the prices not been adjusted.

Table of Contents

List of Figures

List of Tables

1 Introduction

The aim of the project is to classify key points from stock market trading prices using multiple deep learning methods.

Predicting future values based on observed stock market prices is a challenging task. Some even argue that it is theoretically impossible [Malkiel, 2003]. Instead of predicting future values, this project will classify *buy*, *hold* and *sell* points in stock market using multiple financial indicators. The problem size is becomes more feasible when the desired output dimension is reduced to only these three classes.

This project is divided into four main steps: dataset extraction, data labeling and normalizaion (preprocessing), deep learning models and analysis of results.

Stock time series data are extracted using Alpha Vantage's API [Alpha Vantage Inc., 2020]. A labeling algorithms is applied to label regions in the time series as either *buying*, *holding* or *selling* points. From this perspective, this project is in essence a supervised multi-classification task.

Recurrent Neural Networks (RNNs) are a conventional family of networks for processing time series data. This project uses two gated RNNs, namely, long short-term memory (LSTM) [Hochreiter and Schmidhuber, 1997] and gated recurrent unit (GRU) [Cho et al., 2014]. Recent research [Oord et al., 2016, Lea et al., 2017] indicate that Convolutional Neural Networks (CNNs) can also be applied to process time series data and can even outperform RNNs [Bai et al., 2018]. This project will also explore a special kind of CNN, namely a temporal convolutional network (TCN). In order to study the performance of the three models on time series data of different length, each network is tested on data slices of two different lengths.

This project evaluates these three deep learning methods with respect to accuracy, confusion matrix, recall, precision and F1 score.

2 Related Work

Stock market prediction has long been an attractive area of study. In recent years, deep learning based models have started appearing in financial studies [Chen et al., 2016, Fischer and Krauss, 2018, Nabipour et al., 2020]. This has however, proven to be a notoriously difficult task. Unfortunately, the results of applying powerful machine learning methods to these kinds of problems have yet to show promising prospects .

A more pragmatic approach is instead to study simpler tasks. For example, Chen et al. [2016] proposed a deep learning framework based on CNNs to extract numerical features from stock time series data for further financial analysis; Fischer and Krauss [2018] deployed LSTM for predicting out-of-sample directional movements to formalize a rules-based short-term reversal strategy that yields 0.23 percent prior to transaction costs; Nabipour et al. [2020] employed several machine learning algorithms on prediction of for stock market groups (diversified financials, petroleum, non-metallic minerals and basic metals).

This project is mainly inspired by the work of Sezer and Ozbayoglu [2018], and it can therefore be considered an improvement to existing work. The aim of Sezer and Ozbayoglu's work is to determine buying, selling and holding points in stock prices using multiple technical indicators. This project manages to complete a similar task, with the main improvements being the following two aspects:

First, this project adopts a new labeling algorithm. Some issues withSezer and Ozbayoglu's process was firstly that it created an imbalanced dataset that hindered the training and testing of several deep learning models. Secondly, in practice, buying and selling points can not be considered instantaneous points since this would require an instantaneous reaction of a trader in order to be acted on. Because of this, this project proposes thresholds instead of extremum values to determine the buying, selling and holding points.

Second, this project adopts a simplified convolutional method. The work of Sezer and Ozbayoglu [2018] first converts 1D stock time series into 2D image-like data in order to utilize the power of 2D CNNs. This may be an unnecessary and redundant step. Therefore, this project directly applies 1D CNNs on 1D time series data.

3 Data

All sourcing of the data was done using Alpha Vantage's [Alpha Vantage Inc., 2020] API service. This section presents how this data was extracted from Alpha Vantage, what is characteristics are, and how it was preprocessed for training.

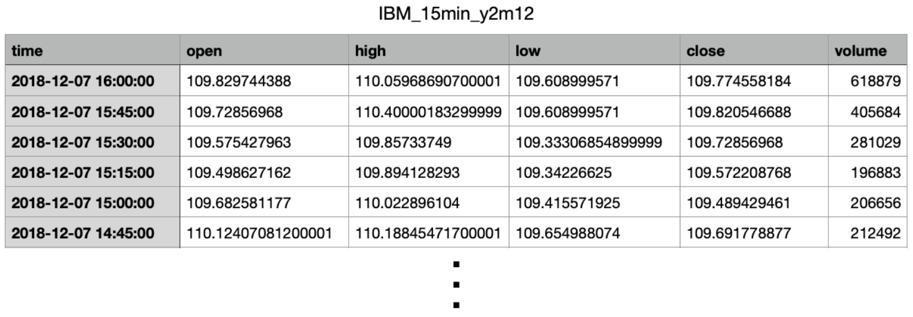
3.1 Dataset extraction

3.1.1 The API

Alpha Vantage's service provides a wide range of data over long term periods. There are a total of 5 classes of requests available. These are *Stock Time Series*, *Fundamental Data*, *Forex (FX)*, *Cryptocurrencies* and *Technical Indicators*. Of these 5 classes, *Stock Time Series* and *Technical Indicators* where used in the data sourcing stage of the project. *Stock Time Series* has API functions for intraday data with extended history. This means that we are able to request data with high temporal resolution (down to 1 minute) for the two trailing years. The return format of the calls are .csv files in monthly slices. This means that for a company that has been listed for two or more years, a two year period will consist of 24 slices of data, each containing a month of data with the specified temporal resolution. Alpha Vantage provides several premium services. All of these aim at increasing the allowed api calls per minute as well as the number of daily allowed calls.

3.1.2 Extraction process

In order to tackle the call restriction of the free service provided by Alpha Vantage and to gain full control over the sourcing process, a python wrapper was created for the api[1]. The temporal resolution of the requested data was set to 15 minutes. The reasoning for choosing this value is that the time windows to be studied are between 1 day to 1 week of trading. The number of data points in this range is then respectively 26 and 130 for normal trading hours (6.5 hours). Figure 1 shows a small slice of what the raw data provided by Alpha Vantage looks like. All the sourced data is adjusted. To ensure the highest quality data possible, the api looked through all companies listed on the S&P500 index that have been listed for two or more years.



