CHINESE STOCK PREDICTION USING DEEP NEURAL NETWORK

Feng Xiaolong
xlfeng886@163.com

ABSTRACT

In this research, we study the problem of Chinese stock market forecasting using traditional Neural Network methods, including Deep Feedforward Network, Convolution Neural Network(CNN), Recurrent Neural Network(RNN), Long Short-Term Memory(LSTM) and we have also integrate with the Bi-direction technology. The purpose of this research is not to find a state-of-art algorithm to make the prediction more precise. We focus on the primary method and explore the relationship between architecture and hyper-parameter(including learning rate, variety of activation ,and variety of the output target designed. Due to the advanced of the Deep Learning method in decades, there is much active research in the Time-Serial Analysis domain, but we rarely not sure whether they are work in reality. The real market is a very sophisticated ecosystem, and many accidents may occur, influent the market price. Even there is an efficient market hypothesis from Eugene Fama, but we should not so sure we can control the market by this theoretical analysis. Although we implement the underlying deep neural network architecture, the experiment has a reassuring result with 31.9% accuracy in the eight classes classification task and 68.6% accuracy in the binary classification task.

1 INTRODUCTION

1.1 NEURAL NETWORK AND DEEP LEARNING

Deep learning is a member of machine learning. It is a re-branded name for neural networks—a family of learning techniques that were historically motivated by the way computation works in the brain, and which can be described as learning of parameterized differentiable mathematical functions. The name deep-learning arises from the fact that many layers of these differentiable functions are often attached together. While all of the machine learning can be defined as learning to make forecasts based on past observations, deep learning approaches work by learning to not only predict but also to exactly represent the data, such that it is suitable for the forecast. Given a broad set of desired input-output mapping, deep learning approaches work by feeding the data into a network that produces progressive transformations of the input data until a final transformation predicts the output. The transformations produced by the network are learned from the given input-output mappings, such that each transformation makes it easier to relate the data to the desired label.

While the human designer is in charge of designing the network architecture and training regime, providing the network with a proper set of input-output examples, and encoding the input data in a proper way, a lot of the heavy-lifting of learning the exact representation is performed automatically by the network, supported by the network's architecture.

1.2 RECENT ADVANCES IN DEEP LEARNING

Deep Learning is one of the charming fields in recent years (Goodfellow u. a., 2016). The breakthrough of Deep Learning since the success of AlexNet (Creswell u. a., 2017) in 2012. Deep Learning is used in the domain of digital image processing to solve severe problems(e.g., image colorization, classification, segmentation, and detection). In that domain, methods such as Convolutional Neural Networks(CNN) mostly improve prediction performance using big data and unlimited computing resources and have pushed the boundaries of what was possible.

Deep Learning has pushed the boundaries of what was possible in the field of Machine Learning and Artificial Intelligence. Now and then, new and new deep learning techniques are being yielded, exceeding state-of-the-art deep learning techniques. Deep Learning is evolving at a considerable speed, its kind of hard to keep track of the regular advances. In our experiment, we are going to briefly review the basic idea in Deep Learning used in the Chinese stock market.

However, that is not so to say that the traditional Machine Learning techniques (Cao u. Tay, 2003),(Kercheval u. Yuan, 2015),(Zhai u. a., 2007),(Armano u. a., 2005) which had been undergoing continuous improvement in years before the rise of Deep Learning have become obsolete. But in this paper, we will not review the advantages and disadvantages of each approach. We only focus on the method of Deep Learning applied to the time series problem.

1.3 Financial Market Prediction and Time Series Analysis

Modeling and Forecasting of the financial market have been a trendy topic for scholars and researchers. For the efficient markets hypothesis, proposed by Eugene Fama, who is one of the winners of Nobel economics prize in 2013. Although we are not sure about Fama's theory, we should know that Lars peter Hansen shared the same prize at the same time. That mean peoples are confused the question about whether the market could be predicted. Nevertheless, the application for the time series problem is not only used in the market, it also could be used in weather forecasting, statistics, signal processing, pattern recognition, earthquake prediction, electroencephalography, control engineering, astronomy, communications engineering, and primarily in any field applied science and engineering which requires temporal measurements.

Time series analysis comprises methods for analyzing time-series data in order to extract meaningful statistics and other characteristics of the data. Time series forecasting is the application of a model to predict future values based on previously observed values.

1.4 Dataset Description

Modern Deep Network method is supported by big data and high-speed data process capability. For the aspect of the data, stock market data is the natural source of the research target. Notably, we dived into the stock market data in China. To find out how the stock market works in China, we should be at first mentioned where the stocks are traded. Thus, we can trade Chinese stocks on the most popular stock exchanges of China: Shanghai Stock Exchange (SSE) and the Shenzhen Stock Exchange (SZSE).

In our experiment, we collect all the daily trade data in the past three decades. The data starts with the first stock released and ends in December 2019. We do not consider the subset of all stocks, but all the stocks data are considered in our experiment. Considering all stocks helps us analyze the internal laws of the Chinese stock market. Because of the difficulty of data processing, we do not use the minute level and hour level data but use the transaction day level data. Although the time scale of the amount of data we use is large, the amount of data at this scale still exceeds 10 million records.

In order to have an intuitive understanding of China's stock market, we draw the trend chart of the shanghai composite index in Fig. 1.

Figure 1: The history shanghai composite index.

2 Related Work

Financial time-series prediction is vital for developing excellent trading strategies in the financial market (Wang, 2014). In past decades, it has attracted much attention from researchers of many fields, especially the Machine Learning community (Lee u. Ready, 1991). These researches mainly focus on a specific market, e.g., the stock market (Kim, 2003)(Li u. a., 2016), the foreign exchange market (Lin u. a., 2017),(Atsalakis u. Valavanis, 2009), and the futures market (Cheung u. a., 2019) ,(Kim u. a., 2017). Unsurprisingly, this is a holy grail challenge in the finance field due to their irregular and noisy environment.

From the perspective of the learning target, existing researches can be divided into the regression approaches and classification approaches. For the regression approaches, they treat this task as a regression problem (Zirilli, 1996),(Bollen u. a., 2011), aiming to predict the future value of financial time-series. While the classification-oriented approaches treat this as a classification problem (Schumaker u. Chen, 2009),(Hsieh u. a., 2011) focusing on financial time-series classification (FTC).

In most cases, the classification approaches achieve higher profits than the regression ones (Huang u. a., 2008). Accordingly, the effectiveness of various approaches in FTC has been widely explored (Li u. a., 2016),(Leung u. a., 2000). There were also many overview papers on Deep Learning (DL) in the past years. They described DL methods and approaches in significant ways as well as their applications and directions for future research.

In (Young u. a., 2017), the researchers talked about DL models and architectures, mainly used in Natural Language Processing (NLP). They showed DL applications in various NLP fields, compared DL models, and discussed possible future trends. Furthermore, in (Goodfellow u. a., 2016), the researchers discussed deep networks and generative models in detail. Starting from Machine Learning (ML) basics, pros, and cons for deep architectures, they concluded recent DL researches and applications thoroughly.

3 Advances In Deep Learning

In this section, we will discuss the leading recent Deep Learning (DL) approaches derived from Machine Learning and brief evolution of Artificial Neural Networks (ANN), which is the most common form used for deep learning.

