Forecasting Data Center Resource Usage With Artificial Neural Networks

Master's thesis in data science

Miika Malin
Research Unit of Mathematical Sciences
University of Oulu

Spring 2021

Tiivistelmä

Tämä työ esittelee koneoppimismenetelmiä sekä tilastollisia menetelmiä aikasarjojen ennustamiseen palvelinkeskus kontekstissa. Työn teoriosuus lähtee liikkelle aikasarjojen esikäsittelystä, jossa esitellään aikasarjojen differensointi sekä minimi-maksimi menetelmä aikasarjojen skaalaukselle. Tällä skaalausmenetelmällä saadaan eri skaalalla olevat aikasarjat vertailukelpoisiksi syötteiksi neuroverkolle.

Aikasarjojen esikäsittelyn jälkeen työ siirtyy aikasarjojen ennustamisen pariin. Mallintamismenetelmänä esitellään perinteinen tilastotieteen ARIMA -malli ja koneoppismenetelmänä kaksi erilaista takaisinkytkettyä neuroverkkoarkkitehtuuria, LSTM ja GRU. Ennen takaisinkytkettyjen neuroverkkojen esittelyä työ kertoo neuroverkkojen perusidean, ja millä tavalla neuroverkot oppivat. Lisäksi koneoppimispuolelta esitellään kuvantunnistuksesta tuttu konvoluutiokerrosta hyödyntävä neuroverkkoarkkitehtuuri muokattuna aikasarjoille sopivaksi.

Neuroverkkojen esittelyn jälkeen syvennetään neuroverkkojen ominaisuuksiin ja opetukseen liittyviä yksityiskohtia: Työssä esitellään kolme usein neuroverkoissa käytettyä epälineaarista aktivaatiofunktiota neuroverkon opetusvaiheessa tarvittavineen derivaattoineen. Tämän jälkeen työ esittelee kaksi optimointialgoritmia neuroverkon parametrien päivittämistä varten, ja yhden optimointialgoritmin jota käytetään ARIMA -mallin parametrien optimoimiseen. Lopuksi teoriaosuudessa esitellään erilaisia aikasarjoihin ja ennustevirheen suuruuteen liittyviä tunnuslukuja.

Teoriaosuuden jälkeen seuraa työn käytännön osuus. Tässä osuudessa käytetään ensin perättäishakua löytämään parhaat mahdolliset hyperparametrien arvot eri neuroverkkoarkkitehtuureille. Perättäishaulla saatuja tuloksia käytetään hyödyksi ennustaessa tulevaa oikean palvelinkeskuksen resurssin käyttöastetta kaikilla teoriaosuudessa esiteltyjen menetelmien avulla. Tämän jälkeen käytännön osuudessa vertaillaan eri mallien ennustetarkuutta, sekä mallien opetukseen kuluvaa aikaa.

Tärkeimpänä tuloksena työssä saatiin esille, että takaisinkytketty neuroverkkoarkkitehtuuri GRU konvoluutiokerroksella antoi tarkimmat ennusteet tulevalle palvelinkeskuksen resurssien tarpeelle, lyhentäen samalla mallin opettamiseen tarvittavaa aikaa.

Abstract

This thesis theoretical part presents some traditional time series forecasting methods (ARIMA) and recurrent neural network methods (LSTM and GRU) combined with convolution layer. The training process (backpropagation) of neural network is also explained in this thesis, and different algorithms to optimize the learning. Multiple metrics for evaluating forecast accuracy, and data preprocessing techniques are also introduced in theory section.

The practical side of this thesis focuses on predicting real-world resource usage data of data center. In the analysis section grid search for optimal hyperparameters of the models is performed. Based on the results found in hyperparameter optimization multiple different neural network architectures are compared with each other taking into account forecasting accuracy and the computational complexity of training the model.

The main result is that the recurrent neural network architecture GRU with convolution layer outperforms other models in forecast accuracy and in the time required to train the model. Proposed model can be effectively applied to load prediction as a part of data center computing cluster.

Contents

1 Introduction

In the field of data center studies common subject is data center energy consumption. This is an important subject since data center computing is growing and therefore generating more emissions (Andrae and Edler, 2015). In addition, more energy-efficient datacenters have lower operating costs since less energy is consumed.

One way to lower energy consumption of data centers is using a dynamic scalable clusters where the amount of powered on physical machines can be altered (Horvath and Skadron, 2008). These physical machines are later referred to as nodes. Reliable and accurate resource usage forecast would help to make these scalable clusters have the same level of quality of service which traditional datacenter has.

Methods introduced in this thesis methods section are used to forecast resource usage short-term utilizing neural networks and time-series modelling. Real-world data from Bitbrains data center is used in the practical side of this thesis.

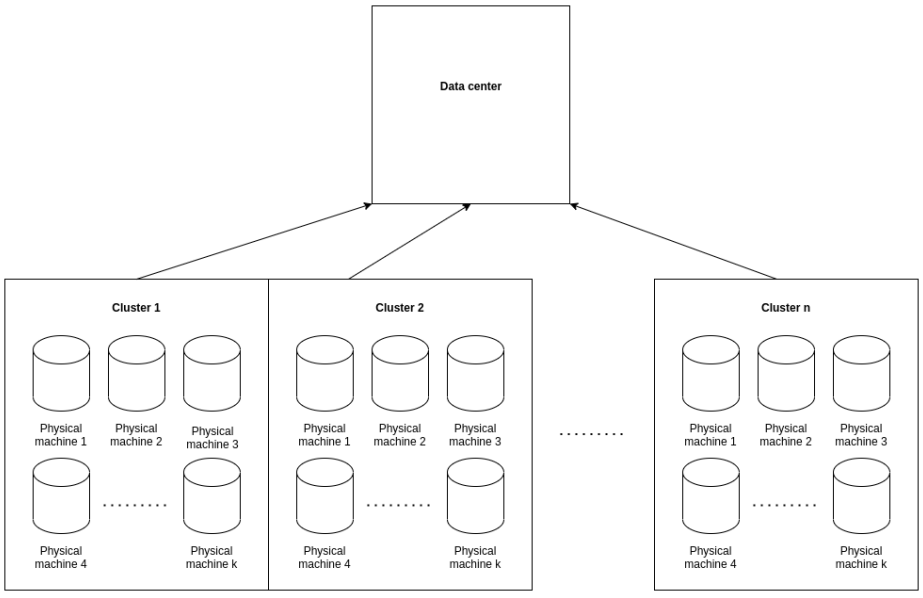

Figure 1: Data center hierarchy. Example data center has n clusters and each cluster has k nodes.

Reliable resource usage forecast has also many more applications in data center context. As seen in figure 1 nodes are often somehow clustered, and load balancing needs to be done in and between the clusters for data center to work efficiently. Resource usage forecast helps in this and many other controlling tasks inside data center.

Time series forecasting is also a interesting research subject since time series is the only data type involving time and are in this way unique. Machine learning advancement has bring many powerful tools to forecast these time series with more accuracy and less computational work required, and two state of the art recurrent neural networks are introduced in this thesis.

2 Methods

Time series are sequence of points $(y_1, y_2, y_3, ..., y_n)$ which is ordered by time (Sammut and Webb, 2011). This means that not only the observations $y_1, ..., y_n$ are important, but also the order in which these observations has been made. This requires use of special methods in both of data preprocessing and forecasting time series. The methods differ from traditional models or neural network architectures in a way that takes order of the sequence in account, and thus utilize the time aspect of data.

All the methods used in this thesis from data preprocessing to forecasting is introduced in this section.

2.1 Data Preprocessing

2.1.1 Differencing

Many time series forecasting methods require the time series to be stationary. Stationary time series looks the same on any time interval. This means that stationary time series cannot have any trends or seasonal patterns. (Hyndman and Athanasopoulos, 2019)

If the time series has a seasonal patterns it is not stationary. One way to make a seasonal time series stationary is to use seasonal differencing where observations are defined as:

$$y'_t = y_t - y_{t-m}, \tag{1}$$

where m is length of the season. This is called a "lag-m difference". Differenced time series will have m observations less than the original time series, because lag-m difference cannot be calculated for the first m observations.(Hyndman and Athanasopoulos, 2019)

A model, which uses only lag-m difference and error, is defined as:

$$y_t = y_{t-m} + \epsilon_t. \tag{2}$$

This is called the seasonal random walk model. The seasonal random walk model can be used to give naïve forecasts. (Hyndman and Athanasopoulos, 2019)

Sometimes second order differencing is needed for the time series to make it stationary. Second order differencing is defined as:

$$\begin{aligned} y''(t) &= y'_t - y'_{t-m} \\ &= y_t - y_{t-m} - (y_{t-m} - y_{t-2m}) \\ &= y_t - y_{t-m} - y_{t-m} + y_{t-2m} \\ &= y_t - 2y_{t-m} + y_{t-2m}, \end{aligned} \tag{3}$$

where y'_t is first order differenced time series defined in Equation 1. (Hyndman and Athanasopoulos, 2019)

2.1.2 Min-Max Normalization

Resource usage forecasting is performed on multivariate time series on the analysis section. This means that multiple features are used to predict a single resource usage. Because the features are recorded on different units, (e.g. network usage is recorded by kilobits per second (Kbps) and Central Processing Unit (cpu) usage is recorded by percentage), the scales of the features will also be different.

Different units will be weighted differently by the neural network, thus data needs to be transformed to eliminate the effect of different scales of features. The Min-Max normalization for the time series $Y = (y_1, y_2, ..., y_n)$ to range $[a, b]$ is defined as:

$$y_i^* = \frac{(y_i - \min(Y))(b-a)}{\max(Y) - \min(Y)} + a, \tag{4}$$

where y_i^* is the i-th normalized observation in the normalized timeseries $Y^* = (y_1^*, y_2^*, ..., y_n^*)$. (Han et al., 2012)

If forecast is performed on the normalized time series, the Min-Max normalization process needs to be inversed to get the forecast back to the original scale. The back transformation formula can be derived from Equation 4:

$$\begin{aligned} &(y_i^* - a)(\max(Y) - \min(Y)) = (y_i - \min(Y))(b-a) \\ \iff &y_i = \frac{(y_i^* - a)(max(Y) - \min(Y))}{b-a} + \min(Y). \end{aligned} \tag{5}$$

2.2 Resource Usage Forecasting

2.2.1 Autoregressive Model

Autoregressive (AR) model uses old observations of the time series to model the current timestep. Autoregressive model of order p is defined as:

$$y_t = \alpha + \phi_1 y_{t-1} + \phi_2 y_{t-2} + ... + \phi_p y_{t-p} + \epsilon_t, \tag{6}$$

where y_t is the time series value at time t, α is the intercept, $\phi_1, ..., \phi_p$ are the coefficients and ϵ_t is the error at timestep t. (Hyndman and Athanasopoulos, 2019)

2.2.2 Moving Average Model

Moving average (MA) model uses past errors to model the current timestep. Moving average model of order q is defined as:

$$y_t = \alpha + \theta_1 \epsilon_{t-1} + \theta_2 \epsilon_{t-2} + ... + \theta_q \epsilon_{t-q} + \epsilon_t, \tag{7}$$

where y_t is the time series value at time t, α is the intercept, $\theta_1, ..., \theta_k$ are the coefficients and ϵ_t is the error at timestep t. (Hyndman and Athanasopoulos, 2019)

2.2.3 Autoregressive Moving Average Model

When AR and MA models are combined, the result is Autoregressive Moving Average (ARMA) model, which uses both old observations of the time series and past forecasting errors. ARMA model of order (p, q) is defined as:

$$\begin{aligned} y_t &= \alpha + \phi_1 y_{t-1} + \phi_2 y_{t-2} + ... + \phi_p y_{t-p} + \theta_1 \epsilon_{t-1} + \theta_2 \epsilon_{t-2} + ... + \theta_q \epsilon_{t-q} + \epsilon_t \\ &= \alpha + \sum_{i=1}^{p} \phi_i y_{t-i} + \sum_{j=1}^{q} \theta_j \epsilon_{t-j} + \epsilon_t, \end{aligned} \tag{8}$$

where y_t is the time series value at time t, α is the intercept, $\theta_1, ..., \theta_q$ are the MA coefficients, $\phi 1, ..., \phi_p$ are the AR coefficients, and ϵ_t is the error at timestep t. (Adhikari and Agrawal, 2013)

2.2.4 AR Integrated Moving Average Model

Autoregressive Integrated Moving Average (ARIMA) model ARIMA model has parameters of (p, d, q), where p refers to the amount of used past values and q to the amount of used past errors as in ARMA model. The parameter d is the order of differencing performed.

