`Vectoral Symbols

– exchange rates

– interval of bins

– observations

– returns

– forward time steps

– ranking of returns

– ordinal vector of bins of returns

– one-hot vector of bins of returns

# INTRODUCTION

Improvement of neural networks have increased significantly last two decades due to two factors:

1. Larger datasets occurred to due to increase on capacity of storing data and time impact.
2. Powerful computers have started to be used that overcome computation cost challenges.

Novelty of this study can be listed as follows:

1. Development of methodology for application of DOE for hyperparameter optimization in financial time series multivariate multi-step prediction.
2. Analysis of efficiency of DoE method comparing with Hyperband Tuner in hyperparameter optimization for financial data analysis.
3. Development of custom loss and metric functions complying with portfolio optimization problem.

Main motivation of this study is to have a significant improvement of multivariate multi-step prediction of financial time series.

# METHODOLOGY

As it is illustrated in Business Process Modelling Notation (BPMN) diagram in Fig. 1.1, firstly the exchange rates are imported.

Diagram

Description automatically generated

* 1. BPMN Graph of Applied Method

Define input & output datasets

Define datasets train, validation and test

Define optimizers.

Define the architectural hyperparameters.

Define the hyperparameters that can be optimized during the training with validation dataset.

DEFINE STRUCTURED EXPLORATORY DATA ANALYSIS TO IDENTIFY THE DATASETS THAT COME FROM SAME DISTRIBUTION FOR TRAIN, VALIDAITON AND TEST DATASETS. ADD OUTLIER DETECTION MANUAL INVESTIGATION FROM GOOGLE AS WELL.

# time series prediction problem

## Informal Definition of Exchange Rate Prediction Problem

In this paper, informal time series problem is defined based on the taxonomy that is described in the book written by (Brownlee, 2018). In this context, eight questions are considered:

1. What are the inputs and outputs for a forecast?
   1. Inputs could be one or more of following:
      1. Seasonal features of timestamp.
      2. Historical return of relevant exchange rates.
      3. Historical rankings of returns of exchange rates.
   2. Outputs could be one of following:
      1. Returns of exchange rates.
      2. Rankings of returns of exchanges.
2. What are endogenous and exogenous variables?
   1. Endogenous variables are the input variables that could be correlated with other features.
      1. Historical return values of the exchange rates that are under same category could be dependent on each other.
      2. Timestamp features are also dependent on each other.
   2. Exogenous variables are the input variables that are not influenced by the other features. Exogenous variables are beyond the scope of this paper. However, a social media or news dataset regarding the exchange rates could be an exogenous variable.
3. Are you working on a regression or classification predictive modeling problem?
   1. Regression is to predict a numerical value.
      1. Returns of exchange rates could be predicted with regression logic.
   2. Classification is to predict the class.
      1. Returns of exchange rates could be predicted with classification logic. Bins of the return values can be defined based on exploratory data analysis.
      2. Rankings of returns of exchange rates could be considered as classification problem.
4. Are the time series variables unstructured or structured?
   1. Unstructured time series variables are not based on a pattern such as trend or seasonality.
      1. In general, return of exchange rates can not be explained with trend or seasonality. However, there could be some exceptions. It could be identified with exploratory data analysis.
   2. Structured time series variables are based on trend of seasonality.
      1. Timestamp features are structured.
5. Are you working on a univariate or multivariate time series problem?
   1. Univariate time series problem consists of only one feature.
   2. Multivariate time series problem consists of multiple features.
      1. As it is mentioned on first and second questions, there are multiple inputs are considered such as return values, timestamp features. Due to multiple variables, the time series problem that is in scope of this paper is a multivariate time series problem.
6. Do you require a single-step or a multi-step forecast?
   1. Single-step forecast focuses on to forecast only next time step.
   2. Multi-step forecast is forecasting multiple time steps at once.
      1. It will be multi-step forecast so that these predictions will be input for portfolio optimization problem.
7. Do you require a static or a dynamically updated model?
   1. Static model is type of forecasting model that is fit only once and it is used to make predictions of unseen values.
      1. In this paper, static models will be used. However, performance of the models should be tracked for online prediction. If the performance of the model starts to decrease, then the model should be fit again.
   2. Dynamic model is the model that is fit newly before each prediction.
8. Are your observations contiguous or discontiguous?
   1. Contiguous observations are the observations where the frequency between timestamps of each observation is constant.
      1. In this study, most of the timestamps are contiguous with a predefined frequency.
   2. Discontiguous observations are the observations that don’t have a structured frequency between the observations.
      1. Due to some public holidays or special reasons about a particular exchange rate, some data could be missing.
      2. It is recommended to train the exchange rates in which timestamps are common. In case two stock exchanges of different countries, it is likely to have a discontiguous observations due to different working times of stock exchanges.

## Formal Description of Output Dataset

As it is mentioned in informal time series problem definition, there could be various forms of output dataset. At the beginning of the study, we don’t know what the best design for output dataset is. That’s why, different candidate designs are created. Since the main purpose of exchange rate prediction is to build an optimization that brings the highest return, the main feature of output is return. The feature of return in exchange rate prediction has a special characteristic because it has meaning of more profit when its absolute value is greater. Smaller negative return brings greater profit for sell position while greater positive return brings greater profit for buy position. Due to this characteristic of return value, in this paper, it is suggested to consider positive and negative returns separately within a model. Design of model will be discussed later but, in this section, output dataset can be represented in a way that it has two sub-datasets: one for negative returns and one for positive returns. Under this section, different designs of output datasets are presented. For the sake of simplicity, figures that illustrate the draft of the output datasets are used.

### Return Prediction

Main purpose of return prediction is to predict floating values of returns. Fig. 1.5 shows the general draft of output dataset that is used to predict return values. In this design, for an observation , for a return sign , for a future time step , return vector can be represented as shown in Formula (##):

|  |  |
| --- | --- |
|  |  |

where is an exchange rate. Length of is equal to length of meaning that for each exchange rate, there is a return value.

Diagram

Description automatically generated

* 1. Draft of Output Dataset for Return Prediction

If an exchange rate’s return value is negative, then its return value in positive sign is set as zero. This logic is represented in Formula (##):

|  |  |
| --- | --- |
|  |  |

where is opposite sign of .

### Ranking Prediction

As it will be discussed in later sections, portfolio optimization is done based on giving higher prioritization to higher return. That’s why, rankings of return values can be considered in output dataset. Ranking vector represents the descending ranking from the highest absolute return to lowest absolute return. The return values that have the same absolute value has equal ranking. In this design, for an observation , for a return sign , for a future time step , ranking vector can be represented as shown in Formula (##):

|  |  |
| --- | --- |
|  |  |

Diagram

Description automatically generated

* 1. Draft of Output Dataset for Ranking Prediction

If an exchange return value is negative, then corresponding element in positive ranking vector is set as zero. This logic is represented in Formula (##):

|  |  |
| --- | --- |
|  |  |

where is opposite sign of .

### Ordinal Bin Prediction

Return values can be split into bins. Interval of the bins can be determined by exploratory data analysis. This dataset is convenient to be output dataset of a classification problem. Fig. 1.7 shows the draft of output dataset for bin prediction design. A bin vector consists of the ordinal values of the bins. A bin which has greater ordinal value has greater interval boundaries. In this design, for an observation , for a return sign , for a future time step , bin vector can be represented as shown in Formula (##):

|  |  |
| --- | --- |
|  |  |

where is an exchange rate.

Diagram, table

Description automatically generated

* 1. Draft of Output Dataset for Ordinal Bin Prediction

If an exchange return value is negative, then corresponding element in positive bin vector is set as zero. This logic is represented in Formula (##):

|  |  |
| --- | --- |
|  |  |

where is opposite sign of .