3.1 Architecture

3.1.1 Multi Layer Perceptron

As we abandon the brain metaphor and describe networks exclusively in terms of vector-matrix operations. The simplest neural network is called a perceptron. It is simply a linear model is:

$$NN_{Perceptron}(x) = xW + b \tag{1}$$

$$s.t. \quad x \in R^{d_{in}}, W \in R^{d_{in}d_{out}}, b \in R^{d_{out}} \tag{2}$$

Where W is the weight matrix, and b is a bias term. In order to go beyond linear functions, we introduce a nonlinear hidden layer, resulting in the Multi Layer Perceptron. A feed-forward neural network with two hidden-layer has the form as :

$$MLP_2(x) = (g^2(g^1(xW^1 + b^2)W^2 + b^2))W^3 \tag{3}$$

$$s.t. \quad x \in R^{d_{in}}, W^1 \in R^{d_{in} \times d_1}, b^1 \in R^{d_1}, W^2 \in R^{d_1 \times d_2}, b^2 \in R^{d_2} \tag{4}$$

Where g is the nonlinear function, which could be relu, tanh, sigmoid, and so on.

3.1.2 The CNN architecture

The convolution-and-pooling (also called convolutional neural networks, or CNNs) architecture, which is tailored to this modeling problem. A convolutional neural network is designed to identify indicative local predictors in a large structure, and to combine them to produce a fixed-size vector representation of the structure, capturing the local aspects that are most informative for the prediction task at hand.

The CNN is, in essence, a feature-extracting architecture. It does not constitute a standalone, useful network on its own, but rather is meant to be integrated into a more extensive network, and to be trained to work in tandem with it in order to produce an end result. The CNN layer's responsibility is to extract meaningful sub-structures that are useful for the overall prediction task at hand.

Convolution-and-pooling architectures evolved in the neural network's vision community, where they showed great success as object detectors—recognizing an object from a predefined category ("cat," "bicycles") regardless of its position in the image (Wang u. Choi, 2013). When applied to images, the architecture is using 2D (grid) convolutions. When applied to the time series problem, we have mainly concerned with 1D (sequence) convolutions. Because of their origins in the computer-vision community, a lot of the terminology around convolutional neural networks is borrowed from computer vision and signal processing, including terms such as filter, channel, and receptive-field. We can have an intuitive understanding of CNN from Fig. 2 .

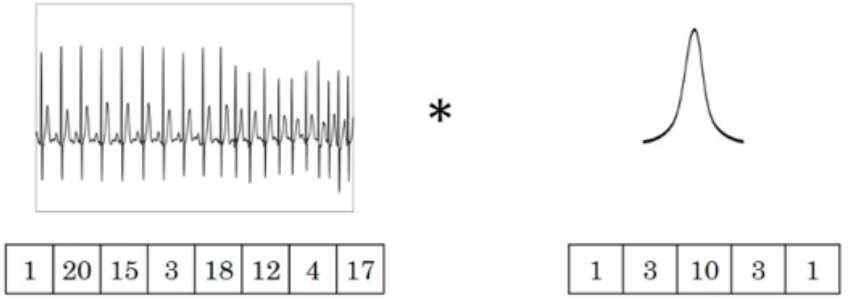

Figure 2: A brief figure for CNN Network Archriculture. This is a 1D Convolution Network, the input in the figure has a data dimension of 8, and the filter has a dimension of 5, and the output data dimension after convolution is 4

3.1.3 THE RNN ARCHITECTURE

As describes RNNs as an abstraction: an interface for translating a sequence of inputs into a fixed sized output that can then be plugged as components in larger networks. Various architectures that use RNNs as a component are discussed. We use $x_{i:j}$ to denote the sequence of vectors $x_i, \cdots, x_j$. On a high-level, the RNN is a function that takes as input an arbitrary length ordered sequence of n d_{in} dimensional vectors $x_{1:n} = x_1, x_2 \cdots x_n$,($x_i \in R^{d_{in}}$) and returns as output a single d_{out} dimensional vector $y_n \in R^{d_{oiut}}$:

$$y_n = RNN(x_{1:n}). \tag{5}$$

$$s.t. \quad x \in R^{d_{in}}, y_n \in R^{d_{out}} \tag{6}$$

This implicitly defines an output vectory_i for each prefix $x_{1:i}$ of the sequence x $x_{1:n}$. We denote by RNN^* the function returning this sequence:

$$y_{1:n} = RNN^*(x_{1:n}) \tag{7}$$

$$y_i = RNN(x_{1:i}) \tag{8}$$

$$s.t. x_i \in R^{d_{in}}, y_n \in R^{d_{out}} \tag{9}$$

The output vector y_n is then used for further prediction. For example, a model for predicting the conditional probability of an event e given the sequence $x_{1:n}$ can be defined as the equation below.

$$p(e = j|x_{1:n}) = softmax(RNN(x_{1:n}) \cdot W + b)_{[j]} \tag{10}$$

The j_{th} element in the output vector resulting from the softmax operation over a linear transformation of the RNN encoding.

$$y_n = RNN(x_{1:n}) \tag{11}$$

We can denote the RNN as the form of recursive as Fig. 3.

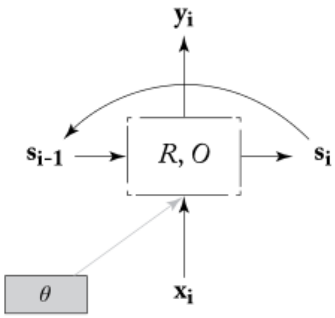

Figure 3: A brief figure for RNN Network Archriculture.

Bidirectional RNNs (BIRNN) A useful elaboration of an RNN is a bidirectional-RNN (also commonly referred to as biRNN) [20]. Consider an input sequence $x_{1:n}$. The biRNN works by maintaining two separate states, s_i^f , and s_j^b for each input position i . The forward state s_i^f is based on $x_1, x_2 \cdots \cdot x_i$, while the backward state s_j^b is based on $x_n, x_{n-1} \cdots \cdot x_i$. The forward and backward states are generated by two different RNNs. The first $RNN(R^f, O^f)$ is fed the input sequence $x_{1:n}$ as is, while the second $RNN(R^b, O^b)$ is fed the input sequence in reverse. The state representation s_i is then composed of both the forward and backward states. The output at position i is based on the concatenation of the two output vectors $y_i = [y_i^f : y_i^b] = [O^f(s_i^f) : O^b(s_i^b)]$, taking into account both the past and the future. In other words, y_i , the biRNN encoding of the i_{th} word in a sequence is the concatenation of two RNNs, one reading the sequence from the beginning, and the other reading it from the end. We define $biRNN(x_{1:n}, i)$ to be the output vector corresponding to the i_{th} sequence position:

$$biRNN(x_{1:n}, i) = y_i = [RNN^f(x_{1:i}) : RNN^b(x_{x_{n:i}})] \tag{12}$$

The vector y_i can then be used directly for prediction or fed as part of the input to a more complex network. While the two RNNs are run independently of each other, the error gradients at position i will flow both forward and backward through the two RNNs. Feeding the vector y_i through an MLP prior to prediction will further mix the forward and backward signals. Visual representation of the biRNN architecture is given in Fig. 4.