Earlier models can be derived by using ARIMA and set some of the parameters to 0:

- ARIMA$(p, 0, 0)$ = AR(p)
- ARIMA$(0, 0, q)$ = MA(q)
- ARIMA$(p, 0, q)$ = ARMA(p, q)

Random walk model can also be defined as ARIMA(0, 1, 0):

$$y_t = y_{t-1} + \epsilon_t. \tag{9}$$

(Adhikari and Agrawal, 2013)

2.2.5 Neural Network

Neural network has its origins based on neurobiology result that the human brain computes in a different way than a computer does. The brain consists of a large amount of neurons, which allows it to make complex nonlinear computations. (Haykin, 2009)

An artificial neuron has been derived from these results to simulate a neuron of the brain. Neuron output is defined as:

$$y_k = \varphi(u_k + b_k) = \varphi(b_k + \sum_{i=1}^{m} w_{ki} x_i), \tag{10}$$

where y_k is the output of neuron k, φ is called an activation function and x_i is the ith input signal for the neuron and w_{ki} is its weight. Every neuron also has its own bias b_k. The job of an activation function is to regulate the output of the neuron to some limits. (Haykin, 2009)

Different activation functions (and corresponding derivates) used on this thesis are covered in Section 2.3.

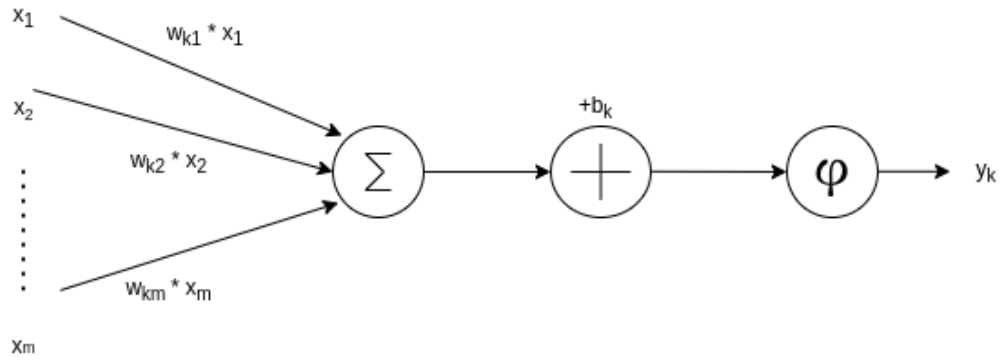

Figure 2: Neuron n_k with m inputs. Based on the figure in (Haykin, 2009) p. 41.

When using multiple of these neurons together the result is a neural network. The architecture of a neural network used in this section is a fully connected single-layer feedforward network, which is described in Figure 3.

As seen in Equation 10 and Figure 2 that the only way to change the output of neuron (when input stays the same) is to adjust weights w_{ki} and biases b_k. This process is called training the network.

To get the right direction and magnitude on how to change weights and biases, gradients of error function w.r.t. weights and biases need to be calculated. Let the error function for neuron k be defined as a half square error:

$$E_k = \frac{1}{2}(y_k - \hat{y_k})^2, \tag{11}$$

where y_k is the output of neuron k and $\hat{y_k}$ is the observed value. Let a_k be the linear combination of weights and inputs of neuron k such that:

$$a_k = \sum_i w_{ki} x_i. \tag{12}$$

By using the chain rule from Calculus the result is:

$$\nabla E_{kw} = \frac{\partial E_k}{\partial\ w_{ki}} = \frac{\partial E_k}{\partial y_k} \frac{\partial y_k}{\partial a_k} \frac{\partial a_k}{\partial w_{ki}}. \tag{13}$$

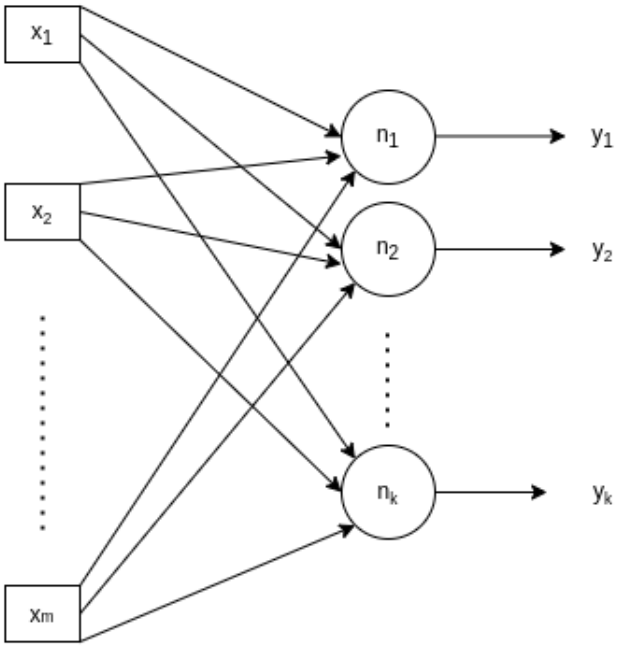

Figure 3: Fully connected single-layer feedforward neural network with m inputs, k neurons and outputs. Based on the figure in (Haykin, 2009) p. 51.

By adding Equation 10 to Equation 13 the result is:

$$\nabla E_{kw} = \frac{\partial E_k}{\partial y_k}\frac{\partial \varphi(a_k)}{\partial a_k}\frac{\partial a_k}{\partial w_{ki}} = (y_k - \hat{y_k})x_i\frac{\partial \varphi(a_k)}{\partial a_k}, \tag{14}$$

where $\frac{\partial \varphi(a_k)}{\partial a_k}$ is the derivate of the activation function used.

Since bias is actually just a weight with input of constant 1, the bias b_k can be updated with Equation 14 by having $x_i = 1$:

$$\nabla E_{kb} = (y_k - \hat{y_k})\frac{\partial \varphi(a_k)}{\partial a_k}. \tag{15}$$

(Bishop, 2006)

The process of calculating gradients of error function w.r.t. weights and biases is called backpropagation.

Different kind of optimization algorithms are used to update weights and biases based on the gradient, and they are covered in Section 2.4.

2.2.6 Convolutional Neural Network

Convolutional neural networks (CNN) are often used in neural networks dealing with image data. A convolutional layer in a neural network can be used to detect subsections of the image. (Bishop, 2006)

Convolution can be seen as sliding a window over the data. Because good results have been achieved with CNNs on image and natural language processing, CNNs have been recently applied with time series analysis as well. (Fawaz et al., 2019)

In CNNs processing image data convolution is two dimensional filter sliding over width and height of the image. In time series the only dimension that can be slided over is time, so one dimensional convolution is used (Fawaz et al., 2019). Example of 2D convolution can be seen in Figure 4.

Two dimensional convolution starting from point (i, j) (with indexing starting from (1,1)) can be defined as:

$$C(i,j) = \sum_{m=0}^{h} \sum_{n=0}^{w} I_{i+m,j+n} K_{1+m,1+n}, \tag{16}$$

where I is the input data, K is the kernel of the convolution with dimensions $h \times w$. Kernel contains weights w for the convolution. Here, h is the height, and w is the width of the convolution window. (Goodfellow et al., 2016)

Since the 1D convolution can only slide through one dimension, w is always same as number of features in input. Definition for 1D convolution can be derived from Equation 16:

$$C(i) = \sum_{m=0}^{h} \sum_{n=1}^{w} I_{i+m,n} K_{1+m,n}, \tag{17}$$

where i is the row of input where the convolution starts and w is the number of features in dataset. In time series context w can be seen as a number of features recorded in each time point. Again, in time series context this means

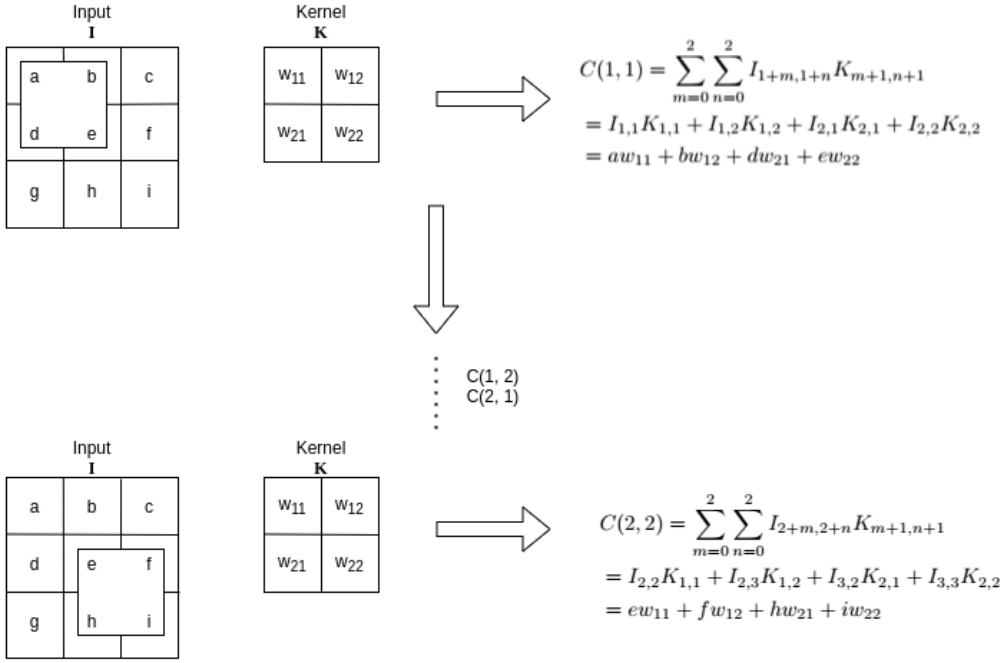

Figure 4: 2D convolution illustrated. Here Input data has dimensions of 3×3 and the kernel of convolution has 2×2.

that kernel is slided over the time dimension. This has been illustrated in Figure 5.

Easier way to implement convolution in code is to define it with element-wise Hadamard product $\odot$: Let w_i be the current convolution window, then:

$$C(i) = \sum w_i \odot K, \tag{18}$$

where K is the convolution kernel. Often bias is added to convolution result, and the result is sent through some activation function (Fawaz et al., 2019). In this case, Equation 18 becomes:

$$C(i) = f(b + \sum w_i \odot K), \tag{19}$$

where f is the activation function used, and b is the bias. This has been illustrated (without activation function) in Figure 5.

