End-to-End time-continuous emotion recognition for spontaneous interactions

Master's Thesis

by

Bobae Kim

Reviewer:	Prof. Dr. Dr.-Ing. Wolfgang Minker
Co-Reviewer:	Privatdozent Dr. Friedhelm Schwenker
Supervisor:	M.Sc. Dmitrii Fedotov

Institute of Communications Engineering
University of Ulm
29 January 2018

I certify that I have prepared this Master's Thesis by my own without any inadmissible outside help.

Ulm, 29 January 2018

(Bobae Kim)

Contents

List of Figures

List of Tables

1 Introduction

1.1 Motivation

Speech emotion recognition has emerged in the area of speech signal research since it can have a significant role in Artificial intelligent such as movie recommendation system that recommends music depends on a user's emotions or computer tutor system that teaches a user with efficiently organized teaching strategy depending on the emotions of the user [1]. While a human can easily recognize other human's emotions, it is more difficult for the machine to recognize emotions. Thus it requires well-designed machine learning algorithms and extraction of correlated information from the speech signal [2]. For solving these issues, new types of neural networks have been developed: recurrent neural networks with long short term memory (LSTM) and convolutional networks (CNN).
The standard neural networks such as multi-layer neural networks (MLP) has a drawback that it can't handle a series of information in the time domain such as speech signals. As a result, RNN has been developed that perform on the series of information in the time domain. However, RNN cannot keep information for a long period. Therefore LSTM that includes memory cells has been developed and it makes an extended period of speech signals to be examined without losing the previous information in a context. The recurrent neural networks with LSTM is getting popular as the emotion recognition structure for this reason.
Extracting characteristics from speech signals requires specialized knowledge to improve the performance of emotion recognition. Psychoacoustic sharpness, jitter, shimmer, Mel-Frequency Cepstral Coefficient, etc. have been used to analyze emotions [3]. CNN is specialized to extract features from the raw signal without specialized knowledge. Thus it is able to use raw speech signals without pre-processing for emotion recognition. For this reason, recent research is focusing on end-to-end speech emotion recognition since it requires little a priori knowledge to extract characteristics [4].
Although the neural network for emotion recognition is getting popular, it is hard to build the network because many parameters are hard to be generalized. This factor makes it is still challenging to build the optimal neural network for emotion recognition.

1.2 Thesis Focus

In this thesis, different types of the speech signal are examined to compare the performance of emotion recognition. The first input type is a set of acoustic Low-Level Descriptors used in AVEC'16 challenge [5]. The second input type is a waveform of the speech signal without pre-processing. The last input type is spectrogram that represents speech signal

in the time domain and frequency domain. Experiments are conducted to find the optimal neural networks for each input type. The performance of three input types is measured on the optimal neural network.

1.3 Related work

The RECOLA database is widely used to recognize emotions in continuous dimensions [6]. From this database, the feature set is used for emotion recognition in [2]. Also, this feature set has been used in AVEC challenge [7] and [5]. Raw speech signal with convolutional neural networks is investigated for speech recognition in [8] and emotion recognition in [4]. Spectrogram with convolutional neural networks for emotion recognition has been examined in [9], [10], and [11]. However, the research on spectrogram mentioned above has focused on emotion recognition in discrete categories of emotions.

1.4 Thesis Outline

Chapter 2 discusses the fundamentals of machine learning algorithm and neural networks. Furthermore, new structures of neural networks such as CNN, RNN with LSTM are presented. Chapter 3 presents the database that is used for this thesis. In chapter 4, experiments and results are described. Chapter 5 concludes this thesis.

2 Background

The neural network is getting popular for emotion recognition. Since the neural network is considered as a sub-part of machine learning, this chapter discusses the basic concept of machine learning and the standard neural network. Furthermore, the specialized neural networks (the recurrent neural networks and the convolutional networks) are also discussed.

2.1 Machine learning

Machine learning is described in three aspects: Task, performance measure and experience.

2.1.1 Task

When we build a computer program, there would be a goal which the computer program is asked to achieve. Such goal is called a task. A computer program can also have various tasks such as self-driving, playing the board game Go. Although there are many types of tasks, we here mention only two main tasks: classification task and regression task.
In regression tasks, a computer program predicts a numerical value from a given input. An example of a regression task is a prediction of a car price. The computer program for this task estimates a car price (numerical value) by the car performance (input) which comprises top speed, fuel economy, seating capacity, and so on. Components of input are called features in machine learning and are represented by a $n \times 1$ vector $\boldsymbol{x}$, where n is a number of features of the input, and input is called example or data point. If we have m number of data points, then the i_{th} given input data can be represented by a $n \times 1$ vector $\boldsymbol{x}_i$, where $1 \leq i \leq m$. An output which is predicted by the learning algorithm is denoted by a scalar $y \in \mathbb{R}$. The output that corresponds to $\boldsymbol{x}_i$ is represented by a y_i. The example of the regression task is depicted in figure 2.1 (left).
In classification tasks, a computer program predicts which category a given input would belong. In other words, a computer program is asked to map a given input to a category. For example, if we ask the computer program to categorize a type of cars by given data, then it tells us if the input is an SUV car or a convertible car or a sedan. An input is denoted by a $n \times 1$ vector $\boldsymbol{x}$, where n is a number of features, same as a regression task. Unlike regression tasks, an output is represented by a $k \times 1$ vector $\boldsymbol{y}$, where k is a number of categories. The output set must be interpreted in vectors before the computer program starts to learn the given data. Suppose we have three categories: an SUV, a convertible car, and a sedan, then three categories must be denoted by vectors. For example, the SUV can be represented as $[100]^T$, the convertible can be represented as $[010]^T$, and the

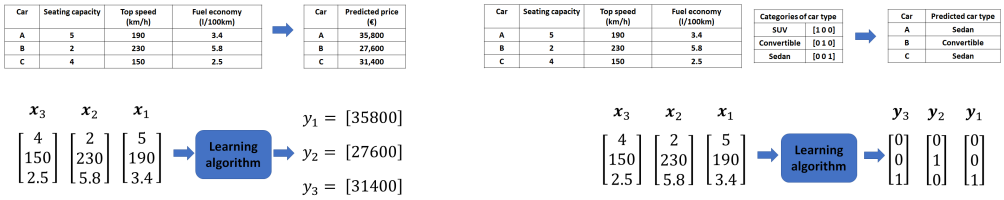

Figure 2.1: Example of a regression task and a classification task.
Left (regression task): A prediction of car prices by seating capacity, top speed, and fuel economy. A dataset has 3 examples (car A, B, and C) and each car has 3 features (seating capacity, top speed, and fuel economy). Thus, the computer program takes $\boldsymbol{x}_1, \boldsymbol{x}_2, \boldsymbol{x}_3$ as the input data, and predicts y_1, y_2, y_3 corresponding to the input. It is the regression task since the outputs of the computer program are numerical values.
Right (classification task): The computer program predicts which category the given input belongs to from the given input data. Before the prediction, the car types were interpreted in vectors. Car A and car C are categorized into Sedan ([001]) and Car B is predicted as a convertible car ([010]). It is the classification task since the outputs of the computer program are the categories.

sedan can be $[001]^T$, where $\boldsymbol{y}^T$ is a transpose of $\boldsymbol{y}$. Therefore, each output of this example is represented by a 3×1 vector $\boldsymbol{y}$. The example of the classification task is depicted in figure 2.1 (right).

2.1.2 Experience

Machine learning algorithms can also be separated into two categories in terms of experience: supervised learning, and unsupervised learning. The main difference between supervised learning and unsupervised learning is the existence of the original data called label and is denoted by $\hat{y}$ that the computer program is supposed to predict. In supervised learning, A dataset that includes labels is separated by a training set and a test set. The computer program predicts values from the training input set and each predicted value is compared to the corresponding label whether a predicted value is close to the label. After learning to solve the given task from the training set, the computer program predicts values from the test set. The key point is that the computer program uses only the training set, and the test set is used after learning. The example of supervised learning is depicted in fig 2.2 (left).
Unlike supervised learning algorithms, unsupervised learning algorithms experience an input dataset without labels and learn to divide the dataset into a cluster of similar examples [12]. Since the computer program doesn't have an answer, it finds the similarity of the training dataset and makes categories. After that, each input of the test dataset is

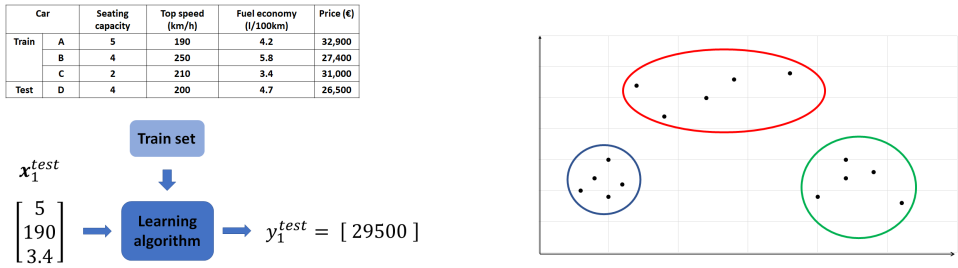

Figure 2.2: Example of a supervised learning (left) and unsupervised learning (right). Left: The supervised algorithm learns on the training set and predicts the price of the test input. The training dataset has 3 examples (car A, B, and C), and each car has 3 features (seating capacity, top speed, and fuel economy) and the original prices (labels).
Right: Outputs are clustered by similar data points. Here three clusters are predicted.

mapped to a given category. The example of unsupervised learning is depicted in figure 2.2 (right).

2.1.3 Performance Measure

Although we have a label set for supervised learning, we can't say yet how close a predicted value is to the corresponding label. Thus we need to quantify how close a computer program predicts a value to the corresponding label or how good it learns to solve the given task. A function that we use to evaluate the performance of the computer program, is called the cost function or loss function that is denoted by $\mathcal{L}$. The value of loss function on the training set is called the training error, and the value of loss function on the test set is called the test error. The computer program tries to minimize the training error while experiencing the training dataset. If the training error is enough to be small, the learning is stopped and the test error is measured. The sum-squared error (SSE) is commonly used as the loss function for the regression tasks and is defined as

$$SSE = \frac{1}{2}\sum_{i=1}^{m}(\hat{y}_i - y_i)^2 \tag{2.1}$$

, where m is a number of data points, $\hat{y}_i$ is the i_{th} label in the dataset, and y_i is the predicted value given i_{th} input data point.

2.1.4 Linear regression

We have mentioned what learning algorithms are, but it is still missing how learning algorithms predict the output. In other words, a function that is used to predict outputs

is not defined yet. The first choice of such a function can be a linear regression algorithm. The linear regression algorithm which takes a vector $\boldsymbol{x} \in \mathbb{R}^n$ as an input is used to solve regression tasks, where $\mathbb{R}^n$ refers the vector $\boldsymbol{x}$ is in n-dimensional real number space and predicts a numerical value y as its output by using a linear function [12]. The predicted output of the linear regression algorithm is defined as

$$y = \boldsymbol{w}^T \boldsymbol{x} = w_1 x_1 + w_2 x_2 \cdots + w_n x_n \tag{2.2}$$

, where $\boldsymbol{w}^T$ is a transpose of a vector $\boldsymbol{w} \in \mathbb{R}^n$ and $\boldsymbol{x}$ is $n \times 1$ input vector. The vector $\boldsymbol{w}$ is a set of parameters or weights that determines how each feature affects the predicted output [12]. If an i_{th} weight is a positive value, then the corresponding i_{th} feature of the input affects the value of its predicted output y to be increased. By contrast, the value of that predicted output y decreases if a feature receives negative weight. If an absolute value of weight is large, then the predicted value gets a large effect. A problem of equation 2.5 is that the linear line always has to pass through the origin. If we add a constant b in equation 2.2, it does not need to pass through the origin and is represented by

$$y = b + w_1 x_1 + w_2 x_2 \cdots + w_n x_n \tag{2.3}$$

. The constant b is called a bias. The bias b is also denoted by w_0 to simplify the linear function. In this case, the weight vector has $n + 1$ elements and it does not map to the input vector $\boldsymbol{x} \in \mathbb{R}^n$. We can also add one extra element x_0 that is always set to 1, to use the same function as equation 2.5. Therefore, we can represent the linear regression function again as

$$y = \boldsymbol{w}^T \boldsymbol{x} = w_0 x_0 + w_1 x_1 + \cdots + w_n x_n \tag{2.4}$$

, where $\boldsymbol{w}$ is an $(n + 1) \times 1$ weight vector and $\boldsymbol{x}$ is an $(n + 1) \times 1$ input vector.
If an input set and a corresponding label set are given, then this linear regression algorithm is categorized as supervised learning. If m number of data points are given, the input set that each input has n number of features and $x_0 = 1$, is denoted by $m \times (n + 1)$ matrix X, and each row of matrix X represents an input. The label set is denoted by $m \times 1$ vector $\hat{\boldsymbol{y}}$. A predicted output set is also denoted by $m \times 1$ vector $\boldsymbol{y}$ and computed as

$$\boldsymbol{y} = \begin{bmatrix} y_1 \\ y_2 \\ \vdots \\ y_m \end{bmatrix} = \begin{bmatrix} x_{1,0} & x_{1,1} & \cdots & x_{1,n} \\ x_{2,0} & x_{2,1} & \cdots & x_{2,n} \\ \vdots & \vdots & & \vdots \\ x_{m,0} & x_{m,1} & \cdots & x_{m,n} \end{bmatrix} \begin{bmatrix} w_0 \\ w_1 \\ \vdots \\ w_n \end{bmatrix} = X\boldsymbol{w} \tag{2.5}$$

, where $x_{i,j}$ is j_{th} input feature of the i_{th} data point, $x_{i,0}$ is always set to 1, and $\boldsymbol{w}$ is the $(n + 1) \times 1$ weight vector that includes a bias. After the values are predicted from the training set, the linear regression algorithm is evaluated by the loss function. The sum squared error (SSE) is used to measure its performance as mentioned in section 2.1.3. The labels of the dataset are compared to the predicted output set by equation 2.1, then we have an equation

$$\mathcal{L} = \frac{1}{2} \sum_{i=1}^{m} (\hat{y}_i - y_i)^2 = \frac{1}{2} \{ (\hat{y}_1 - y_1)^2 + (\hat{y}_2 - y_2)^2 + \cdots + (\hat{y}_m - y_m)^2 \} \tag{2.6}$$

, where m is a number of data points in the dataset, $\hat{y}_i$ is the i_{th} label of the dataset, and y_i is the predicted value corresponding to the i_{th} input. If the error is high, then the difference between the labels and the predicted values is big. If the error is small, then the predicted values are close to the labels. If the error is equal to 0, then the labels and the predicted values are exactly the same. So, the learning algorithm learns to decrease the error. The error is varied by weights, therefore we need to find proper weights that minimize the error. One of the methods to find the proper weights is gradient descent algorithm and it is explained in next section 2.1.5.

