UNIVERSITY OF MESSINA

DEPARTMENT OF ENGINEERING
MASTER'S DEGREE COURSE IN ENGINEERING AND COMPUTER SCIENCE

PREDICTIVE MAINTENANCE USING LONG SHORT-TERM MEMORY AND ATTENTION MECHANISM

Supervisor: Prof. Dario Bruneo

Co-Supervisor: Prof. Fabrizio De Vita

Student: Hoang Giang Vu - 504392

ACADEMY YEAR 2021-2022

Table of Contents

Abbreviations

AI	Artificial Intelligence
AM	Attention Mechanism
ANN(s)	Artificial Neural Network(s)
BiLSTM	Bidirectional LSTM
BRNN	Bidirectional recurrent neural networks
CBM	Condition-Based Maintenance
CNN(s)	Convolutional Neural Network(s)
DAG(s)	Directed Acyclic Graph(s)
DL	Deep Learning
FCL(s)	Fully Connected Layer(s)
GAN(s)	Generative Adversarial Network(s)
GD	Gradient Descent
GRU	Gated Recurrent Units
IoT	Internet of Things
LSTM	Long Short-Term Memory
ML	Machine Learning
MLP	Multilayer perceptron
MSE	Mean Square Error
PHM	Prognostic Health Management
ReLU	Rectified Linear Unit
RMSE	Root Mean Square Error
RNN(s)	Recurrent Neural Network(s)
RUL	Remaining Useful Life
SGD	Stochastic Gradient Descent
Tanh	Hyperbolic Tangent Function

Acknowledgements

I would like to express my deepest appreciation to the University of Messina, Department of Engineering for giving me a chance to come and study here. I also could not have undertaken this journey without the support, instruction, and feedbacks from Professor Dario Bruneo and Prof. Fabrizio De Vita. I would like to extend my sincere thanks to my classmates for their helps.

Lastly, I'd like to mention my family who always stand behind and support me unconditionally.

Abstract

Predictive maintenance is a maintenance technique that identify the machine condition to apply the maintenance activities before possible breakdowns. Playing an important role in predictive maintenance is the Remaining Useful Life (RUL) prediction, which determine the remaining time to the failure point of the machine. Many state-of-the-art technologies are introduced to predict RUL, and among them Long Short-Term Memory (LSTM) based approach is considered as an efficient way. Therefore, in this thesis, a new model based on LSTM is proposed by combining convolutional layers, LSTM layers, and Attention Mechanism. Then, a dataset provided by NASA is used to train the model. Finally, we evaluate the result, and compare them to other state-of-the-art results.

CHAPTER 1: INTRODUCTION

Nowadays, in the industrial era, more and more complex systems and production lines are built up to serve human's demand, many factors are operated by these machines performance. However, regardless of how good the machine is, the degradation of them is inevitable. The machine under degeneration can perform some failures which may cost a lot of money and, time for the employers and in the worst cases, it can harm the human life. In order to avoid the consequences, maintenance is required to keep the performance of machine.

Beside using popular original maintenance techniques listed in by Martin [1], the Condition-Based Maintenance (CBM) is considered as a potential approach. By predicting remaining useful life (RUL) of a machine, maintenance can be scheduled in advance before the failure appear [2]. With the development of new technologies, RUL can be predicted pretty accurately by using Machine Learning (ML) and Deep Learning (DL) method [3].

In term of DL, Recurrent Neural Network (RNN) and Convolutional Neural Network (CNN) are two famous architectures for predicting RUL due to their abilities in handling time series data type. In addition, a new technique named Attention Mechanism (AM) is proposed as a good combination with RNN and CNN. Therefore, attention-based approaches LSTM or CNN are increasingly popular nowadays.

In the following chapter, we will go through a brief history of Artificial Intelligence (AI). Then, an overview of ML and DL will be provided. After that, we will clarify the basics of Artificial Neural Network (ANN), and its components such as: layers, activation, and optimizers. Based on ANN, we will explore the basic of CNN and RNN, especially LSTM network. In addition, we will know about the AM and how can we calculate attention score correctly. Last but not least, on-device ML will be mentioned.

In chapter 3, C-MAPSS dataset will be introduced briefly. Next, due to the original data is not processed, we will analysis and clean the data before passing them into the model.

In chapter 4, methodology approach will be built by evaluating many different which approaches from simple to complex ones. Firstly, a single method between CNN and LSTM will be chosen to predict RUL. Next step, we will choose a combination between two factors among CNN, LSTM and AM. Lastly, the attention-based CNN-LSTM will be introduced.

After that, we will introduce important components in implementing a model such as: execution environment, configuration. Then a model can be evaluated and improved by using performance metrics RMSE, R2, and MSE. In addition, we will explore how can we embed a model into a microcontroller. Running model according to these implementations, in chapter 6, and we will compare our result to other state-of-the-art architectures.

The final chapter, we have a review of our works during thesis making process from the ideas to the result. Then the end of the thesis will indicate the limitations of this model, also how can we improve them in further study.

CHAPTER 2: BACKGROUND KNOWLEDGE

2.1. A brief history of AI, ML, and DL

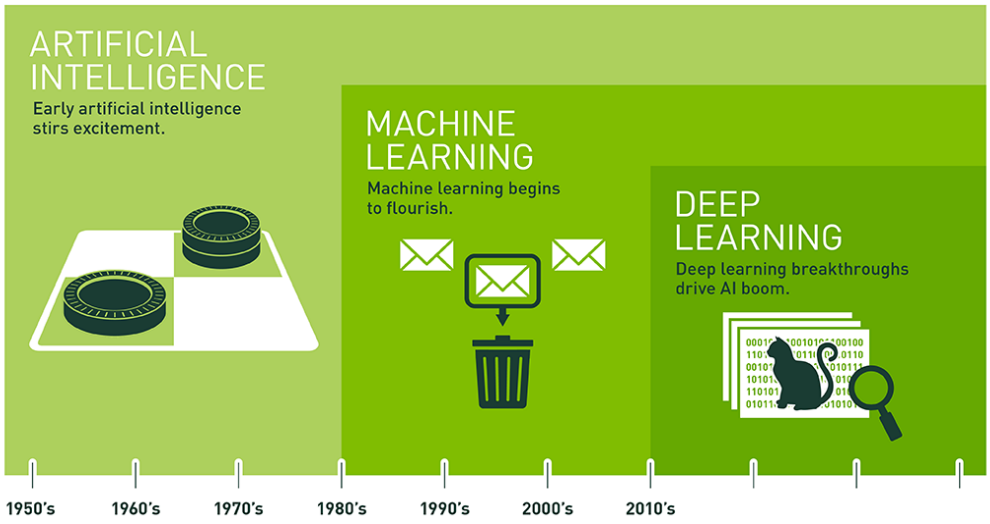


Figure 2.1. AI, ML, and DL [4]

The concept of "machine learning" was first adopted in 1952 by Arthur Samuel, in one of his articles where he developed a sort of computer program for playing checkers. In particular, he described the term as "[…] a field of study that gives the ability to the computer for self-learn without being explicitly programmed" [5]. In 1957, the first successful neurocomputer named Mark 1 perceptron is developed. In the next few decades, the discovery of feed forward neural network and backpropagation made machine learning, artificial intelligent more complex and practical, led ML to a stage of practicality. At this point, machine could enhance its performance through human knowledge and experience: "A computer algorithm/program is said to learn from performance measure P and experience E with some class of tasks T if its performance at tasks in T, as measured by P, improves with experience E." [6]. However, according to a decreasing funding and interest, along with its own limitations, ML and AI suffered an "AI winter" in the

1990's. At present, we observe a wave of "Deep Learning" applications are led to a new flourish era of ML.

2.2. Machine learning

2.2.1. Overview

ML is suggested to be the future of human's technology. Nowadays, this new sector introduces many applications, which have aided humanity's advancement. In the beginning, the major distinction between people and computers was basically in the ability of autolearning. In other terms, human learns from their experiences, which implies a process of self-development through time, but computers and robots must follow a pre-determined method, programed by human. Computers are logical devices that function merely through commands. Particularly, if we want the machine to perform a task, we must provide specific, step-by-step methods, or clear and simple instructions. The learning process initiates with data or observations. For instance by using the inputs we provide, computers are able to look for patterns in data and to make better decisions in the future. The basic goal of this process is for computers to learn on their own, without the need of intervention, and to adapt their activities accordingly. As a result, ML is widely applied for modern solutions to many problems, such as those which involve a lot of fine-tuning and rules or are difficult to be solved. Moreover, in regard to the advancement of Big Data, the combination of ML, AI and Cloud Technology is an optimized solution in handling a huge volume of data.

2.2.2. Categories

ML algorithms are often split into four categories, which are based on the the learning method as a framework: supervised learning, unsupervised learning, semi-supervised learning, and reinforcement learning.

- **Supervised learning**

Supervised machine learning algorithms can apply what they have learned in the past to new data and predict future events using labeled examples. Nowadays more and more data is labeled, so Supervised Algorithm is the most popular among ML algorithms. Labeled data is used to classify different values (classification) or predict a single value (regression). With Supervised Learning, besides a robust and efficient models, collecting, and labeling healthy data is also a key factor in improving model performance [7].

- **Unsupervised Learning**

Unsupervised Learning techniques, on the other hand, are utilized when the data used to train is neither classed nor labeled. The goal of algorithms that take these techniques is to represent the data's structure or hidden information. To put it another way, employing these approaches is more about expressing the data's attributes or features [7].

- **Semi-Supervised Learning**

Problems where we have a large amount of data, but only a portion of them is labeled, is called Semi-Supervised Learning. The problems in this group fall between the two groups mentioned above [7].

- **Reinforcement Learning**

Reinforcement Learning are algorithms that help a system automatically determine behavior based on circumstances to achieve the most benefit (maximizing the performance). Currently, Reinforcement Learning is mainly applied to Game Theory, algorithms need to determine the next move to achieve the highest score.

In this project, since data is labelled, linear regression and random forests from supervised learning are used to solve the problem [7].

2.3. Deep learning

2.3.1. Overview

DL is a sub-field of ML and one of the key factors for the new ML era. It simulates the structure of neuron network inside human brain to perform human brain behavior as similar as neural networks but in a deeper level. In other words, DL model tries to mimic the process that

whenever human detect a new object, human brain is going to extract object's properties and compare to recognized objects in the memory. In an article published by Nature journal in 2015, Yann LeCun and other authors described the functions and applications of deep learning as: "Deep learning allows computational models that are composed of multiple processing layers to learn representations of data with multiple levels of abstraction. These methods have dramatically improved the state-of-the-art in speech recognition, visual object recognition, object detection and many other domains such as drug discovery and genomics." [8].

2.3.2. Categories

Nowadays, DL is categorized in many ways, such as by algorithms, by architectures … In this thesis, deep learning is classified according to types of neural networks as: CNN, RNN, and ANN, which are the core networks.

- **Artificial Neural Networks**

ANN is a multiple perceptron, which has a stack of numerous hidden layers.

- **Convolutional Neural Networks**

CNN is a regularized multiple perceptron, which is inspired by the architecture of the visual cortex of the brain [9], which is one of the reasons why CNN model performs well on image tasks. This architecture will be discussed in the detail in the next section.

- **Recurrent Neural Networks**

RNN is considered as an extension to feedforward networks and it has connections pointing back. According to this structure, RNN is able to handle long sequences of text or time series data, which are mainly used in nature language processing or forecasting. This architecture will be discussed in the detail in the next section.

2.3.3. Neural Network

Neural Network was first introduced in 1944 by Warren McCullough and Walter Pitts in their paper *A Logical Calculus of Ideas Immanent in Nervous Activity*. According to biological neurons' inner function, two researchers created a simplified computational model which is called as threshold logic unit.

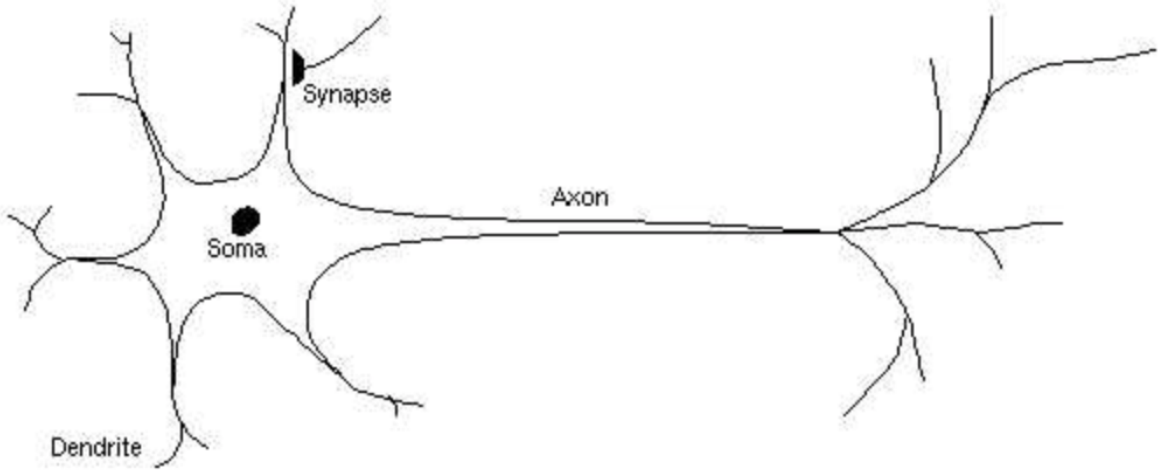


Figure 2.2. Biological Neuron [10]

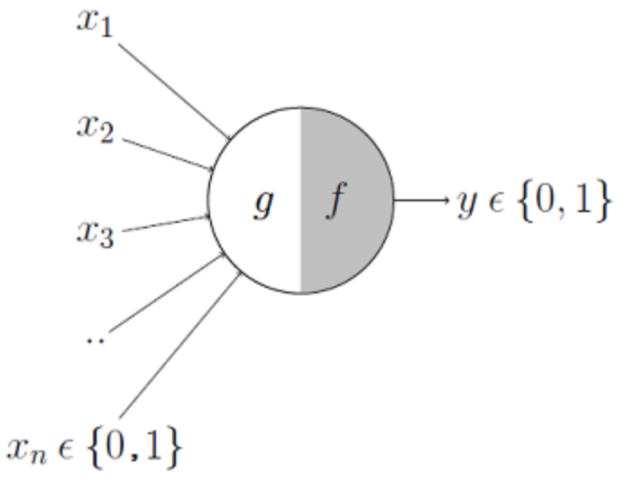


Figure 2.3. Threshold logic unit [10]

In an actual neuron, dendrite receives electrical signals from other neuron's axon, sends the signals in numerous amounts to its axon by synapse, and be ready to fire when the signals surpass a threshold. In a threshold logic model, the signals are numerical values, the output is determined by the combination between calculating the summary of input's weights and applying a threshold.

- Perceptron

Perceptron is one of the simplest mathematical models of a biological neuron, it is based on Warren McCullough and Walter Pitts' s research - threshold logic unit. Perceptron calculates a weighted sum of the inputs as well as an adjustable, numerical term, called bias and applies a step function to get the output.

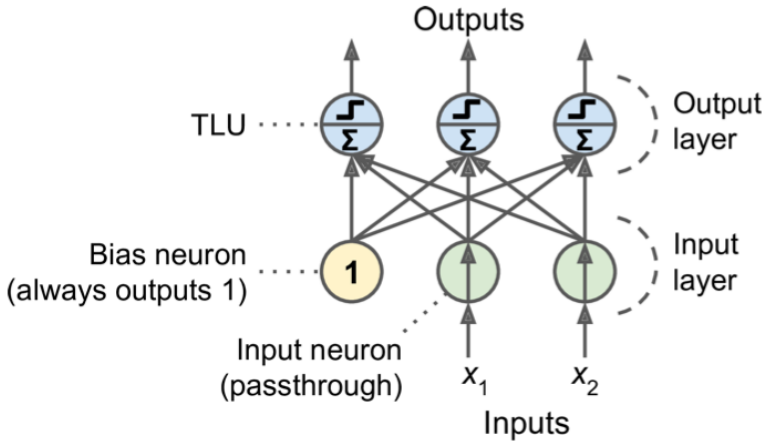


Figure 2.4. The structure of Perceptron [10]

The output layer is called fully connected layer or dense layer, since output neurons are connected with all neurons in input layer.

The output layer or fully connected layer is computed by an equation:

$$h_{w,b}(X) = \phi(XW + b)$$

In which, X is the matrix of input values, W is the connection weights, b is the connection weights from bias, and ϕ is the activation function, which will be mentioned later in this thesis.

A perceptron is trained by a learning rule, which reinforces the connection weights each time the predicted values is not as good as expected ones. Learning rule is described as:

$$w_{i,j}^{(next\ step)} = w_{i,j} + \eta(y_j - \hat{y}_j)x_i$$

In this equation, w is the connection weights between layer i^{th} and j^{th}, η is the learning rate, y_j, $\hat{y}_j$ are the target value and predicted value respectively, and x_i is the values of layer i^{th}.

- The multilayer perceptron and backpropagation

Multilayer perceptron (MLP) contains an input layer, an output layer as a normal perceptron. In between input and output layer, there are a series of layers called hidden layers. MLP is a feedforward neural network with all layers are fully connected except the input layer. With the development of computational power, more and more hidden layers are added into MLP. As a result, MLP became a deep neural network, which became the cornerstone of deep learning studies.

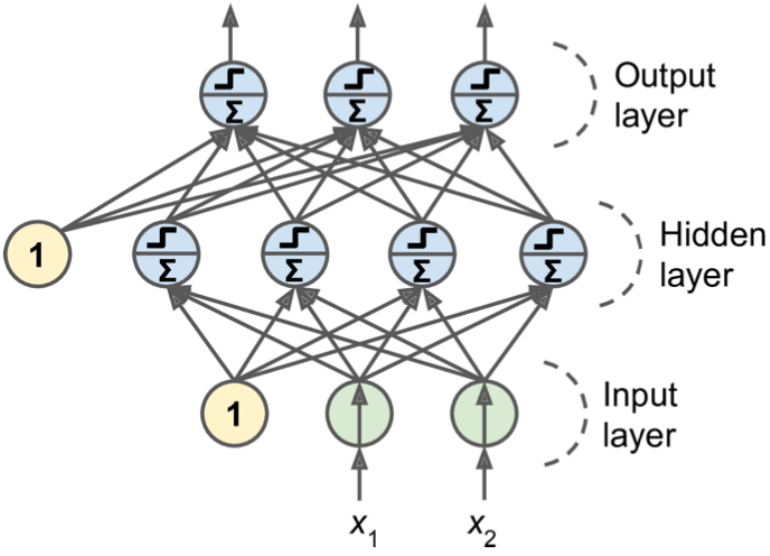


Figure 2.5. The structure of Multilayers Perceptron [10]

Backpropagation algorithm was first introduced in 1960s and it quickly became an important breakthrough by solving the MLP training problem, which is described in a paper called *Learning representations by back-propagating errors* in 1980. Backpropagation algorithm aims to minimize the difference between the predicted values and actual values by tweaking the connection weights every training instance [11].