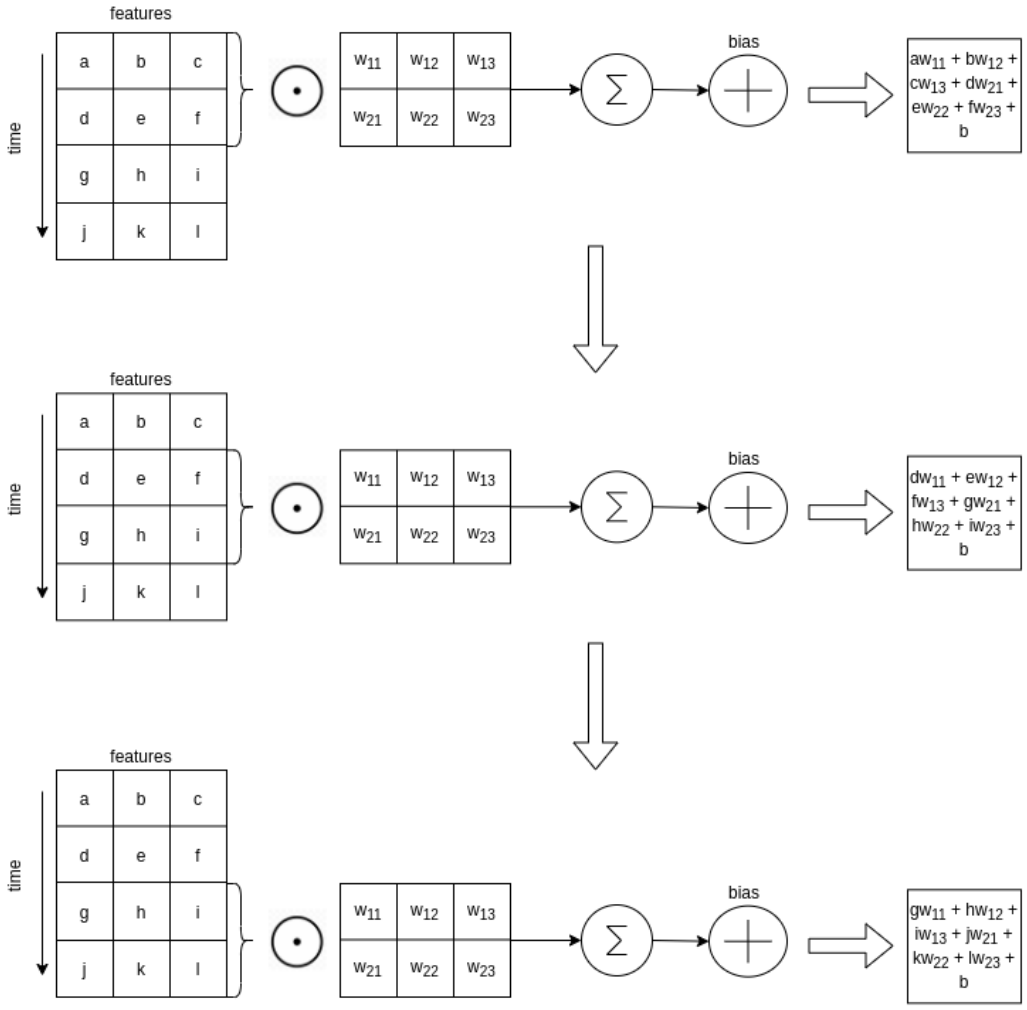

Figure 5: 1D convolution where convolution slides through time illustrated. Here the time series input has three features and five time points. The kernel of the convolution has been defined with length of two. This means that kernel has dimensions of 2×3.

2.2.7 Recurrent Neural Network

Recurrent neural network (RNN) is a combination of units with feedback connection (Hewamalage, 2020) and are designed specially for sequential data such as time series (Aggarwal, 2018).

Recurrent units in recurrent neural network pass data from previous timesteps to proceeding units in its hidden state h as illustrated in Figure 6.

Consecutive values in time series are often correlated. If these values were fed as unattached into regular neural network, information would be lost about the sequence. Recurrent neural networks can utilize the sequence information. (Aggarwal, 2018)

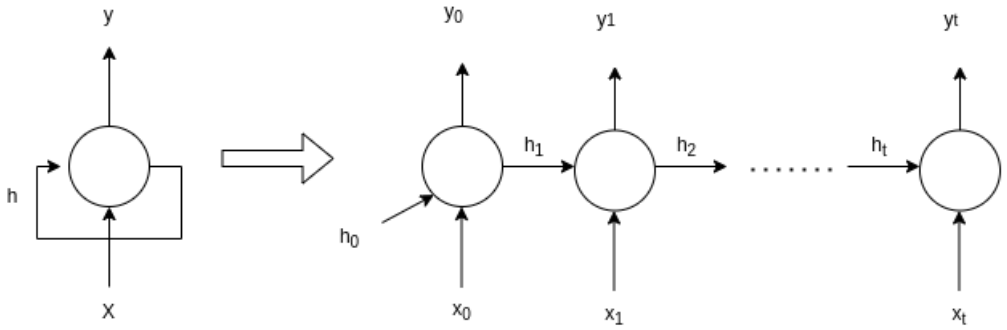

Figure 6: Unfolded recurrent neural network: x stands for input vectors, h is hidden state and y is output vector of RNN. Based on the figure in (Olah, 2015).

Inputs for the recurrent neural network are vectors with d -dimensions (Aggarwal, 2018). The example RNN in Figure 6 has t d-dimensional vectors.

The backpropagation of RNN is called backpropagation through time since it is performed for the unfolded RNN. This process takes into account all the hidden states of network.

Downside of long chains in gradient calculations is that the gradient for weights far away from the final output closes to zero. Because the gradient is near zero, the far away weights will only be updated by a small value and the learning becomes slow. This is called the vanishing gradient problem and is the reason why traditional RNN cannot learn long term depencies. (Aggarwal, 2018; Haykin, 2009)

One of the most basic RNN architecture uses Elman recurrent units, where every unit passes its hidden state h_t at timestep t to next unit handling

timestep $t+1$. The hidden state of unit at timestep t is defined as:

$$h_t(x_t, h_{t-1}, W_h, W_x, b_h) = S(W_h h_{t-1} + W_x x_t + b_h), \tag{20}$$

and the output of Elman recurrent unit at timestep t is then defined as:

$$y_t(x_t, h_{t-1}, W_h, W_x, b_h, W_y, b_y) = T(W_y h_t(x_t, h_{t-1}, W_h, W_x, b_h) + b_y), \tag{21}$$

(Hewamalage, 2020). In Equations 20 and 21 W_h are weights for hidden state with dimension of $d \times d$, W_x are weights for input vector with dimension of $d \times f$. Hidden state h_t and biases b are vectors with dimension of d. Here d is the length of the hidden state, f f is the number of features in time series.

2.2.8 Long Short-Term Memory

Long Short-Term Memory (LSTM) is a recurrent neural network, which uses LSTM units and tries to solve vanishing gradient problem of RNNs (Hochreiter and Schmidhuber, 1997). This means that the arthitecture can find more efficiently long term depencies from time series.

LSTM network with forget gate and biases consists of LSTM cells, where each cell has three gates:

- Forget gate

$$f(h, x, w_{hf}, w_{xf}, b_f) = S((w_{hf} h + w_{xf} x) + b_f). \tag{22}$$

- Input Gate

$$\begin{aligned} &i(h, x, w_{his}, w_{xis}, w_{hit}, w_{xit}, b_{is}, b_{it}) = \\ &S(h, x, w_{his}, w_{xis} + b_{is}) \odot T(h, x, w_{hit}, w_{xit}, b_{it}) = \\ &S((w_{his} h + w_{xis} x) + b_{is}) \odot T((w_{hit} h + w_{xit} x) + b_{it}). \end{aligned} \tag{23}$$

- Output gate

$$o(h, x, c, w_{ho}, w_{xo}, b_o) = S((w_{ho} h + w_{xo} x) + b_o) \odot T(c)), \tag{24}$$

where S and T are activation functions. Forward pass of the cell can then be defined by the following functions:

$$\begin{aligned} &c_t = \\ &(c_{t-1} \odot f(h_{t-1}, x_t, w_{hf}, w_{xf}, b_f))+ \\ &i(h_{t-1}, x_t, w_{his}, w_{xis}, w_{hit}, w_{xit}, b_{is}, b_{it}). \end{aligned} \tag{25}$$

$$o_t/h_t = o(h_{t-1}, x_t, c_t, w_{ho}, w_{xo}, b_o). \tag{26}$$

Again in Equations 22 - 26 w_h are weights for hidden state with dimension of $d \times d$, w_x are weights for input vector with dimension of $d \times f$. Hidden state h_t and biases b are vectors with dimension of d. Here d is the length of the hidden state, f f is the number of features in time series and $\odot$ is an element-wise Hadamard product.

(Gers et al., 2000; Hewamalage, 2020)

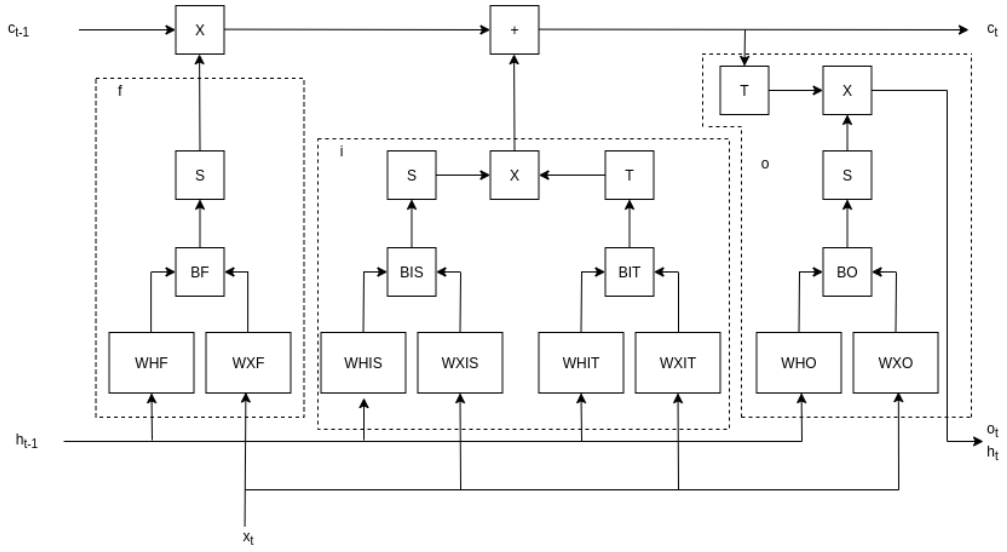

Figure 7: LSTM cell with forward pass of data at timestep t. Based on the figure in (Condit, 2019).

As seen in Figure 7 each LSTM cell indeed has three gates where the data passes, and two different activation functions. These activation functions are defined in this work as S=sigmoid() and T=tanh() functions.

2.2.9 Gated Recurrent Unit

Gated recurrent unit (GRU) has been motivated by LSTM unit, but it has a simpler design. GRU unit does not have an output gate what LSTM has, so it is faster to implement and train. Since it has less gates, it also has less weights to optimize and the backpropagation through time is faster.

GRU network consists of multiple GRU cells, where each cell has two gates:

- Reset gate

$$r(h, x, w_{xr}, w_{hr}, b_r) = S((w_{xr}x + w_{hr}h) + b_r). \tag{27}$$

- Update gate

$$u(h, x, w_{xu}, w_{hu}, b_u) = S((w_{xu}x + w_{hu}h) + b_u). \tag{28}$$

By using these two gates in the forward pass a new candidate hidden state is defined as $\tilde{h}$:

$$\begin{aligned} &\tilde{h}(h_{t-1}, x, w_{xh}, w_{hh}, b_h) = \\ &T(w_{xh}x + w_{hh}(r(h_{t-1}, x, w_{xr}, w_{hr}, b_r) \odot h_{t-1}) + b_h). \end{aligned} \tag{29}$$

and finally the new cell hidden state / output h_t:

$$\begin{aligned} &h_t(h_{t-1}, x, w_{xr}, w_{hr}, w_{xu}, w_{hu}, b_r, b_u) = \\ &(1 - u(h_{t-1}, x, w_{xu}, w_{hu}, b_u)) \odot h_{t-1} + \\ &u(h_{t-1}, x, w_{xu}, w_{hu}, b_u) \odot \tilde{h}(h_{t-1}, x, w_{xh}, w_{hh}, b_h), \end{aligned} \tag{30}$$

where T=tanh and S=sigmoid.

All w_{xx} are weights, and all b_x are biases, which are the learnable parameters. Here $\odot$ is element-wise Hadamard product. GRU has the same dimension on weights, biases and hidden state as LSTM does. (Hewamalage, 2020)

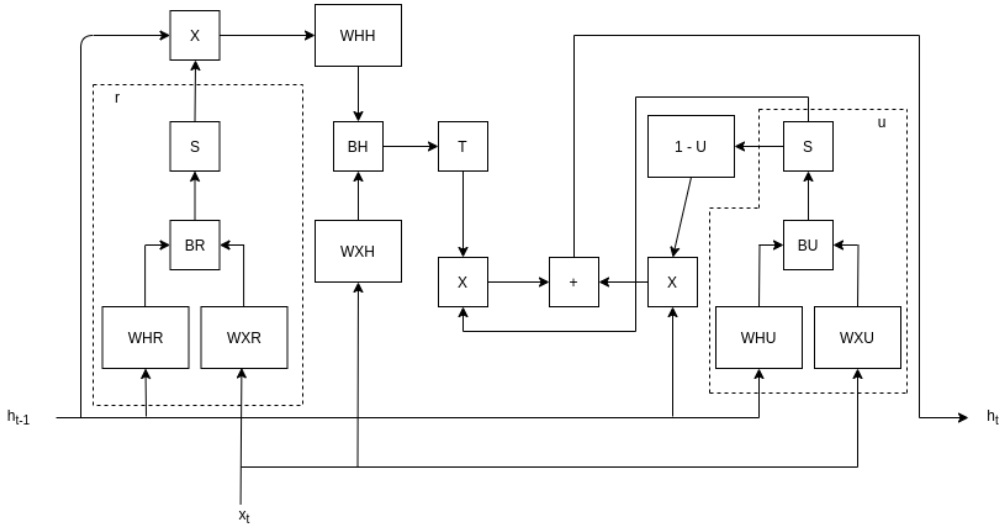

Figure 8: GRU cell with forward pass of data at timestep t.