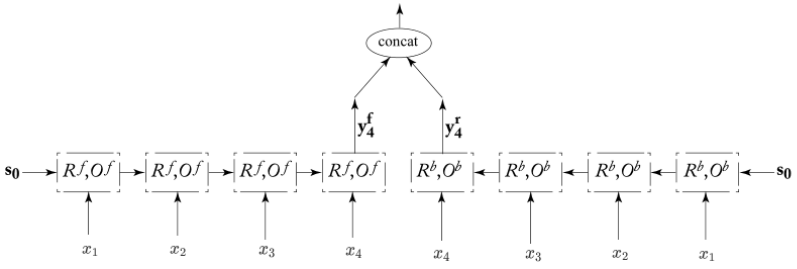

Figure 4: A brief figure for BIRNN Network Archriculture.

Multi-Layer(Stacked) RNNs RNNs can be stacked in layers, forming a grid (Hihi u. Bengio, 1995). Consider k RNNs, $RNN_1 \cdots \cdot RNN_k$, where the j_{th} RNN has states $s_{1:n}^j$ and outputs $y_{1:n}^j$. The input for the first RNN is $x_{1:n}$, while the input of the j_{th} $RNN(j \geq 2)$ are the outputs of the RNN below it $y_{1:n}^{j-1}$, The output of the entire formation is the output of the last RNN, $y_{1:n}^k$. Such layered architectures are often called deep RNNs. A visual representation of a three-layer RNN is given in Fig. 5..

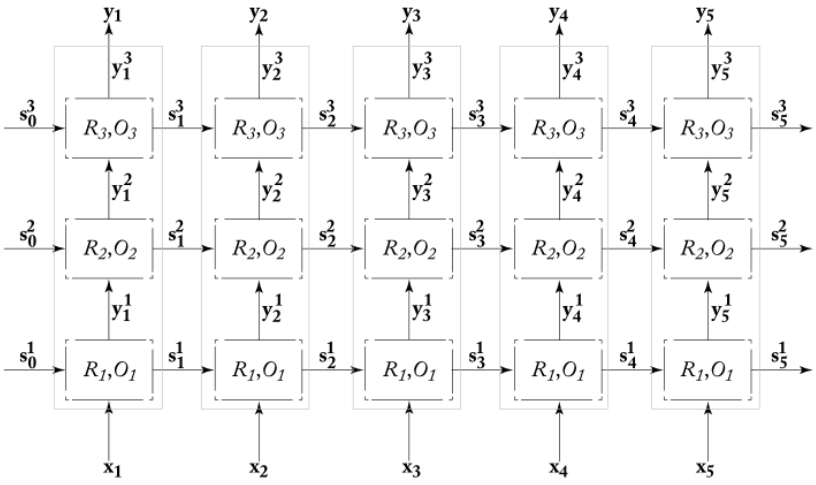

Figure 5: A brief figure for Multi-Layer(Stacked) RNNs Network Archriculture.

Simple RNN The simplest RNN formulation that is sensitive to the ordering of elements in the sequence is known as an Elman Network or Simple-RNN (S-RNN). The S-RNN was proposed by (Elman, 1990). The S-RNN takes the following form:

$$s_i = R_{SRNN}(x_i, s_{i-1}) = g(s_{i-1}W^s + x_iW^x + b) \tag{13}$$
$$y_i = O(s_i) \tag{14}$$
$$s.t. \quad s_i, y_i \in R^{d_s}, x_i \in R^{d_x}, W^x \in R^{d_x \times d_s}, W^s \in R^{d_s \times d_s}, b \in R^{d_s} \tag{15}$$

That is, the state s_{i-1} and the input x_i are each linearly transformed, the results are added(together with a bias term) and then passed through a nonlinear activation function g (commonly tanh or ReLU). The output at position i is the same as the hidden state in that position.

Gated Architectures The S-RNN is hard to train effectively because of the vanishing gradients problem (Pascanu u. a., 2012). Error signals (gradients) in later steps in the sequence diminish quickly in the back-propagation process and do not reach earlier input signals, making it hard for the S-RNN to capture long-range dependencies. Gating-based architectures, such as the LSTM (Hochreiter u. Schmidhuber, 1997) and the GRU (Cho u. a., 2014a) are designed to solve this deficiency.

LSTM The Long Short-Term Memory (LSTM) architecture (Hochreiter u. Schmidhuber, 1997) was designed to solve the vanishing gradients problem and is the first to introduce the gating mechanism. The LSTM architecture explicitly splits the state vector s_i into two halves, where one half is treated as "memory cells" and the other is working memory. The memory cells are designed to preserve the memory, and also the error gradients, across time, and are controlled through differentiable gating components —smooth mathematical functions that simulate logical gates. At each input state, a gate is used to decide how much of the new input should be written to the memory cell, and how much of the current content of the memory cell should be forgotten. Mathematically, the LSTM architecture

is defined as:

$$s_j = R_{LSTM}(S_{j-1}, x_j) = [c_j : h_j] \tag{16}$$
$$c_j = f \odot c_{j-1} + i \odot z \tag{17}$$
$$h_j = o \odot tanh(c_j) \tag{18}$$
$$i = \delta(x_j W^{xi} + h_{j-1} W^{hi}) \tag{19}$$
$$f = \delta(x_j W^{xf} + h_{j-1} W^{hf}) \tag{20}$$
$$o = \delta(x_j W^{xo} + h_{j-1} W^{ho}) \tag{21}$$
$$z = tanh(x_j W^{xz} + h_{j-1} W^{hz}) \tag{22}$$
$$y_j = O_{LSTM}(s_j) = h_j \tag{23}$$
$$s.t.\ \ s_j \in R^{2 \cdot d_h}, x_i \in R^{d_x}, c_j, h_j, i, f, o, z \in R^{d_h}, W^s \in R^{d_s \times d_s}, W^{h_0} \in R^{d_h \times d_h} \tag{24}$$

The state at time j is composed of two vectors, c_j, and h_j, where c_j is the memory component, and h_j is the hidden state component. There are three gates, i , f , and o , controlling for input, forget, and output. The gate values are computed based on linear combinations of the current input $x_j$and the previous state h_{j-1} , passed through a sigmoid activation function. An update candidate z is computed as a linear combination of x_j and h_{j-1} , passed through a tanha ctivation function. The memory c_j is then updated: the forget gate controls how much of the previous memory to keep $f \odot c_{j-1}$, and the input gate controls how much of the proposed update to keep $i \odot z$. Finally, the value of h_j (which is also the output y_j) is determined based on the content of the memory c_j, passed through a tanh nonlinearity, and controlled by the output gate. The gating mechanisms allow for gradients related to the memory part c_j to stay high across very long time ranges. LSTMs are currently the most successful type of RNN architecture, and they are responsible for many state-of-the-art sequence modeling results. The main competitor of the LSTM-RNN is the GRU, to be discussed next.

GRU The LSTM architecture is instrumental, but also quite complicated. The complexity of the system makes it hard to analyze and also computationally expensive to work in practice. The gated recurrent unit (GRU) was recently introduced by (Cho u. a., 2014b) as an alternative to the LSTM. It was subsequently shown by (Chung u. a., 2014) to perform comparably to the LSTM on several datasets.

