Deep learning toolbox for classification prediction models

Zacharie Guibert

August 2022

Supporting theoretical note to main paper:
LSTM and GRU neural networks as a support to investment and risk management strategies

Abstract

Main paper **LSTM and GRU neural networks as a support to investment and risk management decisions** [1] focuses on a classification prediction exercise in order to support investment and risk management strategies using neural networks. Two variations of Recurrent Neural Networks (RNN), the Long-Short Term Memory (LSTM) and the Gated Recurrent Units (GRU) models are employed to predict whether the volatility spread between the realized and implied volatility (RV-IV spread) ends up positive or negative on the next trading day. In particular, LSTM and GRU models adopt the same structure design - prior tuning of hyperparameters, such that their prediction performances can be compared. SP500 Index and the Eurostoxx 50 Index are used as support for the classification prediction exercise and design of investment and risk management strategies. For both indices, the GRU model outperforms the LSTM model. Both investment and risk management strategies are built around the interpretation that a positive RV-IV spread prediction is indicative of a bear market to come as a result of upcoming volatility. The investment strategy, derived from SP500 GRU predictions, highlights that the GRU model does well in identifying a bear market and avoiding losses, albeit initially struggling to identify a bull market. The risk management strategy incorporates Eurostoxx 50 GRU predictions and machine learning techniques applied to volatilities and correlations to derive forward-looking risk measures in the case of an investment portfolio.

Several deep learning concepts are employed in this paper, which are key for the design of the prediction models. The objective of this document is to explain how deep neural networks learn and give further insights on all aspects of the machine learning and deep learning processes that are covered in the main paper [1].

1 Introduction

1.1 Context

The stock market reflects the interaction of buyers and sellers, influenced by their beliefs and the economic situation. These market players may be inclined to buy or sell with clear trends at times, or act erratically at others as uncertainty rise. Therefore, being able to quantify the level of randomness in the stock market gives insights in the general behavior of the participants and trends. The concept of randomness is key in finance: randomness can be simply defined as the lack of pattern. A financial instrument would be qualified as:

- predictable if it always follows the same price pattern: market participants are able to have an idea of the next price.
- random if there is no repetition of the pattern: market participants have no way to determine the next price.

Naturally, mathematics have been from the start a support to quantify, manage or predict randomness in financial markets. Starting from the 50's, econometric science provided market players with methods and techniques to quantify, model and predict randomness. The GARCH model (Generalized AutoRegressive Conditional Heteroskedasticity), developed in 1982 by Robert F. Engle, described an approach to estimate volatility in financial markets [2]. On the other side, other theories emerged, such as the random walk theory, defending that changes in stock market prices are random and thus cannot be predicted. This theory proved to be key later for modeling financial assets through Monte-Carlo simulations.

Today, machine learning plays a key role in our day-to-day, being used by the car industry, biological science, administrations and of course finance. Despite it is no longer so new - the first case of a neuron network model on computer was developed in 1943 by the neurophysiologist Warren McCulloch and the mathematician Walter Pitts, the increase of available data combined to a considerable rise of computer capacities contributed to democratize machine learning in many different fields. As of today, machine learning can be defined as a subset of data sciences that uses statistical models to draw insights ans make predictions. Nevertheless, the key difference about machine learning solutions is that they learn from experience without being explicitly programmed. In traditional econometric models and classical programming, rules and data are fed to the engine in order to obtain answers. On the contrary, in machine learning, data and answers are fed to the engine in order to obtain the rules. From the rules, modeling and predictions can be made. In short, **machine learning is all about learning some useful representation of the input data in order to make predictions**.

Another key difference which explains the rapid expansion of machine learning over classical statistics in 2022 comes from the fact that unlike econometric models which need no or a few data points to run, machine learning models are data hungry. In general, the more data fed to a machine learning model, the more accurate the results - so some caveat however, as illustrated in sections 3.2 and 3.3 below. As the finance industry sits on gigantic amounts of data, it makes it a natural eligible candidate for machine learning models.

1.2 Artificial Intelligence, Machine Learning and Deep Learning

Although these three expressions are part of the day-to-day vocabulary, it is crucial to understand their meaning to grasp all concepts in this thesis, as they happen to be often misused. As illustrated by the figure below, deep learning is part of machine learning, which is part of artificial intelligence. More particularly,

- Artificial intelligence (AI) is a technique that enables the machine to mimic the human behavior.
- Machine learning (ML) is a technique to achieve artificial intelligence through algorithms trained with data.
- Deep learning (DL) is a type of machine learning inspired by the structure of the human brain - the structure is called a neural network.

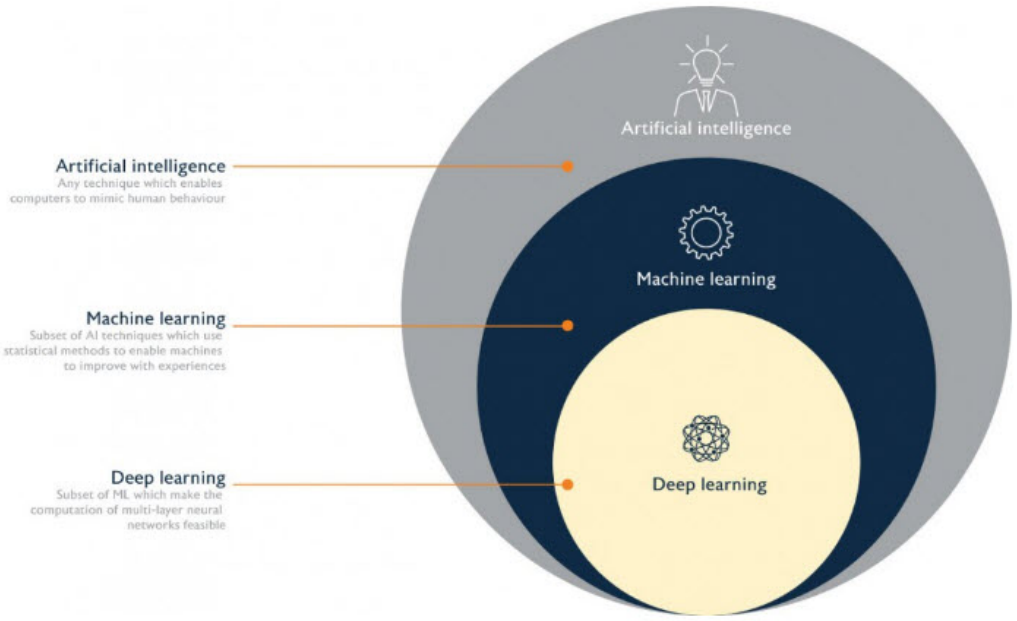

Figure 1: AI, ML and DL (*source: Medium*)

1.3 Objective

The objective of this note is to illustrate how deep neural networks learn. More precisely, it summarizes all the underlying theory which is necessary to support the main paper, from feature selection to model implementation, tuning of hyperparameters and regularization, among others.

2 Machine learning fundamentals

2.1 Learning process

Machine learning can be summarized in an optimization problem, as the main objective is to minimize the loss score by varying the weights, which is how the learning happens.

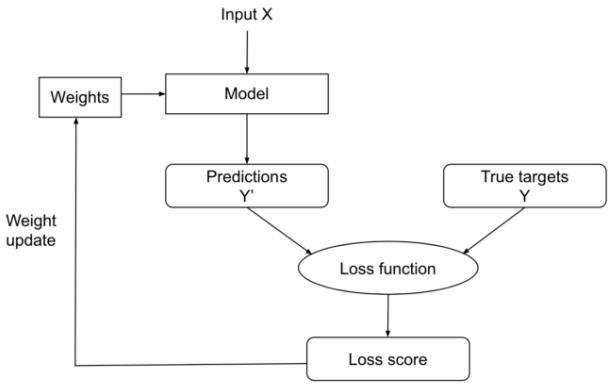

Figure 2: Machine learning process (*source: Deep Learning with Python, F. Chollet (2017)*)

The learning process for neural networks is more complex, as it involves additional distinct layers and an optimizer to update the weights. In other terms, this describes the process through which the model learns.