2.1.5 Gradient Descent algorithm

The gradient descent algorithm starts with a weight vector $\boldsymbol{w}$ which of initial values are randomly produced, and repeatedly updates the values of the weight vector $\boldsymbol{w}$ as

$$w_i \leftarrow w_i - \alpha \frac{\partial}{\partial w_i} \mathcal{L}(w) \tag{2.7}$$

, where w_i is the i_{th} element of the weight vector $\boldsymbol{w} \in \mathbb{R}^{n+1}$, α is a learning rate, and $\mathcal{L}(w)$ is the loss function. The learning rate α decides how much the derivative affects the update its range is usually from 0.0001 to 0.1 [13]. Let's consider we have only one training data point (an input $\boldsymbol{x}$ and a label $\hat{y}$) to simplify the update rule in equation 2.7. The partial derivative is derived as

$$\begin{aligned}
\frac{\partial}{\partial w_i} J(w) &= \frac{\partial}{\partial w_i} \frac{1}{2} (y - \hat{y})^2 \\
&= \frac{\partial}{\partial w_i} \frac{1}{2} (\boldsymbol{w}^T \boldsymbol{x} - \hat{y})^2 \\
&= 2 \frac{1}{2} (\boldsymbol{w}^T \boldsymbol{x} - \hat{y}) \frac{\partial}{\partial w_i} (\boldsymbol{w}^T \boldsymbol{x} - \hat{y}) \\
&= (\boldsymbol{w}^T \boldsymbol{x} - \hat{y}) \frac{\partial}{\partial w_i} \Big(\sum_{j=0}^{n} w_j x_j - \hat{y} \Big) \\
&= (\boldsymbol{w}^T \boldsymbol{x} - \hat{y}) x_i \\
&= -(\hat{y} - \boldsymbol{w}^T \boldsymbol{x}) x_i \\
&= -(\hat{y} - y) x_i
\end{aligned}$$

Thus, we have the update rule as following

$$w_i \leftarrow w_i + \alpha (\hat{y} - y) x_i \tag{2.8}$$

We can see that this update rule in equation 2.8 is proportional to the gap between the predicted output and the label $(\hat{y} - y)$. Thus, if the gap is big, then weights will be changed with large amount. In contrast, if the predicted output is close to the label, then a small change will be applied to the weights. If the label and the predicted output are matched $(\hat{y} = y)$, there will be no need to change weights.

We have assumed that we had only one data point in the training set, when we derived the update rule. Next, we will apply the update rule to m number of data points, where $m > 1$. The ways that update weights with a training set with multiple data points, are categorized by how many data points are used to update in one step. The three methods of using the gradient descent with the multiple data points are called batch gradient descent, mini-batch gradient descent, and stochastic gradient descent.

Batch gradient descent

In batch gradient descent, weights are updated after all data points are experienced and all predicted values are produced. One cycle that the entire dataset is experienced and all the outputs are predicted by the dataset, is called an epoch. The update is performed only once per epoch and is repeated until the error is enough to be small. The update rule per epoch is performed as following rule:

for $i = 0; i \leq n; i = i + 1$ **do**
  $w_i \leftarrow w_i + \alpha \sum_{j=0}^{m} (y_j - \hat{y}_j) x_{j,i}$
end for

, where m is a number of data points in the dataset, y_j is the predicted output from the j_{th} input data point, $\hat{y}_j$ is the j_{th} element of the label set $\boldsymbol{y} \in \mathbb{R}^m$, and $x_{i,j}$ is the i_{th} feature of the j_{th} input in the $m \times (n + 1)$ input dataset.

Mini-batch gradient descent

In mini-batch gradient descent, the dataset is divided into small batches, and the weights are updated after each batch is experienced. Unlike the batch gradient descent, the weights are updated more than once per epoch. The update weights per epoch is performed as following rule:

for $i = 0; i \leq n; i = i + 1$ **do**
  for $k = 0; k \leq n; k = k + l$ **do**
    $w_i \leftarrow w_i + \alpha \sum_{j=k}^{k+l-1} (y_j - \hat{y}_j) x_{j,i}$
  end for
end for

, where l is a number of data points per batch ($1 < l < m$). A number of data points per batch is usually called a batch size.

Stochastic gradient descent

In stochastic gradient descent algorithm, a update is performed after each data point is experienced. Therefore, weights are updated m times per epoch as following rule:

for $i = 0; i \leq n; i = i + 1$ **do**
  for $j = 0; j \leq n; j = j + 1$ **do**
    $w_i \leftarrow w_i + \alpha (y_j - \hat{y}_j) x_{j,i}$
  end for
end for

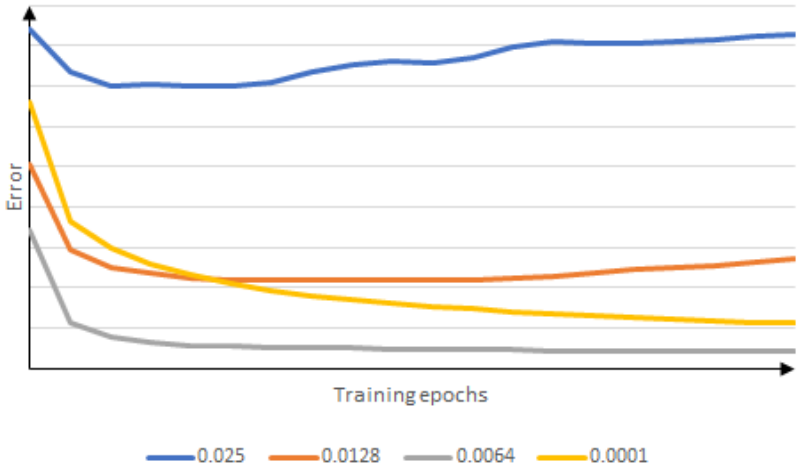

Figure 2.3: Example of various learning rates.
When the learning rate is 0.025, the error is decreased at the beginning and is decreasing at some point. Therefore, this learning rate is considered too high for this network.
When the learning rate is 0.0128, the error is decreased faster than other learning rates but it is stuck at some point. This learning rate is considered high learning rate.
When the learning rate is 0.0064, it reaches the minimum error. It is regarded as the optimal learning rate for this network.
When the learning rate is 0.0001, the error is decreasing slowly. This learning rate is considered low learning rate.

Learning rate

The learning rate α decides how much the derivative of the loss function is affected to update values of weights. The proper learning rate can be found by observing the training error. If the learning rate is high, the training error tends to decay faster, but it gets stuck at some value of error and the error is not minimized after some epochs. If the learning rate is small, the error is decreasing slowly. In other words, the network model should be trained for more epochs with a lower learning rate. Therefore the learning rate should be chosen carefully. This characteristic is changed when the learning range is increased or decreased by the factor of 2 [12]. For example, the learning rate of 0.2 and 0.25 has no significant difference. However, the learning rate of 0.001 and 0.015 has a big difference, even though the gap of 0.05 is same for two cases. The effect of different learning rate is depicted in figure 2.3.

2.2 Neural network

The linear regression algorithm mentioned in the previous section 2.1.4 has a problem that it can learn only linear function. To overcome this defect, a combination of the linear summation in equation 2.4 and a nonlinear function has been proposed known as a neuron [14]. The neuron is an essential component of artificial neural networks or neural network that is motivated by the biological brain as the name implies. A structure of neural networks can be varied by how neurons are connected to each other. In this section, 3 main structure (Multilayer networks, Recurrent networks, and Convolutional networks) are described.

2.2.1 Multilayer networks

In the earliest era of artificial neural networks, a neuron was used to build neural networks. In a neuron, a weighted summation of input is operated ($w_0x_0 + w_1x_1 + \cdots + w_nx_n$ as mentioned in section 2.1.4) and either a nonlinear function or linear function is applied. But this model could not learn more complicating problem such as the XOR function, where $f([0,0], \boldsymbol{w}) = 0, f([0,1], \boldsymbol{w}) = 1, f([1,0], \boldsymbol{w}) = 1$, and $f([1,1], \boldsymbol{w}) = 1$ [12]. However, it can be solved by Multilayer perceptrons (MLPs) also known as multilayer networks. MLPs comprise layers that include many units or cells. There are three types of layers: an input layer, an arbitrary number of Hidden layers, and an Output layer as seen in figure 2.4.

The input layer takes an input data point, and each input feature is connected to every neuron of the next layer. If each neuron of every layer is connected to all the neurons or cells in the previous layer, we say that the network model is fully connected [14]. The layers that are located between the input layer and the output layer are called hidden layers because the network model does not show us the output of each hidden layer and we can see only the output of the output layer. Each neuron of hidden layers and the output layer also include either a nonlinear function or linear function called activation function, and it makes the computer program to solve not only linear functions but also nonlinear functions that the linear regression algorithm is not able to solve.

Feedforward

An input signal taken by the input layer propagates through the hidden layers to the output layer, and the output signal is produced at the end of the network model. This process is called forward pass because its input signal passes through the model from left to right in a forward direction [14]. Suppose we have only one input example which has n features, and a structure of a network model is same as figure 2.4. Therefore, the first hidden layer has 4 neurons, the second hidden layer has 3 neurons, and one output at the end of the output layer. Having only one numerical value at the end of the output layer is used for solving the regression tasks. Each neuron of layers has a weight vector and the weight vector $\boldsymbol{w}_i^{(k)} \in \mathbb{R}^{n+1}$ represents the i_{th} neuron of the k_{th} layer. The input

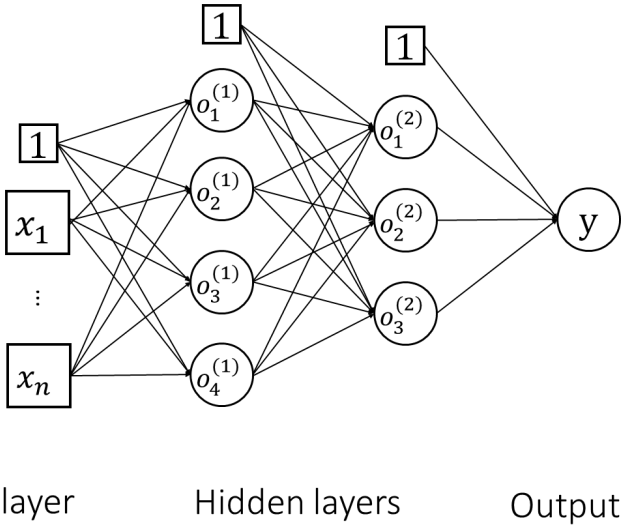

Figure 2.4: A structure of multilayer perceptron with two hidden layers. The first hidden layer consists of 4 neurons, and the second hidden layer has 3 neurons. Each neuron at the hidden layers includes a weighted sum and an activation function. A value is predicted by MLP model at the end of the output layer. $o_i^{(k)}$ is the output of the i_{th} neuron of the k^{th} layer.

layer is not counted as a layer. Thus the network model in figure 2.4 has 3 layers. The j_{th} element of the i_{th} cell is denoted by w_{ij}. First, the input vector $\boldsymbol{x} \in \mathbb{R}^{n+1}$ is taken to the input layer and is passed to the next hidden layer. Then, each neuron computes a weighted sum of the input features by

$$a_i^{(1)} = (w_{i0}^{(1)} x_0 + w_{i1}^{(1)} x_1 + \cdots + w_{in}^{(1)} x_n) = \sum_{j=0}^{n} w_{ij}^{(1)} x_j = \boldsymbol{w}_i^{(1)T} \boldsymbol{x} \tag{2.9}$$

, where $a_i^{(1)}$ is the weighted sum of the input features at the i_{th} neuron of the first layer. As mentioned above, a nonlinear function is applied to $a_i^{(k)}$ and the output of the activation function is the final output of each neuron and is passed to the next layer. The commonly used activation functions are sigmoid function ($f(x) = \frac{1}{1+e^{-x}}$, $f'(x) = f(x)(1 - f(x))$), hyperbolic tangent ($f(x) = \frac{e^{2x}-1}{e^{2x}+1}$, $f'(x) = 1 - f(x)^2$), and REctified Linear Unit(ReLU) as depicted in figure 2.5.
If each neuron has the hyperbolic tangent function, then the output of each neuron is presented by

$$o_i^{(1)} = f(a_i^{(1)}) = \frac{e^{2\boldsymbol{w}_i^{(1)T}\boldsymbol{x}} - 1}{e^{2\boldsymbol{w}_i^{(1)T}\boldsymbol{x}} + 1} \tag{2.10}$$

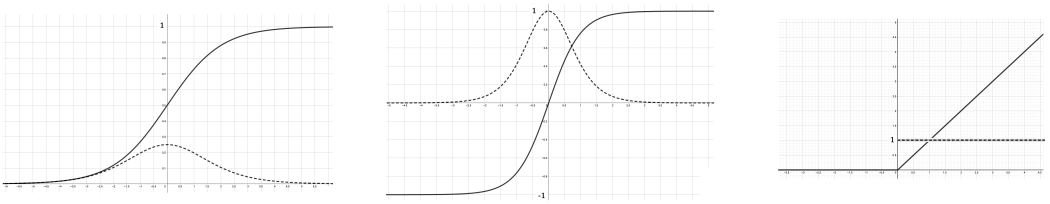

Figure 2.5: Activation functions (solid line) and derivatives (dashed line)
Left: Sigmoid function and its derivative
Middle: Hyperbolic tangent (tanh) and its derivative
Right: Rectified linear unit (ReLU) and its derivative

, where $b_i^{(1)}$ is the output of the i_{th} neuron of the first layer. An output of each neuron is considered as an input feature of the next layer, and bias is also added to the next layer. The output of the i_{th} neuron of the second layer is denoted by

$$o_i^{(2)} = \tanh(a_i^{(2)}) = \frac{e^{2\boldsymbol{w}_i^{(2)T}\boldsymbol{b}^{(1)}} - 1}{e^{2\boldsymbol{w}_i^{(2)T}\boldsymbol{b}^{(1)}} + 1} \tag{2.11}$$

. The values of the second hidden layer are again sent to the following layer that is the output layer, and the same computation is repeated. Thus, the final output is represented by

$$y = w_0^{(3)}b_0^{(2)} + w_1^{(3)}b_1^{(2)} + w_2^{(3)}b_2^{(2)} + w_3^{(3)}b_3^{(2)} = \sum_{k=0}^{3} w_k^{(3)}b_k^{(2)} = \boldsymbol{w}^{(3)}\boldsymbol{b}^{(2)} \tag{2.12}$$

, where $\boldsymbol{w}^{(3)}$ consists of 3 weights of the output layer and a bias $w_0^{(3)} = 0$. The output layer has a linear function as an activation function in regression tasks. Each neuron has the same operator. Therefore the summation and activation are repeated, even if more hidden layers are added. Output of a multilayer network is represented by $y = f^{(k)}(f^{(k-1)}(\cdots f^{(2)}(f^{(1)}(\boldsymbol{x}))))$, where k is a number of layers.

