ŠKODA AUTO VYSOKÁ ŠKOLA o.p.s.

Course: N0413A050001 Business Administration

Field of study/specialisation: Specialization International Supply Chain Management

Application of neural network edges for a possible embedding into supply chain management

Diploma Thesis

Temur Dzhuraev

Thesis Supervisor: Ing. Tomáš Malčic, Ph.D.

Remove and replace this list by final confirmed bachelor´s assignment with the electronic signatures. Please note that the bachelor´s assignment must be printed on both sides in the original printed version.

I declare that I have prepared this thesis on my own and listed all the sources used in the bibliography. I declare that, while preparing the thesis, I followed the internal regulation of ŠKODA AUTO VYSOKÁ ŠKOLA o.p.s. (hereinafter referred to as ŠAVŠ), directive Thesis guidelines.

I am aware that this thesis is covered by Act No. 121/2000 Coll., the Copyright Act, that it is schoolwork within the meaning of Section 60 and that under Section 35 (3) ŠAVŠ is entitled to use my thesis for educational purposes or internal requirements. I agree with my thesis being published in accordance with Section 47b of Act No. 111/1998 Coll., on Higher Education Institutions.

I understand that ŠAVŠ has the right to enter into a licence agreement for this work under standard conditions. If I use this thesis or grant a licence for its use, I agree to inform ŠAVŠ about it. In this case, ŠAVŠ is entitled to demand a contribution to cover the costs incurred in the creation of this work up to their actual amount.

Mladá Boleslav, Date ..... *Signature*

Contents

List of abbreviations and symbols

AI	Artificial Intelligence
AdaGrad	Adaptive Gradient
BPTT	Backpropagation Process Trough Time
BN	Batch Normalization
BRNN	Bidirectional Recurrent Neural Networks
CNN	Convolutional Neural Networks
GN	Group Normalization
GRU	Gated Recurrent Units
LN	Layer Normalization
LSTM	Long Short-Term Memory
MAE	Mean Absolute Error
ML	Machine Learning
MSE	Mean Squared Error
MB – SGD	Mini Batch Stochastic Gradient Descent
NAG	Nesterov Accelerated Gradient
NN	Neural Networks
RNN	Recurrent Neural Network
RMSE	Root Mean Squared Error
SGD	Stochastic Gradient Descent
WS	Weight Standardization

Introduction

The industry 4.0 is consistently being implemented through all levels of supply chain and AI is part and parcel of this inevitably coming tendency. The need for automation in the automotive production industry has bounced up in the last two decades than ever before. The demand for new technologies and new solutions in the supply chain are the most sought out tools in the production. Neural networks (NN) have been elaborated in order to optimize procedures alleviating human's job decreasing cost generating higher revenue. However, NN embedding is accompanied by higher risks since any teeny imprecision in predictions can lead to unwanted consequences such as cost, overproduction, a break in assembly line and etc. The main objective of the thesis is to practically implement one type of neural network which has the ability to memorise data from the past considering it for future prediction. This tool can become useful for supply chain if we want to gauge demand for production of a good for next week, month or year under unstable market.

Nowadays all industries are facing significant challenges, which are increasingly demanding customers, immense competition, and permanently mounting demand for decreasing total cost. One way to deal with the new circumstances is definitely the automation of logistic processes by means of taking attempt to embed neural networks in logistics. Basically, it is essential in supply chain that everything happens at the right time, in right place and in the right amount. These requirements can be easily converted into task for a NN to fulfil the assigned requirements. For instance, predict timely delivered of some items to the right place and in the right amount to the plant. Another example, predict unloading or loading good in a plant by DHL for a next month. The derived data can be a good source for choosing better suppliers optimizing supply chain. Thus, it is substantial to understand the significance of the development of modern technologies and concentrate more on them.

The diploma thesis is structured into three parts. The first chapter is dedicated to the theoretical aspect of the thesis introducing AI's implementation into supply chain tackling various tasks beginning from computer vision to inventing smart forklift. Moreover, a subchapter explains specification of recurrent type of neural

network which is predecessor of long short-term memory network which is utilized later. Second chapter is crucial one since it explains the nuances of neural network learning procedure from mathematical viewpoint delving into optimization algorithms and their differences which have significant influence on the final output. One subchapter unleashes mystery from distinction between normalization and standardization, another one reveals hidden problems with underfitting and overfitting which have their own options to treat them and evaluation indicators enable to assess model's performance to determine the steps which are necessary to do either to implement it into process or keep improving it up to desired requirements. The third chapter is dedicated to hands-on elaboration of neural network using python with corresponding libraries to invent, learn and assess LSTM type of neural network with stock market price data utilization as a primary time-series structured one. The result of it should give the answer for the task of possible embedding of NN into supply chain. As a core method to solve the thesis topic is inventing python code with main documentation sources of Keras, Pandas, Numpy, Matplotlib and Tensorflow.

1 Artificial intelligence and digital transformation in Supply Chain Management: theory and instances.

Digitalization is the fast-moving phenomena trends which has enormous impact on continuous the industrial world shaping opening contemporary opportunities around a wide range of areas. Nowadays artificial intelligence is conceived as a key-point driving force of industrial digital transformation that has the capacity to present a new comprehensive source of development. Relatively recent breakthrough in both machine learning and neural network have created a whole new business ecosystem which is energetically being rendered by IT companies, car manufacturers such as TESLA Motors and one of the largest e-commerce companies in the world which titles Amazon. Pursuant to a survey by the VDI, approximately 25% of companies are about to consider artificial intelligence (AI/NN) and machine learning (Moroff and Sardesai, 2019).

There are some vagueness and unclarity are over about the role of AI in manufacturing since most of researchers have not defined a clear statement of it. Nonetheless, the unity between them converged on AI value for the industry 4.0 in field of operations where advanced robotics and NN are involved (Tjahjono et al.,2017).

One of the fundamental topics where AI implementation is deemed vital is digital twin. It turned out that researchers made a mountain out of a molehill because the overwhelming majority of literatures on digital twins are related to the area of production management revealing the basic theoretical analysis and only quarter of them describes precise use cases. However, the simulation level or prototypes are the maximum what is given (Zhihan and Shuxuan el al., 2022).

Another example of AI hands-on application is smart forklift. A forklift is fitted with several sensors modules which collect data into digital cloud, processed in real-time and displayed on a monitor. Since there is collected data, it is possible to analyze it and as the result, some ML-engineers decided to embed a K-Nearest Neighborhoods algorithm (KNNs) for classification problem solving collecting time series data from equipped sensors (Jie, et al., 2022).

Another prominent example is computer vision implementation in cars where nowadays Tesla has attained the best results. Auto-pilot system is based on

computer vision accompanied by ML and video streams from the cameras. Afterword, the raw footage is processed by Convolutional Neural Networks (CNNs) for object tracking and detection. In addition to cameras, each Tesla car has radar and ultrasonic sensors where the first one gauges the distance between car and objects and the second one measure proximity with passive objects enhancing drivers' safety. As for the neural network embedded in the car, the title is ResNet-50 that can run 1000×1000 images at time. For instance, one NN solves a task with stop signs, another one recurrently tackles pedestrians crossing a road and yet another copes with traffic lights. The title for all together networks is hydranets. Overall Tesla trained 48 networks fulfilling 1000 predictions consuming 70000 GPU hours. That is what they call a real breakthrough (Almeida, 2021).

However, the digitalization process and its survey began in the 1990s with consistent data records process collecting and networked systems especially in supply chain management. Although computing power has not been ample to store data, along with increasing computing power changing and inventing new algorithms it is today possible to analyze the converged data sets of companies and render them wholesomely. Algorithms have been developed in order to analyze large amounts of information to distinguish patterns that can be employed as decision framework for future planning and processes (Tarpend, Tyler, Krause, Handfield, 2008). One of the most conventional approaches in ML is taking historical data or so-called time series one which evolves consistently. In this regard demand forecasting in the supply chain planning becomes an important employment area for various ML's methods, as multiple various real-lives variables affect the market and traditional statistical methods reaches their ceiling.

Forecasting in Supply Chain Management describes the operation of further resources requirements to quantity and time which must be fulfilled. In SCM unquestionably everything is highly dependent on the demand planning output therefore it is vital for the company both profitability and prolificacy to reach a high forecast rate and to diminish vagueness (Muddassir, 2016). Consequently, machine learning and neural networks come to play. There is a bundle of forecasting models analyzing the past in order to foresee future events. It is deemed that those events follow the same implications and constitute a pattern which is quite stable and so-called as time stability theory. According to this

supposition, there are discrepancies between the predicted output and reality which are evaluated as deviation or error. Simply saying, demand planning is based on prediction results of the historical recorded data outlined in the past. However, in order to attain the desired high rate of forecasting, both ample database and computing effort are part and parcel of hence selection procedure is necessary of the forecasting model.

Because of the rapidly advancing progress of computer technologies, conventional prediction algorithms which come to our minds from statistics or econometrics are being substituted by much sophisticated approaches. Nowadays a myriad number of companies tend towards the machine learning or neural networks to enhance the prediction by means of internal and external data.

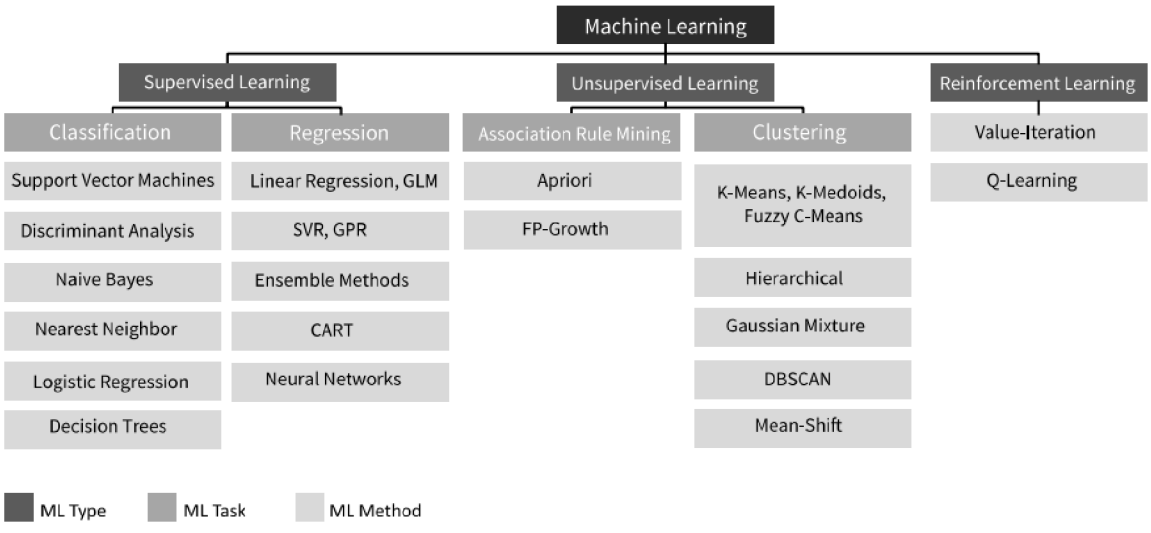
However, on the way of data analysis data scientists stumble across uncovered predicaments such as multidimensionality, preliminary wrong data filling-in or the lack of data rules which should be stipulated by an inner company charter. Once Habla analyzed various number machine learning methods and have deduced that innovative products have a significantly smaller product life cycle than standard ones thereby sales behavior can be forecasted much less precisely (Moroff and Sardesai, 2019). Apart from it, even with well-structured data basis there are some hidden challenges might be like data interpretation or its refining. In 2017 there have been elaborated a data extraction process which embraces several steps: the definition of the targeted quantity of data, which is sufficient for proper analyzing, data cleansing step is crucial one since it is highly important to decide which of the given columns must be chosen and which of them must be taken out from the observation and resultantly it has direct influence on the final forecast (Mathes, Klaben and Pieper, 2017).

Machine learning models divide into three types: supervised, unsupervised and reinforcement learning. The first one is trained by means of using known previously recorded data and the output is already known as well, this learning approach targets at looking-up a relation in the form of the stated rules that connect input data to its result and ultimately apply the learned rules to new data. From this point of view, the computer program is getting trained. Once the model is being trained it can predict future input and output data. Usually, the supervised models are rendered in solving such tasks as classification and regression.