A neural network is constructed by a lot of components, and one of them is hyperparameters. Hyperparameters are the values which decide the structure or the training method of the network, "[…] machine learning algorithms involve hyperparameters that have to be set before running them. Options for setting hyperparameters are default values from the software package, manual configuration by the user or configuring them for optimal predictive performance by a tuning procedure." [12]. In this thesis, I will introduce some sort of popular hyperparameters such as: layer, activation functions, and optimizers.

2.3.3.1. Layers

In among neural network's components, layers lay in the highest-level building block in a neural network architecture. Information is passed through all the layers, in which the first layer called input layer, the last one called output layer and others called hidden layer. In order to optimize the neural network model, there are several kinds of layers such as: convolutional layer and pooling layer in the CNN, dense layer in the traditional neural network, RNN layer and LSTM layer in RNN, LSTM, etc.

2.3.3.2. Activation functions (AFs)

In a neural neuron, the AFs convey the sum of the whole incoming electrical signal from previous cell to an expected output or an input of the next cell [13]. In other words, AFs represent by an equation, which converts all incoming values to an appropriate form that can become next cell's input.

Apart from the overview that was mentioned above, the AF is also the main reason, which makes DL stand out other technology. It provides the model ability to learn high-level patterns and solve the complex, non-linear problems by adding non-linear AF such as training virtual assistant

in nature language processing, face recognition in computer vision, etc. Moreover, there are some other properties of AF, which are listed below:

- Vanishing gradient problem

  Vanishing gradient problem is experienced when applying backpropagation algorithm in training process. Particularly, the gradient descent step in backpropagation algorithm, after every epoch it tweaks the connection weights to reduce the error and, in some case, the gradient becomes relatively small, preventing the weight updating [14].

- Zero-centered

  A zero-centered AF is a function that consists of both negative and positive numbers. If the function is not zero-centered, the output of a layer will tend to be all positive or all negative. Thus, training process needs more time to be completed [13].

- Computational cost

  Since each neuron has its own activation function, so the number of computations can become millions. Beside the computational cost of activation functions, model also need to do computations of the gradient. In training process, if these computations are high computational cost, it will require more time, and vice versa. A low computational cost activation function is preferable when it cost less money, time, and energy [13].

- Differentiable

  Differentiable property is required to allow backpropagate the error in training process [13].

In this thesis, I will introduce some widely used non-linear AFs such as: Sigmoid, Softmax, Hyperbolic Tangent Function (Tanh), Rectified Linear Unit (ReLU), Leaky ReLU.

- **SoftMax**

  The SoftMax function is one of the earliest AFs in the history of neural networks. It is normally used in multi-class classification problem by turning a vector of n real values into the same size vector of probabilities that sum to 1 as described in the page 184 of DL book “Any time we wish to represent a probability distribution over a discrete variable with n possible values, we may use the SoftMax function.” [15]. In detail, the SoftMax function takes an array of input, which can be positive, negative, zero, and turns them into a normalized probability distribution or possibility of classes in range [0:1]. As a result, the SoftMax function is usually used in the output layer of the network to classify. For the example below, the input is a dog picture, the network extracts the properties and put them into the SoftMax function. The output is the probabilities of all classes, in which the dog class turns out the highest score.

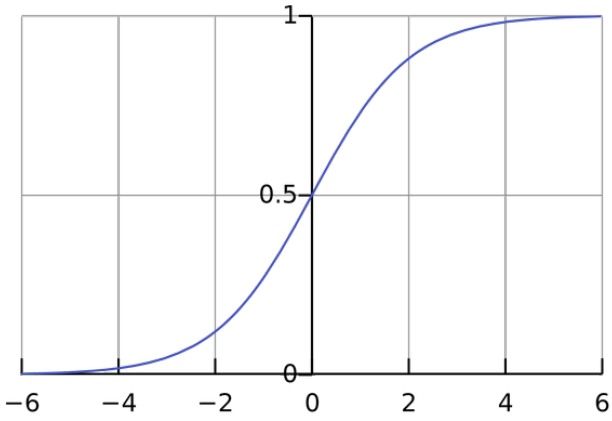


Figure 2.6. SoftMax function [10]

A SoftMax function is expressed by the formular below:

$$f(x) = \frac{\exp(x_i)}{\sum_j^n \exp(x_j)}$$

Where x is the input vector of AF, exp is the standard exponentiation function applied to each input value. The output of SoftMax offers a small probability if the one of the input values is negative and turns out a large probability if the one of the input values is positive. In spite of small or large probability, it always in range [0:1], which optimizes the calculating, interprets probability easily, and minimizes the chance of extreme values or outliers.

In fact, the contribution of the SoftMax AF in the beginning of ML, DL era is undeniable, but with the occurrence of deep neural networks, researchers proposed many other AFs with better performance and the SoftMax function is almost used in the output layer.

- **Sigmoid**

  The sigmoid function has used in ML and DL from the beginning. Since the characteristic S-shape curve, this mathematical function is widely used in classification problem, particularly, in binary classification problem. In ML, a sigmoid function normally is used to convert a number to a probability, which is applied to next step or to presenting. In DL, the sigmoid function is the core of logistic regression model, which is the popular solution for binary classification problem. In depth, in a logistic regression model, sigmoid functions outputs a probability x in range of 0 to 1, a threshold y, and the model output will become 0 if probability x < y, and become 1 if probability x > y.

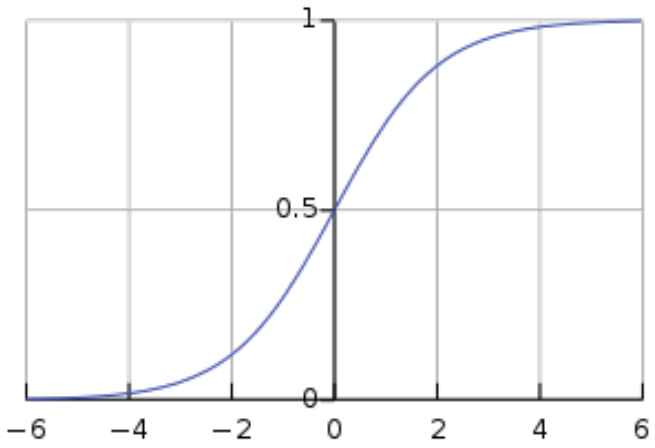


Figure 0-1.7. Sigmoid function [10]

A sigmoid function is expressed by the formular below:

$$f(x) = \frac{1}{1 + e^{-x}}$$

Where x is the input of activation function, f(x) is a probability, which is the output of the sigmoid function. When x goes ahead to more positive value, f(x) becomes as close as 1, and vice versa. Suppose the threshold equals 0.5, two classes: dog representing as "1", cat representing as "0", and logistic regression model prediction is represented as:

$$\hat{y} = \begin{cases} 1 \; if \; \hat{p} > 0.5 \\ 0 \; if \; \hat{p} < 0.5 \end{cases}$$

In which, $\hat{y}$ is the prediction, and $\hat{p}$ is the output from sigmoid function.

On the other hand, sigmoid function is not a zero-centered activation function, as well as it suffers vanishing gradient problems, other functions were established.

- **Tanh**

  In order to overcome the "not a zero-centered activation function" of sigmoid function, tanh function was introduced. Tanh functions are slightly similar to sigmoid functions, it has all properties of sigmoid function such as: S-shape curve,

differentiable, continuous, and gets a range of -1 to 1instead of 0 to 1, which gives Tanh function the zero-centered property. As a result, Tanh functions offer better performance in training multi-layers neural network and aids the operation of backwards progress [16]. These key advantages make the Tanh function vaguely popular than the sigmoid function. It is widely employed in RNN applications as speech recognition, nature language processing, etc.

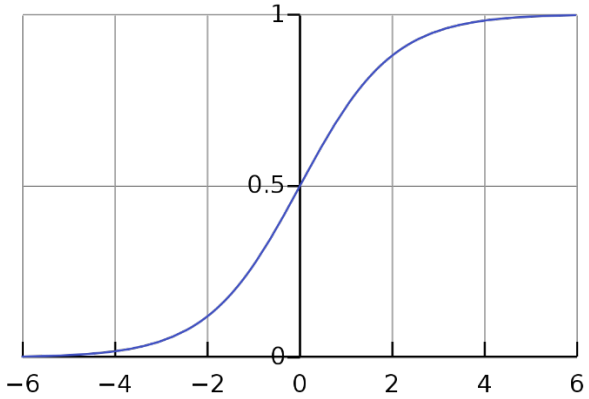


Figure 2.8. Tanh function [10]

A Tanh function is expressed by the formular below:

$$f(x) = \frac{1 - e^{-x}}{1 + e^{-x}}$$

Where x is the input of AF.

In spite of enhancements, Tanh function still gets the vanishing gradient problem and expensive computational cost as sigmoid function. These two problems were spurred the advent of ReLU AF.

- **ReLU**

In 2010, ReLU was first proposed by Nair and Hinton in their paper "Rectified linear units improve restricted boltzmann machines", which has been introduced a state-of-the art to improve the performance of DL in many applications til today.

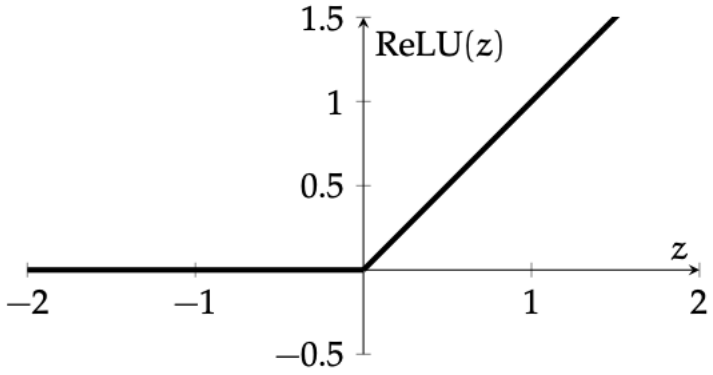


Figure 2.9. ReLU function [10]

The ReLU function is defined by the below formular:

$$f(x) = \max(0, x)$$

Where x is the input of AF. The function will compare input x to zero, in which if input is negative, function is going to put zero, and if input is positive, function is going to put x.

$$f(x) = \max(0, x_i) = \begin{cases} x_i \; if \; x_i > 0 \\ 0 \; if \; x_i < 0 \end{cases}$$

In DL, ReLU is proved that it is six times faster than tanh and sigmoid function , as well as it needs extremely cheap computational cost to compute comparing to predecessors since all negative values are put to zero and there are only comparison, addition, and multiplication in computation. In general, ReLU activation function performs an oustanding performance in all aspects in comparison to the other activation functions to prior 2011. As a result, ReLU is widely used in neurons in hidden layers or output layer and becomes the most popular activation function for deep neural networks [17].

In contrast, the ReLU still exists some issues such as: non-differentiable, not zero-centered, unbounded, and especially dying ReLU problem, which makes the network

flimsy during training process because neurons can be fell into inactive state by negative inputs, and there is no contribution from these neurons. The Leaky ReLU is suggested as a solution to dying ReLU problem.

- **Leaky ReLU**

  The leaky ReLU was first introduced as a variant of ReLU in 2013 by Andrew L. Maas and his colleagues in their paper "Rectifier Nonlinearities Improve Neural Network Acoustic Models". This new variant solves the dying ReLU problem by adding a small part of negative slope to maintain the state of neurons instead of being zero during training process.

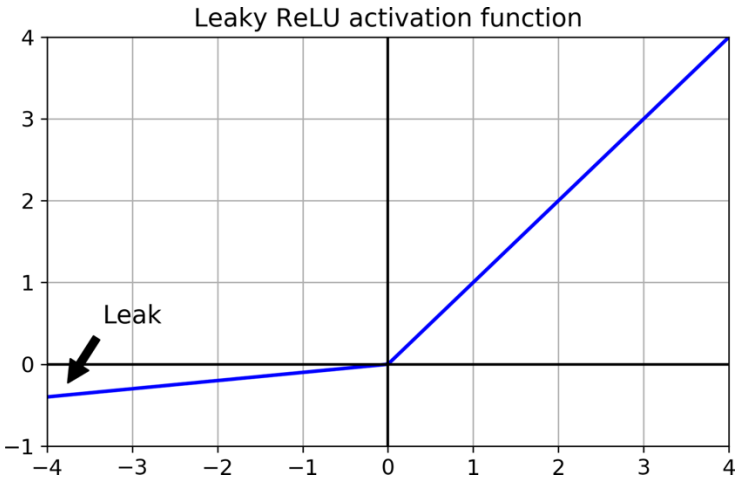


Figure 2.10. Leaky ReLU function [18]

The Leaky ReLU function is defined by the below formular:

$$f(x) = \alpha x + x = \begin{cases} x \; if \; x > 0 \\ \alpha x \; if \; x < 0 \end{cases}$$

Where x is the input of activation function. The gradient is always set at 0.01, and it leads to the possibility of gradient vanishing problem. Since the Leaky ReLU is a ReLU variant, they share some same properties such as: cheap computational cost, high performance in deep neural networks, unbounded problem. However, the Leaky

Relu also has its own properties, which may get the better of ReLU in some cases. They are the zero-centered property and training speed. In detail, zero-centered property solves the dying ReLU problem and makes the Leaky ReLU more balanced than ReLU, as well as the training speed is increased by the closer distance between mean activation to zero, the faster training speed.

2.3.3.3. Optimizers

In the beginning of this thesis, learning rule was introduced as an algorithm to reduce the losses and increase the accuracy of the neural network during training process by modifying and updating the network's parameters. This method is called the optimization, and these algorithms are called optimizer, learning rule is the most basic one among them. In general, optimizers are algorithms or methods, which adjust the parameters of a network such as: weight, bias, learning rate in order to minimize the loss function and help the network get more accurate. In addition, there exist a lot of common optimizers as well as their performances are distinct from each other, or even the same optimizer but it offers separate result in different parameter setting. For that reason, it is necessary to choose an appropriate optimizer, and in this thesis, I will introduce some popular ones such as: Gradient Descent (GD), Batch Gradient Descent, Stochastic Gradient Descent (SGD), Mini-batch Gradient Descent, Stochastic Gradient Descent with Momentum, Adaptive Learning Rate methods (AdaGrad, RMSprop, Adam).

- **Gradient Descent**

  GD is well-known as one of the most basic and used optimization algorithms, it is also employed in backpropagation stage, as well as combined with other algorithms to archive better result in optimizing. In mathematical terms, GD is a first-order iterative optimization algorithm, that uses calculus to modify values iteratively in

order to minimize a function, or specifically loss function. The concept of gradient is considered as a slope, in particular, the value of gradient is the crucial element that determines the steepness of slope and the velocity with which the model learns: "GD … - the stepwise process that moves downhill to reach a local minimum." [19]. Suppose a blind man is lost in a mountain, and the nearest residential area is located at the mountain's base. The best strategy to reach the residential area as quick as possible is to go to the steepest direction of the mountain. This is a visual example of how GD works.

The GD is defined by the below formular:

$$\theta^{next} = \theta - \eta \frac{\partial}{\partial_\theta} J(\theta)$$

Where θ is the parameter vector, η is the learning rate, and J(θ) is the cost function. In the first place, the value of θ is set randomly (random initialization step), then it is tweaked gradually step by step each time to decrease the cost function J(θ) until reaching the minimum.

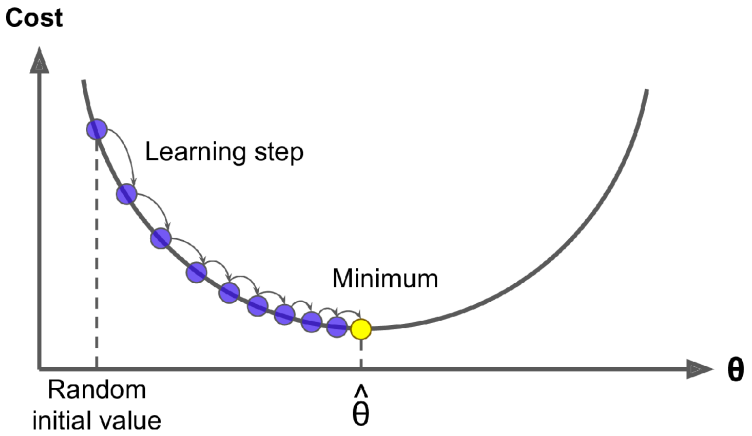


Figure 2.11. GD [10]

By this graph, GD is easy to understand and implement. However, unlike blind man finding the way to mountain's base by only steepest direction, GD has another

important element, which is size of steps or learning rate. If the learning rate is extremely low, the velocity is slow and it takes a lot of time to reach the minimum. On the other hand, if the learning rate is excessively high, it might reach the minimum very fast, but it also can pass the minimum and reach the other side of the slope. It is essential to choose the optimal learning rate.