IBM_15min_y2m12

time	open	high	low	close	volume
2018-12-07 16:00:00	109.829744388	110.05968690700001	109.608999571	109.774558184	618879
2018-12-07 15:45:00	109.72856968	110.40000183299999	109.608999571	109.820546688	405684
2018-12-07 15:30:00	109.575427963	109.85733749	109.33306854899999	109.72856968	281029
2018-12-07 15:15:00	109.498627162	109.894128293	109.34226625	109.572208768	196883
2018-12-07 15:00:00	109.682581177	110.022896104	109.415571925	109.489429461	206656
2018-12-07 14:45:00	110.12407081200001	110.18845471700001	109.654988074	109.691778877	212492

⋮

Figure 1: A small slice of data (IBM) showing all variables for each data point.

[1] https://github.com/HaakonSvane/Icarus/tree/master/src/alphavantage

3.2 Data labeling

The process of labeling the data[2] with *buy*, *hold* and *sell* is inspired from the method presented in [Sezer and Ozbayoglu, 2018]. The authors present a windowed min-max approach for labeling. They later discuss the shortcomings of this approach as this has the effect of creating an imbalanced dataset since the labels for *buy* and *sell* occur very seldom in the dataset. This was also reflected in their result since the network had a tendency to eagerly classify any points as *hold* since this label occured so often in the training process. This motivated creating labeled *regions* instead of single points.

The determination of the labels for a point $P(t)$ requires insight into future values. The idea behind the new labeling algorithm is to use convolutions with a weighted window to determine how a closing price $P(t)$ compares to future prices. Multiple convolution windows was tried, but a cubic weighted window gave the best results. The results of applying this window is shown in Figure 2. For a convolution window of size `WIN_SIZE`, the *look ahead* period for the labeler is `WIN_SIZE`/2.

The difference between the closing stock price and the weighted average is calculated, and a sliding median window is applied to the difference. This is illustrated in Figure 3. A label is determined based on the value of the factor `med_val/close_val`. If this fraction is larger some buying threshold value, the point is determined to be a buying point. If the fraction is lower than some selling threshold value, the corresponding point is determined to be a selling point. If none of the above, it is labeled as a holding point. The final result of the labeler is illustrated in Figure 4

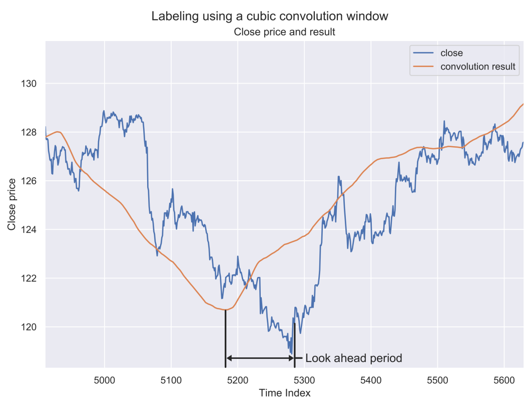


Figure 2: Slice of IBM stock (blue line) with a cubic convolution window applied (orange line). The look ahead period is also shown.

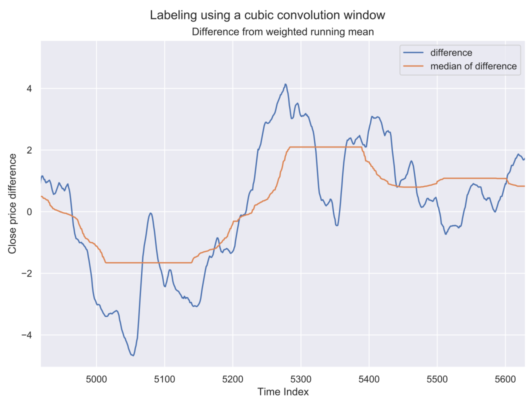


Figure 3: Difference from the closing price and the convolution result (blue line) for a slice of IBM stock. A sliding median window is applied to this difference (orange line).

3.3 Data preparation and normalization

3.3.1 Trading period trimming and RSI

Some of the data that was sourced from Alpha Vantage lists before/after hours trading prices as well. These are special trading periods that are reserved for some certain exchanges and investors. Since this only applies to some of the sourced data and since the investment patterns during these periods differ from the normal trading hours, it was decided to trim all of these periods away. This ensures that all the data only list stock prices during normal trading hours.

RSI was also added to the data in the preprocessing stage. The reason for adding RSI is due to its popular usage in investing. The period of the RSI was set to half the window size used by the labeler.

[2]https://github.com/HaakonSvane/Icarus/tree/master/src/stocklabeler

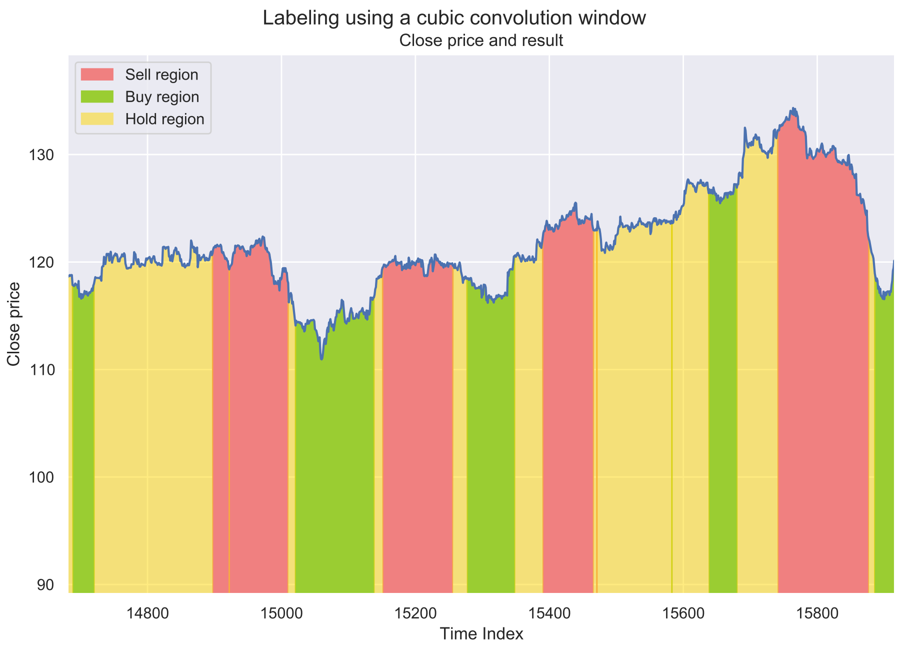


Figure 4: Labeled data from a slice of IBM stock (blue line) showing the assigned labels for buy, hold and sell as green, yellow and red respectively.