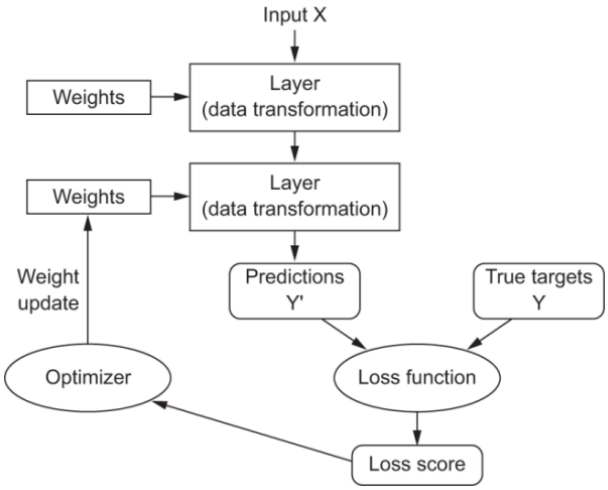

Figure 3: Deep learning process (*source: Deep Learning with Python, F. Chollet (2017)*)

Machine Learning can be divided in two sub-parts: regression and classification. In mathematical terms,

- Classification predictive modeling is the task of approximating a mapping function f from input variables X to a discrete output variable y.
- Regression predictive modeling is the task of approximating a mapping function f from input variables X to a continuous output variable y.

As a reminder, the main paper is dedicated to classification, therefore regression will not be developed further in this note.

2.2 Classification

"Classification is the task of learning a function that maps or classifies a dependent variable into one of several predefined classes, with the objective to learn a model that accurately predicts an independent discrete variable" - Bratko, 2000. The process starts with predicting the class of given data points. Classes are often referred to as target, label or categories. Classification predictive modeling is the task of approximating a mapping function f from input variables x to discrete output variables y. In short, **the main goal of classification is to identify which class or category the new data will fall into**.

Binary classification is a supervised learning algorithm that categorizes new observations into one of two classes. A classifier utilizes some training data to understand how given input variables relate to the class it aims to predict. When the classifier is trained accurately, it can be used to predict the respective classes in the future.

3 Machine learning challenges

3.1 Bias-variance trade-off

The bias-variance trade-off is a problem faced in machine learning and statistics, where the bias and the variance of a predictive model cannot be optimized at the same time. Many models usually have parameters that have to be estimated, i.e. fitted, against data.

- **Increasing the model complexity increases its variance and reduces its bias**.
- **Reducing the model complexity reduces its variance and increases its bias**.

How many parameters a model has and how the model is defined determines the model complexity. A perfect model would have both low bias and low variance since those two traits make models generalize better. However, as illustrated by the chart below, this is not possible in reality, hence the notion of trade-off between bias and variance [3].

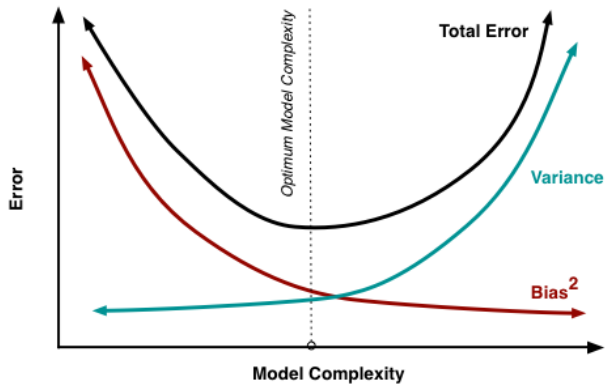

Figure 4: Bias-variance trade-off (*source: Understanding the Bias-Variance trade-off, S.Fortmann (2012)*)

The figure below maps the different types of machine learning models as a function of flexibility (e.g. complexity) and interpretability (e.g. model error). The transposition of the figure 5 on the figure 4 indicates that neural networks and deep neural networks have a low bias and a high variance. In short, **neural networks and deep neural networks are proned to overfitting**.

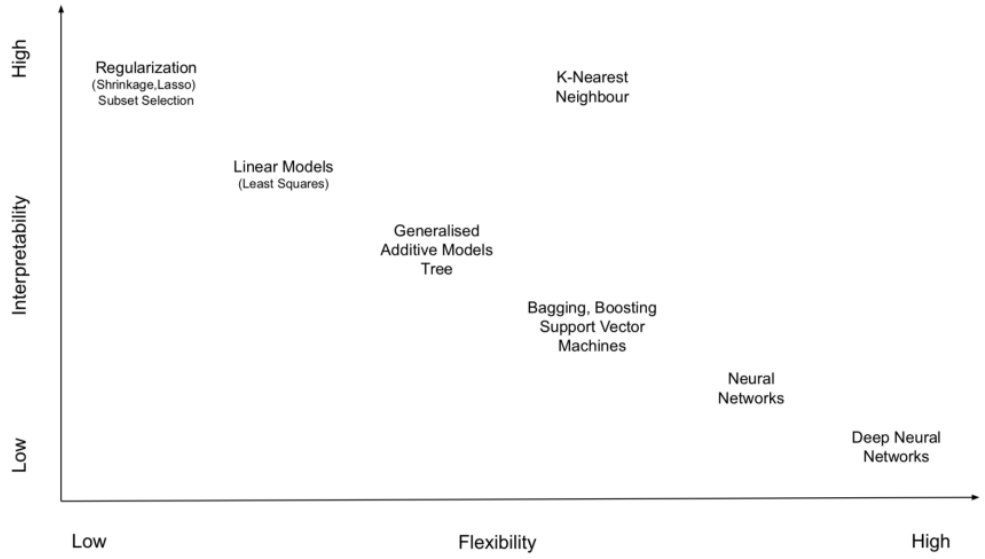

Figure 5: Model interpretability-flexibility trade-off (*source: Introduction to statistical learning (2013)*)

3.2 Overfitting

Artificial neural networks (neural networks and deep neural networks) are very good at fitting data because of their many parameters. However, this comes at a cost, as neural network models tend to have high variance which causes them to overfit data. Overfitting simply means that a model has high variance and is able to fit the data so perfectly that it becomes a problem when trying to make predictions for unobserved data. In short, **neural network models likely fit the training data perfectly and get very low in-sample errors, but when making out-of-sample forecasts, they might get a very high error because of the model overfitting the training data**. For this reason, datasets are generally split between a training and a test part in order to obtain at the end models which perform well out-of-sample.

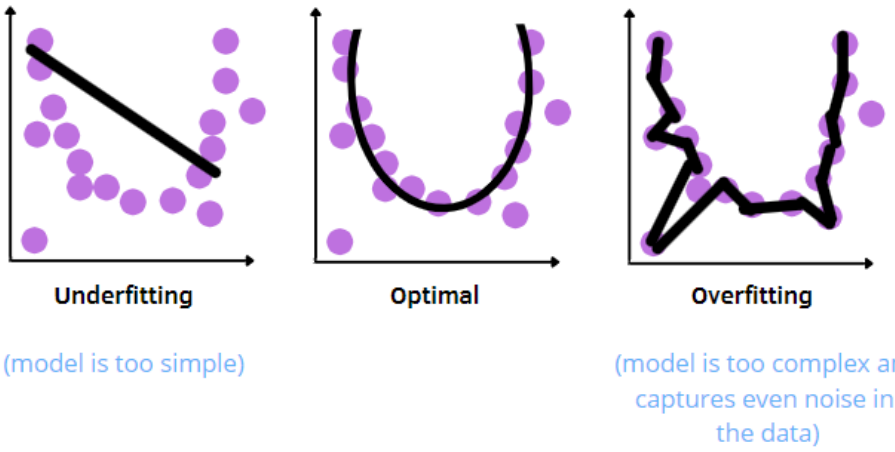

Figure 6: Under and overfitting concepts (*source: Medium*)

A consequence of overfitting is that the model is not able to generalize properly, and cannot make accurate forecasts [4]. Regularization techniques allow to mitigate the overfitting problem.