Cost functions

We measure the performance of the network model after obtaining the predicted value y as seen in equation 2.12. The sum squared error (SSE) can also be applied to multilayer perceptron models as a cost function, then we have

$$\mathcal{L} = \frac{1}{2}(\hat{y} - y)^2 \tag{2.13}$$

For m number of data points of the training set, the loss function is given as the following equation

$$\mathcal{L} = \frac{1}{2}\sum_{i=1}^{m}(\hat{y}_i - y_i)^2 \tag{2.14}$$

Backpropagation

We can use gradient descent as described in section 2.1.5 to minimize the loss function of MLPs since MLPs consists of differentiable functions. In MLPs model, partial derivatives are applied in a chain to calculate the gradient descent efficiently, and this algorithm is called backpropagation [15]. The partial derivative in equation 2.7 is converted for the output layer of MLPs to

$$\frac{\partial \mathcal{L}}{\partial w_i^{(3)}} = \frac{\partial}{\partial w_i^{(3)}} \frac{1}{2}(\hat{y} - y)^2 \tag{2.15}$$

$$= \frac{\partial}{\partial w_i^{(3)}} \frac{1}{2}(\hat{y} - \boldsymbol{w}^{(3)}\boldsymbol{b}^{(2)})^2 \tag{2.16}$$

As seen in above equation 2.15, the derivative of the loss function $\mathcal{L}$ with respect to $w_i^{(3)}$ is not available at once. That's why the partial derivatives are efficient for gradient descent of MLPs. Therefore, we can calculate the partial derivate at the output layer by

$$\frac{\partial \mathcal{L}}{\partial w_i^{(3)}} = \frac{\partial \mathcal{L}}{\partial y} \frac{\partial y}{\partial w_i^{(3)}} \tag{2.17}$$

$$= -(\hat{y} - y) \frac{\partial}{\partial w_i^{(3)}} \boldsymbol{w}^{(3)}\boldsymbol{b}^{(2)} \tag{2.18}$$

$$= -(\hat{y} - y) b_i^{(2)} \tag{2.19}$$

For the second hidden layer and the first hidden layer, the partial derivatives are also applied as

$$\frac{\partial \mathcal{L}}{\partial w_{ij}^{(2)}} = \frac{\partial \mathcal{L}}{\partial b_i^{(2)}} \frac{\partial b_i^{(2)}}{\partial a_i^{(2)}} \frac{\partial a_i^{(2)}}{\partial w_{ij}^{(2)}} \tag{2.20}$$

$$\frac{\partial \mathcal{L}}{\partial w_{ij}^{(1)}} = \frac{\partial \mathcal{L}}{\partial b_i^{(2)}} \frac{\partial b_i^{(2)}}{\partial a_i^{(2)}} \frac{\partial a_i^{(2)}}{\partial b_i^{(1)}} \frac{\partial b_i^{(1)}}{\partial a_i^{(1)}} \frac{\partial a_i^{(1)}}{\partial w_{ij}^{(1)}} \tag{2.21}$$

, respectively. After calculating the partial derivatives, weights are updated to minimize the loss function by

$$\boldsymbol{w}_{ij}^k \leftarrow \boldsymbol{w}_{ij}^k - \alpha \frac{\partial \mathcal{L}}{\partial \boldsymbol{w}_{ij}^k} \tag{2.22}$$

, where $\boldsymbol{w}_{ij}^k$ is a j_{th} element(weight) of i_{th} neuron at the k_{th} layer.

Weight initialization

Since our aim of learning is to find the optimal values of weights, we don't know the proper values of weights before training. If the initial values of weights are close to the optimal values, then the network model can find the optimal values faster and efficiently.

In contrast, the network model might have a difficulty to find the optimal values of weights if the initial values are so far from the optimal values. Therefore, weight initialization is also a vital parameter to affect the performance. The commonly used method to initialize weights is Glorot initialization [16]. In MLPs, the partial derivatives that are used to update weights, are computed by

$$\frac{\partial \mathcal{L}}{\partial w_{jk}^{(i)}} = \frac{\partial \mathcal{L}}{\partial o_{jk}^{(i+1)}} \frac{\partial o_{jk}^{(i+1)}}{\partial w_{jk}^{(i)}} \tag{2.23}$$

$$= \frac{\partial \mathcal{L}}{\partial w_{jk}^{(i)}} = \frac{\partial \mathcal{L}}{\partial o_{jk}^{(i+1)}} \frac{\partial o_{j}^{(i)}}{\partial w_{jk}^{(i)}} \tag{2.24}$$

$$= \frac{\partial \mathcal{L}}{\partial w_{jk}^{(i)}} = \frac{\partial \mathcal{L}}{\partial o_{jk}^{(i+1)}} f'(a_j^i) o_j^{(i)} \tag{2.25}$$

. The update rule is affected by the derivative of the activation function. If $f'(a_j^{(i+1)} = 0$, the update rule is stopped. Therefore, Glorot initialization suggests that the initial values of input of activation function ($a_k^{(i)}$) should be near to zero to avoid $f'(a_j^{(i+1)} = 0$, since the derivative of activation functions is usually 1 when an input value is near to zero as seen in figure 2.5. Glorot initialization initializes the weights in the i_{th} layer to be uniformly distributed ranging from $\sqrt{\frac{6}{n_i+n_{i+1}}}$ to $\sqrt{\frac{6}{n_i+n_{i+1}}}$

Validation set

As mentioned in section 2.1.3, the training set is used to train the network, and the test error is measured after training is finished. In other words, backpropagation proceeds on the training set, then the test error is measured after updating weights is done. If the training epoch is increasing, the training error tends to be decreasing. However, the test error has different behavior. The test error is usually decreasing for some epochs and is increasing after some epochs. It is called overfitting if the training error is low and the test error is high. It is hard to find some point that the test error starts being increasing since the test error is measured after the train is finished. Instead of using two datasets, the entire dataset can be split into three datasets. The training set is used to train the network the same way as when we have only two sets. The validation set is used to measure the error during training. After weights are updated at each epoch using the training set, the validation error is computed so that the network can be checked if it is overfitting or not during training. The network stops training at the point where the validation error begins to increase. Then, the test error is computed after the training. The example of overfitting is depicted in figure 2.6.

2.2.2 Recurrent neural network

We have discussed the linear regression algorithm and the multilayer perceptron in the previous sections. The linear regression performs well to solve for linear functions. How-

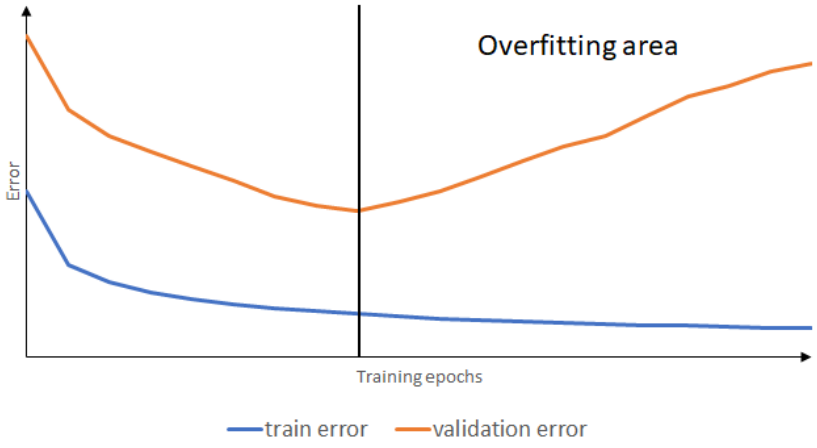

Figure 2.6: Example of overfitting. While the training error (blue line) is decreasing during training, the gap between the training and validation error is increasing after some point. Thus, the network can stop training before going to the overfitting area by checking the training error and the validation error.

ever, this algorithm could not learn to complete more complicating tasks such as the XOR problem. Multilayer perceptron could perform not only linear operators but also nonlinear operators. The problem of MLP is that it can provide only one output from input at once. The predicted values were produced by MLP from the input dataset independently, for example, an input $\boldsymbol{x}_1$ maps to y_1, $\boldsymbol{x}_2$ maps to y_2, etc, where $\boldsymbol{x}_i$ is the i_{th} input data point in the dataset. So, it cannot experience continuous input data, for example, a speech signal $\boldsymbol{x}_1, \boldsymbol{x}_2, \cdots, \boldsymbol{x}_t$ with the time step index t. A recurrent neural network (RNN) is also a type of artificial neural network architecture, but the main difference from MLP is that RNN has a memory so that it can learn the entire input dataset by repeating calculations with the saved sequence in the memory. Therefore, if any types of repetitive structures are included, that type of models is called RNN. The typical types of RNN that are widely used are depicted in figure 2.7 and figure 2.8. In the first recurrent neural network model as seen in figure 2.7, each output of hidden neurons is saved in memory and is summed at the same neuron at the next time step, and the first model predicts an output at each time step. At the time t, an input $\boldsymbol{x}_t$ of the entire input dataset that is a $m \times (n+1)$ matrix X, where m is a number of examples and n is a number of features, is weighted by a $h \times (n+1)$ weight matrix U, where h is a number of neurons at a hidden layer. After the weighted values are summed, an activation function is applied and the values are saved in a memory. The output of the activation function is summed to the values that are weighted by a $h \times h$ weight matrix W from the previous memory. An activation function is applied to these summed values, and the output of the activation function is sent to the output layer. An input of the output layer is also weighted by a $h \times 1$ weight

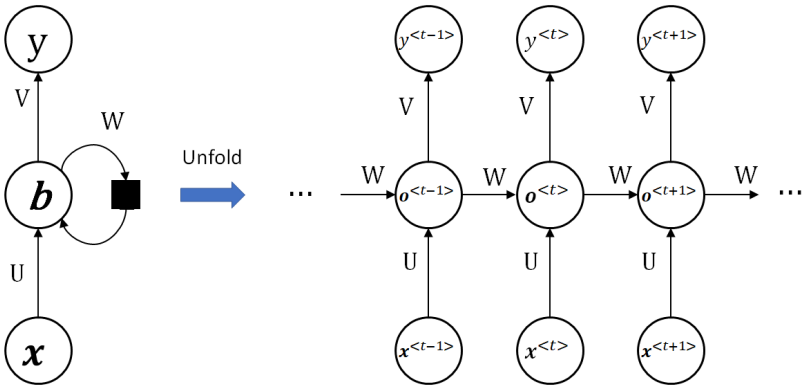

Figure 2.7: Structure of the first RNN example and its unfolding.

vector $\boldsymbol{v}$ which includes a bias, and output $y^{<t>}$ is predicted. In the second RNN model as seen in figure 2.8, each output of the output layer is saved in a memory and is sent to the hidden layer. An input $\boldsymbol{x}_t$ at time t is weighted by a weight matrix U and summed to the values that are weighted by a weight matrix W from the previous memory. The summed values are sent to the output layer after passing through an activation function. At the output layer, an input is weighted by a weight vector $\boldsymbol{v}$. An output is produced by the second RNN model at each time and is saved in memory for the next time step. This model performs worse than the first model due to less information from the past [12].

Feedforward

Suppose we have an $m \times (n+1)$ matrix X as an input dataset and the RNN model with one hidden layer and h neurons at the hidden layer. At time 1, the first row of the matrix X is taken as an input $\boldsymbol{x}_1$. Then, each neuron of the hidden layer has

$$a_i^{<1>} = u_{i0}x_{10} + u_{i1}x_{11} + \cdots + u_{in}x_{1n} = \sum_{j=0}^{n} u_{ij}x_{1j} \tag{2.26}$$

$$o_i^{<1>} = f(a_i^{<1>}) \tag{2.27}$$

, where i refers the i_{th} neuron of the hidden layer thus u_{ij} is an element of i_{th} row and j_{th} column in the $h \times (n+1)$ weight matrix U, x_{1j} is an element of the first row and j_{th} column in the $m \times (n+1)$ input matrix X, and $f(x)$ is an activation function. The

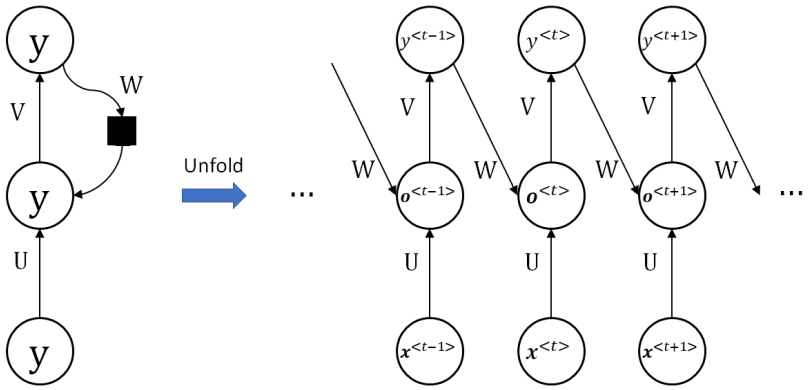

Figure 2.8: Structure of the second RNN example and its unfolding.

output of the hidden layer at the time i is expressed as

$$\boldsymbol{a}^{<1>} = U\boldsymbol{x}_1 = \begin{bmatrix} u_{10} & u_{11} & \cdots & u_{1n} \\ u_{20} & u_{21} & \cdots & u_{2n} \\ \vdots & \vdots & \cdots & \vdots \\ u_{h0} & u_{h1} & \cdots & u_{hn} \end{bmatrix} \begin{bmatrix} 1 \\ x_{11} \\ \vdots \\ x_{1n} \end{bmatrix} = \begin{bmatrix} a_1^{<1>} \\ a_2^{<1>} \\ \vdots \\ a_h^{<1>} \end{bmatrix} \tag{2.28}$$

$$\boldsymbol{o}^{<1>} = f(\boldsymbol{a}^{<1>}) = \begin{bmatrix} f(a_1^{<1>}) \\ f(a_2^{<1>}) \\ \vdots \\ f(a_h^{<1>}) \end{bmatrix} \tag{2.29}$$

$\boldsymbol{o}^{<1>}$ is saved with a bias in a memory and is also sent to the output layer. At the output layer, the output is produced by

$$y^{<1>} = b + v_1 o_1^{<1>} + v_2 o_2^{<1>} + \cdots + v_h o_h^{<1>} = b + \boldsymbol{v}^T \boldsymbol{o}^{<1>} \tag{2.30}$$

, where b is a bias, $\boldsymbol{v}$ is a $h \times 1$ weight vector of the hidden layer. At the next time 2, a calculation at the hidden layer is expressed as

$$\begin{aligned} a_i^{<2>} &= u_{i0}x_{20} + u_{i1}x_{21} + \cdots + u_{in}x_{2n} + w_{i1}o_1^{<1>} + w_{i2}o_2^{<1>} + \cdots + w_{ih}o_h^{<1>} \\ &= \sum_{j=0}^{n} u_{ij}x_{2j} + \sum_{k=1}^{h} w_{ik}o_k^{<1>} \end{aligned} \tag{2.31}$$

$$o_i^{<2>} = f(a_i^{<2>}) \tag{2.32}$$

Before the activation function is applied, the weighted sum of the output $\boldsymbol{o}^{<1>}$ of the hidden layer by the weight matrix W at time 1 is summed to the weighted sum of the

input $\boldsymbol{x}_2$ by the weight matrix U as seen in equation 2.31. Thus, the entire output of the hidden layer at time 2 is expressed as

$$\begin{aligned}\boldsymbol{a}^{<2>} &= \begin{bmatrix} u_{10} & u_{11} & \cdots & u_{1n} \\ u_{20} & u_{21} & \cdots & u_{2n} \\ \vdots & \vdots & \cdots & \vdots \\ u_{h0} & u_{h1} & \cdots & u_{hn} \end{bmatrix} \begin{bmatrix} 1 \\ x_{21} \\ \vdots \\ x_{2n} \end{bmatrix} + \begin{bmatrix} w_{11} & w_{12} & \cdots & w_{1h} \\ w_{21} & w_{22} & \cdots & w_{2h} \\ \vdots & \vdots & & \vdots \\ w_{h1} & w_{h2} & \cdots & w_{hh} \end{bmatrix} \begin{bmatrix} o_1^{<1>} \\ o_2^{<1>} \\ \vdots \\ o_h^{<1>} \end{bmatrix} \\ &= U\boldsymbol{x}_2 + W\boldsymbol{o}^{<1>} \end{aligned} \tag{2.33}$$

$$\boldsymbol{o}^{<2>} = f(\boldsymbol{a}^{<2>}) = \begin{bmatrix} f(a_1^{<2>}) \\ f(a_2^{<2>}) \\ \vdots \\ f(a_h^{<2>}) \end{bmatrix} \tag{2.34}$$

$\boldsymbol{o}^{<2>}$ is also saved in the memory and is sent to the output layer. The output $y^{<2>}$ at time 2 is produced in the same way as equation 2.30, thus we have a predicted value at time 2 as

$$y^{<2>} = b + v_1 o_1^{<2>} + v_2 o_2^{<2>} + \cdots + v_h o_h^{<2>} = b + \boldsymbol{v}^T \boldsymbol{o}^{<2>} \tag{2.35}$$