### Binary Bin Prediction

The design that is mentioned in Section 6.2.3 serves to a classification problem where each class represents ordinal number of a bin. In this design (see Fig. 1.8), ordinal classes are converted to binary fields. Each bin has its own binary field. Output vector is a one-hot vector in which the target bin is equal to one and all other bins are equal to zero. In addition to above-mentioned designs, there is additional dimension that represents the bins is added. In this design, for an observation , for a return sign , for a future time step , for a bin interval ,ranking vector can be represented as shown in Formula (##):

|  |  |
| --- | --- |
|  |  |

where is a bin of interval of return value.

Diagram, table

Description automatically generated

* 1. Draft of Output Dataset for Binary Bin Prediction

One-hot logic can be formulated as shown in Formula (##):

|  |  |
| --- | --- |
|  |  |

If an exchange rate’s return value is negative, then all the elements in positive one-hot vector is set as zero. This logic is represented in Formula (##):

|  |  |
| --- | --- |
|  |  |

where is opposite sign of .

## Formal Description of Input Dataset

DESCRIBE INPUT DATASET CONSISTS OF RETURN VALUES + LAST DIGITS + SEASONAL FEATURES. LAG VALUES CAN BE IDENTIFIED PACF (PARTIAL AUTOCORRELATION FUNCTIONS) PLOTS. CANDLESTICK FEATURES COULD BE OPTION.

## White Noise Testing

IN THIS STUDY, SO FAR LOOKS LIKE, WHITE NOISE TESTING IS NOT VERY APPLICABLE DUE TO BINARY OUTPUTS… HOWEVER, WE CAN PRESENT SOMETHING. IF A TIME SERIES HAS WHITE NOISE, THAT TIME SERIES IS NOT PREDICTABLE. (WITH AUTOCORRELATION). ZERO MEAN, CONSTANT DEVIATION, ZERO AUTOCORRELATION.

## Dataset Split

Unlike statistical time series prediction models, deep learning models don’t require input data to be stationary. However, a model that is trained with historical data should be applicable for future unseen data which is the main motivation of time series prediction. In order to ensure that a machine learning model has comprehensive performance, input and output datasets are split into 3 parts with corresponding index of each other:

1. **Training dataset** – is used to train the parameters (weights) of the model.
2. **Validation dataset** – is used to validate the performance of training dataset covers unseen data as well. In addition, validation dataset is used to set the hyperparameters.
3. **Test dataset** – represents the unseen datapoints to test final performance of the model.

Usually, train-validation-test datasets are identified based on the ratio 60-20-20 percent, respectively. The performance of a model over the validation and test datasets, is called **generalization error**. The aim of machine learning models is to estimate the generalization error via using the **training error**. If there is a significant gap between training error and generalization error, it means that the model **overfits**. If the training error is not good enough, it means that the model **underfits**. In order to avoid overfitting, there are two important actions that need to be taken:

1. Make sure that training, validation, and test datasets are **independently and identically distributed (i.i.d)** with each other.
2. If there is still a gap between training error and generalization error, then increase the size of training dataset.

Independence means that data points (observations) should not be impacted from each other. Unfortunately, this is not very applicable in time series prediction due to latent autocorrelations which means that it is likely that one observation on is impacted with the some of its previous observations (e.g., ). In order to overcome dependency issues between data points, the following strategies could be applied:

* Transform datasets in a way that whole initial datasets are represented as a vector. This requires huge computational costs.
* Use Markov assumptions instead of using i.i.d. assumption. Rather than saying “all observations are independent”, Markov assumption states “each observation is dependent previous observations in same order”.
* Use time-lags or covariance structures to state autocorrelation.
* Using sequence-to-sequence algorithms which considers autocorrelations between each time step such as attention mechanisms.

The second condition of i.i.d is that data distributions of train, validation and test datasets are identically distribution. Each of these datasets should come from the same data distribution. In order to assess the identicality of two distributions the following techniques can be used:

* Apply Kolmogorov-Smirnov test to identify if these datasets are from same distribution or not.
* Use Kullback-Leibler (KL) divergence to identify how different these datasets are from each other.

In case these datasets are not from same distributions, following decisions can be done:

* Converting the time series datasets to stationary.
* Do not consider these futures at all.

## Cost Function

In this paper, **maximum likelihood estimation** principle is used to find the best model generating distribution which is the most similar with true data generating distribution . Authors of (Goodfellow, Bengio, & Courville, 2016) define maximum likelihood as an attempt to make the model distribution match with the empirical data distribution. Let’s denote input dataset with consists of data points as . We represent corresponding output vector with We assume that each observation of and are independent. If is the parameters of predictive model, then model parameters that gives maximum likelihood is calculated as shown in Equation (##):

|  |  |
| --- | --- |
|  |  |
|  |  |

where – is the probability distribution of having the outputs with the given input and model parameters. Multiplying the probability of for each data point will cause the numerical underflow problem in which the cost function will be too small that is almost zero. For the purpose of prevention from numerical underflow problem, logarithmic transformation is used. Logarithmic transformation is equivalent with the original version of optimization. It is helpful in terms of computational perspectives since it converts consecutive multiplication operations to summation operations as shown in Equation (##):

|  |  |
| --- | --- |
|  |  |

In order to make the maximum likelihood independent from the size of dataset, cost function is divided by to calculate the expectation with respect to true data distribution . The objective is to find the model parameters that gives the minimum dissimilarity between two distributions and . In this context, **KL divergence** is calculated as shown in Equation (##):

|  |  |
| --- | --- |
|  |  |

There are two terms in KL divergence. First term represents the logarithmic output of true data generation process. Since we can’t change the true data, we need to minimize the cost function that has only negative log of model distribution as shown in Equation (##):

|  |  |
| --- | --- |
|  |  |

where – is the cost function. It is called also as **cross-entropy** or **negative log-likelihood**.

The advantage of maximum likelihood estimation approach is that it gives more **consistent** model that show better convergence with greater .

MAKE THIS STUDY BAYESIAN APPROACH. MENTION IN THIS SECTION ABOUT BAYESIAN APPROACH AND MAXIMUM A POSTERIORI ESTIMATION AS AN ALTERNATIVE TO MAXIMUM LIKELIHOOD ESTIMATION… RE-WRITE THIS PART ACCORDINGLY. ALSO ARCHITECTURE OF KERAS MODELS WILL CHANGE. TENSORFLOW-PROBABILIESTIC LIBRARY WILL BE USED.

COST FUNCTION WILL BE SAME BUT IT WILL REQUIRE ADDITIONAL TRANSFORMATIONS FOR EACH OUTPUT DESIGN. THINK ABOUT SHARP RATIO. OR AT LEAST CONSIDER IT IN PORTFOLIO OPTIMIZAITON.

# GRADIENT BASED OPTIMIZATION algorithms

Optimization procedure to find the best weights of deep learning models is called **training** procedure. Deep learning models are multi-input models that are usually not simple linear models or 2nd order convex models. In order to train these types of models, **gradient based optimization algorithms** are used. Be aware that optimization algorithms that are presented under this section are not the optimization algorithms that are used to optimize the hyperparameters of the deep learning models. Target of gradient based optimization algorithms is to convert non-convex deep learning models to convex models, and to apply an optimization procedure based on the cost function of the model. In this context, some important numerical and algebraic concepts such as Taylor series approximation, eigendecomposition, 1st and 2nd order gradients are presented to understand the mechanism of optimization algorithms. In addition, some common gradient based optimization algorithms that are used in this thesis, is explained.