3.3 Regularization

Regularization is a technique used in machine learning and statistics to minimize the risk of overfitting the data. In short, **regularization makes slight modifications to the learning algorithm such that the model generalizes better, which in turn improves the model performance on the unseen data as well**. From a practical perspective, in machine learning, regularization penalizes the coefficients. In deep learning, regularization penalizes the weight matrices of the nodes. Many different types of regularization can be used in the context of deep learning. Among the most popular,

3.3.1 L1 (Lasso) and L2 (Ridge)

L1 and L2 are the most common types of regularization. L1 and L2 update the general cost function by adding another term known as the regularization term.

$$Cost\ function = Loss\ function + Regularization\ term$$

For example, the loss function could be the binary cross entropy for a classification problem, or the mean square error for a regression problem. The values of weight matrices decrease due to the addition of this regularization term, as it assumes that a neural network with smaller weight matrices leads to simpler models. Therefore, it will also reduce overfitting to some extent.

L1 and L2 differ by the regularization term:

- The L1 penalty term not only shrinks the coefficients, but shrinks some of them to zero (useful for feature selection): $\lambda \sum_{j=1}^{n} |w_j|$, where λ is the regularization penalty
- The L2 penalty term not only shrinks the coefficients, but alters the cost function by adding a L2 penalty equivalent to square of the magnitude of the coefficients (helps to reduce the multicollinearity): $\lambda \sum_{j=1}^{n} w_j^2$, where λ is the regularization penalty

3.3.2 Dropout

At every iteration, **the dropout technique randomly selects some nodes and removes them along with all of their incoming and outgoing connections** [5]. As a consequence, each iteration has a different set of nodes, which results in a different set of outputs.

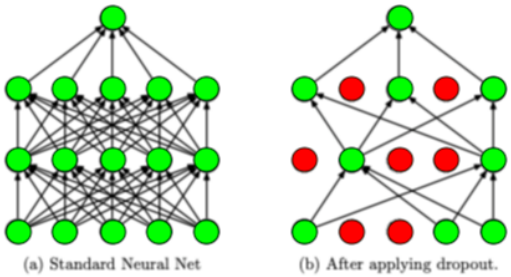

Figure 7: Dropout (*source: JMLR 2014*)

Dropout is similar in some way to the ensemble technique in decision trees, where ensemble models usually perform better than a single model, inasmuch as they are able to capture randomness. Similarly, dropout also performs better than a normal neural network model.

3.3.3 Early Stopping

Early Stopping is a kind of cross-validation strategy where part of the training set is kept aside as the validation set [4]. Early Stopping is when model training is stopped as the model performance on the validation set gets worse.

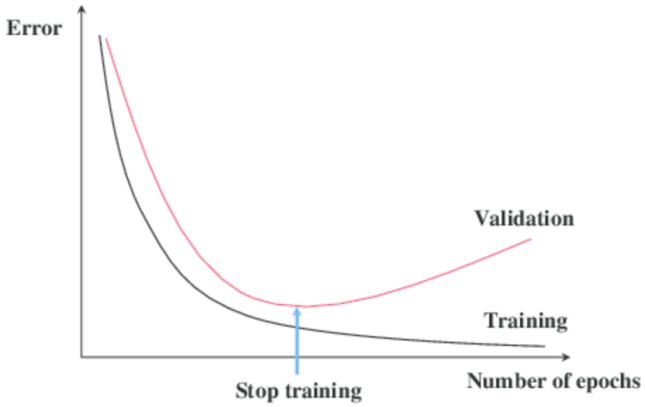

Figure 8: Early stopping (*source: Towards Data Science*)

As illustrated by the figure above, model training stops as soon as the model error on the validation dataset increases. **In Keras, Early Stopping happens in what is called in a callback function. One of the parameter of a callback function is 'Patience', which denotes the number of epochs with no further improvement after which the training stops**. After the blue arrow, each epoch results in a higher value of validation error. **The model stops learning as soon as no improvement in the validation error is seen after a number of epochs equal to 'Patience'**.

3.3.4 Data augmentation

Finally, the simplest regularization technique, is, when possible, data augmentation. Data augmentation consists to reduce overfitting by increasing the size of the training data.

3.4 Autocorrelation

Autocorrelation is the correlation of a process to lagged values of itself, e.g. the process has some kind of predictable pattern based on past values. The autocorrelation function measures the correlation between lags of different lengths. If the autocorrelation does not return any significant correlation between lags, then the assumption that the process is made of independent and identically distributed (i.i.d) variables become reasonable. All machine learning as well as statistical models are based on the assumption that the target variable is independent and identically distributed. **This i.i.d assumption in machine learning and statistics is critical, inasmuch as if not true, it might jeopardize the model conclusions**: if the target variable is correlated with its lagged values, then the target variable is no longer i.i.d, and therefore the model's assumptions are wrong.

More particularly in machine learning, the trained model remains unbiased, e.g. its predictive accuracy is not directly affected by the effects of autocorrelation. However, what does have an indirect impact on predictive accuracy is the fact that the degree of autocorrelation constitutes a proportion of the overall variation of the response. In conclusion, having a high degree of autocorrelation implies that some 'hidden' features are integrated in the target variable, which will make it difficult to predict using the exogeneous input alone.

3.5 Stationarity

Stationarity implies that taking consecutive samples of data with the same size should have identical covariances regardless of the starting point. In practical terms, a time series is considered stationary if it satisifies three conditions, which are constant mean, constant variance and constant covariance between periods of identical distance. The easiest way to test for stationarity is with the Augmented Dicky Fuller test (ADF test). If the test statistic is lower than the critical value, the null hypothesis H0 (time series is not stationary) can be rejected.

It is always easier to work with stationary data, as non stationary time series data in financial models may result in unreliable and spurious results leading to poor understanding and forecasting. In particular, algorithms build relationships between inputs and outputs by estimating the core parameters of the underlying distributions. When these parameters are all time-dependent, algorithms will face different values at each point in time. For the majority of algorithms, time series must be stationary, in order for the analysis and predictions to be performed.

4 Feature importance

4.1 Feature scaling

Scaling is the most common and essential pre-processing task in machine learning. In a nutshell, scaling consists to scale all features prior their use in any model so that they become comparable. Feature scaling makes it easy for a model to learn and understand the problem, as features are now expressed with the same scale. In other terms, scaling allows features to equally account in the model - for instance, GDP can be compared with inflation only when scaled. Among others, two scaling methods are popular.

4.1.1 Standard scaler

The Standard scaler transforms each feature to have zero mean and unit variance. In other words, standardization is z-scoring, the features, as features are rescaled to be approximately standard normally distributed.

$$X_{Standard} = \frac{X - \mu_X}{\sigma_X}$$

4.1.2 Min-Max scaler

Min-Max scaler uses the minimum and maximum values of a feature to rescale values to within a range.

$$X_{MinMax} = \frac{X - X_{min}}{X_{max} - X_{min}}$$

Normalization is often preferred for neural networks, while standardization is the default scaling methods for other approaches [6].

4.2 Feature selection

Machine learning models follow a simple rule: whatever goes in goes out - the famous garbage in garbage out syndrome. In machine learning, considerable quantities of data are fed to help the machine learn better. However, it is unlikely that all the variables in the dataset are useful in terms of explanatory power. More importantly, **adding redundant variables reduces the generalization capability of the model and may also reduce the overall accuracy of a classifier**. Furthermore, adding more and more variables to a model increases the overall complexity of the model. As a consequence, the selection of features is a critical step prior jumping on the model design. An arbitrage between the number of features and their relative added value in the prediction model is also required.

Feature selection is the process of reducing the number of input variables when developing a predictive model.

It is desirable to reduce the number of input variables to both reduce the computational cost of modeling and, in some cases, to improve the performance of the model. In short, **the feature selection's objective is to find the best set of significant features for the model**.