2.3 Activation Functions

Activation functions are a important part of neural network since those allow nonlinearity to output of a neuron. Without activation functions the output of neurons would only be linear combination of its inputs. Other important reason to use activation functions is that activation function can regulate the scale of output. (Feng and Lu, 2019)

Regulated outputs is an important feature to have, since it also regulates the derivates on backpropagation. Three different activation functions are used in this thesis and those are defined in this section and a graph of those functions (with respecting derivates) is show in Figure 9.

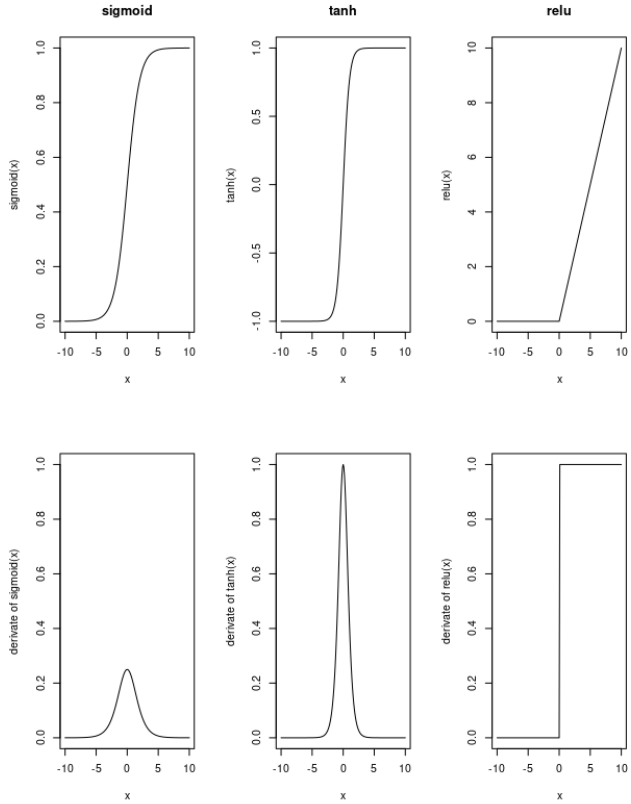

Figure 9: Curves of Logistic Sigmoid Function, Hyperbolic Tangent Function and Rectified Linear Unit. In the first row of plots Y-axis has the output of the function when input is x. In the second row of plots Y-axis has the value of the derivate function at point x.

2.3.1 Logistic Sigmoid Function

Logistic sigmoid function is presented as:

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

which codomain is [0, 1]. (Feng and Lu, 2019)

Derivate of sigmoid function is needed in backpropagation step, and it can be deduced from Equation 31 as follows:

$$\frac{dS(x)}{dx} = \frac{d}{dx}\frac{1}{1 + e^{-x}} = \frac{d}{dx}(1 + e^{-x})^{-1}. \tag{32}$$

Now let $g = 1 + e^{-x}$ so $S(x) = g^{-1}$. The chain rule can now be applied to Equation 32 so it becomes:

$$\begin{aligned} \frac{dS(x)}{dx} &= \frac{dS(x)}{dg}\frac{dg}{dx} = (-g^{-2})(-e^{-x}) = \frac{-e^{-x}}{-g^2} = \frac{e^{-x}}{(1 + e^{-x})^2} \\ &= (\frac{1}{1 + e^{-x}})(\frac{e^{-x}}{1 + e^{-x}}) = S(x)(\frac{e^{-x} + 1 - 1}{1 + e^{-x}}) \\ &= S(x)(\frac{1 + e^{-x}}{1 + e^{-x}} - \frac{1}{1 + e^{-x}}) = S(x)(1 - S(x)). \end{aligned} \tag{33}$$

2.3.2 Hyperbolic Tangent Function

Hyperbolic tangent function (tanh) is presented as:

$$T = \tanh(x) = \frac{e^{x} - e^{-x}}{e^{x} + e^{-x}}, \tag{34}$$

which codomain is [-1, 1]. (Feng and Lu, 2019)

Tanh is often defined by its simpler form by modifiying Equation 34:

$$\frac{e^{x} - e^{-x}}{e^{x} + e^{-}x} = \frac{\frac{e^{2x}}{e^{x}} - \frac{1}{e^{x}}}{\frac{e^{2x}}{e^{x}} + \frac{1}{e^{x}}} = \frac{\frac{e^{2x}-1}{e^{x}}}{\frac{e^{2x}+1}{e^{x}}} = \frac{e^{x}(e^{2x} - 1)}{e^{x}(e^{2x} + 1)} = \frac{e^{2x} - 1}{e^{2x} + 1}, \tag{35}$$

which is also faster to compute, since it contains only two exponential functions versus four exponential functions in Equation 34.

Derivate of tanh function is needed in backpropagation step and it can be deduced from Equation 34 by using the quotient rule:

Let $g(x) = e^x - e^{-x}$ and $h(x) = e^x + e^{-x}$. Now $tanh(x) = \frac{g(x)}{h(x)}$ and the quotient rule can be applied to Equation 34 to get

$$\begin{aligned}
\frac{d}{dx}tanh(x) &= \frac{g'(x)h(x) - g(x)h'(x)}{h(x)^2} \\
&= \frac{(e^x + e^{-x})(e^x + e^{-x}) - (e^x - e^{-x})(e^x - e^{-x})}{(e^x + e^{-x})^2} \\
&= \frac{(e^x + e^{-x})^2 - (e^x - e^{-x})^2}{(e^x + e^{-x})^2} \\
&= \frac{(e^x + e^{-x})^2}{(e^x + e^{-x})^2} - \frac{(e^x - e^{-x})^2}{(e^x + e^{-x})^2} = 1 - tanh(x)^2.
\end{aligned} \tag{36}$$

2.3.3 Rectified Linear Unit

Rectified linear unit (ReLU) is presented as:

$$R = R(x) = max(x, 0), \tag{37}$$

which codomain is [0, ∞[.

Derivate of ReLU is simply:

$$R'(x) = \begin{cases} 0, if x < 0 \\ 1, if x > 0 \end{cases} \tag{38}$$

(Feng and Lu, 2019)

It can be seen from Equation 38 that the derivate is not defined at x=0, because the right and the left derivatives differs. However this is not a problem at the backpropagation step since we can apply convention eg. that the derivate is 0 at x=0.

2.4 Optimizers

This section introduces an algorithm to optimize ARIMA model parameters p, d, q, and two optimizers for updating neural network weights with precalculated gradients.

There are many ways to update the weights of a neural network, and each of them affects the speed of convergence in learning or ability to get over from global minimum (Nawi et al., 2015). Because of this multiple different algorithms has been proposed to optimize the neural network weights, and two of them (Adam and RMSprop) is introduced in this section.

2.4.1 ARIMA Optimization Algorithm

The following step-wise Algorithm 1 was proposed by Hyndman and Khandakar (2008) is used to find optimal parameters p, d, q for ARIMA model given the time series to forecast.

Algorithm 1: Hyndman and Khandakar step-wise algorithm

Input: max_d, *timeseries*

Output: Optimized paramaters p, d, q for fitting ARIMA model to data

1 Find the optimal number of differences (d) in between $[0, max_d]$ by using repeated KPSS tests on *timeseries* and select d which yielded the highest p-value.
2 Difference the *timeseries* d-times.
3 Try first 4 possible models for the time series:
- ARIMA(0, d, 0)
- ARIMA(0, d, 1)
- ARIMA(1, d, 0)
- ARIMA(2, d, 2)

the model which gives lowest AIC is selected as $model_c$
4 Test models with $\pm$ 1 for p and q of parameters $model_c$.
Test models with and without the constant
Update model with lowest AIC to be $model_c$
5 Go back to Step 4, if the new $model_c$ was found
6 Return $model_c$ parameters

(Hyndman and Khandakar, 2008; Hyndman and Athanasopoulos, 2019)

Value for the parameter max_d suggested is 2. (Hyndman and Athanasopoulos, 2019)

The main idea of Algorithm 1 is to first find the optimal number of differences for given timeseries, by checking the differenced time series with KPSS test. After this the algorithm repeatedly try values for parameters p and q, until the AIC does not get lower. AIC is defined in section 2.5.1 and KPSS in section 2.5.2.

2.4.2 RMSprop Neural Network Optimizer

RMSprop is a commonly used, but unpublished, optimization algorithm for optimizing weights and biases of a neural network. It was introduced by Hinton in Coursera online course "Neural Networks for Machine Learning" (2012).

Algorithm 2: RMSprop optimization algorithm

Input: Gradient for the optimizable parameter: ∇E_{kp}, Moving average parameter: β, Learning rate: lr, Mean square from last time: MS_{t-1}, The parameter to optimize p_{t-1}

Output: New parameter where gradient has been applied with RMSprop: p_t, new mean square: MS_t

1 Calculate moving average for squared gradient:
$MS_t = \beta * MS_{t-1} + (1 - \beta) * \nabla E_{kp}^2$

2 Update parameter:
$p_t = p_{t-1} - lr * \frac{\nabla E_{kp}}{\sqrt{MS_t}}$

3 return p_t, MS_t

Value for the parameter β suggested in the original presentation of RMSprop is 0.9. (Tieleman and Hinton, 2012)

The main idea of Algorithm 2 is to calculate moving average for squared gradient, and normalize the original gradient with that. This helps with exploding and vanishing gradients, since MS_t becomes greater when the original original gradient ∇E_{kp} is big, and smaller when the ∇E_{kp} is small. The magnitude of parameter update is divided with MS_t, so it makes updates for big gradients smaller and for small gradients greater.

2.4.3 Adam Neural Network Optimizer

Adam is a neural network optimizer based on RMSprop and AdaGrad optimizers.

Algorithm 3: Adam optimization algorithm

Input: Gradient for the optimizable parameter: ∇E_{kp}, Moving average parameters: (β_1, β_2), Learning rate: lr, Moving averages from last time: ($MA1_{t-1}$, $MA2_{t-1}$), The parameter to optimize p_{t-1}, Number of parameter updates so far: i, Epsilon: ϵ

Output: New parameter where gradient has been applied with Adam: p_t, new Moving Averages: ($MA1_t$, $MA2_t$), Number of parameter updates done: i

1 Increment i by one

2 Calculate moving averages for gradient:
$MA1_t = \beta_1 * MA1_{t-1} + (1 - \beta_1) * \nabla E_{kp}$
$MA2_t = \beta_2 * MA2_{t-1} + (1 - \beta_2) * \nabla E_{kp}^2$

3 Calculate bias correction for moving averages:
$\widehat{MA1}_t = \frac{MA1_t}{1-\beta_1^i}$
$\widehat{MA2}_t = \frac{MA2_t}{1-\beta_2^i}$

4 Update parameter:
$p_t = p_{t-1} - lr * \frac{\widehat{MA1}_t}{\sqrt{\widehat{MA2}_t}+\epsilon}$

5 return p_t, ($MA1_t$, $MA2_t$), i

In Algorithm 3 i is initialized as zero. The original paper introducing Adam suggests to use $lr = 0.001$, $\beta_1 = 0.9$, $\beta_2 = 0.999$ and $\epsilon = 10^{-8}$. (Kingma and Ba, 2015)

The main idea in Algorithm 3 is to calculate moving averages for gradient and squared gradient. However these moving averages are biased towards 0, since $MA1_0$ and $MA2_0$ are initialized as 0. To get rid of this bias, bias correction is applied and bias corrected moving averages are used to update the optimizable parameter.

2.5 Metrics

This section introduces different metrics used in the thesis. These include metrics used in model comparison to metrics evuluating time series stationarity.

2.5.1 Akaike Information Criterion

Akaike Information Criterion (AIC) is used on Algorithm 1 to compare different candidate models.

AIC is defined as:

$$-2 \log(\hat{L}) + 2p, \tag{39}$$

where $\hat{L}$ is maximum likelihood of the model, and p is number of parameters in model (Akaike, 1974).

