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 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.

# GRADIENT BASED OPTIMIZATION algorithms

**Gradient based optimization algorithms** are used to optimize the weights of deep learning models. 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. Under this section, optimization algorithms that are used for training the weights of deep learning models are mentioned. Deep learning models are usually not convex models since the term “deep” implies multiplying the inputs with multiple powers of matrices. The target of optimization in deep learning models are to convert non-convexity of them to convexity. In this context, **Taylor series approximation** method is used because Taylor Series approximation converts a non-polynomial function to a polynomial function. In order to elaborate gradient based optimization algorithms, some calculus and algebraic methodologies are mentioned briefly.

## 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, Deep Learning, 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.

## 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, Deep Learning, 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.

(Goodfellow, Bengio, & Courville, Deep Learning, 2016) recommends random selection of minibatches. 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. In the book written by (Goodfellow, Bengio, & Courville, Deep Learning, 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 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.

Text

Description automatically generated

* 1. Psuedocode for Stochastic Gradient Descent

## Adam Optimization

Adam optimization (Kingma & Ba, 2014) is a commonly used learning algorithm to identify optimum weights of neural networks. DESCRIBE

# 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, 2008). 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.

# DEEP LEARNING FOR TIME SERIES PREDICTION

## 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.

## Distribution of Train-Test-Validation Datasets

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 data which is the main motivation of time series prediction. In this context, train, validation, and test datasets should be independently and identically distributed (i.i.d) with each other. Independence means that data points (observations) should not be impacted from each other. This is not very applicable in time series prediction. Because in time series there are autocorrelations which means that it is likely that one observation on is impacted with the previous observations (e.g. ). In order to overcome this solution 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.

Second condition of i.i.d is “identically distributed” which means that data distributions of train, validation and test datasets are same. In order to assess identicality of two distributions following techniques can be used:

* Apply Kolmogorov-Smirnov test to identify if these datasets are from same distribution or not.
* Use Kullback-Leibler 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.

## 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. 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.

## Cross Validation for Time Series

THERE ARE FEW APPROACHS OF CROSS-VALIDATION (NOT K-FOLD) SPECIFICALLY DEVELOPED FOR TIME SERIES PREDICTION.

## Cost Function

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.

### Information Theory

# DEEP LEARNING MODELS

## 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.

### Weight Initialization

In MLPs, all weights are initialized to small random values. Biases can be initialized to zero or small positive random values.

## 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)

# REGULARIZATION TECHNIQUES FOR DEEP LEARNING

We took the section of the book written by (Goodfellow, Bengio, & Courville, Regularization for Deep Learning, 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 a 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.

## REGULARIZATION

regularization is a type of weight decay regularization to reduce the weights of the model towards origin. regularization is called also with different name such as “ridge regression” or “Tikhonov regularization”. Let be the objective function that has input to calculate target , based on the model weights , then we can calculate the regularized objective function, ,as shown in Formula (##):

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

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

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

# HYPERPARAMETER 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

As an application, 5 symbols of cryptocurrencies are taken into consideration: BCHUSD, BTCUSD, ETHUSD, LTCUSD and RPLUSD. The frequency of the data is 30 mins. Market data is imported for the date interval 2021-09-01 and 2022-03-10. In order to import market dataset, MetaTrader 5 (MetaQuotes, 2000) platform’s API (Ltd, 2022) is used. First Prudential Markets (FP Markets) (Markets, 2005) is used as broker to fetch market data. Descriptive statistics of closing prices of imported datasets are demonstrated in Table 1.1.



