Application of Wavenet to financial times series prediction

Author: Endre Moen

Supervisor: Hans Karlsen

Master's Thesis in statistics
Department of Mathematics
University of Bergen

February 15, 2024

Acknowledgements

I would like to express my gratitude to Hans Karlsen, my thesis advisor, for patiently listening and good advice during this thesis. I would like to thank Kristine Lysnes for helping me register for this 30 points master thesis 3 times, and for helping me process an application for extension after a sick leave.

Additionally, I would like to acknowledge the use of ChatGPT, a language model developed by OpenAI, which played a valuable role in assisting me with coding, generating and refining content during the course of this thesis.

Endre Moen
Bergen, 15/02/2024

Abstract

This thesis explores the application of the WaveNet model utilizing dilated causal convolutions, originally designed for text-to-speech synthesis on univariate time series. Here it is adapted to predicting on multivariate financial time series focusing on 43 commodities consisting of currencies, bonds, indexes, soft commodities, metal, grains, energy and live stock. The study trains, and evaluates the models on commodities from *Pinnacle data corp's CLC database*, with the loss function of mean squared error and model performance is evaluated on a scorecard of metrics like accuracy, edge, noise, and calibration ratio. The results reveal varying performance across commodities, with heating oil demonstrating the highest edge of 0.005 and accuracy of 53% and rbob gasoline having the second highest edge with 0.001 and accuracy of 52.2%. Outcomes demonstrate that around half the models had a higher than 50% accuracy in predicting the daily movement and about half, below. New insight into adapting WaveNet to multivariate time-series prediction is found, and discussion highlights the potential of hyper-parameter tuning and suggest further exploration of advanced models and alternative data-sets for enhanced predictions. The findings underscore further investigation into deep learning in financial forecasting, with implications for trading strategies and future model refinement, but does not contradicts the efficient market hypothesis. A comparison is made on the results of one commodity, Sugar, to a GARCH(1,1) model.

Keywords: artificial intelligence, financial time series, time-series momentum, WaveNet, dilated causal convolutions, predictive modeling, deep learning, hyperparameter tuning, efficient market hypothesis

Contents

List of Figures

Chapter 1

Introduction

1.1

1.2 Motivation

Artificial intelligence (AI) is increasingly improving the accuracy of time series predictions in financial competitions like the *M6 Financial Forecasting Competition*. In the autumn 2021, after completing the course in time-series (stat211), I participated in another competition, Kaggle's *Optiver Realized Volatility Prediction*. Optiver is a market maker, and continuously quoting bid and ask prices, thereby providing liquidity to the market. Implied volatility is a key input in options-pricing models like Black-Scholes (*Black and Scholes*, 1973). Having a good estimate of realized volatility Optiver can calibrate their pricing models more accurately. Increasing profit and reducing losses. E.g higher implied volatility generally leads to higher option prices. This was the motivation for Optiver to create the competition. Together with another student we trained 3 machine learning models, a small multilayer perceptron (MLP) of only 5 layers, TabNet, and Light Gradient boosting. With an ensemble of these models we managed to get a top 2% result. This motivated me to define this master thesis.

ML is becoming more prevalent in the financial industry. Companies are using insights gained from these competitions in their day-to-day business like Optiver. This trend is further supported by the increasing computational power of modern graphical processing units (GPUs), and new breakthroughs in machine learning, which again is making these AI models increasingly more capable, and accessible. The initial motivation for this project was to try and reproduce the results published in the article (*Lim et al.*, 2019) which investigated four models on a momentum trading strategy. The models were Multi layer perceptron (MLP) (*Werbos*, 1982), Lasso regression (*Tibshirani*, 1996), Long-short term memory (LSTM) (*Hochreiter and Schmidhuber*, 1997), and WaveNet (*Oord et al.*, 2016), on 88 instruments on the *Pinnacle data corp's CLC database (Continuously Linked Commodity Contracts)* (Pinnacle Data Corp. CLC Database). The work on WaveNet got very involved, and the implementation of WaveNet was not made available or described in the article. Therefore this thesis is about WaveNet only. The findings include new insight on how to apply WaveNet to multivariate time series, and encouraging results on the prediction on applied time series.

1.3 Objective

In this thesis I will investigate how well the WaveNet model (*Oord et al.*, 2016) can predict financial time series. This is a pointless exercise according to the efficient market hypothesis, which states in some form (weak, semi-weak or strong form) that prices in an efficient market incorporate all available information, and shares some conceptual similarities with the idea of a Martingale (*Ville*, 1939).

WaveNet is a relatively mature generative convolutional neural network (CNN) (*LeCun et al.*, 1989) for time series. The article was published by Google's Deep Mind in 2016, for text-to-speech synthesis from raw audio wave-forms, using supervised learning. In this thesis it will be used to learn to predict the price change for a set of 43 instruments, with mean squared error (MSE) as the loss function, and evaluated on a set of metrics like accuracy, edge and noise. The model will be evaluated on a scorecard of these metrics on each model. The instruments are commodities purchased from *Pinnacle data corp's CLC database*. The Continuously Linked Commodity Contracts (CLC) database covers the 98 most popular commodities and includes the most active contracts. This data set is prepared in order to back-test future prices for the most liquid contracts. The CLC database then links them together. The contracts are linked because the contracts has an expiry date. The instruments include commodities like *Australian dollar, British Punds, Coffe, Cotton, Platinum*, and *Crude oil*, and the data set contains prices at *Open, High, Low, close, Open Interest, Total Volume*, and *Total Open Interest* as well as the *Fed fund rate*. The first instrument *Live Hogs*, started trading in 1970, and the last instrument included, *Canadian 10yr Bond*, started trading in 1994. The last trading date is 27.01.2023.

The list of 43 instruments, from the list of 98 instruments, was selected based on having the least missing data, and being traded on the last trading date. They are shown in Table 1.1.

Since the model use multivariate time series we define the random variables of any time series in terms of data at time t as:

$\mathbf{X}_t = (X_{t,1}, \ldots, X_{t,m})$

and we define the random variables to predict, daily returns, in terms of data at time t as:

$\mathbf{R}_t = (R_{t,1}, \ldots, R_{t,m})$

where $m = 43$ so that $R_{-,1}$ =AN, $R_{-,2}$ =BN and so on.

1.4 Time series fundamentals

Some fundamental time-series concepts are introduced to explain the pre-processing, and the model architecture. The pre-processing steps are data cleaning and lagged feature generation into exogenous time series.

Auto correlation: is the degree of correlation of the same variables between two successive time intervals. It measures how the lagged version of the value of a variable is related to the original version of it in a time series.

The covariance function:

$$\gamma(r,s) = \mathbb{E}[(X_r - \mu(r))(X_s - \mu(s))] = \mathrm{Cov}(X_r, X_s) \tag{1.1}$$

Symbol	Market Name	Symbol	Market Name
AN	AUSTRALIAN $$, composite	CN	CANADIAN $$, composite
BN	BRITISH POUND, composite	CT	COTTON #2
CC	COCOA	DT	EURO BOND (BUND)
DX	US DOLLAR INDEX	FB	T-NOTE, 5yr composite
FN	EURO, composite	GI	GOLDMAN SAKS C. I.
GS	GILT, LONG BOND	JN	JAPANESE YEN, composite
JO	ORANGE JUICE	KC	COFFEE
KW	WHEAT, KC	LB	LUMBER
LX	FTSE 100 INDEX	MD	S&P 400 (Mini electronic)
MW	WHEAT, MINN	NK	NIKKEI INDEX
SB	SUGAR #11	SN	SWISS FRANC, composite
TY	T-NOTE, 10yr composite	US	T-BONDS, composite
ZA	PALLADIUM, electronic	ZB	RBOB, Electronic
ZC	CORN, Electronic	ZF	FEEDER CATTLE, Electronic
ZG	GOLD, Electronic	ZH	HEATING OIL, electronic
ZI	SILVER, Electronic	ZK	COPPER, electronic
ZL	SOYBEAN OIL, Electronic	ZM	SOYBEAN MEAL, Electronic
ZN	NATURAL GAS, electronic	ZO	OATS, Electronic
ZP	PLATINUM, electronic	ZR	ROUGH RICE, Electronic
ZS	SOYBEANS, Electronic	ZT	LIVE CATTLE, Electronic
ZU	CRUDE OIL, Electronic	ZW	WHEAT, Electronic
ZZ	LEAN HOGS, Electronic		

Table 1.1: 43 selected commodities to predict from the Pinnacle CLC database

where $\{X(t)\}$ is a time series.

The covariance function defined with only one parameter $\gamma(h) = \gamma(X_{t+h}, X_t)$. The stationary auto correlation function is the normalised covariance function:

$$\rho(h) = \frac{\gamma(h)}{\gamma(0)} \tag{1.2}$$

assuming $\{X(t)\}$ is stationary.

Stationarity: A time series is said to be stationary when its statistical properties, such as mean, variance, and auto correlation, remains constant over time. Stationarity is essential for many time series modeling techniques because it simplifies the analysis. If a time series is not stationary, it may exhibit trends or seasonality, which can make predictions challenging. Stationarity can often be achieved through differencing, which removes a trend.

Autoregressive: Is a model where the value of a variable at the next time step is a linear combination of its past values, and white noise which cannot be modelled. E.g AR(2) which describes the next value as a linear combination of its past 2 values. Autoregressive models are widely used for modeling and forecasting time series data, such as stock prices, weather patterns, or any data that evolves over time.

Definition:

$$X_t = c + \phi_1 X_{t-1} + \phi_2 X_{t-2} + \ldots + \phi_p X_{t-p} + \varepsilon_t \tag{1.3}$$

where

- X_t represents the value of the time series at time t
- c is a constant term.
- $\phi_1, \phi_2, \ldots \phi_p$ are the autoregressive coefficients, which determine the influence of past values on the current value. These coefficients are estimated from the data.
- $X_1, X_2, \ldots X_p$ are the past values of the time series, lagged by $1, 2, \ldots p$ time units. p is the models window size.
- ε_t is a random error term, which represents the noise or unpredictability in the data at time t

The **Simple Moving Average** (SMA) of a time series x_t with a window size of n is defined as:

$$\mathrm{SMA}_t = \frac{1}{n} \sum_{i=t-n+1}^{t} X_i$$

and it identifies the time series trend.

The **Exponential Moving Average** (EMA) of a time series X_t with a smoothing factor α and a window size of n is defined as:

$$\mathrm{EMA}_t(n) = \alpha \cdot X_t + (1-\alpha) \cdot \mathrm{EMA}_{\mathrm{t}-1}(\mathrm{n}-1)$$

where α is the smoothing factor between 0 and 1. EMA identifies trends, as well as dynamic support or resistance. EMA(n) can also be expressed as an infinite sum

$EMA_t(n) = \sum_{i=0}^{n}(1-\alpha)^i \alpha X_{t-i}$. In this project a truncated smoothing factor of $n = 60$ is used.

The **Moving Average Convergence Divergence**, $\text{MACD}_t(8,24)$, is defined as the difference between the 8-day Exponential Moving Average ($\text{EMA}_t(8)$) and the 24-day Exponential Moving Average ($\text{EMA}_t(24)$):

$$\text{MACD}_t(8,24) = \text{EMA}_t(8) - \text{EMA}_t(24)$$

and it can be used for identifying potential trend reversal or convergence. Here $\text{EMA}_t(24)$ means that the window size is 24.

The conditional **variance** today, σ_t^2, is another feature generated using EMA:

$$\sigma_t^2 = \lambda \sigma_{t-1}^2 + (1 - \lambda R_{t-1}^2) \tag{1.4}$$

where

- σ_t^2 is variance today
- σ_{t-1}^2 is variance yesterday
- λ is the weight, the smoothing parameter
- μ_{t-1}^2 squared return yesterday

which is a Garch(1, 1) where $\alpha_0 = 1, \beta = -\alpha = \lambda$ model.

Volatility is defined by σ_t. Using EMA we say that the volatility today is more important than yesterday, and the volatility is biased toward more recent data. Volatility can be useful for identifying current market risk.

Concept drift: occurs when the statistical properties of the data change, and the model that was trained on historical data may become less accurate over time. Trend-related concept drift implies a shift in the long-term pattern of the time series.

Trend: represents a long-term increase or decrease in the data points over time. It reflects the underlying direction or pattern that is not due to seasonal or short-term fluctuations.

Trend as a concept drift: Trends can be seen as a form of concept drift, because they reflect a shift in the underlying patterns of the time series. Identifying and modeling trends is crucial in time series analysis, because they can significantly impact future predictions. For example, if a model is trained during a period of upward trend, and there is a subsequent shift to a downward trend, the model may make less accurate predictions.

Time series split: When working with time series data, it's essential to use a proper time series split for model evaluation. Unlike random data splits, commonly used in machine learning, time series data should be split sequentially to simulate real-world forecasting scenarios. This means that earlier data is used for training, and later data is used for testing, to avoid data leakage and ensure that the model is tested on unseen future data. Overfitting can occur if this sequential splitting is not followed.

Inflation: Inflation is the rate at which the general level of prices for goods and services rises, leading to a decrease in the purchasing power of a currency. Inflation is typically measured using indices like the Consumer Price Index (CPI) or Producer Price

Index (PPI). Inflation rates can vary significantly from year to year due to economic factors, policy changes, and other factors.

Supervised learning: Supervised learning is a machine learning paradigm where a model is trained on a labeled data set, meaning that it learns to make predictions or classifications based on input-output pairs provided during training. In the context of time series forecasting, supervised learning involves using historical data to predict future values or events, treating the time series as a sequence prediction problem.

Causality and the arrow of time: When working with time series data, it's important not to violate causality, which means not assuming that past events are caused by future events. The arrow of time signifies that causality flows from the past to the future. Models and analyses should respect this principle to avoid making unrealistic assumptions or predictions.

Aggregating for long-term predictions: Time series models often perform better when predicting short time periods rather than long ones. Aggregating data, such as taking monthly averages, can help capture long-term patterns more effectively. This reduces the noise associated with daily data and focuses on the broader trends, making long-term predictions more accurate and stable.

1.5 History is a filtration

A filtration is a sequence of sigma-algebras that represent the information available at different points in time within a stochastic process, $\{X_t, t \in T\}$. In other words, it's a way to formalize the concept of how information becomes available over time.

The history is the collection of all past observations or data points in the time series up to a specific point in time. It represents the information available at that particular moment. As time progresses, a model has access to more and more information, and this information is represented by the sigma-algebras in the filtration.

If H_t represents the history of observations up to time t, and F_t represents the sigma-algebra of information available at time t, then filtration means that H_t is a sub-sigma-algebra of F_t, or more precisely, $H_t \subseteq F_t$. In other words, the history contains all the information available up to that point in time.

This concept is important in understanding how information flows and accumulates in time series data, and is often used in the context of stochastic processes and mathematical modeling to describe how past observations affect future predictions and the evolution of the system. It's a fundamental idea in time series analysis, particularly in the field of stochastic calculus and mathematical finance, where it's used to model and analyze financial time series data.

If the stochastic process is a fair game then it is a martingale (*Ville*, 1939) and the long term expected return of the fortune of the gambler is constant. This is what is referred to as the efficient market hypothesis (*Bachelier*, 1900).

A stochastic process $\{X_t, \mathcal{F}_t, t \in T\}$ is a **martingale** if:

(i) The family of sigma-algebras $\{\mathcal{F}_t\ , t \in T\}$ is a filtration.

(ii) The process $X_t, t \in T$ is adapted to the filtration $\{\mathcal{F}_t\}$. X_t is $\mathcal{F}_t$-measureable.

(iii) For each $t \in T$, X_t is integrable, i.e., $\mathbb{E}|X_t| < \infty$.

(iv) For each $s < t \in T$, the Martingale property holds: $X_s = \mathbb{E}[X_t|\mathcal{F}_s]$

1.5.1 Convolutional neural network

CNNs are most commonly used to classify images, and had its breakthrough with the introduction of AlexNet by *Krizhevsky et al.* (2012) which pushed the state-of-the-art performance by significantly reducing the top-5 error rate on the ImageNet benchmark (*Deng et al.*, 2009). However, CNNs can equally well be used to do regression on images *Moen et al.* (2018, 2021, 2023), or classify and regress on 1D datasets like financial time series or speech.