Equation 39 shows that AIC does not take into account sample size therefore models with different sample size can not be compared using AIC. Lower AIC values can be considered as a better model.

2.5.2 Kwiatkowski–Phillips–Schmidt–Shin Test

Kwiatkowski–Phillips–Schmidt–Shin (KPSS) test performs a null hypothesis test where H_0 = "time series is stationary". This is done by decomposing a time series at timestep t to the deterministic trend, random walk and the stationary error:

$$y_t = \delta + r_t + \epsilon_t, \tag{40}$$

where r_t is random walk:

$$r_t = r_{t-1} + u_t, \tag{41}$$

where u is independent and idetically distributed (i.i.d) $N(0, \sigma_u^2)$. The null hypothesis which is equivalent for "time series is stationary" is now $\sigma_u^2 = 0$. If $\delta = 0$ in equation 40 the null hypothesis is for level stationarity and if $\delta \neq 0$ null hypothesis is for trend stationarity. (Kwiatkowski et al., 1992)

2.5.3 Mean Squared Error

Different hyperparameters are compared to each other by calculating Mean Squared Error (MSE) for the models forecasts for a given hyperparameter. MSE is defined as:

$$\frac{1}{T}\sum_{i=1}^{T}(\hat{y}_i - y_i)^2, \tag{42}$$

where T is the number of forecasted points (time points), $\hat{y}_t$ is forecast at time point t and y_t is real value of time series at time t. (Sammut and Webb, 2011)

2.5.4 Root Mean Square Error

Forecasts of different models (ARIMA and RNNs) are evaluated with Root Mean Square Error (RMSE), which is defined as:

$$\sqrt{\frac{1}{T}\sum_{i=1}^{T}(\hat{y}_i - y_i)^2}, \tag{43}$$

where T is the number of forecasted points (time points), $\hat{y}_t$ is forecast at time point t and y_t is real value of time series at time t. (Hyndman and Athanasopoulos, 2019)

2.5.5 Interquartile range

Interquartile range (IQR) is a statistic for deviance. It is defined as

$$Q_3 - Q_1, \tag{44}$$

where Q_3 and Q_1 are the upper and lower quartiles of data. This statistic tells on how wide range 50 % of the observations are distributed.

Boxplot is closely related to IQR since its left wall is drawn on lower quartile and right wall to upper quartile. After this, the median is drawn as a line inside the box. Whiskers are drawn from left and right walls, and the whisker

length is 1.5 times the IQR. Observations past this line are considered as outliers and marked with x.

(Upton and Cook, 1996)

2.5.6 Autocorrelation

Autocorrelation is the correlation between a time series and its lagged version. Because of this, it can be used to find repeating patterns in a time series. Autocorrelation metric r_k has its values in within range [-1, 1] where negative values imply negative correlation and positive values imply positive correlation. The higher the value that $|r_k|$ has, the stronger the correlation is. Autocorrelation can be used as a tool to choose which features to train a neural network on. (Xue et al., 2015)

Because autocorrelation tells the correlation with time series itself in different lags, it can also be used to find an optimal lag value for differencing the time series to remove the seasonal term.

Let Y be a time series with values $Y = (y_1, y_2, ..., y_n)$. Then autocorrelation coefficients r_k are defined as:

$$r_k = \frac{\sum_{t=k+1}^{n} (y_t - \overline{y})(y_{t-k} - \overline{y})}{\sum_{t=1}^{n} (y_t - \overline{y})^2}, \tag{45}$$

where n is the number of observations in time series. (Hyndman and Athanasopoulos, 2019)

3 Data And Experiments

This section consists of the practical side of thesis. The objective of the analysis is to forecast Central Processing Unit (CPU) resource usage from real world dataset called GWA-T-12 Bitbrains. (Bitbrains, 2013)

3.1 Dataset Summary

The dataset contains resource usage traces of 1750 Virtual Machines (VM) from Bitbrains distributed datacenter. The usage trace length is 1 month: from August 12, 2013 to September 11, 2013. The dataset has samples in 5 minute intervals. Bitbrains has customers from many industries so the use cases and usage traces are very different from VM to VM.

Dataset consists of two different sets. The first set contains data for 1250 VMs in fast Storage Area Network (SAN) and second set contains data for 500 VMs from faster and slower Network Attached Storage (NAS) devices. Only the traces for VMs in fast storage area network are used in this thesis.

Figure 10 has all the used variables plotted for machine with id 21 as an example. All the features (with descriptions) used in analysis are defined in Table 1.

Name	**Description**	**Unit**
Timestamp	Number of milliseconds since 1970-01-01	ms
cpu_usage	Cpu usage	%
memory_usage	Memory usage	%
disk_read	Disk read throughput	KB/s
disk_write	Disk write throughput	KB/s
net_receive	Network received throughput	KB/s
net_transmit	Network transmitted throughput	KB/s

Table 1: Dataset variables

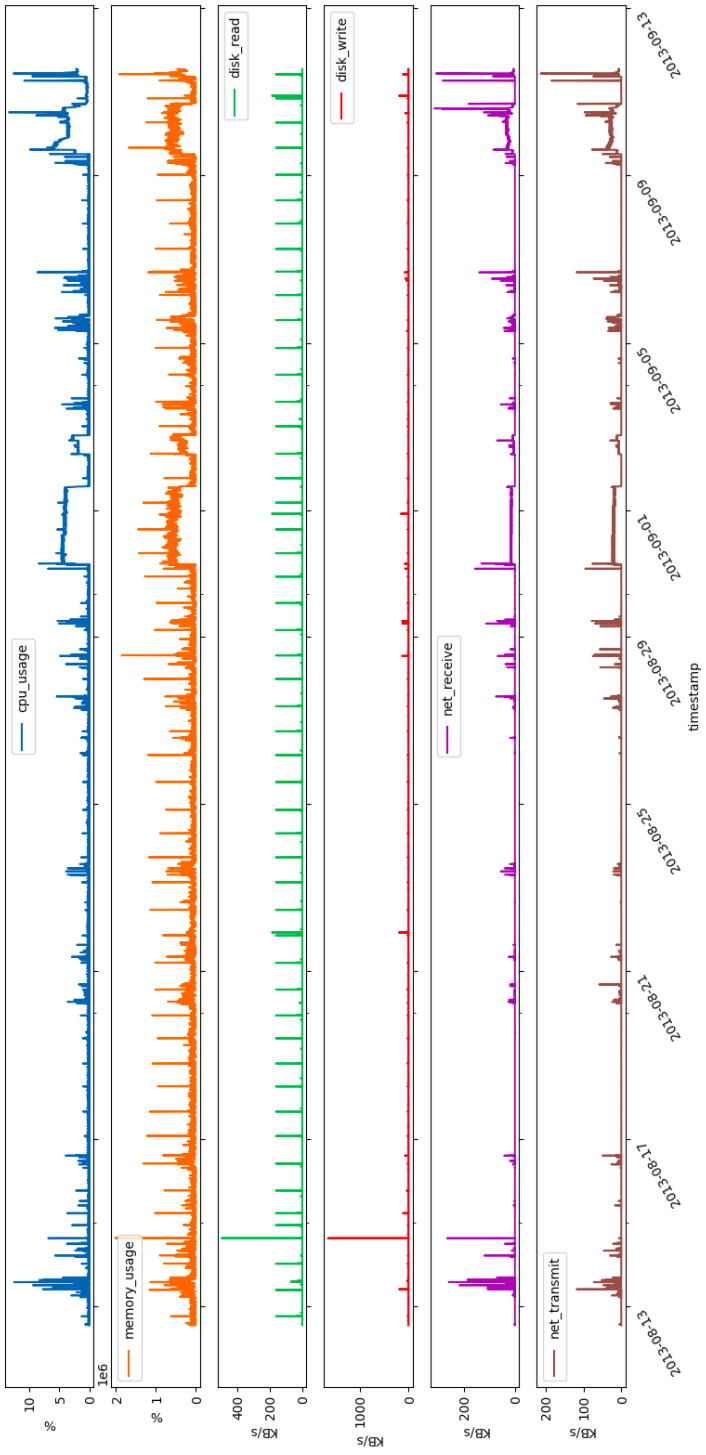

Figure 10: Data for machine id 21.

3.2 Neural Network Architecture

Two different recurrent neural network architectures were used on the dataset: GRU and LSTM. Both architectures were also used with and without preceeding convolutional layer. Neural network architecture is shown in Figure 11.

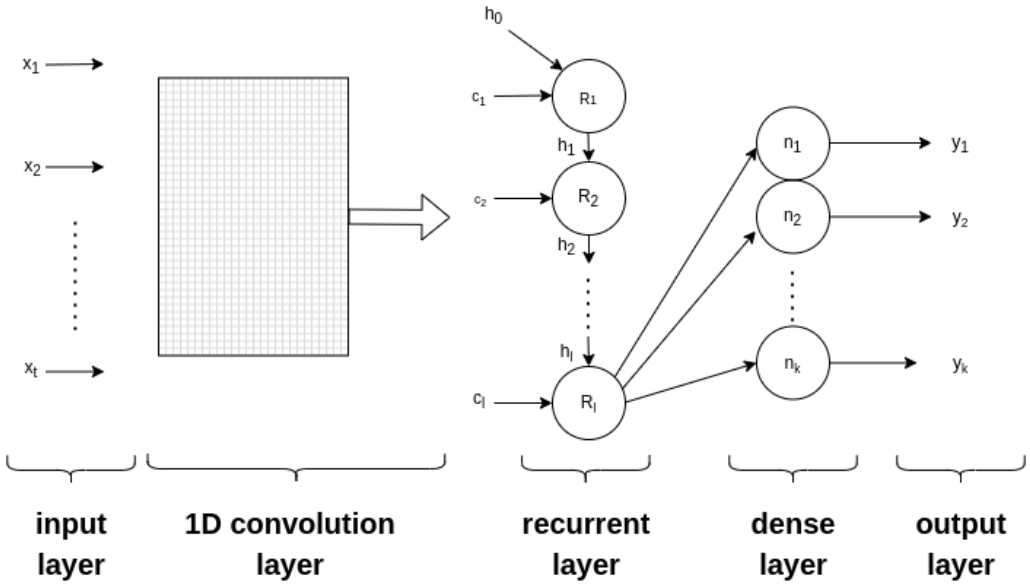

Figure 11: Model architecture with convolution layer. This architecture has t timesteps as input and k timesteps as output.

The input for a neural network had all the 6 features of the dataset (cpu_usage, memory_usage, disk_read, disk_write, net_receive and net_transmit). Different length of input sequences were tried on the Section 3.3. The length of the input layer affects its dimensions: if input sequence with t timesteps is used, the input layer has dimension of (t, 6) where the latter is number of features in the dataset.

Output dimension of convolution layer depends on the dimension of input layer and two hyperparameters of convolution layer: Number of filters (c_f) used in the convolution and size of the kernel (c_k). Number of filters are just number of kernels sliding through the dataset (on time axis). Size of the kernel is number of timesteps taken into account on one sliding kernel when convolution is calculated. Output dimension of convolution layer is then (c_l,

c_f), where $c_l = t - c_k - 1$.

Every kernel on convolution layer is initialized by drawing random numbers from U($-\sqrt{\frac{6}{t+c_h}}$, $\sqrt{\frac{6}{t+c_h}}$) distribution, which means that every kernel has a different starting weights. This initialization allows each kernel to find different patterns from data and is called Xavier (or Glorot) uniform initialization (Glorot and Bengio, 2010).

Recurrent layer dimension depends on the convolution layer output dimensions, if convolution layer is used on the architecture. If architecture has no convolutional layer, dimension of recurrent layer depends from shape of the dataset (input layer). Length of the hidden state (l_h) affects the output dimension of recurrent layer, and is a tunable hyperparameter. Hidden state (with legnth l_h) of last RNN cell is sent to next layer on the architecture which is dense layer. Weights on the recurrent layer are initialized with Xavier uniform initialization as in the convolution layer. If convolution layer is used this means that the weights are drawn from U($-\sqrt{\frac{6}{c_h+l_h}}$, $\sqrt{\frac{6}{c_h+l_h}}$) distribution. If the input layer is directly feeded into recurrent layer the weights are drawn from U($-\sqrt{\frac{6}{t+l_h}}$, $\sqrt{\frac{6}{t+l_h}}$) distribution.