## Eigendecomposition

In order to have more information about a matrix and also in order to reduce computation costs due to matrix operations, matrices can be decomposed to smaller parts. There are various matrix decomposition techniques such as eigendecomposition, singular value decomposition etc. In this section, eigendecomposition method is described based on the book section written by (Goodfellow, Bengio, & Courville, 2016).

**Eigendecomposition** is decomposing a matrix to a set of **eigenvectors** and **eigenvalues**. Eigendecomposition can be applied to only square matrices. The concept of eigenvalues and eigendecomposition can be described as follows in Formula (##):

|  |  |
| --- | --- |
|  |  |

where  **–** is a square matrix;

**–** is a unit vector called as “eigenvector”;

– is a scalar value called as “eigenvalue”.

A matrix can have multiple eigenvectors and eigenvalues. Suppose that the matrix has linearly independent eigenvalues, which can be represented vector . The corresponding eigenvectors can be represented with matrix . Eigendecomposition of the matrix can be represented as shown in Formula (##):

|  |  |
| --- | --- |
|  |  |

where – is a diagonal matrix of vector .

In this study, we are interested in the matrices that can be composed to eigenvectors and eigenvalues that contain only real numbers. If a matrix is a real-valued and symmetric, decomposing it to real-valued eigenvalues and eigenvectors is possible. Eigenvalue decomposition can give us following functional attributes:

1. The matrix is singular if and only if has at least one eigenvalue that is equal to zero. Recall that singular matrices are the matrices whose determinant is zero. WRITE HERE WHAT IS THE IMPACT OF SINGULAR MATRIX ON TRAINING ALGORITHM. IF THERE IS NO IMPACT, THEN DELETE THIS.
2. The matrix can be used to optimize a quadratic expression where is eigenvector of . Optimization of the quadratic expression can be done with a constraint subject to . Maximum and minimum eigenvalues give maximum and minimum values of respectively within the constraint region.

Based on the values of eigenvalues in , matrices can be categorized as follows:

1. Positive definite – all of the eigenvalues are positive.
2. Positive semidefinite – all of the eigenvalues are either positive or zero.
3. Negative definite – all of the eigenvalues are negative.
4. Negative semidefinite – all of the eigenvalues are either negative or zero.

## First Derivative

Suppose a function which can be defined as . **Derivative** of this function is the slope of at the point and it is denoted as or . Derivative of a function is useful to minimize a function that is a cost function particularly in deep learning. It specifies how changes when there is a small change on . Sign of derivative gives the direction of change of . In case of minimizing a cost function, the direction of improvement of should be opposite direction of sign of . In case , is called as “critical. There are 3 types of critical points:

1. Minimum point
2. Maximum point
3. Saddle point (neither maximum nor minimum)

Maximum and minimum points could indicate local or global points. In cost function of deep learning, it is very rare to find global minimum. However, a local optimum could be good enough to find a sensible cost value.

In deep learning, a cost function that is targeted to be minimized is defined as where the function has multiple inputs and a scalar output. In the case of multiple inputs, **partial derivative** concept is used. Partial derivative specifies how the output will change on the point , only if the variable changes. **Gradient** is the general term which is the derivative of the function with respect to a vector. Gradient of function is denoted where is the input vector. Critical points of functions with multiple inputs are the points where each partial derivative is equal to zero.

As it is mentioned above, direction of changing input to minimize the output is defined as opposite sign of derivative of the function. When it comes to minimize the functions that have multiple inputs, the change should be on the direction of the negative gradient. The general terminology of **steepest descent** or **gradient descent** is coming from this approach. In this context, new input of the function can be calculated as shown in Formula (##):

|  |  |
| --- | --- |
|  |  |

where – is the current input of the function;

– is the updated input of the function;

– is a positive scalar value that indicates step size of changing. It is called as **learning rate**.

There are various ways to choose the learning rate:

* It can be chosen as a constant value. It can be changed on the later stages of learning algorithms.
* Cost function can be calculated for several options of learning rate. The option that gives the lowest cost function can be determined. This strategy is called as **line search**. In this paper, we propose alternative methods to line search to identify learning rate and also other hyperparameters.

Gradient descent algorithm converges on critical points which means that when every element of the gradient is equal to zero. Practically, it is not very common to have zero gradient vector. However, a vector who has values very close to zero is also acceptable. The term “very close” is relative approach per study. This is an iterative based approach with smaller improvements.

## Second Derivative

Derivative of derivative is called as **second derivative**. It can be denoted as or for single dimensional inputs. In multidimensional space, let’s consider a function , then the derivative with respect to of the derivative of with respect to is denoted as .

As it is mentioned in Section 4.2, when first derivative of a function is equal to zero, on point , it means that point is critical point. However, it doesn’t give information if the point local minimum, local maximum, or saddle point. In order to identify the type of critical point, second derivative of the function is used. In single dimensional functions, on critical point, second derivative can be interpretated in three scenarios when :

1. – point is local minimum.
2. – point is local maximum.
3. – point is saddle point or part of a flat region.

Second derivative specifies how much the first derivative will change when the input is changed. It shows whether the gradient step will make the expected improvement. It represents the **curvature** of the function. There are three different forms of second derivative:

1. If second derivative of a function is zero, it means that there is no curvature. The function is like a line. That’s why expected value of the improvement on cost function can be calculated via using only gradient.
2. If second derivative of a function is negative, it means that function has downward curvature which means function has a hill. Improvement on the cost function will be more than learning rate.
3. If second derivative of a function is positive, it means that there is an “upward curvature” of function. In this case, cost function will be improved less than learning rate. Upward curvature means that if the step size is too big, the value of cost function will start to increase inadvertently.

## Jacobian and Hessian Matrices

Consider a function where is a function of which both of the input and output are vectors. The matrix that consists of all partial derivatives of function is called as **Jacobian matrix**. Jacobian matrix of is shown as such that .

When the function has multiple inputs, the matrix where the second derivatives are represented is called as **Hessian matrix**. In other words, Hessian matrix is the Jacobian matrix of the gradients. Hessian matrix can be demonstrated as . In case the second partial derivatives are continuous, Hessian matrix is symmetric (). When the Hessian matrix is symmetric and real, eigendecomposition (see Section 4.1) can be applied on it. Eigendecomposition helps to find the optimum values of the function that is in form of a quadratic expression where is eigenvector of . The minimum eigenvalue, ,will give the minimum second derivative, while maximum eigenvalue, ,will be the maximum second derivative.

In Section 4.3, second derivative of single dimensional functions at the critical points () is interpreted to identify if the critical point is local minimum, local maximum, or saddle point. However, we didn’t describe how to use second derivative information for the critical points of multidimensional functions. In order to identify the type of the critical point in multidimensional functions, all of the second derivatives of the function should be calculated with respect to each input. That’s why, Hessian matrix is used. At a critical point of where , Hessian matrix can be used in four situations:

1. If the Hessian matrix is positive definite which means all of the eigenvalues of Hessian matrix is positive, then the critical point is local minimum.
2. If the Hessian matrix is negative definite, then the critical point is local maximum.
3. If at least one of the eigenvalues of Hessian matrix is positive and at least one of the eigenvalues is negative, then is local minimum on one direction while it is a local maximum on another direction. It means that it is a saddle point.
4. If all of the eigenvalues have the same sign but at least one eigenvalue is zero, it means that second derivative test is inconclusive.

Condition number (see Section 4.2) of Hessian matrix gives how much the second derivatives differs from each other. It tells how much the direction of the most curvature of the function is bigger than the direction of the least curvature of the function. If condition number is very large, gradient descent performs poorly. Because in one direction derivative increases too rapidly, while in other direction it increases too slowly. It also makes it difficult to choose a good learning rate. Because in order to overcome this challenge, learning rate should be very small. Too small learning rate will not improve the cost function significantly.