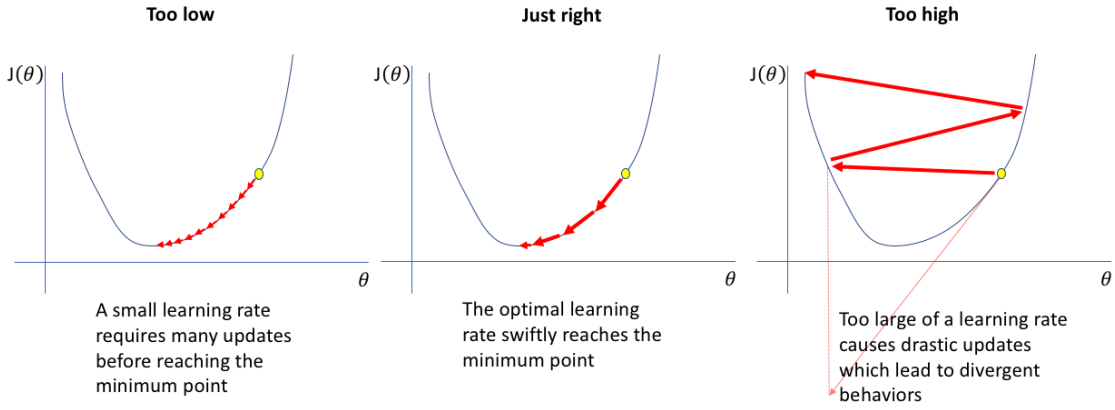


Figure 2.12. Three situations of GD [10]

No matter how GD works well for most situation, it still has some drawbacks. It might have expensive computational cost if dealing with the large amount of data and does not work well with nonconvex function. To cope with the first problem, some other variants of GD are proposed such as: Batch Gradient Descent, Stochastic Gradient Descent, and Mini-batch Gradient Descent.

- **Batch Gradient Descent**

  As mentioned above, GD is applied in the model by calculating the gradient of the cost function regarding each neural network parameters. In Batch GD, the gradient of the cost function of the whole parameters are calculated at the same time by using the entire batch of training data at each step. Obviously, this method still gives out a slow computational in very large data set, but it offers better performance compared

to traditional GD. In addition, Batch GD performs quite well with the large number of features, as well as a stable convergence.

- **Stochastic Gradient Descent**

  SGD is proposed to overcome the downside of Batch GD in dealing with the large amount of data. Instead of computing the cost function of the entire parameters of the whole training data, SGD picks randomly an instance from the training data at every step, then calculates the gradient of picked instance. As a result, handling an instance per step makes SGD much faster than Batch GD, and has a cheaper computational cost. However, this method also has its own drawback. This method offers a faster speed, but in a stochastic direction, that is why it is called SGD : "instead of gently decreasing until it reaches the minimum, the cost function will bounce up and down, decreasing only on average … the final parameter values are good, but not optimal." [10].

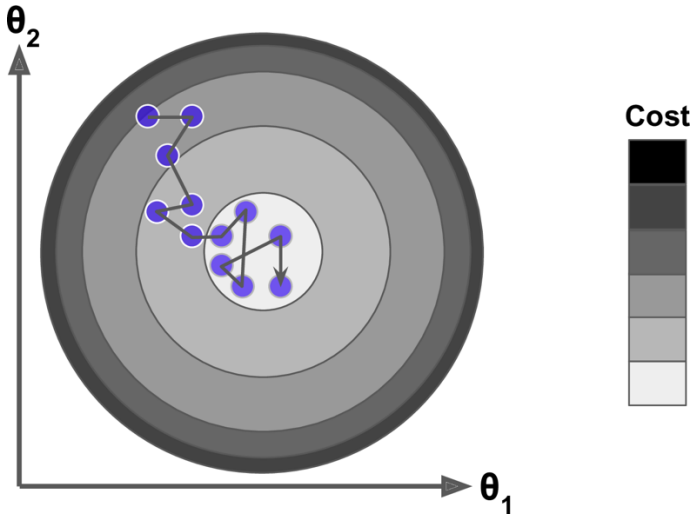


Figure 2.13. SGD [10]

- **Mini-batch Gradient Descent (Mini-batch GD)**

Mini-batch GD is a combination between Batch GD and SGD. In particular, Mini-batch GD computes gradient of cost function of a small random group instances from the training data at every steps rather than the whole training data in Batch GD, or only an instance in Stochastic GD. This strategy gives Mini-batch GD a balanced achievement between Batch GD's accuracy and Stochastic GD's training speed.

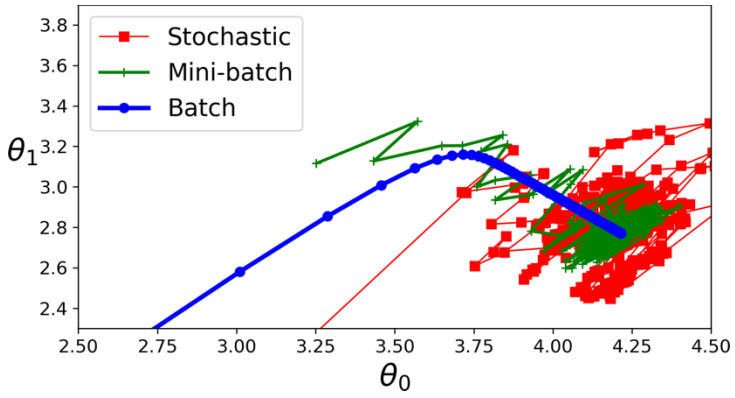


Figure 2.14. GD, SGD, and Batch GD [10]

The above figure describes three variants of GD in parameter space. Both Mini-batch GD and SGD end up around the minimum, while Batch GD end up exactly at the minimum. However, Batch GD costs more time to reach the minimum than the other variants. Finally, learning rate is also an important element, that affects the speed and accuracy of optimizer a lot, Mini-batch can stop at the minimum or Batch GD can reach the minimum faster. The batch size is normally set to 32 as the default [20].

- **Stochastic Gradient Descent with Momentum**

  As outlined above, SGD offers a high-performed training speed, but also a high variance in cost function every step which leads to the non-optimal minimum or sometimes to the high computational cost: "instead of gently decreasing until it reaches the minimum, the cost function will bounce up and down, decreasing only on average … the final parameter values are good, but not optimal." [10]. In order to

overcome this stochastic direction in reaching the minimum, SGD with Momentum was proposed. The momentum is a strategy that increases the speed of convergence in SGD algorithm by averaging the oscillations along the training process [21]. This implementation is done by adding a hyperparameter γ, known as the momentum term, into SGD algorithm.

The SGD with Momentum is defined by the below formulars:

$$v^{next} = \gamma v + \eta \frac{\partial}{\partial_\theta} J(\theta)$$

$$\theta^{next} = \theta - v^{next}$$

Where θ is the parameter vector, η is the learning rate, γ is the momentum, ν is the update vector and J(θ) is the cost function. The value of γ is ranged from 0 to 1, but it is typically initialized to 0.9, sometimes between 0.8 and 0.99. The higher momentum value, the more stable in gradient. For instance, the smaller value of γ, like γ = 0.5, the path gives out to be fluctuated, On the other hand, the bigger value of γ, like γ = 0.9, the path turns out to be smoother.

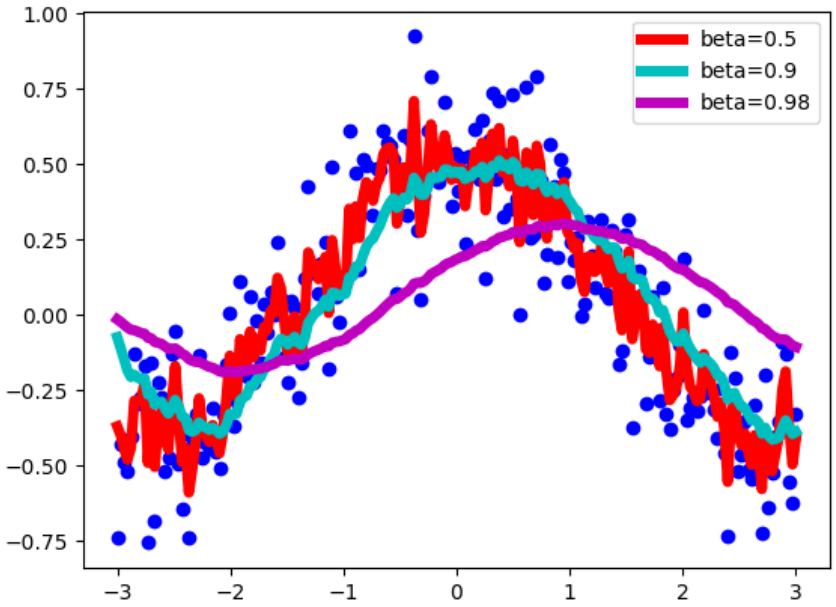


Figure 2.15. SGD with different momentums [10]

- **AdaGra**

  In the traditional GD, the important of the learning rate is undeniable, but this hyperparameter has to be initialized and heavily depend on the neural network architecture and training data. In addition, the learning rate is unchangeable during the training process, that makes the model less adaptive in the change of parameters. In order to get over these challenges, researchers were proposed Adaptive Gradient Descent algorithms.

  AdaGra algorithm belongs to Adaptive Gradient Descent algorithms, this algorithm guides and correct the gradient direction toward the global minimum from beginning of training process by scaling down the gradient vector.

  The AdaGra is defined by the below formulars:

$$s = s + \nabla_\theta J(\theta) \otimes \nabla_\theta J(\theta)$$

$$\theta_{next} = \theta - \frac{\eta \nabla_\theta J(\theta)}{\sqrt{s + \mathcal{E}}}$$

Where θ is the parameter vector, η is the learning rate, s and ε are elements used for scaling down the learning rate, and J(θ) is the cost function. The scale down rate is normally set to 0.01.

The main benefit of AdaGra implementation is the elimination of tuning learning rate manually, and the short training process. However, this method usually stops too early before reaching the minimum, since the scaled down learning rate is extremely small after a time of training. This issue is the main reason AdaGra algorithm is not widely implemented in the deep neural networks. Instead of using AdaGra, RMSProp and Adam are normally in use.

- **RMSProp**

  RMSProp was first proposed by Tieleman & Hinton in his online course "Neural Networks for Machine Learning" in 2012, in order to overcome the drawbacks of AdaGra. Its structure and benefit are quite similar to AdaGra, but rather than scaling down the learning rate by decaying sum of gradient squared, RMSProp scales the learning rate by decaying average of squared gradient, using a decay factor β. This method prevents the significant decrease in the learning rate: "RMSProp extends Adagrad to avoid the effect of a monotonically decreasing learning rate." [22]

  The RMSProp is defined by the below formulars:

$$s = \beta s + (1 - \beta)\nabla_\theta J(\theta) \otimes \nabla_\theta J(\theta)$$

$$\theta_{next} = \theta - \frac{\eta \nabla_\theta J(\theta)}{\sqrt{s + \mathcal{E}}}$$

Where θ is the parameter vector, η is the learning rate, β is the decay rate, s and ε are elements used for scaling down the learning rate, and J(θ) is the cost function. The decay factor is normally set to 0.9.

- **Adam**

  After the succession of RMSProp, an other Adaptive Gradient Descent algorithm was proposed by Diederik P. Kingma and Jimmy Ba in their paper "Adam: A Method for Stochastic Optimization" in 2014. The new algorithm is named Adam, and its idea is based on a combination between RMSProp and momentum optimization [23]. In particular, it scales down the learning rate by decaying average of the gradient like the momentum method, and also scales down the learning rate by decaying average of squared gradient like RMSProp.

  The Adam is defined by the below formulars:

$$m = \beta_1 m - (1 - \beta_1)\nabla_\theta J(\theta)$$

$$s = \beta_2 s - (1 - \beta_2)\nabla_\theta J(\theta) \otimes \nabla_\theta J(\theta)$$

$$\widehat{m} = \frac{m}{1 - \beta_1^t}$$

$$\hat{s} = \frac{s}{1 - \beta_2^t}$$

$$\theta_{next} = \theta - \frac{\eta \widehat{m}}{\sqrt{\hat{s} + \mathcal{E}}}$$

  Where θ is the parameter vector, η is the learning rate, β_1 is the momentum decay, β_2 is the scaling decay, $\hat{s}$ and ε are the elements used for scaling down the learning rate by applying RMSProp idea, $\widehat{m}$ is the element used for scaling down the learning rate by applying the momentum method, and J(θ) is the cost function. The

momentum decay β_1 is typically set to 0.9, and the scaling decay β_2 is typically initialized to 0.999 [10].

2.3.4. Convolutional Neural Network

2.3.4.1. Overview

CNN was first introduced in the 1980s as the name "Neocognitron" by Kunihiko Fukushima in his paper. The CNN is designed in order to learn spatial hierarchies of the features by a sequence of layers such as convolutional layer, pooling layer, and fully connected layer [24]. In detail, CNN model extracts the small low-level features from images such as line and curves to combine them into higher-level features through a sequence of convolutional layers and pooling layers, and then these intermeditate-level features are used to identify the objects.

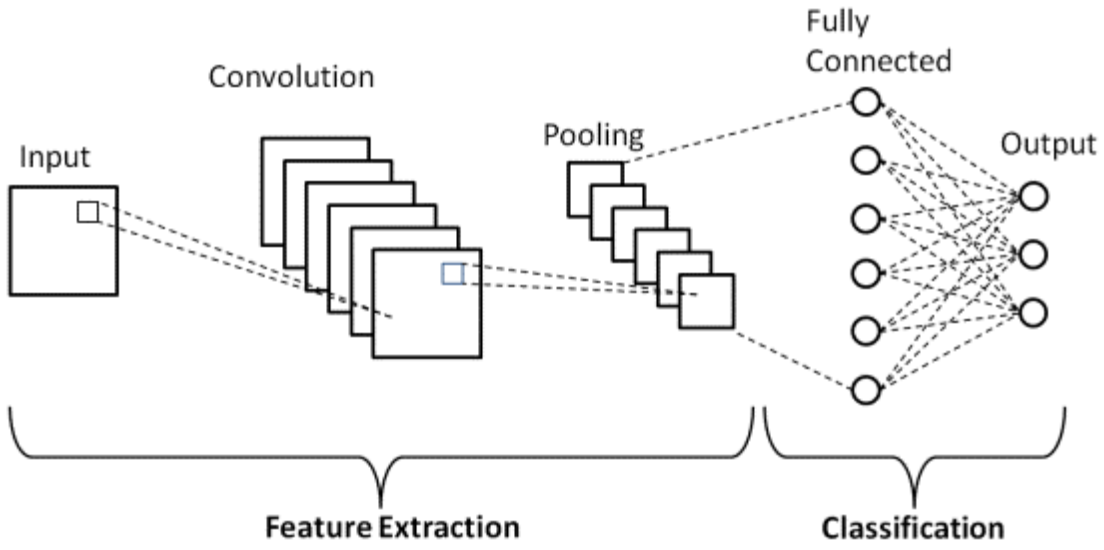


Figure 2.16. CNN [25]

- Convolutional layer

  Convolutional layer is the vital component of CNN, it is designed based on the mathematical operation called "convolution" to extract the outstanding features and combine them into more complex patterns. Particularly, this convolution process

focusses around a small 2D matrix called kernel or filter, it slides along the height and width of the input data to do a multiplication, which is a dot product between filter matrix and filter-sized patch of the data. After multiplication, the result is summed up into a single value.

The input data is normally 2-dimensional for CNN to extract features by stacking the filters along the depth of the input data. Filter size commonly ranges from 32 to 512.

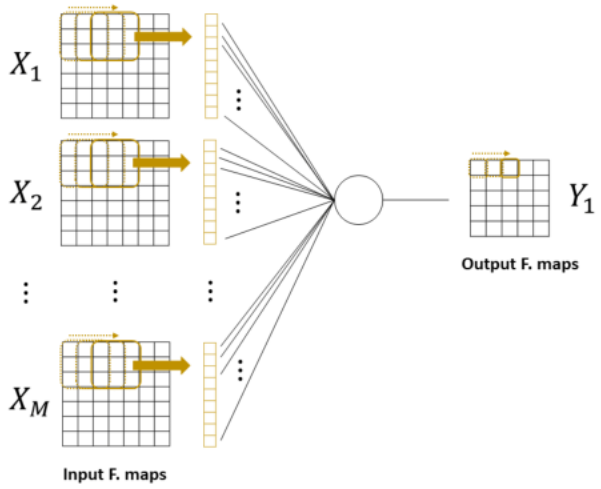


Figure 2.17. Convolutional layer [25]

Besides, extracting the specific features, convolutional layer offers parameter sharing scheme, which means all neuron in an output feature map share the same set of parameters. As a result, this scheme significantly decreases the number of parameters in the CNN

- Pooling layer.

  Pooling layer is connected to the output feature map of convolutional layer in order to shrink the dimensionality of the feature map, leading to the decrease in computational load and number of parameters. As a result, this strategy prevents the

overfitting risk of the model and reduces computational cost during training process [24].

There is a variety of methods to implement pooling, but in most CNN, max-pooling layer with kernel size 2x2 and stride 2 is used. Max-pooling layer slides along height and width of the feature map to take the maximum value in each pooling section.

2.3.5. Recurrent Neural Network

2.3.5.1. Overview

RNN concept was first introduced by David Rumelhart in his paper *Learning representations by back-propagating errors* in 1986. In the next few years, one of the most famous architectures in RNN's family was invented: LSTM architecture. Nowadays, since the growth of nature language processing, time series processing in many aspects of human's life such as: business, commerce, media, industry … RNN, encoder-decoder and attention mechanism are widely in use.

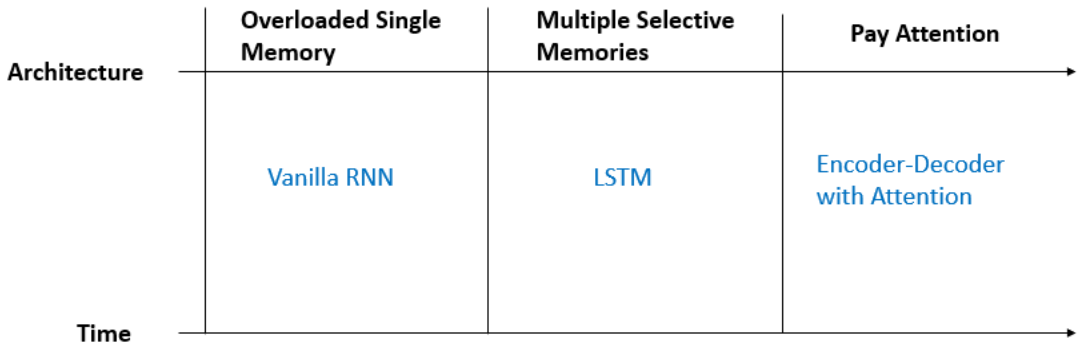


Figure 2.18. The developments of RNN [26]

A simple RNN is a recursive pointing structure, which allows previous outputs to be used as a part of the next input.