Dimension of the dense layer needs to be same as the output layer, which is 6 on this application. Weights of the dense layer are initialized with Xavier uniform initialization, which mean that all the weights are drawn from U($-\sqrt{\frac{6}{l_h+6}}$, $\sqrt{\frac{6}{l_h+6}}$) distribution.

Dimension of the output layer was 6, which equals forecast for next 30 minutes of CPU usage.

All biases on all layers are initialized as 0.

3.3 Grid Search for Tuning Hyperparameter Selection

The model architecture shown in Figure 11 contains multiple tunable hyperparameters. These hyperparameters are listed in Table 2. Performing grid search on these hyperparameters means, that some finite set of hyperparameter values were manually chosen (based on prior trial and error on the architecture), and performance for each set of hyperparameters was evaluated on the validation set. The search was done with brute-force (exhaustive) search so that every possible combination of hyperparameters was evaluated.

Grid search for models hyperparameters was done by training different models on data from same machine (machine id 21) which can be seen in Figure 10. This machine was selected for hyperparameter training data because it had high autocorrelation values with long lag. Network transmit, CPU and memory usage had its highest autocorrelation (when lag was higher than 400) with lag of 2845 which is equivavalent with 9.88 days. Also the network receive had its highest autocorrelation with lag 2847 which is very close to 2845. All the autocorrelation results for machine id 21 can be seen in Table 3 and autocorrelation plot can be seen in Figure 12.

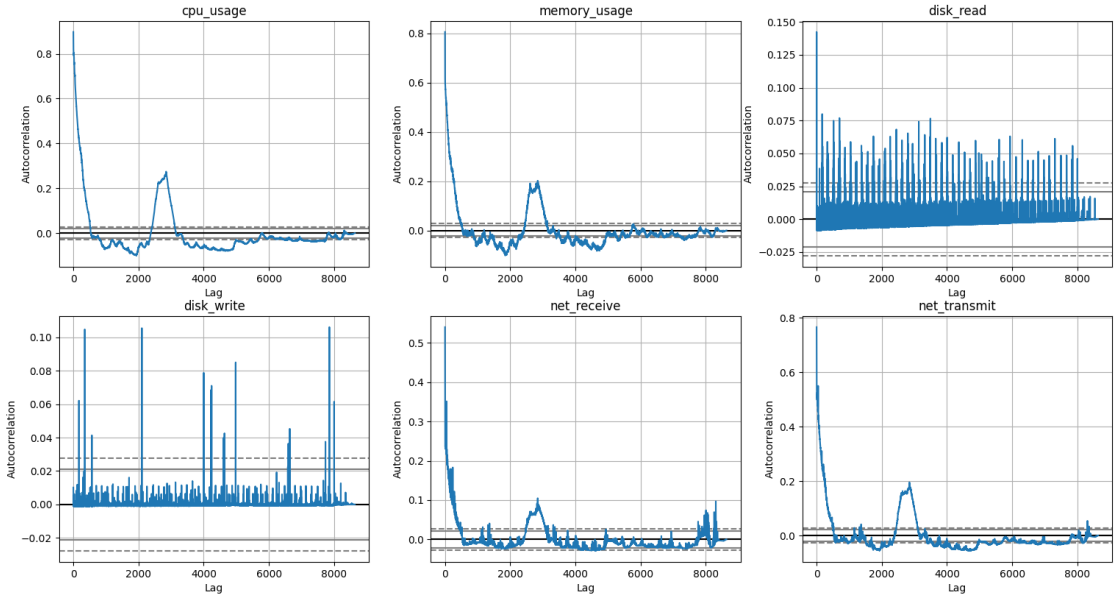

Figure 12: Autocorrelation plot for all the features of machine id 21.

As said in the Section 3.2 both RNN architecures (LSTM and GRU) were used, with and without 1D convolution layer. On top of that architectures containing 1D convolution layer were tried with and without activation function in output layer. This makes 6 different architectures in total.

Models were trained on first 70% of the data for 50 epochs for each set of hyperparameters. One epoch means, that the whole training dataset is iterated through the neural network training process. Remaining 30% of machine id 21 data was used as a validation set. After each epoch loss was calculated on both training and validation data sets with MSE. After 50

epochs of training minimum of losses in both datasets were recorded.

Models including 1D convolution layer had 4 tunable hyperparameters:

1. Number of filters in 1D convolution layer
2. Kernel size
3. Length of the hidden state in recurrent layer
4. Length of history data inputted to model

and models without 1D convolution layer had learning rate and optimizer as tunable hyperparameters instead of hyperparameters relating to convolution (1 and 2). On convolutional models optimizer was fixed to Adam with learning rate of 0.00012. Models including activation function in output layer had it fixed as ReLU.

Total of 985 different hyperparameter combination for 6 models were tried. When ordering results from loss in validation set all first 13 models were with GRU architecture without convolution layer. First architecture with convolution used was on place 105 , it had GRU used as recurrent layer and had activation function used in output layer. Best model with LSTM and 1D convolution was on place 311 and it had also activation function used in output layer.

Best 20 models sorted by loss (MSE) on validation set are shown in Table 4. Boxplot for loss in both datasets for every hyperparameter tuned is shown in Figure 13 as example. When fitting models to larger dataset in Section 3.4 the parameters were chosen based on the grid search results of each model. This means that both boxplot of hyperparameters and the best combinations of hyperparameters were taken into acccount when training with larger data.

Layer	Hyperparameters
Input	Number of input time points
Convolution	Kernel size, number of filters
Recurrent	Hidden state length
Other	Optimizer, learning rate

Table 2: Tunable hyperparameters in grid search.

Feature	Autocorrelation	Lag
CPU usage	0.275	2845
Memory usage	0.202	2845
Disk read	0.077	703
Disk write	0.106	2102
Net receive	0.105	2845
Net transmit	0.195	2847

Table 3: Machine id 21 maximum autocorrelations for all fetures with lag greater than 400.

Optimizer	Learning rate	Hidden state length	Input length	Minimum loss (training set)	Minimum loss (validation set)	RNN architecture
RMSProp	0.00012	1024	360	0.182	0.773	GRU
Adam	0.00024	2224	360	0.174	0.779	GRU
Adam	0.00024	1024	360	0.185	0.781	GRU
RMSProp	0.00024	512	360	0.186	0.789	GRU
Adam	0.00024	1824	360	0.164	0.792	GRU
Adam	0.00012	2224	360	0.189	0.794	GRU
Adam	0.00012	1824	360	0.193	0.796	GRU
Adam	0.00024	1424	360	0.180	0.801	GRU
RMSProp	0.00024	1024	360	0.189	0.802	GRU
Adam	0.00024	1024	360	0.188	0.803	GRU
RMSProp	0.00012	512	360	0.188	0.806	GRU
RMSProp	0.00006	1024	360	0.193	0.807	GRU
RMSProp	0.00024	1424	360	0.189	0.810	GRU
Adam	0.00054	1824	360	0.196	0.811	LSTM
RMSProp	0.00024	1024	360	0.185	0.811	GRU
RMSProp	0.00012	1424	360	0.195	0.816	GRU
RMSProp	0.00024	128	360	0.193	0.816	GRU
RMSProp	0.00012	1024	360	0.184	0.816	GRU
RMSProp	0.00024	1024	360	0.175	0.818	LSTM
RMSProp	0.00012	1824	360	0.187	0.823	GRU

Table 4: Best 20 models in hyperparameter grid search. Results are sorted ascending by MSE in validation set. Values for all hyperparamaters which grid search was done is shown.

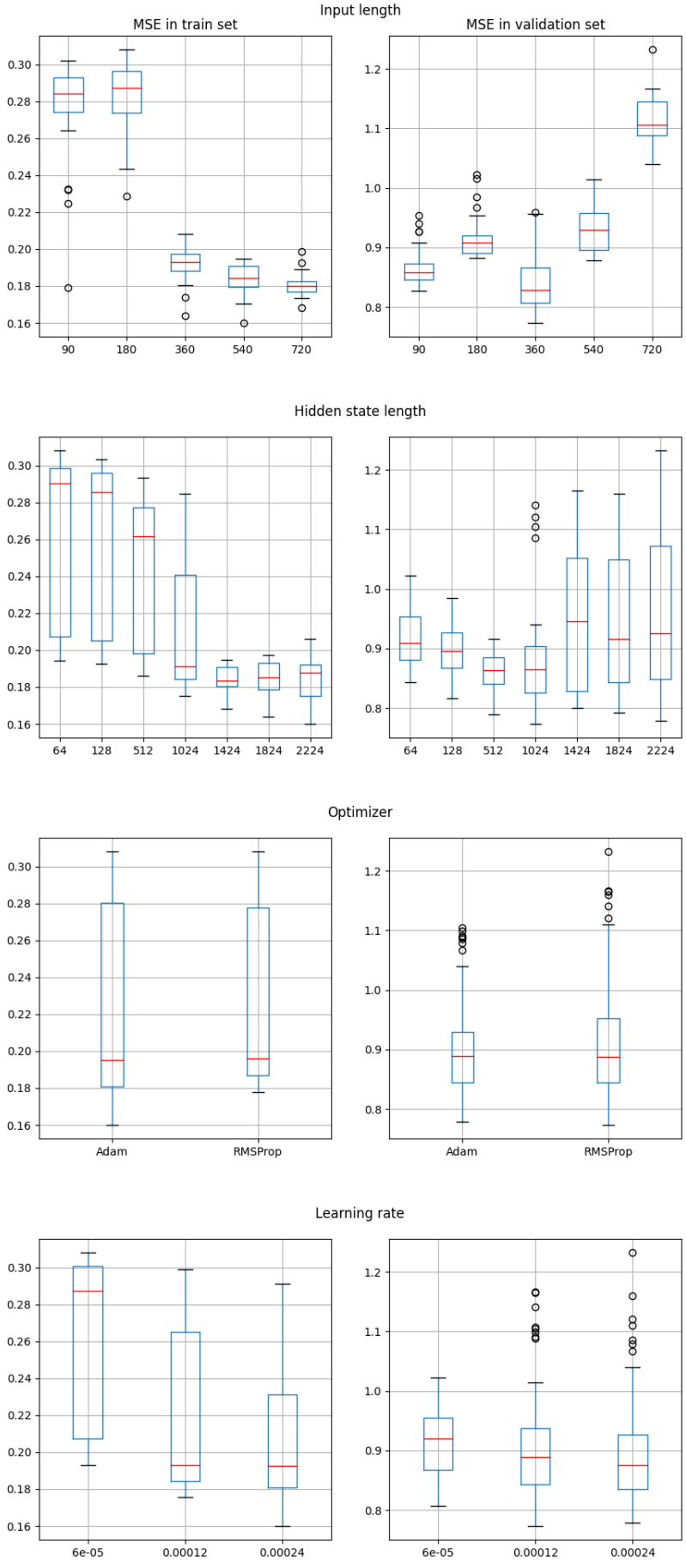

Figure 13: Grid search results for GRU model without 1D convolution layer. Boxplots of MSE grouped by different hyperparameter values.

3.4 Forecasting Accuracy Results

Forecasting was done to 18 machines. These 18 machines were randomly selected from dataset and 3 random clusters were generated from these machines. This means that each cluster has 6 machines. Clusters formed this way are described in Table 5.

Cluster	Machine IDs in cluster
1	357, 832, 358, 764, 362, 609
2	500, 570, 504, 430, 531, 791
3	513, 417, 554, 1164, 261, 796

Table 5: All 3 randomly generated clusters and corresponding machine IDs in cluster.

After the clusters were formed, data was preprocessed with Min-Max normalization and multiple models were trained on each machine. Model hyperparameters were chosen on results got in Section 3.3. Each model was trained with two different hyperparameters: Best marginal hyperparameters and best combination of hyperparameters.

Best marginal hyperparameters means that the value for every hyperparameter was chosen by selecting the hyperparameter value which provided best performance in the validation set. Each boxplot seen in Figure 13 shows MSE of all models where the hyperparameter has been fixed to some value, and other hyperparameters can vary. For example hyperparameters for GRU model with best marginal hyperparameter, each parameter value was chosen based on results seen in Figure 13. Best combination of hyperparameters means that the combination of hyperparameters which yilded the smallest loss (MSE) on validation set when doing grid search. The fundamental difference here is that the best marginal hyperparameter only takes into account one hyperparameter at a time, and the best combination of hyperparameters tries to find best value for all hyperparameters.