Like the LSTM, the GRU is also based on a gating mechanism, but with substantially fewer gates and without a separate memory component.

$$s_j = R_{GRU}(s_{j-1}, x_j) = (1 - z) \odot s_{j-1} + z \odot \widetilde{s}_j \tag{25}$$
$$z = \delta(x_j W^{xz} + s_{j-1} W^{sz}) \tag{26}$$
$$r = \delta(x_j W^{xr} + s_{j-1} W^{sr}) \tag{27}$$
$$\widetilde{s}_j = tanh(x_j W^{xs} + (r \odot s_{j-1}) W^{sg}) \tag{28}$$
$$y_j = O_{GRU}(s_j) = s_j \tag{29}$$
$$s.t..\ \ s_j, \widetilde{s}_j \in R^{d_s}, x_i \in R^{d_x}, z, r \in R^{d_s}, W^{xo} \in R^{d_x \times d_s}, W^s \in R^{d_s \times d_s}, W^{so} \in R^{d_s \times d_s} \tag{30}$$

One gate (r) is used to control access to t()dhe previous state s_{j-1} and compute a proposed update $\widetilde{s}_j$. The updated state s_j (which also serves as the output y_j) is then determined based on an interpolation of the previous state s_{j-1} and the proposal $\widetilde{s}_j$, where the proportions of the interpolation are controlled using the gate z.

3.2 Activation Function

There are some major types of neurons that are used in practice that introduce nonlinearities in their computations. As the list below is the most common activation function :

(a) Sigmoid neurons (Elfwing u. a., 2017) :

$$\delta(x) = \frac{1}{1 + e^{-x}} \tag{31}$$

(b) Tanh neurons (Nwankpa u. a., 2018) :

$$y = tanh(x) \tag{12}$$

(c) ReLU neurons (Agarap, 2018) :

$$y = max(0, x) \tag{32}$$

(d) Leaky ReLU neurons (Zhang u. a., 2017):

$$y = max(0.1x, x) \tag{33}$$

(e) Maxout neurons (Goodfellow u. a., 2013):

$$y = max(w_1^T x + b_1, w_2^T x + b_2) \tag{34}$$

(f) ELU neurons (Clevert u. a., 2015) :

$$y = \begin{cases} x & x \geq 0 \\ \alpha(e^x - 1) & x < 0 \end{cases} \tag{35}$$

3.3 Cost Function and Regularization

An important aspect of the design of a deep neural network is the choice of the cost function. Fortunately, the cost functions for neural networks are more or less the same as those for other parametric models, such as linear models. In most cases, our parametric model defines a distribution $p(y|;\theta)$, and we use the principle of maximum likelihood. This means we use the cross-entropy between the training data and the model's predictions as the cost function.

Let $y = y_{[1]} \cdots \cdot y_{[n]}$ be a vector representing the true multinomial distribution over the labels $1.....n$, and let $\hat{y} = \hat{y}_{[1]} \cdots \cdot \hat{y}_{[n]}$ be the linear classifier's output, which was transformed by the softmax function, and represent the class membership conditional distribution $\hat{y}_{[i]} = P(y = i|x)$. The categorical cross entropy loss measures the dissimilarity between the true label distribution y and the predicted label distribution $\hat{y}$, and is defined as cross entropy (Amos u. Yarats, 2019):

$$L_{cross-entropy}(\hat{y}, y) = -\sum_i y_{[i]} log(\hat{y}_{[i]}) \tag{36}$$

The total cost function used to train a neural network will often combine one of the primary cost functions described here with a regularization term to avoid overfitting issues.

$$\hat{\Theta} = \arg\min_{\Theta} L(\Theta) + \lambda(R(\Theta)) \tag{37}$$

$$= \arg\min_{\Theta} \frac{1}{n} \sum_{i=1}^{n} L(f(x_i; \Theta)) + \lambda R(\Theta) \tag{38}$$

The weight decay approach (Nakamura u. Hong, 2019) used for linear models is also directly applicable to deep neural networks and is among the most popular regularization strategies (Murugan u. Durairaj, 2017).

3.4 Optimization

In order to train the model, we need to solve the optimization problem. A standard solution is to use a gradient-based method. Roughly speaking, gradient-based methods work by repeatedly computing an estimate of the loss L over the training set, computing the gradients of the parameters , with respect to the loss estimate, and moving the parameters in the opposite directions of the gradient. The different optimization methods differ in how the error stimulate is computed, and how "moving in the opposite direction of the gradient" is defined. While the SGD algorithm can and often does produce good results, more advanced algorithms are also available. The SGD+Momentum (Duda, 2019) and Nesterov Momentum algorithms (Botev u. a., 2016) are variants of SGD in which previous gradients are accumulated and affect the current update. Adaptive learning rate algorithms, including AdaGrad (Ward u. a., 2018), AdaDelta (Zeiler, 2012), RMSProp (Ruder, 2016), and Adam are designed to select the learning rate for each minibatch, sometimes on a per-coordinate basis, potentially alleviating the need of fiddling with learning rate schedules.

3.5 BACK-PROPAGATION

In the Deep Learning regime, we assumed the function is differentiable, and we can explicitly compute its derivative. In practice, a neural network function consists of many tensor operations chained together, each of which has a simple, known derivative.

Applying the chain rule to the computation of the gradient values of a neural network gives rise to an algorithm called Backpropagation (also sometimes called reverse-mode differentiation). Back-propagation starts with the final loss value and works backward from the top layers to the bottom layers, applying the chain rule to compute the contribution that each parameter had in the loss value.

Nowadays, people will implement networks in modern frameworks that are capable of symbolic differentiation, such as TensorFlow (Abadi u. a., 2016) and PyTorch. This means that, given a chain of operations with a known derivative, they can compute a gradient function for the chain (by applying the chain rule) that maps network parameter values to gradient values. When we have access to such a function, the backward pass is reduced to a call to this gradient function.

3.6 HYPER-PARAMETERS

Deep learning (DL) systems expose many tuning parameters ("hyper-parameters") that affect the performance and accuracy of trained models. Increasingly users struggle to configure hyperparameters, and a substantial portion of time is spent tuning them empirically. Here we review the most critical part of the hyper-parameters.

Mini-Batch Gradient Descent Hyperparameters First, we describe the hyperparameters of mini-batch gradient descent (Peng u. a., 2019), which updates the network's parameters using gradient descent on a subset of the training data (which is periodically shuffled, or assumed infinite). We'll define the $t_{th} < T$ mini-batch gradient descent update of network parameters θ as :

$$\theta^{(t)} \leftarrow \theta^{(t-1)} - \epsilon_t \frac{1}{B} \sum_{t'=Bt+1}^{B(t+1)} \frac{\partial L(z_{t'}, \theta)}{\partial \theta} \tag{39}$$

where $z_{t'}$ is example t' in the training set and the hyperparameters are the loss function L, the learning rate at step t ϵ_t, the mini-batch size B, and the number of iterations

$$T$$

. Note that we must define θ^0, which is also a hyperparameter. For a specific optimization method, as we describe the momentum, it helps to "smooth" the gradient updates using a leaky integrator filter with parameter β by

$$\bar{g} \leftarrow (1-\beta)\bar{g} + \beta \frac{\partial L(z_{t'}, \theta)}{\partial \theta} \tag{40}$$

$\bar{g}$ can then be used in place of the "true" gradient update in gradient descent. Some mathematically motivated approaches can ensure much faster convergence when using appropriate momentum; however, for pure stochastic gradient descent, standard gradient updates $\beta = 1$ with a harmonically decreasing learning rate is optimal.