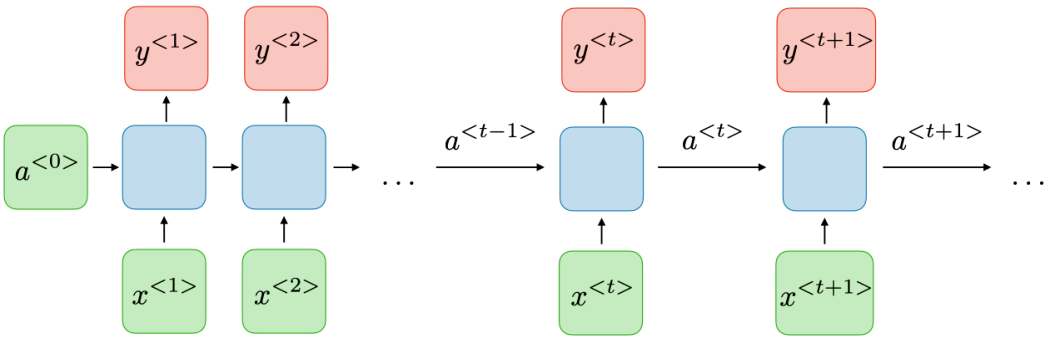


Figure 2.19. A simple RNN [27]

For each timestep, a simple RNN is described as formulars below:

$$a^t = \alpha(W_{aa}a^{t-1} + x^t + b_a)$$

And

$$y^t = \alpha(W_{ya}a^t + b_y)$$

Where a^t, a^{t-1} are the AFs at timestep t and t-1, x^t, b_a is the input and bias at timestep t, W_{aa}, W_{ax} are matrices containing the connection weights for the inputs of the current time step. W_{ya} is a matrix containing the connection weights for the outputs of the previous time step.

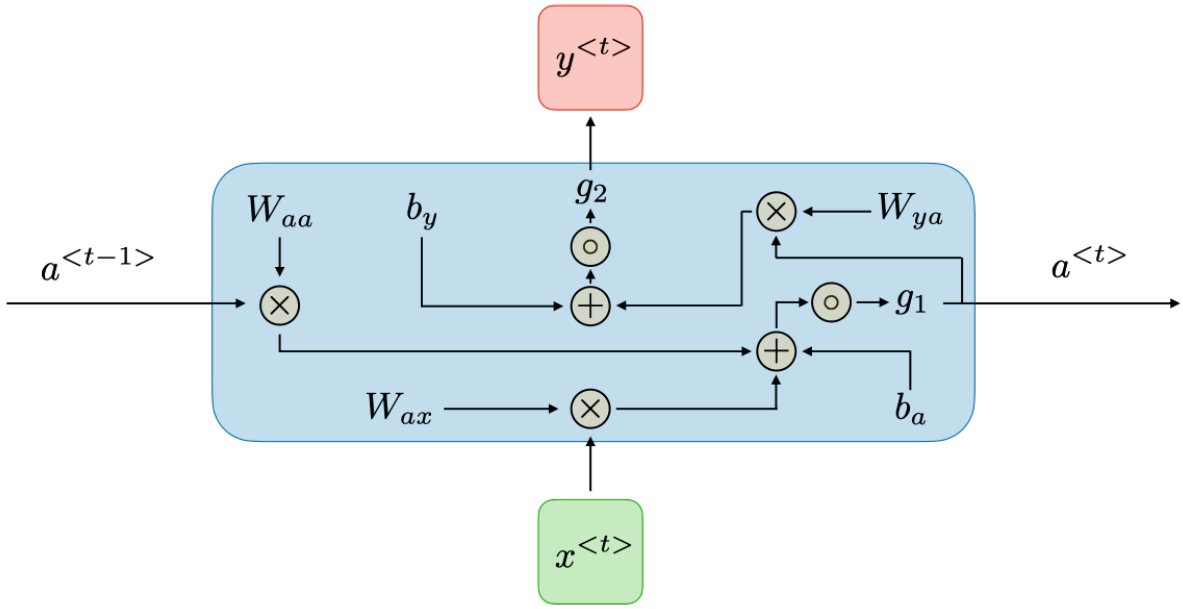


Figure 2.20. Memory cell [27]

Since above mechanism or memory cells, RNN models are well-known in the fields of sequential data, particularly nature language processing and speech recognition.

2.3.5.2. Other RNN architectures

In order to solve simple RNN problems, some new RNN architectures are introduced: Bidirectional recurrent neural networks (BRNN), Gated Recurrent Units (GRU), and Long Short-Term Memory.

- **Bidirectional Recurrent Neural Networks**

As its name, BRNN is constructed from two opposite directional RNN hidden layers to offer a single output. In the detail, rather than running only a forward RNN, BRNN starts a backward RNN at the same time. This architecture not only allows BRNN take information from the past, but also information from the future, and increase the accuracy of the model.

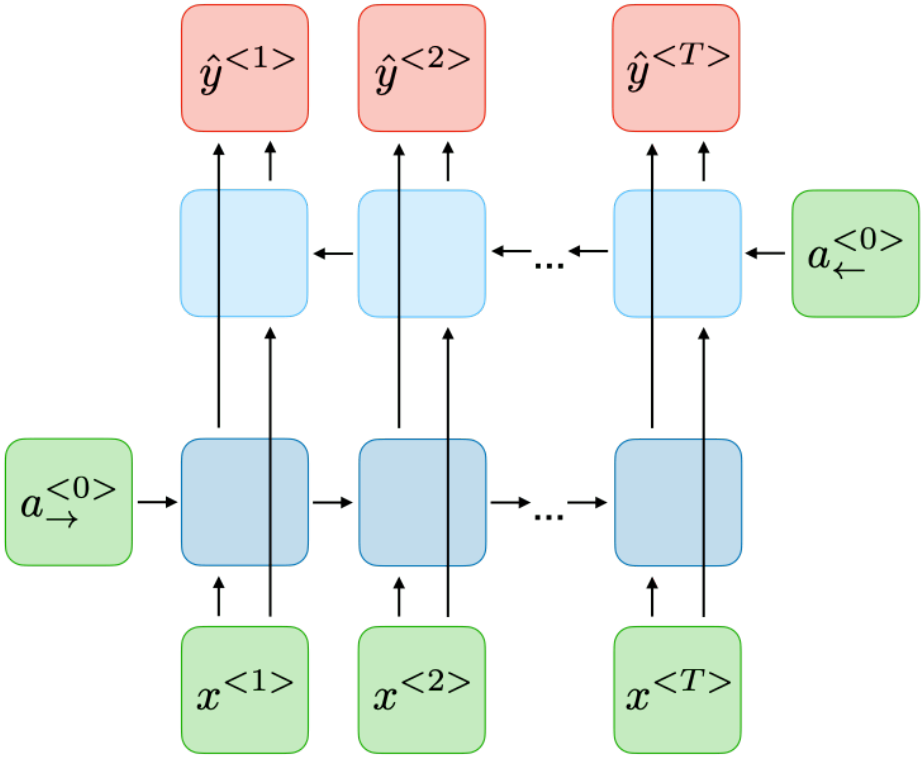


Figure 2.21. BiRNN [27]

- **Gated Recurrent Units**

GRUs are an upgraded version of traditional RNN in an effort to handle longer and more complex sequence data. The key differences between GRUs and traditional RNN are two

mechanism update gate and reset gate, which allow GRUs to learn and keep relevant information. In depth, the update gate decides which past information is passed through new stage, and the reset gate decides how much past information is deleted. This mechanism makes GRUs can store, filter important information for a long time, and avoid short-term memory problem of traditional RNN.

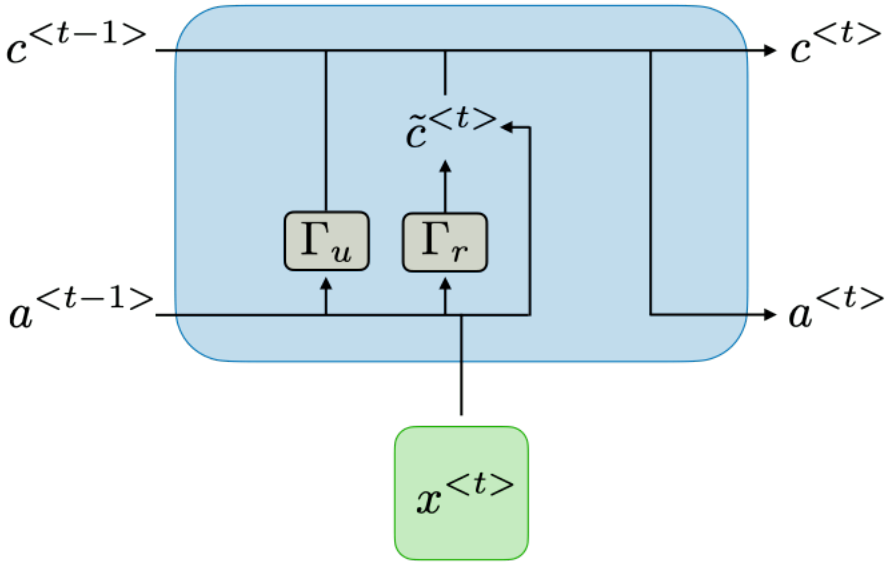


Figure 2.22. GRU [27]

GRUs cell can be described by below formulars:

- Gate function:

$$\Gamma = \sigma(W_{x^t} + Ua^{t-1} + b)$$

- Gate controller:

$$\tilde{c}^t = \tanh\left(W_c[\Gamma_{reset} * a^{t-1}, x^t] + b_c\right)$$

- Output:

$$a^t = c^t = \Gamma_{update} * \tilde{c}^t + \left(1 - \Gamma_{update}\right) * c^{t-1}$$

- **Long Short-Term Memory**

LSTM is one of the most popular RNN architectures, it is designed to solve vanishing gradient problem. LSTM has some similarities to GRU architecture, but it is more complex than just two update and reset mechanisms. In this thesis, LSTM is used as a backbone architecture, so it will be considered in depth in the next section.

2.3.5.3. Long Short-Term Memory network

LSTM network was introduced by two researchers Hochreiter and Schmidhuber in their paper *Long Short-Term Memory* in 1997 as: "Truncating the gradient where this does not do harm, LSTM can learn to bridge minimal time lags in excess of 1000 discrete-time steps by enforcing constant error flow through constant error carousels within special units. Multiplicative gate units learn to open and close access to the constant error flow." [28], and the network has been refined and popularized by many other researchers until today. In 2015, Google started the wave of using LSTM in speech recognition and released Google Neural Machine Translation system. After that, Apple, Amazon, Facebook, … started applied LSTM in their applications: "Facebook announced in August 2017 that it is using LSTM to do a whopping 4.5 billion translations each day, or more than 50,000 per second. LSTM is also used to improve Apple's Siri and QuickType on nearly 1 billion iPhones." [29].

As mentioned above, LSTM architecture has a quite similar structure to GRU. Particularly, there are three gate mechanisms and a cell state such as: input gate, forget gate, output gate and LSTM cell.

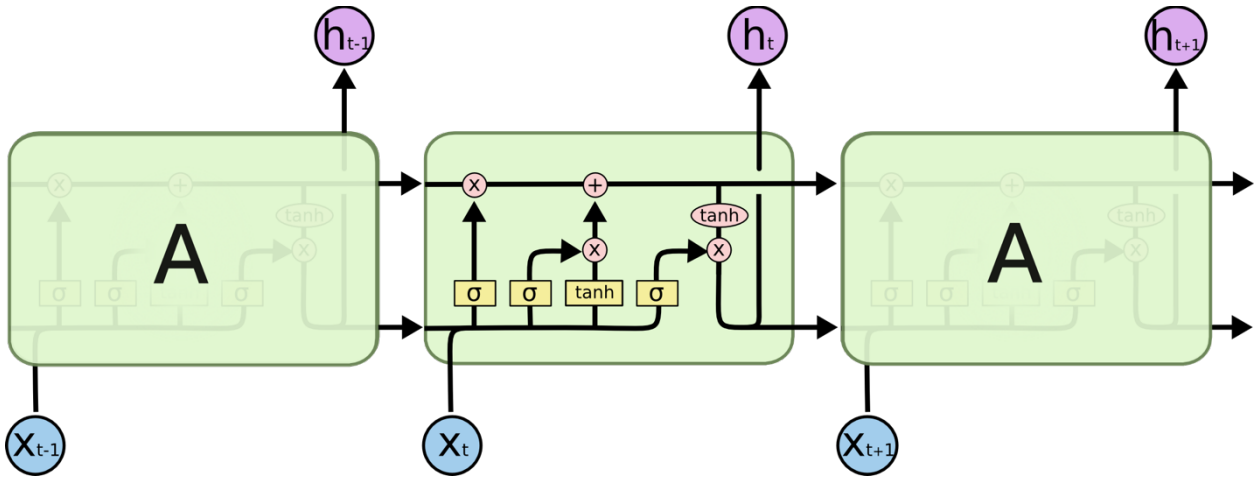
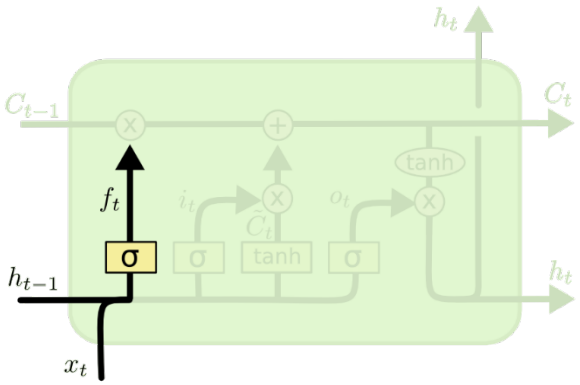


Figure 2.23. LSTM [30]

In order to clarify the function of each component, it is necessary to understand them step-by-step.

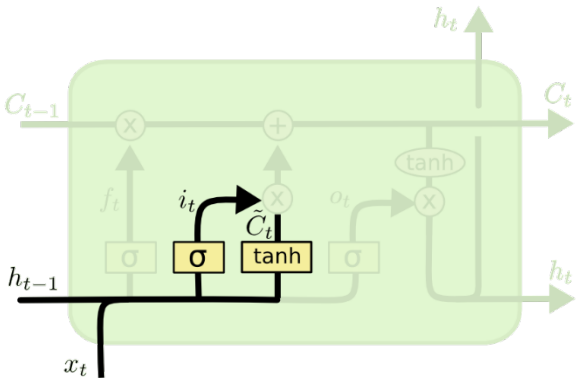
The first step in the process is to determine what information is kept being added and what information is removed by forget gate. This stage is made by a sigmoid layer, which outputs a 1 as “keep” command, and a 0 as “remove” command.



$$f_t = \sigma\left(W_f \cdot [h_{t-1}, x_t] \; + \; b_f\right)$$

Figure 2.24. LSTM cell state 1 [30]

In the next step, there are two parts which is combined into an update to the state. The first part is constructed by a sigmoid layer to decide which values will be updated or not by sigmoid function outputs: a 1 for “update” and a 0 for “not update”. Next, a Tanh layer creates a new value, and this value will be decided to be added into cell or not by the first part.

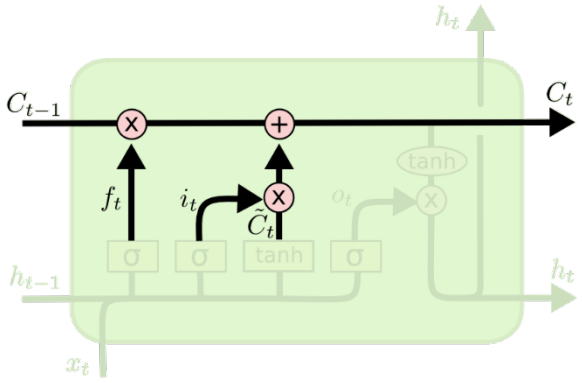


$$i_t = \sigma\left(W_i \cdot [h_{t-1}, x_t] \; + \; b_i\right)$$

$$\tilde{C}_t = \tanh(W_C \cdot [h_{t-1}, x_t] \; + \; b_C)$$

Figure 2.25. LSTM cell state 2 [30]

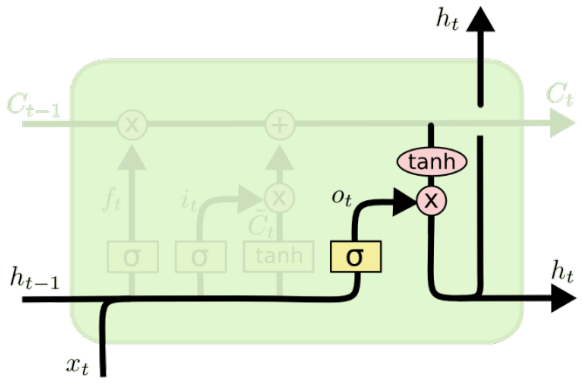
As a result of input and forget gate, the new LSTM cell state is updated.



$$C_t = f_t * C_{t-1} + i_t * \tilde{C}_t$$

Figure 2.26. LSTM cell state 3 [30]

Finally, the model outputs a filtered version of cell state by using a sigmoid and a Tanh layer, in which the sigmoid function decides which part of previous state and new input will be used, then the tanh layer will use the cell state, and apply pointwise multiplication operation with the result from sigmoid function to become the final output.

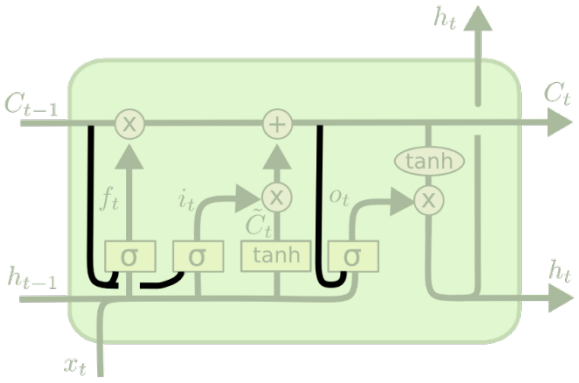


$$o_t = \sigma\left(W_o\left[h_{t-1}, x_t\right] + b_o\right)$$
$$h_t = o_t * \tanh\left(C_t\right)$$

Figure 2.27. LSTM cell state 4 [30]

Since LSTM is a well-known architecture, researchers have been improving from the beginning until today. In this thesis, some notable variants on LSTM will be listed below.