Convolutional **filters**, also called kernels, are designed to detect specific patterns or features in the input data, and may be the input to the first layer like an image or in this case a time series. The output becomes the input to the first hidden layer, and the output of that layer becomes the input to the next layer and so on. The output of a convolutional layer are the result of applying learnable filters (or kernels), having small widths and heights and the same depth as that of the input volume. When applying multiple filters the depth increases. With images the first layer typically has depth three, from the colors red, green and blue. Applying a convolutional filter of size 2 x 2 on a 4 x 4 input matrix is illustrated in Figure 1.1.

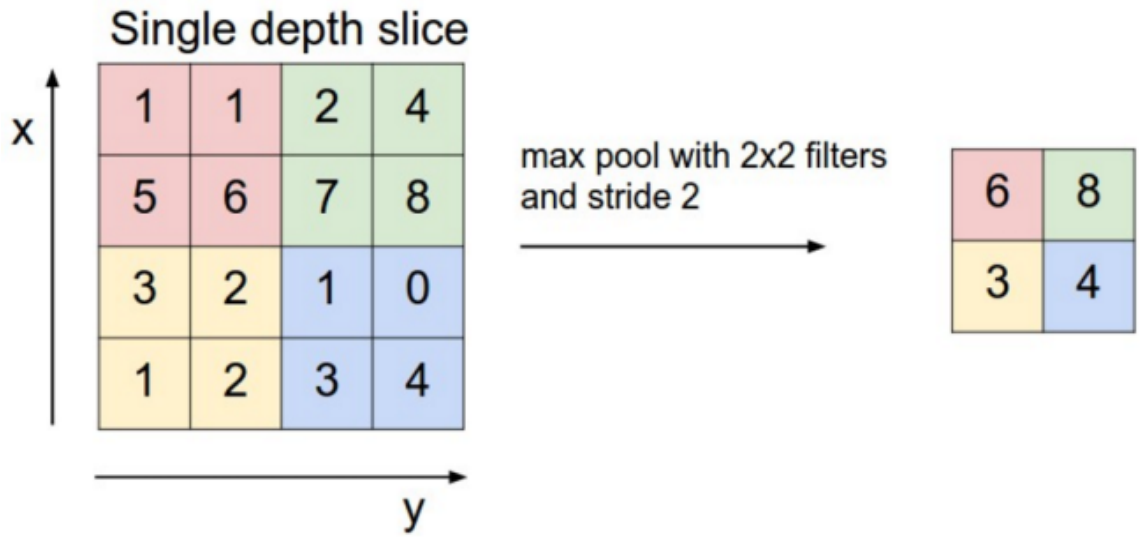

Figure 1.1: A 4x4 image with a 2x2 max pooling filter applied and a stride of 2. Source: https://kuse-manohar.info/2016/12/30/convolutional-networks/

A **stride** describes how much a filter is sliding over the input data during the forward-propagation to produce an output to the next layer.

Padding is done to preserve the spatial dimensions of the input layer before applying the filter. Padding involves adding extra pixels around the border of the input feature map before convolution, if it is an image, or before and after a sequence if it is a time series. **Valid padding** reduces the dimension of the feature map and **same padding** preserves the spatial dimensions of the feature maps.

1.5.2 The WaveNet architecture

The WaveNet architecture, introduced by *Oord et al.* (2016), is designed for generative tasks, such as generating realistic audio waveforms from text. It's a convolutional neural network used to predict sequential data. CNNs are popular for classification and regression on images, due to its spatial structure and hierarchical patterns, which reduces the number of parameters in the model by having translational invariance, which means it can recognize patterns regardless of position in the image. WaveNet uses 1-dimensional convolutions as opposed to 2-dimensional convolutions used on images.

CNNs also have a hierarchical architecture with multiple convolutional layers. Each layer captures increasingly complex and abstract features. WaveNet uses a particular hierarchical architecture called dilated, causal-convolution. To model long term relationships the input layers are connected with **dilated causal convolutions**, see Figure 1.2. **Dilated convolutions** is a convolution where the filter or kernel is applied over an area larger than its length by skipping input values with a certain step. The length skipped is described by the dilation rate. This increases the receptive field of the model, and allows the receptive field of the network to grow exponentially with the number of dilated layers.

The **causal convolution** describes that there is no information processed from the future into predicting the past. The causal convolution means that the filter applied on x_1, x_2 can only be applied to predicting y_3 and greater.

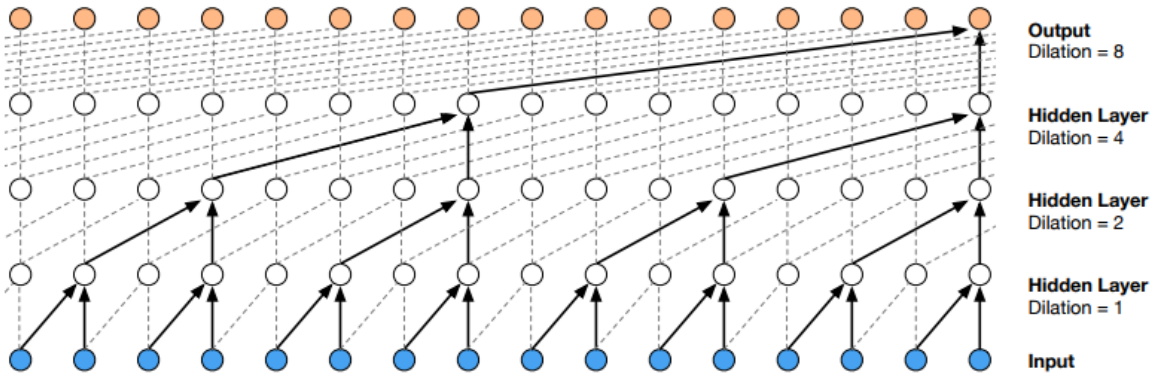

Figure 1.2: Causal, dilated convonlutions with filter size of 2, 4 dilated layers, connected in a causal way, and a depth of 5. Source: (Oord et al., 2016)

Each node in a layer is connected with a residual block to the next layer. The residual block takes the previous layer's output as its input. An output is generated by applying a convolution, and then applying two activation functions on the output, $sigmoid(x)$ and $tanh(x)$. The final activation is given by multiplying the two. A new convolution with kernel size of 1, same padding of 1, and filters are applied to the output. Finally a residual connection is made for the layer. This is normal for deep neural networks and is a regularization technique. This is done by adding the input and output as the new output. If the layer learns the weights of $\mathbf{0}$ then that is the same as a skip connection since $\mathbf{0}$ is the identity element on addition. See Figure 1.3 for an illustration of the composition of each node.

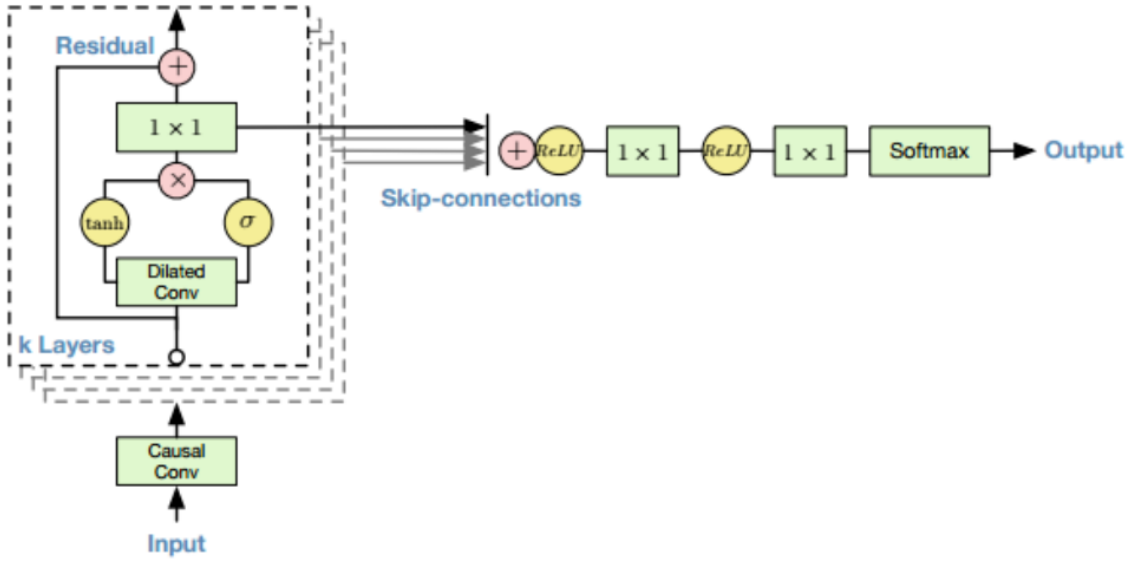

Figure 1.3: A detailed illustration of one hidden layer in WaveNet. It has 3 convolutions, two activation functions and a residual connection. The final output layer is not used in this project. Source: (Oord et al., 2016)

CNNs benefits from being able to be accelerated on a GPU. In *Oord et al.* (2016) the model was trained on a large set of high frequency speech in real-time. Several preprocessing steps were applied for this use case. In this project there is neither a requirement for high processing speed as one time steps consists of a days trading, nor are there compression or sampling preprocessing steps needed to aggregate the data in real time.

1.5.3 Receptive field

In a WaveNet-like architecture, the sequence length or receptive field required to convolve over all dilations depends on the maximum dilation rate used in the network. The receptive field size increases exponentially with the depth of the network. To calculate the required receptive field, we must consider the maximum dilation rate in the model.

WaveNet typically uses dilation rates that are powers of 2, because the filter width is 2 (e.g., 1, 2, 4, 8, etc.). In a network with 10 dilation and input layers, the dilation rate of each layer ranges from 1 to 512 (2^0 to 2^9). The required input data sequence length needed to saturate the receptive field of the model is then 512.

The formula to calculate the required total receptive field (RF_{total}) of the entire network with a depth of D:

$$\mathrm{RF}_{total} = 2^{D-1} \tag{1.5}$$

assuming a filter width of 2 and a stride of 1. This number is required when preparing the data into windows.

1.6 Outliers

In statistical analyses, outliers may exert an undue influence on the results, making the results unreliable. Financial data are notoriously subject to outliers. Commonly, empirical asset pricing researchers usually take an ad hoc approach when dealing with outliers.

One such technique is winsorization. It is performed by setting the values of a variable X_n with n observations, that are in the top h percent of all values of X to the $(100h)$th percentile of X. Similarly, values of X in the bottom l percent of X values are set to the l-th percentile of X.

Winsorization of outliers was not performed in the final training of the models, but has been experimented with using the 99-percentile and 1-percentile clipping.

The data transformation can be represented as:

$$X_t' = \min\left(\max(X_t, q_{0.01}), q_{0.99}\right) \tag{1.6}$$

where $q : [0,1] \to \mathbb{R}$ is the quantile function that maps its input p to a threshold value x so that the probability of X being less or equal to x is p.

Chapter 2

Methods

2.1 Feature engineering

Creating information rich features from the raw time series data was the first step taken in creating a model to predict the movement of the instrument prices. Lag features, moving averages, technical indicators and other transformations were created. The features derived were from the Github repository *Lamberti* (2020), and implemented parts of the article *Lim et al.* (2019). The raw data for each instrument contained seven time series. These were:

- **Date**
- **Open** - first trade of the day
- **High** - highest price during the day
- **Low** - lowest price during the day
- **Close** - Settle price established by the exchange
- **Volume** - total contract volume
- **Open Interest** - contract open interest. Total number of contracts held by market participants

The Federal fund rates were provided in another file:

- **ShortRate**$_{\text{Daily}}$ - daily federal funds rate
- **ShortRate**$_{\text{Annual}}$ - annual federal funds rate

Lagged features were generated:

- **ReturnsDaily** - Percentage change of *Close* prices, R_t
- **NextReturnsDaily** - Shifted back 1 day of ReturnsDaily. This is the time series to predict! It holds the next days return
- **Sigma** - Standard deviation of $\text{EMA}_t(60)$ of daily returns

- **NormReturns**$_{\text{Daily}}$ - Daily returns normalized by Sigma
- **NormReturns**$_{\text{Monthly}}$ - Montly returns normalized by Sigma
- **NormReturns**$_{\text{Quarterly}}$ - Quarterly returns normalized by Sigma
- **NormReturns**$_{\text{Semiannually}}$ - Semi-annual returns normalized by Sigma
- **NormReturns**$_{\text{Annually}}$ - Annual returns normalized by Sigma
- **MACD**$_t(8,24)$ - MACD of close price with trend estimate of short timescales 8, $\text{EMA}_\text{t}(8)$, and long time-scale 24, $\text{EMA}_\text{t}(24)$
- **MACD**$_t(16,48)$ - MACD of close price with short timescales 16 and long time-scale 48
- **MACD**$_t(32,96)$ - MACD of close price with short timescales 32 and long time-scale 96
- **Binary_MACD**$_t(8,24)$ - True if short time scale is larger, $\text{EMA}_\text{t}(8) > \text{EMA}_\text{t}(24)$
- **Binary_MACD**$_t(16,48)$ - True if $\text{EMA}_\text{t}(16) > \text{EMA}_\text{t}(48)$
- **Binary_MACD**$_t(32,96)$ - True if $\text{EMA}_\text{t}(32) > \text{EMA}_\text{t}(96)$
- **Sigma**$_{\text{Norm}}$ - Log of Sigma divided by the mean of $\text{EMA}_\text{t}(181)$ of Sigma.
- **ReturnsWeekly** - calculates the rolling 5 day return, by adding 1 to the past 5 days return, taking the product and subtracting 1.
- **ExcessReturns**$_{\text{Daily}}$ - The difference between daily returns and short rate daily

A set of 25 time series was recorded for each instrument, with 24 of these time series serving as features for prediction. The specific time series targeted for prediction was NextReturnsDaily which is the 9th feature in the code. We have defined it as R_t but we can also define it as $X_{t,i,9}$ as the *9*th feature, which is the return of the *i*'th asset at time t

The list of features were selected based on the implementation of (*Lim et al.*, 2019) with source available at https://github.com/maxlamberti/time-series-momentum.

Formalizing the objective, we consider the random variable $X_{t,i,j}$ where $i = 1,\ldots,m$, in this case $m = 43$, the number of instruments. $j = 1,\ldots,k$, and in this case, $K = 25$ time series pr instrument. The goal is to predict $X_{t+1,i,9}$ such that

$$\hat{X}_{t+1,i,9} = \mathbb{E}\left(X_{t+1,i,9} \mid \bigvee_{i=1}^{m} \mathcal{F}_t^i\right) \quad \text{for } i = 1,\ldots,m.$$

where $\mathcal{F}$ is the filtration, and $\{\mathcal{F}_t\ , t \in T\}$ is a discrete time sub-sigma-algebras from the sigma-algebra of continuous time.

2.2 Data cleaning and transformation

Financial time series are intermittent and lumpy with missing data on holidays, weekends, special market closures, trading halts, regulatory actions, technical issues or other events that may stop one or more instruments from trading.

The dates with missing data has been forward filled:

```
data[asset].ffill(inplace=True)
```

In Figure 2.1 shows graphs of close price, volume, sigma squared (the variance or volatility squared) and daily price change of close price for commodity sugar (SB) after data cleaning.

Prior to creating lagged features, the input features (the first 7 features) have been transformed. This has been done using min-max normalization to set the values to between 0 and 1.

```
scaler = MinMaxScaler()
scaled = scaler
    .fit_transform(
      df[[
        'Open', 'High', 'Low', 'close', 'Volume', 'Open_Interest'
      ]]
      .values
     )
```

MinMaxScalar transforms the data with:

```
X_range = (X - X.min(axis=0)) / (X.max(axis=0) - X.min(axis=0))
```

where *min* and *max* is given by the feature range of each series. MinMaxScalar is an alternative to zero mean, unit variance scaling. MinMaxScaling is more sensitive to outliers but preserves the distribution and the values are more intuitive. They are simply scaled down.