Model Hyperparameters The structure of the neural network itself involves numerous hyperparameters in its design, including the size and nonlinearity of each layer. The numeric properties of the weights are often also constrained in some way, and their initialization can have a substantial effect on model performance. Finally, the preprocessing of the input data can also be essential for ensuring convergence. As a practical note, many hyperparameters can vary across layers.

(a) Number of hidden units

(b) Weight decay, the purpose is to reduce overfitting with the regularization term.

(c) Activation sparsity, it may be advantageous for the hidden unit activations to be sparse. We should consider the type of penalty, L1, L2, or combined L1 and L2.

(d) Nonlinearity, we also should select the type of activation function, as we have described in the previous chapter.

(e) Weight initialization, biases are typically initialized to 00, but weights must be initialized carefully. Their initialization can have a significant impact on the local minimum found by the training algorithm.

(f) Random seeds (Colas u. a., 2018) and model averaging. Many of the processes involved in training a neural network involve using a random number generator (e.g., random sampling of training data, weight initialization.). As a result, the seed passed to the random number generator can have a slight effect on the results. However, a different random seed can produce a non-trivially different model (even if it performs about as well). As a result, it is common to train multiple models with multiple random seeds and use model averaging (bagging, Bayesian methods) to improve performance.

(g) Preprocess input data, the statistics of the input data can have a substantial effect on network performance. Element-wise standardization (subtract the mean and divide by the standard deviation), Principal Component Analysis, uniformization (transform each feature value to its approximate normalized rank or quantile), and nonlinearities such as the logarithm or square root are common.

Hyperparameters Space Exploration The number of hyperparameters delineated above indicates that there are a substantial number of choices to be made when creating a neural network learner and that these choices will affect the success and failure of the model. In order to ensure reproducibility of results, a principled approach should be used for setting hyperparameters, or, at the very least, they should be explicitly stated in the model description.

There is some suggestion we can take when we explore the hyperparameters, including coordinate descent, grid search, and random search.

4 Experiments Setup

4.1 Tensorflow and Keras

In our experiment, we use Tensorflow and Keras to build and train the model. Tensorflow is a numeric computing framework and machine learning library. It is the most famous framework for the implementation of Deep Learning. Keras is a high-level neural networks API, written in Python and capable of running on top of TensorFlow, CNTK, or Theano. It was developed with a focus on enabling fast experimentation. Being able to go from idea to result with the least possible delay is key to doing proper research.

4.2 feature select

In the initial version of the data, we have collected 48 features for each stock. In the experiment stage, we find that the number of features does not affect the accuracy too much and reduce to 10 features as we described the dataset above.

To have the intuitive sense of the dataset, here we post a data fragment for the stock code sh600033 in Table 1.

Each row represents one day in one market day, columns from left to right are the date, stock code name, opening price, day high price, day low price, close price, changes, volumes, five days moving average, ten days moving average and twenty days moving average.

4.3 Label Design

We divide the experiment into two different problems, binary classification and multiclass classification problem.

In our binary classification problem, we design the output as two output labels. Label 0 represent the stock in which the price rose, and label 1 represent the stock price fell. Out network always be

Data	Code	Open	High	Low	Close	Change	Volumes	MA_5	MA_10	MA_20
2018/2/2	sh600033	3.68	3.72	3.62	3.71	0.008152	14190281	3.748	3.783	3.741
2018/2/5	sh600033	3.69	3.71	3.65	3.7	-0.0027	12766982	3.726	3.772	3.741
2018/2/6	sh600033	3.67	3.68	3.58	3.58	-0.03243	14442600	3.684	3.751	3.7355
2018/2/7	sh600033	3.63	3.64	3.55	3.56	-0.00559	10008747	3.646	3.723	3.7295
2018/2/8	sh600033	3.59	3.6	3.56	3.57	0.002809	7135401	3.624	3.699	3.724
2018/2/9	sh600033	3.54	3.55	3.35	3.41	-0.04482	16922628	3.564	3.656	3.71
2018/2/12	sh600033	3.42	3.47	3.42	3.45	0.01173	5891331	3.514	3.62	3.699
2018/2/13	sh600033	3.47	3.51	3.46	3.47	0.005797	5348475	3.492	3.588	3.689
2018/2/14	sh600033	3.49	3.49	3.46	3.49	0.005764	3863871	3.478	3.562	3.6765

Table 1: We report the data fragment for the stock code sh600033 from 2018-2-2 to 2018-2-14.

ended with a Dense layer with one unit and a sigmoid activation. The output of our network is a scalar between 0 and 1, encoding a probability.

In our multiclass classification problem, we design the output as 8 classes and a single label. Our network ended with a softmax activation so that it will output a probability distribution over the 8 output classes. Categorical crossentropy is the default loss function for our multiclass classification problem. It minimizes the distance between the probability distributions output by the network and the true distribution of the targets. In the Chinese stock market, the maximum daily increase of each stock is 10

$$Class1 \in [-0.1, -0.075) \tag{41}$$
$$Class2 \in [-0.075, -0.05) \tag{42}$$
$$Class3 \in [-0.05, -0.025) \tag{43}$$
$$Class4 \in [-0.025, 0) \tag{44}$$
$$Class5 \in [0, 0.025) \tag{45}$$
$$Class6 \in [0.025, 0.05) \tag{46}$$
$$Class7 \in [0.05, 0.075) \tag{47}$$
$$Class8 \in [0.075, 0.1] \tag{48}$$

4.4 Model design

we implement the experiment by considering various networ architectures, including MLP, CNN, SimpleRNN, GRU, and LSTM.

(a) In MLP, we mainly use 512 neurons units and following the dropout layer with 0.5 to drop. We also considered different numbers of dense layers in our design.

(b) For CNN, we use the 1D convolution neural layer, with the different filters, kernel size, strides.

(c) For SimpleRNN, LSTM, and GRU: we try different units and activation functions.

(d) For BIRNN, we use the bi-direction trick for all the RNN models.

4.5 Main hyperparameter design

Learning rate: In our experiment, the RMSprop optimizer is the default choice. In some cases, we select some specific learning rate for the RMSprop optimizer, but we let it automated running for most of the cases. We also used the callback function to reduce the learning rate when the validation loss has stopped improving.

Activation function: we only consider ReLU and tanh as our activation function pool in the nonoutput layers. In the output layer, we use sigmoid for the binary classification problem and for the multiclass classification problem.

Epochs: we set the maximum epochs as 200 and use the early stop trick to interrupt training once the target metric being monitored has stopped improving for a various number of epochs.

Lookback and delay steps: The exact formulation of the stock prediction problem will be given data going as far back as lookback timesteps (a timestep is one trading day), we set the lookback trading days as 20 and consider the output in one delayed trading day.

Batch size: we set the batch size as 1024 in our whole experiment.