From the above process, we can generalize the algorithm at any time t ranges from 2 to m, as

$$\boldsymbol{a}^{<t>} = U\boldsymbol{x}_t + W\boldsymbol{o}^{<t-1>} \tag{2.36}$$

$$\boldsymbol{o}^{<t>} = \begin{bmatrix} f(a_1^{<t>}) \\ f(a_2^{<t>}) \\ \vdots \\ f(a_h^{<t>}) \end{bmatrix} \tag{2.37}$$

$$y^{<t>} = b + \boldsymbol{v}^T \boldsymbol{o}^{<t>} \tag{2.38}$$

Loss function

If the SSE is used as a loss function, the predicted value $y_{<t>}$ at time t is compared to the corresponding label $\hat{y}_t$ in the label set $\hat{\boldsymbol{y}} \in \mathbb{R}^m$ by following

$$\mathcal{L}^{<t>} = \frac{1}{2}(\hat{y}_t - y^{<t>})^2 \tag{2.39}$$

and the loss function for the entire dataset is represented by

$$\mathcal{L} = \mathcal{L}^{<1>} + \mathcal{L}^{<2>} + \cdots + \mathcal{L}^{<m>} = \sum_{t=1}^{m} \mathcal{L}^{<t>} \tag{2.40}$$

Backpropagation

To update weights of RNNs, the chain rule can be applied. However, the standard back-propagation for MLPs cannot be directly used, since the values at the end of the hidden

layer in RNNs include the term from the previous time. Backpropagation through time (BPTT) is used in this case because it is more efficient in computation time than other algorithms [15]. The partial derivatives for updating the weight vector $\boldsymbol{v}$ are the same as the standard backpropagation. Therefore we have

$$\frac{\partial \mathcal{L}}{\partial \boldsymbol{v}} = \frac{\partial \mathcal{L}}{\partial \mathcal{L}^{<t>}} \frac{\partial \mathcal{L}^{<t>}}{\partial y^{<t>}} \frac{\partial y^{<t>}}{\partial \boldsymbol{v}} = \frac{\partial \mathcal{L}^{<t>}}{\partial y^{<t>}} \frac{\partial y^{<t>}}{\partial \boldsymbol{v}} \tag{2.41}$$

and the weight vector $\boldsymbol{v}$ is updated as the following rule

$$\boldsymbol{v} \leftarrow \boldsymbol{v} - \alpha \frac{\partial \mathcal{L}^{<t>}}{\partial y^{<t>}} \frac{\partial y^{<t>}}{\partial \boldsymbol{v}} \tag{2.42}$$

, where α is a learning rate. The partial derivative of the weight matrix W is given by

$$\frac{\partial \mathcal{L}^{<t>}}{\partial W} = \frac{\partial \mathcal{L}^{<t>}}{\partial y^{<t>}} \frac{\partial y^{<t>}}{\partial \boldsymbol{o}^{<t>}} \frac{\partial \boldsymbol{o}^{<t>}}{\partial W} \tag{2.43}$$

Since $\boldsymbol{b}^{<t>}$ depends on W and $\boldsymbol{o}^{<t-1>}$ as seen in equation 2.36 and 2.37, the chain rule in equation 2.43 cannot be applied directly. The chain rule of W is extended to

$$\begin{aligned}
\frac{\partial \mathcal{L}^{<t>}}{\partial W} &= \frac{\partial \mathcal{L}^{<t>}}{\partial y^{<t>}} \frac{\partial y^{<t>}}{\partial \boldsymbol{b}^{<t>}} \frac{\partial \boldsymbol{o}^{<t>}}{\partial W^{<t>}} + \frac{\partial \mathcal{L}^{<t>}}{\partial y^{<t>}} \frac{\partial y^{<t>}}{\partial \boldsymbol{o}^{<t>}} \frac{\partial \boldsymbol{o}^{<t>}}{\partial \boldsymbol{o}^{<t-1>}} \frac{\partial \boldsymbol{o}^{<t-1>}}{\partial W} \\
&\quad + \frac{\partial \mathcal{L}^{<t>}}{\partial y^{<t>}} \frac{\partial y^{<t>}}{\partial \boldsymbol{o}^{<t>}} \frac{\partial \boldsymbol{o}^{<t>}}{\partial \boldsymbol{o}^{<t-1>}} \frac{\partial \boldsymbol{o}^{<t-1>}}{\partial W} + \cdots + \frac{\partial \mathcal{L}^{<t>}}{\partial y^{<t>}} \frac{\partial y^{<t>}}{\partial \boldsymbol{o}^{<t>}} \frac{\partial \boldsymbol{o}^{<t>}}{\partial \boldsymbol{o}^{<1>}} \frac{\partial \boldsymbol{o}^{<1>}}{\partial W} \qquad (2.44) \\
&= \frac{\partial \mathcal{L}^{<t>}}{\partial y^{<t>}} \frac{\partial y^{<t>}}{\partial \boldsymbol{o}^{<t>}} \sum_{k=1}^{t} \frac{\partial \boldsymbol{o}^{<t>}}{\partial \boldsymbol{o}^{<k>}} \frac{\partial \boldsymbol{o}^{<k>}}{\partial W} \qquad (2.45)
\end{aligned}$$

In equation 2.45, $\frac{\partial \boldsymbol{o}^{<t>}}{\partial \boldsymbol{o}^{<k>}}$ is also extended to

$$\frac{\partial \boldsymbol{o}^{<t>}}{\partial \boldsymbol{o}^{<k>}} = \frac{\partial \boldsymbol{o}^{<t>}}{\partial \boldsymbol{o}^{<t-1>}} \frac{\partial \boldsymbol{o}^{<t-1>}}{\partial \boldsymbol{o}^{<t-1>}} \cdots \frac{\partial \boldsymbol{o}^{<k+1>}}{\partial \boldsymbol{o}^{<k>}} = \prod_{j=k+1}^{t} \frac{\partial \boldsymbol{o}^{<j>}}{\partial \boldsymbol{o}^{<j-1>}} \tag{2.46}$$

Therefore, the update rule of W is given by

$$W \leftarrow W - \alpha \sum_{k=1}^{t} \frac{\partial \mathcal{L}^{<t>}}{\partial y^{<t>}} \frac{\partial y^{<t>}}{\partial \boldsymbol{o}^{<t>}} \Big(\prod_{j=k+1}^{t} \frac{\partial \boldsymbol{o}^{<j>}}{\partial \boldsymbol{o}^{<j-1>}} \Big) \frac{\partial \boldsymbol{o}^{<k>}}{\partial W} \tag{2.47}$$

The update rule of U is similar to the update rule of W in equation 2.47, since it requires chain derivatives of $\boldsymbol{o}^{<t>}$, thus we have the update rule of U as following

$$U \leftarrow U - \alpha \sum_{k=1}^{t} \frac{\partial \mathcal{L}^{<t>}}{\partial y^{<t>}} \frac{\partial y^{<t>}}{\partial \boldsymbol{o}^{<t>}} \Big(\prod_{j=k+1}^{t} \frac{\partial \boldsymbol{o}^{<j>}}{\partial \boldsymbol{o}^{<j-1>}} \Big) \frac{\partial \boldsymbol{o}^{<k>}}{\partial U} \tag{2.48}$$

LSTM

Although RNNs perform efficiently for a given continuous data or a sequence, RNNs also have a problem which is known as vanishing gradient problem. As seen in equation 2.47, the gradient descent of RNNs has the term

$$\prod_{j=k+1}^{t} \frac{\partial \boldsymbol{o}^{<j>}}{\partial \boldsymbol{o}^{<j-1>}} \tag{2.49}$$

Suppose we get a product of $\boldsymbol{o}^{j}$ with a time range from 2 to t, then we have

$$\prod_{j=2}^{t} \frac{\partial \boldsymbol{o}^{<j>}}{\partial \boldsymbol{o}^{<j-1>}} = \frac{\partial \boldsymbol{o}^{<2>}}{\partial \boldsymbol{o}^{<1>}} \frac{\partial \boldsymbol{o}^{<3>}}{\partial \boldsymbol{o}^{<2>}} \cdots \frac{\partial \boldsymbol{o}^{<t>}}{\partial \boldsymbol{o}^{<t-1>}} \tag{2.50}$$

Each partial derivative in equation 2.50 depends on a derivative of an activation function. Values of the derivative of activation functions range from 0 to 1 as seen in figure 2.5. Due to this characteristic, a value of the equation 2.49 is getting small as t is increasing. Therefore, the influence of a given input on the hidden layer and the output layer decays exponentially, and this phenomenon is called vanishing gradient problem [15]. Long Short-Term Memory (LSTM) model is one of the solutions to solve this problem. The difference between an LSTM and a standard RNN is that the LSTM has a memory block instead of a weighted sum unit in the hidden layer. Each memory block consists of a cell that is connected recurrently, and three gates that have two conditions (open and close): input gate, forget gate, and output gate. An activation function of gates is usually the sigmoid to have a value between 0 and 1. The gating units make the memory cells to store information for a long time by controlling their conditions, thus LSTMs can avoid the vanishing gradient problem [15]. Given input is sent not only to each memory block in the hidden layer but also to every gate of cells in the hidden layer. When the input arrives at a memory block, the input gates decides whether it is accepted or discarded. If a condition of the input gate is open, then the input is taken into the cell. The forget gate decides whether the previous input is saved or discarded. If the forget gate is also open, the present input and the previous input are summed at the cell. The output gate decides whether an output of the cell is sent to the output layer or not. If a condition of the output gate is open, then the output of the cell goes to the output layer. The output of the cell goes to all the gates of cells as well. The structure of an LSTM memory block is depicted in figure 2.9.
Suppose we have one hidden layer with k memory blocks and a sequence matrix X. Then, input $\boldsymbol{x}^{<1>} \in \mathbb{R}^n$, where $\boldsymbol{x}^{<}1>$ refers an input at time 1, is sent to each memory block includes three gates. At time 1, each forget gate receives the input $\boldsymbol{x}^{<1>}$, thus each forget

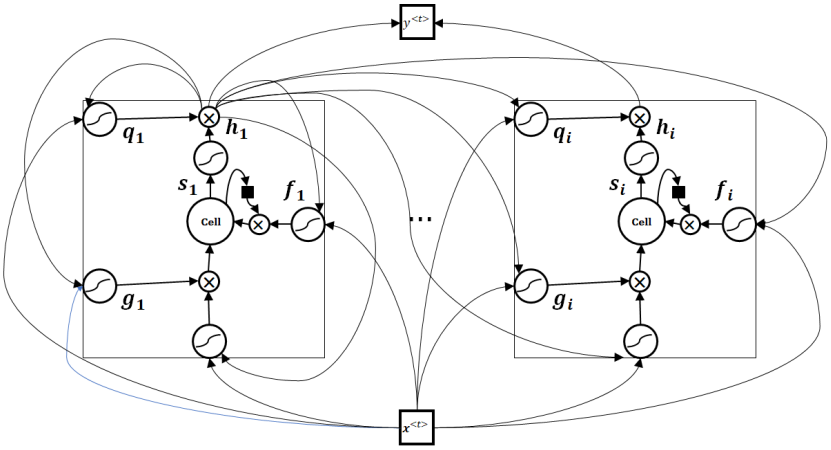

Figure 2.9: Structure of an LSTM network. Each cell consists of a forget gate (f_i), internal state (s_i), input gate (g_i), output gate (q_i). the input $\boldsymbol{x}^{<t>}$ at time t propagates to every cell and the output $h_i^{<t>}$ of the i_{th} cell at time t propagates not only to the next layer but also to every cell.

gate unit $f_i^{<1>}$ (i_{th} cell at time 1) is expressed as

$$f_1^{<1>} = \sigma(b_1^f + u_{11}^f x_0^{<1>} + u_{12}^f x_2^{<1>} + \cdots + u_{1n}^f x_n^{<1>}) = \sigma(b_1^f + \sum_{j=1}^{n} u_{1j}^f x_j^{<1>})$$

$$f_2^{<1>} = \sigma(b_2^f + u_{21}^f x_0^{<1>} + u_{22}^f x_2^{<1>} + \cdots + u_{2n}^f x_n^{<1>}) = \sigma(b_2^f + \sum_{j=1}^{n} u_{2j}^f x_j^{<1>})$$

$$\vdots$$

$$f_k^{<1>} = \sigma(b_k^f + u_{k1}^f x_0^{<1>} + u_{k2}^f x_2^{<1>} + \cdots + u_{kn}^f x_n^{<1>}) = \sigma(b_k^f + \sum_{j=1}^{n} u_{kj}^f x_j^{<1>})$$

, where b_i^f is a bias of the forget gate in the i_{th} memory block, u_{ij}^f is an element of the $k \times n$ weight matrix U^f. The internal state of cell $s_i^{<1>}$ is represented by

$$s_1^{<1>} = g_1^{<1>}\sigma(b_1 + u_{11}x_1^{<1>} + u_{12}x_2^{<1>} + \cdots + u_{1n}x_n^{<1>}) = g_1^{<1>}\sigma(b_1 + \sum_{j=1}^{n} u_{1j}x_j^{<1>}$$

$$s_2^{<1>} = g_2^{<1>}\sigma(b_2 + u_{21}x_1^{<1>} + u_{22}x_2^{<1>} + \cdots + u_{2n}x_n^{<1>}) = g_2^{<1>}\sigma(b_2 + \sum_{j=1}^{n} u_{2j}x_j^{<1>}$$

$$\vdots$$

$$s_k^{<1>} = g_k^{<1>}\sigma(b_k + u_{k1}x_1^{<1>} + u_{k2}x_2^{<1>} + \cdots + u_{kn}x_n^{<1>}) = g_k^{<1>}\sigma(b_k + \sum_{j=1}^{n} u_{kj}x_j^{<1>}$$

, where b_k is a bias of i_{th} cell, u_{ij} is an element of the weight matrix U. An output of an input gate $g_i^{<1>}$ is computed as

$$
\begin{aligned}
g_1^{<1>} &= \sigma(b_1^g + u_{11}^g x_1^{<1>} + u_{12}^g x_2^{<1>} + \cdots + u_{1n}^g x_n^{<1>} = \sigma(b_i^g + \sum_{j=1}^{n} u_{1j}^g x_j^{<1>}) \\
g_2^{<1>} &= \sigma(b_2^g + u_{21}^g x_1^{<1>} + u_{22}^g x_2^{<1>} + \cdots + u_{2n}^g x_n^{<1>} = \sigma(b_2^g + \sum_{j=1}^{n} u_{2j}^g x_j^{<1>}) \\
&\vdots \\
g_k^{<1>} &= \sigma(b_k^g + u_{k1}^g x_1^{<1>} + u_{k2}^g x_2^{<1>} + \cdots + u_{kn}^g x_n^{<1>} = \sigma(b_k^g + \sum_{j=1}^{n} u_{kj}^g x_j^{<1>})
\end{aligned}
$$

, where b_i^g is a bias of an input gate, u_{ij}^g is an element of the weight matrix U^g of input gates. Any non-linearity function can be applied to $s_i^{<t>}$ unlike gating units, if the hyperbolic tangent is applied to $s_i^{<1>}$, then we have

$$
\begin{aligned}
h_1^{<1>} &= tanh(s_1^{<1>}) q_1^{<1>} \\
h_2^{<1>} &= tanh(s_2^{<1>}) q_2^{<1>} \\
&\vdots \\
h_k^{<1>} &= tanh(s_k^{<1>}) q_k^{<1>}
\end{aligned}
$$