## Taylor Series Approximation

## Challenges of Training Algorithms

### Overflow and Underflow

DESCRIBE OVERFLOW AND UNDERFLOW. MENTION WHY IT CAUSES A CHALLENGE IN TRAINING ALGORITHMS.

### Ill-Condition Number

Suppose a function where and . If has eigenvalue decomposition, the **conditioning number** can be represented as shown in Formula (##):

|  |  |
| --- | --- |
|  |  |

where – is the maximum eigenvalue of matrix ,

– is the minimum eigenvalue of matrix .

Condition number of a function implies the sensitivity of changes on output when there is a small change on input:

1. If condition number is large, it means that the function will change bigger when its inputs are changed even small amount. Matrices with condition number greater than 1 are called as **ill-conditioned** matrices.
2. If condition number is small, it means that the function will change slower when its inputs are changed small amount. Matrices with condition number less than 1 are called as **well-conditioned** matrices.

In deep learning trainings, we aim to optimize the functions whose parameters can be represented with well-conditioned matrices. Ill-conditioned Hessian matrix shows that small changes in weights of neural networks will increase the cost function, however in a gradient descent algorithm value of cost function should decrease on each step. In order to overcome issue of ill-conditioned Hessian matrix, Newton’s method can be used. Newton’s method will be described in later sections.

### Saddle Points

As it is mentioned above, on the point where the gradient is equal to zero, there are 3 options for the type of critical points: local minimum, local maximum, and saddle points. In order to understand the type of critical point, Hessian matrix is used. In multidimensional non-convex functions with Hessian matrix that has both negative and positive eigenvalues, the type of critical point is saddle point. While moving towards the directions of eigenvectors corresponding with positive eigenvalues, cost function arises. While moving directions of eigenvectors corresponding with negative eigenvalues, cost function decreases. In training algorithms of deep learning models, it is more likely that the critical points will be saddle point instead of local minimum or local maximum. Because in order a critical point to be a local minimum on multidimensional functions, all eigenvalues of Hessian matrix should be positive which has lower probability of having mixed signs. That’s why, we can conclude that saddle points are real challenge for training algorithms of deep learning models.

## Stochastic Gradient Descent Optimization

**Stochastic gradient descent (SGD)** is a special type of gradient based algorithm that performs learning not with full set of training dataset but with a limited size of it due to avoid high computation cost of large training sets. The small set of training dataset that is used to compute the cost function is called as **minibatch**. In the book (Goodfellow, Bengio, & Courville, 2016), recommends using power of 2 minibatch sizes between 32 and 256 for computations with GPU. Smaller batch sizes have better generalization error, and they require smaller learning rate. This leads longer runtime requirements. In order to observe entire training dataset, smaller batch size require more steps. This will also cause longer runtime. EXPLAIN BATCH SIZE MORE CLEARLY. IS IT FIXED PER EPOCH OR IT GOES THROUGH ALL BATCHES IN ONE EPOCH. The biggest advantage of SGD is that computation cost per epoch is independent from the size of training set since it works with the fixed size of sampled dataset per each epoch.

In the book (Goodfellow, Bengio, & Courville, 2016) random selection of minibatches is recommended. In time series prediction problem, it is a challenge to select minibatches randomly so that subsequent minibatches are independent from each other. After preprocessing the dataset in a way that each sample has its inputs and outputs ready, then training dataset can be shuffled. Let be a function with multiple input and scalar output; then expected gradient based on a minibatch is calculated as shown in Equation (##):

|  |  |
| --- | --- |
|  |  |

where – is expected gradient of loss function with respect to model parameters;

– model parameters (weights) to be optimized;

– is minibatch size;

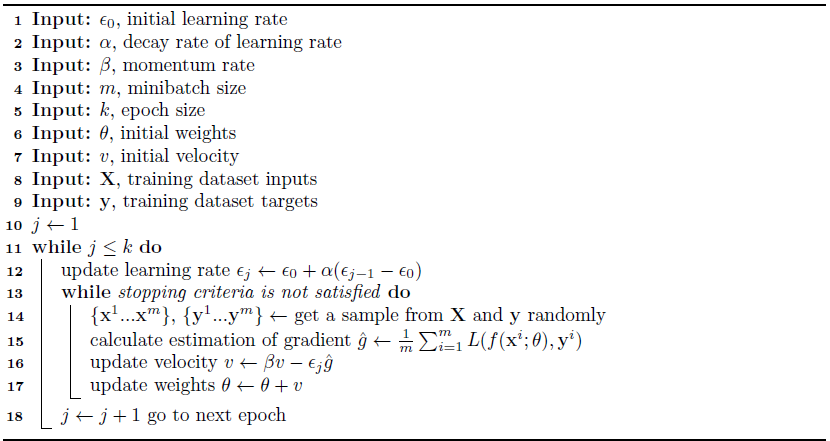
– is input vector of th example in minibatch;

– is the corresponding output of ;

– is cost function and

– is the function that maps inputs to prediction value via using parameters .

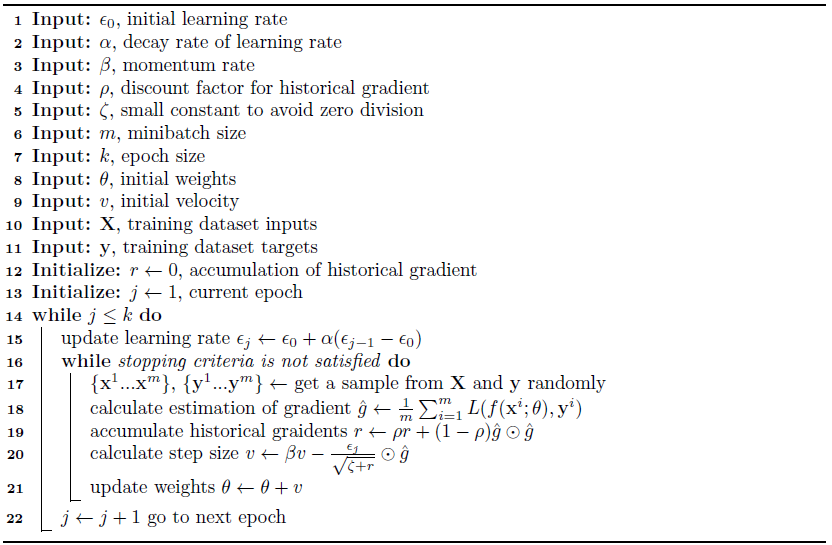
In Fig. 1.2, SGD algorithm is shown step by step for each epoch. In SGD, it is practical to decay the learning rate on each epoch with a decay rate . In the book written by (Goodfellow, Bengio, & Courville, 2016), it is suggested to use value as 1% of . In this context, we can consider only to be part of hyperparameter optimization. It is suggested to monitor first several iterations and then choose the learning rate that yields best. The SGD that is used in this study considers the variation of **momentum** method(Polyak, 1964). Momentum method works with a **velocity** that represents the exponentially decaying average of negative gradient. A hyperparameter **momentum rate**, , is used to determine how quickly the contributions of previous gradients exponentially decay. Momentum method overcomes the challenges of poorly conditioned Hessian matrix and variance in the gradient descent. When is set to 0, new weights are not affected from previous gradients, weight update will be towards direction of current gradient. If is set greater value than , then weights of the deep learning model will be updated more towards the direction of previous gradients than current gradient. When the consecutive gradients are in the same direction, velocity will be larger in each iteration. This will lead to have larger step sizes for weight updates. Authors of (Goodfellow, Bengio, & Courville, 2016) address the practical usage of momentum rate as 0.5, 0.9 and 0.99.