Another model called unsupervised learning describes a system that is able to obtain knowledge without correct answers providing thus there are no pre-labelled target values. That pattern is so-called as “learning without a teacher”. A widely applied task of unsupervised learning is clustering issue whereas the model identifies affinity between the input data classifying them by common patterns.

The last unique approach is known as reinforcement learning where feasible or optimal solutions are unknown to the model at the beginning of learning step and therefore it must be defined iteratively based on rewards and punishments system. Being in the learning phase, sensible approaches are rewarded, and vice versa wrong steps are prone to be punished. As a result, the system looks for its own solutions autonomously through directional rewards and punishments (Hannah, Daniel and Saskia, 2019).

To better understand the primary point of ML the model scheme can be presented in the following-up image where each type of algorithm solves the corresponding task. For instance, supervised machine learning conventionally encompasses such tasks as both classification and regression or, for example, unsupervised algorithms are in clustering and association rule mining and reinforcement learning the rest of them in the Figure 1:



Source: (Witten, Eibe and Hall, 2011)

Figure 1: The Machine Learning algorithm types with its corresponding tasks.

As it has already been pointed out that each algorithm oversees its own particular task, but it is utterly important to comprehend the outright task’s essences to be

not confused by variety of existing ones. The classification procedure is used to categorize objects in compliance with their properties based on the underlying dataset. Input data undergoes assortment where finally gets into the calculated grid according to their attributes. Since the classification procedure is quite discrete one thus the destination is usually unambiguous. Having sorted the data point, the classes are recalculated on the basis of the newly formed set of data to attain continuous enhancement of the classification output (Witten, Eibe and Hall, 2011).

Regression procedure is widely rendered by various number of companies around the world for forecasting demand because it may predict the future trends considering the historical variables. As in the traditional statistics, there are both endogenous and indigenous values which are examined in order to track back their interplay to each other. The range of tools in ML ranges from well-known simple regression to Lasso method or poison regression. For these concrete, fathomable reasons, they are wholesome in prediction case of scenario since they are able to take into account of several variables with subsequent inherent restrictions which in case of their breaking can lead to distorted results (Moroff and Sardesai, 2019).

Clustering analysis is utilized in grouping and classifying data points pursuant to their properties so that it is crucial to have measurable data point to commit comparative analysis. The whole process is divided into two phases. At the beginning, it is necessary to find out similarity between the new data point and the existing set of data basis. This is followed by a division of similar sets of data into several independent groups thus forming clusters. At this stage, difference is accomplished between hard and soft algorithms of clustering and hence the unique data point assortment points to a cluster. If clustering algorithm is soft the data point might belong to several clusters as well and vice versa it is a hard clustering algorithm if a data point is always assigned uniquely to a cluster.

These above-mentioned tasks are solving the machine learning algorithms in hands-on application in the real-life world (Moroff and Sardesai, 2019).

However, machine learning algorithms have significant outright shortcomings which might cost huge amount of money in case of making mistakes or false

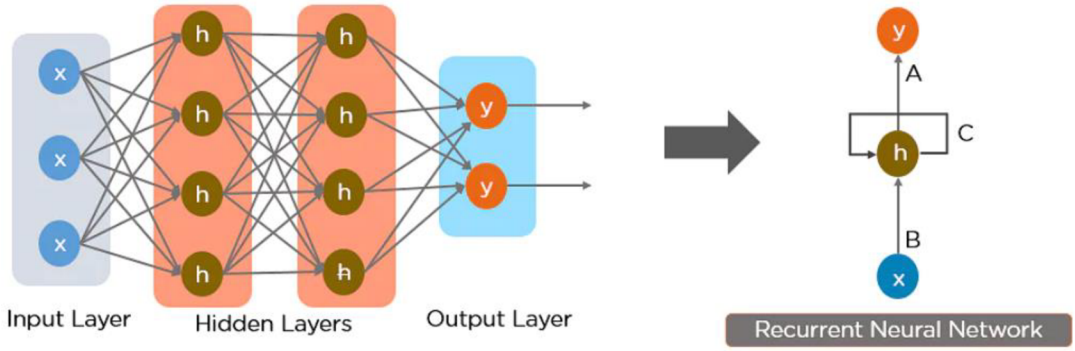
positive predictions compared to neural networks which are much more resistible in case of outliers or emissions in a set of data. To better understand the core difference between two at the first glance similar items is human-hand interference in the process of learning.

The input data must be processed in ML algorithms, features are extracted, and classification is done. But deep learning performs both steps together. Moreover, a major privilege of NN is time saving contrary to the high training period required for ML models. Apart from it, NN secures input data flexibility not needing preliminary well-structured data where ML algorithms are sensible to any tiny changes in a data set. Finally, is the size of set of data where NN models are improved consistently, however ML algorithms would deteriorate. Finally, to learn any NN model expensive hardware is required while ML implementation itself requires professional expertise (Turing, 2020).

Having briefly described advantages and shortcoming of both approaches we may induce that ML algorithms are suitable only in case of moderate amount of data with the least quantity of outliers otherwise the final output may change depending on the increasing number of observations. Furthermore, Supply Chain Management is an area where data is being refreshed on a daily basis and any wrongly predicted result may cost huge amount of money. Therefore, the following narration will be dedicated to only neural networks and its hands-on application.

1.1 Recurrent neural network (RNN)

Recurrent neural network is a type of existing artificial neural network which utilizes sequential flow of data or time series one. Nowadays this type of algorithm is widely used for ordinal problems like image captioning, speech recognition, forecasting or language translation. This kind of architecture is already embedded into Siri, voice assistant, and google translator. One of the primary advantages of RNN is memory while conventional deep neural networks presume that both inputs and outputs are connectionless of each other. Meanwhile the output of RNN relies on the preceding elements in format of the sequence (IBM, 2020). To better grasp the neural network architecture, you may take a look at Figure 2 presented bellow which picturizes the difference between RNN and other traditional ones.



Source: (Biswal, 2022)

Figure 2: Simple RNN.

As you may observe from the image, RNN works on the principle of saving the result of the hidden layer which is brown one and feeding this back to the input in order to forecast the output of the layer. “x” labeled variables represent input layer, “h” is known as hidden one and “y” is the output layer. A, B, C are the network elements whose aim is to improve overall output result by fetching back the combination of input vector at time step x(t) and old state of x(t-1) (Biswal,2022).

Supremacy of this type of NN over the others is possibility of processing inputs of any length, the stability of model size while increasing of inputs and weights are shared across time. However, computation speed as well as absence of a possibility to take into account of any future input for the current state (Amidi, 2020).

Another distinguishing characteristic of RNN is that they share parameters across each layer of the whole neural network. Meanwhile feedforward networks have divers’ weights across the nodes, RNN have the same weight parameter within each corresponding layer of the network, but the weights are still adjusted in the through the processes of backpropagation and gradient descent to promote reinforcement learning.

However, there is a slightly different type of backpropagation process which is called through time BPTT algorithm to measure the gradients. The pattern of BPTT is the same as conventional backpropagation where the algorithm trains itself by calculating errors from its output layer to its input one. These calculations enable us to adjust and adapt the parameters of the model appropriately.

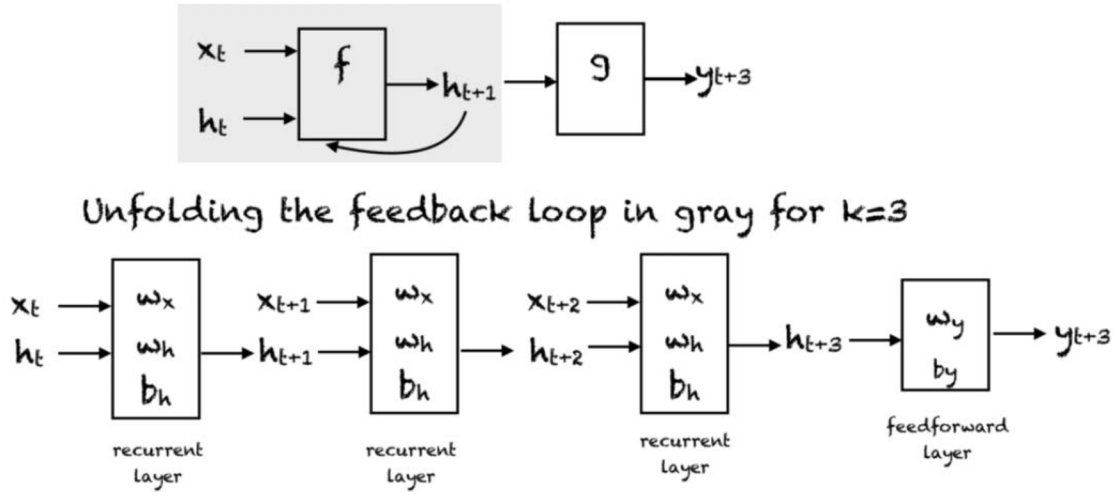
Backpropagation through time differentiates in sums of errors at each time step whereas traditional architecture of NN don't need to sum errors.

Thus, the RNNs are prone to run into other problems such as exploding gradients and vanishing ones. It may arise by the size of the gradient which is the slope of the loss function along the error curve. When the gradient is too small, it keeps going to consistently diminishing up to 0. Hence the algorithm is no longer learning, and it results in updating the weight parameters which stop its adjusting and overall accuracy becomes immutable. Vice versa tendency may occur when the gradient is too large, and it directly have effect on model's stability where the algorithm weights grow too large and become as NaN type. However, the solution is simple it is necessary to shrink the number of hidden layers eliminating the RNN model complexity (IBM, 2020).

Apart from advantages and drawbacks there are several types of RNN (Biswal, 2022):

- One – to – one. This variant of neural network is known as the Vanilla NN which has a single input gate and a single output gate.
- One – to – many. The type of NN has a single input and multiple outputs. Usually, the model is applied for image caption.
- Many – to – one. The RNN take several inputs and generate only one output gate.
- Many – to – many. This type of RNN takes sequence of inputs and generates a sequence of outputs. Usually, you may find the model application in machine translation.

Each of the models has been elaborated to solve respective real-life tasks. We may also present the neural network mathematically to understand the principle of the model in Figure 3.



Source: (Saeed, 2021)

Figure 3: Unfolding A Recurrent Neural Network.

In the gray rectangle can be unfold in 3-time steps to form the second network depicted below but during a RNN development it is possible to vary the architecture n time steps. The following-up notation is leveraged (Saeed, 2021):

- $x_t \in R$ is the input gate at time step t. The input data may be a scalar value with a single feature or n-dimensional vector.
- $y_t \in R$ is the output gate at time step t. One or multiple outputs in the network may be produced.
- $h_t \in R^m$ is the vector keeping the values of the hidden states at time t. This is usually called the current context where "m" is the number of hidden states and h_0 vector is initialized to zero.
- $w_x \in R^m$ are weights which are bound with inputs in recurrent layer.
- $w_h \in R^{m \times m}$ are weights connected with hidden states in recurrent layer.
- $w_y \in R^m$ are weights which are associated with hidden output states.
- $b_h \in R^m$ is just the bias which is associated with the recurrent layer.
- $b_y \in R$ is the bias as well which is associated with the feedforward layer.

At each time step we can unroll the network for k time steps to receive the output at time step with k+1. As you may notice that the unfolded network resembles a feedforward neural network and general formula (1) looks accordingly:

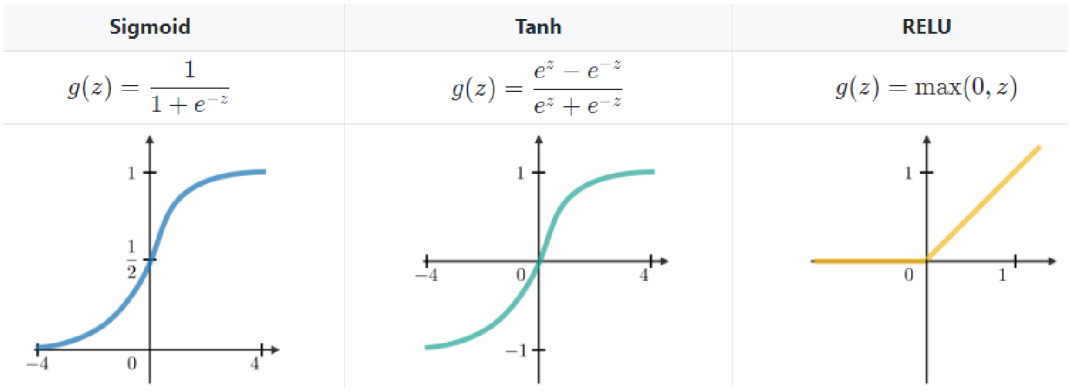
$$h_{t+1} = f(x_t, h_t, w_x, w_h, b_h) = f(w_x x_t + w_h h_t + b_h) \quad (1)$$

The output (2) of "y" at time t is compiled as:

$$y_t = f(h_t, w_y) = f(w_y * h_t + b_y) \quad (2)$$

As the result, the feedforward pass of a RNN, calculate the values of the hidden units and the output after k time steps iterations. The weights which are connected with the neural network are shared temporally and each recurrent layer has two set of weights where one is geared to input and the second one for hidden unit. The last layer computes the total result for k-th time step as a usual layer of a conventional neural network (Saeed, 2021).