, where $q_i^{<1>}$ is an output of the output gate. An output of the output gate is calculated by

$$
\begin{aligned}
q_1^{<1>} &= \sigma(b_1^o + u_{11}^o x_1^{<1>} + u_{12}^o x_2^{<1>} + \cdots + u_{1n}^o x_n^{<1>} = \sigma(b_1^o + \sum_{j=1}^{n} u_{1j}^o x_j^{<1>} \\
q_2^{<1>} &= \sigma(b_2^o + u_{21}^o x_1^{<1>} + u_{22}^o x_2^{<1>} + \cdots + u_{2n}^o x_n^{<1>} = \sigma(b_2^o + \sum_{j=1}^{n} u_{2j}^o x_j^{<1>} \\
&\vdots \\
q_k^{<1>} &= \sigma(b_k^o + u_{k1}^o x_1^{<1>} + u_{k2}^o x_k^{<1>} + \cdots + u_{kn}^o x_n^{<1>} = \sigma(b_k^o + \sum_{j=1}^{n} u_{kj}^o x_j^{<1>}
\end{aligned}
$$

, where b_i^o is a bias of output gates, u_{ij}^o is an element of the weight matrix U^o of output gates. At time 1, all the operation are similar due to lack of previous information. However, from time 2, all gates receive not only an input but also values from at the end of every cell of the previous time. Therefore, forget gates at time 2 operates following

equation

$$
\begin{aligned}
f_1^{<2>} &= \sigma(b_1^f + u_{11}^f x_1^{<2>} + \cdots + u_{1n}^f x_n^{<2>} + w_{11}^f h_1^{<1>} + w_{12}^f h_2^{<1>} + \cdots + w_{1k}^f h_k^{<1>} \\
&= \sigma(b_1^f + \sum_{j=1}^{n} u_{1j}^f x_j^{<2>} + \sum_{l=1}^{k} w_1 l^f h_l^{<1>} \\
f_2^{<2>} &= \sigma(b_2^f + u_{21}^f x_1^{<2>} + \cdots + u_{2n}^f x_n^{<2>} + w_{21}^f h_1^{<1>} + w_{22}^f h_2^{<1>} + \cdots + w_{2k}^f h_k^{<1>} \\
&= \sigma(b_2^f + \sum_{j=1}^{n} u_{2j}^f x_j^{<2>} + \sum_{l=1}^{k} w_2 l^f h_l^{<1>} \\
&\vdots \\
f_k^{<2>} &= \sigma(b_k^f + u_{k1}^f x_1^{<2>} + \cdots + u_{kn}^f x_n^{<2>} + w_{k1}^f h_1^{<1>} + w_{k2}^f h_2^{<1>} + \cdots + w_{kk}^f h_k^{<1>} \\
&= \sigma(b_k^f + \sum_{j=1}^{n} u_{kj}^f x_j^{<2>} + \sum_{l=1}^{k} w_k l^f h_l^{<1>})
\end{aligned}
$$

, where w_{ij} is an element of the $k \times k$ recurrent weight matrix W^f of forget gates. Each internal states $s_i^{<2>}$ of cells is also updated as following equation

$$
\begin{aligned}
s_1^{<2>} &= f_1^{<2>} s_1^{<1>} + g_1^{<2>} \sigma(b_1 + u_{11} x_1^{<2>} + \cdots + u_{1n} x_n^{<2>} + w_{11} h_1^{<1>} + \cdots + w_{1k} h_k^{<1>}) \\
&= f_1^{<2>} s_1^{<1>} + g_1^{<2>} \sigma(b_1 + \sum_{j=1}^{n} u_{1j} x_j^{<2>} + \sum_{l=1}^{k} w_1 l h_l^{<1>}) \\
s_2^{<2>} &= f_2^{<2>} s_2^{<1>} + g_2^{<2>} \sigma(b_2 + u_{21} x_1^{<2>} + \cdots + u_{2n} x_n^{<2>} + w_{21} h_1^{<1>} + \cdots + w_{2k} h_k^{<1>}) \\
&= f_2^{<2>} s_2^{<1>} + g_2^{<2>} \sigma(b_2 + \sum_{j=1}^{n} u_{2j} x_j^{<2>} + \sum_{l=1}^{k} w_2 l h_l^{<1>}) \\
&\vdots \\
s_k^{<2>} &= f_k^{<2>} s_k^{<1>} + g_k^{<2>} \sigma(b_k + u_{k1} x_1^{<2>} + \cdots + u_{kn} x_n^{<2>} + w_{k1} h_1^{<1>} + \cdots + w_{kk} h_k^{<1>}) \\
&= f_k^{<2>} s_k^{<1>} + g_k^{<2>} \sigma(b_k + \sum_{j=1}^{n} u_{kj} x_j^{<2>} + \sum_{l=1}^{k} w_k l h_l^{<1>})
\end{aligned}
$$

, where u_{ij} is an element of the recurrent weight matrix U. An input gate also receives information from the input and the previous output of all k cells. Thus, g_i^2 is represented

by

$$
\begin{aligned}
g_1^{<2>} &= \sigma(b_1^g + u_{11}^g x_1^{<2>} + \cdots + u_{1n}^g x_n^{<2>} + w_{11}^g h_1^{<1>} + w_{12}^g h_2^{<1>} + \cdots + w_{1k}^g h_k^{<1>} \\
&= \sigma(b_1^g + \sum_{j=1}^{n} u_{1j}^g x_j^{<2>} + \sum_{l=1}^{k} w_1 l^g h_l^{<2>} \\
g_2^{<2>} &= \sigma(b_2^g + u_{21}^g x_1^{<2>} + \cdots + u_{2n}^g x_n^{<2>} + w_{21}^g h_1^{<1>} + w_{22}^g h_2^{<1>} + \cdots + w_{2k}^g h_k^{<1>} \\
&= \sigma(b_2^g + \sum_{j=1}^{n} u_{2j}^g x_j^{<2>} + \sum_{l=1}^{k} w_2 l^g h_l^{<2>} \\
&\vdots \\
g_k^{<2>} &= \sigma(b_k^g + u_{k1}^g x_1^{<2>} + \cdots + u_{kn}^g x_n^{<2>} + w_{k1}^g h_1^{<1>} + w_{k2}^g h_2^{<1>} + \cdots + w_{kk}^g h_k^{<1>} \\
&= \sigma(b_k^g + \sum_{j=1}^{n} u_{kj}^g x_j^{<2>} + \sum_{l=1}^{k} w_k l^g h_l^{<2>}
\end{aligned}
$$

Each output gate also receives the input and the previous output of every cell, therefore $q_i^{<2>}$ is computed as follows

$$
\begin{aligned}
q_1^{<2>} &= \sigma(b_1^o + u_{11}^o x_1^{<2>} + \cdots + u_{1n}^o x_n^{<2>} + w_{11}^o h_1^{<1>} + w_{12}^o h_2^{<1>} + \cdots + w_{1k}^o h_n^{<1>}) \\
&= \sigma(b_1^o + \sum_{j=1}^{n} u_{1j} x_j^{<2>} + \sum_{l=1}^{k} w_{1l}^o h_l^{<2>} \\
q_2^{<2>} &= \sigma(b_2^o + u_{21}^o x_1^{<2>} + \cdots + u_{2n}^o x_n^{<2>} + w_{21}^o h_1^{<1>} + w_{22}^o h_2^{<1>} + \cdots + w_{2k}^o h_n^{<1>}) \\
&= \sigma(b_2^o + \sum_{j=1}^{n} u_{2j} x_j^{<2>} + \sum_{l=1}^{k} w_{2l}^o h_l^{<2>} \\
&\vdots \\
q_k^{<2>} &= \sigma(b_k^o + u_{k1}^o x_1^{<2>} + \cdots + u_{kn}^o x_n^{<2>} + w_{k1}^o h_1^{<1>} + w_{k2}^o h_2^{<1>} + \cdots + w_{kk}^o h_n^{<1>}) \\
&= \sigma(b_1^o + \sum_{j=1}^{n} u_{1j} x_j^{<2>} + \sum_{l=1}^{k} w_{1l}^o h_l^{<2>}
\end{aligned}
$$

The output $h_i^{<2>}$ of LSTM cells is produced by a multiplication of the output of the output gate and the output of the internal states after an activation function is applied by

$$
\begin{aligned}
h_1^{<2>} &= tanh(s_1^{<2>}) q_1^{<2>} \\
h_2^{<2>} &= tanh(s_2^{<2>}) q_2^{<2>} \\
&\vdots \\
h_k^{<2>} &= tanh(s_k^{<2>}) q_k^{<2>}
\end{aligned}
$$

From the above process, we can generalize all operations for $t > 1$. The generalized operations for all gates and cells are listed as follows

$$f_i^{<t>} = \sigma(b_i^f + \sum_{j=1}^{n} u_{ij}^f x_j^{<t>} + \sum_{l=1}^{k} w_{il}^f h_l^{<t-1>}) \tag{2.51}$$

$$s_i^{<t>} = f_i^{<t>} s_i^{<t-1>} + g_i^{<t>} \sigma(b_i + \sum_{j=1}^{n} u_{ij} x_j^{<t>} + \sum_{l=1}^{k} w_{il} h_l^{<t-1>}) \tag{2.52}$$

$$g_i^{<t>} = \sigma(b_i^g + \sum_{j=1}^{n} u_{ij}^g x_j^{<t>} + \sum_{l=1}^{k} w_{il}^g h_l^{<t-1>}) \tag{2.53}$$

$$q_i^{<t>} = \sigma(b_i^o + \sum_{j=1}^{n} u_{ij}^o x_j^{<t>} + \sum_{l=1}^{k} w_{il}^o h_l^{<t-1>}) \tag{2.54}$$

$$h_i^{<t>} = tanh(s_i^{<t>}) q_i^{<t>} \tag{2.55}$$

2.2.3 Convolutional Networks

Convolutional neural network (CNN) is also a type of the neural networks model that is specialized to classify images or time-series data such as audio waveform. CNN also consists of hidden layers and an output layer, but the hidden layer of CNN is separated into two parts. First part is a convolutional layer that includes neurons with weights and biases that operates a weighted sum and a non-linear function, and a second part is a pooling layer that decreases a size of an output of a convolutional layer. In typical neural networks, a number of weights of each neuron at the first hidden layer match to the number of input features, therefore all input features of each given input interacts with all weights of each neuron at once. Unlike the typical neural networks, a number of weights of each neuron at the first hidden layer are smaller than the number of input features, and the weights of each neuron are called kernel or filter. In other words, the size of the filter is smaller than the size of the input in CNNs. Suppose our task is to classify an image to a corresponding category. An image comprises the raw pixel values in red, green, and blue color, respectively. Therefore, an input is represented as a $3D$ tensor with dimensions $(X_h, X_w, 3)$, where X_h is a height of the image, x_w is width, and 3 is a number of channels (red, green, and blue). A filter is represented as a $3D$ tensor with dimensions $(F_h, F_w, 3)$, where F_h is a height of a filter, F_w is width, and a depth of a filter must be same as the depth of the input. When each neuron receives an input, a filter interacts to a local region of an image and moves to the next local region. An amount of pixels that the filter shifts is called stride. The stride is one of the hyperparameters that controls a volume of the output at a neuron as seen in figure 2.10. By varying the size of the filter and value of stride, we can decrease the volume of input for fewer computations. Zeros can be added at sides of an input to control a volume of the output and is known as zero-padding. Thus a height and width of input are affected by the size of the filter, the value of stride and size of zero-padding. A depth of the output is decided by a number of

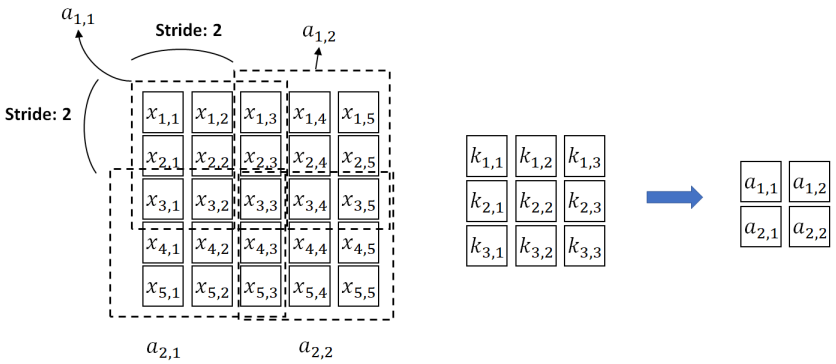

Figure 2.10: Example of convolutional network. The input image consists of 5×5 pixels with one channel. The filter that has a 3×3 size, strides by 2×2 pixels. Therefore, the dimension of the output is 2×2.

filters. Therefore, a volume of the output is represented as

$$
\begin{aligned}
A_h &= \frac{X_h - F_h + 2P}{S} + 1 \\
A_w &= \frac{X_w - F_w + 2P}{S} + 1 \\
A_c &= F_n
\end{aligned}
$$

, where O_h is a height of the output, P is a number of padded zero on a side, S_h is a value of a stride, O_w is a width of the output, O_c is a depth of the output, and F_n is a number of filters. An activation function can also be applied to the output of the hidden layer. The most commonly used pool layer is a max pool layer. The max pool layer can be described as a filter that has a max operation. The max pool layer picks the maximum value in a local region, then shifts to the next local region as seen in figure 2.11.

Feedforward

We begin with an example of CNNs model with one hidden layer. An input data point with dimensions $(2, 5, 5)$ is given and the hidden layer consists of 3 neurons that conclude a filter with dimension $(2, 3, 3)$, respectively. If a filter strides by 2×2, then we have the

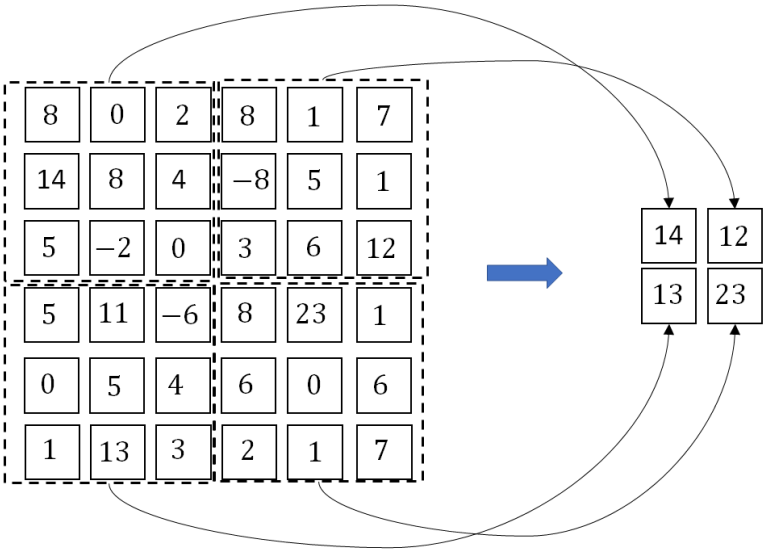

Figure 2.11: Example of max pooling in CNN network. The output of the convolutional layer has the size of 6×6. The max pooling filter that has a 3×3 size, strides by 3×3 pixels same as the kernel size. Therefore, the dimension of the output is 2×2.