* 1. Psuedocode for Stochastic Gradient Descent Algorithm

## RMSProp Optimization

In the online courses facilitated by (Hinton, 2012), **RMSProp** optimization which is a variant of **AdaGrad** optimization (Duchi, Hazan, & Singer, 2011) has been presented. Authors of (Goodfellow, Bengio, & Courville, 2016) stated that practical results AdaGrad algorithm is not very promising in non-convex models such as deep learning models. Due to this reason, in this study, RMSProp has been considered instead of AdaGrad optimization. Pseudocode of RMSProp algorithm is presented in Fig. 1.3. Learning decay per epoch and momentum method are also applied in RMSProp. In addition to SGD, accumulation of squared values of historical gradients are considered. In SGD’s momentum method, velocity (step size) was representing the historical gradient that is updated based on multiplication of learning rate and gradient. In RMSProp’s momentum method, historical gradient is kept in squared form. In order to discard too old historical gradients and to keep more current historical gradients, exponential decay method is applied via hyperparameter . In addition, a very small scalar hyperparameter is used to avoid zero divisions.



* 1. Pseudocode of RMSProp Algorithm

## Adam Optimization

**Adam optimization** introduced by (Kingma & Ba, 2014) is a commonly used learning algorithm to identify optimum weights of neural networks. As shown in Fig. 1.4, Adam optimization includes two momentum rates and for exponential decay rate of accumulation of plain gradient and squared gradient respectively. In addition, it applies bias correction to both of the accumulations. Bias correction provides less bias during the training. Adam algorithm is one of the most common algorithm in recent deep learning practices.

Text, letter

Description automatically generated

* 1. Pseudocode for Adam Algorithm

# BACK-PROPAGATION

When a neural network is fed with input and propagates based on architecture, and finally produces an output , this process is called as “forward propagation”. As a result of forward propagation, a scalar cost value is calculated. The information is flowed backwards starting from the cost value. This flow operation is called as “back-propagation” (Rumelhart, Hinton, & Williams, 1986). Output of back-propagation process is called as “gradient”. Mostly gradient of cost function with respect to parameters is calculated. A learning algorithm uses this gradient to perform learning. Learning algorithms are described in subsequent section 4.3. In this section, gradient computation is described.

## Computational Graphs

In order to describe the back-propagation algorithm in more simple way, “computational graphs” are used in libraries specified for deep learning such as TensorFlow (Abadi, et al., 2015), Torch (Collobert, Kavukcuoglu, & Farabet, 2011), Theano (Theano Development Team, 2016) etc. Computational graphs are directed graphs where “nodes” represent the “variables” which can be scalar, vector, matrix, or tensor. In computational graphs “edges” represent the mathematical “operations” that can also be named as “functions”. An example of computational graph is illustrated in Fig. 1.2. The computing the output of the graph in a computer is called as “evaluation”. In order to evaluate a graph, a “graph evaluation engine” is used.

Diagram

Description automatically generated

* 1. An Example of Computational Graph

### Chain Rule

Chain rule is studied firstly by the authors of (Leibniz, 1676; L'Hopital, 1696).

**EXPLAIN CHAIN RULE.**

In Fig. 1.2, the gradient of with respect to , , is calculated based on chain rule of calculus in Equation (##)

|  |  |
| --- | --- |
|  |  |

where – is any mathematical operation.

**EXPLAIN JAKOBIAN MATRIX DESCRIPTION WITH FORMULA 6.47.**

Using the algebraic expressions can look easy however, it requires additional considerations to evaluate subexpressions. For example, in order to evaluate the gradient , the operation should be computed 3 times. When deeper neural networks are considered, the number of the repeated evaluation of subexpressions will increase exponentially. In the context, the computing subexpressions can be considered in two approaches:

1. In case memory of computer is low – subexpression is re-computed several times. This will result high runtime.
2. In case memory of computer is high – subexpression can be calculated only once and stored in the memory. Back-propagation algorithm uses this approach.

## General Back-Propagation Algorithm

The authors of (Goodfellow, Bengio, & Courville, 2016) described back-propagation algorithm as shown in Fig. 1.3. Back-propagation algorithm returns a gradient table (*grad\_table*) that is the data structure where each variable in the target variable set whose gradients must be computed. Line number 6 in Fig. 1.3 indicates the partial derivative of with respect to itself, , is set to 1. Main part of back-propagation algorithm is “build\_grad” method is applied for each variable in set of variables in .

Graphical user interface, text, application

Description automatically generated

* 1. Pseudocode of Outer Skeleton of General Back-Propagation Algorithm

In Fig. 1.4, method is illustrated as a pseudocode. Each node in the graph corresponds to a variable which is a tensor. Since it is general algorithm, a tensor can have any number of dimensions and sizes such as matrix, vector even a scalar. Each has following subroutines:

* get\_operation() – returns the operation “” (function) that returns the . Each has a subroutine operation that is able to calculate the Jacobian vector product as shown in Equation (##).
* get\_consumers(,) – returns the list of variables that are children of within the computation graph .
* get\_input(, ) – returns the list of variables that are parents of within the computation graph .

If the variable is already in , then method returns the gradients of variable . Otherwise, children of variable is calculated via function. For each child node, the pointer of function that calculates that node is assigned to via function. In line number 11, the gradient of child node is calculated. All parents of the child node of is identified in line number 12. function of is used to identify the Jacobian vector product. After Jacobian vector product of each child node is identified, a summed gradient value is calculated, and it is set in for the variable . Finally calculated gradient value and corresponding operations, computational graph is updated in line number 17. Such way of storing the gradients in is called as “table filled approach” or “dynamic programming” by the authors of (Goodfellow, Bengio, & Courville, 2016).

Text

Description automatically generated

* 1. Pseudocode of Building Back-Propagatation (“build\_grad”)

## Complexity of Back-Propagation Algorithm

The runtime of back-propagation algorithm is directly proportional with the number of the edges in computational graph under the assumption that each operation has similar cost. Please note that, runtime of each individual operation can differ significantly. The field of computing derivatives in deep learning is called “automatic differentiation”. In terms of the sequence of evaluating the chain rule, different approaches could be considered in automatic differentiation. In this paper, back-propagation algorithm is presented as “reverse mode accumulation” since number of inputs are usually more than number of outputs. In case number of outputs are larger than number of inputs, “forward mode accumulation” technique should be used. While reverse mode accumulation is right-to-left multiplication of matrices, forward mode accumulation is left-to-right multiplication of matrices. However, finding the optimal sequence of operations is stated as NP-complete problem in the study done by (Naumann, 2006). Packages of (Abadi, et al., 2015) and (Theano Development Team, 2016) use some heuristic algorithms to identify the sequence of differentiation. Optimizing the automatic differentiation is an area which is very open for improvement. This is out of scope of this paper but it is identified as a future research area.

# ARCHITECTURE OF PREDICTIVE MODELS

## Output Units

In this study, cost function is based on maximum likelihood estimation as it was mentioned in Section 3.6. Output units are very related with cost function. Let denote the hidden features of output units then output units can produce a result based on Equation (##):

|  |  |
| --- | --- |
|  |  |

where – weights from hidden units to output units and

– bias for output units.

The predicted output of model can be denoted by and it can be calculated based on the transformations of with following types of units:

1. **Linear units** – don’t apply any transformation to as shown in Equation (##):

|  |  |
| --- | --- |
|  |  |

In case actual output values follow a Gaussian distribution then, linear units are used. Linear units don’t have any saturation that’s why it is easy to apply gradient based optimization to the models that have linear output units.