The techniques for feature selection in machine learning can be broadly classified in two categories:

- **Supervised techniques**: for labeled data, to identify the most relevant features in the data.
- **Unsupervised techniques**: for unlabeled data, to identify similar patterns across the different features.

Supervised and unsupervised feature selection techniques are classified under three methods:

- **Filter methods**: pick up the intrinsic properties of the features, measured via univariate statistics instead of cross-validation performance. When dealing with high-dimensional data, it is computationally cheaper to use filter methods. Some examples: Information gain, Chi-square test, Fischer score, correlation, etc.
- **Wrapper methods**: require some method to search the space of all possible subsets of features, assessing their quality by learning and evaluating a classifier with each feature subset. The feature selection process is based on a specific machine learning algorithm which is best fitted on a given dataset. It follows a greedy search approach by evaluating all the possible combinations of features against the evaluation criteria. The wrapper methods usually result in better predictive accuracy than filter methods. Some examples: Forward Feature Selection, Backward or Recursive Feature Elimination, etc.
- **Embedded methods**: encompass the benefits of both the wrapper and filter methods, by including interactions of features but also maintaining reasonable computational cost. In particular, embedded methods extract features which contribute the most to the training at each iteration. Some examples: Lasso (L1), Ridge (L2) as illustrated in section 3.3.1 above, Random Forest Classifier, etc.

Three examples of embedded features selection methods are presented below. These three methods support the feature selection process in the main paper.

4.2.1 Decision Tree Classifier (DTC)

CART stands for Classification and Regression Trees [7]. The CART algorithm creates a binary tree, which means that each node has exactly two outgoing edges, in order to find the best numerical or categorical feature to split using an appropriate impurity criterion.

- For classification, Gini impurity or entropy criterion can be used.
- For regression, CART introduces variance reduction using least squares (mean square error).

The main advantage of CART, among which belongs the Decision Tree Classifier, is that the algorithm will itself identify the most significant variables and eliminate the non-significant ones. More particularly, feature importance is calculated as the decrease in node impurity weighted by the probability of reaching that node. The node probability can be calculated by the number of samples that reach the node, divided by the total number of samples. The higher the value, the more important the feature.

4.2.2 Self-Organizing Maps (SOM)

A self-organizing map (SOM) is an unsupervised machine learning technique [8] used to produce a low dimensional (typically two-dimensional) representation of a higher dimensional data set while preserving the topological structure of the data. SOM is a topology-preserving feature map that represents the distribution of features in the feature space. In SOM similar feature vectors are mapped to adjacent map elements. During the training phase, the map is fed with a large number of unclassified feature vectors. More exctaly, a SOM is a type of artificial neural network but is trained using competitive learning rather than the error-correction learning (e.g., backpropagation with gradient descent) used by other artificial neural networks, and is used for feature selection.

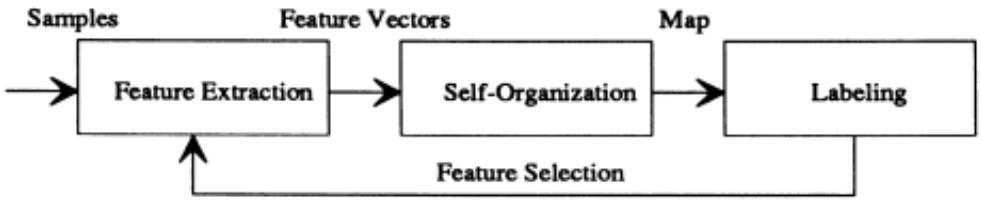

Figure 9: Self-Organazing Maps (*source: Helsinki University of Technology*)

The SOM is a practical method to find a set of features. It is straightforward and easy to implement, and the need of preclassified samples is minimized. SOM is used in a feature selection as follows:

- Take n features.
- Train the map with a large set of these features. Label the map with a set of pre-classified samples. If the classes in the labeled map overlap, add a new feature or replace a feature with a new one. If the feature does not improve the separability of the classes, the feature is replaced.
- Train the map again with a set of new features and label the map.
- Repeat this procedure until you have a set of features that can reach the desired classification rate.

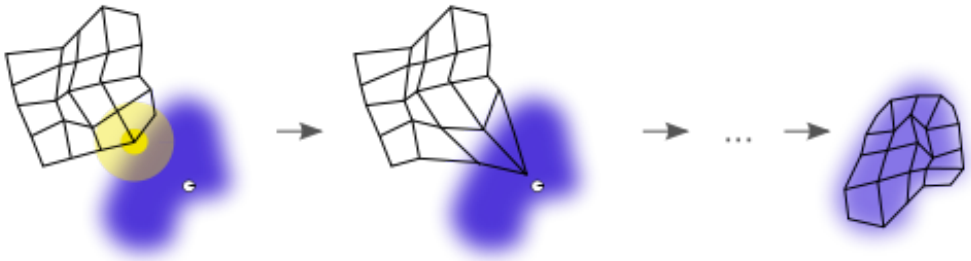

Figure 10: SOM training (*source: Wikipedia*)

4.2.3 Boruta

Boruta algorithm [9] is a wrapper built around the random forest classification algorithm, relatively quick and able to run without tuning of parameters, which returns a numerical estimate of the feature importance. The importance measure of an attribute is obtained as the loss of accuracy in classification caused by the random permutation of attribute values between objects. Due to the fact that the number of random forest runs during Boruta is limited by the maximum of iterations, the calculation can be forced to stop prematurely, when there are still attributes which are judged neither to be confirmed nor rejected, and thus finally marked as tentative. As last point, Boruta is an all relevant feature selection method, thus the selected features are all relevant to the outcome and may be correlated. Therefore, Boruta must be complemented with a different feature importance method or correlation filter in order to remove the correlated features from the feature set.

5 Artificial Neural Networks

An artificial neural network (ANN) is a computational model inspired by biological neural networks such as the human brain. The purpose of the network is to learn to perform tasks by training on given examples. An example of such task is signal classification. A neural network would in this case be given correct labels for each signal in a first place - for example, attribute 1 to daily returns above 1% ('Up' move), 0 otherwise ('Down' move). Second, it would be trained with part of the data in order to predict labels for given signals. Another example of such task is time series forecasting: a neural network would in this case be trained for predict for example the stock price in a week's time. Inputs to a neural network usually are past observations of time series of various factors. On the other side, outputs are either the predicted signal (e.g. classification) or the forecast values (e.g. regression).

The figure below illustrates an Artificial Neural Network. In this example, the ANN has three input nodes, one hidden layers with three nodes, and two output nodes.

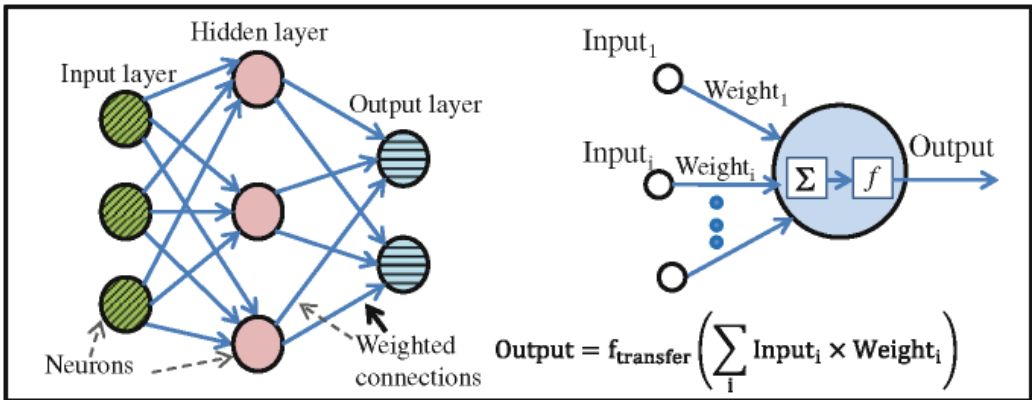

Figure 11: Representation of an Artificial Neural Network (*source: Ebrary*)

Some key concepts and notations must be introduced at this stage.