output with dimension $(3, 2, 2)$. The output of the first neuron is computed by

$$\begin{aligned}
a_{1,1,1} &= k_{1,1,1,1}x_{1,1,1} + k_{1,1,1,2}x_{1,1,2} + \cdots + k_{1,1,3,2}x_{1,3,2} + k_{1,1,3,3}x_{1,3,3} \\
&\quad + k_{1,2,1,1}x_{2,1,1} + k_{1,2,1,2}x_{2,1,2} + \cdots + k_{1,2,3,3}x_{2,3,3} + b_{1,1,1} \\
a_{1,1,2} &= k_{1,1,1,1}x_{1,1,3} + k_{1,1,1,2}x_{1,1,4} + \cdots + k_{1,1,3,2}x_{1,3,4} + k_{1,1,3,3}x_{1,3,5} \\
&\quad + k_{1,2,1,1}x_{2,1,3} + k_{1,2,1,2}x_{2,1,4} + \cdots + k_{1,2,3,3}x_{2,3,5} + b_{1,1,2} \\
a_{1,2,1} &= k_{1,1,1,1}x_{1,3,1} + k_{1,1,1,2}x_{1,3,2} + \cdots + k_{1,1,3,2}x_{1,5,2} + k_{1,1,3,3}x_{1,5,3} \\
&\quad + k_{1,2,1,1}x_{2,3,1} + k_{1,2,1,2}x_{2,3,2} + \cdots + k_{1,2,3,3}x_{2,5,3} + b_{1,2,1} \\
a_{1,2,2} &= k_{1,1,1,1}x_{1,3,3} + k_{1,1,1,2}x_{1,3,4} + \cdots + k_{1,1,3,2}x_{1,5,4} + k_{1,1,3,3}x_{1,5,5} \\
&\quad + k_{1,2,1,1}x_{2,3,3} + k_{1,2,1,2}x_{2,3,4} + \cdots + k_{1,2,3,3}x_{2,5,5} + b_{1,2,2}
\end{aligned}$$

, where $a_{i,j,k}$ is a value of j_{th} row and k_{th} column in channel i (the output of i_{th} neuron), $k_{i,l,j,k}$ is a weight value of j_{th} row and k_{th} column in l_{th} channel of i_{th} neuron, $x_{l,j,k}$ is an input value of j_{th} row and k_{th} column in channel l, and $b_{i,j,k}$ is a bias of a value of j_{th} row

and k_{th} column in channel. Each output $a_{i,j,k}$ of neurons can be generalized by

$$a_{i,j,k} = \sum_{l=1}^{X_f} \sum_{m=1}^{F_h} \sum_{n=}^{F_w} x_{l,(j-1)\times s+m,(k-1)\times s+n} k_{i,l,m,n} + b_{i,j,k} \tag{2.56}$$

, where s is a size of stride, X_f is a number of input channels, F_h is a height of a filter, and F_w is a width of a filter. As a typical neural network, an activation function is applied to the output of neurons $a_{i,j,k}$ in equation 2.56, respectively. After an activation function is applied, we have the output value $o_{i,j,k}$ of the j_{th} row and the k_{th} column in the i_{th} output channel as following

$$o_{i,j,k} = tanh(a_{i,j,k}) = tanh(\sum_{l=1}^{X_f} \sum_{m=1}^{F_h} \sum_{n=}^{F_w} x_{l,(j-1)\times s+m,(k-1)\times s+n} k_{i,l,m,n} + b_{i,j,k}) \tag{2.57}$$

Backpropagation

We can also use the gradient descent that is mentioned in section 2.2.1 for CNNs. So we can apply the chain rule to CNN, then we get a partial derivative for a single hidden layer as following

$$\frac{\partial \mathcal{L}}{\partial w_{i,l,j,k}} = \sum_{m=1}^{A_h} \sum_{n=1}^{a_w} \frac{\partial \mathcal{L}}{\partial a_{i,j,k}} \frac{\partial a_{i,j,k}}{\partial w_{i,l,j,k}} = \sum_{m=1} \sum_{n} \frac{\partial \mathcal{L}}{\partial a_{i,j,k}} x_{l,(m-1)\times s+k,(n-1)\times s+k} \tag{2.58}$$

2.3 Generalization

The goal of network training is not only minimizing the training error but also minimizing the error of inexperienced data such as the validation error or the test error. The ability of the network to make the error of inexperienced data small is called generalization [12]. Although more data points in the training set can improve the generalization, there are many cases that collection of more data points is limited. Although there are many methods of improving generalization with a fixed number of data points, two methods that are applied in this thesis are discussed in this section. More methods of improving generalization can be found in [12], [17].

2.3.1 Input noise

Before an input data point is fed to the network, adding Gaussian noise is a method of improving generalization. Although it may be expected that the noise disturbs to train the network model, it has been found that training with noise can improve generalization in practice [17]. The Gaussian noise with n-dimensional, where n is the number of features, is added to each input data points. When the Gaussian noise is added, its variance should be smaller than 1.0 [18].

2.3.2 Dropout

Dropout is a technique to avoid overfitting by limiting the number of neurons to be updated on the training set while all the neurons are used to compute the validation error and the test error. For example, if dropout with probability 0.2 is applied to a hidden layer that includes 100 neurons, around 20 neurons are disconnected during feedforward on the training set. The simple way of disconnection neuron is to multiply the output of neuron by zero [12]. Another algorithm is each output of neurons is multiplied by a probability p at each time [19]. The standard feedfoward

$$o_j^{(i)} = f(\boldsymbol{w}_j^{(i)} \boldsymbol{a}^{(i)}) \tag{2.59}$$

is replaced by

$$o_j^{(i)*} = f(p\vec{w}_j^{(i)} \boldsymbol{a}^{(i)}) \tag{2.60}$$

The example of the dropout is depicted in figure 2.12.

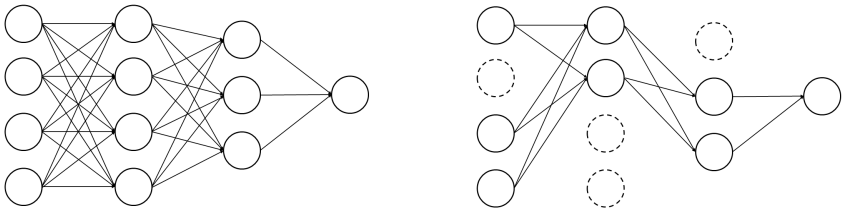

Figure 2.12: Example of dropout.
Left: The standard neural network. Each layer is fully connected.
Right: The neural network when dropout is applied. Some neurons are disconnected.

3 Dataset

The dataset for this thesis is provided by Remote Collaborative and Affective Interactions(RECOLA) [6]. RECOLA database was built for collecting spontaneous interactions from a collaborative task. RECOLA contains multimodal data that consists audio, video, electrocardiogram and electro-dermal activity and annotations that include two emotional dimensions(arousal and valence) which were annotated by their developed annotation tool called ANNEMO [6]. Although it consists of more than 9.5 hours of recordings, a set of audiovisual data was reduced to have 3.8 hours by keeping only their first five minutes of each record from 46 participants. Among data of 46 participants, audio recordings of 23 participants are used as the input set, and its annotated arousal is used as the label set for this thesis. 23 participants are separated into three parts: the training, validation and test dataset as mentioned in section 2.2.1. Their mother tongue, age, and gender are considered to distribute participants into three datasets as seen in Table 3.1.

subset	Mother tongue	Gender	Age: mean (standard deviation)
Total	17 French 3 German 3 Italian	10 males 13 females	21.48(2.0)
Training	6 French 1 German 1 Italian	3 males 5 females	21.38(1.5)
Validation	6 French 1 German 1 Italian	4 males 4 females	21.38(2.5)
Test	5 French 1 German 1 Italian	3 males 4 females	21.71(1.83)

Table 3.1: Subsets of the RECOLA database: training, validation, and test

3.1 Pre-processing

Before each input dataset is fed to the network, its values are rescaled since it reduces the training period. Suppose there are two features, for example, if the unit of the first

feature is much bigger than the unit of the second feature, the bigger unit has more effect to the predicted value even if the importance of two features are same. Two techniques of re-scaling are considered to be applied for this thesis: min-max normalization and z-score normalization [20]. Min-max normalization scales values to have a range from -1 to 1 by

$$x_{i,(j,k)} \leftarrow \frac{x_{i,(j,k)} - min(x_k)_{train}}{max(x_k)_{train} - min(x_k)_{train}} \tag{3.1}$$

, where $x_{i,(j,k)}$ is an element of j_{th} row (data point) and k_{th} column (feature) in the dataset of i_{th} subject, $min(x_k)_{train}$ is the minimum value of k_{th} feature in the training set, and $max(x_k)_{train}$ is the maximum value of k_{th} feature in the training set. For the min-max normalization, the minimum value of each feature in the training set and the maximum value of each feature in the training set are used to normalize not only the training set but also the validation set and the test set. Z-score normalization scales values to have the mean of zero and the unit variance by

$$x_{i,(j,k)} \leftarrow (x_{i,(j,k)} - \mu_{train,k})/\sigma_{train,k} \tag{3.2}$$

, where $\mu_{train,k}$ is the mean of k_{th} feature in the training set and $\sigma_{train,k}$ is the standard deviation of k_{th} feature in the training set. The mean and standard deviation of each feature in the training set are used to normalize the training, the validation, and the test sets.

3.2 Label and input dataset

3.2.1 Label dataset

Emotions can be presented to continuous dimensions that consists of arousal dimension(calm/excited) and valence dimension(negative/positive) [1]. A particular emotion is represented on two continuous dimensions as seen in figure 3.1 [21]. In AVEC challenge, it was shown that speech signal is more correlated to the arousal dimension than the valence dimension, and the speech signal performs better to predict emotions on the arousal dimensions than other modalities such as facial descriptor, electrocardiogram, and electrodermal [7]. Thus, the arousal dimension is chosen as the label to recognize emotions for this thesis. In RECOLA database, the arousal dimension was annotated time-continuously by 6 annotators using the annotation tool ANNEMO [6]. Values of arousal are ranged from -1 to 1 with a step of 0.01. Each annotated data was binned with a 40 ms frame rate to reduce blanks or jumps [6]. The 6 arousal data from each annotator were summed and averaged after being normalized to have a mean of zero. After the preprocessing of label data, the label set has values of arousal from 0 s to 300 s at 25 Hz sampling rate, thus each annotated data consists of 7501 values of arousal for each subject. We denote the label set of the i_{th} subject by

$$\hat{\boldsymbol{y}}_i^T = [\hat{y}_{i,1}, \hat{y}_{i,2}, \hat{y}_{i,3}, \cdots, \hat{y}_{i,7500}, \hat{y}_{i,7501}] \tag{3.3}$$

, where $\hat{y}_{i,j}$ is the value of the annotated arousal corresponding to the j_{th} data point in the label set of the i_{th} subject.

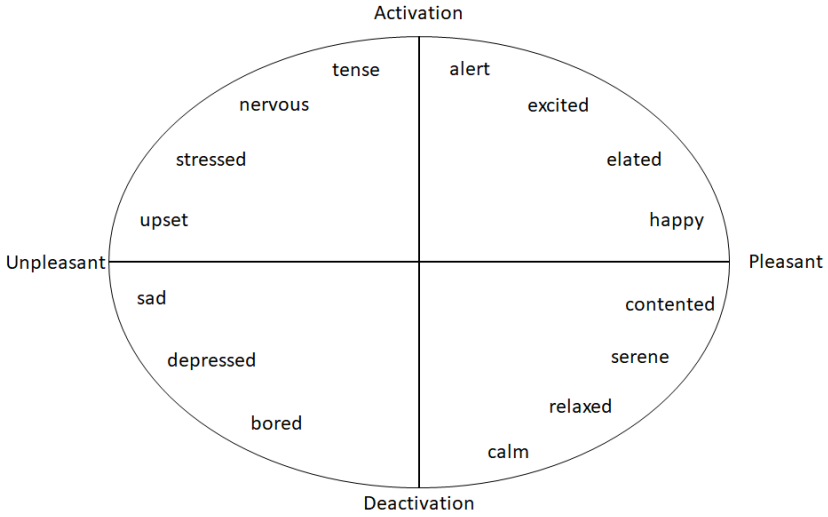

Figure 3.1: Representation of emotions on arousal and valence dimensions. The horizontal axis represents the valence dimension, and the vertical axis represents the arousal dimension [21].

3.2.2 Input dataset

The audio data from the RECOLA database is chosen as the input dataset for this thesis. Audio data that were captured by unidirectional microphones were recorded at 44.1 kHz, 16 bits by Audacity software [6]. The training set and the validation set contain 40-minute audio, and the test set includes 35-minute audio. We transform the raw audio data to three different types of sound: a set of acoustic features, waveform, and spectrogram. Those three different types of audio are segmented by an overlapping window shifting by 40 ms. Each chunk of segmented audio data is considered as a sequence that is fed to neural networks.

Feature set

The first input type is a set of acoustic characteristics. For this thesis, the same set of acoustic Low-Level Descriptors(LLD) is used as RECOLA database, extracted by open-source feature extractor openSMILE [2]. This feature set consists of 55 spectral LLD, 6 voicing related LLD, four energy-related LLD, and their first order derivate - total 130 LLD [2]. Spectral LLD and energy LLD were extracted from 25 ms frames using Hamming window that was overlapping 10 ms windows and was sampled at 100 Hz, voicing related LLD was extracted from 60 ms frame using the Gaussian window that was overlapping 10 ms windows and was sampled at 100 Hz [2]. 65 LLD are listed in Table 3.2.

As a result of the feature extraction, each subject has values of features from 0.04 s to 299.96 s at 40 Hz sampling rate , thus it consists of 7499 data points. We denote a

6 voicing related LLD	**Group**
F_0 (SHS & Viterbi smoothing) (1 LLD)	prosodic
Prob. of voice	Sound quality
Log. HNR, Jitter(local, delta), Shimmer(local) (4 LLD)	Sound quality
4 energy related LLD	**Group**
Sum of auditory spectrum (loudness)	prosodic
Sum of RASTA-filtered auditory spectrum	prosodic
RMS energy, zero-crossing rate (2 LLD)	prosodic
55 spectral LLD	**Group**
MFCC 1 - 14 (14 LLD)	cepstral
RASTA-filtered auditory spectrum, bands 1 - 26 (26 LLD)	spectral
Spectral energy 250 Hz to 650 Hz, 1 kHz to 4 kHz (2 LLD)	spectral
Spectral roll off point 0.25, 0.50, 0.75, 0.90 (4 LLD)	spectral
Spectral flux, centroid, entropy, slope (4 LLD)	spectral
Psychoacoustic sharpness, harmonicity (2 LLD)	spectral
Spectral variance, skewness, kurtosis (3 LLD)	spectral

Table 3.2: Input feature set [22]

feature set of each speaker by a 7499×130 matrix F_i, where F_i refers to the feature set of the i_{th} subject. Each data point at time step j is denoted by a 130×1 vector $\boldsymbol{f}_{i,j}$. Each column of the matrix F_i can be normalized either by the min-max normalization or the z-score normalization as mentioned in section 3.1. For the min-max standardization, the minimum value of each column in the training set and the maximum values of each column in the training set are used to normalize not only the training set but also the validation set and the test set. Thus, each input set of a subject is normalized by

$$f_{i,(j,k)} \leftarrow \frac{f_{i,(j,k)} - min(f_k)_{train}}{max(f_k)_{train} - min(f_k)_{train}} \tag{3.4}$$

, where $f_{i,(j,k)}$ is the element of the j_{th} row and k_{th} column in the input set of i_{th} subject, $min(f_k)_{train}$ is the minimum value of k_{th} feature in the training set, $max(f_k)_{train}$ is the maximum value of k_{th} feature in the training set. For the z-score normalization, the mean of each feature in the training set and the standard deviation of each feature in the training set are also used to normalize the training set, the validation set, and the test set. Each input set of a subject is normalized by

$$f_{i,(j,k)} \leftarrow (f_{i,(j,k)} - \mu_{train,k})/\sigma_{train,k} \tag{3.5}$$

, where $\mu_{train,k}$ is the mean of k_{th} feature in the training set, and $\sigma_{train,k}$ is the standard deviation of k_{th} feature in the training set. After the normalization, each input set F_i is segmented by overlapping windows that has a size of s in seconds and shifts by 0.04 s. As a result of segmentation, each chunk has $s \times 25$ samples, where 25 refers to the number of samples at 25 Hz sampling rate. 7499 chunks are produced. Each chunk is considered as

an input sequence at each time step of recurrent neural networks. We represent the input sequence at time step j of each subject by $(s \times 25) \times 130$ matrix $X_i^{<j>}$. For example, the first sequence is a 50×130 matrix $X_i^{<1>}$ if the window size is 2 s since the first data point includes 130 features from 0.04 s to 2 s. The next sequence $X_i^{<2>}$ consists of input feature data points from 0.08 s to 2.04 s. Moreover, extra zeros are padded at the end of the sequence to keep the same length of sequences if it is needed. Therefore the last sequence $X_i^{<7499>}$ of the i_{th} subject consists of 130 features at 299.96 s and $(s \times 25 - 1) \times 130$ zeros, where s is a window size in seconds. The process of producing sequences with the two-second overlapping window is depicted in figure 3.2.