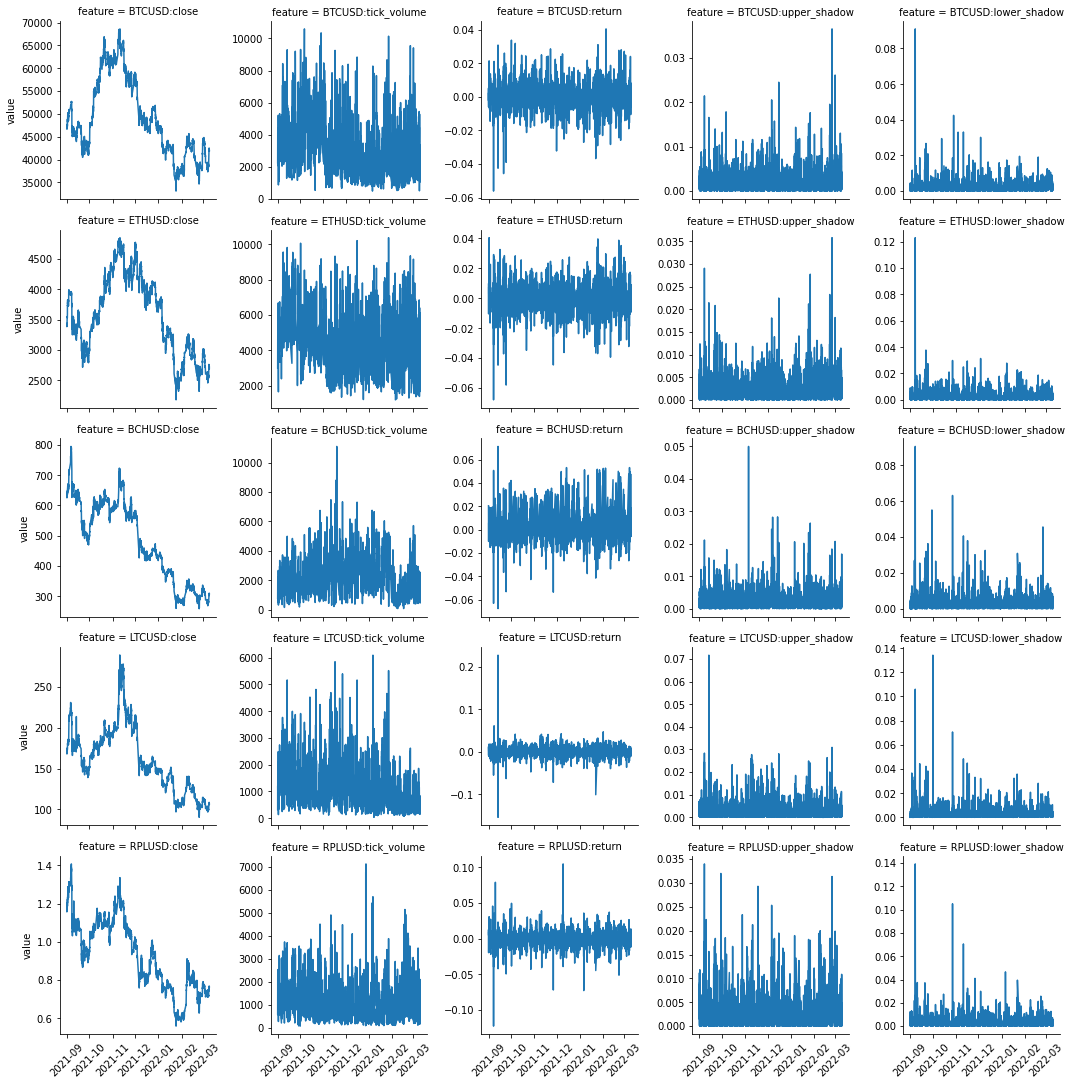
Descriptive Statistics of Closing Prices

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
|  | BCHUSD | BTCUSD | ETHUSD | LTCUSD | RPLUSD |
| count | 6516.00 | 6516.00 | 6516.00 | 6516.00 | 6516.00 |
| mean | 481.58 | 49089.19 | 3563.92 | 161.14 | 0.93 |
| std | 132.84 | 8509.96 | 649.56 | 39.90 | 0.18 |
| min | 259.73 | 33098.05 | 2181.45 | 90.39 | 0.56 |
| 25% | 355.08 | 42633.72 | 3053.48 | 131.84 | 0.78 |
| 50% | 478.23 | 47299.22 | 3548.43 | 154.77 | 0.93 |
| 75% | 599.14 | 56888.63 | 4096.23 | 187.79 | 1.09 |
| max | 68609.11 | 4845.78 | 794.64 | 288.53 | 1.41 |

A picture containing window

Description automatically generated

* 1. Data Distributions for Some Features



* 1. Trends of Some Features

A picture containing text

Description automatically generated

* 1. Corralation Analysis

A picture containing window, building

Description automatically generated

* 1. Scatter Plot for BTCUSD Features with Other Currencies

Each symbol is trained with 4 prediction models: Multi-layer perceptron (MLP), Long-short term memory (LSTM), Convolutional encoder-decoder model (Conv-EncDec) and Luong’s Attention (Luong-Att).

## Full Factorial Experiments

In each prediction model, hyperparameters that are considered in DOE are given in Table 1.2 with the levels of them. For each cryptocurrency symbol & predictive model types a full factorial design of the above-mentioned hyperparameters is created. Full factorial design consists of 16 runs with 4 repetitions to minimize the impact of random weight initializations of the predictive models.



Levels of Factors

|  |  |  |
| --- | --- | --- |
| Hyperparameter | Lower Level | Upper Level |
| Batch size | 60 | 70 |
| Number of hidden neurons | 10 | 14 |

Full factorial design is executed for each symbol and predictive models and response values of experiments are calculated based on custom metric function. Experimental results of full factorial design for each symbol and predictive model is presented in Appendix 1. After completing all runs of full factorial design, a 1st order mathematical model is created. Coefficient of determination scores of 1st order mathematical models are summarized in Table 1.3.



Coefficient of Determination of 1st Order Models

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
|  | MLP | LSTM | Conv-EncDec | Luongs-Att |
| BCHUSD | 0.14 | 0.3 | 0.43 | 0.34 |
| BTCUSD | 0.04 | 0.07 | 0.14 | 0.02 |
| ETHUSD | 0.05 | 0.04 | 0.13 | 0.23 |
| LTCUSD | 0.27 | 0.16 | 0.14 | 0.13 |
| RPLUSD | 0.09 | 0.36 | 0.10 | 0.12 |

Based on the first order model, steepest descent process is executed until the custom metric value starts to increase. Iterations of steepest descent process for each model type and exchange rate pair is shown as a bubble plot in Fig. 1.6.