5.1 Perceptrons

A perceptron in a neural network is a neuron with an activation function [10]. An activation function transforms a node input into an activation/output. The figure below helps to grasp the concept: comparing with what a neuron is in a human brain, a node is a replica of a neuron that receives a set of input signals.

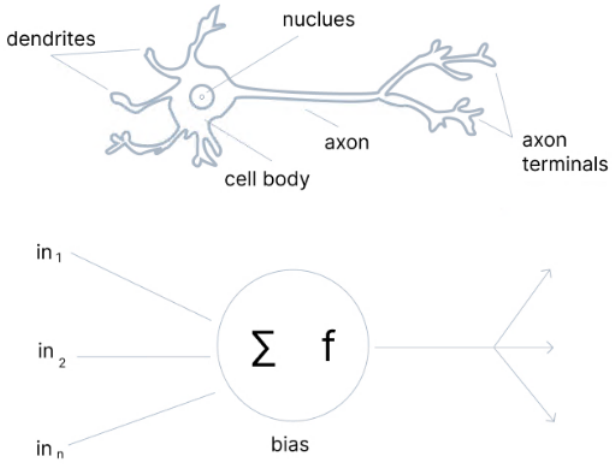

Figure 12: Representation of a physical neuron and a node *(source: V7 labs)*

5.2 Activation functions

Activation function decides whether a neuron should be activated or not by calculating the weighted sum of the different inputs and further adding bias with it. The purpose of the activation function is to introduce non-linearity into the output of a neuron, as a neural network without a non linear activation function simply is a linear regression model. The activation function does the non-linear transformation to the input making it capable to learn and perform more complex tasks.

The role of the activation function is to derive an output from the sum of weighted inputs and bias fed to a node or a layer. What function to use depends on the data that the model will fit, and what the output of the model should be. It is important to acknowledge that there is therefore no right or wrong activation function and testing is often needed to determine the most optimal function for a given problem. Some examples of the most classic activation functions for neural networks:

- Sigmoïd: $s(x) = \frac{1}{1+e^{-x}}$
- Hyperbolic tangent: $tanh(x) = \frac{e^{x}-e^{-x}}{e^{x}+e^{-x}}$
- ReLU: $g(x) = max(0, x)$

- Leaky ReLU: $f(x) = max(0.1x, x)$
- eLU: $e(x) = \begin{cases} x & \text{if } x \geq 0 \\ \alpha\left(e^x - 1\right) & \text{if } x < 0 \end{cases}$

As a rule of thumb, sigmoïd and tanh functions should not be used in hidden layers as they make the model more susceptible to issues during training because of vanishing gradients. In addition, the sigmoïd activation function is preferred for the output layer in case of binary classification.

5.3 Layers

A layer in a neural network is a collection of nodes that hold numbers [10]. The first layer in a network is an input layer, where each node is an input x_1, x_2, ...,x_n. After the first layer are several layers often referred to as hidden layers $h^{(1)}$, $h^{(2)}$, ..., $h^{(n)}$. Each of these layers are composed of nodes $h_1^{(i)}$, $h_2^{(i)}$, ..., $h_n^{(i)}$, which are connected to all nodes in the previous layer and connects to all nodes in the next layer. Each of these connections has a weight, in such way that all connections between two layers can be described as a weight matrix $W^{(i,j)}$ - where i is the index of the previous layer and j is the index of the next layer. Since each node has several inputs, they need to be multiplied by the connection weights and then summed to a single number before being output to the next layer. After one or more hidden layers is the last layer, called the output layer, where the nodes y_1, y_2, ..., y_n, represent the outputs of the neural network. For example, a single node can represent the output from a regression, and multiple nodes can represent probabilities for different classes in a classification setting.

5.3.1 Input layer

An input layer provides information from the outside world to the neural network, no computation is performed on this layer, nodes here just pass on the information from the different features to the hidden layer.

5.3.2 Hidden layer

A hidden layer performs all sorts of computation on the features entered through the input layer and transfers the result to the next (hidden or output) layer. It is important to note that nodes of a hidden layer are not exposed to the outer world. They are the part of the abstraction provided by any neural network.

5.3.3 Output layer

The output layer is the final layer of the neural network that brings the information learned through the hidden layers, and delivers the final value as a result.

5.4 Backpropagation

Backpropagation is the method used to train neural networks, and is the key to the learning process. To perform backpropagation, a loss function for the network needs to be defined first [11]. A loss function is any measure that defines the error between model predictions and actual outcomes. Since any layer in a neural network can be represented as function of the previous layers, the loss function $L(x)$ can be defined as a function of the whole neural network $L(x) = g(f(x))$. $f(x)$ can also be expressed as $f(x) = f_n(f_{n-1}(...)) = f_n \circ f_{n-1} \circ ... \circ f_1(x)$, with $f_i(x) = \sigma\left(W^{i,i-1}\right)x + b_i$, where $i = 1, ..., n$ represent layer indices, σ is an activation function, $W^{i,j}$ is the weight matrix between layer i and layer j, and b_i is a bias term for layer i.

Backpropagation works by taking the gradient of the model loss function with regards to the weight matrices, starting from the last layer all the way back to the first layer. All weights, including the bias term, are then updated using their respective gradients. In practice, this comes down to calculating gradients using the chain rule. However, a neural network using the sigmoïd function for example, may experience issues during training since backpropagation, which can cause vanishing or exploding gradients [12, 13]. Recent research has therefore proposed different activation functions such as $ReLU$ or eLU, in an attempt to solve this problem.

In short, **backpropagation aims to minimize the loss function by adjusting the weights and biases across the neural network. The gradients of the loss function determine the level of adjustment with respect to parameters like activation function, weights, bias, etc. This is how the model learns**.

6 Recurrent Neural Networks

A Recurrent Neural Network (RNN) is a special type of Artificial Neural Network where there is a temporal sequence of networks that connects through a set of weights [13]. This characteristic allows RNNs to handle temporal data well. In short, **RNNs have a memory, which happens to be particularly useful for sequential modelling such as signal prediction, time series forecasting or when it is required to predict the next word of a sentence where previous words matter. In RNNs, the outputs from previous steps are fed as inputs to the current step, unlike traditional neural network where inputs and outputs are independent of each other**. Therefore, the main and most important feature of a RNN is the hidden state, which remembers information about the previous steps of the sequence. RNNs are said to be a class of feedforward neural networks, inasmuch as the flow of information occurs in the forward direction. In a feedforward propagation, the activation function can be seen as a gate between the input feeding the neuron and its output going to the next layer.

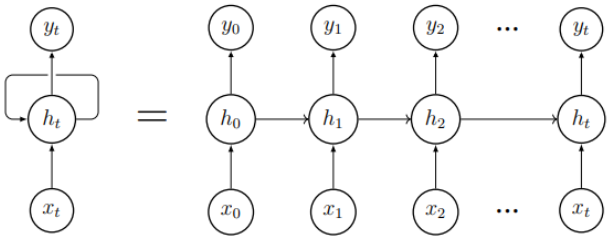

Figure 13: Recurrent Neural Network (*source: Medium*)

A RNN can be described by an input vector x_t and the following vectors:

- hidden layer (can be multiple): $h_t = \sigma_h \left(W_h x_t + U_h h_{t-1} + b_h\right)$
- output: $y_t = \sigma_h \left(W_h h_t + b_y\right)$

where W_h is the weight matrix between x_t and h_t, U_h is the weight matrix between hidden layer vectors, and W_y is the weight matrix between h_t and y_t. b_h and b_y are bias vectors for the hidden layers and output. Finally, σ_h is the activation function for the hidden layers.

Among the various RNNs solving this issue, two particular RNNs are developed in this note: the Long-Short Term Memory and Gated Recurrent Unit models.

6.1 Long-Short Term Memory

A Long-Short Term Memory (LSTM) model is a modified RNN that partially solves the vanishing gradient issue by introducing feedback loops in the form of memory cells in the network [13].