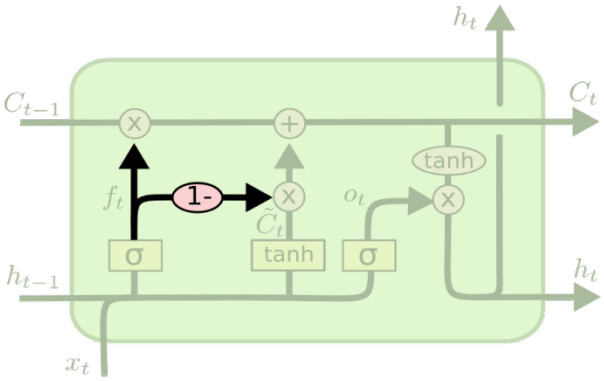
- Peephole connections, which add cell state into each gate layer



$$f_t = \sigma\left(W_f \cdot [\boldsymbol{C_{t-1}}, h_{t-1}, x_t] \; + \; b_f\right)$$
$$i_t = \sigma\left(W_i \cdot [\boldsymbol{C_{t-1}}, h_{t-1}, x_t] \; + \; b_i\right)$$
$$o_t = \sigma\left(W_o \cdot [\boldsymbol{C_t}, h_{t-1}, x_t] \; + \; b_o\right)$$

Figure 2.28. Peephole connections [30]

- Another variation, where forget and input gate are combined



$$C_t = f_t * C_{t-1} + (\mathbf{1} - \boldsymbol{f_t}) * \tilde{C}_t$$

Figure 2.29. LSTM where forget and input gates are combined [30]

According to LSTM's architecture, advantages in handling long sequence data and a large number of variants, I decided to use this architecture in my thesis.

2.4. Attention Mechanism

The AM was first introduced by Dzmitry Bahdanau and his colleague in their paper "Neural Machine Translation by Jointly Learning to Align and Translate" in 2014, it is one of the most significant milestones in the development of deep learning in recent decades. The idea behind the AM is that it imitates the mechanism of cognitive attention. In detail, the cognitive attention is a mechanism inside the brain, which concentrates on the small important pieces of the information and ignore the unnecessary ones rather than focusing on the whole input information. In ML, AM

is implemented to concentrate the special parameters, which contribute the large amount of information and have the specific properties of the input object. As the result, the deep neural networks with attention mechanism have the astonishing performance in handling the very long and complex sequence data, which need to extract a lot of information and have the short-term memory problem such as: NLP, timeseries. Since my thesis is a timeseries problem, these advantages of AM are the key factor why I apply this technology in my project. In addition, AM is now widely used in the other applications of machine learning such as: Residual Attention Network for Image Classification [31], Attentive Language Models [32], and Attention Network for Detecting Healthcare Misinformation [33].

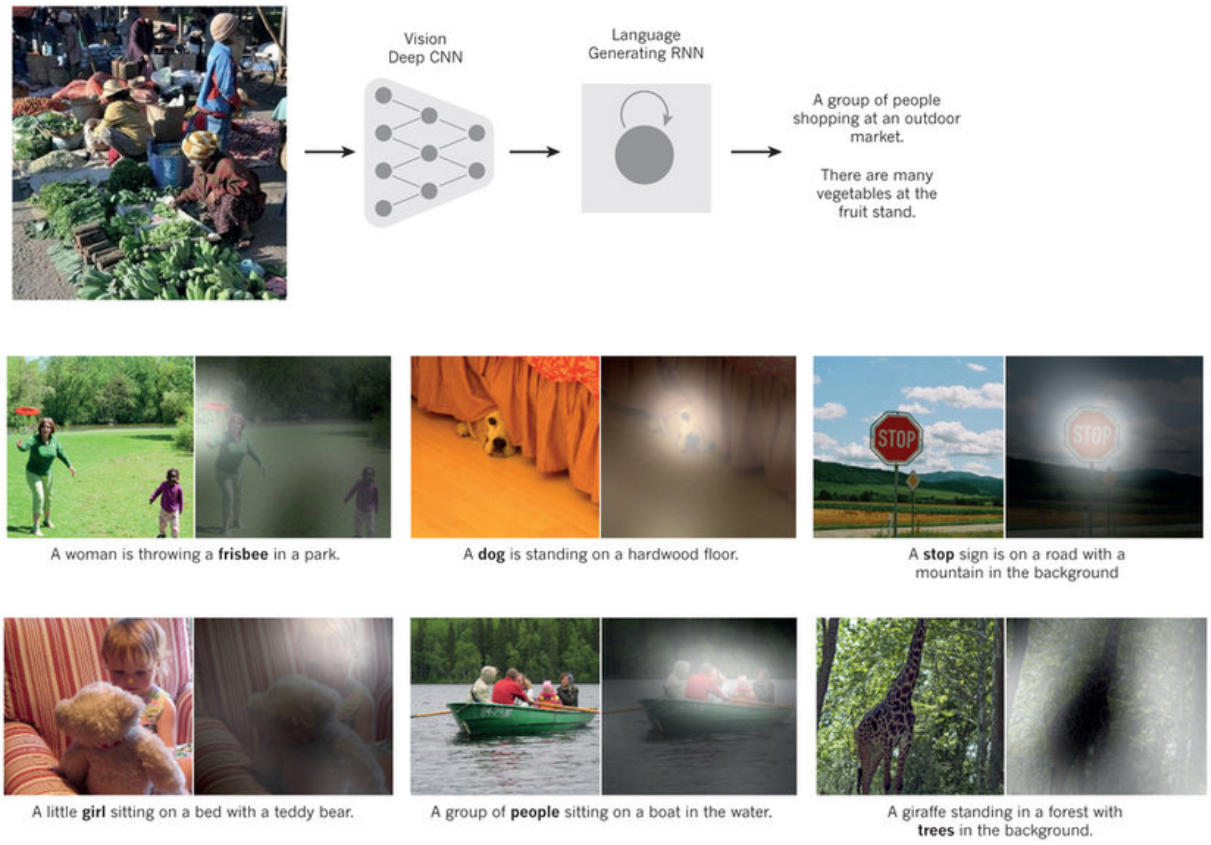


Figure 2.30. Examples of AM [8]

AM has a numerous number of variants, but there are two major types among them, and they are Bahdanau Attention and Luong Attention.

2.4.1. Bahdanau Attention

Bahdanau Attention or sometimes concatenative attention is the first introduced AM, its idea is based on a RNN encoder-decoder framework, in which the encoder is a bidirectional RNN, and the decoder has the responsibility to skim a source sentence during decoding a translation [34]. The operation of Bahdanau Attention is described by the picture below.

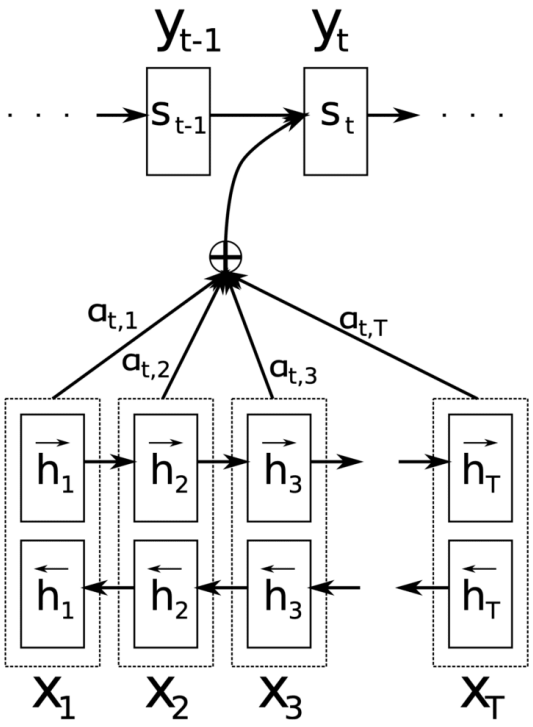


Figure 2.31. Bahdanau Attention [34]

The Bahdanau Attention is defined by the below formulars:

$$c_i = \sum_{j=1}^{T_x} \alpha_{ij} h_j$$

With

$$a_{ij} = \frac{\exp(e_{ij})}{\sum_{k=1}^{T_x} \exp(e_{ik})}$$

And

$$e_{ij} = {v_a}^T \tanh\left(W_a s_{i-1} + U_a h_j\right)$$

Where s_i is a hidden state from previous step, c_i is the context vector, $\alpha_{i,j}$ is the weight, V ,U, H are the weight matrices and h_i is the hidden state from encoder. From these elements, an attention score e_{ij} is calculated [34].

2.4.2. Luong Attention

Luong Attention was proposed in 2015 by Minh-Thang Luong and his colleague in their paper "Effective Approaches to Attention-Based Neural Machine Translation,". Since it is built on the last part Bahdanau Attention, Luong Attention also uses a RNN encoder-decoder framework. However, instead of computing the similarity between encoder's output and the previous decoder's hidden state by concatenating both, Luong Attention offers three different scoring functions. These functions are calculating by encoder's output and the current decoder's hidden state, rather than previous one, and both of them has to have the same dimensionality.

The Luong Attention is defined by the below formulars:

$$\tilde{h}_t = \sum_i \alpha_{t,i} y_i$$

With

$$\alpha_{t,i} = \frac{\exp\left(e_{t,i}\right)}{\sum_j \exp\left(e_{t,j}\right)}$$

And

$$e_{t,i} = {h_t}^T y_i \qquad \text{dot}$$

$$e_{t,i} = {h_t}^T W y_i \qquad \text{general}$$

$$e_{t,i} = v^T \tanh\left(W[h_t; y_i]\right) \qquad \text{concat}$$

Where y_i is the encoder output, $\tilde{h}_t$ is the context vector, $\alpha_{t,i}$ is the weight, W is the weight matrix and h_t is the current hidden state. From these elements, an attention score $e_{t,i}$ is calculated [35].

The first scoring function or "dot product" approach is calculated by computing the dot product of encoder's output and the current decoder's hidden state, then the scores will go through a softmax layer to turn out the final weights. This method gives a similar performance as Bahdanau Attention, but it has a cheaper computational cost. Secondly, the "general dot product" approach is calculated by computing the dot product of encoder's output and the current decoder's hidden state. In addition, the encoder's output sometimes is put in a linear transformation before computing dot product. The result from this approach is applied directly to the decoder prediction rather than going through a softmax layer like Bahdanau Attention or "dot product" approach. The final, "concat" approach has the same mechanism as Bahdanau Attention but using the current decoder's hidden state [35].

2.5. STM32 Microcontroller

The STM32 microcontroller which created STMicroelectronics by is used commonly in many types of devices. There are variety of STM32 microcontroller types in which STM32 Arm Cortex is well-known for its abilities to perform AI model.

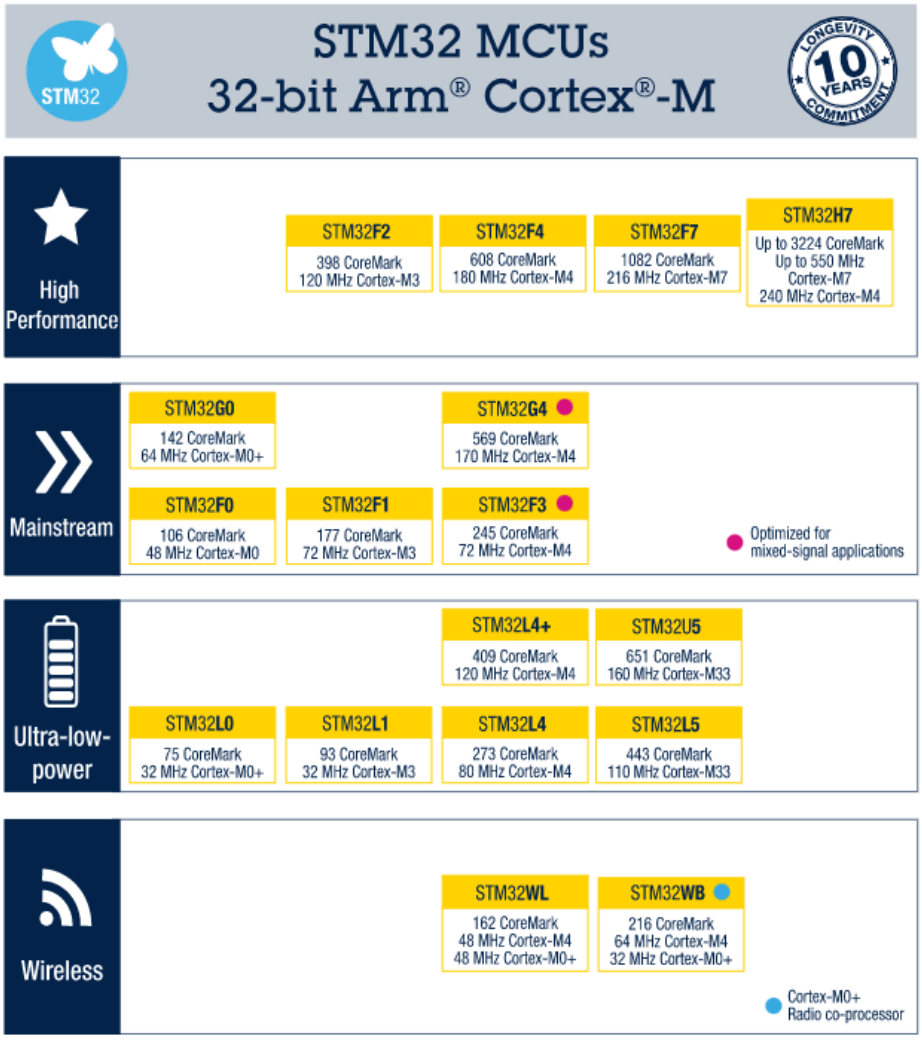


Figure 2.32. STM32 Arm Cortex [36]

CHAPTER 3: DATA

3.1. Overview

This thesis uses the C-MAPSS dataset from NASA's data repository - The Prognostics Data Repository, which is donated and collected from a large number of universities, companies, institutions. The repository concentrates on prognostics sets, in particular timeseries data from normal condition to failed condition.

The C-MAPSS dataset is described as: "C-MAPSS stands for 'Commercial Modular Aero-Propulsion System Simulation' and it is a tool for the simulation of realistic large commercial turbofan engine data … The intent is to identify which flight and when in the flight the fault occurred." [37]. In detail, the C-MAPSS dataset contains 4 different datasets FD001, FD002, FD003, and FD004. Each smaller dataset provides the information of a large number of engines in three different settings. These engines conditions are represented by 21 sensors, and the operational cycles from a starting time to the failed time.

Sub-Dataset	FD001	FD002	FD003	FD004
Engine models for training	100	260	100	249
Engine models for testing	100	259	100	248
Simulation conditions	1	6	1	6
Fault modes	1	1	2	2

Figure 0-1.1. C-MPASS Dataset

In addition, the dataset is chosen as the dataset for the PHM Challenge in 2008, this event made the dataset widely in use until now [38].

3.2. Data process

3.2.1. Data analysis

In order to understand deeply about this dataset, it is necessary to examine it part by part.

- **Engines**

  The below figure will give out an overview of unit numbers of this dataset.

	unit_nr
count	20631.000000
mean	51.506568
std	29.227633
min	1.000000
25%	26.000000
50%	52.000000
75%	77.000000
max	100.000000

	time_cycles
count	100.000000
mean	206.310000
std	46.342749
min	128.000000
25%	177.000000
50%	199.000000
75%	229.250000
max	362.000000

Figure 3.2. Engines detail

This dataset contains 20631 rows of data, belonged to 100 engines. The operational cycles of these engine ranges from 128 to 362, and it normally falls around 206. From this information, the time series data must be set less than 128 frame. In addition, standard deviation is 46.342 between operation cycles, so there is a quite significant difference among each engines' s operation time.

- **Settings**

  There are three different settings in this dataset, it is important to know how these settings affect the engines.

	setting_1	setting_2	setting_3
count	20631.000000	20631.000000	20631.0
mean	-0.000009	0.000002	100.0
std	0.002187	0.000293	0.0
min	-0.008700	-0.000600	100.0
25%	-0.001500	-0.000200	100.0
50%	0.000000	0.000000	100.0
75%	0.001500	0.000300	100.0
max	0.008700	0.000600	100.0

Figure 3.3. Settings detail

As described in the table, there are moderate fluctuations in the setting_1 and setting_2 during the operation, in particular the standard deviation is 0.002187 and 0.000293 respectively. Meanwhile, the setting_3 stays unchanged with standard deviation of 0. Therefore, all three settings are considered to bring no significant effect on the dataset.

- **Sensors**

  The descriptive data of sensors is illustrated as the below table

	count	mean	std	min	25%	50%	75%	max
s_1	20631.0	518.670000	0.000000e+00	518.6700	518.6700	518.6700	518.6700	518.6700
s_2	20631.0	642.680934	5.000533e-01	641.2100	642.3250	642.6400	643.0000	644.5300
s_3	20631.0	1590.523119	6.131150e+00	1571.0400	1586.2600	1590.1000	1594.3800	1616.9100
s_4	20631.0	1408.933782	9.000605e+00	1382.2500	1402.3600	1408.0400	1414.5550	1441.4900
s_5	20631.0	14.620000	1.776400e-15	14.6200	14.6200	14.6200	14.6200	14.6200
s_6	20631.0	21.609803	1.388985e-03	21.6000	21.6100	21.6100	21.6100	21.6100
s_7	20631.0	553.367711	8.850923e-01	549.8500	552.8100	553.4400	554.0100	556.0600
s_8	20631.0	2388.096652	7.098548e-02	2387.9000	2388.0500	2388.0900	2388.1400	2388.5600
s_9	20631.0	9065.242941	2.208288e+01	9021.7300	9053.1000	9060.6600	9069.4200	9244.5900
s_10	20631.0	1.300000	0.000000e+00	1.3000	1.3000	1.3000	1.3000	1.3000
s_11	20631.0	47.541168	2.670874e-01	46.8500	47.3500	47.5100	47.7000	48.5300
s_12	20631.0	521.413470	7.375534e-01	518.6900	520.9600	521.4800	521.9500	523.3800
s_13	20631.0	2388.096152	7.191892e-02	2387.8800	2388.0400	2388.0900	2388.1400	2388.5600
s_14	20631.0	8143.752722	1.907618e+01	8099.9400	8133.2450	8140.5400	8148.3100	8293.7200
s_15	20631.0	8.442146	3.750504e-02	8.3249	8.4149	8.4389	8.4656	8.5848
s_16	20631.0	0.030000	1.387812e-17	0.0300	0.0300	0.0300	0.0300	0.0300
s_17	20631.0	393.210654	1.548763e+00	388.0000	392.0000	393.0000	394.0000	400.0000
s_18	20631.0	2388.000000	0.000000e+00	2388.0000	2388.0000	2388.0000	2388.0000	2388.0000
s_19	20631.0	100.000000	0.000000e+00	100.0000	100.0000	100.0000	100.0000	100.0000
s_20	20631.0	38.816271	1.807464e-01	38.1400	38.7000	38.8300	38.9500	39.4300
s_21	20631.0	23.289705	1.082509e-01	22.8942	23.2218	23.2979	23.3668	23.6184

Figure 3.4. Sensors detail

Overall, it can be observed that the number of data row is the same for each sensor with 20631 rows, and it is as same as the number of engines' s. This turns out that there is no missing data in sensors data.