1. **Sigmoid units** – are used when follows Bernoulli distribution. Usually, binary classification models are appropriate for sigmoid units. Sigmoid units apply transformation as shown in Equation (##):

|  |  |
| --- | --- |
|  |  |

where – is the symbol of sigmoid transformation. Sigmoid units saturate the output value between interval . When is extremely small, then sigmoid output saturates to 0 while is extremely large, then it saturates to 1. However, output never becomes equal to 0 or 1. This gives ability of applying gradient based optimization because if output starts to be equal to 0 or 1, gradient would be equal to zero. DESCRIBE ABOUT LOGIT.

1. **Softmax units** – are used when output data fits with Multinoulli distribution. It is a kind of extension of sigmoid units because both are applied for classification problems. The main difference is that sigmoid is applied in case classification problem is a binary classification problem.

## Hidden Units

Designing hidden units is a different research area. In this study, we will be considering to use only the hidden units that are used most commonly. Because authors of (Goodfellow, Bengio, & Courville, 2016) state that commonly used hidden units are already well enough and compatible with the recently designed hidden units. In this context, following hidden units are used:

1. **Rectified linear units** – are known as **ReLU** as well. Activation function of ReLU is shown in Equation (##)

|  |  |
| --- | --- |
|  |  |

ReLU units are very similar to linear units except negative inputs. ReLU returns zero for negative inputs. This can deactivate some units in the layer. Because first derivative of negative inputted units will be always zero. Authors of (Goodfellow, Bengio, & Courville, 2016) suggest to set bias weights to 0.1 in order to ensure that most of the ReLU units will be active initially.

1. **Leaky rectified linear units** – are extension of ReLU. They are known as **Leaky ReLU**. Leaky ReLU considers to avoid having zero output of ReLU. For this purpose, it adjusts ReLU as shown in Equation (##):

|  |  |
| --- | --- |
|  |  |

where – is a hyperparameter that represents the slope for the negative outputs. Authors of (Maas, Hannun, & Ng, 2013) suggest to set value equal to 0.01.

1. **Linear units** – is simply not applying any activation function to inputs. It is same as linear units mentioned in output units. In order to build a non-linear model, linear units will not be applied to all layers. Instead, it will be applied to only some few of hidden layers. In the book written by (Goodfellow, Bengio, & Courville, 2016), it is stated that linear units are good when the number of hidden units are small.

In addition to linear units, sigmoid units or hyperbolic tangent units could also be used. These units were common before introducing ReLU units. Since computing the gradient of sigmoid and hyperbolic tangent are not easy due to saturation, in this paper, we are not considering sigmoid and hyperbolic units in hidden layers.

## Multi-Layer Perceptron (MLP)

In this section, we summarize description from the book chapter (Goodfellow, Bengio, & Courville, 2016). Multilayer perceptrons (MLP) are called also as “feedforward neural networks” because information flows from inputs through some intermediate computations and finally to the output. The output of the system is not considered as an input to system. If output is returned to neural network as input, it would be called as “recurrent neural network”. MLPs use chain structure where a set of functions are connected like a chain. For example, let’s consider 3 functions , and that are connected in a chain. If the input of the chain is demonstrated with , then we can form the function as . In this case, we can say that is the “first layer" of the neural network, is the “second layer” of the neural network and so on so forth. First layer of the network is called as “input layer” and final layer of the network as called as “output layer”. Learning algorithm is used to decide the correct output of the neural network. Input and output layers should represent the inputs and outputs of the training data respectively. However, there are layers in the neural networks that don’t represent the training data directly. These layers are called as “hidden layers”. Each neuron in the layers is called as “unit”. Each unit receives input from many other units and computes their own value with “activation functions”.

### Activation Functions of Hidden Units

It is recommended to use “Rectified Linear Unit (ReLU)” functions are activation functions (Glorot & Antoine Bordes, 2011) of hidden layers of MLPs. ReLU function is easy to optimize because half of the space of ReLU function is linear and the other half is zero. There are variations of the ReLU function considering with a non-zero slope: “absolute value rectification” (Jarrett, 2009) is with the slope of -1, “leaky ReLU” (Andrew L Maas, 2013) is with the slope of small value like 0.01 or “parameteric ReLU” (He, Zhang, Ren, & Sun, 2015) is with learnable slope parameter and so on so forth. It is also an option to use linear activation function in all layers of MLP. However, then MLP will be a linear form. That’s why, it is recommended to use non-linear activation functions in at least some of the hidden layers of MLP. In addition, softmax activation function also can be considered if the architecture of MLP requires a memory that represent a probability distribution.

It is not recommended to use the functions that saturate in hidden layers of MLP. Gradient-based learning is not very easy for the activation functions that saturate. Activation functions such as sigmoid function, hyperbolic tangent function or radial basis function are not suggested to use in hidden layers of MLPs. However, in recurrent neural networks, these functions can be considered. Please note that if an activation function saturates and if it is differentiable, it doesn’t mean that it will give better results. Authors of (Goodfellow, Bengio, & Courville, 2016) gives softplus function as an example to this statement. They state that softplus function demonstrates performance of hidden units very counterintuitive. The researchers of (Xavier Glorot, 2011) found better results with rectifier than softplus function.

### Architecture

Architecture of an MLP refers to the number of the layers and units that are connected to each other. The length of the chain is called as “depth” of the model. Dimensions of hidden layers are the “width” of the model. To sum up, length and width are the main architectural considerations of MLPs. Most of the time, MLPs with single hidden layer are sufficient to generalize training dataset. According to “universal approximation theorem” (Hornik, Stinchcombe, & White, 1989), we know that a large MLP has ability to represent the function between input and output of MLP. However, we don’t know if the training algorithm is able to learn this function due to lack of convergence and overfitting. Deeper networks are harder to optimize. In the book that is written by (Goodfellow, Bengio, & Courville, 2016), it is recommended to follow an experimentation process to identify the optimum network architecture. Experimenting the depth and width of MLPs in different applications demonstrated that there is a clear improvement between experiments. They are recommending that in the worst-case scenario, an exponential number of hidden units based on number of inputs can be used. However, please note that there is still not a clear suggestion about what should be the correct width and depth of MLP in the literature.

There are also several architectural considerations about how to connect the hidden layers with each other. Default way of connecting the hidden layer is to connect each hidden layer with the subsequent. However, it could be considered to skip connections from layer to or higher orders instead of subsequent layer . In addition to the way of skipping connections, the way how to connect hidden units with subsequent layer can be considered. By default, each hidden unit in a layer is connected with each hidden unit in subsequent layer. This way of connection is called as “dense” connection. However, it can also be a case to connect a layer with subsequent layer partially instead of fully.

## Long-Short Term Memory (LSTM)

LSTMs are mostly used in natural language processing since LSTMs are efficient to extract sequential relationships within input data. In the book written by (Brownlee, 2018), capability of LSTM in natural language processing could be used in time series prediction.

## Convolutional Encoder Decoder (Conv-EncDec)

CNNs are mostly used in image processing. In the book written by (Brownlee, 2018), CNNs are recommended for time series prediction via treating sequence of observations like a one-dimensional image. CNNs are useful for identification, extraction and distillation of features from raw input data. Please not that in CNNs, sequence of input data is not consideration.

## Luong’s Attention Mechanism (Luong-Att)

## Bahdanau’s Attention Mechanism (Bahdanau-Att)

## Vaswani’s Attention Mechanism (Transformer-Att)

# METHODS TO IMPROVE PREDICTION PERFORMANCE

We took the section of the book written by (Goodfellow, Bengio, & Courville, 2016). As the authors suggest, parameter norm penalty regularization that penalizes only the weights of deep learning model has been used. Biases are suggested to leave unregularized to avoid underfitting. There are also suggestions to use different regularization coefficient for each layer of neural network. In this section, we will introduce the regularization techniques with the same regularization coefficient for each layer of neural networks. In practice, we will consider their suggestion of using different regularization coefficients for each layer to optimize hyperparameter.