Since the RNN's concept has been comprehended it is high time to precede to another integral part of any neural network which is known as activation function. Nowadays there are three of the most widely applied activation functions which employed according to the task which a network should fulfill. Activation function determines whether a neuron should be activated. If an ML developer builds a neural network with 7 or 8 hidden layers so it is recommended to leverage whether a sigmoid function or hyperbolic tangent. If a ML developer elaborates a neural network with over 7 or 8 hidden layers so it means an automatic switchover to the field of deep learning and the abovementioned activation function types become irrelevant because the result might be distorted or lead to appalling outputs. Hence, scientists have taken on another activation function for deep learning which is called RELU. The mathematical formulas and corresponding graphs are essential to understand how they function in the Figure 4 below (Amidi, 2020).



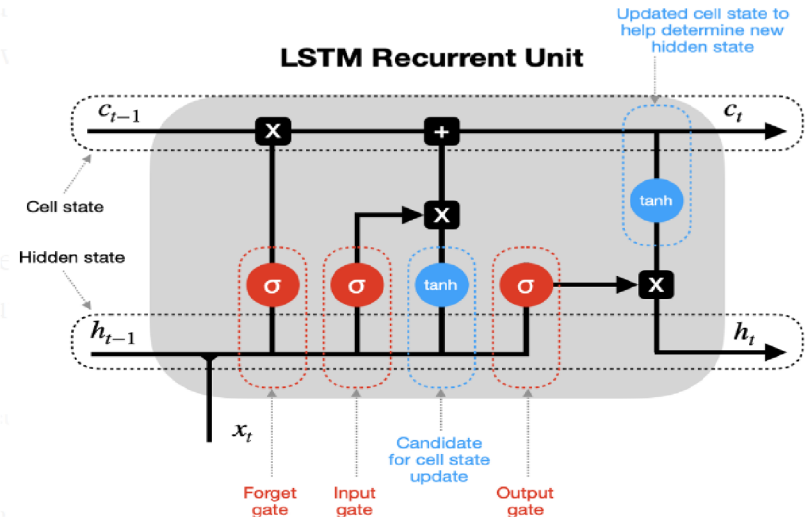
Source: (Amidi, 2020)

Figure 4: Activation Functions.

The nonlinear functions usually convert the output of a given neuron to a range of values between 0 and 1 or -1 and 1 (IBM, 2020). RNN can have as many hidden states as a ML developer may wish but there is one significant constant. Each hidden state must always have the same activation function and the output of each layer is calculated using the same function. You may ask which one of the given activation functions to leverage during a neural network elaboration? If the network stays within easy-solving issues you may use sigmoid, but some ML developers would recommend utilizing hyperbolic tangent because it decreases occurrences of vanishing gradient problem and RELU is always for deep learning (Bento, 2017).

We have already discussed the types of RNN, but another important topic is RNN architectures which should be mentioned. In today's day there is three types of architectures of RNN. The first one is bidirectional recurrent neural networks (BRNN). The difference between unidirectional RNNs is in drawing inputs from the previous records in order to make predictions about the current state but BRNN pulls in future data to improve the accuracy of it. Simply saying a neural network learning goes in two directions.

Long short-term memory networks (LSTM) are a unique type of RNN which can remember long-term dependencies. It also has a chain-like structure and instead of a single NN layer, four interacting layers are communicating as shown on Figure 5.

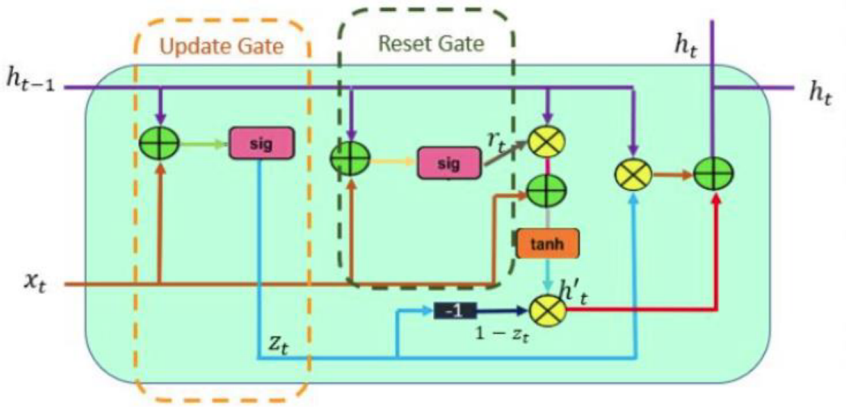


Source: (Dobilas, 2019)

Figure 5: LSTM's unfolded scheme.

To understand the contemporary LSTM model each part plays a significant role. h_{t-1} is hidden state from a previous timestep and x_t is the input at a current timestep which are all together combined before passing its copies through various gates. There is a forget gate controlling the information which should be forgotten. Sigmoid activation function ranges between 0 and 1 and it decides which values in the cell state must be thrown out multiplying an incoming value by 0 and to remember it multiplied by 1, partially remembered it multiplies by some value between 0 and 1. The input gate helps to distinguish significant elements that must be added to the cell state. The essential note where the results of the input gate get multiplied by cell state candidate with only the information deemed important by the input gate being added to the cell state. After that, c_{t-1} is the previous cell state which gets multiplied by the results received from the forget gate and then we add new information from input gate which has been multiplied by the cell state candidate in order to obtain the latest cell state denoted as c_t. The last step of neural network learning is hidden state updating. The latest cell state denoted as c_t is passed through the tanh activation function and multiplied by the results of the output gate. At the end the latest cell state c_t and the hidden state h_t come back in order to repeat the process at timestep t+1 and it lasts until we reach the end of the sequence (Dobilas, 2019).

In 2014 Kyunghyun Cho introduced a new type of RNN which is called gated recurrent units (GRU). GRUs are very similar to LSTM, and both have gates to control the flow of information. However, GRU is deemed as more alleviated than LSTM in terms of several differences. For instance, the former one has only two gates compared to the latter one having three gates. Moreover, GRU does not possess any internal memory and doesn't have an output gate. Apart from it, LSTM's both the input and target gates are coupled by an update one but in GRU reset gate is utilized directly for the antecedent hidden state (Lendave, 2021). To better understand the GRU's functionality is highly important to have a look at the neuron's structure that depicted on the Figure 6.



Source: (Lendave, 2021)

Figure 6: The unfolded GRU's neuron.

As you may see again the architecture reminds us LSTM where at each timestamp t, it takes an input x_t and the hidden state h_{t-1} from the previous timestamp t-1. Later, it produces a new hidden state h_t which again passed to another timestamp cycle and now there are primarily two gates in a GRU. The update gate is in charge of long-term memory, the equation (3) of the gate presented below.

$$z_t = \sigma(x_t * U_u + H_{t-1} * W_u) \tag{3}$$

U_u is simultaneously weight metric and differ from LSTM. Having processed elements through update gates they undergo the reset gate that is liable for the short-term memory of the whole network and the respective formula (4) presented below.

$$r_t = \sigma(x_t * U_r + H_{t-1} * W_t) \tag{4}$$

The r_t value ranges from 0 to 1 because of the sigmoid function and the U_r and W_r are weight metrices for reset gate.

Having completed both procedures, they head down to another two-step process. The first one oversees generating the candidate hidden state which has a corresponding formula (5).

$$H_t \quad = \tanh\left(x_t * U_g + (r_t * H_{t-1}) * W_g\right) \tag{5}$$

The formula (5) encompasses input data and the hidden state from the antecedent timestamp t-1 which is multiplied by the reset gate result r_t. Later, it passed the information through the tanh activation function, the output value is the candidate's hidden state. The most important part of equation is how the reset gate value is

used to control the hidden state influence on the candidate state. In case of r_t's value mads up 1 then it makes us out that the entire information from previous hidden state h_{t-1} is being taken into account of. Likewise, if r_t value makes up 0 then the entire data from antecedent hidden state is neglected.

Once the candidate state has arisen, it is time to generate the ongoing hidden state H_t and the update state comes into play. LSTM utilizes separate gate, but GRU applies only update gate in order to control both previous information which is denoted as H_{t-1} and the new information (6) from candidate state.

$$H_t = z_t * H_{t-1} + (1 - z_t) * H_t \tag{6}$$

Let's suppose that the z_t is around 0 then the first part of the equation will disappear which makes us out that the hidden state will not have much data from antecedent hidden state. However, the second term of formula gets one that means the current timestamp will contain data from the candidate state only and it may happen vice versa that the candidate state may equal 0 but the hidden state with historical information can be equal 1 (Saxena,2021).

To sum it up, you may observe some petite discrepancies between RNN, LSTM and GRU types but the final performance of the up-to-date approaches generates more robust result which is highly important in predictive analysis applying neural network as the main tool in solving real-life Supply Chain Management's task on demand prediction.

2 The factors of neural network training.

Any type of neural network has its own specifications and features, but some common things leave and must be considered during the development. Talking about details data scientist as well as ML developers have outlined the following attributes which must be calibrated in order to obtain a desired result:

- Optimization
- Standardization vs Normalization
- Overfitting and underfitting issues
- Evaluation indicators

2.1 Optimization algorithms

Any neural network model aims to generalize the incoming data with an algorithm and endeavors to predict unseen data. To map the inputs in order to receive the expected output it is necessary to optimize weights that minimize the error. Optimization algorithms significantly affect neural network accuracy. During the deep model learning the modification of epoch's weights and minimization or the loss function are desired. An optimizer is an algorithm that modifies both weights and learning rate. Hence, it improves the overall accuracy, but the task is to choose the correct weights for the model is becoming daunting since there are millions of combinations (Gupta, 2021).

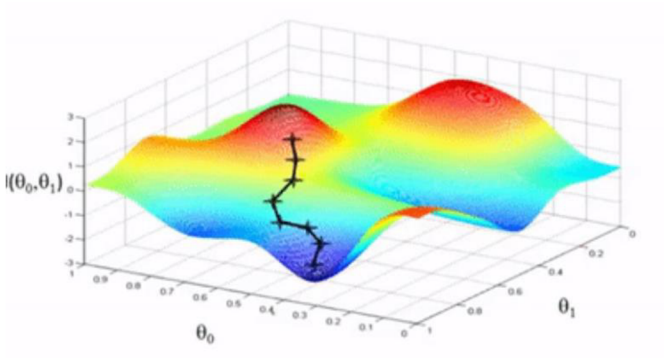
Nowadays there are nine optimizers which are well-known in hands-on ANN application (Nagesh, 2020):

- Gradient Descent
- Stochastic Gradient Descent (SGD)
- Mini Batch Stochastic Gradient Descent (MB-SGD)
- SGD with momentum
- Nesterov Accelerated Gradient (NAG)
- Adaptive Gradient (AdaGrad)
- AdaDelta
- RMSprop
- Adam

Gradient descent is the first foundational optimization algorithm of how the model is trained tweaking its parameters iteratively reaching out to its local minimum of a differentiable function. The primary goal is to minimize a cost function (7) as far as possible.

$$\theta = \theta - \eta * \nabla J(\theta) \tag{7}$$

The above-mentioned one is the parameter updates where 'ŋ' is learning rate and '$\nabla J(\theta)'$ is the loss gradient function. Gradient descent was used in order to optimize weights updating them in a direction of the loss function minimization. Once the Backpropagation approach has been mentioned and when the model propagates backwards the Network carries error terms updating weights values with the Gradient Descent.