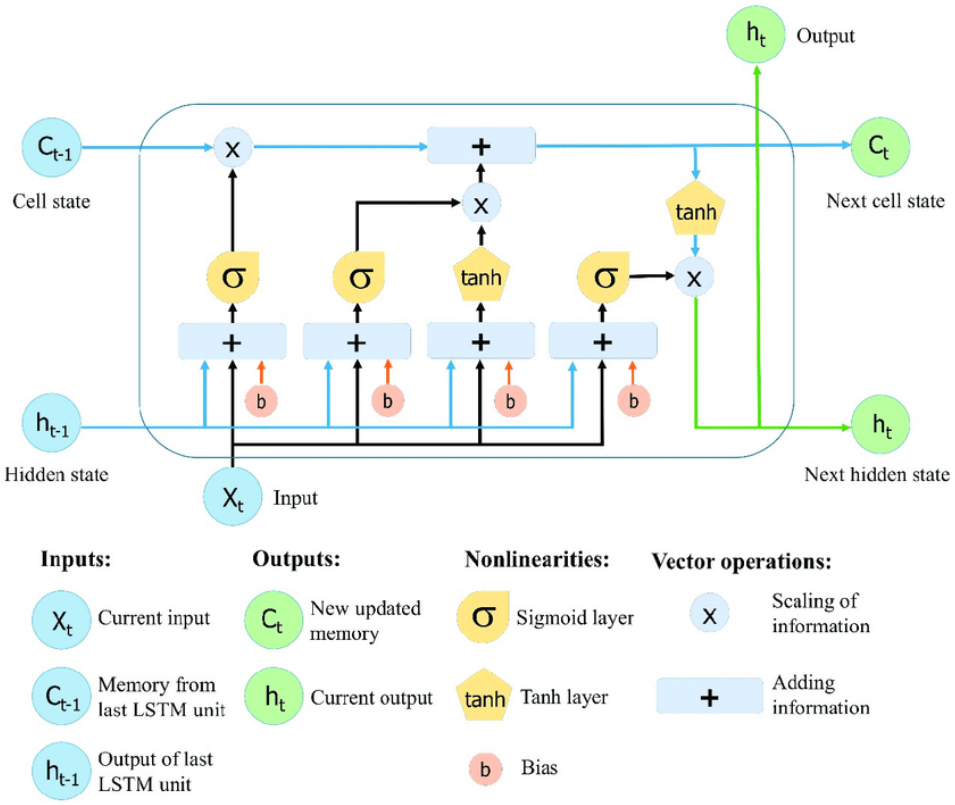

Figure 14: LSTM Neural Network (*source: Yan, Giha Lee*)

A LSTM cell, which takes h_{t-1} and x_t as inputs, and h_t as outputs, is defined by:

- input gate: $i_t = \sigma_g \left(W_i x_t + U_i h_{t-1} + b_i\right)$
- forget gate: $f_t = \sigma_g \left(W_f x_t + U_f h_{t-1} + b_f\right)$
- output gate: $o_t = \sigma_g \left(W_o x_t + U_o h_{t-1} + b_o\right)$
- cell state: $c_t = f_t \circ c_{t-1} + i_t \circ c_t$
- output: $h_t = o_t \circ \sigma_h \left(c_t\right)$

where W and U are weight matrices, and b is the biased vector. σ_g and σ_h are the activation functions.

6.2 Gated Recurrent Units (GRU)

The key difference between GRU and LSTM neural networks sits in the fact that GRU only has two gates that are reset and update gates, unlike LSTM which has three gates that are input, output, and forget [13]. This property makes GRU less complex.

A GRU cell, which takes h_{t-1} and x_t as inputs, and h_t as outputs, is defined by:

- update gate: $z_t = \sigma_g \left(W_i x_t + U_i h_{t-1} + b_i\right)$
- reset gate: $r_t = \sigma_g \left(W_f x_t + U_f h_{t-1} + b_f\right)$
- candidate output: $\tilde{h_t} = \sigma_h \left(W_h x_t + U_h \left(r_t \circ h_{t-1} + b_h\right)\right.$
- output: $h_t = z_t \circ \tilde{h_t} + (1 - z_t) \circ h_{t-1})$

where W and U are weight matrices, and b is the biased vector. σ_g and σ_h are the activation functions.

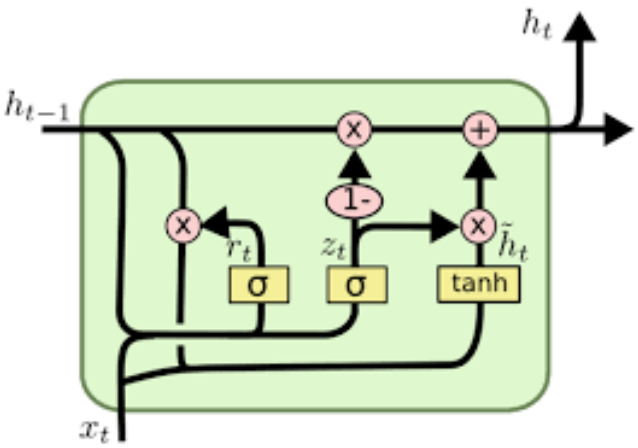

Figure 15: GRU Neural Network (*source: Wikipedia)*

To conclude, the figure below illustratres the differences in construction for RNNs, LSTMs and GRUs.

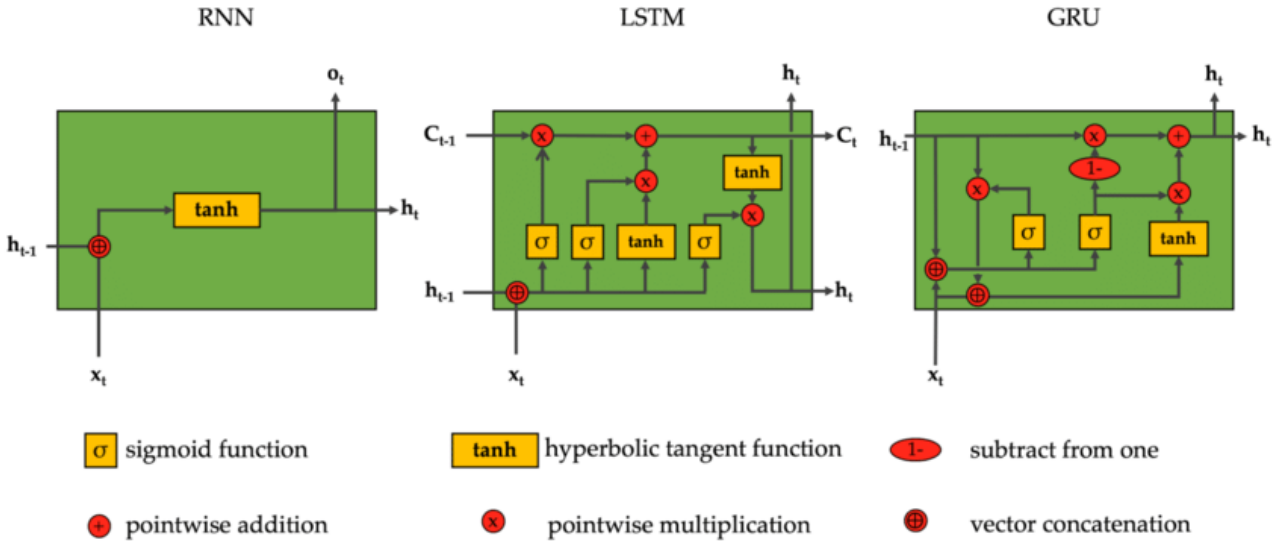

Figure 16: RNN vs LSTM vs GRU (*source: University of Iowa)*

7 Evaluation of model performance

In machine learning, performance measurement is essential, and some expressions must be introduced at this stage when dealing with classification problems.

- True Positive (TP): A test result that correctly indicates the presence of a condition or characteristic.
- True Negative (TN): A test result that correctly indicates the absence of a condition or characteristic.
- False Positive (FP): A test result which wrongly indicates that a particular condition or attribute is present.
- False Negative (FN): A test result which wrongly indicates that a particular condition or attribute is absent.