In term of standard deviation, sensors numbered 1,10,18, and 19 show that their data stay unchanged during the operation since the standard deviation equal zero. Meanwhile contrary, the fluctuation of sensors numbered 9 and 14 is significantly higher than the rest 2.2e+01 and 1.9e+01 respectively. In contrast, sensors numbered 5, 6 and 16 data illustrate an extremely low in their standard deviation, which is

nearly to zero. These very high and low standard deviations might be the outliners, and they will be inspected in the next step.

- **Correlation**

  In order to understand deeply about the data, it is necessary to consider the correlation between each element. In particular, I plotted a heatmap to show how each element affect the others where the color represents how strong correlation between these elements from dark color to light color. The dark color shows the negative correlation between the elements, in contrast the light color shows the positive correlation.

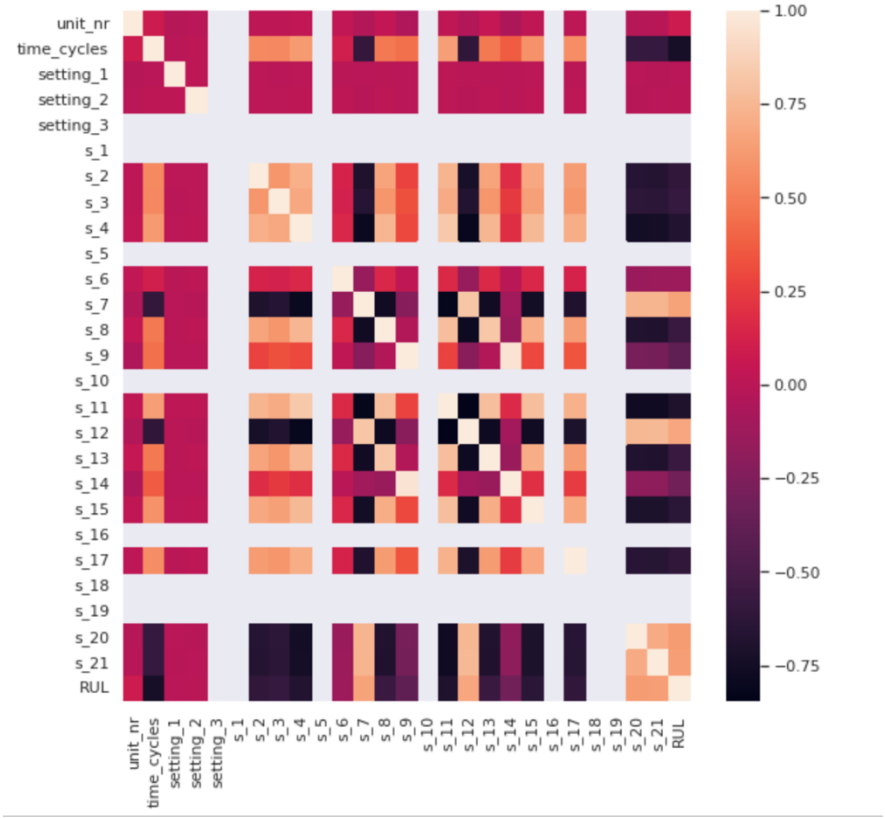


Figure 3.5. Correlation between parameters

As observed from the table, the setting 3 does not show any correlation with the rest. Meanwhile, the setting 1 and 2 keep the same value of correlation with all other parameters, but their correlation is nearly zero. Since, three settings keep almost

unchanged during the operation, and they do not contribute any significant impact on these other elements. It is possible to ignore three settings values in training data.

In term of sensors, a group of sensors numbered 1,10,18, and 19, which keep the same value during the operation, they do not have any correlation with these other elements. In addition, the inspected sensors numbered 5, 6 and 16 shows the same behavior as the previous group that they do not contribute any impact on other sensors. For the purpose of discarding the unnecessary sensors, sensors numbered 5, 6 and 16 will be plotted to examine the correlation between these sensors and targeted value or remaining useful life (RUL) of the engine.

Before plotting, the RUL need to be calculated by the below formular:

$$\mathrm{RUL} = \max \_\mathrm{time\_cycle} - \mathrm{time\_cycle}$$

After calculating the RUL, the relation between RUL and 3 sensors is represented in the below graph

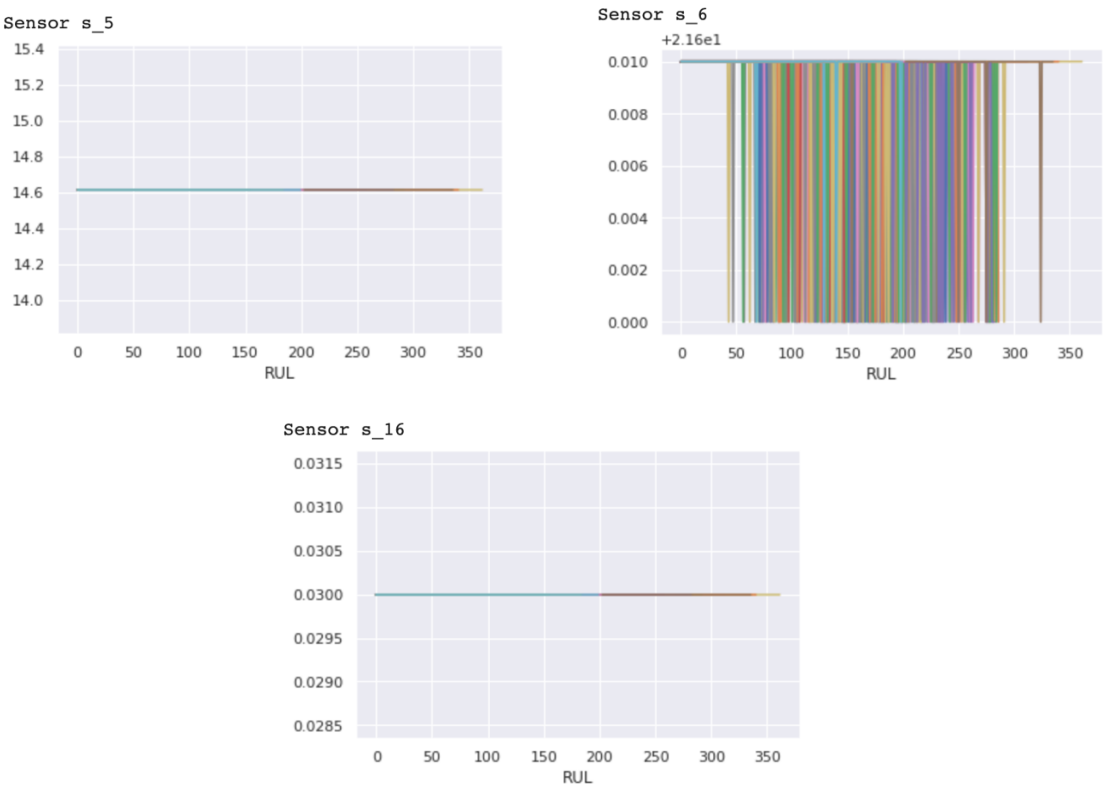


Figure 3.6. Distribution of RUL

Where the y-axis is the value of sensor and x-axis is the value of RUL. The sensors numbered 5 and 16 have a flat line in the graph, it means that there is no contribution from these two sensors. Meanwhile, there is no clear trend in the relation between sensor numbered 6 and RUL of the whole engine. By observing graphs, these three sensors carry no useful information, and they will be excluded.

From the analysis, three setting environments, as well as sensors numbered 1, 5, 6, 10, 16, 18, 19 will be eliminated. This sensor selection is also mentioned by Caifeng Zheng and his colleagues in their paper [39]. Meanwhile, some other models used all the sensor data [40].

3.2.2. Feature Scaling

Feature scaling is one of the most popular techniques in data processing, it is widely applied in statistics and ML. In ML, feature scaling increases the performance of the model, as well as decrease the computational cost. In detail, the input data might fall into a wide range of scale, feature scaling provides some methods to rescale this range of feature data into a similar scale. For

instance, the convergence of GD is faster with applying feature scaling [41]. There are a numerous number of feature scaling methods. However, in this thesis, I will mention two most used feature scaling methods, and they are Normalization and Standardization.

- **Normalization**

  Normalization is a technique that shifts and rescales the features data into a fixed range [0; 1].

  The Normalization is defined by the below formulars:

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

  Where X is a value needed to be rescaled, X_{max} is the maximum value among the feature values, X_{min} is the minimum value among the feature values. Since Normalization is calculated by using the min and the max value, so this technique is also called Min-Max scaling. Applying normalization turns out a small standard deviation in feature data, it makes the data overcome the effect of outliers [42].

- **Standardization**

  Standardization is a scaling method that rescales the feature data into a form of a standard normal distribution, in particular, standardization method determines a mean value as the center and the feature data will be rescaled around the mean by a unit standard deviation.

  The Standardization is defined by the below formulars:

$$z = \frac{x - \mu}{\sigma}$$

Where x is a feature data value, σ is the mean of feature data, μ is the standard deviation, z is the standard score. In ML, Standardization usually rescales the feature data into a standard normal distribution with $\mu = 1$, and $\sigma = 0$ [42].

CHAPTER 4: METHODOLOGY APPROACHES

In predictive maintenance problems, data normally is given in form of time series. In this thesis, some approaches used for C-MAPPS benchmark dataset will be mentioned. These approaches is separated into two categories.

4.1. A single network CNN or LSTM.

One of the methods is using a single network CNN or LSTM. Deep CNN approach is first used by Babu et al. [43] , it outperformed the old approaches such as: SVM, MLP, … Li et al. [44] introduced a deeper CNN network called DCNN, it showed a better result than previous CNN networks. Wen et al. [45] developed a new kind of CNN called ResCNN in 2018, but the authors did not report the result. In term of LSTM, Yuan, Wu, and Lin. [46] presented an overall report about the performances of RNN, GRU and vanilla LSTM on C-PASS dataset. Yu, Kim et al. [47] suggested an architecture based on Bidirectional LSTM (BiLSTM).

Authors	Year	Technique Used	Result Achieved
Babu et al [43]	2016	Deep CNN	RMSE for FD002 (30.29)
Li, Ding, and Sun. [44]	2018	DCNN	RMSE for FD001 (13.32)
Wen et al [45]	2018	ResCNN	No RMSE is reported
Yuan, Wu, and Lin [46]	2018	RNN	MSE (1333.1466)
Yuan, Wu, and Lin [46]	2018	GRU	MSE (1299.4646)
Yuan, Wu, and Lin [46]	2018	LSTM	MSE (1205.4032)
Yu, Kim et al [47]	2019	BiLSTM	RMSE for FD001 (14.47)

Table 4.1. Single learning method used in RUL prediction

Based on figure, performance of vanilla LSTM overcomes other approaches. In order to get a better result, a new configuration is chosen for vanilla LSTM. Vanilla LSTM network in this thesis consists of two LSTM layers, a fixed number of 100 units for the first layer, 50 units for the second layer. Then three fully connected layers are used to predict the RUL, each layer contains 30,20, and 1units. Finally, a dropout for the first LSTM layer with a 0.3 drop rate and 0.1 for the first FCL to deal with overfit.

LSTM architecture	
Window size	30
No. of LSTM layers	2
No. of units	[100,50]
No. of fully-connected layers	3
No. of units	[30,20,1]
Learning rate	0.001
Training epochs	200
Dropout	[0.3,0.1]
Optimizer	Adam

Table 4.2. LSTM architecture configuration

4.2. Hybrid learning method

Another approach is combining some state-of-the-art technologies into a network. Li, Li, and He. [48] suggested a directed acyclic graph (DAG) network that is built by using Convolutional layers and LSTM layers. Muneer et al. [49] proposed a combination between DCNN and AM. Another approach is also proposed by Muneer et al. [50]. These authors combined a LSTM network and attention mechanism.

Authors	Year	Technique Used	Result Achieved
Li, Li, and He [48]	2019	CNN combined with LSTM	RMSE for FD002 (30.29) RMSE for FD003 (19.81)
Muneer et al. [49]	2021	Attention-based DCNN	RMSE for FD001 (11.81)
Muneer et al. [50]	2021	Attention-based LSTM	RMSE for FD002 (12.87) RMSE for FD003 (11.23)

Table 4.3. Hybrid learning method used in RUL prediction

Based on figure, a combination of state-of-the-art technologies shows a better performance than a single network.

4.3. Combination of CNN, LSTM and AM

Realized that the hybrid learning method is a potential method, and inspired by a new paper in 2021 by Yuan Xie and his colleges [51]. An architecture combined CNN-LSTM model and AM is introduced in this thesis. Proposed network consists of four separate components: convolutional 1D layer, BiLSTM layers, a custom AM layer, and Fully Connected Layers (FCLs).

- Convolutional 1D layer

  Instead of normal use convolutional 2D layer, convolutional 1D layer is a better choice to deal with time series data type [52]. Raw data is fed into a Conv1D layer and a Maxpooling1D to extract local features.

- LSTM layers

  LSTM layers are used to encode extracted local features into temporal patterns. These patterns are presented in form of a tensor which is sent to AM.

- Attention Mechanism

A custom AM takes input as a 3D tensor with shape (batch_size, time_steps, input_dim) from LSTM layers hidden states, then calculating Luong attention scores, and returning a 2D tensor with shape (batch_size, unit).

- FCLs to predict the RUL.

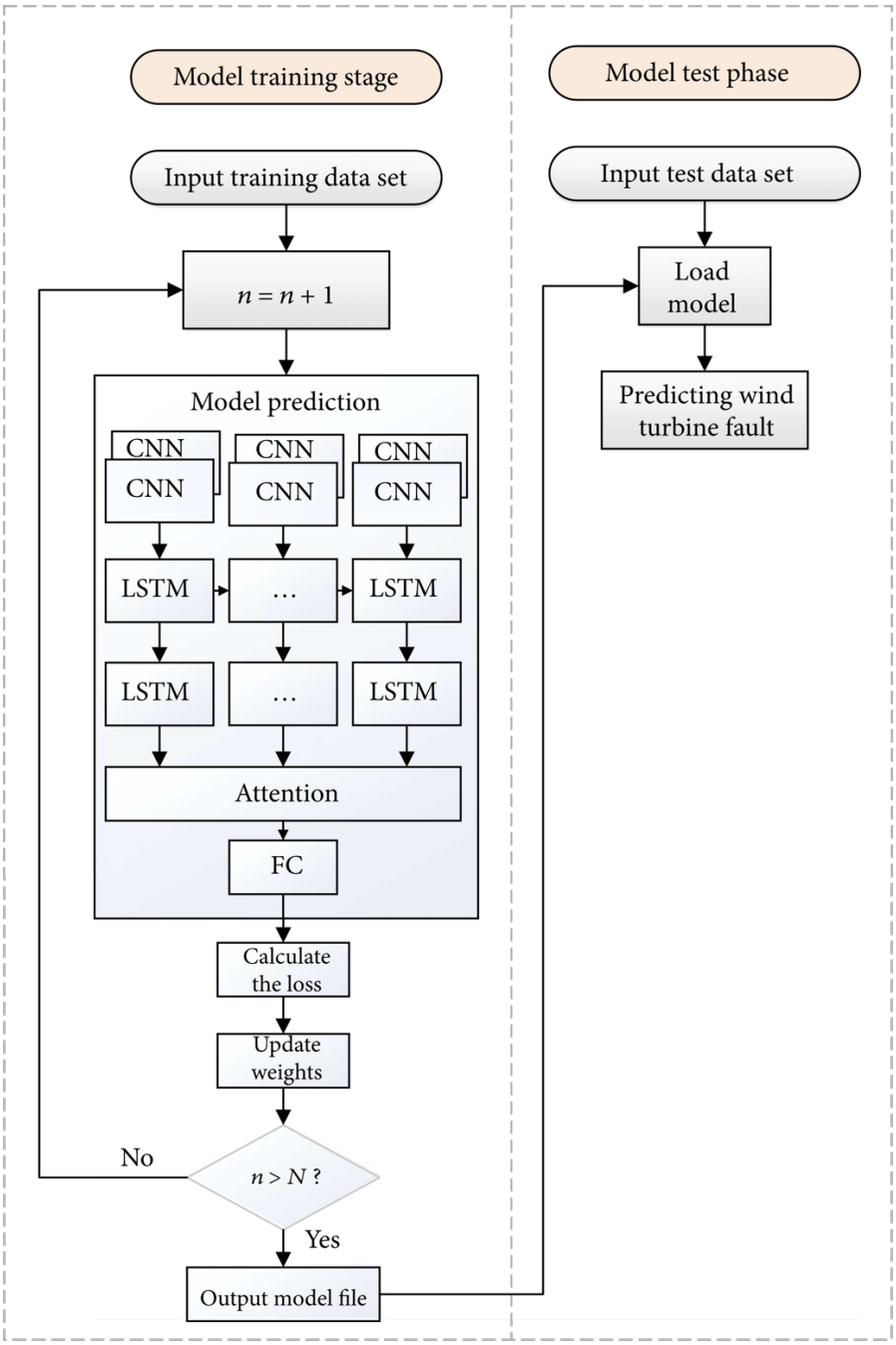


Figure 4.1. Attention based LSMT-CNN architecture

CHAPTER 5: IMPLEMENTATION

5.1. Execution environment

Keras [53] a high level API is used to run on popular ML frameworks such as: Tensorflow, Theano, and Caffe [54] [55] [56]. Due to the need of powerful hardware, Google Colab is considered as the execution environment with GPU of NVIDIA TESLA P100 16GB, 27.3 GB for memory, and 168 GB for the local storage. After training, the model will be embedded into a STM32F767 microcontroller to test the practical performance.