2.3 Data windowing

To train the model, the data was transformed from a matrix of multiple time series to a 3 dimensional tensor in the format of *(batch size, window length, feature length)*, where *batch size* is the dimension used for one training step of the model (forward- and backwards-propagation). The window length is the number of time steps the model sees to base its prediction on, and the feature length is the number of time series. Initially WaveNet was developed for speech, which uses a univariate time series, and it predicts multiple time steps. While this project predicts one time step from a multivariate time series. Note: After the first hidden layer the dimension changes to (batch size, window length, number of filters), and it does no longer make sense to talk about multiple time series but output from applying multiple filters. The model is time independent of a batch of steps as the weights are equally updated by the error residuals. While,

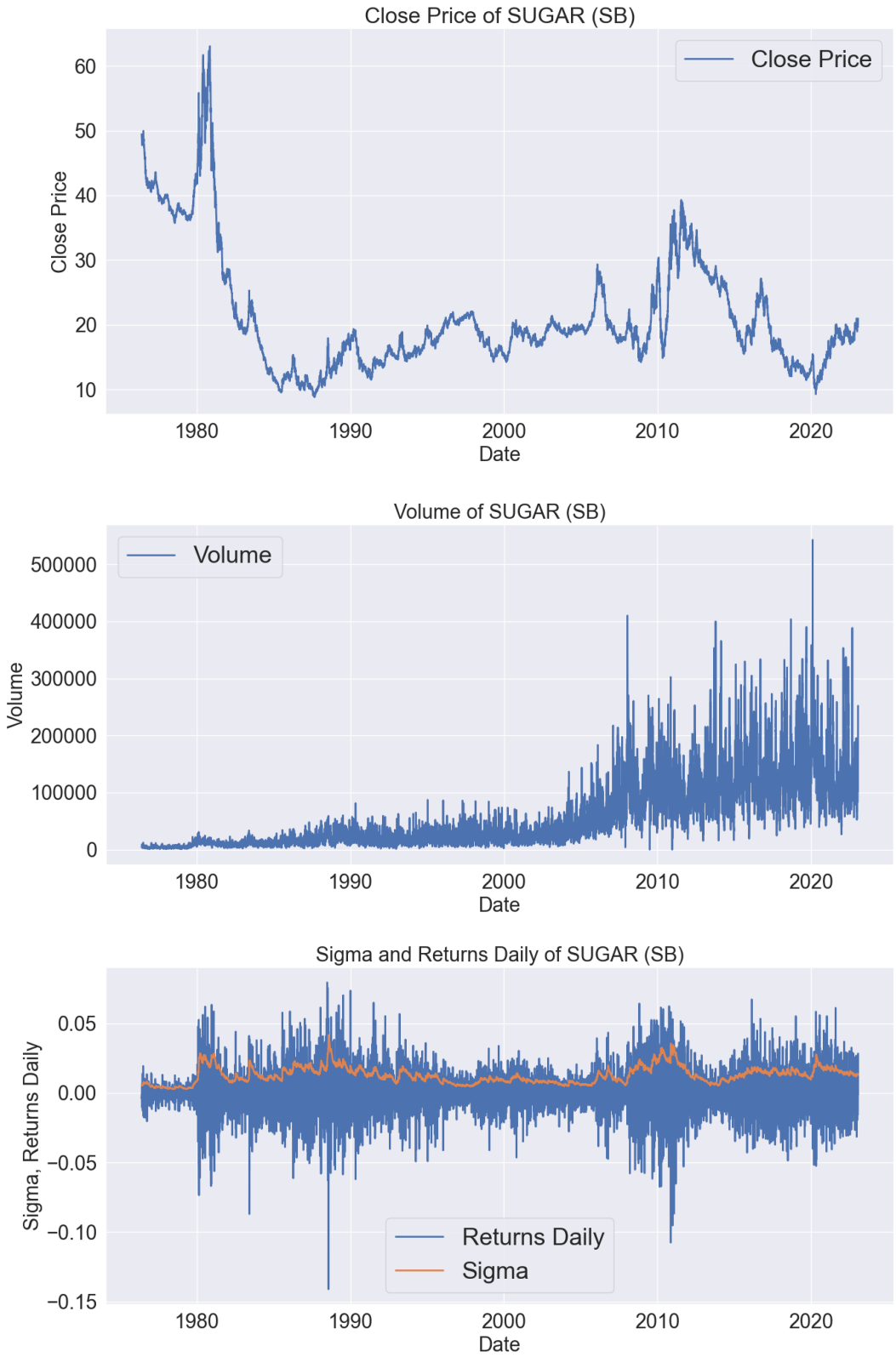

Figure 2.1: Close price, Volume, and daily return with sigma (volatility^2) of sugar (SB) contracts plotted from the start of the time series.

the order the batches, and the model is trained, as the weights depend on the order of updates.

A **horizon** is the number of time steps into the future we want to predict, and a **window size** is the number of time steps we're going to use to predict the horizon. The window size corresponds to the order-p of a autoregressive model. The window size must be calculated from the depth of the WaveNet, and is given by $filter - width^{numLayers}$ which in this case is $2^8 = 256$. Since we're predicting 1 time step, the horizon is 1. There are two approaches to preparing data to be forecast by neural networks. A sliding window or an expanding window. A sliding window uses a fixed width window and predicts the horizon outside the window. The expanding window uses a larger and larger window length of training data to predict the horizon.

A fixed window size was chosen because the receptive field of a WaveNet model is defined by the depth of the WaveNet given by the number of hidden layers. Therefore it will not increase the available information if the window size increase past the networks receptive field, also having a window smaller than the receptive field means the model could be made smaller.

Data windowing is done with:

```
def create_window(df1):
    window_size = 255
    shift = 1
    num_windows = (df1.shape[0] - window_size) // shift + 1
    result = np.empty((num_windows, window_size, df1.shape[1]))
    for i in range(num_windows):
        result[i] = df1[i * shift:i * shift + window_size]
    return result
```

2.4 Building WaveNet for multivariate time series

The architecture can be altered in six different ways to make predictions. It can be defined to predict one or multiple time steps from one time series, or from two or more time series with exogenous time series. The architecture can also predict one or multiple time steps for multiple time series. This architecture is build to predict one time step from the input of multiple time series.

There are two changes made to the original WaveNet. For the model to allow for multiple time series as input, the first layer does not have a residual connection. After the first layer in the CNN, the number of time series becomes the number of filters defined for the network. A skip connection, which is implemented as an add operation of the input to a layer and the output of the layer, has different shape in the last dimension. This is because the number of features becomes the number of filters. When the input layer has 25 time series, and the number of features is 256, there is a dimension mismatch. In the original use of WaveNet, the input was only of 1 time series. Tensorflow (*Abadi et al.*, 2016) handles this gracefully (via Numpy) by broadcasting the 1 dimension to 256. However, when there are more dimensions, Tensorflow (using Numpy) doesn't know how to do this. The solution for allowing WaveNet to work for multiple exogenous time series, is to not implement the residual connection on the first

layer. This resolves the problem because the input and output of the second layer is then the same. For a deep network, say of depth 15, the difference between having only 14 residual connection vs 15 is minor, as the residual connections work to regularize, allowing networks to be deeper (*He et al.*, 2016).

The second change is to add 2 hidden layers with MLP because the output of the last CNN layer has 10 000's of parameters. To make a prediction from this will be a too large data compression. To compress the information more gradually, the two last layers are MLP's with 256 and 64 fully connected nodes, and then a linear output of 1 parameter (plus bias) predicts the next time step. See Figure 2.2 for the new architecture.

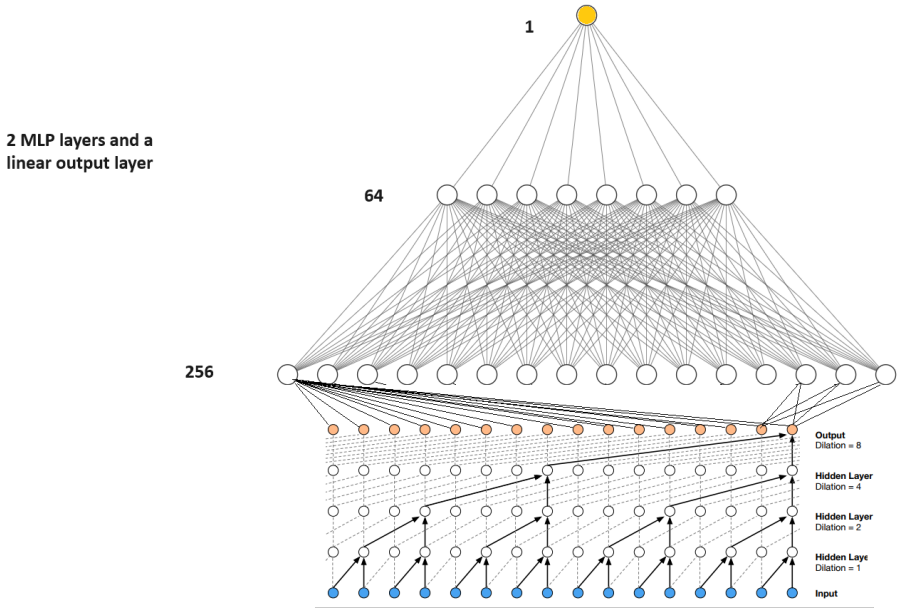

Figure 2.2: WaveNet with 2 MLP layers and a linear output.

The model has a total of 9.798.017 parameters. There are also ca about 10 million data points in the data set containing both training, validation and test set. The model has 8 dilated causal convolutional layers and a filter width of 2. In Tensorflow the causal convolution is implemented with `padding = 'causal'`. The receptive field from 8 convolutions is 256. The batch size is 32 and there are a total of 25 time series, of which 24 are exogenous. So the input of a training batch has dimension (32, 256, 25). The last time step is predicted. After the first convolution, this changes to (32, 256, 256) because there are a total of 256 filters learned. A model summary is available in supplementary information4.1.5.

Convolutional filters with learnable weights are applied from left to right, from past values to new values and across the input series, but the batches can be in random order or in sequential order. It does not matter because the model is trained only on the data window. Filter of width 2 was chosen (and height 1). A stride of 1 was chosen, which means the filters are applied to overlapping regions of the series. No padding was applied.

In the implementation the dilation rate is 128, and the required minimum input is $2^8 = 256$. This means a sequence length of at least 256 data points are needed to ensure that all dilation convolutions in the WaveNet model has access to past context

when making predictions. The defined window uses 256 data points. Full source code and implementation of the model can be found in 4.1.5.

Source code for the model definition can be found in supplementary information.

2.4.1 Hyperparameter tuning

Little effort has been made to tuning hyperparameters, therefore it is likely possible to improve the performance of the models by exploring the performance of different hyper parameters. A standard learning rate is 0.001, and the range of percentage change in the closing price of an instrument was in the order of 0.01. Therefore a learning rate of 10^{-5} was chosen. Also a relatively large batch size of 96 was chosen because time series data does not occupy as much memory on the GPU as images which typically have a batch size of between 8 and 32 on a consumer-end GPU. Training was done for 100 epochs with a patience of 20 (validation loss does not decrease in 20 epochs). The Adam gradient descent algorithm *Kingma and Ba* (2017) was used.

2.5 Training and Time Series Cross Validation

In general, we cannot randomly split time series models into train, validation and test data sets. That will allow the model to learn data points from the future. Financial time series splits are based on walk forward training, which is a way of back testing your model. The model is trained on past values, and tested on newer data by dividing the historical data into multiple chronological parts.

To saturate the parameters and because transfer learning is a thing also for deep learning on time-series (*Fawaz et al.*, 2018), the model was first trained on all the training sets of all instruments, expect a small subset of instruments (CT and MW) due to divergence of training. Then fine-tuned to the training set of the model to predict.

Each data set was split into training, validation and a test set. The last date for all the 43 instruments was 26. January 2023, and the last 1430 to the last 715 time steps were used for validation. The last 715 time steps were used for testing. For one of the smallest time series, like the Nikkei Index (NK), this corresponds to 80% training, 10% validation and 10% test set. By selecting a fixed point in time for validation- and test-sets, this guaranteed that no time series contained training data past any other. This was necessary to prevent data leakages, since the model was trained on all the 43 time series, before being fine-tuned on the time series to predict. Figure 2.3 shows an example of a train-, validate- and test-set split for COCOA.

The implementation was done using Tensorflow (*Abadi et al.*, 2016) and Keras (*Chollet and others*, 2018) software packages in Python. Computation was done using CUDA 11.7 and CuDNN with Nvidia (Nvidia Corp., 148 Santa Clara, California) RTX3090 accelerator card with 24 GB.

2.5.1 Evaluating models

There is no trading strategy defined for trading the signal from the predictions, hence there is not a sharpe ratio (*Sharpe*, 1966) to base the evaluation on. However, using a

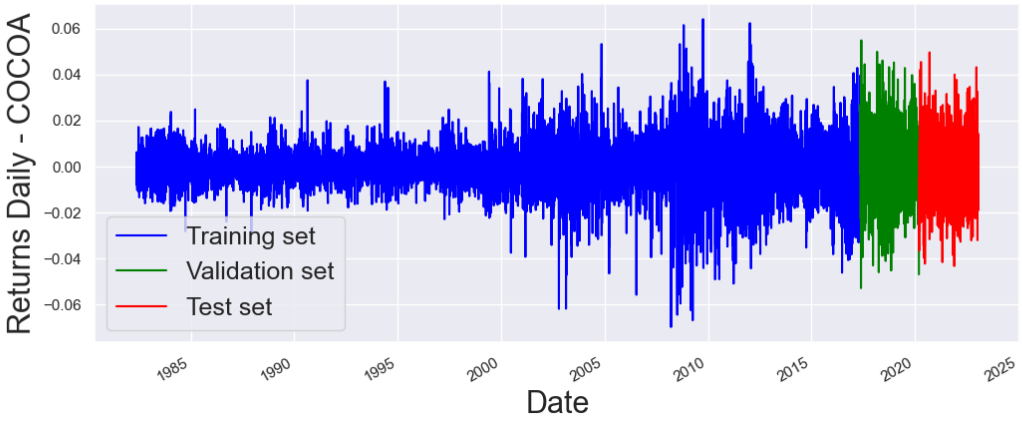

Figure 2.3: Train- validation- and test-set split

scorecard of metrics in an effort to analysing the models (*Gray*, 2018) is given (*Gray*, 2018).

$$\textbf{Noise} = \frac{1}{n}\sum_{i=2}^{n}\left|\hat{R}_i - \hat{R}_{i-1}\right| \tag{2.1}$$

The noise metric quantifies the extent of fluctuations in a model's predictions from one day to the next. A higher noise value suggests that the model exhibits abrupt and frequent changes in its predictions, making it challenging to interpret and potentially more costly for trading activities. In contrast, a lower noise value, indicates a more stable model that maintains a steadier prediction pattern, facilitating easier comprehension and potentially reducing trading expenses.

$$\textbf{Edge} = \frac{1}{n}\sum_{i=1}^{n}\text{sign}(\hat{R}_i)\cdot R_i \tag{2.2}$$

Arguably one of the most valuable metrics, this represents the expected average prediction value across a significantly large number of simulations or draws. Analogous to a blackjack card counter assessing the expected profit for each dollar bet under favorable odds, this metric provides insight to the average outcome over a diverse range of scenarios, aiding in strategic decision-making and risk assessment.

$$\textbf{Accuracy} = \frac{\text{Number of Correct Predictions}}{\text{Total Number of Predictions}} \times 100 \tag{2.3}$$

This measures the percent of sign-correct predictions.

$$\textbf{Prediction Calibration} = \frac{\text{mean}(|\hat{R}|)}{\text{mean}(|R|)} \tag{2.4}$$

A simple ratio of the magnitude of the predictions vs. magnitude of truth. This gives some indication of whether the model is properly tuned to the size of movement, in addition to the direction of it.

$$\textbf{Capture Ratio} = \frac{\text{Edge}}{\text{mean}(|R|)} \times 100 \tag{2.5}$$

Capture ratio is edge scaled by mean return. The ratio measures how well our predictions align with the actual daily changes. A value of 100 indicates perfect alignment, signifying that our predictions accurately capture the true movements of the target variable.