3.3.2 Normalization

Normalization of the data[3] was done using a modified version of the modified tanh estimator. This method resembles z-score normalization, but is clamped by a tanh function resulting in data within the range $[-1, 1]$. The reason for choosing this normalization method is because of its robustness and efficiency [Anil Jaina, 2005]. The modified tanh estimator is defined as

$$x' = \frac{1}{2}(\tanh{(0.01\frac{x-\mu}{\sigma})}+1)$$

This has a codomain of $[0, 1]$, but since a codomain of $[-1, 1]$ is desired, this was modified to

$$x' = \tanh{(0.1\frac{x-\mu}{\sigma})}$$

This modified version has a codomain of $[-1, 1]$ and a mean of zero. All values that lie within one standard deviation are mapped between approximately $[-0.1, 0.1]$. Since most applications of financial data come from real-time data, it is often impossible to properly normalize the data. What is meant by properly is that whatever normalization parameters are used to normalize some price point $P(t)$ can not guarantee to yield a valid normalization for all future values of $P(t)$. A good approach is then to instead only rely on previous values. Computing the mean and standard deviation then becomes the task of computing the running mean and running standard deviation of the data over some period T. For this project, the period for this process was set to half the window size, just like the RSI.

[3]https://github.com/HaakonSvane/Icarus/tree/master/src/preprocessing

3.3.3 Finalizing the data

After initial trimming, labeling and normalization has been performed on the data, a final trimming was applied where the first and last `WIN_SIZE`/2 values was trimmed away from each company dataset. This was done to remove edge effects that occur when computing running averages and convolutions. The finalized dataset available for learning consists of 15 min resolution data for a total of 491 companies, each over approximately a two year period.

3.4 Training and testing sets

The data used in this project are from 4 companies, namely, Apple Inc (AAPL), Alphabet Inc (GOOG), International Business Machines Corp (IBM), and Lincoln National Corp (LNC). For each company, two labeling lengths are employed, namely, 52 and 260. For each company and each length, this project samples 10400 instances of which 80% are training sets and 20% are testing sets. As mentioned previously, these datasets are imbalanced sets. Table 1 shows the class distribution.

Table 1: Class distribution of datasets.

Length 52

Company	Traning			Testing		
	buy	*hold*	*sell*	*buy*	*hold*	*sell*
AAPL	1297 (0.1559)	5852 (0.7034)	1171 (0.1407)	327 (0.1572)	1427 (0.6861)	326 (0.1567)
GOOG	1160 (0.1394)	6163 (0.7407)	997 (0.1198)	299 (0.1437)	1500 (0.7212)	281 (0.1351)
IBM	935 (0.1124)	6367 (0.7653)	1018 (0.1224)	241 (0.1159)	1565 (0.7524)	274 (0.1317)
LNC	1682 (0.2022)	4955 (0.5956)	1683 (0.2023)	422 (0.2029)	1225 (0.5889)	433 (0.2082)

Length 260

Company	Traning			Testing		
	buy	*hold*	*sell*	*buy*	*hold*	*sell*
AAPL	1470 (0.1767)	5276 (0.6341)	1574 (0.1892)	351 (0.1688)	1337 (0.6428)	392 (0.1885)
GOOG	1649 (0.1982)	4989 (0.5996)	1682 (0.2022)	412 (0.1981)	1276 (0.6135)	392 (0.1885)
IBM	1595 (0.1917)	5243 (0.6302)	1482 (0.1781)	378 (0.1817)	1342 (0.6452)	360 (0.1731)
LNC	2108 (0.2534)	4244 (0.5101)	1968 (0.2365)	545 (0.2620)	1057 (0.5082)	478 (0.2298)

4 Methods

4.1 LSTM, GRU and TCN

The aim of this project is to classify time series data as either *buy*, *hold* or *sell*. Therefore. three sequence models are compared, namely LSTM [Hochreiter and Schmidhuber, 1997], GRU [Cho et al., 2014], and TCN [Lea et al., 2017].

4.1.1 Recurrent networks

RNNs are a powerful family of neural networks that are specialized for processing sequential data [Goodfellow et al., 2016, p. 373-374]. RNNs can scale to long sequences and can also process sequences of variable length. In a recurrent network, each member of the output is a function of the previous members of the output. A recurrent network share the same weights across several time steps. This means that each member of the output is produced using the same update rule applied to the previous outputs.

For example, a typical recurrent network can be expressed as

$$s^{(t)} = f(s^{(t-1)}; \theta)$$

where $s^{(t)}$ and $s^{(t-1)}$ refer to the current state and the previous state respectively. θ represents the same parameters shared across the whole structure. Its unfolded computational graph is illustrated in Figure 5.

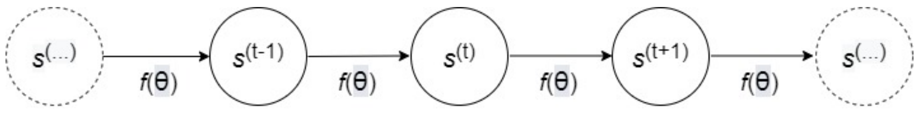


Figure 5: Typical RNN architecture

In order to make RNNs more expressive, a hidden unit is applied to every time step to add extra features. Hidden units have a similar equation

$$h^{(t)} = f(h^{(t-1)}, x^{(t)}; \theta)$$

where $h^{(t)}$ and $h^{(t-1)}$ refer to the current hidden unit and the previous hidden unit respectively. $x^{(t)}$ is the current input; θ is the sharing parameter.

RNNs maintain a vector of activations for each time step, which makes them extremely deep. However, this depth often makes them difficult to train due to the exploding and the vanishing gradient problems [Bengio et al., 1994].

LSTM networks are resistant to these problems and can remember long-term dependency because it introduces three gates in each time step to regulate the flow of information into and out of the cell. *Forge*t gates decides what information should be thrown away or kept. *Input* gates are used to update the cell state. *Output* gates decides what the next hidden state should be.