Source: (Nagesh, 2020)

Figure 7: The Gradient Descent function.

In the gradient descent algorithm, there is a disguised variable which is part and parcel of final loss function output which is known as learning rate. Learning rate defines the convergence level by the assigned steps of gradient descent. In order to reach the local minimum, it is highly important to properly set the learning rate value which is neither too low nor too high. If the steps are too big it may not reach the local minimum bouncing back and forth between the convex function of the gradient descent. Vice versa, setting too small steps will get the local minimum, but it takes a while (Nagesh, 2020). Hence, we can deduce that gradient descent has both advantages and setbacks. On the one hand, it is easy to compute, to implement and to understand but it may trap at local minima, model's weights are changed after the gradient on the whole dataset where it takes years to converge to the minima if the dataset is too enormous and the approach consumes large

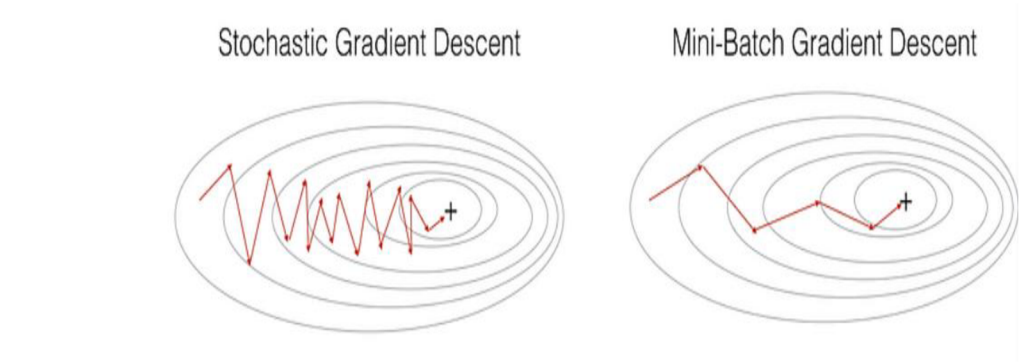
memory to calculate the gradient on the whole dataset. Hence the stochastic gradient descent has been elaborated rectifying previous disadvantages.

SGD is deemed as the extended version of the gradient descent where it performs a parameter update for each training example (8). The algorithm is faster and presents one update at a time.

$$\theta = \theta - \eta * \nabla J\big(\theta; x(i); y(i)\big) \tag{8}$$

Due to these frequent updates the high variance of parameters updates occurs and causes the loss function fluctuation. On the one hand, this is a good sign in a direction of the better local minima discovering but because of both frequent updates and fluctuations it compounds the convergence to the precise minimum keeping overshooting. Since most of disadvantages had left unsolved, another approach comes to play which is known as mini batch gradient descent (Nagesh, 2020).

MB-SGD is an extended version of the SGD, and it handled large time-consuming problems. This algorithm takes a subset of points from the dataset to compute derivative. However, after a while some surveys showed that the derivative of the loss function for MB-SGD is almost the same as a derivative of the loss function for conventional GD after some number of iterations. Moreover, the number of required iterations to get a local minimum is still large for MB-SGD as for GD. The updating of weights is dependent on the derivative of loss for a subset of points. MB-SGD' updates are much fuzzy because the derivative doesn't go toward minima. The basic principle of the algorithm is dividing the datasets into small batches and after each subset, the parameters are renewed. On average the mini batch is made up of size 50. The core advantage is less time-consuming in convergency way compared to the above-mentioned algorithms. However, some setbacks emerge such as noisiness and stuck at a local minimum (Nagesh, 2020).



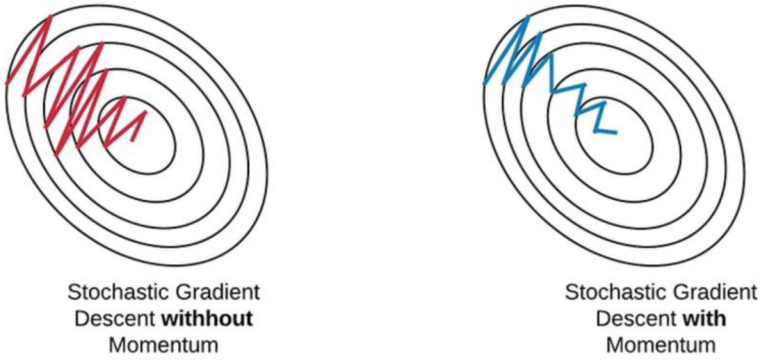
Source: (Nagesh, 2020)

Figure 8: The SGD vs MB-GD.

Since the problem is severe the demand for denoising left. Therefore, the SGD with momentum has been invented to overcome this issue denoising the gradients. Having denoised the gradients, the convergence time is decreased updating of weights by means of exponential weighting average using. As a result, it gives more weightage to the recently renewed updates in comparison with the previous update speeding up convergence and reducing fluctuation. However, a petite hyperparameter is embedded in this method (9) known as momentum and the formula looks accordingly:

$$\vee(t) = \gamma \vee (t-1) + a.\nabla J(\theta) \qquad (9)$$

The momentum term 'y' is usually set manually to 0.9 or a close value. 't' is momentum at time which is computed employing all previous updates providing with more weightage to recent updates compared to the past update accelerating the convergence. Here the momentum reminds the momentum in classical physics, when we cast a ball down a hill it collects momentum, and its terminal velocity keeps on mounting. The same one happens with parameters leading to the swifter and stable convergence reducing oscillations. As such, only one disadvantage is manually adjusted momentum (Nagesh, 2020).



Source: (Nagesh, 2020)

Figure 9: The SGD vs SGD with momentum.

As has been mentioned before, the momentum is set manually and is still a good approach but if the parameter is too high the overall algorithm may miss the local minima and can continue to increase. This problem endeavored to tackle Yuri Nesterov where the method makes a big jump based on the past momentum in the direction of the updated accumulated gradient then it calculates the gradient and then make a correction which results in a NAG update. This update avoids going too fast and does not miss the minima.

NAG compared to SGD with momentum secures from vague and unknown momentum's parameter giving him kind of prescience. Since we are already aware that we will use momentum term $\gamma \vee (t-1)$ to move forward the parameters θ modifying the weights (10) so,$\theta - \gamma V(t-1)$ roughly tells the next position of the parameters (11) which are going to be. Now, the calculation is based on these future parameters rather than the current one and the formulas look accordingly (Nagesh, 2020).

$$V(t) = \gamma V(t-1) + \eta \tag{10}$$

$$J = \left(\theta - \gamma V(t-1)\right) \tag{11}$$

The second one updates the parameters. But again, the hyperparameter must be selected manually.

Up to this moment all optimizers had the constant learning rate which is set 0.01 by default. To handle this indicator the Adaptive Gradient Descent (AdaGrad) has been created in order to have an adaptive learning rate for each of the weights. Hence, the algorithm performs tinier updates for parameters associated with frequently occurring features and bigger updates for rarely occurring features. For

this reason, it is well-suited for dealing with sparse data. To understand the main point, scientists used g_t to denote the gradient at time step and $g_{t,i}$ is then the partial derivative of the loss function to the parameter θ_i at time step, η is the learning rate and the $\nabla\theta$ is partial derivative of loss function (12) where the final formula is:

$$g_{t,i} = \nabla J(\theta_{t,i}) \tag{12}$$

And the SGD update function (13) looked like this:

$$\theta_{t+1,i} = \theta_{t,i} - \eta * g_{t,i} \tag{13}$$

But now the rule has been updated where the learning rate modifies at each time step for every parameter based on the previous gradient and the formula (14) looks accordingly:

$$\theta_{t+1,i} = \theta_{t,i} - \frac{\eta}{\sqrt{G_{t,ii} + \epsilon}} g_{t,i} \tag{14}$$

G is the sum of the squares of the previous gradients to all parameters. The advantage is that we don't need to set the learning rate, but the curse of the algorithm is squared gradients in the denominator because each newly added term is positive. Therefore, the accumulated sum keeps increasing during training of a neural network making the learning rate decrease and getting small and as the result in vanishing gradient problem. Simply speaking, with increasing number of iterations the learning rate leads to shrinking causing slow convergence (Nagesh, 2020).

That's why another couple of algorithms have been elaborated which are known as AdaDelta and RMSprop. To rectify the previous issue, AdaDelta algorithm has an idea to take an exponentially decaying average. AdaDelta adapts learning rate based on a moving window of gradient updates, instead of accumulating all previous gradients. Instead of saving previous squared gradients, the sum of gradients is determined as decaying average of all previous squared gradients. The running average depends only on the past average and ongoing gradient and formula (15) for this is:

$$E[g^2] = \gamma E[g^2]_{t-1} + (1-\gamma) g_t^2 \tag{15}$$

Where 'y' is set conventionally around 0.9 and rewriting the AdaGrad formula (16) we derive the following-up:

$$\Delta\theta_t = -\frac{\eta}{\sqrt{E[g^2]_t + \epsilon}} g_t \tag{16}$$

In RMSprop as almost identical one but Hinton proposed to use the learning rate by default is 0.001 (Nagesh, 2020).

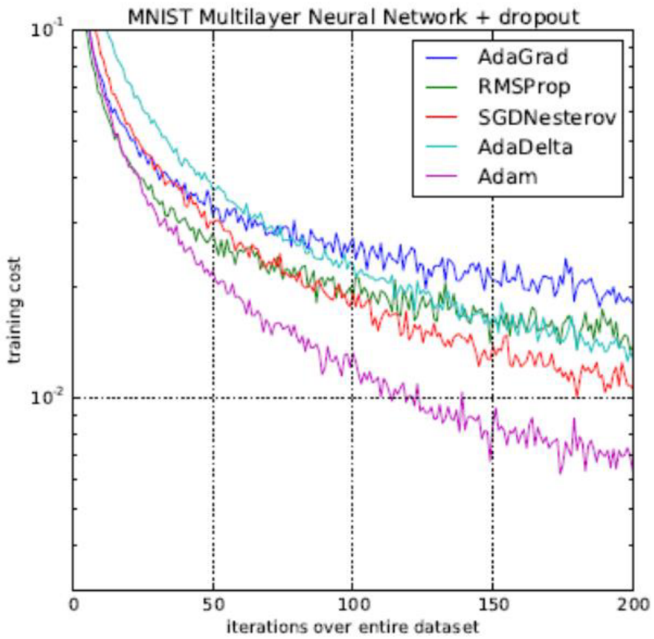
The last optimization function is Adaptive Moment Estimation (Adam) which can be considered as a mixing of RMSprop and SGD with momentum. Adam computes adaptive learning rates for each given parameter. Apart from storing an exponentially decaying average of previous squared gradients like RMSprop, Adam saves an exponentially decaying average of past gradients, like momentum.

Hyperparameters $\beta 1, \beta 2 \epsilon [0,1]$ control the exponential decay level of these moving averages. Scientists compute the decaying averages of previous ones and past squared gradients respectively:

$$m_t = \beta 1 m_{t-1} + (1 - \beta 1) g_t \tag{17}$$

$$v_t = \beta 2 v_{t-1} + (1 - \beta 2) g_t^2 \tag{18}$$

'mt' and 'vt' are evaluating of the first mean moment (17) and the second uncentered variance of the gradients (18).



Source: (Nagesh, 2020).

Figure 10: Comparative analysis of all optimization functions.

As the chart shows, the training cost attained the least moment for Adam. As for the rest of them, the SGD algorithm got stuck at a saddle point, so it can be used only for small networks.

AdaDelta is being the fastest followed by momentum algorithms. AdaGrad and AdaDelta algorithm are well-suited for sparse data. As for NAG and algorithms with momentum, they work well for most cases but much slower.

2.2 Normalization vs Standardization

Having figured out with the various types of optimizations, another petite one but essential detail plays role which is called data normalization. However, occasionally ML-engineers employ standardization instead of that's why it becomes important to clear the question up in order to enhance the neural network's performance.

Why should we normalize inputs in a neural network? The answer is simple. Usually, datasets have numeric input features which could take on values in potentially different ranges. For instance, let's assume that we have a dataset with several columns containing the weight and temperature of people who have got over COVID-19. It is obvious that weight is measured in kilograms, but temperature is in Celsius, and ranges may start from 0 to 200 ibs and 36 up to 45, for example. Hence, some mathematical techniques must be leveraged in order to eliminate the data scatter. Neglecting this step may lead to longer training time and to propagate through the layers of the network heading to the huge, accumulated gradients error that makes the training procedure unstable.