Chapter 3

Results

3.1 Scorecard results and analyses

The scorecard table 3.1.1 is sorted in descending order on edge, the most useful betting metric. The columns edge, noise y-true-change, y-prediction-change, edge-long and short are multiplied by 100.

We see that 22 out of 43 models (44%) have an accuracy of predicting the daily directional change of more than 50%. This gives a p-value of 44% for the hypothesis that accuracy is greater than 50% using a one tailed Z-test:

$$z = \frac{p - p_0}{\frac{p_0 - (1 - p_0}{\sqrt{n}}}$$

where $p = 22/43$ the sample proportion, $p = 0.5$ the hypnotized population proportion, and $n = 43$ is the sample size, which gives $Z = 0.152$. The probability of the area above the Z-score is 44%, which is not significant (p<0.05).

Three models has an accuracy of 50% and 18 models has an accuracy of less than 50%. The highest accuracy is gold (ZG) with 54.14%, but has only an edge of only 0.0006. 21 models has a positive edge, and 16 models has a negative edge, and 6 models has no edge. The highest edge is for Heating oil (ZH) with an edge of 0.0053. The lowest accuracy is wheat, KC (KW) with an accuracy of only 46.33%. It has also the lowest edge of -0.002.

3.1.1 Regression plots

From Figure 3.1 we see that the models looks randomly spread around 50%. Only 4 models has accuracy above when including standard deviation, AN, ZG, ZH, and ZU are above, but not for 2 x standard deviation.

From Figure 3.2 we see that all models has low noise, except two outliers, heating oil, ZH, and soybean meal, ZM. ZH has the best edge. With high noise it means that the model, and possibly also the underlying instrument changes sign of the return often. This can make the model difficult to follow, as there is little trend movement. It also increases the transaction cost of following the model.

From Figure 3.3 we see that there is a correlation between accuracy and edge. Most models are within the confidence interval of the regression plot, with a cluster in the

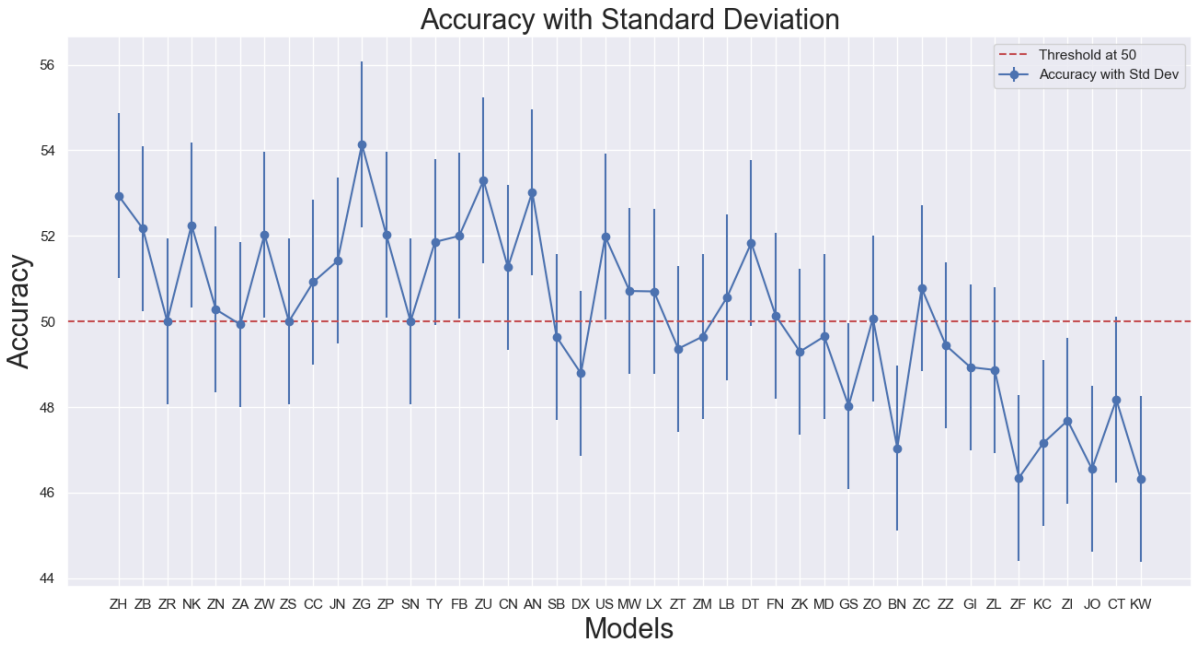

Figure 3.1: The y-axis shows accuracy with standard deviation, and the x-axis shows each model, sorted by edge in descending order. We see that most models has accuracy no lager than 50% when we include only 1 x standard deviation. AN, ZG, ZH, and ZU are above, but not for 2 x standard deviation.

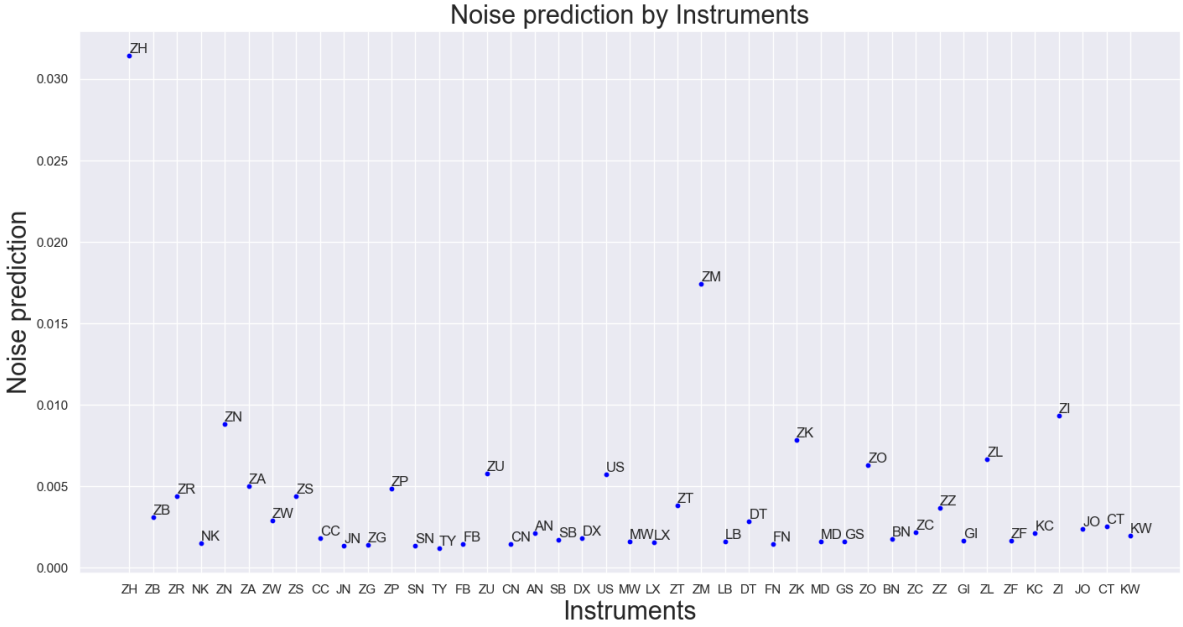

Figure 3.2: The y-axis shows noise, and the x-axis shows each instrument sorted by edge in descending order. Heating oil, ZH, and soybean meal, ZM, has high noise.

middle. Also here heating oil, ZH, is an outlier.

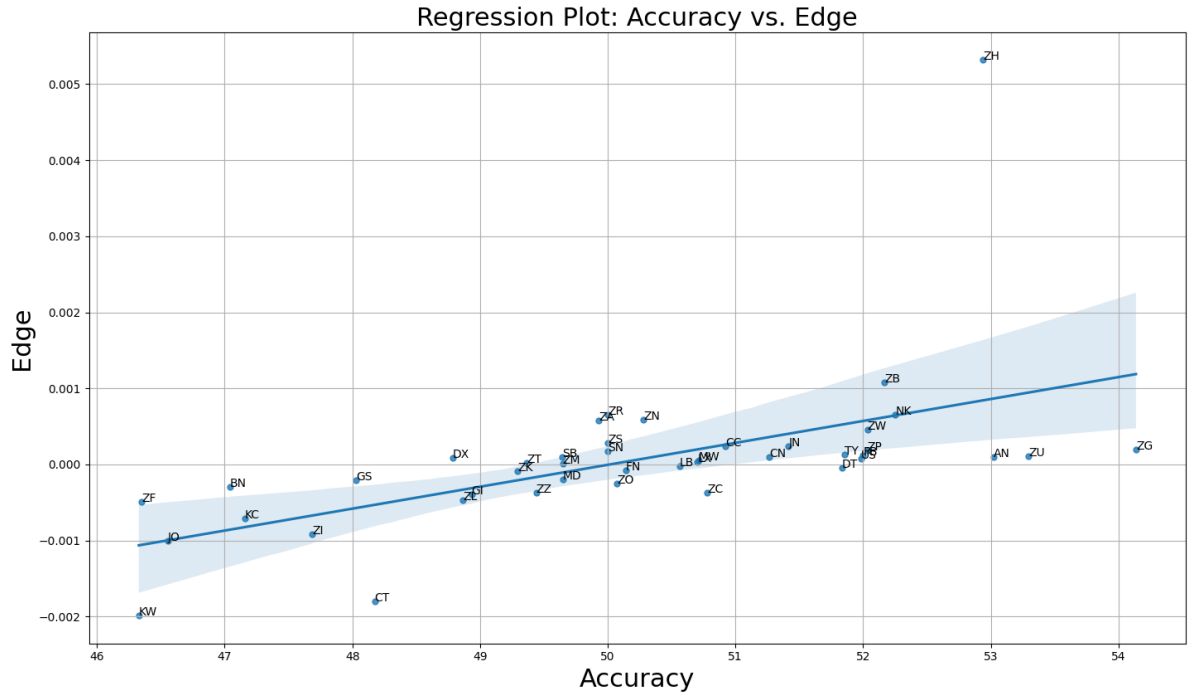

Figure 3.3: The y-axis is edge, and the x-axis is accuracy. There is a correlation between accuracy and edge. Most models are within the confidence interval of the regression plot, with a cluster in the middle. Also here heating oil, ZH, is an outlier.

From Figure 3.4 we see that round half the models has capture ratio above and below 0, with US T-note, 5years(FB), Nikkei index (NK), and heating oil (ZH) having the highest capture ratio and wheat, KC (KW) having the lowest. This means that the models FB, NK and ZH captures the daily changes well and are easy to follow. In the case of ZH which has high noise, it means the instrument is noisy but it is predictable, because it is captured well by the model. The models are sorted by edge in descending order, and there does not look like there is a correlation between edge and calibration ratio.

From Figure 3.5 we see that there is a positive relationship between capture ratio and accuracy as expected. Most models are clustered in the middle, and they fit relatively close to the regression line.

From Figure 3.6 we see that T-bonds, 5 years (US) and heating oil (ZM) has the highest calibration ratio, and the other models has a calibration ratio between 0.2 and 0.4. This describes the relationship between magnitude of our predictions vs. magnitude of truth. A model with high calibration ratio captures the true magnitude of the daily movement and we can have higher confidence in the true movement if also the accuracy is high. The models are sorted by edge in descending order. It looks like values with mean edge are more likely to have high calibration.

From Figure 3.7 we also see there is a positive correlation with calibration ratio and accuracy. US and ZM are again positive outliers.

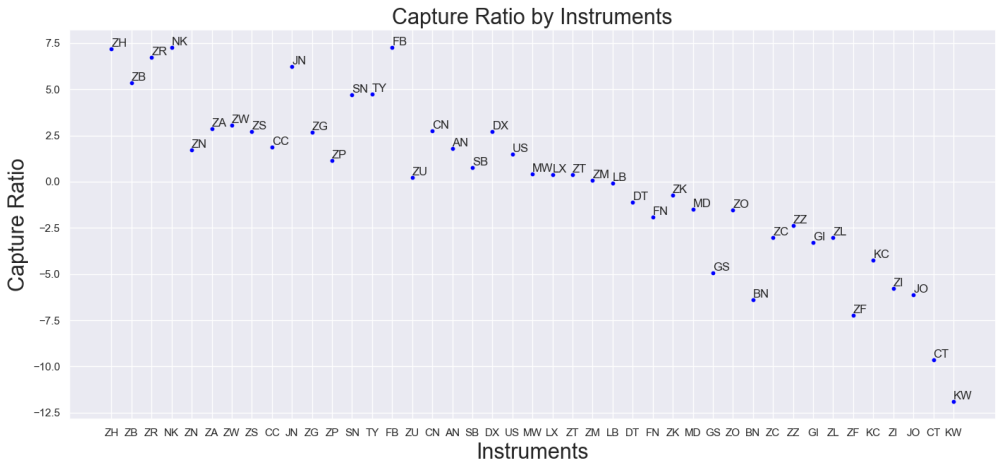

Figure 3.4: The y-axis shows capture ratio, and the x-axis shows each instrument sorted by edge in descending order. We see that LB has the highest capture ratio as well as highest accuracy and edge

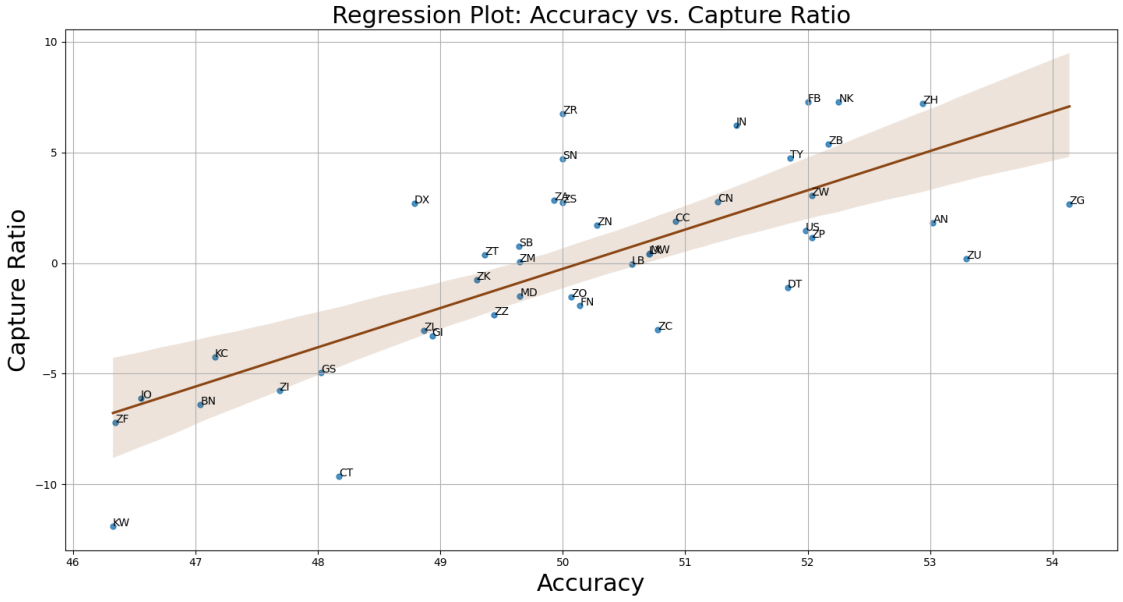

Figure 3.5: The y-axis shows capture ratio, and the x-axis shows accuracy. It is similar to the noise-accuracy plot.