GRU is similar to LSTM. However, GRU gets rid of the cell state and thus it only has two gates. It is simple, but it is able to achieve similar results as LSTM . The *update* gates decide what information to throw away and what new information to add. *Reset* gates are used to decide what past information to discard.

4.1.2 Temporal convolutional networks

Traditionally, sequence modeling has been synonymous with recurrent networks, but in recent years, some research results indicate that certain convolutional networks can reach state-of-the-art accuracy using sequence modeling task [Kalchbrenner et al., 2016, Dauphin et al., 2017, Gehring et al., 2016, 2017]. The common association between sequence modeling and recurrent networks should perhaps therefore be reconsidered as convolutional networks could be regarded as a superior alternative method for sequence modeling task.

In order to process time series data, this project uses a special class of convolutional networks called temporal convolutional networks (TCNs), which have the following structures.

Casual convolutions

For time series data, the prediction at time t cannot depend on any of the future time steps. Therefore, TCNs use causal convolutions, where an output at time t is convolved only with elements from t and earlier in the previous layer.

Dialated convolutions

A simple causal convolution is not able to look back for an extended history which makes processing long time series data a challenge. One solution is to employ dilated convolutions that support exponential expansion of receptive fields [Yu and Koltun, 2015]. That is, the input stride is no longer 1 while the output stride is still 1 [Chen et al., 2014]. The structure of a dilated causal convolutional network is illustrated in Figure 6.

Residual convolutions

Theoretical work and numerical experiments show that residual connections can improve the performance of convolutional networks [He et al., 2016a,b]. Therefore, TCNs also follow this design. A residual block contains a branch which adds the input of a layer to its output:

$$O = Activation(x + F(x))$$

where x and $F(x)$ are the input and output of a layer, respectively.

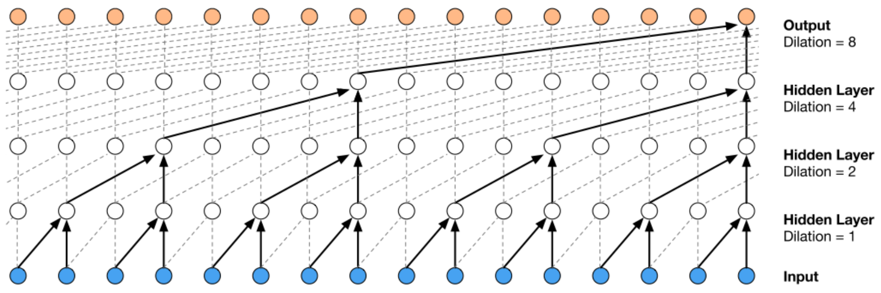


Figure 6: Dilated causal convolution architecture

Source: Oord et al. [2016]

Unlike RNNs where a later time step must wait for their predecessors to complete, TCNs can be done in parallel. This is a major advantage when tit comes to processing time.

4.2 Experiments

4.2.1 Dataset

This project uses the data from 4 companies. For each company, two sequence lengths, 52 and 260 are used. In addition, this project tests both non-normalized and normalized datasets. They are all imbalanced datasets where points labelled as *hold* occur more often than *buy* and *sell*. This is expected from the this type of data.

4.2.2 Models

This project compares three models, namely, LSTM, GRU, and TCN. For TCN, the depth of the network and the kernel size should be chosen carefully so that the receptive field of the last output covers the series data. For LSTM and GRU, this project keeps the number of parameters at about the same size with TCN. Table 2 presents the number of parameters for each model.

Table 2: Number of model parameters

Length	**LSTM**	**GRU**	**TCN**
52	1115	1233	1158
260	2513	2247	2308

4.2.3 Experimental setup

The project is mainly written in Python and utilizes external libraries such as pandas, numpy and scikit_learn. For modelling, testing and training the networks, the PyTorch library [Paszke et al., 2017] was used. Table 3 lists the hyperparameters and settings used for the three networks. These are held the same for all the models.

Table 3: Default hyperparameters and settings used for the networks.

Parameter	**Value**
Learning rate	0.0001
Epochs	1000
Batch size	32
Optimizer	Adam[4]
Loss function	Cross-entropy

5 Results

5.1 Accuracy

Accuracy is one of the most popular metrics in multi-classification task. The formula is defined as

$$\text{Accuracy} = \frac{\text{Number of correct predictions}}{\text{Total number of predictions}}$$

Accuracy is the probability that the model prediction is correct.

Testing accuracy results of LNC are shown in Figure 7. The results from the other companies are found in Appendix A.

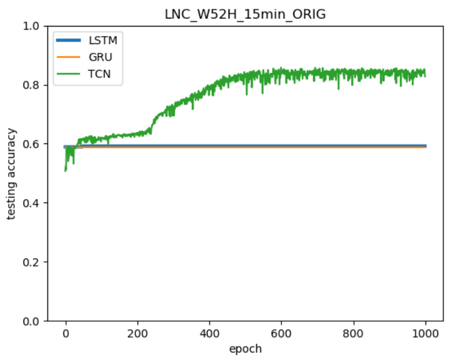
According to all of the testing accuracy results, the accuracy of TCN is increasing while both LSTM and GRU fail in most cases. LSTM shows minor improvement in two cases and GRU in three cases. All of these cases are for the normalized datasets.

The two most important results emerging from this is that TCN performs the best of the three networks and that normalization plays an important role in helping to train models.

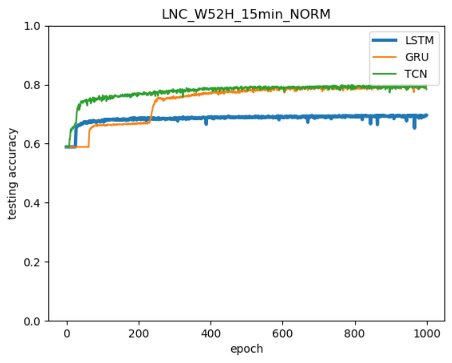
However, for an imbalanced dataset like the ones that are beeing used in the training, accuracy alone does not tell the full story. From Figure 7a, LSTM and GRU achieve about 60% accuracy. The accuracy diverge rapidly towards this value as well. By inspecting the predictions made by these models, it becomes clear why this is the case: The networks seem to (almost unconditionally) always predict *hold* labels for all the data. The class distribution table (see: Table 1) shows that the accuracy values are very close to the actual percentage of hold points in the labeled data. For the more imbalanced datasets such as IBM (Figure 10a), this is even more dramatic. All though the network achieves a 75%, which seems like a good result, LSTM and GRU therefore fails to classify the data properly.