5.2. Configuration

By combining the configurations that are derived from all the measurements and the information from research papers. A topology with a decrease in the number of units each layer is very common in ML architectures. In particular, three hyperparameters (number units of Conv1D, BiLSTM, and fully-connected layer) are chosen. Then, changing two hyperparameters, and keeping one iteratively. By analyzing RMSE while changing the hyperparameters values, the best configuration is selected. Table 5.1 shows the best performance configuration of the proposed CNN-LSTM-Attention network.

CNN-LSTM-Att architecture	
Window size	30
No. of Conv1D layers	1
No. of units	64
No. of BiLSTM layers	2
No. of units	[100,50]
Attention layer	128
No. of fully-connected layers	3
No. of units	[30,20,1]
Learning rate	0.001
Training epochs	200
Dropout	[0.3,0.1]
Optimizer	Adam

Table 5.1. Attention based LSTM-CNN architecture configuration

Window size is chosen by evaluating the performance of different window size values [10,20,30,40,50,60]. Table 5.1 shows experimental results of different window size.

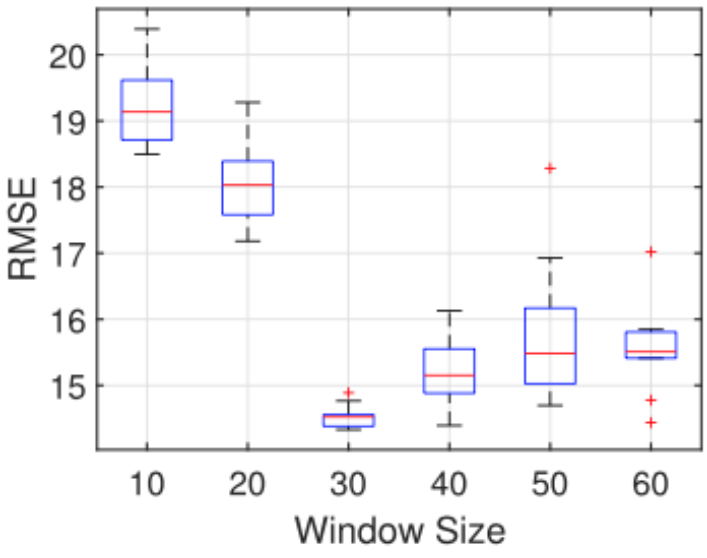
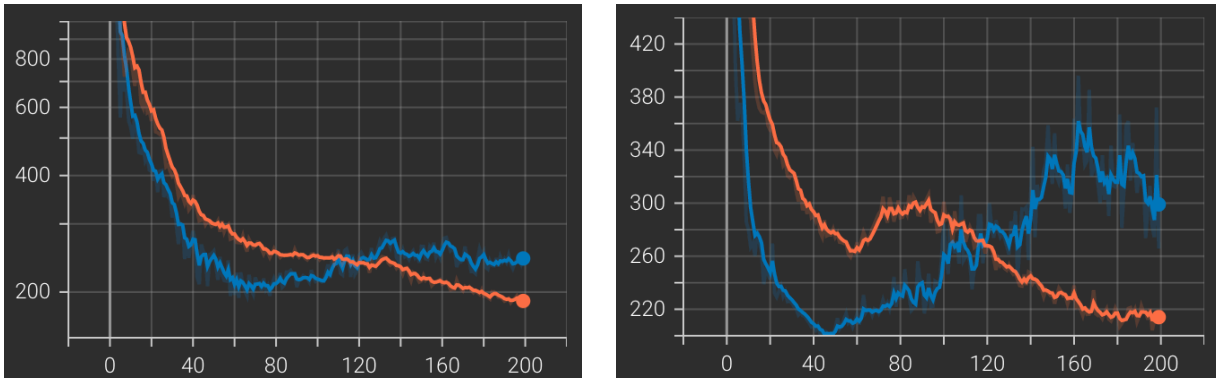


Figure 5.1. RUL comparisons between window sizes

From the beginning an enhancement in the performance is illustrated by increasing the window size. Due to the fact that by increasing the window size, more information is able to be stored internally larger sequences and to learnt possible correlations between "distant" timesteps [57]. However, the larger window size leads to the higher chance of overfitting. The experimental result shows that the value 30 of window size gives a best result in RMSE.

In term of optimizer, Adam and RMSProp are two state-of-the-art algorithms. Both of them are compared in the same configuration of window size, learning rate and number of layers.



a) Adam b) RMSProp

Figure 5.2. Loss of Adam and RMSProp optimizer based on different training epochs in training (blue) and validation (red) set.

The differences between two optimizers can be seen in the figure. Adam has a better result overall two training and validation sets. The learning rate is set at default in this thesis.

5.3. Evaluation

Evaluation is a process to quantify and quality the model's performance, based on the prediction of model with the new data. In order to examine the result from new prediction, performance metrics are used as the key factor in the evaluation framework. In prognostic performance, specially, in Prognostic Health Management (PHM) like this thesis, which is an emerging research field. Performance metrics is the vital element to decide the prognostic method to be certificated and established [58].

In this thesis, two popular performance metrics are used to evaluate the model's prediction such as: Root Mean Square Error (RMSE), and R2 score.

- RMSE

  RMSE is the square root of MSE, and it can be known as the quadratic mean of the differences between predicted values and the true value.

  The RMSE is defined by the below formulars:

$$RMSE = \sqrt{\frac{1}{n}\sum_{i=1}^{n}\left(y_{pred} - y_{true}\right)^2}$$

  Where, n is the number of samples, y_{pred} is the predicted value, and y_{true} is the true value.

- R2 score

  R2 score shows how well the model fits data.

  The R2 score is defined by the below formulars:

$$R2 = 1 - \frac{\sum\left(y_{true} - y_{pred}\right)^2}{\sum(y_{true} - \bar{y})^2}$$

$$\bar{y} = \frac{1}{n}\sum y_{true}$$

  Where, n is the number of samples, y_{pred} is the predicted value, y_{true} is the true value, and $\bar{y}$ is the mean of the observed data.

5.4. Loss function

In ML, during the training process, it is necessary to examine the differences between the current output and the estimated output in order to adjust the weights to improve the performance [59]. The loss function is used as the method to compute the differences. In regression problem, Mean Square Error (MSE) is described as a popular choice "A few basic functions are very

commonly used. The mean squared error is popular for function approximation (regression) problems" [60]. Since the problem of predicting RUL is the regression problem, I have used the MSE loss function.

- MSE

  MSE is the average of the squares of the errors, which is the differences between the predictions and the true targets.

  The MSE is defined by the below formulars:

$$MSE = \frac{1}{n}\sum_{i=1}^{n}\left(y_{pred} - y_{true}\right)^2$$

  Where, n is the number of samples, y_{pred} is the predicted value, and y_{true} is the true value.

CHAPTER 6: RESULTES

An overview of the prediction ability of the five networks with the proposed set of hyperparameters is presented in the following section. Due to the nature of uncertainty inherit in RUL prediction, each model is trained and tested ten times and take the average result. LSTM and CNN+LSTM models are used as the baseline model.

Architecture Used	No. of parameters	Result Achieved
LSTM	59.771	RMSE for FD001 (15.1) R2 for FD001 (0.85)
CNN+LSTM	187.123	RMSE for FD001 (14.74) R2 for FD001 (0.86)
LSTM+Attention	77.411	RMSE for FD001 (14.54) R2 for FD001 (0.87)
CNN+GRU+Attention	95.163	RMSE for FD001 (14.54) R2 for FD001 (0.87)
CNN+LSTM+Attention	118.763	RMSE for FD001 (13.86) R2 for FD001 (0.88)
CNN+BiLSTM+Attention	204.763	RMSE for FD001 (12.96) R2 for FD001 (0.9)

Table 6.1 Average result each architecture

Table 6.1 gives information about the average RMSE of each architecture. In detail, two less complex architecture LSTM and CNN-LSTM archived the highest RMSE 14.98, and 14.74 respectively. LSTM – Attention and CNN-GRU-Attention display the same result, although LSTM architecture normally perform better than GRU, however Convolutional Layers helped

GRU be able to perform equally LSTM. The proposed architecture performs significantly better, especially CNN-BiLSTM-Attention with RMSE 12.96.

Moreover, the RUL predicted by proposed model tends to remain stable during 10 training rounds. In figure 6.1, the RMSE from CNN-BiLSTM-Attention prediction fluctuates around 13, the highest RMSE is 13.88, and the lowest one is 11.72. In contrast, other approaches demonstrate a significant difference between the highest and lowest RMSE. From these experiments, the proposed CNN-BiLSTM-Attention model show the robust and efficiency in its architecture.

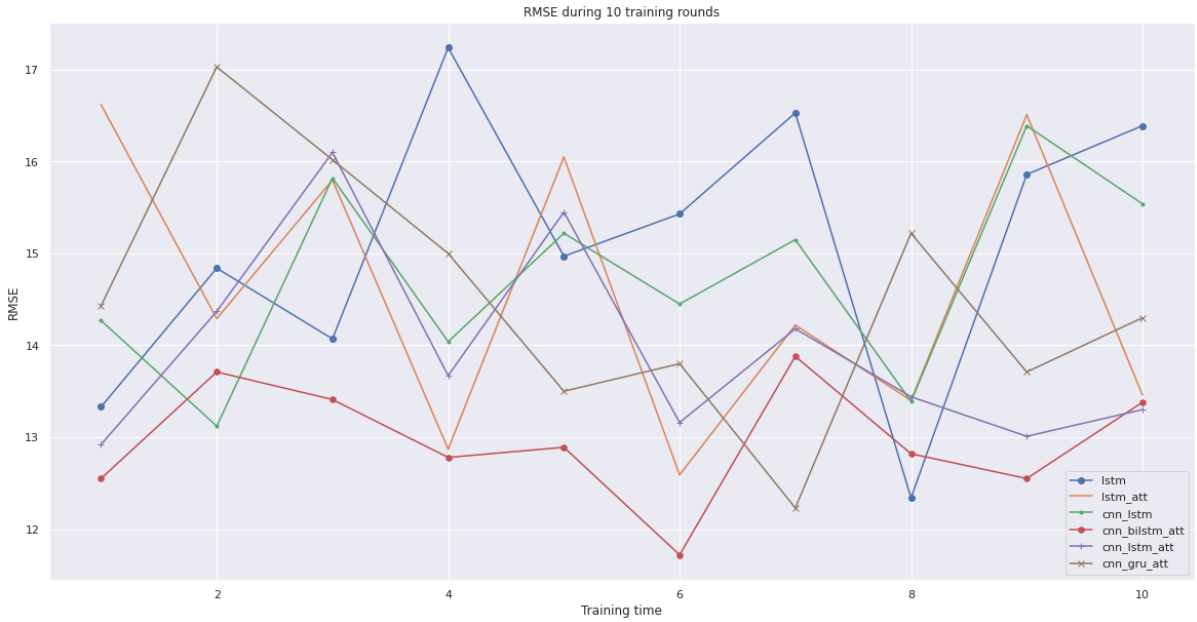


Figure 6.1. RMSE during 10 training rounds

Due to the nature of PHM, the predicted RUL should be the same, or lower than the original one. A high RUL prediction value can make employers care less about maintenance, and it may lead to a breakdown. Meanwhile a low RUL can become a wrong alert, but it is allowable with a permissible error. Figure 6.2 illustrates the best result archived from CNN-BiLSTM-Attention

RMSE = 11.72. By tunning and training process, the predicted RUL line is under the actual RUL, it makes the model perform even better in practical environment.

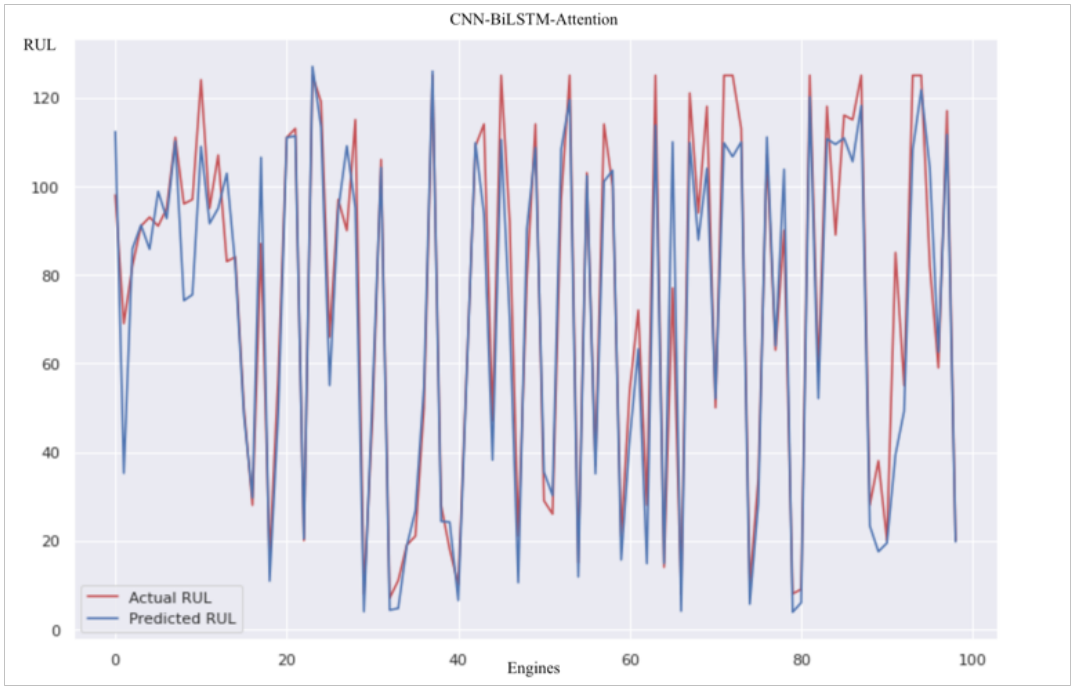


Figure 6.2. CNN-LSTM-Att architecture prediction line graph

Figure 6.3 shows the result of CNN-LSTM-Att in bar graph.

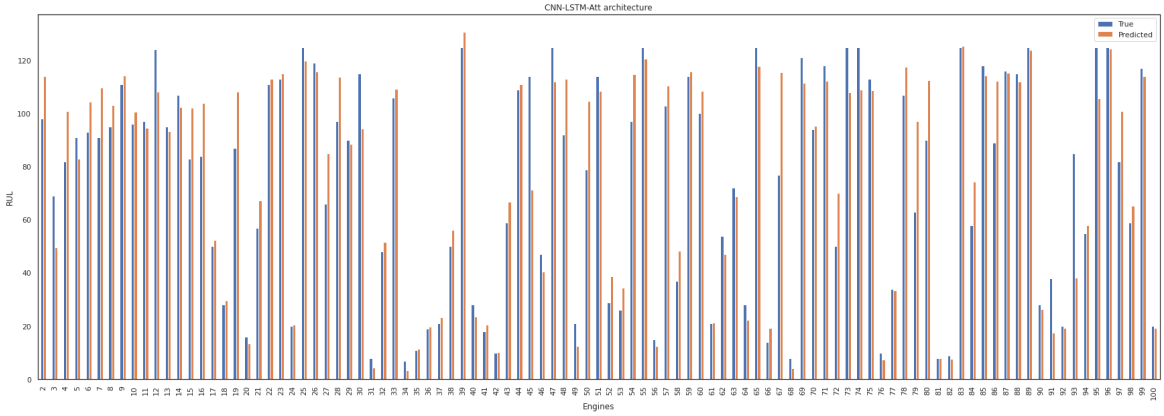


Figure 6.3. CNN-LSTM-Att architecture prediction bar graph

On STM32, a saved weight file is used in order to load model. Because the library does not support customize activation function, instead of using CNN-LSTM-Att architecture, the combination between CNN and LSTM is chosen.