To avert the pointed-out problem there are two options for an input layer to choose either normalization (19) which works by mapping values of features to be in the range 0 to 1 applying the transformation:

$$x_{norm} = \frac{x - x_{min}}{x_{max} - x_{min}} \tag{19}$$

Or to use the standardization (20) one that follows a normal distribution with zero mean and unit variance in accordance with Gaussian distribution. Mathematically speaking, the transformation undergoes with mean μ, and standard deviation is given by:

$$x_{std} = \frac{x - \mu}{\sigma} \tag{20}$$

These two above-mentioned methods help to avoid the long-lasting convergence procedure of a neural network (Bala, 2021).

In 2015 Sergey Ioffe and Christian Szegedy found out that preprocessing normalization or standardization doesn't tackle problem with internal covariate shift or simply saying that the input layer is constantly changing due to weight update. Therefore, the corresponding layer must always be adapted to the new distribution. This tool performs a way of controlling and optimizing the distribution after each layer (Balawejder, 2020). Mathematically, batch normalization layer converts each input into mini batch by means of subtracting mean and dividing it by the standard deviation. However, we shouldn't expect inputs with zero mean and unit variance, but it can perform better with some other mean and variance. Hence, the BN layer introduces two learnable variables 'y' and 'β'. The corresponding formulas look respectively:

$$\mu_\beta \leftarrow \frac{1}{m}\sum_{i=1}^{m} x_i - mini - batch\ mean \tag{21}$$

$$\sigma_\beta^2 \leftarrow \frac{1}{m}\sum_{i=1}^{m}\left(x_i - \mu_\beta\right)^2 - mini - batch\ variance \tag{22}$$

$$x_i \leftarrow \frac{x \quad - \mu_\beta}{\sqrt{\sigma_\beta^2 + \epsilon}} - normalize \tag{23}$$

$$y_i \leftarrow y * x_i + \beta = BN_{y,\beta(x)} - scale\ and\ shift \tag{24}$$

Practically, the network calculates the gradient based on the ongoing inputs to any layer and reduces the weights in the way indicated by the gradient. However, the layers are stacked one after other, the data distribution to any layer changes swiftly due to slight weights update of earlier layer and as the result the gradient might perform sub optional signals for the network. But BN restricts the distribution to any layer which aids the network to constitute better gradients for weights update. Therefore, BN secures a much more stable and accelerated training pace.

However, there are some cons of BN. First, BN does the calculation of mean and variance in every training iteration, therefore it demands larger batch sizes while training so that it is able to prolifically approximate the mean and variance from

mini batch and it makes BN harder to train networks for solving such tasks as object detection, segmentation. It happens because of working with high input resolution and is applicable for large batch sizes. Another setback of BN is that it isn't well-suitable for RNN type of a neural network. The problem of RNN is a recurrent connection to previous timestamps and would requests a separate β and 'y' for each timestep in the batch normalization layer instead supplements additional complexity. The last interesting detail is difference in calculation approaches where during the test time, the BN doesn't calculate the mean and variance from the test data mini batches but applies the fixed mean and variance from precalculated training dataset (Vijayrania, 2020).

Inspired by the results of Batch Normalization, Jimmy Lei Ba, Jamie Ryan Kiros and Geoffrey E. Hinton presented the transformer paper where the layer normalization has been put forward in order to handle vectors mostly in the RNN outputs since BN didn't perform well. In comparison with BN, layer normalization is computed across all channels and spatial dims. Thus, the variance and mean value are independent of the batch. The formulas look respectively:

$$\mu_i = \frac{1}{m}\sum_{j=1}^{m} x_{ij} - mean\ value \tag{25}$$

$$\sigma_i^2 = \frac{1}{m}\sum_{j=1}^{m} x_{ij}\left(x_{ij} - \mu_i\right)^2 - variance \tag{26}$$

$$x_{i,j}^{new\ value} = \frac{x_{ij} - \mu_i}{\sqrt{\sigma_i^2 + \epsilon}} - normalize \tag{27}$$

It refers to the batches and j refers to the features, so as can be seen from the above-mentioned formulas, the features are also normalized and not the batches. LN is mainly applicable to RNN by normalizing separately at each time step. From this point of view, it reminds batch normalization but instead of its statistics, the mean and variance correspond to exact input to the neurons in a particular layer (Ampadu, 2021).

In 2016 Dmitry Ulyanov introduced Instance Normalization as another attempt to reduce dependency on the batch to improve the results of the style transfer network for CNN architecture. Normalization across both batch and channel allows removing specific contrast information from the image which provides the

generalization. Mainly the method has acquired popularity among generative types of models such as Pix2Pix or CycleGAN and became predecessor for Adaptive Instance Normalization which is used in the StyleGAN2 model.

In 2018 group normalization was introduced which directly addresses the batch normalization for CNN since the spike of computer vision occurred. The main disadvantage of BN as it has already mentioned before is dependency on the huge number of batches, at least 32 required but GN was invented with the idea of splitting tensor(batch) into many machines. Hence, these are trained on a small number of examples like 6 or 8 and sometimes even 1 or 2 keeping the stable low error rate compared to BN. It is kind of hybrid of layer and instance normalization where GN just divides channels into groups and normalizes across them. This scheme makes the computation independent of the batch sizes. However, when the group size falls to 16 or 32 and higher the error rate performs better for BN.

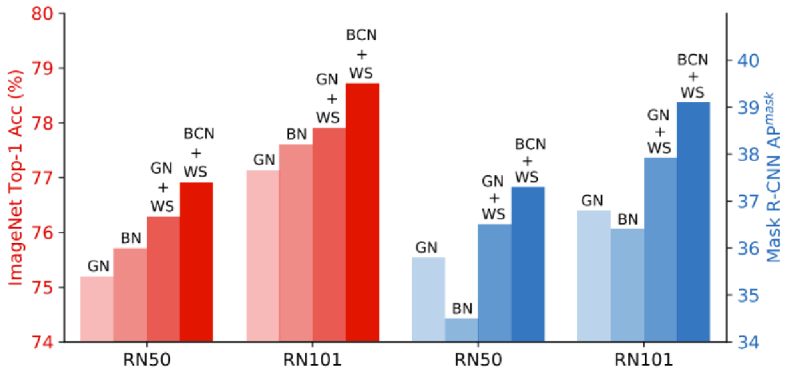
Almost every hyperparameter has been mentioned for normalization or standardization but weight has left intake. This theory has been proposed in 2019 about the weight standardization. Weights can grow large being out of any controller especially when the output normalizes. As a result, smoothing effect of weights is desirable. Since the conversation is about standardization the formulas are obvious:

$$W = \frac{W_{i,j} - \mu W_i}{\sigma W_{i,j}} \tag{28}$$

$$\mu_{wi} = \frac{1}{i}\sum_{j=1}^{i} W_{i,j} \tag{29}$$

$$\sigma_{wi} = \sqrt{\frac{1}{i}\sum_{i=1}^{i}\left(W_{i,j} - \mu_{wi}\right)^2} \tag{30}$$

In essence, weight standardization controls the first-order statistics of the weights of each output channel separately. In this way, WS is oriented to normalize gradients during the training. Theoretically speaking, the core idea is to keep the convolutional weights in a compact space smoothing the loss landscape and improving training. The result of different types of normalizations and standardizations and its hybrid forms presented on Figure 11 (Adaloglou, 2020).



Source: (Adaloglou, 2020)

Figure 11: Comparing normalization methods on ImageNet and COCO.

2.3 Overfitting and underfitting issues

Out of all existing things that go wrong with ML model or NN is a detrimental overfitting or underfitting error. It is a common pitfall for deep learning algorithms where a model attempts to fit the training data considering the intricacies and noise in the training data to the point where it detracts from its effectiveness on new data. It also involves noise or fluctuations in the training dataset. Simply saying, overfitting takes place because of too much learning from the data, and it can have influence on model's ability to generalize limiting its overall performance.

The model performs exceptionally well only in its training set however it does not generalize prolifically enough when used for predictions outside of the training dataset. Unfortunately, nonlinear models have less flexibility and more freedom in the target function during the learning process resulting in an overfitting problem. Moreover, no sample of the population can ever be unbiased. Overfitted models are biased toward sample rather than being independent and representative of the entire dataset.

Underfitting is a situation when it fails to detect the primary dataset's trend within the data leading to both training mistakes and poor model performance. Hence, if a model's ability to generalize to new data is limited as the result it can't be used for classification problem or forecasting tasks.

In order to recognize an underfitting or overfitting problem the model must be taught on the training dataset to see the results. If the model achieved 95% percent accuracy on the training set and 60% percent accuracy on the test set that

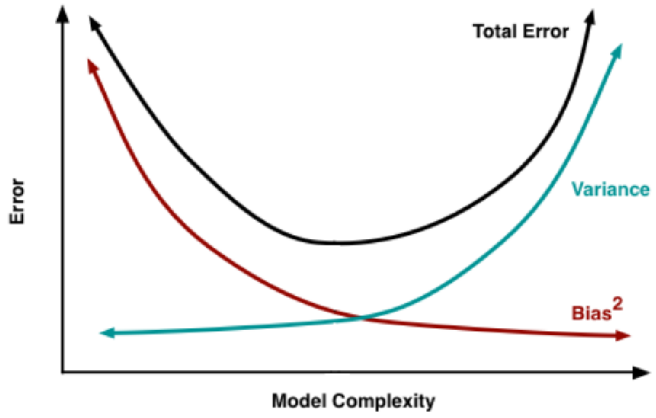
would be a huge overfitting problem. Vice versa, if the result shows a high level of bias and a low level of variance it means the model is underfitted (Larkin, 2022).

Having found out the definitions of both overfitting and underfitting problems, we haven't pinpointed the reasons for both issues which occur after training.

First of all, the overfitting emerges when data isn't cleansed and contains redundant values which distort the main data's trend capturing noise in a training dataset resulting to generalization failure. Secondly, the model can encompass high variance. Thirdly, the training dataset isn't ample, and the neural network is taught on the limited training data for several epochs. The last common reason is model's complexity, bigger number of neural network layers, longer time is required to teach the model and as the result it leads to overfitting problem.

As for underfitting causes, unclean data with noises or outliers might be reasons for a model's inability to trace down the primary pattern from the dataset. The second one is high bias due to the huge difference between the predicted value and target one. The last common mistake is oversimplification of the model when number of layers is too small, for instance (Baheti, 2022).

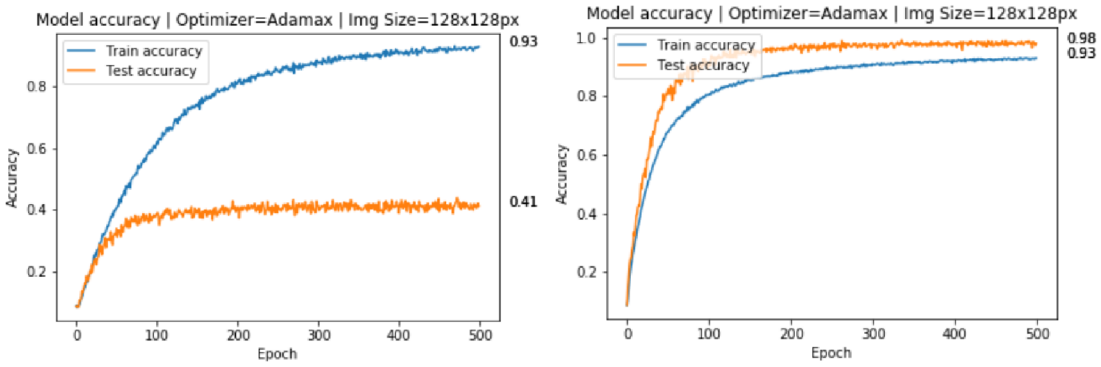
However, the reality might be tough and in practice most engineers run up against a phenomenon which is known as bias-variance tradeoff. The concept is simple where increasing model's complexity causes a lower bias error on the one hand but makes variance spike on the other. The ideal option is finding a sweet spot where the curves of variance and bias intersect as it is depicted on the Figure 12.