In a classification problem, evaluation metrics differ from regression situations. Instead of using metrics like Mean-Square Error (MSE), Root-Mean-Square-Error (RMSE), the performance of machine learning models used in classification situations use metrics like Accuracy, Precision, Recall, F1 score or ROC curve. All these classification metrics result in a number that measures the performance of the classification model when it comes to assigning observations to classes. Among the most important ones:

7.1 Accuracy

Accuracy measures how many observations, both positive and negative, were correctly classified. The main advantage of accuracy sits in its simplicity to explain and grasp the concept.

$$Accuracy = \frac{TP + TN}{TP + FP + TN + FN}$$

Accuracy should not be used on **imbalanced datasets**, as it becomes easy to obtain a high accuracy by simply classifying all observations as the majority class.

7.2 Precision / Recall

Precision (also called **Positive Prediction Value**) measures how many observations predicted as positive are in fact positive.

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

On the other hand, Recall (also called **True Positive Rate** or **Sensitivity**) measures how many observations out of all positive observations are classified as positive.

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

Precision and Recall must be looked together, as looking one without the other may lead to biased conclusions.

7.3 F1 score

The F1 score is the harmonic mean between Precision and Recall. In other terms, it is a score where Precision and Recall are equally important.

$$F1score = 2 \times \frac{Precision \times Recall}{Precision + Recall}$$

The F1 score has value from 0 to 1. The closer the F1 score to 1, the better the model. As for Accuracy, F1 score should not be used on **imbalanced datasets**, as it becomes easy to obtain a high F1 score by simply classifying all observations as the majority class.

7.4 AUC-ROC curve

The ROC curve is a performance measurement for the classification problems at various thresholds settings. ROC is a probability curve, and the area under the curve, the AUC score, represents the degree of separability. In short, **The AUC-ROC curve indicates the model capability to distinguish between classes**. For instance, in the case of binary classification, the higher the AUC, the better the model predicts 0 classes as 0, and 1 classes as 1.

True Positive Rate (TPR)

$$TPR = Recall = Sensitivity = \frac{TP}{TP + FN}$$

False Positive Rate (FPR)

$$Specificity = \frac{TN}{TN + FP}$$

$$FPR = 1 - Specificity = \frac{FP}{TN + FP}$$

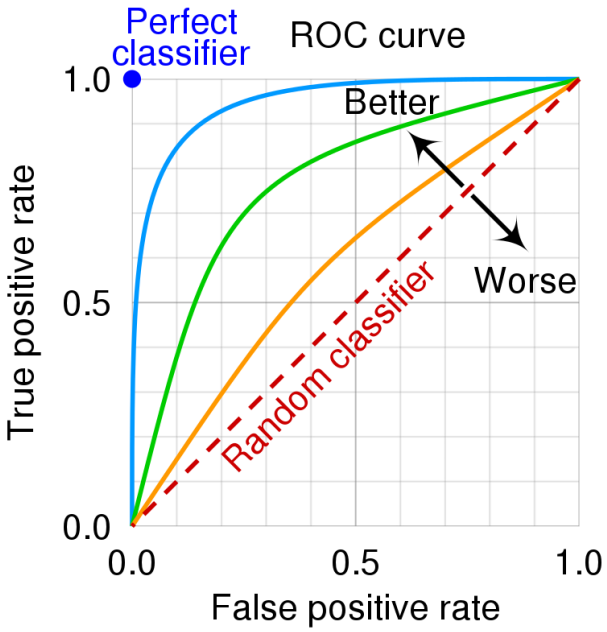

Figure 17: AUC-ROC curve (*source: Wikipedia*)

The area under the ROC curve is a measure of model accuracy. When a model is closer to the diagonal, it is less accurate. On the contrary, a model with perfect accuracy has an area of 1.

7.5 Confusion matrix

Last but not least. Despite not being an evaluation metric, the confusion matrix remains essential to the evaluation of a classification model, as it allows the visualisation of the model performance.

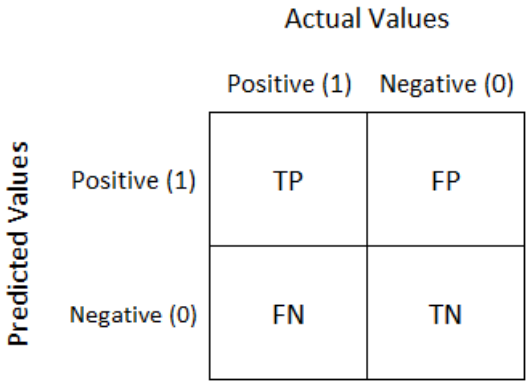

Figure 18: Confusion matrix (*source: Medium*)

7.6 Address class imbalance

An imbalance dataset refers to a dataset where the number of observations across the known classes differ. The class with the lowest number of observations is called the minotority class and the class with the highest number of observations is called the majority class. A dataset is severely imbalanced when the number of observations in the majority class largely exceeds the number of observations in the majority class. Imbalanced classifications pose a challenge for predictive modeling as most of machine learning algorithms used for classification were designed around the assumption of an equal number of examples for each class. This results in models that have poor predictive performance, specifically for the minority class. This is a problem because typically, the minority class is more important and therefore the problem is more sensitive to classification errors for the minority class than the majority class.

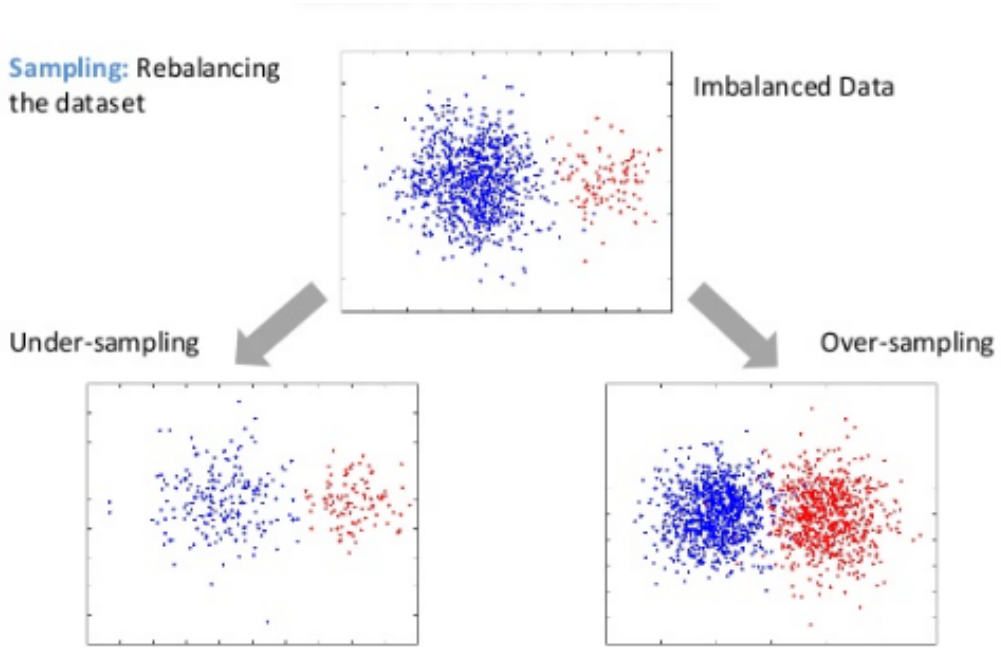

Figure 19: Address class imbalance (*source: Towards Data Science*)

Addressing class imbalance can be performed in two ways.

- Oversampling consists in creating additional points in the minority class in order to obtain a balanced dataset aligned on the majority class. A balanced dataset with the oversampling method is achieved with the SMOTE algorithm.
- Undersampling consists in removing data points from the majority class in order to obtain a balanced dataset aligned on the minority class. A balanced dataset with the oversampling method is achieved with the Near-Miss algorithm.