80% of each machines data was used on training, and 20% of the data was used to calculate validation metrics (MSE). Model performances were compared by comparing model forecast accuracy in clusters validation set.

ARIMA(p, d, q) was used as a baseline model, parameters were chosen for each machine automatically with Hyndman and Khandakar ARIMA optimization algorithm introduced in Section 2.4.1.

Figures 14, 15 shows boxplot of all the RMSEs for each model trained. Quartiles for RMSEs can also be seen in Table 6. Example forecast over the clusters can be seen in Figures 16, 17 which shows the forecast of Model 9 vs the baseline ARIMA model. Model 9 was model with convolution layer and LSTM recurrent layer, no activation used in output layer. The RMSE was calculated for each machine from whole validation set forecast. All the models are described in Table 7.

Model	First quartile	Median	Third quartile
arima	0.168	0.238	1.090
model1	0.161	0.255	1.052
model2	0.158	0.249	1.037
model3	0.189	0.294	1.156
model4	0.183	0.281	0.997
model5	0.185	0.250	1.062
model6	0.160	0.235	1.003
model7	0.180	0.304	1.104
model8	0.166	0.248	1.054
model9	0.157	0.228	0.914
model10	0.161	0.228	0.988
model11	0.157	0.227	0.972
model12	0.162	0.228	0.909

Table 6: RMSE quartiles for all models.

Model name	**Model explanation**	**RNN architecture**	**Input length**	**Hidden state length**	**Convolution layer (kernel size, number of filters)**	**Learning rate**
ARIMA	Baseline ARIMA model	-	-	-	-	-
Model 1	LSTM model with best marginal hyperparameters	LSTM	360	1024	-	0.00024
Model 2	LSTM model with best combination of hyperparameters	LSTM	360	1824	-	0.00024
Model 3	GRU model with best marginal hyperparameters	GRU	360	512	-	0.00006
Model 4	GRU model with best combination of hyperparameters	GRU	360	1024	-	0.00006
Model 5	Convolution LSTM model with best marginal hyperparameters	LSTM	270	128	(24, 5)	0.00012
Model 6	Convolution LSTM model with best combination of hyperparameters	LSTM	90	1024	(18, 15)	0.00012
Model 7	Convolution GRU model with best marginal hyperparameters	GRU	180	1024	(24, 5)	0.00012
Model 8	Convolution GRU model with best combination of hyperparameters	GRU	60	1424	(24, 35)	0.00012
Model 9	Convolution LSTM model with best marginal hyperparameters. No activation on output layer	LSTM	90	1024	(12, 15)	0.00012
Model 10	Convolution LSTM model with best combination of hyperparameters. No activation on output layer	LSTM	90	1024	(6, 15)	0.00012
Model 11	Convolution GRU model with best marginal hyperparameters. No activation on output layer	GRU	90	1024	(6, 35)	0.00012
Model 12	Convolution GRU model with best combination of hyperparameters. No activation on output layer	GRU	60	1424	(24, 35)	0.00012

Table 7: Description of all models used in Section 3.4.

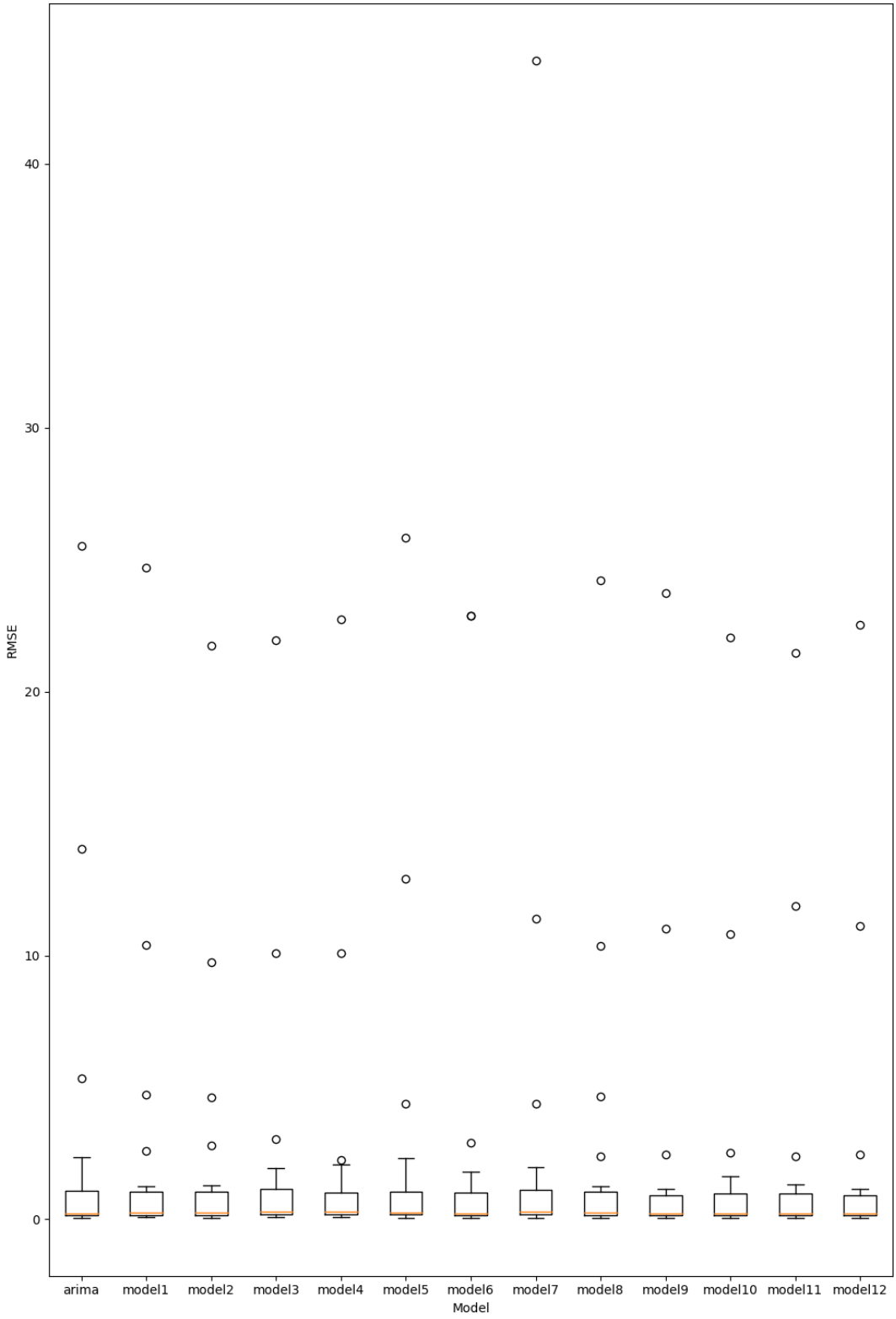

Figure 14: Boxplot of RMSEs for all 18 machines with every model.

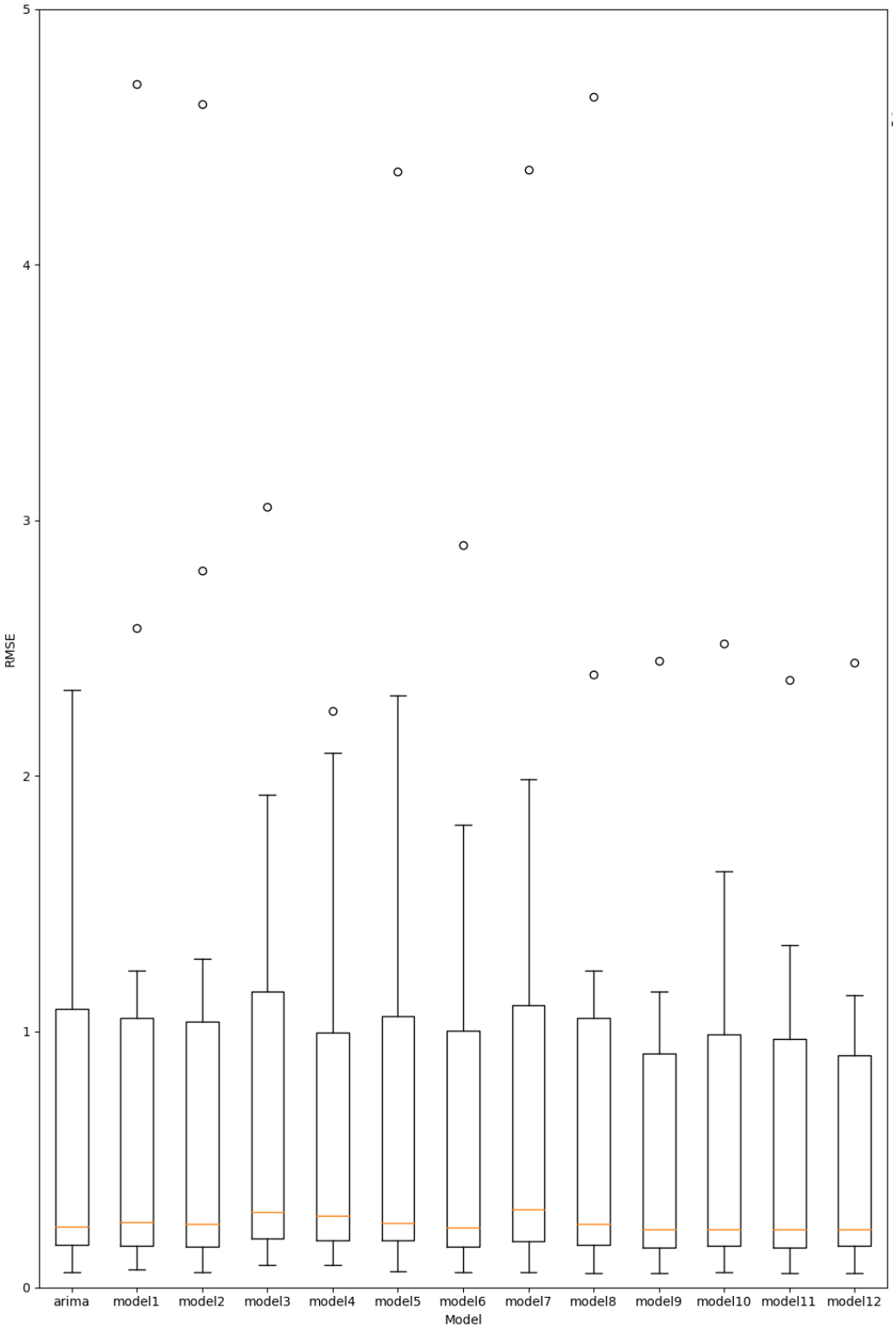

Figure 15: Boxplot of RMSEs for all 18 machines. Y-axis has been cut from 5, so some of the outliers has been cut off.