Source: (Oppermann, 2021)

Figure 12: Variance-Bias-Tradeoff.

We came off to define the most common reasons for overfitting and underfitting problems. Up to today the huge experience of all engineers has been collected in this area and there are some conventional solving options. Adding more data is the first approach which can eliminate this problem. Since the model fails to generalize to new data it means the data wasn't representative enough. So, the model must be retrained on a bigger and more diverse set of data to improve its performance. However, the data collection procedure can be expensive, or the number of data might be limited. Hence, the second option was put forward which is called data augmentation which is more affordable. Data augmentation makes a sample look a bit different every time the model processes it. As I mentioned before computer vision tasks became popular among engineers therefore this approach is geared to CNN's structure. Data transformation happens by means of artificially adding more images with horizontally flipped ones or vertically, they might be cropped or rotated up-side-down. This helps in increasing data size and thus reduces overfitting, as we add more information, the model gets incapable of overfitting all the samples and is enforced to generalize. The following result you can see on Figure 13 (Goyal, 2021).



Source: (Goyal, 2021)

Figure 13: Embedding of data augmentation into a neural network.

Since augmentation approach isn't applicable for RNN's structure another experience of engineers prompts that the model was trained on too complex number of data features thus a pattern hasn't been detected. Removing some features and making the data simpler can help to reduce overfitting problems.

However, some ML-engineers would like to keep a model's complexity and there is a list of tools which is known as regularization. The main idea standing behind it is complexity penalty to a model. If the model wants to avoid incurring a penalty it

must concentrate on the best patterns which contain a higher chance of generalizing well. These techniques are the most preferable and powerful and almost all the models contain them in some way (Chemama, 2018).

The first tool out of all existing ones is changing the network structure (number of weights). For instance, the structure could be tuned by means of grid search until a suitable number of nodes or layers is sought to reduce overfitting. Alternately, the model could be pruned by removing nodes until it satisfies suitable result on a validation dataset (Brownlee, 2019). The second tool is changing network parameters (value of weights) by adding a weight penalty term which penalizes the large value of weights in the network. Forcing the optimization algorithm to reduce larger weights values to smaller weights leads to stability of the network and introduces good performance. In this approach, the network configuration leaves unchanged since it only does modification of the weights value minimizing loss function as well apart from error reduction between predicted value and actual one.

$$(L1\ Regularization)\ Cost = \sum_{i=0}^{N} (y_i - \sum_{j=0}^{M} x_{ij} W_j)^2 + \lambda \sum_{j=0}^{M} |W_j| \quad (31)$$

$$(L2\ Regularization)\ Cost = \sum_{i=0}^{N} (y_i - \sum_{j=0}^{M} x_{ij} W_j)^2 + \lambda \sum_{j=0}^{M} W_j^2 \quad (32)$$

The first component represents loss function and the second one is the weight penalty of the regularization term. Without the regularization term the overall loss of the network is the same as the output of loss function. If the network overfits on training data, the error between predicted value and the actual one for a training data is very small. Therefore, since training error has a very small value, then the error gradient value is also very small. As the result, a change in weights is very small and when iteration is finished, the new weights value is almost the same as the previous one, thus the network still has overfit issue.

Vice versa, embedding regularization term causes error increasing, then the error gradient weights also rise which results in a significant change in weight update. If weights value gets larger the regularization penalizes it. So, larger values of weights result in a larger penalty to the loss function, thus pushing the network toward smaller and stabilized values of weights. There are obviously some differences between L1 and L2 presented in Figure 14.

L1 Regularization	L2 Regularization
1. L1 penalizes sum of absolute values of weights.	1. L2 penalizes sum of square values of weights.
2. L1 generates model that is simple and interpretable.	2. L2 regularization is able to learn complex data patterns.
3. L1 is robust to outliers.	3. L2 is not robust to outliers.

Source: (Goyal, 2021)

Figure 14: L1 vs L2.

L1 regularization (31) is usually applied for simple data and L2 one (32) is better choice if the data is too complex. However, there is not a silver bullet for it and a technique depends on the problem statement that engineer is trying to tackle (Goyal, 2021).

Apart from the above-mentioned ones there are additional regularization techniques such as activity regularization which penalizes the model during training base on the magnitude of the activations, dropout method which deactivates neurons ranging over from 20% to 50%, noise technique is added as statistical one to inputs during a training procedure, and the last one is early stopping which is widely employed monitoring model performance on a validation set and it stops training when performance degrades. Based on the ML-engineers' experience they prepared some recommendations for different structures of neural networks. For instance, during CNN's structure elaboration it is conventionally recommended to utilize early stopping and L2 regularization. Alternately, you can combine early stopping and added noise with a weight constraint establishing as a max value equal 1 and min one equals 0. However, contemporary approach suggests combining early stopping and dropout in addition to a weight constraint. As for RNN's structure, engineers conventionally use early stopping with added weight noise and weight constraint. However modern technique is combination of early stopping with dropout and weight constraint (Brownlee,2018).

Underfitting runes come across much rarely compared to previous problems but if it occurs it is necessary to increase complexity model switching from linear model to non-linear one adding more layers and neurons. If the model is not linear but there is an underfitting issue it is recommended to reduce regularization function influence. These techniques may help to overcome the underfitting problem (Chemama, 2018).

2.4 Evaluation indicators

Up to this moment we have almost everything to create any type of NN, however one missing significant component remains and must be embedded into the structure. The last part is evaluation metric which provides us with the most valuable information which determines further decision of a company. Depending on problem solving either classification task or regression one, there are related evaluation lists of metrics.

For Regression solving problem, ML-engineers traditionally as the first indicator uses mean absolute error (MAE). The MAE is simply defined as the sum of all the residuals (difference between the actual value and predicted one) divided by the total number of points in the dataset. It is the absolute average distance of our model prediction with the corresponding formula (33).

$$MAE = \frac{1}{N}\sum_{i=1}^{N}|y_i - \hat{y}| \tag{33}$$

As you can understand from the formula that negative residual is converted to positive so that it doesn't cancel out other positive residuals. Indicator depicts how close the predictions are to the actual model on average. Larger MAE values indicate that the model is poor at prediction. The MAE compared to the following indicators is robust towards outliers since it doesn't exaggerate errors. However, on the other hand, it doesn't give us an idea of the direction of the error, i.e., whether the value is under-predicting value or over-prediction the data.

Another popular one is mean squared error (MSE). Perhaps, it is the most popular metric in the field of regression analysis. Compared to the MAE it is a differentiable metric, so it can be optimized better. Moreover, it penalizes even small errors by squaring them which essentially leads to an overestimation of how bad the model is. However, the core disadvantage is the metric's propensity to outliers.

Since the MSE is inclined to outliers, root mean squared error (34) has been created as an improvement to the MSE. Mathematically, it can be represented as:

$$RMSE = \sqrt{\frac{1}{N}\sum_{J=1}^{N}(y_i - \hat{y})^2} \tag{34}$$

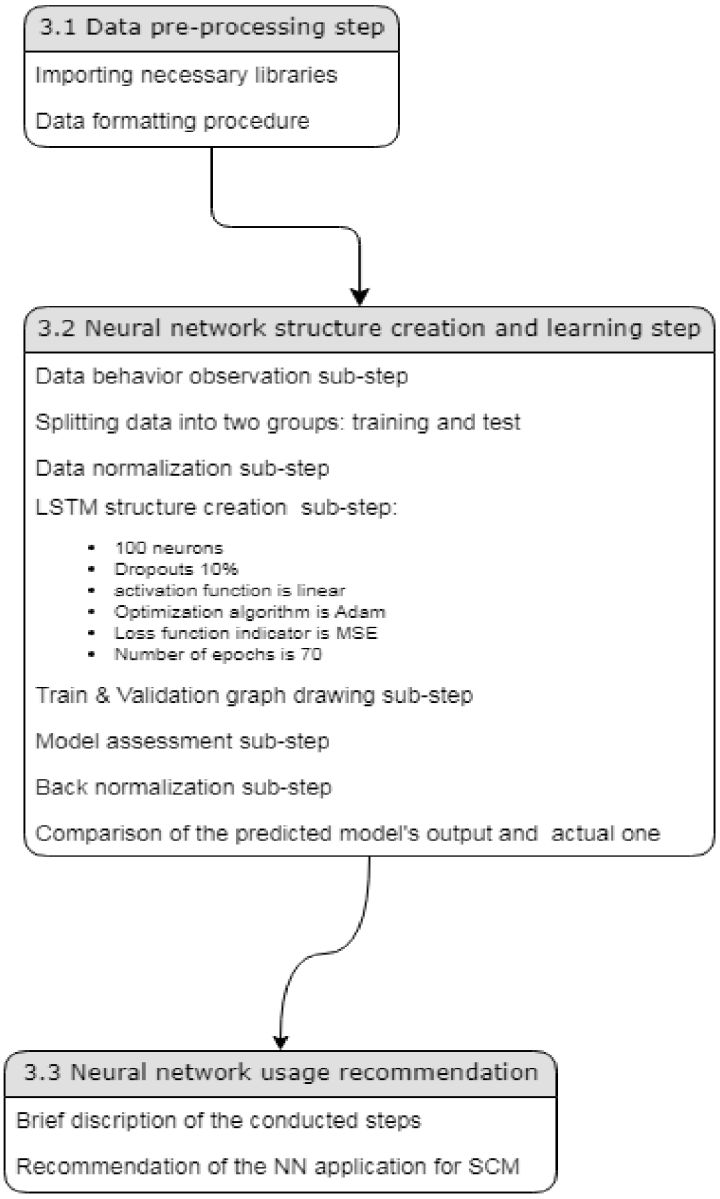
It mainly addresses a few downsides in MSE. It still retains the differentiable property of MSE. It still handles the penalization of smaller errors done by MSE by square rooting it and it is less prone to struggle with possible outliers (Shwetha, 2021).

Other types of metrics relate to ML types of algorithms which I decided to omit.

To sum it up by now, we can certainly say that neural network training is quite a challenging procedure since each petite component must be considered to attain the highest possible precision score and model's correctness toward predictions. Each percentage of accuracy matters especially when we intend to try neural network models toward possible embedding into supply chain management where even tiny mistake can cause possible subsequent expenses. Nevertheless, huge companies shouldn't rule out such algorithms which potentially can automate or even generate more revenue or avert from unseen predicted expenses.

3 Process of the neural networking learning to predict future price based on example of PAO Gazprom shares.

Every single neural network must be taught and tailored according to the given data since each dataset contains its own tendency, features, and difficulties. To better understand the neural network process teaching it is compulsory to demonstrate a hands-on example. As an instance, the time-series data has been taken to show the Gazprom share price prediction being taught on LSTM approach, which is deemed as contemporary approach to forecast weather, or a team win during a World Cup. However, a bid has been taken to embed the model in Supply Chain since for neural network doesn't matter the data's classification, but time-series structure is crucial for it.



Source(Dzhuraev, 2022)

Figure 15: The practical part map.

As you can observe from the Figure 15, each step in the scheme corresponds to the sub-title of the chapter 3 and has its one sub-steps which delve into details of the neural network elaboration.

3.1 Data pre-processing step

The first part of any neural network teaching is data extraction and its data pre-processing step. At the beginning, it is necessary to import libraries which will be employed during the above-mentioned steps.

For data manipulation procedure, there is a bundle of libraries such as ‘NumPy’, ‘pandas’ and ‘matplotlib’. As for ‘NumPy’ and ‘pandas’, there are needed to extract data with different formats like xml, excel, csv, txt and many others. Moreover, we can use these libraries for the creation of tables, adding or deleting columns, rows, values containing other helpful functions. Another library is geared to build graphs of various types to depict tendencies, to show dependencies of x and y values.

Another group of libraries such as ‘keras’ and ‘tensorflow’ are required for direct neural network teaching. It is traditionally to import these two libraries because they have been elaborated by Google who delves into neural network for a long time securing its status in this field having invented such prominent neural network like LaMDA who ponders like human, or another example is google translator which is leveraged by a lot of users around the world.

```
import tensorflow as tf
import numpy as np
import matplotlib.pyplot as plt
import pandas as pd
```

```
from keras.models import Model
from keras.layers import Input, Dense, LSTM, Dropout
from tensorflow.keras.layers import BatchNormalization
from keras.callbacks import EarlyStopping, ModelCheckpoint
```

Source: (Dzhuraev, 2022)

Figure 16: Libraries importing.