8 Tuning of hyperparameters

Deep learning models have many different parameters that can be optimized to better fit the available data. Hyperparameters are parameters whose values are set before the learning process begins - e.g. parameters that are not directly learnt within estimators. By contrast, the value of simple parameters are derived through training. Hyperparameter tuning is the process of choosing the optimal set of parameters in order to get the best prediction model, and is consequently essential in model design. In the case of a neural network, hyperparameters can be, among others, the number of neurons in layers, the number of hidden layers, the learning rate, the dropout rate or the activation function.

Behind the tuning of hyperparameters for our model underlies an optimization problem, as the objective is to minimize the loss function by changing model parameters. It is key to find the best hyperparameters of a model inasmuch as they drive the model performance and speed as well as the quality of the predictions. Wikipedia sums it best: "Hyperparameter optimization finds a tuple of hyperparameters that yields an optimal model which minimizes a predefined loss function on given test data. The objective function takes a tuple of hyperparameters and returns the associated loss". Three main techniques exist to find hyperparameters.

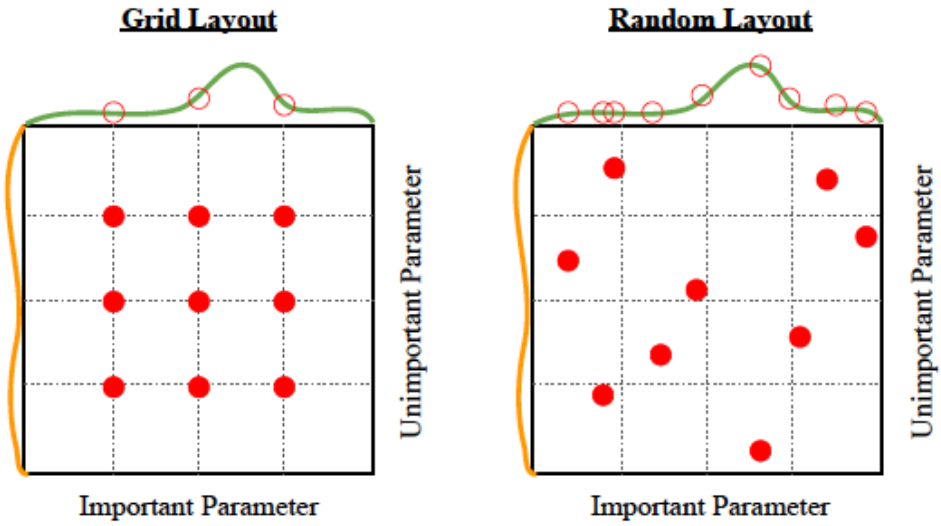

Figure 20: GridSearch and RandomSearch (*source: Medium*)

8.1 GridSearch

The conventional way of performing hyperparameter optimization is through a grid search [14], which happens to be a parameter sweep. It is an exhaustive search through a manually specified subset of the hyperparameter space of a learning algorithm. A grid search algorithm must be guided by some performance metric, typically measured by cross-validation on the training set or evaluation on a validation set. GridSearch performs exhaustive search over specified parameter values for an estimator. It implements a "fit and score" method among other methods. The parameters of the estimator used to apply these methods are optimized by cross-validated grid-search over a parameter grid.

8.2 RandomSearch

The RandomizedSearchCV implements a "fit and score" method and performs randomized search on hyper parameters [15]. The parameters of the estimator used to apply these methods are optimized by cross-validated search over parameter settings. In contrast to GridSearchCV, not all parameter values are tried out, but rather a fixed number of parameter settings is sampled from the specified distributions.

8.3 Bayesian optimization

Hyperparameter tuning is performed through a Bayesian optimization [16]. Bayesian methods attempt to build a probability distribution over some possible function, called the surrogate function, that estimates how good the model might be for a given choice of hyperparameters. Bayesian optimization runs models many times with different sets of hyperparameter values, and evaluates the past model information to select hyperparameter values to build the newer model based on the surrogate function.

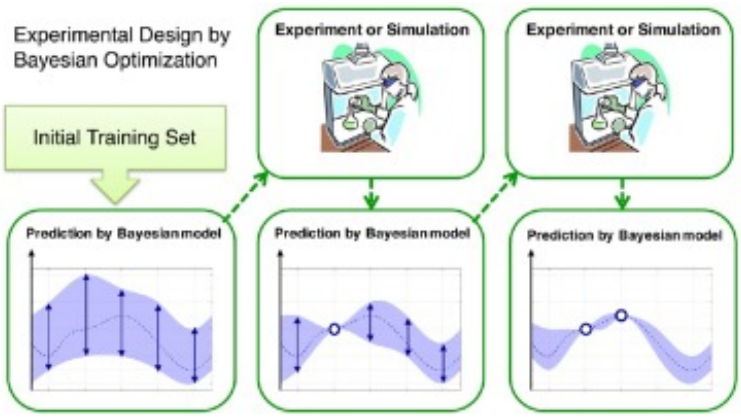

Figure 21: Bayesian optimization (*source: Science Direct*)

9 Conclusion

The objective of this note is to give a complete theoretical framework to all the concepts employed in the main paper **Long-Short Term Memory and Gated Recurrent Units neural networks as a support to investment and risk management strategies**.

The note covers theoretical aspects which are essential to understand how modelling with deep learning models works, and aggregate many sources and documents into one single place to give the reader a complete understanding about the various steps which are necessary to make classification predictions with neural networks. Beyond focusing on feature selection and model design, the note also focuses on the various challenges met in machine learning in general, such as regularization, model evaluation and tuning of hyperparameters as they remain as important as the rest to build a sound prediction model.

References

[1] Zacharie Guibert. "Long-Short Term Memory and Gated Recurrent Units neural networks as support to investment and risk management strategies". *CQF cohort 39 - final exam* (2022).

[2] Stephen J Taylor. "Asset Price Dynamics, Volatility, and Prediction". Chap. 9.3. ISBN: 9780691134796.

[3] Kannan Singaravelu. "Practical machine learning". *CQF cohort 39 - Module 4 Lecture 4* (2022).

[4] Xue Ying. "An Overview of Overfitting and its Solutions". *Journal of physics* (2018).

[5] Srivastava and Hinton. "Dropout: A Simple Way to Prevent Neural Networks from Overfitting". *Journal of Machine Learning Research 15* (2014).

[6] Kannan Singaravelu. "Introduction to ML using Scikit-learn". *CQF cohort 39 - lab 8* (2022).

[7] R.T. Clarke H.R. Bittencourt. "Feature selection by using classification and regression trees (CART)". *Faculty of Mathematics, Pontifícia Universidade Católica do RS, Porto Alegre, Brazil* (2016).

[8] Jukka Iivarinen and Kimmo Valkealahti. "Feature Selection with Self-Organizing Feature Map". *Helsinki University of Technology* (1994).

[9] Witold R. Rudnicki Miron B. Kursa Aleksander Jankowski. "Boruta – A System for Feature Selection". *ICM, University of Warsaw* (2010).

[10] v7 labs. *Activation Functions in Neural Networks, 12 Types Use Cases.* URL: `https://www.v7labs.com/blog/neural-networks-activation-functions`.

[11] Kannan Singaravelu. "Introduction to deep learning neural networks". *CQF cohort 39 - Module 5 Lecture 3* (2022).

[12] Sepp Hochreiter. "The vanishing gradient problem during learning Recurrent Neural Nets and problem solutions" (1997).

[13] Gregory D. Hager Robert DiPietro. "Deep learning: RNNs and LSTM". *Handbook of Medical Image Computing and Computer Assisted Intervention* (2020).

[14] Kannan Singaravelu. "K-Nearest Neighbors". *CQF cohort 39 - lab 9* (2022).

[15] Kannan Singaravelu. "eXtreme Gradient Boosting". *CQF cohort 39 - lab 10* (2022).

[16] Wu Chen al. "Hyperparameter Optimization for Machine Learning Models Based on Bayesian Optimization". *Journal of electronic science and technology* (2020).