To better asses the performance of the networks, more metrics are needed. These are presented in Section 5.2 and 5.3

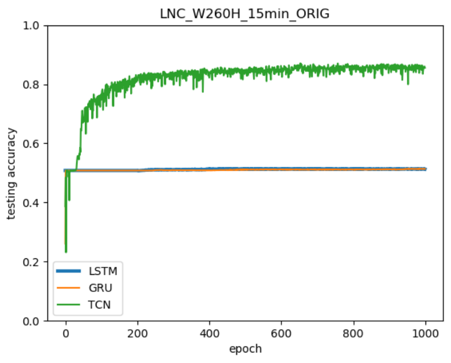
[4]Kingma and Ba [2014]



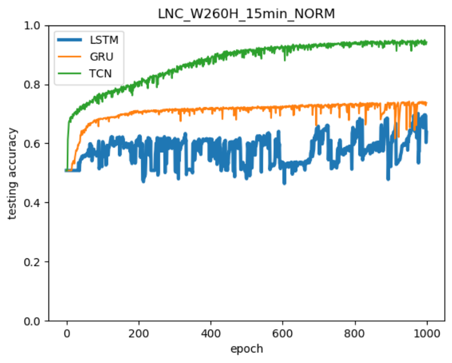
(a) Length=52, non-normalized



(b) Length=52, normalized



(c) Length=260, non-normalized



(d) Length=260, normalized

Figure 7: Testing accuracy results of LNC

5.2 Confusion matrix

The confusion matrix is a cross table that records the number of occurrences between the true classification and the predicted classification. It encloses all the relevant information about the classifier prediction performance.

Confusion matrices of the results from training on the LNC dataset are presented in Table 4. Rows (a) - (d) correspond to Figure 7. The confusion matrices for the other companies are found in Appendix B. Columns represents the network predictions whereas rows represent the true classifications. Because of this, the correctly classified elements are located on the main diagonal from top left to bottom right.

By inspecting these matrices, the results presented in 5.1 are better highligted for LSTM and GRU (non-normalized datasets) (a) and (c). Almost all predictions are classified into the second class, which means the two models fail in these cases. For the other cases, the networks seem to perform well however.

5.3 Precision, recall and f1-score

For binary-classification, precision is the fraction of true positive instantces divided by the total number of positively predicted instances (true positive and false positive). Precision tells us how much we can trust the model when it predicts an instance as positive.

$$\text{Precision} = \frac{\text{TP}}{\text{TP} + \text{FP}}$$

Recall is the fraction of true positive elements divided by the total number of positively classified

Table 4: Confusion matrices of LNC.
(a): Non-normalized data of sequence length 52. **(b)**: Normalized data of sequence length 52. **(c)**: Non-normalized data of sequence length 260. **(d)**: Normalized data of sequence length 260.

	LSTM				GRU				TCN			
		buy	*hold*	*sell*		*buy*	*hold*	*sell*		*buy*	*hold*	*sell*
(a)	*buy*	0	422	0	*buy*	0	422	0	*buy*	372	47	3
	hold	0	1225	0	*hold*	0	1225	0	*hold*	129	1068	28
	sell	0	427	6	*sell*	0	433	0	*sell*	6	146	281
		buy	*hold*	*sell*		*buy*	*hold*	*sell*		*buy*	*hold*	*sell*
(b)	*buy*	0	421	1	*buy*	305	117	0	*buy*	301	121	0
	hold	0	1153	72	*hold*	67	1058	100	*hold*	108	1080	37
	sell	0	138	295	*sell*	0	137	296	*sell*	0	184	249
		buy	*hold*	*sell*		*buy*	*hold*	*sell*		*buy*	*hold*	*sell*
(c)	*buy*	7	538	0	*buy*	7	538	0	*buy*	488	54	3
	hold	0	1057	0	*hold*	0	1057	0	*hold*	95	891	71
	sell	0	475	3	*sell*	0	474	4	*sell*	4	74	400
		buy	*hold*	*sell*		*buy*	*hold*	*sell*		*buy*	*hold*	*sell*
(d)	*buy*	141	400	4	*buy*	350	150	45	*buy*	514	31	0
	hold	41	918	98	*hold*	32	861	164	*hold*	22	998	37
	sell	1	211	266	*sell*	6	150	322	*sell*	2	28	448

units (true positive and false Negative). Recall measures the model's ability to find Positive in the dataset.

$$\text{Recall} = \frac{\text{TP}}{\text{TP} + \text{FN}}$$

Generally, a model cannot have both high precision and high recall. There is a cost associated with getting higher points in precision or recall. In order to comprehend both metrics, the f1-score is introduced. This is the harmonic mean of precision and recall.

$$\text{F1-score} = 2 \times \frac{\text{Precision} \times \text{Recall}}{\text{Precision} + \text{Recall}}$$

F1-score ranges from 0 to 1. 1 represents the best model while 0 the worst.

For multi-classification, all classes can be considered as Positive one by one, while other classes as a whole is regarded as Negative. In this way, for every class, precision, recall and f1-score can be calculated.

Precision, recall and f1-score of LNC are presented in Table 5. (a) - (d) correspond to Figure 7 and Table 4. The results from the other companies are found in Appendix C.

The results are similar to sec. 5.1 and 5.2. Among all three models, TCN gives the best results because the f1-scores are high for every class. For the same reason, GRU is better than LSTM. (a) and (c) use non-normalized dataset while (b) and (d) use normalised dataset. F1-scores of (b) and (d) are higher than (a) and (c), which seem to emphasize the importance of proper normalization. Considering LSTM and GRU for the non-normalized data (a) and (c), only the second class has high f1-score and others are close to zero. This further emphasizes their failure to properly classify the data.