On the figure 16 you can see how it is committed practically importing some necessary tools for further application. If it is presumed to upload whole tools from

a library, you need to begin the command from the word ‘import’. If you plan to import only a few of them, it is necessary to start from the word ‘from’ specifying a concrete tool which you are interested in.

Having summoned the methods, the next step is to extract data from a file using ‘pandas’ with the command ‘read_csv’ and in parentheses to write a file title and specifying separator. Afterword, some columns have been renamed to remove redundant characters from headers. Another function is converting date type column into readable format tapping ‘pd.to_datetime’ and in brackets pointing out column and time format.

```
df = pd.read_csv('GAZP_000101_221118.csv', sep =',')
print(df.shape, df.columns)
df.rename(columns={'<DATE>':'Date','<TIME>':'Time', '<OPEN>':'Open',
                   '<HIGH>':'High','<LOW>':'Low','<CLOSE>':'Close','<VOL>':'Volume'}, inplace = True)
print(df.shape, df.columns)
df['Date'] = pd.to_datetime(df['Date'], format='%Y%m%d')
print(df.shape, df.columns)
df = df.drop('Time',1)
```

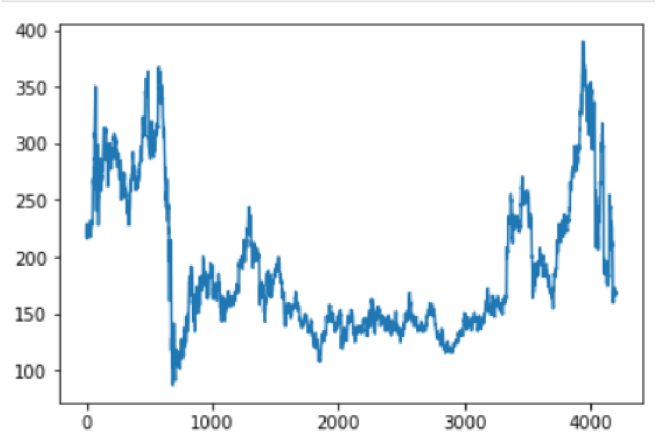
Source: (Dzhuraev,2022)

Figure 17. Data pre-processing.

Having pre-processed data in the respective step, we have approached to direct neural network teaching step.

3.2 Neural network structure creation and learning step

Before beginning to neural network structure creation, we should have a look at data tendency in order to highlight some crucial data districts on Figure 18.



Source:(Dzhuraev, 2022)

Figure 18: PAO Gazprom shares tendency.

As you may see from the graph, vertical axes depict price per share and horizontal one is row number over the time. The data has been taken from January beginning 2006 up to 16 of December 2022 at close moment on stock market in order to receive as hard data structure as possible under an unstable market when most of existing prediction models depict extremely distorted result. However, we may presume that is a demand for cars per day where a company should produce, and our goal is to predict demand of required cars for tomorrow under the extremely fluctuated market. According to the figure, the company had several nosedives because of market instability in 2008 and in 2022 being sanctioned. In terms of current models such as regression models or logistic one is strict requiring to follow some rules. As for LSTM model, it can remember the input data with all fluctuations, nosedives and spikes and considers them during the prediction output.

First crucial part is splitting whole data set into trained one and test. From the title is understandable that train chunk will be fed to LSTM and test one will assess the mistake between trained one and actual one. Conventionally data scientists split according to the ratio of 80% per cent for training and 20 % per cent for test part. In our particular case the goal is to predict price of share at the close moment. That's why we split data taking only 'close' column and 'volume' one depicted in Figure 18.

```
split = 0.80
i_split = int(len(df)* split)
cols = ['Close','Volume']
data_train = df.get(cols).values[:i_split]
data_test = df.get(cols).values[i_split:]
len_train = len(data_train)
len_test = len(data_test)
print(len(df), len_train, len_test)
```

```
4201 3360 841
```

Source:(Dzhuraev, 2022)

Figure 19: Data splitting step.

As we can see from the picture, the data set has 4201 rows where 3360 samples will be trained and 841 will assess the result during the validation procedure.

Next crucial part is data normalization process for both chunks training one and testing batch. Without normalizing of data, the model would never converge since the value is constantly moving to absolute type. To combat this n-sized window of training and test data will be taken and normalized each one to reflect percentage

changes from the start of that window. The normalization procedure will employ the formula which has already been mentioned before:

$$x_{norm} = \frac{x - x_{min}}{x_{max} - x_{min}} \tag{35}$$

According to this equation (35) the values will be converted to range from 0 to 1. These windows will also allow us to draw the data result at the end utilizing python generator which means memory utilization will be minimized dramatically. The way the windows creation goes depicted on the Figure 20.

```
def normalise_windows(window_data, single_window=False):
    normalised_data=[]
    window_data=[window_data] if single_window else window_data
    for window in window_data:
        normalised_window=[]
        for col_i in range(window.shape[1]):
            max_in_column = max(window[:, col_i])
            min_in_column = min(window[:, col_i])
            normalised_col = [((float(p) - float(min_in_column)) / (float(max_in_column) - float(min_in_column))) for p in windo
            normalised_window.append(normalised_col)
        normalised_window = np.array(normalised_window).T
        normalised_data.append(normalised_window)
    return np.array(normalised_data)
```

Source: (Dzhuraev,2022)

Figure 20: Data normalization step.

Having shown the data normalization step, the most significant one is a neural network model creation. The whole theory part which has been dedicated to the parameters which are set in a NN model considering data specification and task description. For this part we have everything to build it and the example is depicted in Figure 21.

```
model = tf.keras.Sequential([
    tf.keras.layers.LSTM(100, input_shape=(sequence_length-1, input_dim), return_sequences=True),
    tf.keras.layers.Dropout(.1),
    tf.keras.layers.LSTM(100, return_sequences=True),
    tf.keras.layers.LSTM(100, return_sequences=False),
    tf.keras.layers.Dropout(.1),
    tf.keras.layers.Dense(1, activation='linear')
])

model.compile(optimizer='adam', loss='mse', metrics=[tf.keras.metrics.MeanSquaredError()])
callback = [tf.keras.callbacks.TensorBoard(log_dir=log_dir, histogram_freq=1), EarlyStopping(monitor='val_loss', patience=2), Mod
```

```
history=model.fit(x_train, y_train, epochs=70, batch_size=batch_size, validation_split=0.2, callbacks=callbacks)
```

Source:(Dzhuraev, 2022)

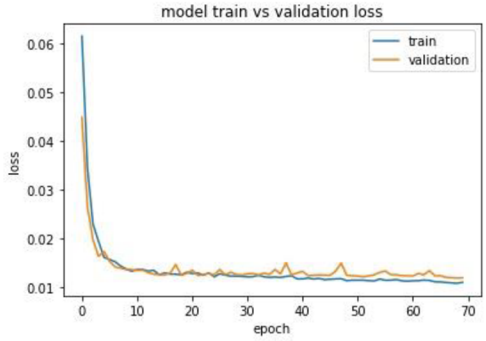
Figure 21: NN model creation step.

First, as you may see from the scheme, the type of the model is sequential one because we need to build one-to-one model with single input and output gates. Since we consider only the ‘close’ column for prediction that’s why input gate and

output one has only one entrance and one exit. Each layer will consider 100 neurons. It is a standard number which is just assigned by ML-engineers conventionally at the beginning to see the result to decide either to increase number of them or decrease depending on many factors as underfitting or overfitting problem presence, number of features and task hardness for the neural network. In our case we have only around 4 thousand rows with 1 column hence there is no necessity to increase the neural network structure hardness. Another important part is 'return_sequences' is assigned as 'True' since we have LSTM structure which must return result and memorize the past data specification. After the first layer and last one we assigned dropout function to preliminary prevent overfitting issue turning off 10% neurons to decrease the model hardness. As for activation function, we assigned 'linear' one since the number of observations is small and quantity of layers is petite. Below of the model structure, you can see the 'adam' optimizer which is deemed the best one elaborated by scientists. As it has been mentioned before, it computes adaptive learning rates for each given parameter storing an exponentially decaying average of previous squared gradients like RMSprop and saving an exponentially decaying average of past gradients, like momentum. For assessing the model result we consider the 'mean squared error' metric to see lowest possible error between real data and predicted one to achieve the best possible result decreasing the number of epochs if it is needed.

After 70 epochs of learning, we should see graphically the coincidence of training data and validation one to assess the model's quality on Figure 22.

```
plt.plot(history.history['loss'])
plt.plot(history.history['val_loss'])
plt.title('model train vs validation loss')
plt.ylabel('loss')
plt.xlabel('epoch')
plt.legend(['train', 'validation'], loc='upper right')
plt.show()
```



Source:(Dzhuraev, 2022)

Figure 22: Train vs validation loss.

As we can deduce from the Figure 22, two curves depict stable learning procedure without underfitting of overfitting issues thanks for both techniques such as earlier stopping and dropouts which have been engaged in the code as preventive measure according to the contemporary recommendations (Browniee,2018). Train and validation curves are almost coincided, it signals the completed healthy teaching step. However, it is compulsory to verify it according to measuring metric which clearly assesses the outcome depicting the least mean square error between real result and the predicted one on Figure 23.

```
model.evaluate(x_test, y_test, verbose=2)
```

```
25/25 - 1s - loss: 0.0120 - mean_squared_error: 0.0120 - 692ms/epoch - 28ms/step
[0.01198236271739006, 0.01198236271739006]
```

Source:(Dzhuraev, 2022)

Figure 23: Mean squared error result.

MSE depicts the graph of train and validation curves where the difference between real data and the predicted one is 0.0120 whereas it is low. The larger the number the larger the error. In this case we have obtained the mean squared value less than 0 meaning that the model is perfect.

So, now we must normalize back the values of the data set to see the predicted value and compare it with the actual one to see how close we are. To normalize back the result, we just take each value from the range multiplying it by the difference between maximum value and minimum and adding minimum one. The code implementation is written below in Figure 24.

```
def de_normalise_predicted(price_max, price_min, _data):
    return (_data*(price_max-price_min)+price_min)
```

Source:(Dzhuraev, 2022)

Figure 24: Back normalization function.

Having normalized back the data we can see the predicted price and compare it with the real one. Let's see the predicted output and the actual one in Figure 25.

```
predicted_price=de_normalise_predicted(last_data_2_predict_prices_max, last_data_2_predict_prices_min, predictions2[0][0])
predicted_price
```

```
160.126975011304
```

Source:(Dzhuraev, 2023)

Figure 25: PAO Gazprom share price per one unit.

It is time to see the actual outcome from Moscow's stock market the share price of PAO Gazprom 19 of December keeping in mind that we have data up to 16th of December on Friday in Figure 26.

Source:(PAO Gazprom, 2022)

Figure 26: PAO Gazprom share price per one unit in Stock market by 19th of December.

As we can see from both Figures the predicted price made up 160.12 ruble per one share and the actual one was 160.65. The difference is extremely low what we can strongly claim that the model's accuracy is very high, and it becomes as a candidate for its possible embedding into supply chain to predict not only annual company's demand on monthly or annual basis, but it is able to predict possible upcoming procurement of raw materials based on memorized data from the past.

However, model has one potential disadvantage it is compulsory to maintain its accuracy automatically updating data inputs by means of database plugging via supplement libraries or applying 'rest API' to optimize the data flow into table memorizing new info and producing an adjusted outcome which is close to reality.

To sum it up, we can certainly claim that LSTM NN structure is suitable for supply chain because the model's feature is able to memorize the data from the past considering diverse tendencies. LSTM application unleashes the opportunity to reduce cost of a company cutting off the number of employees, optimizing demand by means of predicting possible demand for upcoming month or year avoiding overproduction. Hence, the model can be considered as a possible tool for supply chain on several levels.