## Disagreeing Neighbours

The main assumption of exchange rate return prediction problem is that if two input samples are very similar to each other, it is likely that these inputs should act similarly on upcoming time step. Similar inputs are called as **neighbours** referring to the nearest neighbour approach based on Euclidean distance. In case neighbour samples belong to unique classes, there is no problem. Otherwise, it will result to overfitting problem. In this paper, such neighbour samples having different output are called as **disagreeing neighbours**. As it is discussed earlier, this problem is related to capacity of the model. To overcome this issue, corresponding timestamps of disagreeing neighbour samples can be investigated. For the sake of simplicity, such investigation procedure is out-of-scope of this study. However, we can’t deny the fact of such generalization problem. That’s why, we need to have another classification model which can learn the disagreeing neighbourhood.

## Balancing Dataset

In exchange rate prediction problem, it is very likely to encounter with imbalanced datasets. Training dataset might have from one class more excessive than other class. The classes that are excessive are called as **majority** **classes**. The classes that have relatively less amounts are called as **minority** **classes**. **Balancing** a dataset is set of procedures to resample the training dataset for the purpose of having either equal or almost equal number of samples per each class. In order to balance a dataset, either minority class samples should be augmented (**over-sampling**) or majority class samples should be dropped (**under-sampling**). In this study the list of under-sampling methods is shown in Table 1.1.

* 1. Under-sampling Methods

|  |  |
| --- | --- |
| Method | Description |
| Condensed nearest neighbour |  |
| Edited nearest neighbour |  |
| Repeated edited nearest neighbour |  |
| All k-nearest neighbour |  |
| NearMiss |  |
| Neighbour cleaning rule |  |
| Random under sampler | Resampling randomly from each class |
| Tomek links |  |

In this study, a Python package called **imbalanced-learn** (Lemaître, Nogueira, & Aridas, 2017) is used for balancing algorithms. Imbalanced-learn is fully compatible with the library **scikit-learn** (Pedregosa, et al., 2011). As it is mentioned earlier, disagreeing neighbour samples should be handled with another classification model. For return prediction model, training dataset should have disagreeing neighbour samples as less as possible.

## Fast Fourier Transform

## Weight Initialization

Initial weights of deep learning models have importance on performance of the model. Optimizing the exact values of initial weights require high computation cost. That’s why, to improve the performance of a model, following conditions should be considered while initializing the weights:

1. Make sure that values of each incoming weights to a neuron are unique.
2. Random initialization based on a high-entropy distribution

WRITE FROM PAGE 302

## Regularization

**regularization** is a type of weight decay regularization to reduce the weights of the model towards origin based on second norm of the weights. regularization is called also with different name such as **ridge regression** or **Tikhonov regularization**. Purpose of regularization is to decay the weights of neural networks to the zero but not exactly zero.

Let be the cost function that has input to calculate target , based on the model weights , then we can calculate the regularized objective function, ,as shown in Formula (##):

|  |  |
| --- | --- |
|  |  |

where – is the weight decay rate,

– is the quadratic (second norm) penalty term.

Gradient of the regularized objective function with respect to weights is calculated as shown in Formula (##):

|  |  |
| --- | --- |
|  |  |

Weight update based on gradient is calculated as shown in Formula (##):

|  |  |
| --- | --- |
|  |  |
|  |  |

where is the learning rate.

regularization will decay the weights as much as in addition to gradient descent. The amount of decay is inversely proportional to the eigenvalues of Hessian matrix. If eigenvalue of a weight is small, it means that this weight’s contribution to cost function is unimportant. That’s why, it decays such weights towards zero but not exactly zero. If eigenvalue is large, regularization preserves corresponding weight. Since eigenvalues are representing covariance between weights and target output, regularization tries to shrink the weights that have low covariance with target output.

## Regularization

**regularization** is a type of weight decay regularization based on the first norm of the weights. regularization is known as **LASSO regression** as well (Tibshirani, 1996). Regularized objective function, , is calculated as shown in Formula (##):

|  |  |
| --- | --- |
|  |  |

where – is the summation of absolute values of individual weights.

Gradient of the regularized cost function is calculated as shown in Formula (##):

|  |  |
| --- | --- |
|  |  |

where – is the function that returns the sign of the weights. Unlike regularization, regularization doesn’t decay the weights based on their magnitude but based on their sign. This leads that regularized weights may have equal to zero. This is a **feature selection** mechanism where unimportant weights are switched off to zero.

## Early Stopping

The criteria to stop a training procedure is a concern in machine learning. **Early stopping** is a mechanism that stores the training and validation errors during different iterations. Once the validation loss starts to have U-shape with still decreasing training loss, the mechanism waits for a fixed number of iterations. Number of iterations to wait is a hyperparameter that is called **patience**. After patience iteration, if still validation loss starts to increase, then early stopping mechanism goes back to the parameters that were used during beginning of the U-shape. Early stopping is a regularization because it reduces the bias via stopping the training process after patience period to improve the variance. The most important advantage of early stop mechanism is the simplicity of it. It can improve the model without making architectural changes on the model. Early stopping can be used in hyperparameter selection process as well as parameter selection process. Early stopping algorithm is presented in Fig. 1.12. The parameters that early stopping mechanism returns are used to train both training and validation data.

Graphical user interface, text, application

Description automatically generated

* 1. Pseudocode for Early Stop Mechanism

## Data Augmentation

MENTION ABOUT GENERATIVE ADVERSARIAL NETWORKS (GAN) FOR ENCODING AND AUGMENTING NEW TRAINING DATA.

## Noise Injection

MENTION ABOUT SUCH NOISE INJECTION EFFECT ON TIME SERIES PREDICTION INPUTS.

## SMOTE

MENTION THAT SMOTE FOR CATEGORIAL VARIABLES BECAUSE X\_TIME IS CATEGORICAL DATA IN DEED.

## Bagging

**Bagging** is an ensemble technique in which multiple models are trained with a different samples of main training dataset. The other common way of naming the bagging method is **bootstrap aggregation**. The way of sampling sub-training datasets is **row sampling with replacement** method. The term **replacement** refers to include an observation (row) multiple times in a sample. In addition to raw sampling, **feature sampling** also can be applied as sampling method. However, feature sampling can’t have a replacement. As a result of training with bagging, there are multiple models that are ready for inference. A final model output can be calculated as mean of output of all ensembled models. The term **bootstrapping** refers the process of sampling while the term **aggregation** refers to the aggregation function to identify the final output of ensembled models. As it is mentioned above, in this study mean aggregation function will be used however other aggregation functions such as maximum, median, majority vote etc. also can be used.

## Dropout

D**ropout** deactivates non-output units based on a ratio called **dropout rate.** Due to this deactivation, there are multiple sub-models occur. It converts a model to an ensemble model which consists of sub-models of it. It applies model averaging to multiple sub-models. In this context, dropout is like bagging. However, there are 2 main advantages of dropout compared to bagging:

1. It does not change the architecture of the model. It just deactivates a unit.
2. It is very easy to compute because deactivating a unit means multiplying the output of unit by zero.

Each unit is deactivated with the same dropout rate but independently from each other. Authors of (Goodfellow, Bengio, & Courville, 2016) states the typical dropout rate for input and hidden units are 0.8 and 0.5 respectively. Dropout is related with the size of model. Since dropout is reducing the capacity of model by deactivating units, the size of neural network should be larger. It means that in small dataset, computation cost of dropout can negligible, but in larger datasets, computation cost should be taken into consideration.

Dropout adds a noise to hidden units. It is very similar to noise injection. Main difference is that noise injection injects noise to raw values of input dataset while dropout adds noise to hidden units as well.