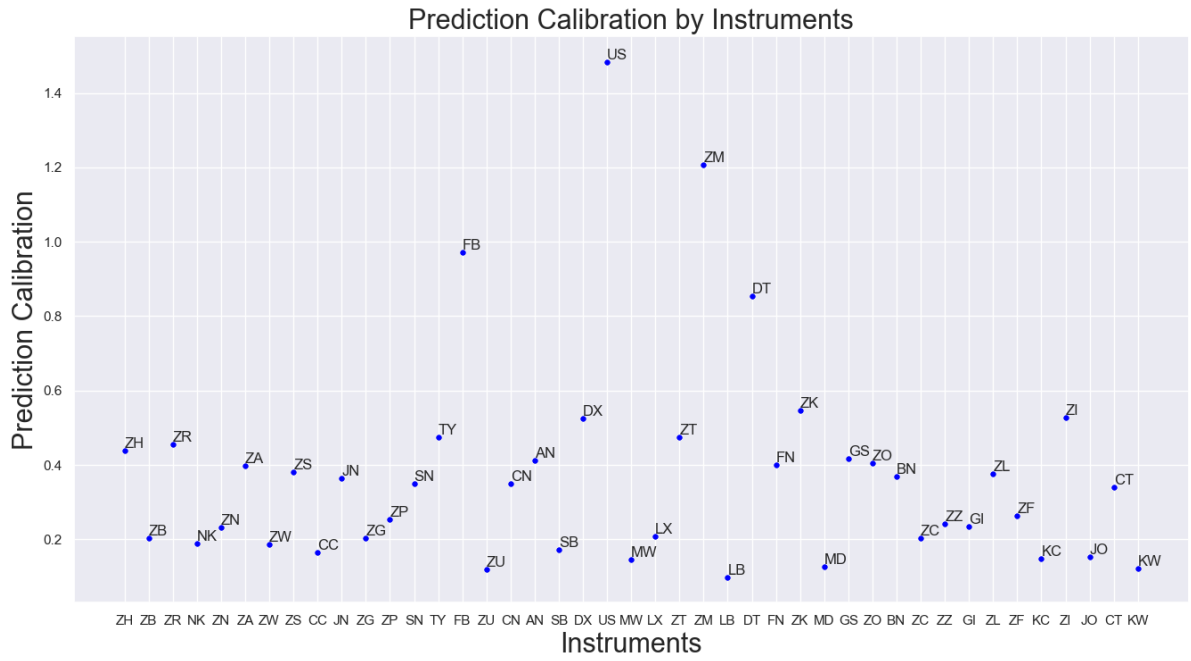

Figure 3.6: The y-axis shows prediction calibration, and the x-axis shows each instrument sorted by edge in descending order. We see that FB and US has highest calibration ratio, but US has lowest edge and accuracy

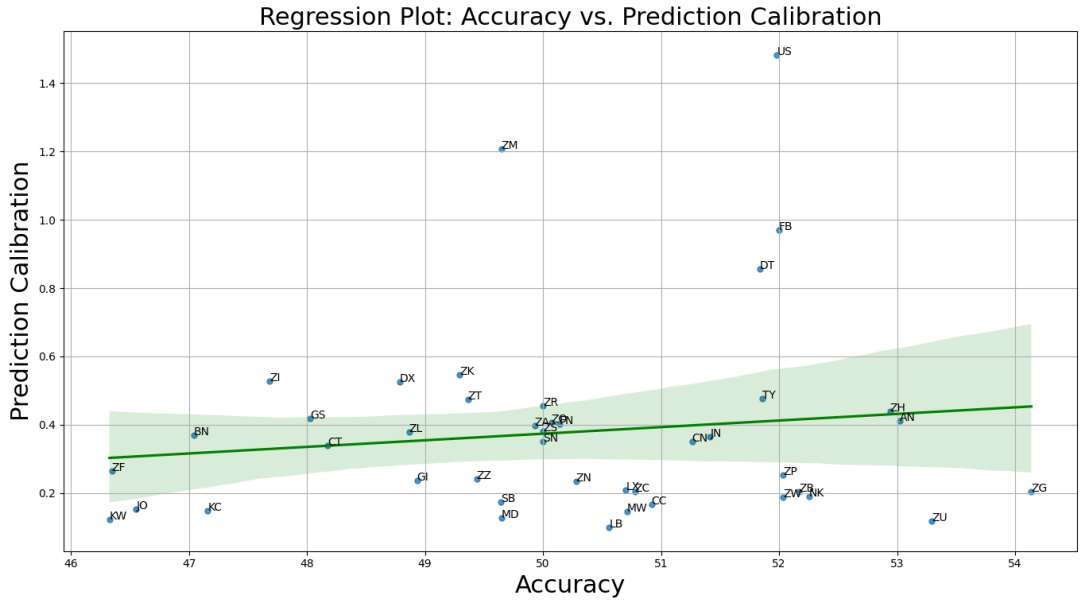

Figure 3.7: The y-axis is prediction calibration, and the x-axis is accuracy. We see that the plot has negative regression line with the best model with low prediction calibration

Instr.	accu-racy	edge	noise	y_{true} chg	y_{pred} chg	prediction calibration	capture ratio	edge long	edge short	edge win	edge lose
ZH	52.94	0.53	3.14	7.39	3.24	43.85	7.19	0.46	-1.12	6.67	-8.12
ZB	52.17	0.11	0.31	2.01	0.41	20.40	5.36	0.06	-0.44	1.77	-2.25
ZR	50.00	0.06	0.44	0.97	0.44	45.55	6.74	0.07	-0.02	1.00	-0.96
NK	52.25	0.06	0.15	0.90	0.17	18.85	7.26	0.09	-0.03	0.88	-0.92
ZN	50.28	0.06	0.88	3.43	0.80	23.30	1.72	0.12	-0.26	3.31	-3.58
ZA	49.93	0.06	0.50	2.04	0.81	39.73	2.85	0.16	0.04	2.10	-1.97
ZW	52.03	0.05	0.29	1.50	0.28	18.62	3.04	0.08	-0.04	1.45	-1.56
ZS	50.00	0.03	0.44	1.03	0.40	38.16	2.72	0.05	-0.20	0.95	-1.13
CC	50.92	0.02	0.18	1.28	0.21	16.54	1.88	0.02	-0.04	1.26	-1.33
JN	51.42	0.02	0.14	0.38	0.14	36.53	6.22	0.02	0.10	0.43	-0.34
ZG	54.14	0.02	0.14	0.72	0.15	20.35	2.66	0.02	-0.01	0.67	-0.78
ZP	52.03	0.02	0.49	1.65	0.42	25.35	1.15	0.02	-0.10	1.55	-1.76
SN	50.00	0.02	0.13	0.37	0.13	34.95	4.70	0.02	0.02	0.40	-0.36
TY	51.86	0.01	0.12	0.27	0.13	47.58	4.76	0.02	0.07	0.31	-0.25
FB	52.00	0.01	0.14	0.17	0.16	97.08	7.26	0.01	0.06	0.20	-0.15
ZU	53.30	0.01	0.58	5.08	0.60	11.84	0.21	-0.05	2.80	6.21	-4.02
CN	51.26	0.01	0.15	0.37	0.13	34.97	2.75	0.01	-0.01	0.37	-0.38
AN	53.02	0.01	0.21	0.56	0.23	41.20	1.81	0.00	-0.04	0.52	-0.62
SB	49.64	0.01	0.17	1.26	0.22	17.33	0.77	0.09	-0.18	1.21	-1.39
DX	48.79	0.01	0.18	0.34	0.18	52.42	2.70	0.01	-0.00	0.36	-0.33
US	51.98	0.01	0.57	0.53	0.79	148.36	1.47	0.01	0.08	0.56	-0.52
MW	50.71	0.00	0.16	1.26	0.18	14.59	0.43	0.03	-0.13	1.20	-1.38
LX	50.70	0.00	0.16	0.79	0.16	20.72	0.39	0.02	-0.12	0.72	-0.86
ZT	49.36	0.00	0.38	0.69	0.33	47.46	0.37	0.00	-0.09	0.67	-0.73
ZM	49.65	0.00	1.74	1.49	1.80	120.76	0.06	0.04	-0.22	1.40	-1.61
LB	50.56	-0.00	0.16	2.76	0.27	9.90	-0.06	0.07	-0.48	2.48	-3.06
DT	51.84	-0.00	0.28	0.35	0.30	85.52	-1.12	0.00	0.05	0.36	-0.34
FN	50.14	-0.01	0.15	0.38	0.15	40.08	-1.91	-0.00	0.00	0.38	-0.39
ZK	49.30	-0.01	0.78	1.22	0.66	54.55	-0.75	0.01	-0.17	1.14	-1.31
MD	49.65	-0.02	0.16	1.31	0.16	12.64	-1.50	-0.03	-0.22	1.21	-1.41
GS	48.03	-0.02	0.16	0.42	0.18	41.79	-4.95	-0.04	0.02	0.44	-0.41
ZO	50.07	-0.02	0.63	1.63	0.66	40.56	-1.54	0.01	-0.15	1.55	-1.76
BN	47.04	-0.03	0.18	0.46	0.17	36.93	-6.39	-0.03	-0.02	0.46	-0.46
ZC	50.78	-0.04	0.22	1.23	0.25	20.40	-3.01	-0.05	-0.34	1.06	-1.44
ZZ	49.44	-0.04	0.36	1.58	0.38	24.17	-2.36	0.02	-0.21	1.45	-1.71
GI	48.93	-0.04	0.16	1.18	0.28	23.57	-3.29	0.07	-0.23	1.06	-1.33
ZL	48.87	-0.05	0.67	1.56	0.59	37.72	-3.04	-0.03	-0.36	1.39	-1.76
ZF	46.35	-0.05	0.16	0.68	0.18	26.44	-7.21	-0.06	-0.05	0.68	-0.68
KC	47.16	-0.07	0.21	1.68	0.25	14.73	-4.24	-0.07	-0.22	1.67	-1.75
ZI	47.69	-0.09	0.93	1.58	0.84	52.75	-5.76	-0.09	-0.23	1.49	-1.69
JO	46.55	-0.10	0.24	1.63	0.25	15.25	-6.12	-0.10	-0.33	1.53	-1.76
CT	48.17	-0.18	0.26	1.87	0.63	33.91	-9.62	-0.10	-0.40	1.57	-2.18
KW	46.33	-0.20	0.20	1.67	0.20	12.19	-11.90	-0.18	-0.45	1.51	-1.85

Chapter 4

Discussion

4.1 Loss function and prediction on the test set

From Figure 4.1 we see training- and validation-loss of Australian $ (AN), heating oil (ZH), Nikkei index (NK), and T-notes 5 years (FB). An expected graph of validation loss declining is seen in AN, and this model was the first one trained. Training could have been longer because we don't see overfitting, but this is just for pre-training. The other graphs are more unusual, but here the model already has been trained on similar time series, and training is not always improving the validation error. This is the case for ZH and NK. However, the model does improve on training FB. An experiment was run where predictions where made where each model was trained from random initialization and then predicted on the test set. On all models the model accuracy was lower than when pre-training first. This also agrees with the general concept of transfer learning.

Little effort was placed on tuning hyperparameters, and likely the models would benefit from individual hyperparameters tuning such as learning rate, batch size, epochs and using a learning rate scheduler. The archictectur could also be explored more, e.g by adding drop-out (*Srivastava et al.*, 2014), and exploring different activation functions.

Another interesting result that has been shown lately is that of Grokking (Generalization Beyond Overfitting on Small Algorithmic Data sets) (*Power et al.*, 2022) which shows that training for a long time after validation loss has started to overfit, then the validation loss drops lower than previous lows (Figure 4.2). This contradicts the practice of early stopping, and it could be interesting to investigate if training these models for much longer without early stopping, could improve the results.

When we look at the predictions made on the first 100 time steps of the test set of of Lumber (LB) and T-bond (US) (Figure 4.3 that has the max accuracy and min accuracy of prediction on the test set, we see that the predictions in blue doesn't follow the movement, but one gets the direction right and the other more wrong.

4.1.1 Interpreting the scorecard

The scorecard shows that around half the models has slightly above 50% accuracy and the same for edge. This is no more accurate than drawing from a distribution of random

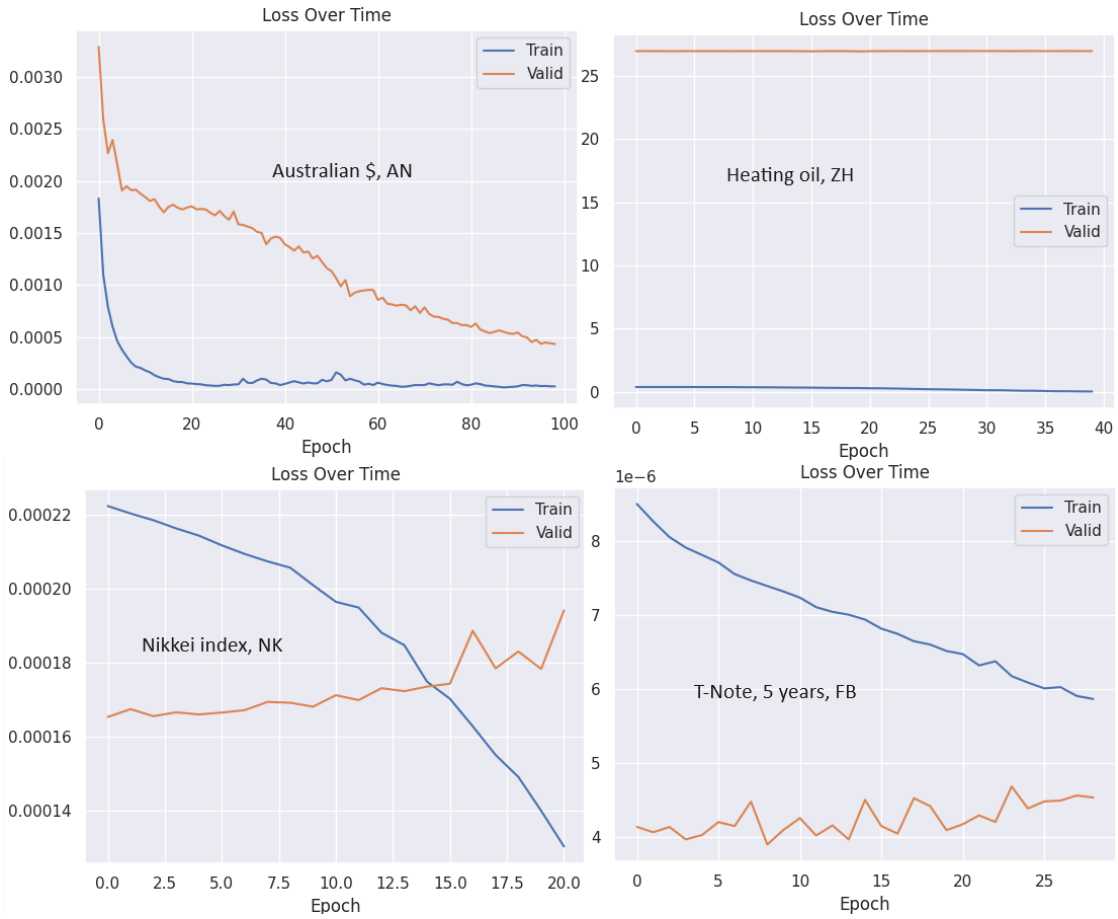

Figure 4.1: Training- and validation-loss on pretraining. From top left, Australian $ (AN), heating Oil (ZH), Nikkei index (NK), and T-notes, 5 years (FB).

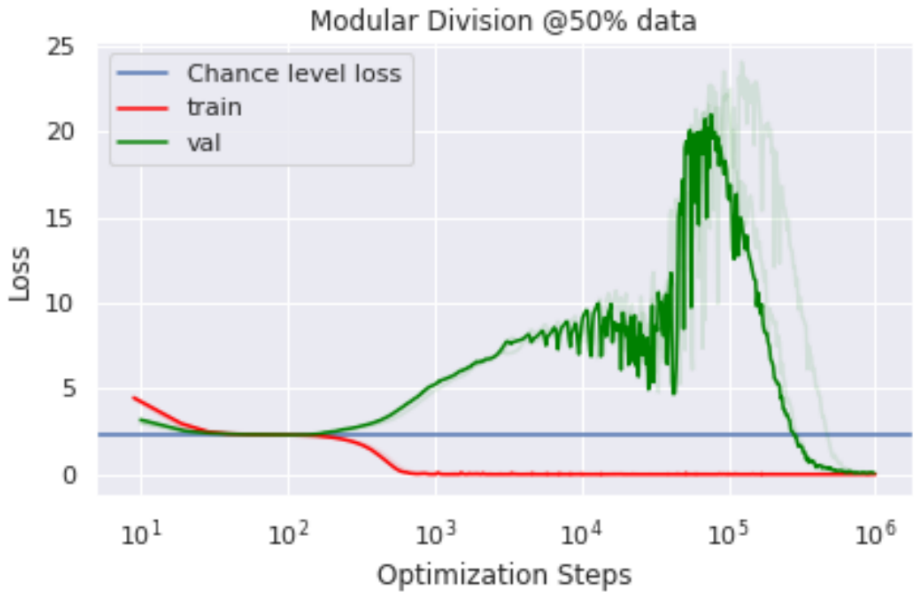

Figure 4.2: Grokking: A double hump on the validation loss and it drops to lower low after overfitting. Source: (Power et al., 2022)

Figure 4.3: First 100 predictions on the test set. Blue line is the true change in price and red line is predicted change.

models. We can not conclude that these models beats the market. Particularly not after accounting for transaction costs.