3.3 Neural network usage recommendation

LSTM isn't intricate tool for application, but some expertise is required to use prolifically. First, any data set needs to be refined before being fed to neural network cleansing out from redundant symbols such as brackets, titles beginning from capital character or adjusting data types converting float format column into integers one. Secondly, the data set must be normalized or standardized to get convergence of a neural network. Sometimes, if the task requires the string format columns must be converted into binary variables. Thirdly, the dataset must be sliced into two groups for training procedure and testing under the proportion 80/20. Another step is splitting data into batches where 32 tensors are recommended one. To map and visualize the result graphically we need to create a window which analyzes the first number of data and the last one oversees data prediction drawing the result on a graph. The crucial part is LSTM structure creation specifying return outcome after each iteration. At the beginning, it is recommended to set up 100 neurons keeping in mind possible overfitting problem employing dropouts, early stopping, regularization and other tools to solve it. Moreover, the number of epochs with loss function indicator provides help tracing down the learning procedure and as soon as the problem arises to stop the process catching the least possible error. Having wound up the neural network teaching process without any problems, the back normalization leaves to see the predicted result. That's the shortcut of LSTM neural network procedure.

However, any data set has its own specifications, intricacies, and features. Apart from them, a task defines the LSTM structure with different well-suitable parameters. It is strongly recommended to apply this model after several years since the model must be tested to check its reliability. If we ignore the recommendation, a business can generate unwanted losses since the model hasn't been adapted to specific requirements and doesn't meet the unpredictable challenges. Moreover, the model is useless if there is not a plugged database with constantly updating data otherwise the model would produce distorted results. Therefore, stable database maintenance and keeping an eye on the trained model are part and parcel of overall prediction result.

As for the direct model embedding into supply chain, it is already possible to implement it into the process. For instance, if a procurement department requires

to order specific number of some items, the LSTM model can fulfill the same task without human interference or in an inbound logistics the number of produced items for tomorrow or next month the system can recommend considering the information from the past.

We can certainly claim that Industry 4.0 is still able to unleash its potential on all levels of a supply chain optimizing production reducing cost of a company by means of cutting off workers consistently substituting them depends on AI overall quality performance.

Conclusion

The purpose of this thesis is application of neural network advantages for a possible embedding into supply chain management. The thesis was carried out individually and under the guidance of the professor ing. Tomash Malchic, Ph.D. The data was provided by the Moscow stock market website to demonstrate the neural network abilities to consider data from the past for further prediction of data's output with extremely low mean squared error between forecasted result and actual one.

The first chapter of the thesis described the theoretical part of the subject that the topic is dealing with. In detail about literature search in field of AI's edges in supply chain utilized by several big companies which test not only computer vision but invent smart forklift. Moreover, there were presented different types of neural networks fulfilling its own tasks but the chapter was delved more into RNN and LSTM structures which have been practically considered and elaborated in subsequent chapters.

The second chapter went into details about factors which must be taken into consideration during the neural network elaboration such as myriad types of optimization algorithms, distinction between standardization and data normalization procedures, overfitting and underfitting issues, evaluation indicators. Each of above-mentioned element can significantly distorts overall final result if the mathematical understanding is omitted. That's why the chapter is fitted with mathematical formulas unleashing shroud from AI and its inherent complexity.

The third chapter is dedicated to direct presentation of practical application of neural network in time series dataset. The neural network model has been designed on python language in the Jupiter Notebook software. The whole process has been described begging from external libraries plugging such as Tensorflow, Keras, Pandas, winding up with comparison between the predicted result and actual one. Moreover, some recommendations have been suggested for its possible embedding into supply chain since the mean squared error made up much less than 0 proving its ability to predict something under unstable situation on the market if the data has at least time series structure.

The problem has been defined and the evaluation has been carried out regarding the data set specification and its features. First of all, some necessary libraries have been uploaded for further application for specific tasks. Secondly, some necessary libraries such NumPy and Pandas have been utilized in order to clean out redundant features, to rename columns, to change time format and many other things. Having completed the data pre-processing step, the neural network structure elaboration process went ahead. The data set has been splitted into two parts training chunk and testing one according to 80/20 ratio with further windows creation for drawing results and prediction of output. Afterword, the LSTM one-to-one structure with embedded dropouts and early stopping attributes has been created with 100 neurons at each layer apart from the last one which was forming the predicted output preventing possible overfitting issue. Having trained the neural network, the graph with training line and validation one has been drawn in order to observe the possible arising of overfitting problem. Apart from it, we assessed the mean squared error to understand the model's quality for this particular data set. The last step was an output prediction which we compared with the actual one where distinction was extremely low thus proving the model's usefulness for a possible application into supply chain. Some recommendations have been put forward to apply neural network in supply chain.

Bibliography

Books and monographs:

HANNAH, Wenzel, Smit DANIEL and Sardesai SASKIA. A literature review on machine learning in supply chain management. Proceedings of the Hamburg International Conference of Logistics (HICL), o.p.s, 2019. ISBN 978-3-7502-4947-9.

MOROFF, Nikolas. Machine Learning in Demand Planning: Cross-industry Overview, Proceedings of the Hamburg International Conference of Logistics (HICL), o.p.s, 2017. ISBN 978-3-7502-4947-9.

TJAHJONO, Benny. What does Industry 4.0 mean to Supply Chain ?. UK: School of Applied Sciences Cranfield University, o.p.s, 2017. DOI 10.1016/j.promfg.2017.09.191.

WITTEN, Ian, Frank EIBE and Mark HALL. Data Mining: Practical Machine Learning Tools and Techniques. USA, o.p.s, 2011. ISBN 978-0-12-374856-0.

Articles in professional journals:

JIE, Ren. Deep Learning-Based Intelligent Forklift Cargo Accurate Transfer System. MDPI. 2022, **22**(1), 6-7.

TERPEND, Regis, TYLER, Beverly, Daniel KRAUSE and Robert HANDFIELD. Buyer-supplier relationships: Derived value over two decades. Journal of Supply Chain Management. 2008, **44**(2), 22-24.

MATHES, Tim, Pauline KLAßEN and Dawid PIEPER. Frequency of data extraction errors and methods to increase data extraction quality: a methodological review. BMC Medical Research Methodology. 2017, **152**(17), 5-6.

ZHIHAN, Lu and Xie SHUXUAN. Artificial intelligence in the digital twins: State of the art, challenges, and future research topics. Digital Twin. 2022, **12**(1), 14-16.

Websites:

AIM [online]. USA: LENDAVE, Vijaysinh, a.s., 2021 [2022-08-20]. Available from: https://analyticsindiamag.com/lstm-vs-gru-in-recurrent-neural-network-a-comparative-study/.

AllCloud [online]. USA: CHEMAMA, Jonathan, a.s., 2018 [2022-09-24]. Available from: https://allcloud.io/blog/how-to-solve-underfitting-and-overfitting-data-models/.

AI Pool [online]. USA: AMPADU, Hyacinth, a.s., 2021 [2022-09-04]. Available from: https://ai-pool.com/a/s/normalization-in-deep-learning.

AI Summer [online]. USA: ADALOGLOU, Nikolas, a.s., 2020 [2022-09-05]. Available from: https://theaisummer.com/normalization/.

Fireblaze AI School [online]. India: ALMEIDA, Riona, a.s., 2021 [2022-11-15]. Available from: https://www.fireblazeaischool.in/blogs/how-tesla-uses-ai-and-cv/.

IBM [online]. USA: Recurrent Neural Networks, a.s., 2020 [2022-08-13]. Available from: https://www.ibm.com/cloud/learn/recurrent-neural-networks.

KDnuggets [online]. USA: NAGESH, Singh Chauhan, a.s., 2020 [2022-08-21]. Available from: https://www.kdnuggets.com/2020/12/optimization-algorithms-neural-networks.html.

Levity [online]. USA: LARKIN, ZOE, a.s., 2022 [2022-09-21]. Available from: https://levity.ai/blog/overfitting-vs-underfitting-in-machine-learning.

Machine Learning Mastery [online]. USA: SAEED, Mehreen, a.s., 2022 [2022-11-17]. Available from: https://machinelearningmastery.com/an-introduction-to-recurrent-neural-networks-and-the-math-that-powers-them/.

Machine Learning Mastery [online]. USA: BROWNLEE, Jason, a.s., 2019 [2022-11-11]. Available from: https://machinelearningmastery.com/introduction-to-regularization-to-reduce-overfitting-and-improve-generalization-error/.

Medium [online]. USA: BENTO, Karolina, a.s., 2022 [2022-08-18]. Available from: https://towardsdatascience.com/recurrent-neural-networks-explained-with-a-real-life-example-and-python-code-e8403a45f5de.

Medium [online]. USA: DOBILAS, Saul, a.s., 2022 [2022-08-19]. Available from: https://towardsdatascience.com/lstm-recurrent-neural-networks-how-to-teach-a-network-to-remember-the-past-55e54c2ff22e.

Medium [online]. USA: SAXENA, Salli, a.s., 2021 [2022-08-20]. Available from: https://www.analyticsvidhya.com/blog/2021/03/introduction-to-gated-recurrent-unit-gru/.

Medium [online]. USA: GUPTA, Allan, a.s., 2021 [2022-08-20]. Available from: https://www.analyticsvidhya.com/blog/2021/10/a-comprehensive-guide-on-deep-learning-optimizers/.

Medium [online]. USA: BALAWEJDER, Maciej, a.s., 2022 [2022-09-02]. Available from: https://medium.com/nerd-for-tech/overview-of-normalization-techniques-in-deep-learning-e12a79060daf.

Medium [online]. USA: VIJAYRANIA, Nilesh, a.s., 2020 [2022-09-03]. Available from: https://towardsdatascience.com/different-normalization-layers-in-deep-learning-1a7214ff71d6.

Medium [online]. USA: GOYAL, Chirag, a.s., 2021 [2022-09-05]. Available from: https://www.analyticsvidhya.com/blog/2021/06/complete-guide-to-prevent-overfitting-in-neural-networks-part-1/.

Medium [online]. USA: SHWETHA, Acharya, a.s., 2021 [2022-09-15]. Available from: https://towardsdatascience.com/what-are-rmse-and-mae-e405ce230383.

Pinecone England [online]. London: BALA, Priya, a.s., 2021 [2022-08-21]. Available from: https://www.pinecone.io/learn/batch-layer-normalization/.

Stanford.edu [online]. USA: AMIDI, Afshine, a.s., 2020 [2022-08-14]. Available from: https://stanford.edu/~shervine/teaching/cs-230/cheatsheet-recurrent-neural-networks.

Supply Chain [online]. USA: Muddassir, Ahmed, a.s., 2016 [2022-09-13]. Available from: https://supplychaindigital.com/top10/seven-reasons-why-you-need-forecast-supply-chain.

Simplilearn [online]. USA: AVIJEET, Biswal, a.s., 2022 [2022-12-01]. Available from: https://www.simplilearn.com/tutorials/deep-learning-tutorial/rnn#GoTop.

Turing [online]. USA: Deep Learning vs Machine Learning: The Ultimate Battle, a.s., 2021 [2022-08-12]. Available from: https://www.turing.com/kb/ultimate-battle-between-deep-learning-and-machine-learning.

V7 [online]. USA: BAHETI, Pragati, a.s., 2022 [2022-09-22]. Available from: https://www.v7labs.com/blog/overfitting.

List of figures and tables

List of figures

ANNOTATION

AUTHOR	**Dzhuraev Temur**		
FIELD	**Specialization International Supply Chain Management**		
THESIS TITLE	**Application of neural network edges for a possible embedding into supply chain management**		
SUPERVISOR	**Ing. Tomáš Malčic, Ph.D.**		
DEPARTMENT	**KRVLK - Department of Production, Logistics and Quality Management**	**YEAR**	**2023**

NUMBER OF PAGES	**56**
NUMBER OF PICTURES	**26**
NUMBER OF TABLES	**0**
NUMBER OF APPENDICES	**0**

SUMMARY	**The purpose of this thesis is application of neural network advantages for a possible embedding into supply chain management. There were presented different types of neural networks fulfilling its own tasks but the chapter 1 was delved more into RNN and LSTM structures which have been practically considered and elaborated in subsequent chapters. The second chapter went into details about factors which must be taken into consideration during the neural network elaboration such as myriad types of optimization algorithms, distinction between standardization and data normalization procedures, overfitting and underfitting issues, evaluation indicators. The third chapter is dedicated to direct presentation of practical application of neural network in time series dataset. The neural network model has been designed on python language in the Jupiter Notebook software. At the end, the problem has been defined and the evaluation has been carried out regarding the data set specification and its features.**
KEY WORDS	**LSTM, ML, Supply Chain Management, RNN, AI, Neural Network, Prediction, Optimization, CNN**