## Adversarial Training

One of the challenges in machine learning that sometimes even very well performed models are giving different outputs for very similar inputs (Szegedy, et al., 2014). **Adversarial training** is the regularization strategy that uses **adversarial samples** that are modified versions of original samples with small but misleading perturbations. Purpose of adversarial sample is to encourage neural network to make incorrect prediction. Usually, adversarial training is used in classification problem of image recognition datasets. In order to apply adversarial training, gradient direction of the function on the point of sample is added to sample as shown in Formula (##):

|  |  |
| --- | --- |
|  |  |

where – is the adversarial sample which is neighbour of the original sample ,

– small **perturbation rate** of original sample.

Recall that improvement of cost function is opposite direction of gradient since it is a minimization problem. That’s why, adding a sample on the reverse direction of improvement will encourage the neural network to predict the adversarial sample incorrectly. This small perturbation will confuse the model. However, by adding this sample to training dataset, we are telling model that original sample and perturbated sample are like each other. That’s why, we try to encourage model to predict a constant value for both original and perturbated samples. In order to apply adversarial training, **nsl.configs.make\_adv\_reg\_config** class of Tensorflow (Abadi, et al., 2015) is used.

## Transfer Learning

ANOTHER NAME “PARAMETER SHARING & TYING”

CONSIDER ABOUT BUILDING MULTIPLE MODELS SUCH THAT MODEL\_! OUTPUT IS T+1, MODEL\_2 OUTPUT ARE T+1 AND T+2. CONSIDER THAT PARAMETERS SHOULD BE SIMILAR ON ONE ANOTHER.

OR CLASSIFICATION + REGRESSION

WHILE SOME EXCHANGE RATE HAS N ROW, ANOTHER ONE HAS 5N ROW, WE CAN SHARE ACROSS THEM.

<https://medium.com/@lmayrandprovencher/building-an-autoencoder-with-tied-weights-in-keras-c4a559c529a2>

SHOWS PARAMETER TYING FOR ENCODER AND DECODER OF AUTOENCODER.

Initializing the weights of your network to the ones of another network with the same architecture pre-trained on a large, generic dataset. I think, we can use between multiple exchange rates.

## Batch Normalization

## Early Stopping

AS MENTIONED ON PAGE 247, EARLY STOPPING CAN BE USED AS HYPERPARAMETER SELECTION ALGORITHM. BUILD A METHODOLOGY IN A WAY THAT WE USE EARLY STOPPING IN HYPERPARAMETER SELECTION PROCESS. (I THINK, ONLINE HYPERPARAMETER SELECTION)

# HYPERPARAMETER OPTIMIZATION

1. Identify cost function, output & input datasets
2. Build a model with single time step to identify the hyperparameters
   1. Hyperparameters to optimize for underfitting scenario – only with training dataset. Use R2 value to determine.
      1. Number of hidden neurons
      2. Number of hidden layers
      3. Weight initialization
      4. Hidden units
      5. Output units
   2. After underfitting is prevented, we will see probably overfitting. In order to avoid from overfitting, we need to apply regularization techniques.
      1. Fit training, validation, and test distributions with each other.
         1. Collect more features
         2. Collect more data
         3. Data augmentation
      2. Batch size, epoch size
      3. Dropout
      4. Ridge and Lasso regression
      5. Early stopping
      6. Bagging
      7. Adversarial training
   3. Hyperparameters during gradient optimization

AIM TO MAKE HYPERPARAMETER OPTIMIZATION NOT TO WHOLE DATASET OR WHOLE EPOCH. BUT TRY TO GENERATE SEARCH SPACE AFTER SEVERAL EPOCHS OR SOME PART OF DATASET.

REGULARIZATION COEFFIEICENT FOR EACH LAYER.

## Design of Experiments (DoE)

### Full Factorial Design

### Steepest Descent

### Response Surface Methodology

## Hyperband Tuner

# Portfolio OPTIMIZATION MODEL

Portfolio optimization problem is a special type of investment problem to select the optimal mix of opportunities that will maximize return while meeting requirements set by the investor and the market (Taha, 1997). In this study, the decision variables of the optimization problem are and representing the amounts of the positions and types of the positions respectively, for the exchange rate to open the position on future time step and to close the position on future time step . There are 3 types of positions: Buy, Sell or Do nothing. In this context, objective function of the mathematical model is maximizing the total return of investments calculated in Formula (##):

|  |  |
| --- | --- |
|  |  |

where – return of investment.

Formula (##) demonstrates the calculation of return of investment (). Return of investment is the difference between the return gained from price difference and loss due to spread.

|  |  |
| --- | --- |
|  |  |

where – price difference that is calculated with the Formula (##);

– the ratio of spread at the time step when the position is opened. It is calculated with Formula (##).

|  |  |
| --- | --- |
|  |  |
|  |  |

where – closing price at the time step when the position is closed;

– opening price at the time step when the position is opened;

– spread at the time step when the position is opened.

Both and are the ratios respective to in order to eliminate the impacts that can occur due to different scales of exchange rates.

The constraint that is calculated with Formula (##) ensures that investment amount is set to zero in case type of investment is “Do nothing”.

|  |  |
| --- | --- |
|  |  |

where – is a very big number.

Balances in each time step with the Formula (##):

|  |  |
| --- | --- |
|  |  |

where – the balance on time step ;

– the balance of previous time step.

Constraints that imply the nonnegativity of returns and balances are represented with the Formula (##) and Formula (##) respectively:

|  |  |
| --- | --- |
|  |  |
|  |  |

Constraints that represent the borders of the decision variables are shown in Figure (##) and Figure (##). Amounts of the investments are nonnegative values. Types of the investment are -1, 0 and 1 that denote respectively to sell, to do nothing and to buy.

|  |  |
| --- | --- |
|  |  |
|  |  |

# Genetic Algorithm (GA)

# literature review

**ADD HISTORICAL NOTES ABOUT DEEP LEARNING NEURAL NETWORKS FROM THE BOOK (PAGES BETWEEN 225-227)**

# APPLICATION

## Exploratory Data Analysis

# RESULTS AND CONCLUSION

Following steps can be considered as further research areas:

1. Using second order optimizers such as conjugate gradients, Broyden-Fletcher-Goldfarb-Shanno algorithms could be tuned with hyperparameter optimizers. However, currently there are not second order Keras optimizers. That’s why, these optimizers should be developed as custom optimizers.
2. Hyperparameter optimization procedure could be applied during some epochs of training.
3. Defining new types of activation functions can be still an interest for future studies. A special activation function for exchange rate prediction can be considered.
4. The performance of automatic differentiation in back-propagation algorithm could be improved with custom definition of sequence of gradient calculations.
5. Social media and financial news dataset can be processed via using natural language processing techniques.
6. There were not many resources where more valuable financial indicators such as open interest are shared by brokers. In the future, it is expected that open interest indicator will be shared more commonly by brokers. In future studies, this indicator would be included as future step to predict features financial instruments.
7. Cloud based cluster systems can be used to run the experiments.
8. Statistical process control of prediction results can be applied to monitor the performance of prediction. In case the model starts to predict with error higher than a threshold, alerts can be generated to perform whole processes.
9. A strategy should be built on during the prediction and optimization durations.
10. A separate prediction algorithm for spread values could be used.
11. In portfolio optimization algorithm, swap prices could be considered.
12. Pending orders could be considered in optimization. Also, stop loss and take profit options also could be considered.
13. Additional features can be predicted such as highest price, lowest price, spread etc.
14. Support and resistance points can be used as inputs.

appendixes

[Appendix 1](#_Toc111302171)

[Experiments of Full Factorial Design](#_Toc111302172)



Experiments of Full Factorial Design