Calendar

Description automatically generated

* 1. Bubbe Plot of Steepest Descent Process

## Response Surface Methodology

Factor levels identified by steepest descent process are considered as central point of central composite inscribed (CCI) design. CCI design consists of 8 central points. Experiment results of CCI designs are shown in Appendix 2.

After execution of CCI design, a 2nd order mathematical model is created. 2nd order mathematical model consists of interaction and square effects of the factors as well as linear affects. Coefficient of determination of 2nd order models are shown in Table 1.4.



Coefficient of Determination of 2nd Order Models

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
|  | MLP | LSTM | Conv-EncDec | Luongs-Att |
| BCHUSD | 0.08 | 0.42 | 0.18 | 0.25 |
| BTCUSD | 0.36 | 0.77 | 0.29 | 0.4 |
| ETHUSD | 0.45 | 0.36 | 0.3 | 0.31 |
| LTCUSD | 0.15 | 0.44 | 0.84 | 0.19 |
| RPLUSD | 0.33 | 0.26 | 0.15 | 0.3 |

Based on the 2nd order mathematical model, a grid search algorithm is applied to find the optimum hyperparameter configuration within the defined interval of CCI design. Surface plots of 2nd order models are given in Fig. 1.7.

Shape, arrow

Description automatically generated

* 1. Surface Plot of Grid Search

Optimum configurations that are identified via RSM are given in Table 1.5.



Optimum Configurations via RSM

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
|  | MLP | LSTM | Conv-EncDec | Luongs-Att |
| BCHUSD | [60, 10] | [55, 4] | [60, 13] | [85, 14] |
| BTCUSD | [64, 11] | [62, 8] | [62, 12] | [66, 6] |
| ETHUSD | [60, 10] | [60, 10] | [68, 13] | [67, 6] |
| LTCUSD | [60, 13] | [69, 10] | [66, 15] | [60, 10] |
| RPLUSD | [67, 8] | [56, 8] | [68, 10] | [70, 15] |

Values are presented in [Batch Size, Number of Hidden Neurons] format.

Predictive models are created for each optimum configuration. Results for test dataset is demonstrated in Table 1.6. The configuration that has the minimum custom metric value is selected for simulation.



Test Results of Optimum Configurations via RSM

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
|  | MLP | LSTM | Conv-EncDec | Luongs-Att |
| BCHUSD | 47.825834 | 55.075167 | 68.721883 | 80.61469 |
| BTCUSD | 36.158233 | 46.668726 | 43.685923 | 1838.94727 |
| ETHUSD | 51.776433 | 52.527063 | 51.703628 | 71.787779 |
| LTCUSD | 52.566987 | 61.888913 | 110.833586 | 127.491509 |
| RPLUSD | 69.440883 | 68.174606 | 71.67011 | 89.41441 |

## Hyperband

Optimum configurations handled via Keras Tuner are shown in Table 1.7.



Optimum Configurations via Hyperband

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
|  | BCHUSD | BTCUSD | ETHUSD | LTCUSD | RPLUSD |
| MLP |  |  |  |  |  |
| LSTM |  |  |  |  |  |
| Conv-EncDec |  |  |  |  |  |
| Luong-Att |  |  |  |  |  |

## Simulation

Optimum models are used to predict each exchange rate for the simulation time interval. The average custom metrics of predictions are shown in Fig. 1.8.

Chart, bar chart

Description automatically generated

* 1. Average Custom Metrics of Simulation Predictions via DOE

It is assumed that closing price of a time step is opening price of a previous time step.

A picture containing building, window, surrounded

Description automatically generated

* 1. Multi-Step Comparison of BCHUSD Simulation

A picture containing window, building, surrounded

Description automatically generated

* 1. Multi-Step Comparison of BTCUSD Simulation

A picture containing window, shoji, building, surrounded

Description automatically generated

* 1. Multi-Step Comparison of ETHUSD Simulation

A picture containing window, building, surrounded

Description automatically generated

* 1. Multi-Step Comparison of LTCUSD Simulation

A picture containing building, window, light, surrounded

Description automatically generated

* 1. Multi-Step Comparison of RPLUSD Simulation

## Portfolio Optimization

# RESULTS AND CONCLUSION

Following steps can be considered as further research areas:

1. 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.
2. The performance of automatic differentiation in back-propagation algorithm could be improved with custom definition of sequence of gradient calculations.
3. Social media and financial news dataset can be processed via using natural language processing techniques.
4. 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.
5. Cloud based cluster systems can be used to run the experiments.
6. 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.
7. A strategy should be built on during the prediction and optimization durations.
8. A separate prediction algorithm for spread values could be used.
9. In portfolio optimization algorithm, swap prices could be considered.
10. Pending orders could be considered in optimization. Also, stop loss and take profit options also could be considered.
11. Additional features can be predicted such as highest price, lowest price, spread etc.

appendixes

[Appendix 1](#_Toc111302171)

[Experiments of Full Factorial Design](#_Toc111302172)



Experiments of Full Factorial Design