Missing data process: Normally, there are many tricks for missing data process method. For example, fill with average value, last value, min value, max value, or zero. In our experiment, we just fill the missing data as zero.

We list all the architecture in our experiment in Table 2.

ID	Architecture Detail	Total Parameters
1	MLP(512)-LSTM(32)-LSTM(64)-MLP(512)-MLP(8)	138120
2	MLP(512)-Bidirectional[LSTM(32)]-MLP(512)-MLP(8)	183048
3	[MLP(512)]x6-Flaten-MLP(8)	1401352
4	[MLP(512)]x3-Flaten-MLP(8)	613384
5	Flatten-MLP(512)-MLP(8)	117256
6	MLP(512)-Flaten-MLP(8)	88072
7	MLP(512)-Conv1D(32,5)-MaxPooling1D(5)-Conv1D(32,3)-MaxPooling1D-MLP(8)	91464
8	MLP(512)-Conv1D(128,3)-MaxPooling1D(5)-Conv1D(128,3)-MaxPooling1D-MLP(8)	253192
9	MLP(512)-Conv1D(512,3)-MaxPooling1D(5)-Conv1D(512,3)-MaxPooling1D-MLP(8)	1584136
10	MLP(512)-Conv1D(1024,3)-MaxPooling1D(5)-Conv1D(1024,3)-MaxPooling1D-MLP(8)	5783560
11	MLP(512)-[Conv1D(32,3),MaxPooling1D(2)]x2-Conv1D(32,3)-MaxPooling1D-MLP(8)	61800
12	MLP(512)-[Conv1D(128,3),MaxPooling1D(2)]x2-Conv1D(128,3)-MaxPooling1D-MLP(8)	302472
13	MLP(512)-[Conv1D(512,3),MaxPooling1D(2)]x2-Conv1D(512,3)-MaxPooling1D-MLP(8)	2371080
14	MLP(512)-GRU(32)-MLP(512)-MLP(8)	79464
15	MLP(512)-SimpleRNN(32)-MLP(512)-MLP(8)	44584
16	MLP(512)-LSTM(32)-MLP(512)-MLP(8)	96.904
17	MLP(512)-LSTM(32)x2-MLP(512)-MLP(8)	105224
18	MLP(512)-Bidirectional[(LSTM(32)x2]-MLP(512)-MLP(8)	207880
19	MLP(512)-LSTM(32,4)-MLP(512)-MLP(8)	121864
20	MLP(512)-Bidirectional[(LSTM(32)x4]-MLP(512)-MLP(2)	257544
21	MLP(512)-(LSTM(32)x4-MLP(512)-MLP(8)	117768
22	MLP(512)-Bidirectional[(LSTM(32)x4]–MLP(512)-MLP(8)	257554
23	MLP(512)-Bidirectional[(LSTM(32)x8]-MLP(512)-MLP(8)	356872
24	MLP(512)-Bidirectional[(LSTM(128)x8]-MLP(512)-MLP(2)	3557896
25	MLP(512)-Conv1D(512,3)-MaxPooling1D(5)-Bidirectional[LSTM(32)]-MLP(8)	2899720
26	MLP(512)-LSTM(32,4)-MLP(512)-BatchNormalization-MLP(8)	126472
27	MLP(512)-Bidirectional[(LSTM(32)x4]-MLP(512)-BatchNormalization-MLP(8)	262664

Table 2: Comparison of all the basic architecture in the experiment. The second column report the architecture details, and the last column report the total number of parameters for each architecture.

In order to have an intuitive understanding of the network architecture, We report the architecture of network number 21 from Table 2 and show the detail in Table 3.

Layer (type)	Output Shape	Parameter Number
dense_1 (Dense)	(None, None, 512)	2048
dropout_1 (Dropout)	(None, None, 512)	0
lstm_1 (LSTM)	(None, None, 32)	69760
lstm_2 (LSTM)	(None, None, 32)	8320
lstm_3 (LSTM)	(None, None, 32)	8320
lstm_4 (LSTM)	(None, 32)	8320
dense_2 (Dense)	(None, 512)	16896
dropout_2 (Dropout)	(None, 512)	0
dense_3 (Dense)	(None, 8)	4104

Table 3: We report the architecture detail for the number 21 network in Table 2, which have 117,768 parameters in total.

4.6 Environment

Most modern neural network implementations are based on graphics processing units(GPU). GPU is specialized hardware components that were originally developed for graphics applications, but the rise of GPU also dramatically improve the efficiency of neural network training (Wang u. a., 2019). Nevertheless, in the daily applications, using GPU is a very high-cost solution, so we also investigated the case of training small neural networks on the CPU. We create some VMs for theses case with the specifications as 6 vCPU and 24G memory.

5 Result and Analysis

Our experiment result performance is compared with the baseline models. We achieve 0.319 accuracies in the eight classes classification task and achieve 0.686 accuracies in the binary classification task.

In our research work, we have tried more than 200 test cases with different architecture, hyperparameters, and output design. We also give a brief overview of the CPU running time for each model.

5.1 Architecture Compare

From the result of the experiment, we find in our designed architecture, and there is a tiny difference in training accuracy, validation accuracy, and test accuracy in the different architecture. We showed the result of the different architecture in Table 4, and Table 5.

In Table 4, we show the best two models in the binary classification task. In Table 5, we show the

Architecture Detail	Accuracy	Valdiation Accuracy
MLP(512)-Bidirectional[(LSTM(128)x8]-MLP(512)-MLP(2)	0.686	0.663
MLP(512)-Bidirectional[(LSTM(32)x4]-MLP(512)-MLP(2)	0.679	0.661

Table 4: We report the best two architecture in our binary classification problems.

best three models in the eight classes classification task.

From the result of the above tables, we observed that:

1. The accuracy in the various model only has a tiny difference.

Architecture Detail	Accuracy	Valdiation Accuracy
MLP(512)-LSTM(32)-MLP(512)-MLP(8)	0.319	0.306
MLP(512)-Bidirectional[LSTM(32)]-MLP(512)-MLP(8)	0.317	0.303
MLP(512)-GRU(32)-MLP(512)-MLP(8)	0.316	0.301

Table 5: We report the best three architecture in our 8 classes classification problem.

2. RNN series model, include LSTM, GRU has regularly outperformed the other architecture.
3. The type of activation does not affect the result, but it impacts the learning progress in the experiment. We find that the ReLU activation function will lead to a smooth learning curve, and tanh activation will lead to a jittery learning curve. Fig. 6 shows a comparison between the two types of activation function.