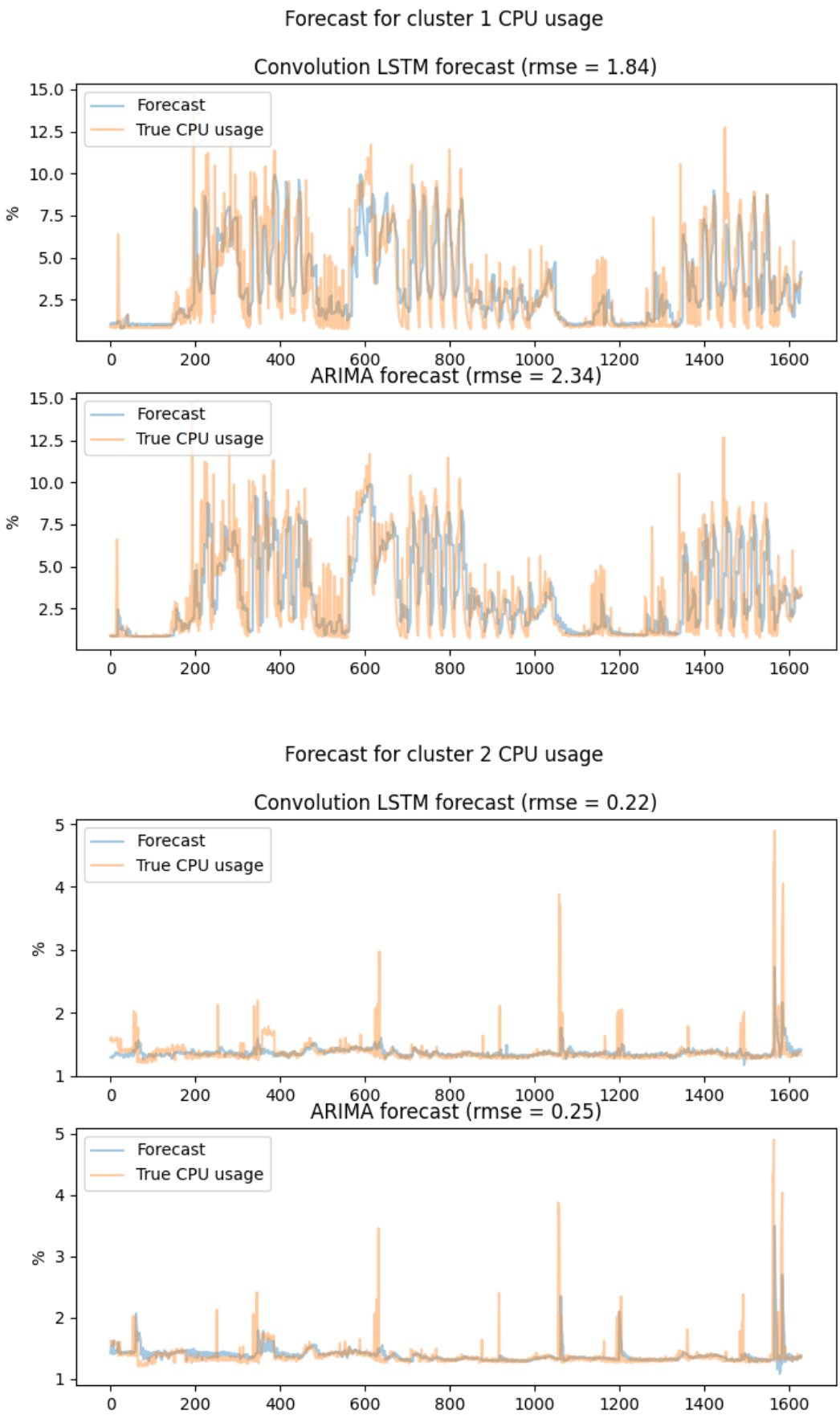

Figure 16: Forecast with Model 9 in clusters 1 and 2 compared to baseline ARIMA model.

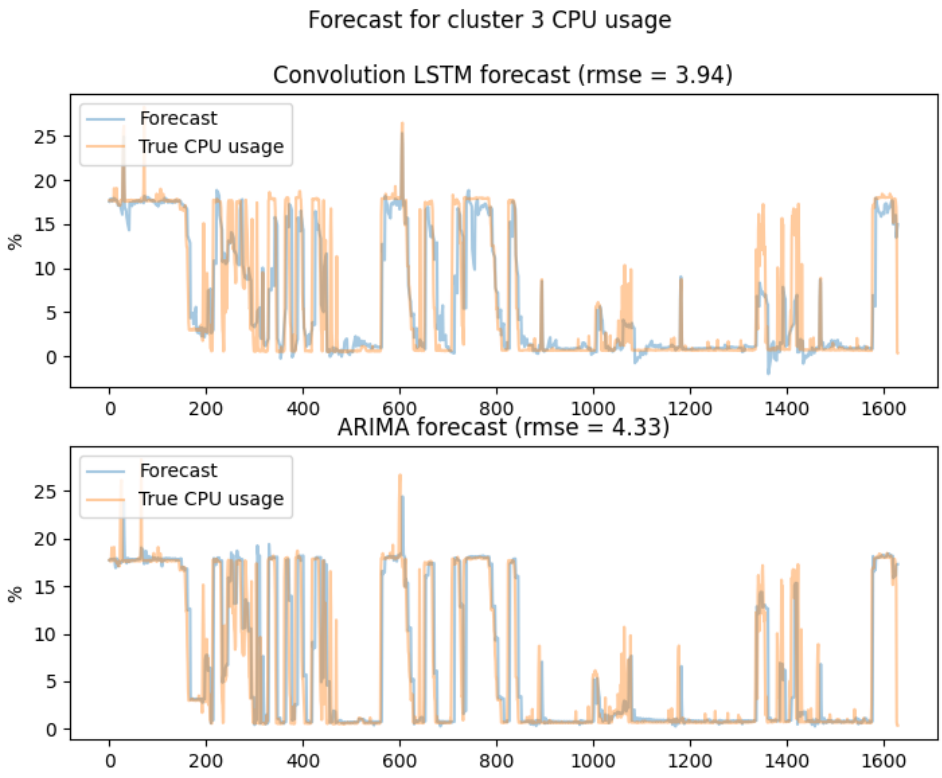

Figure 17: Forecast with Model 9 in cluster 3 compared to baseline ARIMA model.

3.5 Time Complexity of Model Training

Both RNN architectures (LSTM and GRU) was trained with machine ID 357 data for 100 epochs. 90 timesteps was used as a input layer for all models. Both RNN architectures were trained with and without Convolution layer which had 5 kernels and kernel size 24. This makes 4 different models in total, and time taken to train each of them was recorded.

Models were trained on IT Center for Science Ltd. (CSC) supercomputer Puhti using NVIDIA Tesla V100 SXM2 32GB GPU.

Results for time taken to train each model is shown on table 8

Model	Training time (s)
GRU	302.78
LSTM	335.86
GRU with convolution	238.26
LSTM with convolution	263.90

Table 8: Time complexity of all model architectures. Each model was trained for 100 epochs with GPU.

4 Discussion

When comparing models based on RMSEs on validation set by looking at Figures 14, 15 and Table 6 it can be seen that Convolution models seems to perform better when therere is no activation function used in output layer (Models 5-8 vs Models 9-12). Average median RMSE for convolution models with activation function was 0.25917 and for models without activation function in output layer the average median RMSE was 0.22762. Difference in average median RMSE between the models was 0.03155, in favour of models without activation functions in output layer. Average interquartile range (IQR) for RMSE in models with activation function was 0.88293 and for models without activation function 0.78625.

Average median RMSE for LSTM model without convolution layer (Models 1 and 2) was 0.251949 and for GRU modesl without convolution layer (Models 3 and 4) the average median RMSE was 0.287678. LSTM arhitecture outperformed GRU by 0.035729 when measured by average median RMSE

and models did not have convolution layer. Average IQR for RMSE with LSTM models was 0.88504 and for GRU 0.88989.

When comparing the models with convolution layer but without activation function in output layer it can be seen that GRU RNN architecture performs better. LSTM had average median RMSE of 0.22788 with average IQR of 0.79161, and GRU had average median RMSE 0.22735 with IQR of 0.78088.

When looking at Table 8 it can be seen that GRU RNN architecture seems to be faster to train than LSTM. This happens in both cases, with and without convolution. This seems logical since GRU contains less weights to optimize than LSTM. GRU was 29.36 seconds faster than LSTM on average when including models with and without convolution layer, and training for 100 epochs.

Interesting result was that adding convolutional layer also reduced the time required to train the model: It was 68.24 seconds faster to train models with convolutional layer than without it for 100 epochs. This happened because the input dimensions for recurrent layer got smaller from (90, 6) to (67, 5).

Combining the fact that GRU model with convolution layer (without activation function used in output layer) was fastest to train and had best forecasting accuracy, it can be said that it is the best RNN architecture from the ones introduced in this thesis to forecast resource usage in data center context.

This result can be used in data centers on multiple problems, since reliable forecast of future resource usage is valuable information. Load balancing in and between the clusters would be easier if there was reliable forecast of upcoming load. Energy efficient scalable dynamic clusters could be created, where no excess amount of nodes is powered on at any time.

Future work in this subject continues and even more efficient and accurate architectures are tried to find to forecast the data center resource usage. Also research on reinforcement learning utilizing this forecast on data center process control is being done.

References

Adhikari, R. and Agrawal, R. (2013), *An Introductory Study on Time Series Modeling and Forecasting*, LAP Lambert Academic Publishing, Germany.

Aggarwal, C. C. (2018), *Neural Networks and Deep Learning*, Springer, Cham, Switzerland.

Akaike, H. (1974), 'A new look at the statistical model identification', *IEEE Transactions on Automatic Control* 19(6), 716–723.

Andrae, A. and Edler, T. (2015), 'On global electricity usage of communication technology: Trends to 2030', *Challenges* 6, 117–157.

Bishop, C. M. (2006), *Pattern Recognition and Machine Learning*, Springer, New York, USA.

Bitbrains (2013), 'Gwa-t-12 bitbrains dataset'. http://gwa.ewi.tudelft.nl/datasets/gwa-t-12-bitbrains.

Condit, G. (2019), 'The lstm reference card'. https://www.gregcondit.com/articles/lstm-ref-card Accessed on 16.12.2020.

Fawaz, H. I., Forestier, G., Weber, J., Idoumghar, L. and Muller, P.-A. (2019), 'Deep learning for time series classification: a review', *Data Mining and Knowledge Discovery* 33, 917–913.

Feng, J. and Lu, S. (2019), 'Performance analysis of various activation functions in artificial neural networks', *Journal of Physics: Conference Series* 1237, 022030.

Gers, F., Schmidhuber, J. and Cummins, F. (2000), 'Learning to forget: Continual prediction with lstm', *Neural Computation* 12, 2451–2471.

Glorot, X. and Bengio, Y. (2010), 'Understanding the difficulty of training deep feedforward neural networks', *Journal of Machine Learning Research - Proceedings Track* 9, 249–256.

Goodfellow, I. J., Bengio, Y. and Courville, A. (2016), *Deep Learning*, MIT Press, Cambridge, MA, USA. http://www.deeplearningbook.org.

Han, J., Kamber, M. and Pei, J. (2012), *Data Mining: Concepts and Techniques, Third Edition*, Morgan Kaufmann Publishers, Waltham, USA.

Haykin, S. (2009), *Neural Networks and Learning Machines, Third Edition*, Pearson, New Jersey, USA.

Hewamalage, H. (2020), 'Recurrent neural networks for time series forecasting: Current status and future directions', *International Journal of Forecasting* 37, 388–427.

Hochreiter, S. and Schmidhuber, J. (1997), 'Long short-term memory', *Neural Computation* 9, 1735–1780.

Horvath, T. and Skadron, K. (2008), Multi-mode energy management for multi-tier server clusters, In Proceedings of the 17th international conference on Parallel architectures and compilation techniques, pp. 270–279.

Hyndman, R. and Athanasopoulos, G. (2019), 'Forecasting: principles and practice, 3rd edition'. OTexts: Melbourne, Australia. OTexts.com/fpp3. Accessed on 26.08.2020.

Hyndman, R. and Khandakar, Y. (2008), 'Automatic time series forecasting: The forecast package for r', *Journal of Statistical Software* 26, 1–22.

Kingma, D. and Ba, J. (2015), Adam: A method for stochastic optimization, Published as a conference paper at International Conference on Learning Representations, San Diego, USA.

Kwiatkowski, D., Phillips, P. C., Schmidt, P. and Shin, Y. (1992), 'Testing the null hypothesis of stationarity against the alternative of a unit root: How sure are we that economic time series have a unit root?', *Journal of Econometrics* 54(1), 159 – 178.

Nawi, N., Khan, A., Rehman, M., Chiroma, H. and Herawan, T. (2015), 'Weight optimization in recurrent neural networks with hybrid metaheuristic cuckoo search techniques for data classification', *Mathematical Problems in Engineering, Article ID 868375* .

Olah, C. (2015), 'Understanding lstm networks'. https://colah.github.io/posts/2015-08-Understanding-LSTMs/ Accessed on 16.12.2020.

Sammut, C. and Webb, G. I. (2011), *Encyclopedia of Machine Learning*, Springer US, Boston, MA.

Tieleman, T. and Hinton, G. (2012). Lecture 6e rmsprop: Divide the gradient by a running average of its recent magnitude, lecture slides, Neural Networks for Machine Learning, Coursera.

Upton, G. and Cook, I. (1996), *Understanding Statistics*, Oxford University Press.

Xue, J., Yan, F., Birke, R., Chen, L., Scherer, T. and Smirni, E. (2015), Practise: Robust prediction of data center time series, *in* '11th International Conference on Network and Service Management (CNSM)', Barcelona, Spain, pp. 126–134.