Table 5: Precision, recall and f1-score of LNC.
(a): Non-normalized data of sequence length 52. **(b)**: Normalized data of sequence length 52. **(c)**: Non-normalized data of sequence length 260. **(d)**: Normalized data of sequence length 260.

		LSTM			GRU			TCN		
		P	R	f1	P	R	f1	P	R	f1
(a)	*buy*	0.00	0.00	0.00	0.00	0.00	0.00	0.73	0.88	0.80
	hold	0.59	1.00	0.74	0.59	1.00	0.74	0.85	0.87	0.86
	sell	1.00	0.01	0.03	0.00	0.00	0.00	0.90	0.65	0.75
(b)	*buy*	0.00	0.00	0.00	0.82	0.72	0.77	0.74	0.71	0.72
	hold	0.67	0.94	0.79	0.81	0.86	0.83	0.78	0.88	0.83
	sell	0.80	0.68	0.74	0.75	0.68	0.71	0.87	0.58	0.69
(c)	*buy*	1.00	0.01	0.03	1.00	0.01	0.03	0.83	0.90	0.86
	hold	0.51	1.00	0.68	0.51	1.00	0.68	0.87	0.84	0.86
	sell	1.00	0.01	0.01	1.00	0.01	0.02	0.84	0.84	0.84
(d)	*buy*	0.77	0.26	0.39	0.90	0.64	0.75	0.96	0.94	0.95
	hold	0.60	0.87	0.71	0.74	0.81	0.75	0.94	0.94	0.94
	sell	0.72	0.56	0.63	0.61	0.67	0.64	0.92	0.94	0.93

6 Conclusion

This project uses three deep learning methods to classify time series in to three classes. In essence, it is an imbalanced supervised multi-classification task.

Raw stock time series data are extracted using Alpha Vantage's API service. Some labeling and trimming methods are applied to preprocess the data. In addition, both non-normalized and normalized datasets are produced.

Three deep learning methods (LSTM, GRU, and TCN) are used to train and test. Multiple evaluation metrics (accuracy, confusion matrix, precision, recall, and f1-score) are used to analyze the results.

Overall, TCN performs the best among all three deep learning methods. For most of the data, TCN performs better and better during the training while both LSTM and GRU quickly stagnate to failure. LSTM and GRU did however perform better on the normalized data. This results seem to stress the importance of proper normalization.

The project aimed at studying the performance of the network on data with different sequence lengths. However, the degree of imbalance of the dataset also influence the model performance. This might either point to hyperparameters not being properly tuned for each case, or an improper labeling procedure that fails to adapt to differing time scales. Further study includes improving the labeling algorithm and further looking at the architecture of the TCN network that was used for the project. A more in-depth discussion of this as well as other new ideas for the network are presented in the "Further work" section on the project website[5]

[5]https://haakonsvane.github.io/Icarus/further-work.html

Bibliography

Alpha Vantage Inc. Alpha Vantage stock apis in json, excel & google sheets, 2020. URL `https://www.alphavantage.co/`.

Arun Ross Anil Jaina, Karthik Nandakumara. Score normalization in multimodal biometric systems. *Pattern Recognition*, 38:2270–2285, 2005.

Shaojie Bai, J Zico Kolter, and Vladlen Koltun. An empirical evaluation of generic convolutional and recurrent networks for sequence modeling. *arXiv preprint arXiv:1803.01271*, 2018.

Yoshua Bengio, Patrice Simard, and Paolo Frasconi. Learning long-term dependencies with gradient descent is difficult. *IEEE transactions on neural networks*, 5(2):157–166, 1994.

Jou-Fan Chen, Wei-Lun Chen, Chun-Ping Huang, Szu-Hao Huang, and An-Pin Chen. Financial time-series data analysis using deep convolutional neural networks. In *2016 7th International Conference on Cloud Computing and Big Data (CCBD)*, pages 87–92. IEEE, 2016.

Liang-Chieh Chen, George Papandreou, Iasonas Kokkinos, Kevin Murphy, and Alan L Yuille. Semantic image segmentation with deep convolutional nets and fully connected crfs. *arXiv preprint arXiv:1412.7062*, 2014.

Kyunghyun Cho, Bart Van Merriënboer, Caglar Gulcehre, Dzmitry Bahdanau, Fethi Bougares, Holger Schwenk, and Yoshua Bengio. Learning phrase representations using rnn encoder-decoder for statistical machine translation. *arXiv preprint arXiv:1406.1078*, 2014.

Yann N Dauphin, Angela Fan, Michael Auli, and David Grangier. Language modeling with gated convolutional networks. In *International conference on machine learning*, pages 933–941. PMLR, 2017.

Thomas Fischer and Christopher Krauss. Deep learning with long short-term memory networks for financial market predictions. *European Journal of Operational Research*, 270(2):654–669, 2018.

Jonas Gehring, Michael Auli, David Grangier, and Yann N Dauphin. A convolutional encoder model for neural machine translation. *arXiv preprint arXiv:1611.02344*, 2016.

Jonas Gehring, Michael Auli, David Grangier, Denis Yarats, and Yann N Dauphin. Convolutional sequence to sequence learning. *arXiv preprint arXiv:1705.03122*, 2017.

Ian Goodfellow, Yoshua Bengio, and Aaron Courville. *Deep Learning*. MIT Press, 2016. `http://www.deeplearningbook.org`.

Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun. Deep residual learning for image recognition. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 770–778, 2016a.

Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun. Identity mappings in deep residual networks. In *European conference on computer vision*, pages 630–645. Springer, 2016b.

Sepp Hochreiter and Jürgen Schmidhuber. Long short-term memory. *Neural computation*, 9(8): 1735–1780, 1997.

Nal Kalchbrenner, Lasse Espeholt, Karen Simonyan, Aaron van den Oord, Alex Graves, and Koray Kavukcuoglu. Neural machine translation in linear time. *arXiv preprint arXiv:1610.10099*, 2016.

Diederik P Kingma and Jimmy Ba. Adam: A method for stochastic optimization. *arXiv preprint arXiv:1412.6980*, 2014.

Colin Lea, Michael D Flynn, Rene Vidal, Austin Reiter, and Gregory D Hager. Temporal convolutional networks for action segmentation and detection. In *proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pages 156–165, 2017.