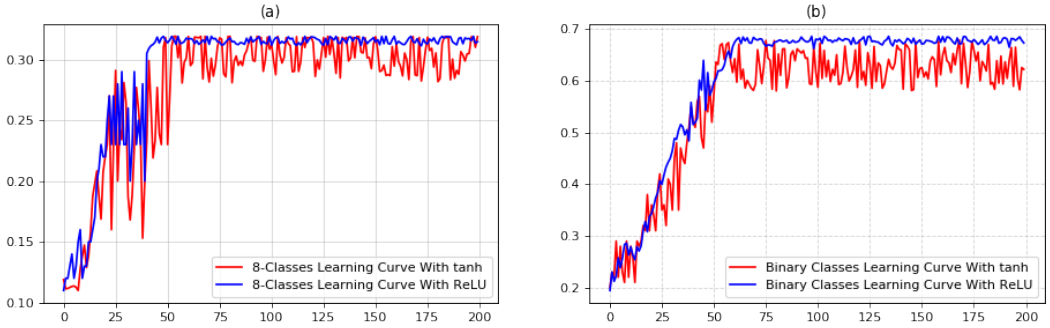

Figure 6: The learning curve compares for tanh and relu activation neurons. The left figure shows the learning curve for the network MLP(512)-LSTM(32)-MLP(512)-MLP(8) in the 8 classes classification task. The right figure shows the learning curve for the network MLP(512)-Bidirectional[(LSTM(128)x8]-MLP(512)-MLP(2) in the binary classification task.

4. the learning rate does not impact the result as we use the RMSprop as the default optimizer.

5.2 Computation Time in CPU

We have created more than 30 VMs with the specifications as 6 vCPU and 24G memory to test the running time per epoch in every architecture. The result showed that different network architectures have considerable differences in CPU running time. In practice, we suggest selecting the simpler architecture and only loss a little accuracy of the result.

We report the best top three inTable 6 and the worst top three inTable 7.

Architecture Detail	CPU running time per epoch(second)	Accuracy	Validation Accuracy
Flatten-MLP(512)-MLP(8)	400	0.3005	0.3012
MLP(512)-[Conv1D(32, 3), MaxPooling1D(2)]*2-Conv1D(32, 3)-MaxPooling1D-MLP(8) 970 0.314 0.285	970	0.314	0.285
MLP(512)-SimpleRNN(32)-MLP(512)-MLP(8)	1163	0.3005	0.3013

Table 6: We reported the best top three efficient architecture when CPU was used with comparable accuracy.

Architecture Detail	CPU running time per epoch(second)	Accuracy	Validation Accuracy
MLP(512)-Conv1D(1024, 3)-MaxPooling1D(5)-Conv1D(1024, 3)-MaxPooling1D-MLP(8)	102828	0.314	0.285
MLP(512)-Bidirectional((LSTM(128)x8)-MLP(512)-MLP(8)	85571	0.301	0.304
MLP(512)x6-Flaten-MLP(8)	29872	0.311	0.3

Table 7: We reported the worst three architecture when CPU was used with similar accuracy.

6 Discussion and Future Work

Focusing on the stock market in China, we propose the basic Deep Neural Network to get a comparable result for the stock prediction problem. According to the competent performance of models in the practical application, deep neural network methods have strong implications for investors as well as the entire stock market in China. The financial industry might integrate deep neural network prediction result in traditional prediction models to make better decisions. This is an interesting filed and promising direction in behavioral finance.

This study has inevitable limitations, which might be exciting directions in future work. We prepare to explore some potential directions to improve accuracy.

First, we could implement different structure of feature extractor, such as Transformer (Jaderberg u. a., 2015), BERT(Devlin u. a., 2018), GAN(Krizhevsky u. a., 2012).

Second, other modern neural network architecture may be worth to try like AlexNet(Creswell u. a., 2017), GoogLeNet(Szegedy u. a., 2014), VGG19(Simonyan u. Zisserman, 2017), ResNet(He u. a., 2015).

Third, the attention mechanism (Liu u. Wang, 2019) can be introduced to handle long-term dependency, which cannot be handled by RNN.

Forth, our hyper-parameter is only designed by heuristic, and if we have the formal policy for hyperparameter search, the result should be better than the current result. The coordinate descent, grid search, and random search policy are the suggested method for the future hyperparameters fine-tune.

Fifth, the deep reinforcement learning method is also well suited for the stock market prediction problem.

Sixth, the feature of the current dataset should not be enough if we use a robust network architecture; we should consider more features for each stock. Furthermore, some other conditions should also be beneficial, such as the international market trends, the future market price, the exchange market price, and the emotion status in the social network.

References

TensorFlow: Large-Scale Machine Learning on Heterogeneous Distributed Systems. In: *CoRR* abs/1603.04467 (2016). http://arxiv.org/abs/1603.04467

Deep Learning using Rectified Linear Units (ReLU). In: *CoRR* abs/1803.08375 (2018). http://arxiv.org/abs/1803.08375

The Differentiable Cross-Entropy Method. In: *CoRR* abs/1909.12830 (2019). http://arxiv.org/abs/1909.12830

A hybrid genetic-neural architecture for stock indexes forecasting. In: *Inf. Sci.* 170 (2005), Nr. 1, 3–33. http://dx.doi.org/10.1016/j.ins.2003.03.023. – DOI 10.1016/j.ins.2003.03.023

Forecasting stock market short-term trends using a neuro-fuzzy based methodology. In: *Expert Syst. Appl.* 36 (2009), Nr. 7, 10696–10707. http://dx.doi.org/10.1016/j.eswa.2009.02.043. – DOI 10.1016/j.eswa.2009.02.043

Twitter mood predicts the stock market. In: *J. Comput. Science* 2 (2011), Nr. 1, 1–8. http://dx.doi.org/10.1016/j.jocs.2010.12.007. – DOI 10.1016/j.jocs.2010.12.007

Nesterov's Accelerated Gradient and Momentum as approximations to Regularised Update Descent. In: *CoRR* abs/1607.01981 (2016). http://arxiv.org/abs/1607.01981

Support vector machine with adaptive parameters in financial time series forecasting. In: *IEEE Trans. Neural Networks* 14 (2003), Nr. 6, 1506–1518. http://dx.doi.org/10.1109/TNN.2003.820556. – DOI 10.1109/TNN.2003.820556

Exchange rate prediction redux: new models, new data, new currencies. In: *Journal of International Money and Finance* 95 (2019), S. 332–362

Learning Phrase Representations using RNN Encoder-Decoder for Statistical Machine Translation. In: *Proceedings of the 2014 Conference on Empirical Methods in Natural Language Processing, EMNLP 2014, October 25-29, 2014, Doha, Qatar, A meeting of SIGDAT, a Special Interest Group of the ACL*, 2014, S. 1724–1734

Learning Phrase Representations using RNN Encoder-Decoder for Statistical Machine Translation. In: *CoRR* abs/1406.1078 (2014). http://arxiv.org/abs/1406.1078

Empirical Evaluation of Gated Recurrent Neural Networks on Sequence Modeling. In: *CoRR* abs/1412.3555 (2014). http://arxiv.org/abs/1412.3555

Fast and Accurate Deep Network Learning by Exponential Linear Units (ELUs). In: *Computer Science* (2015)

How Many Random Seeds? Statistical Power Analysis in Deep Reinforcement Learning Experiments. In: *CoRR* abs/1806.08295 (2018). http://arxiv.org/abs/1806.08295

Generative Adversarial Networks: An Overview. In: *CoRR* abs/1710.07035 (2017). http://arxiv.org/abs/1710.07035

BERT: Pre-training of Deep Bidirectional Transformers for Language Understanding. In: *CoRR* abs/1810.04805 (2018). http://arxiv.org/abs/1810.04805

SGD momentum optimizer with step estimation by online parabola model. In: *CoRR* abs/1907.07063 (2019). http://arxiv.org/abs/1907.07063