The test set encompasses over two years of trading data, from April 2020 to 27. January 2023. This period coinciding with the increasing integration of machine learning methods in trading during this period. If the test-set was of older data (assuming training and validation set was even older) we might have seen a more positive result. The paper this work was inspired from *Lim et al.* (2019) published their work in 2021, and it was done in 2020. So if the test set was as old as their work I might have found a different result. Since *Lim et al.* (2019) did not publish their code, it might also be a result of errors in the implementation, e.g in the architecture or in hyper-parameters. However, these details was not sufficiently described in their work. If *Lim et al.* (2019) found alpha in 2020, that alpha (*Sharpe*, 1964) is perhaps not there anymore.

The effectiveness of following the models' recommendations even for the models with highest edge remain uncertain. Only testing it with live data to compile a Profit and Loss statement will give answers. A related question is how well does the strategy fare compared to inflation or a buy-and-hold approach with a large index fund?

Another consideration is the potential for data leakage during training. This thesis has been rewritten after finding data leakage which substantially improved the accuracy of the models. There might be more data leakage, and the simplest and most reliable method to find out if there still is data leakage is by testing the models on live trading scenarios.

The model employed was published in 2016. It is well-established model and here it was operating on a relatively small dataset featuring only 7 time series with low resolution, capturing daily changes. Accuracy of the prediction can be made by increased by incorporating higher-resolution data. Leveraging newer and more advanced models may also improve accuracy, however, there are results suggests that transformers ((*Vaswani and et al.*, 2017)) is not all we need for time series data (*Zeng et al.*, 2022).

Platforms like Yahoo Finance offer free access to 1-minute resolution data, while market makers operate with resolutions measured in picoseconds. Additionally, the addition of alternative datasets, such as extracting sentiment from Twitter (*Pagolu et al.*, 2016), has demonstrated its potential advantage. The advent of large language models (LLMs) suggests that even more alternative datasets could enhance predictive capabilities in this domain.

4.1.2 Scorecard for Sugar (SB)

Looking at the scorecard for sugar (SB) in table 4.1.2 we see that it has the 8th best edge of 0.0046, an accuracy of 61.8%. It has the 4th best capture ratio of 0.36. The predictions aligns well with the daily changes.

Instr.	accuracy	edge	noise	y_{true} chg	y_{pred} chg	prediction calibration	capture ratio	edge long	edge short	edge win	edge lose
SB	49.64	0.01	0.17	1.26	0.22	17.33	0.77	0.09	-0.18	1.21	-1.39

4.1.3 Fitting an ARMA model to Sugar (SB)

To benchmark the performance of the WaveNet model, a comparison to more traditional time series prediction by modelling the next-days-return as a stochastic process has been done. This investigates if there are any moving-average(MA) or autoregressive(AR) processes. If the neural network outperforms the ARMA model, it suggests that the neural network is capturing more complex patterns in the data. However, if the ARMA model finds patterns, this provide insight into the underlying dynamics of the time series, as the ARMA model is more interpretable than the neural network.

```
import pandas as pd
import statsmodels.api as sm
from pmdarima import auto_arima
from arch import arch_model

#assums data has been read. See supplementary materials.
#selecting the test set from sugar
valuesToFit = df.loc['SB']['Next_Returns_Daily'].values[-715:]

# Perform ARIMA model selection using auto_arima
auto_model = auto_arima(valuesToFit, seasonal=False,
  suppress_warnings=True)

auto_model
#>(ARIMA(order=(0, 0, 0), scoring_args={}),)

# If we had a model we would - fit the ARIMA model with
#model = sm.tsa.ARIMA(valuesToFit,
#  order=(auto_model.order[0], 0, auto_model.order[1]))
#results = model.fit()
```

We see that the value to predict, Next-Returns-Daily, does not have any evidence of an underlying AR or MA process. We see this from the best model fitted, having 0 AR and MA parts in the fitted model. We also observe it in the Figure 4.4.

If there was a MA process, we would expect the Auto-Correlation Function (AFC) to show a sharp drop-off after a certain lag. E.g in an MA(q) process the autocorrelation will be significant at lag q but not for lags greater than q. There should be a spike at lag q.

In an AR process we would expect the ACF to decay gradually. For a AR(p) process the autocorrelation would be significant for the first p lags, and then gradually decay.

Fitting a Garch(1,1) model

A more sophisticated model is Generalized Autoregressive Conditional Heteroskedasticity (GARCH) (*Bollerslev*, 1987) which allows the model to support changes in the time dependent volatility, such as increasing and decreasing volatility.

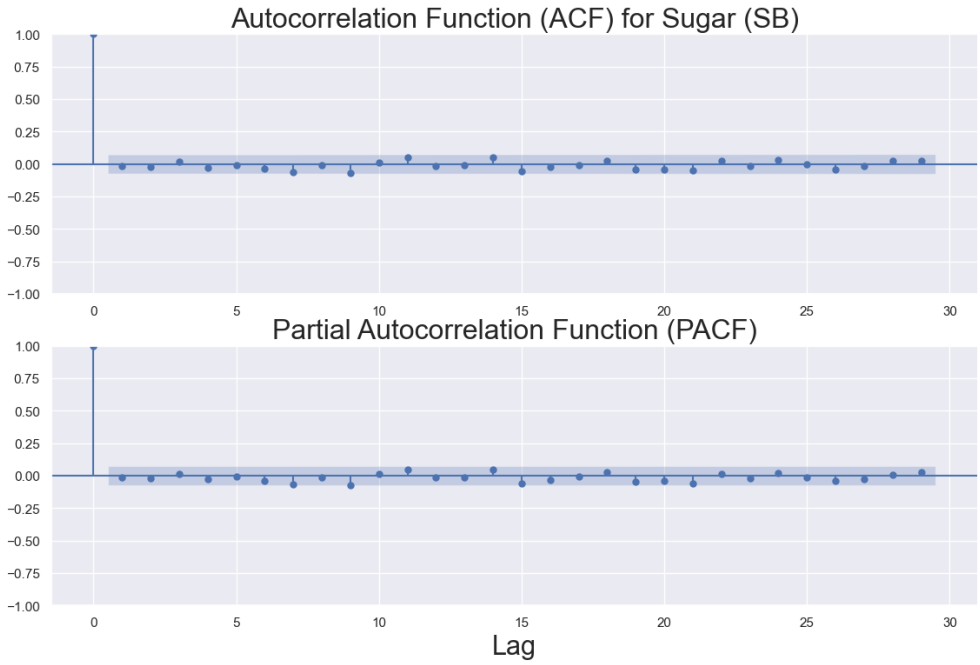

Figure 4.4: Auto-correlation and Partial auto-correlation function plotted for the Next-Returns-Daily of Sugar (SB) on the test-set

```
X = df.loc['SB',:]['Returns_Daily'] * 100

trainValMean = X[0:-715].mean() #-0.005
trainValVar = X[0:-715].var()   #1.953
testMean = X[-715:].mean()      #0.119
testVar = X[-715:].var()        #2.718
# Fit GARCH(1,1) model
model = arch_model(X[0:-715], mean='Zero', vol='GARCH', p=1, q=1)
modelfit = model.fit(disp='off')
#>modelfit.params
#omega        0.001585
#alpha[1]     0.040180
#beta[1]      0.959828

testSetYhat = np.asarray([])
for i in range(0,715):
  model =
    arch_model(X[0:-715+i], mean='Zero', vol='GARCH', p=1, q=1)
  modelfit = model.fit(disp='off')
  forecast = modelfit.forecast(horizon=1)
  yHat = forecast.variance['h.1'].values[0]
  testSetYhat = np.append(testSetYhat, yHat)
```

The model has slightly different parameters when looking at the training and vali-

dation set combined and when including the test set. In the first case the parameters are $\omega = 0.016, \alpha = 0.040, \beta = 0.96$ and with the whole set $\omega = 0.015, \alpha = 0.039, \beta = 0.96$.

With with a horizon of 1, as in the deep learning model, and evaluating the predictions on the scorecard. The GARCH model predicts volatility which is strictly positive and is not the same as daily change, however a comparison to the deep learning model is still interesting. In particular for accuracy it is predicting positive return every day, and this can be viewed as a baseline prediction model to beat.

Instr.	accuracy	edge	noise	y_{true} chg	y_{pred} chg	prediction calibration	capture ratio	edge long	edge short	edge win	edge lose
SB	50.93	0.12	0.11	1.30	2.87	2.22	9.20	0.00	NaN	1.30	-1.35

We see from table 4.1.3 that the GARCH(1,1) model performs better than the deep learning model on almost all scores. In particular it has higher accuracy and higher edge.

4.1.4 Future work

The model size is restricted by the availability of data. Large language models have billions of parameters, and train on large data sets collected by crawling the internet. One interesting work made by *Marti* (2020) is to sample more financial data based on the correlation structure of the existing time series, using a model called CorrGAN. The correlation structure is discovered using a generative adversarial network (GAN) (*Goodfellow et al.*, 2014) model with a discriminator network and a generator network, where the generator network generates new time-series based on the existing time series, while the discriminator learns to discriminate between synthetically generated and real time series. This generates a min-max algorithm (*Neumann*, 1928),and the loss function lives on a saddle-point. The loss function can be notoriously difficult to train due to the fact that the loss-function can fall off the saddle.

Another question that could be interesting to investigate, is which time series contains the most information helpful in predicting a given instruments daily return. This is a rich topic with many options to choose from. One approach is to use cross-correlation analysis to measure the similarity between two time series as a function of their time lag applied to one of them, or on the residuals. Another approach is to use feature importance analysis. A machine learning model like the random forest (*Breiman*, 2001) generates decision trees, which are more interpretable than neural networks, and also easier to understand the feature importance. There are other methods to understand feature importance, that can be applied to neural networks like leave-one-feature-out and measure the change in the loss function.

Yet another approach is to calculate the correlation matrix on the residuals of the predictions of the model, and do a cluster analysis on the matrix with k-means or hierarchical clustering (*Nielsen*, 2016) on Euclidean distance or Mahalanobis distance (*Mahalanobis*, 1936).

4.1.5 Conclusions

There is no reason to question the efficient market hypothesis after this work. Some models has a positive edge, but after accounting for transaction cost, that edge is marginal, if not negative. If *Lim et al.* (2019) found alpha in 2020 that alpha is perhaps not in the market anymore for the commodities in this work, or there are still room for improvements in the features and models used in this thesis. Or, perhaps the findings found was too optimistic. WaveNet does not beat a baseline model that predicts every day a positive return.

Bibliography

Abadi, M., A. Agarwal, P. Barham, E. Brevdo, Z. Chen, C. Citro, G. S. Corrado, A. Davis, J. Dean, M. Devin, et al. (2016), Tensorflow: Large-scale machine learning on heterogeneous distributed systems, *arXiv preprint arXiv:1603.04467*. 2.4, 2.5

Bachelier, L. (1900), Théorie de la spéculation, in *Annales scientifiques de l'École normale supérieure*, vol. 17, pp. 21–86. 1.5

Black, F., and M. Scholes (1973), The pricing of options and corporate liabilities, *Journal of political economy*, *81*(3), 637. 1.2

Bollerslev, T. (1987), A conditionally heteroskedastic time series model for speculative prices and rates of return, *The Review of Economics and Statistics*, *69*(3), 542–547. 4.1.3

Breiman, L. (2001), Random forests, *Machine Learning*, *45*(1), 5–32, doi:10.1023/A:1010933404324. 4.1.4

Chollet, F., and others (2018), Keras 2.1.3, https://github.com/fchollet/keras. 2.5

Deng, J., W. Dong, R. Socher, L.-J. Li, K. Li, and L. Fei-Fei (2009), Imagenet: A large-scale hierarchical image database, in *2009 IEEE Conference on Computer Vision and Pattern Recognition*, pp. 248–255, doi:10.1109/CVPR.2009.5206848. 1.5.1

Fawaz, H. I., G. Forestier, J. Weber, L. Idoumghar, and P. Muller (2018), Transfer learning for time series classification, *CoRR*, *abs/1811.01533*. 2.5

Goodfellow, I. J., J. Pouget-Abadie, M. Mirza, B. Xu, D. Warde-Farley, S. Ozair, A. Courville, and Y. Bengio (2014), Generative adversarial networks. 4.1.4

Gray, C. (2018), Stock prediction with ml: Model evaluation, https://alphascientist.com/model_evaluation.html, published on 09.08.2018. 2.5.1

He, K., X. Zhang, S. Ren, and J. Sun (2016), Deep Residual Learning for Image Recognition, in *Proceedings of 2016 IEEE Conference on Computer Vision and Pattern Recognition*, CVPR '16, pp. 770–778, IEEE, doi:10.1109/CVPR.2016.90. 2.4

Hochreiter, S., and J. Schmidhuber (1997), Long short-term memory, *Neural Computation*, *9*(8), 1735–1780, doi:10.1162/neco.1997.9.8.1735. 1.2

Kingma, D. P., and J. Ba (2017), Adam: A method for stochastic optimization. 2.4.1

Krizhevsky, A., I. Sutskever, and G. E. Hinton (2012), Imagenet classification with deep convolutional neural networks, *Advances in Neural Information Processing Systems*, *25*, 1097–1105. 1.5.1

Lamberti, M. (2020), Time series momentum, https://github.com/maxlamberti/time-series-momentum, accessed and cloned: 31.01.2023. 2.1

LeCun, Y., B. Boser, J. S. Denker, D. Henderson, R. E. Howard, W. Hubbard, and L. D. Jackel (1989), Backpropagation applied to handwritten zip code recognition, *Neural Computation*, *1*(4), 541–551, doi:10.1162/neco.1989.1.4.541. 1.3

Lim, B., S. Zohren, and S. Roberts (2019), Enhancing time series momentum strategies using deep neural networks, *The Journal of Financial Data Science*, available at SSRN: https://ssrn.com/abstract=3369195 or http://dx.doi.org/10.2139/ssrn.3369195. 1.2, 2.1, 4.1.1, 4.1.5

Mahalanobis, P. C. (1936), On the generalized distance in statistics, *Proceedings of the National Institute of Sciences (Calcutta)*, *2*, 49–55. 4.1.4

Marti, G. (2020), Corrgan: Sampling realistic financial correlation matrices using generative adversarial networks, in *ICASSP 2020 - 2020 IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP)*, IEEE, doi:10.1109/icassp40776.2020.9053276. 4.1.4

Moen, E., N. O. Handegard, V. Allken, O. T. Albert, A. Harbitz, and K. Malde (2018), Automatic interpretation of otoliths using deep learning, *PLOS ONE*, *13*(12), e0204,713, doi:10.1371/journal.pone.0204713, https://journals.plos.org/plosone/article/file?id=10.1371/journal.pone.0204713. 1.5.1

Moen, E., R. Vabø, S. Smoliski, Åse Husebø, N. O. Handegard, and K. Malde (2021), Automatic interpretation of salmon scales using deep learning, *Ecological Informatics*, *63*, 101,322, doi:https://doi.org/10.1016/j.ecoinf.2021.101322. 1.5.1