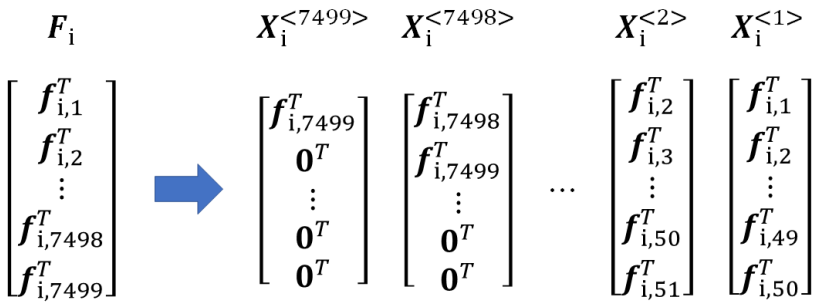

Figure 3.2: Process of producing sequences of the feature set when the window size is 2 s. F_i refers to the input feature set of the i_{th} subject. $\boldsymbol{f}_{i,j} \in \mathbb{R}^130$ includes 130 LLD at time step j. $X^{<j>}$ is the j_{th} input sequence and it consists of 50 data points of LLDs. $\mathbf{0}$ is a 130×1 vector which of elements are all zeros.

Waveform

The second input type is the waveform. For this thesis, the audio waveform of RECOLA database is down-sampled from 44.1 kHz to 6.4 kHz to reduce an amount of computation. After down-sampling, each subject has 1920,000 samples from 0 s to 299.999 843 75 s. The waveform set of the i_th subject is denoted by $\boldsymbol{v}_i$ that includes 1920,000 elements. Each waveform set is normalized by the z-score normalization by

$$\boldsymbol{v}_{i,(k)} \leftarrow (\boldsymbol{v}_{i,(k)} - \mu_{train})/\sigma_{train} \tag{3.6}$$

, where $\boldsymbol{v}_{i,(k)}$ is the k_{th} element of the i_{th} subject, μ_{train} is the mean of the training waveform set, and σ_{train} is the standard deviation of the training waveform set. After the waveform sets are normalized, the waveform set of each subject is segmented by overlapping windows of s seconds size shifting by 40 ms to produce sequences. The process of producing waveform sequences is depicted in figure 3.3.

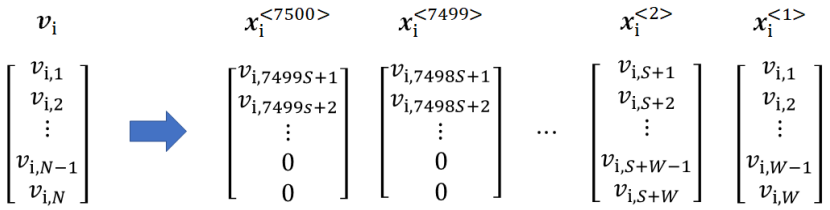

Figure 3.3: Process of producing sequences from waveform set. The window shifts by 0.04 s.
$\boldsymbol{v}_i$ is the waveform set of the i_{th} subject. It has a sample at every 1/6400 s. N is the number of samples per subject, S is the number of shifting samples that correspond to the number of samples for 40 ms at 25 Hz sampling rate and W is the number of samples per sequence.

Spectrogram

The third input type is spectrogram. The spectrogram set is also used to produce sequences.
The spectrogram displays acoustic signals in the time domain and the frequency domain. Each audio waveform is segmented by an overlapping window, and each chunk is calculated by fast Fourier transform(FFT) to get the magnitude of the frequency. If the length of the window is short, then we get high resolution in the time domain and low resolution in the frequency domain. In contrast, we get low resolution in the time domain and high resolution in the frequency domain, if the length of the window is long. Spectrograms generated by the Hamming window of various sizes are depicted in figure 3.4. Another issue is the maximum frequency that is presentable in the frequency domain is always half of the sampling rate. For example, a waveform is sampled at 48 kHz, then spectrogram can present the frequency domain up to 24 kHz. Since the typical vocal ranges of males and females are 75 Hz to 150 Hz and 150 Hz to 300 Hz, respectively, the sampling rate must exceed 600 Hz to present human vocal in the frequency domain [23].
For this thesis, the audio waveform is down-sampled to 6.4 kHz and is transformed to the spectrogram by Scipy that is scientific computing tool for python. The hamming window with the size of 256 frames is chosen to generate spectrograms. The Hamming window is set to have a 75% overlap. After the spectrogram is generated, the frequency domain is limited to 1 kHz, then it is normalized by the z-score normalization. Spectogram is presented by a matrix that the row represents the time domain and the column represents the frequency domain. $\boldsymbol{v}_{i,j}$ represents the magnitudes of frequency spectrum at time step j of the i_{th} subject. This spectrogram is segmented by an overlapping window to produce sequences. The process of producing sequences is depicted in figure 3.5.

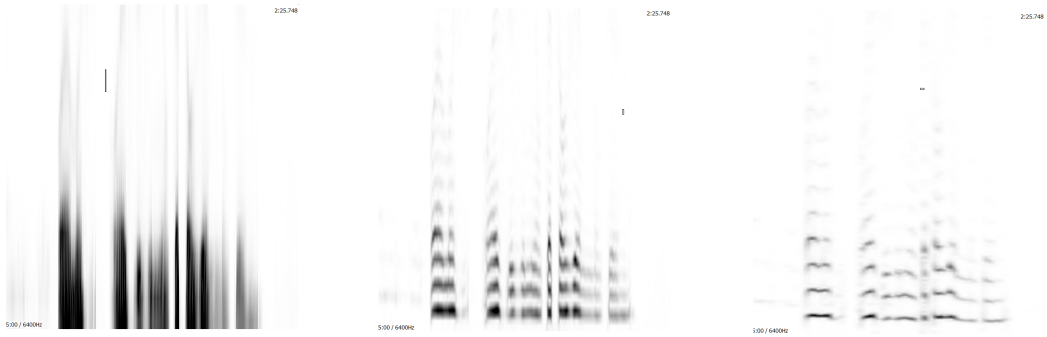

Figure 3.4: Example of spectrogram.
Three spectrograms transformed from the same audio signal. Each Hamming window is in the yellow circle.
Left: The hamming window size is 64 frames. Since the window size is short, it has a low resolution in the frequency domain. Therefore, patterns look blurred.
Middle: The hamming window size is 256 frames. Since the window size is longer than the left one, it has a high resolution in the frequency domain. Thus, patterns are easily found.
Right: The hamming window size is 512 frames. Although it has a high resolution in the frequency domain, patterns are rarely found due to a low resolution in the time domain.

3.3 Post processing

The produced sequences propagate through neural network layers to the output layer. Since the overlapping window is used to generate sequences, multiple numbers of predicted values are produced at each time step. The output sequences of the i_{th} subject are denoted by

$$Y_i = \begin{bmatrix} y_{i,1} & y_{i,2} & y_{i,3} & \cdots & y_{i,W-2} & y_{i,W-1} & y_{i,W} \\ y_{i,2} & y_{i,3} & y_{i,4} & \cdots & y_{i,W-1} & y_{i,W} & y_{i,W+1} \\ y_{i,3} & y_{i,4} & y_{i,5} & \cdots & y_{i,W} & y_{i,W+1} & y_{i,W+2} \\ \vdots & \vdots & \vdots & \cdots & \vdots & \vdots & \vdots \\ y_{i,N-2} & y_{i,N-1} & y_{i,N} & \cdots & 0 & 0 & 0 \\ y_{i,N-1} & y_{i,N} & 0 & \cdots & 0 & 0 & 0 \\ y_{i,N} & 0 & 0 & \cdots & 0 & 0 & 0 \end{bmatrix} \tag{3.7}$$

, where $y_{i,j}$ is the predicted value for the corresponding input sequence at time step j of the i_{th} subject, N is the number of samples per subject, and W is the number of samples per sequence. Thus, the multiple numbers of the predicted values are averaged to get one

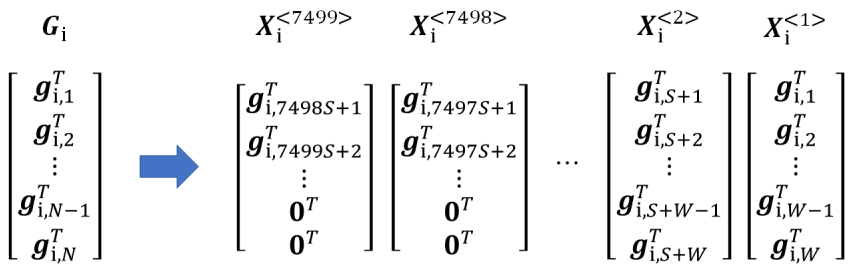

Figure 3.5: Process of producing sequences from spectrogram set. G_i is the spectrogram set of the i_{th} subject that has magnitudes of frequencies from 0.02 s to 299.98 s. The window for segmentation is shifted by 0.04 s from 0.04 s to fit to the label set. N is the number of samples per subject, S is the number of shifting samples that correspond to the number of samples for 40 ms at 25 Hz sampling rate and W is the number of samples per sequence.

predicted value at each time by

$$y_{i,j} = \begin{cases} \frac{1}{j} \sum_{j=1}^{W-1} \sum_{k=1}^{j+1} y_{i,(j-k+1,k)} & \text{if } j \leq W-1 \\ \frac{1}{W} \sum_{j=W}^{N} \sum_{k=1}^{j+j} y_{i,(j+k-W,W-k+1)} & \text{if } j \geq W \end{cases}$$

, where W is the number of samples per sequence, $y_{i,j}$ is the averaged output corresponding to the audio signal at time step j, $y_{i,(j,k)}$ is the element of j_{th} row and k_{th} column of the output matrix Y_i.

4 Experiments and Results

In this thesis, three different types of inputs are examined to recognize emotions on the arousal dimension. Three different input types are fed to three different networks, respectively, because characteristics of three input types are different. The first input type is fed through two recurrent neural networks with LSTMs to the output layer. The second input type is fed to two one-dimensional convolutional networks for extracting features from waveforms. Then, the output of the CNNs is fed through two recurrent neural networks with LSTMs to the output layer. The third input type is fed to two two-dimensional convolutional networks. Then, the output of the CNNs is fed through two recurrent neural networks with LSTMs to the output layer. Three input types are compared to each other after finding the optimal network. Therefore, the experiments are separated into two parts. Experiments of the first part are conducted to find the optimal network by adjusting hyperparameters with the training set and the validation set. The optimal hyperparameters are chosen when it shows the lowest training error and the smallest gap between the training error and the validation error. Experiments of the second part are conducted to get the test error on the optimal network. For three input types, Rmsprop is applied as the mini-batch gradient descent with batch size 128 [24]. The concordance correlation coefficient(CCC) is used to measure the performance of networks. CCC is denoted by

$$\rho_c = \frac{2\frac{1}{2}\sum_{n=1}^{N}(y_n - \overline{y})(\hat{y}_n - \overline{\hat{y}})}{\frac{1}{N}\sum_{n=1}^{N}(y_n - \overline{y})^2 + \frac{1}{N}\sum_{n=1}^{N}(\hat{y}_n - \overline{\hat{y}})^2} \tag{4.1}$$

, where y_n is the label of the n_{th} data point, $\hat{y}_n$ is the predicted value for the n_{th} data point, $\overline{y}$ is the mean of the label set, $\overline{\hat{y}}$ is the mean of the predicted value set(output set), and N is the number of elements in each dataset. The CCC evaluates the agreement between the label set and the output set. The CCC ranges from -1 to 1. If two sets consist of similar values, the CCC is close to 1. If two sets don't have similarities, the CCC is close to 0. Since the network learns to minimize the value of loss function, we modify the CCC to be a proper loss function by

$$\mathcal{L} = 1 - CCC \tag{4.2}$$

. The value of the loss function tends to be zero if two sets consist of similar values.

4.1 The set of features

Before we compare the performance of three input types, experiments are conducted to find the optimal hyperparameters and the optimal methods of generalization. The

network for the first input type comprises two RNNs with LSTMs and the output layer. All the initial values of hyperparameters are arbitrarily chosen. When the optimal values of hyperparameters are decided, those values are applied for the following experiments.

Input normalization

As mentioned in section 3.2.2, two methods of normalization are applied to the first input type and are compared to the raw dataset. Each Input dataset is segmented by the four-second window after the normalization. The first hidden layer consists of 80 neurons and the second hidden layer consists of 60 neurons. Each neuron of the hidden layers includes Tanh as the input and output activation function, and hard-sigmoid as the gate activation function. The output layer has the same number of neurons as the number of input neurons. Since each input sequence consists of 50 data points as mentioned in section 3.2.2, 100 values are predicted at each time step. All weights are initialized by Glorot initialization, and each bias is initialized to have zero. Mini-batch gradient descent with the batch size of 128 and variety learning rate ranging from 0.0001 to 0.0064 is applied for 20 epochs to update weights and biases. The network on each input dataset that shows the lowest error is chosen to compare the effects of the input normalization. The training errors on the different normalization methods are depicted in figure 4.1.

On the raw data, the lowest training error is 0.172 after 20-epoch training with the learning rate 0.0016. On the z-score normalization, the lowest training error after 20-epoch training is 0.0116 when the learning rate is 0.0016. On the min-max normalization, the lowest training error after 20-epoch training is 0.1533 when the learning rate is 0.0016. The result shows the z-score normalized decreases 99% of the training error on the raw data from 0.172 to 0.00116, while the min-max normalization decreases 10.8% of the training error on the raw data to 0.1533. As a result, the z-score normalization is applied to the following experiments.

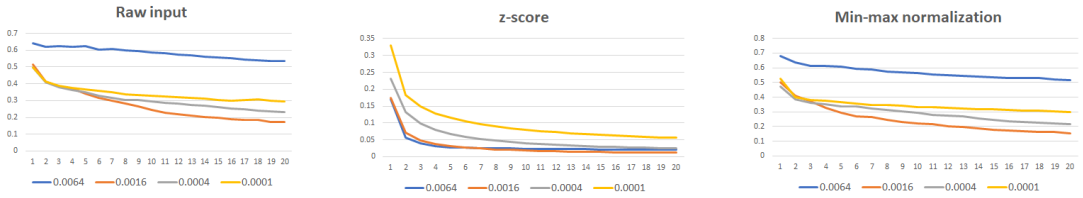

Figure 4.1: Training error depending on various normalization methods: raw input(left), z-score normalization(middle), min-max normalization(right). Learning rate ranges from 0.0001 to 0.0064. The x-axis refers to the number of epoch and the y-axis refers the training error.

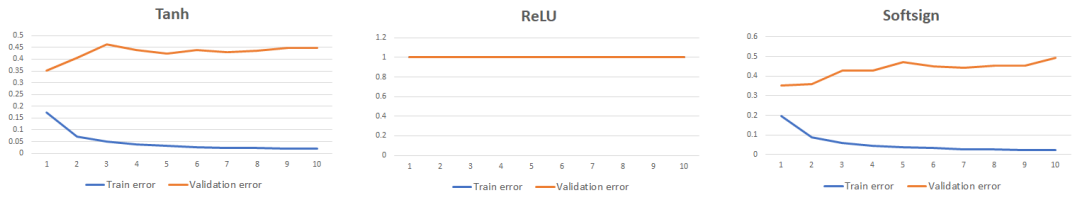

Figure 4.2: Training error depending on various activation functions on the feature set: Tanh(left), ReLU(middle), softsign(right).