Sigmoid-Weighted Linear Units for Neural Network Function Approximation in Reinforcement Learning. In: *CoRR* abs/1702.03118 (2017). http://arxiv.org/abs/1702.03118

Finding Structure in Time. In: *Cognitive Science* 14 (1990), Nr. 2, 179–211. http://dx.doi.org/10.1207/s15516709cog1402_1. – DOI 10.1207/s15516709cog1402_1

Deep Learning. MIT Press, 2016 (Adaptive computation and machine learning). http://www.deeplearningbook.org/. – ISBN 978–0–262–03561–3

Maxout Networks. In: *CoRR* abs/1302.4389 (2013). http://arxiv.org/abs/1302.4389

Deep Residual Learning for Image Recognition. In: *CoRR* abs/1512.03385 (2015). http://arxiv.org/abs/1512.03385

Hierarchical Recurrent Neural Networks for Long-Term Dependencies. In: *Advances in Neural Information Processing Systems 8, NIPS, Denver, CO, USA, November 27-30, 1995*, 1995, S. 493–499

Long short-term memory. In: *Neural computation* 9 (1997), Nr. 8, S. 1735–1780

Forecasting stock markets using wavelet transforms and recurrent neural networks: An integrated system based on artificial bee colony algorithm. In: *Appl. Soft Comput.* 11 (2011), Nr. 2, 2510–2525. http://dx.doi.org/10.1016/j.asoc.2010.09.007. – DOI 10.1016/j.asoc.2010.09.007

Application of wrapper approach and composite classifier to the stock trend prediction. In: *Expert Syst. Appl.* 34 (2008), Nr. 4, 2870–2878. http://dx.doi.org/10.1016/j.eswa.2007.05.035. – DOI 10.1016/j.eswa.2007.05.035

Spatial Transformer Networks. In: *CoRR* abs/1506.02025 (2015). http://arxiv.org/abs/1506.02025

Modelling high-frequency limit order book dynamics with support vector machines. In: *Quantitative Finance* 15 (2015), Nr. 8, S. 1–15

Financial time series forecasting using support vector machines. In: *Neurocomputing* 55 (2003), Nr. 1-2, 307–319. http://dx.doi.org/10.1016/S0925-2312(03)00372-2. – DOI 10.1016/S0925–2312(03)00372–2

An intelligent hybrid trading system for discovering trading rules for the futures market using rough sets and genetic algorithms. In: *Appl. Soft Comput.* 55 (2017), 127–140. http://dx.doi.org/10.1016/j.asoc.2017.02.006. – DOI 10.1016/j.asoc.2017.02.006

ImageNet Classification with Deep Convolutional Neural Networks. In: *Advances in Neural Information Processing Systems 25: 26th Annual Conference on Neural Information Processing Systems 2012. Proceedings of a meeting held December 3-6, 2012, Lake Tahoe, Nevada, United States*, 2012, S. 1106–1114

Inferring trade direction from intraday data. In: *The Journal of Finance* 46 (1991), Nr. 2, S. 733–746

Forecasting stock indices: a comparison of classification and level estimation models. In: *International Journal of Forecasting* 16 (2000), Nr. 2, S. 173–190

Empirical analysis: stock market prediction via extreme learning machine. In: *Neural Computing and Applications* 27 (2016), Nr. 1, 67–78. http://dx.doi.org/10.1007/s00521-014-1550-z. – DOI 10.1007/s00521–014–1550–z

Hybrid Neural Networks for Learning the Trend in Time Series. In: *Proceedings of the Twenty-Sixth International Joint Conference on Artificial Intelligence, IJCAI 2017, Melbourne, Australia, August 19-25, 2017*, 2017, S. 2273–2279

A Numerical-Based Attention Method for Stock Market Prediction With Dual Information. In: *IEEE Access* 7 (2019), 7357–7367. http://dx.doi.org/10.1109/ACCESS.2018.2886367. – DOI 10.1109/ACCESS.2018.2886367

Regularization and Optimization strategies in Deep Convolutional Neural Network. In: *CoRR* abs/1712.04711 (2017). http://arxiv.org/abs/1712.04711

Adaptive Weight Decay for Deep Neural Networks. In: *CoRR* abs/1907.08931 (2019). http://arxiv.org/abs/1907.08931

Activation Functions: Comparison of trends in Practice and Research for Deep Learning. In: *CoRR* abs/1811.03378 (2018). http://arxiv.org/abs/1811.03378

Understanding the exploding gradient problem. In: *CoRR* abs/1211.5063 (2012). http://arxiv.org/abs/1211.5063

Accelerating Minibatch Stochastic Gradient Descent using Typicality Sampling. In: *CoRR* abs/1903.04192 (2019). http://arxiv.org/abs/1903.04192

An overview of gradient descent optimization algorithms. In: *CoRR* abs/1609.04747 (2016). http://arxiv.org/abs/1609.04747

Textual analysis of stock market prediction using breaking financial news: The AZFin text system. In: *ACM Trans. Inf. Syst.* 27 (2009), Nr. 2, 12:1–12:19. http://dx.doi.org/10.1145/1462198.1462204. – DOI 10.1145/1462198.1462204

Bidirectional recurrent neural networks. In: *IEEE Transactions on Signal Processing* 45 (1997), Nr. 11, S. 2673–2681

Very Deep Convolutional Networks for Large-Scale Image Recognition. CoRR. 2014;: abs/1409.1556. In: *arXiv preprint arXiv:1409.1556* (2017)

Going Deeper with Convolutions. In: *CoRR* abs/1409.4842 (2014). http://arxiv.org/abs/1409.4842

Stock price direction prediction by directly using prices data: an empirical study on the KOSPI and HSI. In: *IJBIDM* 9 (2014), Nr. 2, 145–160. http://dx.doi.org/10.1504/IJBIDM.2014.065091. – DOI 10.1504/IJBIDM.2014.065091

Market Index and Stock Price Direction Prediction using Machine Learning Techniques: An empirical study on the KOSPI and HSI. In: *CoRR* abs/1309.7119 (2013). http://arxiv.org/abs/1309.7119

Benchmarking TPU, GPU, and CPU Platforms for Deep Learning. In: *CoRR* abs/1907.10701 (2019). http://arxiv.org/abs/1907.10701

AdaGrad stepsizes: Sharp convergence over nonconvex landscapes, from any initialization. In: *CoRR* abs/1806.01811 (2018). http://arxiv.org/abs/1806.01811

Recent Trends in Deep Learning Based Natural Language Processing. In: *CoRR* abs/1708.02709 (2017). http://arxiv.org/abs/1708.02709

ADADELTA: An Adaptive Learning Rate Method. In: *CoRR* abs/1212.5701 (2012). http://arxiv.org/abs/1212.5701

Combining News and Technical Indicators in Daily Stock Price Trends Prediction. In: *Advances in Neural Networks - ISNN 2007, 4th International Symposium on Neural Networks, ISNN 2007, Nanjing, China, June 3-7, 2007, Proceedings, Part III*, 2007, S. 1087–1096

Dilated convolution neural network with LeakyReLU for environmental sound classification. In: *2017 22nd International Conference on Digital Signal Processing (DSP)*, 2017

Financial prediction using neural networks. International Thomson Computer Press, 1996