Moen, E., R. Vabø, S. Smoliski, C. Denechaud, N. O. Handegard, and K. Malde (2023), Age interpretation of cod otoliths using deep learning, *Ecological Informatics*, *78*, 102,325, doi:https://doi.org/10.1016/j.ecoinf.2023.102325. 1.5.1

Neumann, J. v. (1928), Zur theorie der gesellschaftsspiele, *Mathematische Annalen*, *100*, 295–320. 4.1.4

Nielsen, F. (2016), *Hierarchical Clustering*, pp. 195–211, Springer International Publishing, Cham, doi:10.1007/978-3-319-21903-5_8. 4.1.4

Oord, A. v. d., S. Dieleman, H. Zen, K. Simonyan, O. Vinyals, A. Graves, N. Kalchbrenner, A. Senior, and K. Kavukcuoglu (2016), Wavenet: A generative model for raw audio, cite arxiv:1609.03499. (document), 1.2, 1.3, 1.5.2, 1.2, 1.3, 1.5.2

Pagolu, S., K. Challa, G. Panda, and B. Majhi (2016), Sentiment analysis of twitter data for predicting stock market movements, *2016 International Conference on Signal Processing, Communication, Power and Embedded System (SCOPES)*. 4.1.1

Pinnacle Data Corp. CLC Database (), Pinnacle data corp. clc database, https://pinnacledata2.com/clc.html. 1.2

Power, A., Y. Burda, H. Edwards, I. Babuschkin, and V. Misra (2022), Grokking: Generalization beyond overfitting on small algorithmic datasets, *CoRR*, *abs/2201.02177*. (document), 4.1, 4.2

Sharpe, W. F. (1964), Capital asset prices: A theory of market equilibrium under conditions of risk, *The Journal of Finance*, *19*(3), 425–442, doi:10.2307/2977928. 4.1.1

Sharpe, W. F. (1966), Mutual fund performance, *The Journal of Business*, *39*(1), 119–138. 2.5.1

Srivastava, N., G. Hinton, A. Krizhevsky, I. Sutskever, and R. Salakhutdinov (2014), Dropout: A simple way to prevent neural networks from overfitting, *Journal of Machine Learning Research*, *15*(56), 1929–1958. 4.1

Tibshirani, R. (1996), Regression shrinkage and selection via the lasso, *Journal of the Royal Statistical Society: Series B (Methodological)*, *58*(1), 267–288. 1.2

Vaswani, A., and et al. (2017), Attention is all you need, *NeurIPS*, *30*(2), 5998–6008, doi:10.5555/3295222.3295349. 4.1.1

Ville, J. (1939), Étude critique de la notion de collectif, *Bulletin of the American Mathematical Society. Monographies des Probabilités*, *3*(11), 824–825, doi:10.1090/S0002-9904-1939-07089-4, review by Doob. 1.3, 1.5

Werbos, P. (1982), Applications of advances in nonlinear sensitivity analysis, in *System modeling and optimization*, pp. 762–770, Springer, archived (PDF) from the original on 14 April 2016. Retrieved 2 July 2017. https://werbos.com/Neural/SensitivityIFIPSeptember1981.pdf. 1.2

Zeng, A., M. Chen, L. Zhang, and Q. Xu (2022), Are transformers effective for time series forecasting? 4.1.1

Supplementary information

A. Model Summary

Layer (type)	Output Shape	Param #	Connected to
input_2 (InputLayer)	[(None, 255, 24)]	0	[]
conv1d_16 (Conv1D)	(None, 255, 256)	12544	['input_2[0][0]']
activation_16 (Activation)	(None, 255, 256)	0	['conv1d_16[0][0]']
activation_17 (Activation)	(None, 255, 256)	0	['conv1d_16[0][0]']
tf.math.multiply_8 (TFOpLa mbda)	(None, 255, 256)	0	['activation_16[0][0]', 'activation_17[0][0]']
conv1d_17 (Conv1D)	(None, 255, 256)	65792	['tf.math.multiply_8[0][0]']
conv1d_18 (Conv1D)	(None, 255, 256)	131328	['conv1d_17[0][0]']
activation_18 (Activation)	(None, 255, 256)	0	['conv1d_18[0][0]']
activation_19 (Activation)	(None, 255, 256)	0	['conv1d_18[0][0]']
tf.math.multiply_9 (TFOpLa mbda)	(None, 255, 256)	0	['activation_18[0][0]', 'activation_19[0][0]']
conv1d_19 (Conv1D)	(None, 255, 256)	65792	['tf.math.multiply_9[0][0]']
add_7 (Add)	(None, 255, 256)	0	['conv1d_17[0][0]', 'conv1d_19[0][0]']
conv1d_20 (Conv1D)	(None, 255, 256)	131328	['add_7[0][0]']
activation_20 (Activation)	(None, 255, 256)	0	['conv1d_20[0][0]']
activation_21 (Activation)	(None, 255, 256)	0	['conv1d_20[0][0]']
tf.math.multiply_10 (TFOpL ambda)	(None, 255, 256)	0	['activation_20[0][0]', 'activation_21[0][0]']
conv1d_21 (Conv1D)	(None, 255, 256)	65792	['tf.math.multiply_10[0][0]']
add_8 (Add)	(None, 255, 256)	0	['add_7[0][0]', 'conv1d_21[0][0]']
conv1d_22 (Conv1D)	(None, 255, 256)	131328	['add_8[0][0]']
activation_22 (Activation)	(None, 255, 256)	0	['conv1d_22[0][0]']
activation_23 (Activation)	(None, 255, 256)	0	['conv1d_22[0][0]']
tf.math.multiply_11 (TFOpL ambda)	(None, 255, 256)	0	['activation_22[0][0]', 'activation_23[0][0]']
conv1d_23 (Conv1D)	(None, 255, 256)	65792	['tf.math.multiply_11[0][0]']
add_9 (Add)	(None, 255, 256)	0	['add_8[0][0]', 'conv1d_23[0][0]']
conv1d_24 (Conv1D)	(None, 255, 256)	131328	['add_9[0][0]']
activation_24 (Activation)	(None, 255, 256)	0	['conv1d_24[0][0]']
activation_25 (Activation)	(None, 255, 256)	0	['conv1d_24[0][0]']
tf.math.multiply_12 (TFOpL ambda)	(None, 255, 256)	0	['activation_24[0][0]', 'activation_25[0][0]']
conv1d_25 (Conv1D)	(None, 255, 256)	65792	['tf.math.multiply_12[0][0]']
add_10 (Add)	(None, 255, 256)	0	['add_9[0][0]', 'conv1d_25[0][0]']
conv1d_26 (Conv1D)	(None, 255, 256)	131328	['add_10[0][0]']
activation_26 (Activation)	(None, 255, 256)	0	['conv1d_26[0][0]']

activation_27 (Activation)	(None, 255, 256)	0	['conv1d_26[0][0]']
tf.math.multiply_13 (TFOpLambda)	(None, 255, 256)	0	['activation_26[0][0]', 'activation_27[0][0]']
conv1d_27 (Conv1D)	(None, 255, 256)	65792	['tf.math.multiply_13[0][0]']
add_11 (Add)	(None, 255, 256)	0	['add_10[0][0]', 'conv1d_27[0][0]']
conv1d_28 (Conv1D)	(None, 255, 256)	131328	['add_11[0][0]']
activation_28 (Activation)	(None, 255, 256)	0	['conv1d_28[0][0]']
activation_29 (Activation)	(None, 255, 256)	0	['conv1d_28[0][0]']
tf.math.multiply_14 (TFOpLambda)	(None, 255, 256)	0	['activation_28[0][0]', 'activation_29[0][0]']
conv1d_29 (Conv1D)	(None, 255, 256)	65792	['tf.math.multiply_14[0][0]']
add_12 (Add)	(None, 255, 256)	0	['add_11[0][0]', 'conv1d_29[0][0]']
conv1d_30 (Conv1D)	(None, 255, 256)	131328	['add_12[0][0]']
activation_30 (Activation)	(None, 255, 256)	0	['conv1d_30[0][0]']
activation_31 (Activation)	(None, 255, 256)	0	['conv1d_30[0][0]']
tf.math.multiply_15 (TFOpLambda)	(None, 255, 256)	0	['activation_30[0][0]', 'activation_31[0][0]']
conv1d_31 (Conv1D)	(None, 255, 256)	65792	['tf.math.multiply_15[0][0]']
add_13 (Add)	(None, 255, 256)	0	['add_12[0][0]', 'conv1d_31[0][0]']
max_pooling1d_1 (MaxPooling1D)	(None, 127, 256)	0	['add_13[0][0]']
flatten_1 (Flatten)	(None, 32512)	0	['max_pooling1d_1[0][0]']
dense_3 (Dense)	(None, 256)	8323328	['flatten_1[0][0]']
dense_4 (Dense)	(None, 64)	16448	['dense_3[0][0]']
dense_5 (Dense)	(None, 1)	65	['dense_4[0][0]']
Total params	**9798017**	**(37.38 MB)**	
Trainable params	**9798017**	**(37.38 MB)**	
Non-trainable params	**0**	**(0.00 Byte)**	

B. Source of project

Multivariate WaveNet for backtesting on financial time series

This notebook reads in the Pinnacle CLC database into a Pandas dataframe, creates features from the time series provided by Pinnacle for each instruments. This includes date, open, high, low, close, volumne, and open interest. From these time series new features are derived like exponential Moving Average (EMA) over the last 60 days, Moving Average Convergence Divergence (MACD). The code for generating the features are borrowed from https://github.com/maxlamberti/time-series-momentum which implements some of the deep momentum network timeseries momentum factor paper proposed by Lim, Zohren and Roberts (2019).

After creating the featuers then time series without an explicit type are cast to the correct type.

We define the WaveNet model which handles multivariate time series.

We define a window size for the model, and the training loop. We split the dataset into train, validate and test-sets. The model is pretrained on all 43 instruments before it is fine-tuned on the training set of the instrument it will predict on the test-set. The validation set is used to select the optimal weights with early stopping.

Finally we evaluate the models on a scorecard of metrics.

[ ]:
```
pip install scikit-learn
pip install pandas
pip install numpy
pip install matplotlib
pip install seaborn
```

[ ]:
```
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt

import tensorflow as tf
import seaborn as sns
import os
from sklearn.preprocessing import MinMaxScaler

sns.set()
```

[ ]:
```
os.environ["CUDA_DEVICE_ORDER"] = "PCI_BUS_ID"
os.environ["CUDA_VISIBLE_DEVICES"] = "0"
print(tf.__version__)
print(tf.config.list_physical_devices('GPU'))
```

Reads the .csv files temporarily into a dictionary. Each instrument is merged on Date

[40]:
```
# Source for generating these features: https://github.com/maxlamberti/time-series-momentum
def load_clc_db_records(path, feds_data=None, assets_to_use=None):
    """Load all CLC futures data into dict. One entry per asset. Path specifies␣
 ↪location of .csv files"""
```

```
    if feds_data is not None:
        short_rate = pd.read_csv(feds_data)
        short_rate['DATE'] = pd.to_datetime(short_rate['DATE'],␣
↪format='%Y-%m-%d')
        feds_label = short_rate.columns[-1]
    data = {}
    files = [file for file in os.listdir(path) if '.csv' in file.lower()]
    for file in files:
        # skip assets which are not used for analysis
        asset = file[:-4].split('_')[0]
        if (assets_to_use is not None) and (asset not in ASSETS_TO_USE):
            continue
        # load asset data //Settle
        data[asset] = pd.read_csv(os.path.join(path, file), names=['Date',␣
↪'Open', 'High', 'Low', 'close', 'Volume', 'Open_Interest'])
        #0.0 for dates with 0 trading - error in return calc
        data[asset]['Open'] = data[asset]['Open'].replace(0.0, method='ffill')
        data[asset]['High'] = data[asset]['High'].replace(0.0, method='ffill')
        data[asset]['Low'] = data[asset]['Low'].replace(0.0, method='ffill')
        data[asset]['close'] = data[asset]['close'].replace(0.0, method='ffill')
        data[asset]['Volume'] = data[asset]['Volume'].replace(0.0,␣
↪method='ffill')
        data[asset]['Date'] = pd.to_datetime(data[asset]['Date'], format='%m/%d/
↪%Y')
        # merge with fed short rate if available
        if feds_data is not None:
            data[asset] = data[asset].merge(short_rate, how='left',␣
↪left_on='Date', right_on='DATE')
            data[asset][feds_label].ffill(inplace=True)
            data[asset][feds_label] /= 100
            data[asset]['Short_Rate_Daily'] = (1 + data[asset][feds_label]) **␣
↪(1 / 252) - 1
            data[asset]['Short_Rate_Annual'] = data[asset][feds_label]
            del data[asset][feds_label]
            del data[asset]['DATE']
        #data[asset].set_index('Date', inplace=True)
    return data
```

1. Calculates the normalized period returns for a given time series of daily returns using a rolling window
2. Computes the Moving Average Convergence Divergence (MACD) features for a given price time series with a given short and long period
3. Compute binary MACD
4. Calculate various features
5. Loops over a dictionary of instruments creating data frames of calculated features for each

[ ]:
```
# Source: https://github.com/maxlamberti/time-series-momentum
def calc_normalized_period_returns(daily_returns, daily_std, period):
    period = int(period)
    returns = (1 + daily_returns).rolling(period).apply(np.prod, raw=True) - 1
    return returns / (np.sqrt(period) * daily_std)


def calc_macd_features(price, short_period, long_period):
    short_ma = price.ewm(span=short_period, min_periods=short_period).mean()
    long_ma = price.ewm(span=long_period, min_periods=long_period).mean()
    ewmstd_63 = price.ewm(span=63).std()
    macd = short_ma - long_ma
    q = macd / ewmstd_63
    z = q / q.ewm(span=252, min_periods=252).std()
    return z

def calc_macd_binary(price, short_period, long_period):
    short_ma = price.ewm(span=short_period, min_periods=short_period).mean()
    long_ma = price.ewm(span=long_period, min_periods=long_period).mean()
    signal = short_ma >= long_ma
    return signal


def construct_features_single_asset(df, ewmastd_span=60, inplace=False,␣
↪asset_label=None):
    if not inplace:
        df = df.copy()
    if asset_label is not None:
        df['Asset'] = asset_label
    df['Returns_Daily'] = df.close.pct_change()  # np.log(df['Settle']).diff(),␣
↪Settle
    df['Next_Returns_Daily'] = df['Returns_Daily'].shift(-1)
    df['Sigma'] = df['Returns_Daily'].ewm(span=ewmastd_span,␣
↪min_periods=ewmastd_span).std()
    df['Norm_Returns_Daily'] = df['Returns_Daily'] / df['Sigma']
    df['Norm_Returns_Monthly'] =␣
↪calc_normalized_period_returns(df['Returns_Daily'], df['Sigma'], 252 / 12)
    df['Norm_Returns_Quarterly'] =␣
↪calc_normalized_period_returns(df['Returns_Daily'], df['Sigma'], 252 / 3)
    df['Norm_Returns_Semiannually'] =␣
↪calc_normalized_period_returns(df['Returns_Daily'], df['Sigma'], 252 / 2)
    df['Norm_Returns_Annually'] =␣
↪calc_normalized_period_returns(df['Returns_Daily'], df['Sigma'], 252)
    df['MACD_8_24'] = calc_macd_features(df['close'], 8, 24) #Settle
    df['MACD_16_48'] = calc_macd_features(df['close'], 16, 48)
    df['MACD_32_96'] = calc_macd_features(df['close'], 32, 96)
```

```
    df['Binary_MACD_8_24'] = calc_macd_binary(df['close'], 8, 24)
    df['Binary_MACD_16_48'] = calc_macd_binary(df['close'], 16, 48)
    df['Binary_MACD_32_96'] = calc_macd_binary(df['close'], 32, 96)
    df['Sigma_Norm'] = np.log(df['Sigma'] / df['Sigma'].rolling(181).mean())
    df['Returns_Weekly'] = ((1 + df.Returns_Daily).rolling(5).apply(np.prod,␣
 ↪raw=True) - 1)
    if 'Short_Rate_Daily' in df.columns:
        df['Excess_Returns_Daily'] = df['Returns_Daily'] -␣
 ↪df['Short_Rate_Daily']
    return df


def construct_features_batch(df_map):
    for asset, df in df_map.items():
        construct_features_single_asset(df, inplace=True, asset_label=asset)
    return df_map
```

Merges multiple asset DataFrames into a single DataFrame along the index assets. Creates multi-index Asset and Date.