B. G. Malkiel. The efficient market hypothesis and its critics. *Journal of Economic Perspectives*, Volume 17:Pages 59–82, 2003.

Mojtaba Nabipour, Pooyan Nayyeri, Hamed Jabani, Amir Mosavi, E Salwana, et al. Deep learning for stock market prediction. *Entropy*, 22(8):840, 2020.

Aaron van den Oord, Sander Dieleman, Heiga Zen, Karen Simonyan, Oriol Vinyals, Alex Graves, Nal Kalchbrenner, Andrew Senior, and Koray Kavukcuoglu. Wavenet: A generative model for raw audio. *arXiv preprint arXiv:1609.03499*, 2016.

Adam Paszke, Sam Gross, Soumith Chintala, Gregory Chanan, Edward Yang, Zachary DeVito, Zeming Lin, Alban Desmaison, Luca Antiga, and Adam Lerer. Automatic differentiation in pytorch. 2017.

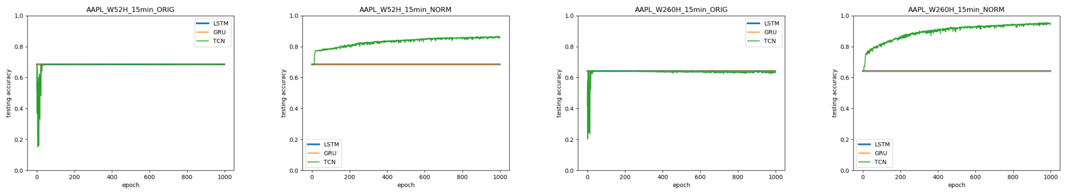
Omer Berat Sezer and Ahmet Murat Ozbayoglu. Algorithmic financial trading with deep convolutional neural networks: Time series to image conversion approach. *Applied Soft Computing*, 70: 525–538, 2018.

Fisher Yu and Vladlen Koltun. Multi-scale context aggregation by dilated convolutions. *arXiv preprint arXiv:1511.07122*, 2015.

Appendix

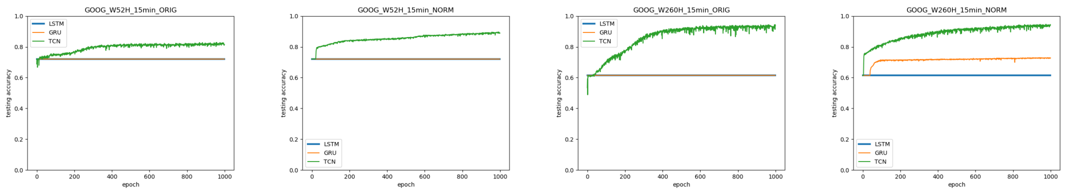
A Accuracy

Testing accuracy results of AAPL, GOOG, and IBM are shown in Figure 8, 9, and 10, respectively.



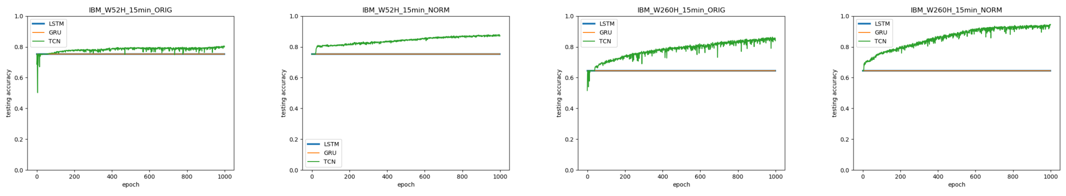
(a) Length=52, ORIG (b) Length=52, NORM (c) Length=260, ORIG (d) Length=260, NORM

Figure 8: Testing accuracy results of AAPL



(a) Length=52, ORIG (b) Length=52, NORM (c) Length=260, ORIG (d) Length=260, NORM

Figure 9: Testing accuracy results of GOOG



(a) Length=52, ORIG (b) Length=52, NORM (c) Length=260, ORIG (d) Length=260, NORM

Figure 10: Testing accuracy results of IBM

B Confusion matrix

Final confusion matrices of AAPL, GOOG, and IBM are shown in Table 6, 7, and 8, respectively.

C Precision, recall and f1-score

Precision, recall and f1-score of AAPL, GOOG, and IBM are shown in Table 9, 10, and 11, respectively.

Table 6: Confusion matrices of AAPL.
(a): Non-normalized data of sequence length 52. **(b)**: Normalized data of sequence length 52. **(c)**: Non-normalized data of sequence length 260. **(d)**: Normalized data of sequence length 260.

		LSTM			GRU			TCN		
		buy	*hold*	*sell*	*buy*	*hold*	*sell*	*buy*	*hold*	*sell*
(a)	*buy*	0	327	0	0	327	0	0	327	0
	hold	0	1427	0	0	1427	0	3	1423	1
	sell	0	326	0	0	326	0	2	323	1
(b)	*buy*	0	327	0	0	327	0	221	106	0
	hold	0	1427	0	0	1427	0	59	1326	42
	sell	0	326	0	0	326	0	1	76	249
(c)	*buy*	0	351	0	0	351	0	1	333	17
	hold	0	1337	0	0	1337	0	7	1294	36
	sell	0	392	0	0	392	0	5	369	18
(d)	*buy*	0	351	0	0	351	0	305	45	1
	hold	0	1337	0	0	1337	0	18	1291	28
	sell	0	392	0	0	392	0	0	13	379

Table 7: Confusion matrices of GOOG.
(a): Non-normalized data of sequence length 52. **(b)**: Normalized data of sequence length 52. **(c)**: Non-normalized data of sequence length 260. **(d)**: Normalized data of sequence length 260.