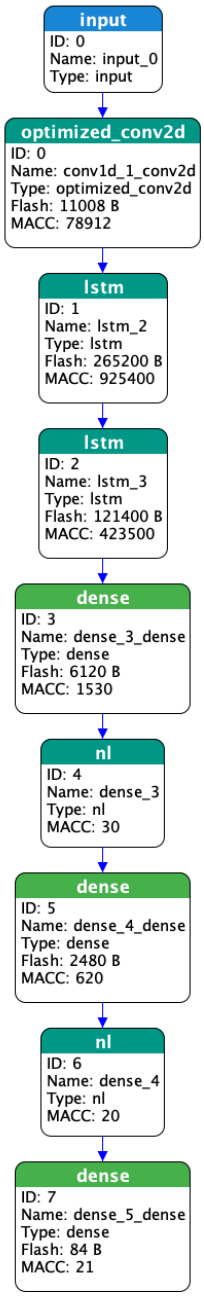


Figure 6.4. CNN-LSTM architecture on STM32

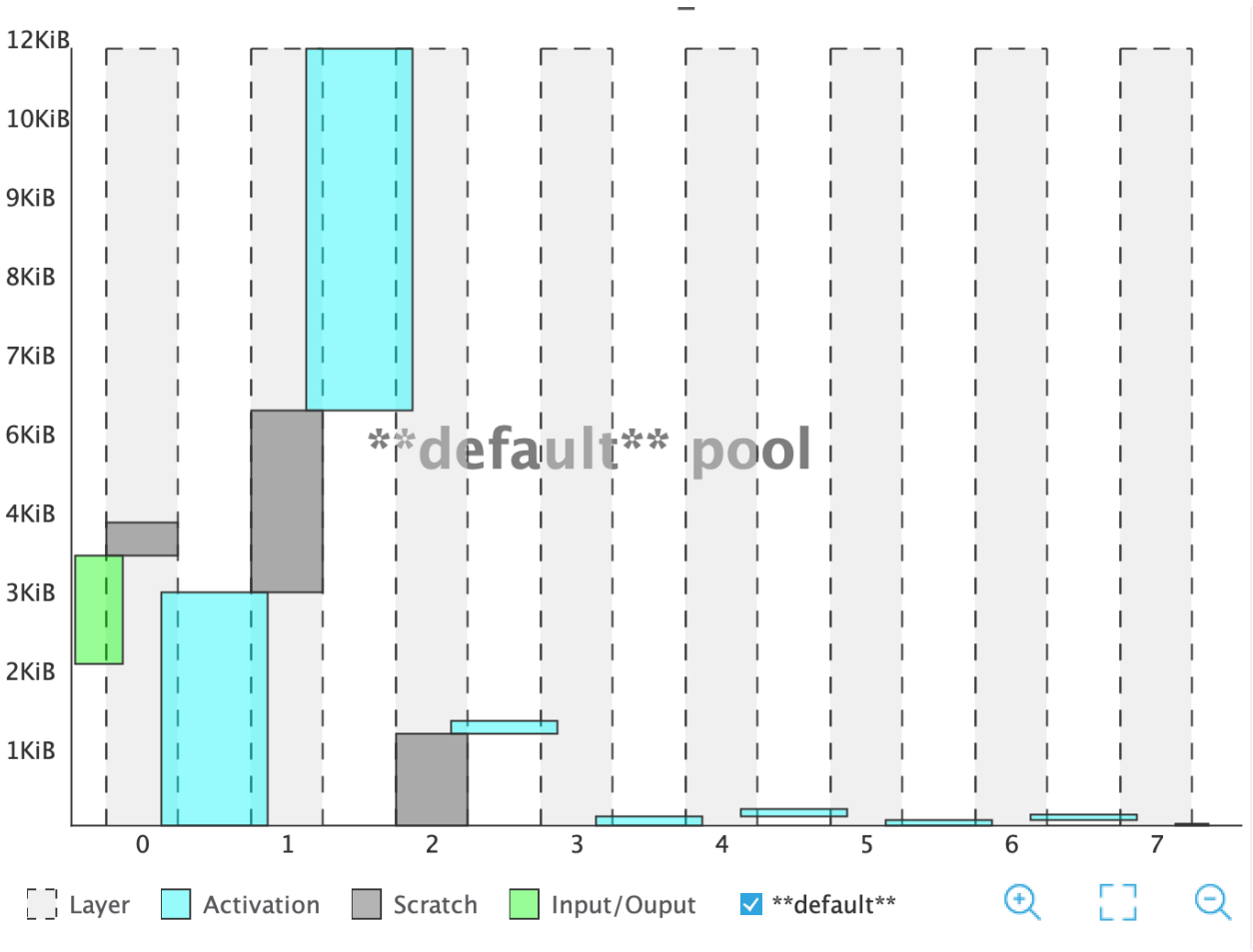


Figure 6.5. Model performance

Used RAM is around 11.7 KiB over 512 KiB internal, used Flash is around 396.77 KiB over 2.00 MiB internal, and the complexity is 1.430.033 MACC. In the detail, two LSTM layers account for 91% the usage of ROM, 5.5% is the percentage of convolutional layer, and the rest is the percentage of three dense layers.

```
Complexity report (model)
------------------------------------------------------------------------------------------
m_id   name              c_macc                    c_rom                     c_id
------------------------------------------------------------------------------------------
1      max_pooling1d_1   ||                 5.5%   |                  2.7%   [0]
2      lstm_2            ||||||||||||||||  64.7%   ||||||||||||||||  65.3%   [1]
4      lstm_3            |||||||           29.6%   |||||||           29.9%   [2]
5      dense_3_dense     |                  0.1%   |                  1.5%   [3, 4]
7      dense_4_dense     |                  0.0%   |                  0.6%   [5, 6]
8      dense_5_dense     |                  0.0%   |                  0.0%   [7]
------------------------------------------------------------------------------------------
macc=1,430,033 weights=406,292 act=11,984 ram_io=0
```

Figure 6.6. Model complexity

The results show a low consumption comparing to the microcontroller's capacity. It proves that the architecture can work well on this microcontroller, and the potential to employ the more complex architecture.

CHAPTER 7: LIMITATIONS AND FUTURE PROPOSAL

7.1. Conclusion

With the development of industry 4.0 and new technologies, the interest for the efficiency in estimating the machine lifetime is increasing. PHM problem is an important steppingstone to understand how and what we should do to avoid compromising the system or human safety.

In this thesis, we did not only analyze LSTM and AM, but also did an in-depth study, which aimed to evaluate many different approaches based on LSTM and compare them to other state-of-the-art architectures.

Firstly, a set of hyperparameters is introduced for turbofan engine RUL prediction. The new configuration for hyperparameters showed a slight improvement in result comparing to other papers using the same LSTM architecture.

Secondly, in our analysis, among proposed approaches, the new CNN-LSTM-Attention architecture has outperformed the others due to the combination of the ability to extract local features by CNN layer, encode them into temporal patterns by LSTM layer, and evaluate the importance of these patterns by AM.

Last but not least, we experienced the progress that the proposed architecture is embedded into a microcontroller. It is an initial step for applying and developing different DL technologies in my further career path.

7.2. Limitations and Future Works

From the beginning, the lack of data is mentioned as one of the reasons that is responsible for poor performances in deep learning. Brophy et al. [61] introduced a new way to generate time series data to enrich the training data by using GANs. However, this new method still needs more research to apply in the future work. The uncertainty in the RUL prediction is an unanswered

question, my thesis enhanced the accuracy and reliability based on ensemble learning method by running ten models and taking the average RMSE. Due to the differences between operating conditions, it is more challenging in predicting RUL in complex system. Therefore, experiments in divergent conditions should be considered in the future.

The proposed architecture showed some improvements in the result comparing to other papers. However, we cannot say it is the best for this problem. Tuning hyperparameters and architecture is a tedious work, but it is necessary to improve the performance of the model. In addition, the custom attention layer of the proposed architecture is not supported by TensorFlow lite and STM32, and it deserves further study to implement.

Bibliography

[1] K. Martin, "A review by discussion of condition monitoring and fault-diagnosis in machine-tools, International Journal of Machine Tools and Manufacture.," *International Journal of Machine Tools and Manufacture,* vol. 34, p. 527–551, 1994.

[2] D. L. D. B. Andrew K.S. Jardine, "A review on machinery diagnostics and prognostics implementing condition-based maintenance,," *Mechanical Systems and Signal Processing,* Vols. Volume 20, Issue 7,, pp. 1483-1510, 2006.

[3] Z. a. C. C. a. T. B. Kang, "Remaining Useful Life (RUL) Prediction of Equipment in Production Lines Using Artificial Neural Networks," *Sensors,* vol. 21, 2021.

[4] M. COPELAND, "Nvidia," Nvidia, 29 July 2016. [Online]. Available: https://blogs.nvidia.com/blog/2016/07/29/whats-difference-artificial-intelligence-machine-learning-deep-learning-ai/.

[5] A. L. Samuel, "Some Studies in Machine Learning Using the Game of Checkers,," *IBM Journal of Research and Development,* vol. 3, pp. 210-229, 1959.

[6] T. M. Mitchell, Machine Learning, McGraw-Hill, 1997.

[7] T. L.-P. I. C. a. D. B. Leslie Kaelbling, "6.036 Introduction to Machine Learning. Fall 2020," Massachusetts Institute of Technology: MIT OpenCourseWare.

[8] B. Y. H. G. LeCun Y, "Deep learning," *Nature,* p. 436–444, 2015.

[9] K. Fukushima, "Neocognitron: a self organizing neural network model for a mechanism of pattern recognition unaffected by shift in position," *Biological cybernetics,* vol. 36, p. 193–202, 1980.

[10] A. GeÌ□ron, Hands-on machine learning with Scikit-Learn, Keras and TensorFlow: concepts, tools, and techniques to build intelligent systems, CA 95472: O'Reilly, 2019.

[11] D. H. G. &. W. R. Rumelhart, "Learning representations by back-propagating errors," *Nature,* vol. 323, p. 533–536, 1986.

[12] P. a. B. B. a. B. A.-L. Probst, "Tunability: Importance of Hyperparameters of Machine Learning Algorithms," arXiv, 2018.

[13] P. a. L. F. A. Dayan, Theoretical neuroscience, Cambridge: MIT Press, 2001.

[14] L. Datta, "A Survey on Activation Functions and their relation with Xavier and He Normal Initialization," arXiv, 2020.

[15] I. G. a. Y. B. a. A. Courville, Deep Learning, MIT Press, 2016.

[16] A. a. K. B. Olgac, "erformance Analysis of Various Activation Functions in Generalized MLP Architectures of Neural Networks," *International Journal of Artificial Intelligence And Expert Systems,* vol. 1, pp. 111-122, 2011.

[17] P. R. a. B. Z. a. Q. V. Le, "Searching for Activation Functions," *CoRR,* vol. abs/1710.05941, 2017.

[18] P. L. K. P. T. L.-P. P. I. C. P. D. Boning, "MIT OpenCourseWare," MIT, 2020. [Online]. Available: https://openlearninglibrary.mit.edu/assets/courseware/v1/9c36c444e5df10eef7ce4d052e4a2ed1/asset-v1:MITx+6.036+1T2019+type@asset+block/notes_chapter_Neural_Networks.pdf.

[19] P. G. Strang, "MATRIX METHODS IN DATA ANALYSIS, SIGNAL PROCESSING, AND MACHINE LEARNING," MIT OpenCourseWare.

[20] Y. Bengio, "Practical Recommendations for Gradient-Based Training of Deep Architectures," in *Lecture Notes in Computer Science*, vol. 7700, Berlin, Springer, Berlin, Heidelberg, 2012.

[21] N. Qian, "On the momentum term in gradient descent learning algorithms," *Neural Networks,* vol. 12, pp. 145-151, 1999.

[22] T. A. W. Mykel J. Kochenderfer, Algorithms for Optimization, MIT Press, 2019.

[23] D. P. a. B. J. Kingma, "Adam: A Method for Stochastic Optimization," arXiv, 2014.

[24] K. a. N. R. O'Shea, "An Introduction to Convolutional Neural Networks," arXiv, 2015.

[25] P. Sharma, "Analytics Vidhya," 1 March 2022. [Online]. Available: https://www.analyticsvidhya.com/blog/2022/03/basic-introduction-to-convolutional-neural-network-in-deep-learning/.

[26] C. Yanhui, "towardsdatascience," 8 March 2021. [Online]. Available: https://towardsdatascience.com/a-battle-against-amnesia-a-brief-history-and-introduction-of-recurrent-neural-networks-50496aae6740.

[27] S. A. Afshine Amidi, "Stanford.edu," [Online]. Available: https://stanford.edu/~shervine/teaching/cs-230/cheatsheet-recurrent-neural-networks#.

[28] S. a. S. J. Hochreiter, "Long Short-Term Memory," *Neural Computation,* vol. 9, no. 8, pp. 1735-1780, 1997.

[29] B. R. Armando Vieira, Introduction to Deep Learning Business Applications for Developers, Apress Berkeley, CA, 2018.

[30] Oinkina, "colah.github.io," 27 August 2015. [Online]. Available: https://colah.github.io/posts/2015-08-Understanding-LSTMs/.

[31] F. a. J. M. a. Q. C. a. Y. S. a. L. C. a. Z. H. a. W. X. a. T. X. Wang, "Residual Attention Network for Image Classification," arXiv, 2017.

[32] Z. a. Y. Z. a. Y. Y. a. C. J. a. L. Q. V. a. S. R. Dai, "Transformer-XL: Attentive Language Models Beyond a Fixed-Length Context," arXiv, 2019.

[33] L. a. S. H. a. T. M. a. M. F. a. W. S. a. L. D. Cui, "DETERRENT: Knowledge Guided Graph Attention Network for Detecting Healthcare Misinformation," Association for Computing Machinery, New York, NY, USA, 2020.

[34] D. a. C. K. a. B. Y. Bahdanau, "Neural Machine Translation by Jointly Learning to Align and Translate," arXiv, 2014.

[35] M.-T. a. P. H. a. M. C. D. Luong, "Effective Approaches to Attention-based Neural Machine Translation," arXiv, 2015.

[36] "st.com," [Online]. Available: https://www.st.com/en/microcontrollers-microprocessors/stm32-32-bit-arm-cortex-mcus.html.

[37] A. a. G. K. Saxena, "C-MAPSS data set," *NASA Ames Prognostics Data Repository,* 2008.

[38] M. a. K. C. a. G. K. a. F. O. Arias Chao, "Aircraft Engine Run-to-Failure Dataset under Real Flight Conditions for Prognostics and Diagnostics," *Data,* vol. 6, p. 5, 2021.

[39] C. a. L. W. a. C. B. a. G. D. a. C. Y. a. Y. Y. a. Z. X. a. L. S. a. H. Z. a. P. J. Zheng, "A Data-driven Approach for Remaining Useful Life Prediction of Aircraft Engines," *2018 21st International Conference on Intelligent Transportation Systems (ITSC),* pp. 184-189, 2018.

[40] G. a. Z. P. a. L. X.-L. e. S. B. a. W. W. a. S. S. a. D. X. a. W. X. S. a. X. H. Sateesh Babu, "Deep Convolutional Neural Network Based Regression Approach for Estimation of Remaining Useful Life," in *Database Systems for Advanced Applications*, Springer International Publishing, 2016, pp. 214--228.

[41] S. a. S. C. Ioffe, "Batch Normalization: Accelerating Deep Network Training by Reducing Internal Covariate Shift," arXiv, 2015.

[42] S. Raschka, "About Feature Scaling and Normalization and the effect of standardization for machine learning algorithms," 2014.

[43] P. Z. X.-L. L. Giduthuri Sateesh Babu, "Deep Convolutional Neural Network Based Regression Approach for Estimation of Remaining Useful Life," in *Database Systems for Advanced Applications*, Springer International Publishing, 2016.

[44] X. L. a. Q. D. a. J.-Q. Sun, "Remaining useful life estimation in prognostics using deep convolution neural networks," *Reliability Engineering & System Safety,* vol. 172, pp. 1-11, 2018.

[45] R. a. S. R. a. H. X. a. S. C. Wang, "Remaining Useful Life Prediction of Rolling Bearings Based on Multiscale Convolutional Neural Network with Integrated Dilated Convolution Blocks," *Shock and Vibration,* vol. 2021, pp. 1-11, 2021.

[46] Y. W. a. M. Y. a. S. D. a. L. L. a. Y. Liu, "Remaining useful life estimation of engineered systems using vanilla LSTM neural networks," *Neurocomputing,* vol. 275, pp. 167-179, 2018.

[47] W. a. K. I. Y. a. M. C. Yu, "Remaining useful life estimation using a bidirectional recurrent neural network based autoencoder scheme," *Mechanical Systems and Signal Processing,* vol. 129, pp. 764-780, 2019.

[48] J. a. L. X. a. H. D. Li, "A Directed Acyclic Graph Network Combined With CNN and LSTM for Remaining Useful Life Prediction," *IEEE Access,* vol. 7, pp. 75464-75475, 2019.

[49] A. a. T. S. M. a. F. S. M. a. A. H. Muneer, "Deep-Learning Based Prognosis Approach for Remaining Useful Life Prediction of Turbofan Engine," *Symmetry,* vol. 13, 2021.

[50] A. a. M. T. S. a. N. S. a. A. R. a. A. I. Muneer, "ata-Driven Deep Learning-Based Attention Mechanism for Remaining Useful Life Prediction: Case Study Application to Turbofan Engine Analysis," *Electronics,* vol. 10, p. 2453, 2021.

[51] J. Z. B. Q. ,. L. M. C. T. a. L. L. Yuan Xie, "Attention Mechanism-Based CNN-LSTM Model for Wind Turbine Fault Prediction Using SSN Ontology Annotation," *Wireless Communications and Mobile Computing,* vol. 2021, 2021.

[52] Z. a. Y. W. a. O. T. Wang, "Time series classification from scratch with deep neural networks: A strong baseline," in *2017 International Joint Conference on Neural Networks (IJCNN)*, 2017, pp. 1578-1585.

[53] E. D. W. H. Michael Ganger, "Double Sarsa and Double Expected Sarsa with Shallow and Deep Learning," *Journal of Data Analysis and Information Processing,* vol. 4, 2016.

[54] M. a. A. A. a. B. P. a. B. E. a. C. Z. a. C. C. a. C. G. S. a. D. A. a. D. J. a. D. M. a. G. S. a. G. I. a. H. A. a. I. G. Abadi, "TensorFlow: Large-Scale Machine Learning on Heterogeneous Distributed Systems," arXiv, 2016.

[55] R. a. A. G. a. A. A. a. A. C. a. B. D. a. B. N. a. B. F. a. B. J. a. B. A. a. B. A. a. B. The Theano Development Team and Al-Rfou, "Theano: A Python framework for fast computation of mathematical expressions," arXiv, 2016.

[56] Y. a. S. E. a. D. J. a. K. S. a. L. J. a. G. R. a. G. S. a. D. T. Jia, "Caffe: Convolutional Architecture for Fast Feature Embedding," arXiv, 2014.

[57] D. a. D. V. F. Bruneo, "2019 IEEE International Conference on Smart Computing (SMARTCOMP)," *On the Use of LSTM Networks for Predictive Maintenance in Smart Industries,* pp. 241-248, 2019.

[58] A. S. S. S. B. S. J. C. Kai Goebel, "Prognostic Performance Metrics," in *Machine Learning and Knowledge Discovery for Engineering Systems Health Management*, Chapman and Hall/CRC, 2012, p. 31.

[59] F. Chollet, Deep Learning with Python, Manning Publications, 2017, p. chapter 1 p.6.

[60] R. D. R. a. R. J. Marks, Neural Smithing Supervised Learning in Feedforward Artificial Neural Networks, MIT Press, 1999, pp. 155-156.

[61] E. a. W. Z. a. S. Q. a. W. T. Brophy, Generative adversarial networks in time series: A survey and taxonomy, arXiv, 2021.