[ ]:
```
# Source: https://github.com/maxlamberti/time-series-momentum
def merge_asset_data(asset_to_df_map, create_time_asset_index=True):
    "Merges multiple asset dataframes together."
    asset_dfs = [df for asset, df in asset_to_df_map.items()]
    combined_assets = pd.concat(asset_dfs, ignore_index=True)
    if create_time_asset_index:
        combined_assets.set_index(['Asset', 'Date'], inplace=True, drop=False)
        combined_assets.sort_index(inplace=True)
        combined_assets['Asset_Col'] = combined_assets['Asset']
        combined_assets['Date_Col'] = combined_assets['Date']
        del combined_assets['Date']
        del combined_assets['Asset']
    return combined_assets
```

Define the Assets to use.

[ ]:
```
assets_to_use = ['AN', 'BN', 'CC', 'CN', 'CT', 'DT', 'DX', 'EC', 'FB', 'FF',␣
 ↪'FN',
       'FX', 'GI', 'GS', 'JN', 'JO', 'KC', 'KW', 'LB', 'LX', 'MD', 'MW',
       'NK', 'SB', 'SN', 'SS', 'TU', 'TY', 'US',
       'ZA', 'ZB', 'ZC', 'ZF', 'ZG', 'ZH', 'ZI', 'ZK', 'ZL', 'ZM', 'ZN',
       'ZO', 'ZP', 'ZR', 'ZS', 'ZT', 'ZU', 'ZW', 'ZZ']
exclude_assets = ['FF', 'EC', 'TU', 'SS', 'PA' 'W','NR', 'SP']
ASSETS_TO_USE = list(set(assets_to_use) - set(exclude_assets))
len(ASSETS_TO_USE)
```

Define path to csv files and create dataframe, df, that contains all features for all instruments.

```
[ ]: RAD_DATA_PATH = '../pinnacle/clc/rad/'
FED_DATA_PATH = '../pinnacle/FEDFUNDS.csv'
# load asset data from clc database
clc = load_clc_db_records(RAD_DATA_PATH, FED_DATA_PATH, ASSETS_TO_USE)
clc = construct_features_batch(clc)
df = merge_asset_data(clc, create_time_asset_index=True)
df.dropna(inplace=True) #this drops 'PA' -PALLADIUM

df = df.reindex(df.index, level=0) #Remove 'PA' from Asset index

print(len(df.index.levels[0]))
```

Some columns has type Object, case these to explicit types.

```
[ ]: def convert_types(df):
    df = df.copy()
    df.loc[:, 'Volume'] = df.loc[:, 'Volume'].astype('int32')
    df.Open_Interest = df.Open_Interest.astype('int32')
    df.Binary_MACD_8_24 = df.Binary_MACD_8_24.astype('int32')
    df.Binary_MACD_16_48 = df.Binary_MACD_16_48.astype('int32')
    df.Binary_MACD_32_96 = df.Binary_MACD_32_96.astype('int32')
    return df

df = convert_types(df)
if 'Date_Col' in df.columns:
    df = df.drop('Date_Col', axis=1)


if 'Asset_Col' in df.columns:
    df = df.drop('Asset_Col', axis=1)

print(df.columns)
print("len columns",len(df.columns))
```

WaveNet for multivariate time series

Its only one change needed from the original WaveNet to add the capability to predict based on multiple times seires and that is to remove the residual on the first hidden layer.

Then a Multilayer Perceptron (MLP) with 256, and 64 connection is added to the last hidden layer of the WaveNet model, and finaly the next time step is predicted as a linear layer of size 1.

The input dimension is (batch-size x sequence-length x number of features). After the first hidden layer it changes to (batch-size x sequence-length x number of filters) and it no longer makes sence to talk about multiple features, but multiple filters.

The sequence length is 255 and this is the window of attention by the model. In the transition from WaveNet to MLP we use MaxPooling1D and Flatten to select the max feature across all filters for each data point in the sequence length. This layer transforms the tensor of shape (batch-size

x sequence-length x filter-length) to (batch-size x sequence-length x 1). Flatten removes the last dimension. This becomes the input to the first MLP layer of size 256.

[ ]:
```
#WaveNet - multivariate
import tensorflow as tf
from tensorflow.keras.layers import Conv1D, Conv2D, Activation, Add
from tensorflow.keras.layers import MaxPooling1D, Flatten, Dense
from tensorflow.keras.models import Model
from keras.optimizers import Adam

n_features = 24
seq_length = 255
input_shape = (seq_length, n_features)  # Input shape for a 1D waveform
num_filters = 256
filter_width = 2
num_layers = 8

def residual_block(x, dilation_rate):
    conv = Conv1D(filters=num_filters, kernel_size=filter_width,␣
 ↪dilation_rate=dilation_rate, padding='causal')(x)
    tanh_out = Activation('tanh')(conv)
    sigmoid_out = Activation('sigmoid')(conv)
    merged = tf.multiply(tanh_out, sigmoid_out)

    # 1x1 convolution to mix the results
    out = Conv1D(filters=num_filters, kernel_size=1, padding='same')(merged)

    if dilation_rate > 1:
        x = Add()([x, out])
    else:  #no residual connection on first layer
        x = out

    return x

input_layer = tf.keras.layers.Input(shape=input_shape)
x = input_layer
for i in range(num_layers):
    x = residual_block(x, 2**i)

output_layer =x
output_layer = MaxPooling1D(pool_size=2)(output_layer)
output_layer = Flatten()(output_layer)
output_layer = Dense(256, activation='relu')(output_layer)
output_layer = Dense(64, activation='relu')(output_layer)
output_layer = Dense(1, activation='linear')(output_layer)

model = Model(inputs=input_layer, outputs=output_layer)
```

```
optimizer = tf.keras.optimizers.Adam(learning_rate=0.00001)
model.compile(optimizer=optimizer, loss='mse')
model.summary()
```

Training loop

The function to start training is defined at the bottom of the cell. to train on all 43 assets define function to create rolling windows of a specified size from a given time series data. The window is of size 255 This transforms a 2D dataframe to a 3D numpy tensor of shape (number of windows x window size, time series lenght)

A training loop is defined which MinMax transforms the time series which was input pr instrument. The training set is defined as the datapoints from 0 to -1430, validation from -1430 to -715. And the last 715 datapoints is the test set. The time series to predict is masked. Finally, in a loop the model is pretrained on all instruments.

[ ]:
```
from sklearn.preprocessing import MinMaxScaler
from sklearn.metrics import mean_squared_error
batch_size = 96
epochs = 100

def create_window(mk1):
    window_size = 255
    shift = 1
    num_windows = (mk1.shape[0] - window_size) // shift + 1
    result = np.empty((num_windows, window_size, mk1.shape[1]))
    for i in range(num_windows):
        result[i] = mk1[i * shift:i * shift + window_size]
    return result

def train_model(train_x, train_y, val_x, val_y):
    callbackEarlyStop = tf.keras.callbacks.EarlyStopping(monitor='val_loss',
                                                         patience=20,␣
 ↪restore_best_weights=True)
    history = model.fit(
                        train_x, train_y,
                        validation_data=(val_x, val_y),
                        batch_size=batch_size, epochs=epochs, verbose=1,
                        callbacks=[callbackEarlyStop])
    return history, model

def train_window_on_a_asset(asset, doTrain):
    print("asset:",asset," has nan:",df.columns[df.isna().any()].tolist())
    mk1 = df.loc[asset, :]
    mk1 = mk1.copy()
    scaler = MinMaxScaler()
```

```
    scaled = scaler.
↪fit_transform(mk1[['Open','High','Low','close','Volume','Open_Interest']].
↪values)
    mk1.loc[asset,['Open','High','Low','close','Volume','Open_Interest']] =␣
↪scaled
    result = create_window(mk1)
    print("result:",result.shape, " val size:",(result.shape[0]*0.2))
    test = result[-715:,:,:]
    val = result[-1430:-715,:,:]
    train = result[:-1430,:,:]

    #Remove Next_Daily_Returns, y_hat, from training X
    exclude_indices_last_dim = np.array([9])
    mask_last_dim = np.ones(train.shape[-1], dtype=bool)
    mask_last_dim[exclude_indices_last_dim] = False

    train_y=train[:,254,9]
    train_x=train[:,:,mask_last_dim]
    val_y=val[:,254,9]
    val_x=val[:,:,mask_last_dim]
    test_y=test[:,254,9]
    test_x=test[:,:,mask_last_dim]
    print("train_x",train_x.shape)
    print("train_y",train_y.shape)
    print("val_x",val_x.shape)
    print("val_y",val_y.shape)
    print("test_x",val_x.shape)
    print("test_y",val_y.shape)
    print("***************"+str(doTrain))
    if doTrain==True:
        history, model = train_model(train_x, train_y, val_x, val_y)
        x = np.arange(0, len(train_y))
        x1 = np.arange(len(train_y), len(train_y)+len(val_y))
        x2 = np.arange(len(train_y)+len(val_y),␣
↪len(train_y)+len(val_y)+len(test_y))
        plt.plot(x, train_y, label="train")
        plt.plot(x1, val_y, label="validation")
        plt.plot(x2, test_y, label="test")
        plt.savefig(asset+'train_val_test.png')
        plt.close("all")
        print(history.history.keys())
        plt.plot(history.history['loss'][1:])
        plt.plot(history.history['val_loss'][1:])
        plt.xlabel('Epoch')
        plt.ylabel('Mean Absolute Error Loss')
        plt.title('Loss Over Time')
        plt.legend(['Train','Valid'])
```

```
        plt.savefig(asset+'_hist.png')
        plt.close("all")
        print("display prediction")
        pred_y = model.predict(test_x)
        plt.plot(mk1.index.values[-715:-615], pred_y[0:100], label="pred_y")
        plt.plot(mk1.index.values[-715:-615], test_y[0:100], label="true_y")
        mse = mean_squared_error(test_y, pred_y)
        plt.title('Test prediction, MSE: '+ str(mse))
        plt.savefig(asset+'_pred_test.png')
        plt.close("all")
    else:
        return train_x, train_y, val_x, val_y, test_x, test_y, mk1.index.
 ↪values[-1430:-715], mk1.index.values[-715:]

for asset in df.index.get_level_values(0).drop_duplicates():
    train_window_on_a_asset(asset, True)
```

Finetune-training, prediction and evaluting models using scorecards

The function to start training is defined at the bottom of the cell. This cell does the finetuning on each instrument, before predicting on the test set. The model weights are choosen as those which predicts with lowest MSE on the validation set and using early stopping. The test predictions are evaluated using a scorecard of metrics. The evaluation metrics are defined here: https://alphascientist.com/model_evaluation.html

- sign_pred: positive or negative sign of prediction
- sign_true: positive or negative sign of true outcome
- is_correct: 1 if sign_pred == sign_true, else 0
- is_predicted: 1 if the model has made a valid prediction, 0 if not. This is important if models only emit predictions when they have a certain level of confidence result: the profit (loss) resulting from betting one unit in the direction of the sign_pred. This is the continuous variable result of following the model

[ ]:
```
file_path = 'output_file.csv'

def make_df(y_pred,y_true):
    df = pd.concat([pd.Series(y_pred, name="y_pred"),pd.Series(y_true,␣
 ↪name="y_true")], axis=1)
    df['sign_pred'] = df.y_pred.apply(np.sign)
    df['sign_true'] = df.y_true.apply(np.sign)
    df['is_correct'] = 0
    df.loc[df.sign_pred * df.sign_true > 0 ,'is_correct'] = 1 # only registers␣
 ↪1 when prediction was made AND it was correct
    df['is_incorrect'] = 0
    df.loc[df.sign_pred * df.sign_true < 0,'is_incorrect'] = 1 # only registers␣
 ↪1 when prediction was made AND it was wrong
    df['is_predicted'] = df.is_correct + df.is_incorrect
    df['result'] = df.sign_pred * df.y_true
```

```
    return df

#Accuracy: Just as the name suggests,
#          this measures the percent of predictions that were directionally␣
 ↪correct vs. incorrect.
#Edge: perhaps the most useful of all metrics, this is the expected value
#      of the prediction over a sufficiently large set of draws.
#      Think of this like a blackjack card counter who knows the expected␣
 ↪profit
#      on each dollar bet when the odds are at a level of favorability
#Noise: critically important but often ignored, the noise metric estimates
#       how dramatically the model's predictions vary from one day to the next.
#       As you might imagine, a model which abruptly changes its mind every
#      few days is much harder to follow (and much more expensive to trade)␣
 ↪than one which is a bit more steady.

# y_true_chg and y_pred_chg: The average magnitude of change (per period) in␣
 ↪y_true and y_pred.
# prediction_calibration: A simple ratio of the magnitude of our predictions vs.
 ↪ magnitude of truth.
#                         This gives some indication of whether our model is␣
 ↪properly tuned to
#                         the size of movement in addition to the direction of␣
 ↪it.
# capture_ratio: Ratio of the "edge" we gain by following our predictions vs.␣
 ↪the actual daily change.
#               100 would indicate that we were perfectly capturing the true␣
 ↪movement of the target variable.

#edge_long and edge_short: The "edge" for only long signals or for short␣
 ↪signals.
#edge_win and edge_lose: The "edge" for only winners or for only losers.
def calc_scorecard(df):
    scorecard = pd.Series()
    # building block metrics
    scorecard.loc['accuracy'] = df.is_correct.sum()*1. / (df.is_predicted.
 ↪sum()*1.)*100
    scorecard.loc['edge'] = df.result.mean()
    scorecard.loc['noise'] = df.y_pred.diff().abs().mean()
    # derived metrics
    scorecard.loc['y_true_chg'] = df.y_true.abs().mean()
    scorecard.loc['y_pred_chg'] = df.y_pred.abs().mean()
    scorecard.loc['prediction_calibration'] = scorecard.loc['y_pred_chg']/
 ↪scorecard.loc['y_true_chg']
    scorecard.loc['capture_ratio'] = scorecard.loc['edge']/scorecard.
 ↪loc['y_true_chg']*100
```

```
    # metrics for a subset of predictions
    scorecard.loc['edge_long'] = df[df.sign_pred == 1].result.mean()  - df.
 ↪y_true.mean()
    scorecard.loc['edge_short'] = df[df.sign_pred == -1].result.mean()  - df.
 ↪y_true.mean()
    scorecard.loc['edge_win'] = df[df.is_correct == 1].result.mean()  - df.
 ↪y_true.mean()
    scorecard.loc['edge_lose'] = df[df.is_incorrect == 1].result.mean()  - df.
 ↪y_true.mean()

    return scorecard

def lastTrain(asset, count):
    train_x, train_y, val_x, val_y, test_x, test_y, val_idx, test_idx =␣
 ↪train_window_on_a_asset(asset, False)

    callbackEarlyStop = tf.keras.callbacks.EarlyStopping(monitor='val_loss',
                                                         patience=20,␣
 ↪restore_best_weights=True)
    history = model.fit(
                    train_x, train_y,
                    validation_data=(val_x, val_y),
                    batch_size=batch_size, epochs=epochs, verbose=1,
                    callbacks=[callbackEarlyStop])
    pred_y = model.predict(test_x).flatten()
    df = make_df(pred_y,test_y)
    scorecard = calc_scorecard(df)
    print(scorecard)
    scorecard_df = pd.DataFrame(scorecard)
    display(scorecard_df)
    theMode = 'a' # mode='a' appends to the existing file
    print("count:",count)
    if count==0:
        theMode = 'w' # mode='w' overwrites the file or creates a new one
    count+=1
    scorecard_df.to_csv(file_path, index=True, header=True, mode=theMode)
    with open(file_path, 'a') as file:
        file.write('Instrument: '+str(asset)+'\n')

count = 0
for asset in df.index.get_level_values(0).drop_duplicates():
    lastTrain(asset, count)
    count += 1
```