		LSTM			GRU			TCN		
		buy	*hold*	*sell*	*buy*	*hold*	*sell*	*buy*	*hold*	*sell*
(a)	*buy*	0	299	0	0	299	0	190	100	9
	hold	0	1500	0	0	1500	0	100	1331	69
	sell	0	281	0	0	281	0	11	97	173
(b)	*buy*	0	299	0	0	299	0	209	90	0
	hold	0	1500	0	0	1500	0	40	1437	23
	sell	0	281	0	0	281	0	0	77	204
(c)	*buy*	0	412	0	0	412	0	376	36	0
	hold	0	1276	0	0	1276	0	37	1214	25
	sell	0	392	0	0	392	0	2	30	360
(d)	*buy*	0	412	0	0	408	4	399	13	0
	hold	0	1276	0	0	1226	50	36	1225	15
	sell	0	392	0	0	104	288	0	58	334

Table 8: Confusion matrices of IBM.
(a): Non-normalized data of sequence length 52. **(b)**: Normalized data of sequence length 52. **(c)**: Non-normalized data of sequence length 260. **(d)**: Normalized data of sequence length 260.

		LSTM			GRU			TCN		
		buy	*hold*	*sell*	*buy*	*hold*	*sell*	*buy*	*hold*	*sell*
(a)	*buy*	0	241	0	0	241	0	64	160	17
	hold	0	1565	0	0	1565	0	49	1453	63
	sell	0	274	0	0	274	0	9	114	151
(b)	*buy*	0	241	0	0	241	0	151	90	0
	hold	0	1565	0	0	1565	0	31	1467	67
	sell	0	274	0	0	274	0	0	75	199
(c)	*buy*	0	378	0	0	378	0	254	106	18
	hold	0	1342	0	0	1342	0	54	1252	36
	sell	0	360	0	0	360	0	16	105	239
(d)	*buy*	0	378	0	0	378	0	349	29	0
	hold	0	1342	0	0	1342	0	14	1297	31
	sell	0	360	0	0	360	0	0	40	320

Table 9: Precision, recall and f1-score of AAPL.
(a): Non-normalized data of sequence length 52. **(b)**: Normalized data of sequence length 52. **(c)**: Non-normalized data of sequence length 260. **(d)**: Normalized data of sequence length 260.

		LSTM			GRU			TCN		
		P	R	f1	P	R	f1	P	R	f1
(a)	*buy*	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00
	hold	0.69	1.00	0.81	0.69	1.00	0.81	0.69	1.00	0.81
	sell	0.00	0.00	0.00	0.00	0.00	0.00	0.50	0.00	0.01
(b)	*buy*	0.00	0.00	0.00	0.00	0.00	0.00	0.79	0.68	0.73
	hold	0.69	1.00	0.81	0.69	1.00	0.81	0.88	0.93	0.90
	sell	0.00	0.00	0.00	0.00	0.00	0.00	0.86	0.76	0.81
(c)	*buy*	0.00	0.00	0.00	0.00	0.00	0.00	0.08	0.00	0.01
	hold	0.64	1.00	0.78	0.64	1.00	0.78	0.65	0.97	0.78
	sell	0.00	0.00	0.00	0.00	0.00	0.00	0.25	0.05	0.08
(d)	*buy*	0.00	0.00	0.00	0.00	0.00	0.00	0.94	0.87	0.91
	hold	0.64	1.00	0.78	0.64	1.00	0.78	0.96	0.97	0.96
	sell	0.00	0.00	0.00	0.00	0.00	0.00	0.93	0.97	0.95

Table 10: Precision, recall and f1-score of GOOG.
(a): Non-normalized data of sequence length 52. **(b)**: Normalized data of sequence length 52.
(c): Non-normalized data of sequence length 260. **(d)**: Normalized data of sequence length 260.

		LSTM			GRU			TCN		
		P	R	f1	P	R	f1	P	R	f1
(a)	*buy*	0.00	0.00	0.00	0.00	0.00	0.00	0.63	0.64	0.63
	hold	0.72	1.00	0.84	0.72	1.00	0.84	0.87	0.89	0.88
	sell	0.00	0.00	0.00	0.00	0.00	0.00	0.69	0.62	0.65
(b)	*buy*	0.00	0.00	0.00	0.00	0.00	0.00	0.84	0.70	0.76
	hold	0.72	1.00	0.84	0.72	1.00	0.84	0.90	0.96	0.93
	sell	0.00	0.00	0.00	0.00	0.00	0.00	0.90	0.73	0.80
(c)	*buy*	0.00	0.00	0.00	0.00	0.00	0.00	0.91	0.91	0.91
	hold	0.61	1.00	0.76	0.61	1.00	0.76	0.95	0.95	0.95
	sell	0.00	0.00	0.00	0.00	0.00	0.00	0.94	0.92	0.93
(d)	*buy*	0.00	0.00	0.00	0.00	0.00	0.00	0.92	0.97	0.94
	hold	0.61	1.00	0.76	0.71	0.96	0.81	0.95	0.96	0.95
	sell	0.00	0.00	0.00	0.84	0.73	0.78	0.96	0.85	0.90

Table 11: Precision, recall and f1-score of IBM.
(a): Non-normalized data of sequence length 52. **(b)**: Normalized data of sequence length 52.
(c): Non-normalized data of sequence length 260. **(d)**: Normalized data of sequence length 260.

		LSTM			GRU			TCN		
		P	R	f1	P	R	f1	P	R	f1
(a)	*buy*	0.00	0.00	0.00	0.00	0.00	0.00	0.52	0.27	0.35
	hold	0.75	1.00	0.86	0.75	1.00	0.86	0.84	0.93	0.88
	sell	0.00	0.00	0.00	0.00	0.00	0.00	0.65	0.55	0.60
(b)	*buy*	0.00	0.00	0.00	0.00	0.00	0.00	0.83	0.63	0.71
	hold	0.75	1.00	0.86	0.75	1.00	0.86	0.90	0.94	0.92
	sell	0.00	0.00	0.00	0.00	0.00	0.00	0.75	0.73	0.74
(c)	*buy*	0.00	0.00	0.00	0.00	0.00	0.00	0.78	0.67	0.72
	hold	0.65	1.00	0.78	0.65	1.00	0.78	0.86	0.93	0.89
	sell	0.00	0.00	0.00	0.00	0.00	0.00	0.82	0.66	0.73
(d)	*buy*	0.00	0.00	0.00	0.00	0.00	0.00	0.96	0.92	0.94
	hold	0.65	1.00	0.78	0.65	1.00	0.78	0.95	0.97	0.96
	sell	0.00	0.00	0.00	0.00	0.00	0.00	0.91	0.89	0.90