Activation functions

This experiment is conducted to find the optimal input and output activation function for the hidden neurons. The same network as the former experiment in section 4.1 is used to feed the set of features after normalized by the z-score normalization. Three different activation functions that are examined with learning rate 0.0016 in this experiment are Tanh, ReLU, and softsign. The training errors and validation errors are depicted in figure 4.2

The lowest training error when each hidden neuron includes Tanh is 0.0184 after ten-epoch training with the learning rate 0.0016. When ReLU is applied, the error is fixed as 1 for training and validation sets. The lowest error when softsign is applied is 0.0211. Although the network has the lowest training error when Tanh is applied, the gap between training error and validation error is smallest when softsign is applied after one-epoch training. As a result, the softsign is applied to hidden neurons as an activation function for the following experiments and the network is trained for one epoch.

Input noise

Adding Gaussian noise to an input sequence helps the network to avoid overfitting. This experiment is conducted to find the optimal standard deviation of the Gaussian noise. Various learning rates ranging from 0.0004 to 0.0032 increasing by the factor of 2 are applied. When the Gaussian noise with the standard deviation of 0.1 is added to the input, the lowest validation error is acquired with learning rate 0.000. When the standard deviation of 0.8 is applied, the smallest gap between the training error and the validation error with learning rate 0.0008. Since the validation error has the lowest value with the standard deviation of 0.1, the Gaussian noise with the standard deviation of 0.1 is applied to the following experiments. The result of various standard deviations of input noise with learning rate 0.0008 are listed in table 4.1.

Dropout

Although the network for the feature set performs well with the Gaussian input noise, dropout is applied to two hidden layers in order to reduce the gap between the training

Standard deviation	0.8	0.4	0.2	0.1
Training error	0.2731	0.2466	0.2256	0.2203
Validation error	0.3154	0.2931	0.3425	0.2899
The gap	0.0423	0.0465	0.1169	0.0696

Table 4.1: Training error and validation error on various input noise with learning rate 0.0008 on the feature set.

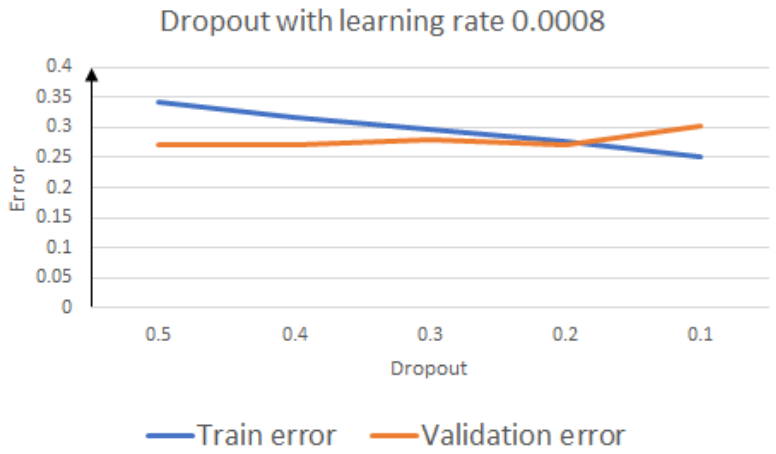

Figure 4.3: Training and validation error depending on various dropout probabilities on the feature set. Overffiting is avoided when the dropout $p = 0.2$ is applied.

and the validation error. As seen in figure 4.3, when the probability of dropout is 0.2, the gap is the smallest with learning rate 0.0008.

4.1.1 CCC on the test set

From the previous experiments, the optimal network for the feature set is acquired. The optimal network comprises two recurrent neural networks with LSTM and the output layer. It is shown that z-score normalization and softsign activation function have a benefit for the network. When the Gaussian noise with standard deviation of 0.1 and dropout of 0.2, the network has the lowest validation error and the smallest gap. Various length of the overlapping window is examined on this optimal network. The results show the feature set segmented by six-second window provide the lowest error among two-second, four-second, and six-second window. The results are listed in table 4.2

window size in seconds	training CCC	validation CCC	test CCC
2 s	0.5505	0.6248	0.6037
4 s	0.7239	0.7302	0.6973
6 s	0.7632	0.765	0.7259

Table 4.2: Training CCC, validation CCC, and test CCC on the feature set

4.2 Raw waveform

The network for raw waveform comprises two CNNs connected to the optimal network for the feature set without dropout. Two CNNs are applied to extract features from the raw waveform and down-sampled waveform from 6.4 kHz to 25 Hz.

4.2.1 Experiment setup

Since raw waveform is presented by a one-dimensional vector, one-dimensional CNNs are applied. The initial filter size of CNNs is set to 3 frames and shifting by one frame. Each CNN has 60 filters with ReLU as the activation function. Weights are initialized by Glorot uniform and biases are initialized to have zeros. Sequences are produced after z-score normalization is applied to the raw waveform.

Pooling size

Since raw waveform that is sampled at 6.4 kHz must be down-sampled to 25 Hz to match to the label set, multiplication of sizes of two pooling layer must be 256 frames. Various pooling sizes are examined, and the results are shown in figure 4.4. There is no significant difference of training error with various pooling size. The lowest validation error is acquired when the pooling size of the first layer is 32 frames and the pooling size of the second layer is 8 with learning rate 0.0016. Therefore, the first pooling layer with the size of 32 frames and the second pooling layer with the size of 8 are applied to the following experiments.

Filter size

After the optimal pooling size is applied, an experiment is conducted to find the optimal filter size. The filter size of the first layer is fixed to 3 frames and various filter sizes of the second layer are examined to simplify the experiment. The results show the various filter sizes of the second layer doesn't affect the training error. However, the lowest validation error is acquired when the first filter size is 3 frames and the second filter size is 15 as seen in figure 4.5.

Although the gap is bigger when the window size of the second layer is 15, it is assumed that it can be reduced if dropout is applied. Thus the first window size 3 frames and the

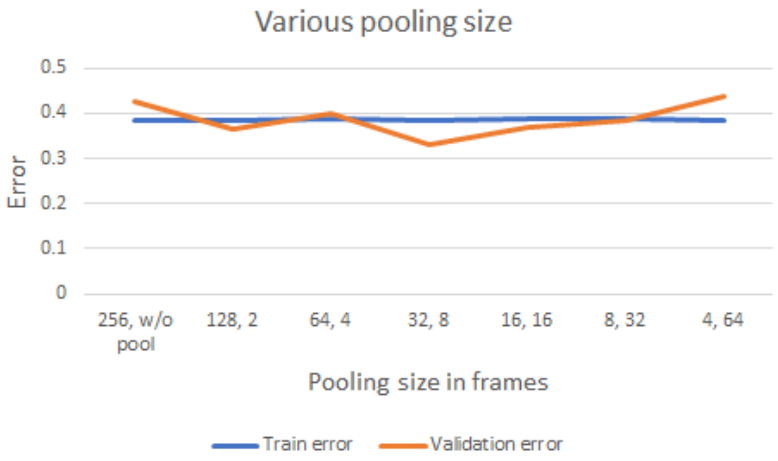

Figure 4.4: Training and validation error depending on various pooling sizes on the waveform input.

second window size 15 is applied to following experiments.

Dropout

The previous experiments to find the optimal pooling size and filter size are conducted with the optimal network for the feature set without dropout. This experiment is conducted to examine if dropout makes the network for the waveform to avoid overfitting. Dropout is applied not to CNNs but to RNNs with LSTM. Dropout with probability 0.2 and learning rate 0.0016 is applied. The result shows the gap is slightly decreased by increasing training error and decreasing validation error. However, dropout does not make a significant difference.

	without dropout	dropout $p = 0.2$
Training error	0.3041	0.3163
Validation error	0.3271	0.3172
Gap	0.023	0.0009

Table 4.3: Comparison of dropout on the waveform set

4.2.2 CCC on the test set

After the optimal network for waveform input, different segmenting window sizes are examined on the training, validation and test set. The first CNN layer comprises 60 filters with the size of 3 frames and the max pooling layer with the size of 32 frames are

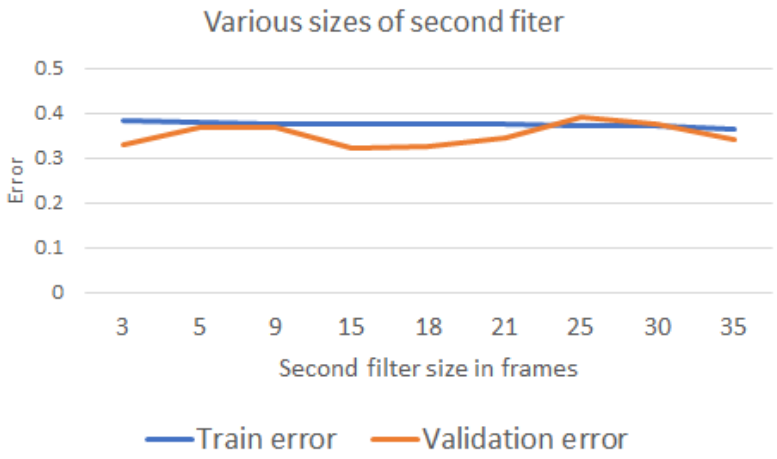

Figure 4.5: Training and validation error depending on various pooling sizes on the waveform input.

connected. The second CNN layer includes 60 filters with a size of 15 frames and the max pooling layer with a size of 8 frames. The waveform input is fed to two CNNs to extract features. The output of CNNs is considered as an input sequence of RNNs with LSTM. The waveform segmented by two-second, four-second, and six-second windows is fed to this optimal network. The results show the six-second segmenting window provide higher CCC than other sizes of window. Raw waveform does not improve generalization. The result is shown in table 4.4.

window size in seconds	training CCC	validation CCC	test CCC
2 s	0.5444	0.5388	0.505
4 s	0.6837	0.6828	0.6073
6 s	0.7284	0.7517	0.6491

Table 4.4: CCC on the wavefor set with the optimal network

4.3 The third proposed input type

The network for spectrogram comprises two CNNs connected to the optimal network of the feature set without dropout. Since spectrogram represents in the time domain and the frequency domain, two-dimensional CNNs are applied. Input features are extracted by two-dimensional CNNs and its output is fed to RNNs. The initial number of filter and activation are the same as the optimal network of the waveform.

Pooling size

Filter size 3×3 in frames is applied to both two-dimensional layers. A spectrogram is generated by the hamming window with the size of 256 frames from the raw speech signal at 6.4 kHz, thus a set of pooling layers that down-sample the raw speech signal to 25 Hz. The results show two different sets of pooling size provide similar to the training and validation error as seen in table 4.5. However, the amount of computation can be reduced if the first pooling layer is bigger. Therefore, the first pooling layer with the size of 4×4 frames without the second pooling layer are applied to the following experiments.

pooling size in frames	training error	validation error
4×4, w/o pooling	0.3898	0.3822
$2 \times 2, 2 \times 2$	0.3877	0.3932

Table 4.5: Training error and validation error on the spectrogram set with different pooling sizes

Filter size

The various filter sizes of the second layer are investigated with the fixed filter size of 3×3 for the first convolutional layer. The results show the filter size in time domain sensitively affects the training error. The lowest validation error is acquired when the second convolutional filter size is 15×3 frames as seen in figure 4.6. The second filter size in the frequency domain is also investigated with the fixed size of 3×3 frames for the first convolutional layer and the fixed size of 15 frames in the time domain for the second convolutional layer. As seen in figure 4.7, the results show the second filter size of 15×3 and 15×7 have the minimum validation error. The filter size of 15×3 is applied to the following experiments since less filter size reduces the amount of computation.

Number of filters

In this experiment, various numbers of filters are investigated. The same number of filters are applied to both convolutional layers to simplify experiments. The result shows the 60 filters provide the lowest validation error and also the gap between the training and validation error is the smallest as seen in figure 4.8. Since overffiting is not seen when 60 filters are applied, dropout is not applied to the following experiments.

4.3.1 CCC on the test set

The optimal network for spectrogram input is decided by the previous experiments. The first hidden layer is a convolutional layer including 60 filters with the size of 3×3 frames.

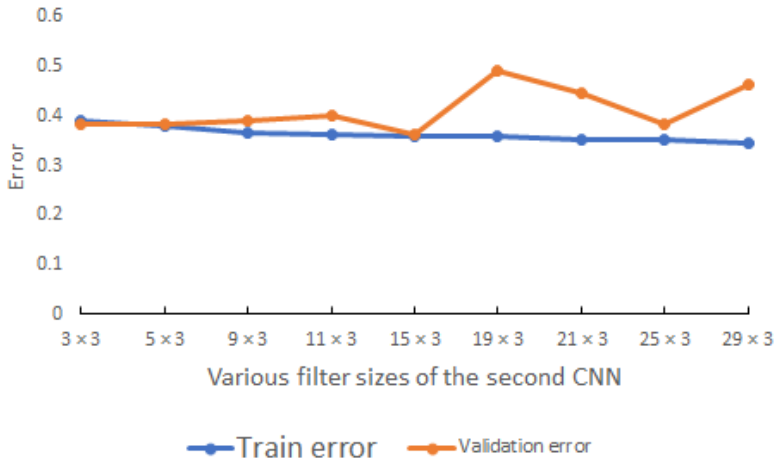

Figure 4.6: Training and validation error depending on various filter sizes on the spectrogram set. The filter size of the first convolutional layer is fixed to 3×3.

The second hidden layer is a max pooling layer with the size of 4×4 frames. The third hidden layer is a convolutional layer including 60 filters with the size of 15×3 frames. Through those three layers, input features for RNNs with LSTM are extracted from the spectrogram input. Dropout is not applied to RNNs with LSTM for the spectrogram input. The test set is fed to this optimal network and the result is seen in table 4.6.

window size in seconds	training CCC	validation CCC	test CCC
2 s	0.5507	0.4574	0.4938
4 s	0.6397	0.6362	0.5986
6 s	0.7194	0.6555	0.6990

Table 4.6: CCC on the spectrogram set with the optimal network

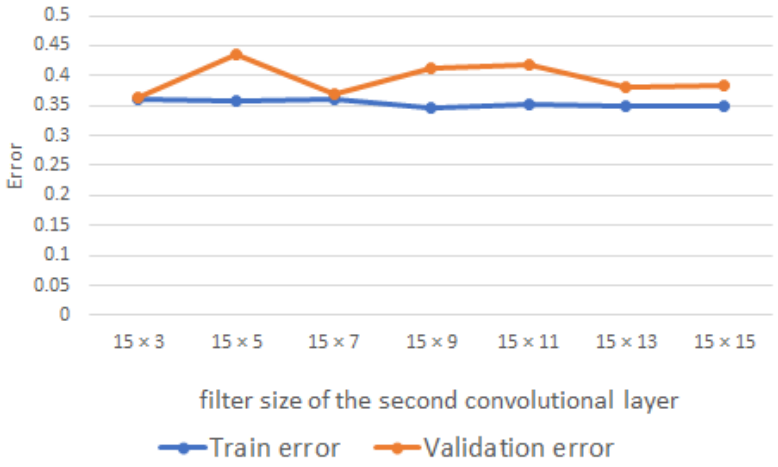

Figure 4.7: Training and validation error depending on various filter sizes in the frequency domain on the spectrogram set. The filter size of the first convolutional layer is fixed to 3×3 frames. The filter size of the second convolutional layer in the time domain is fixed to 15 frames.

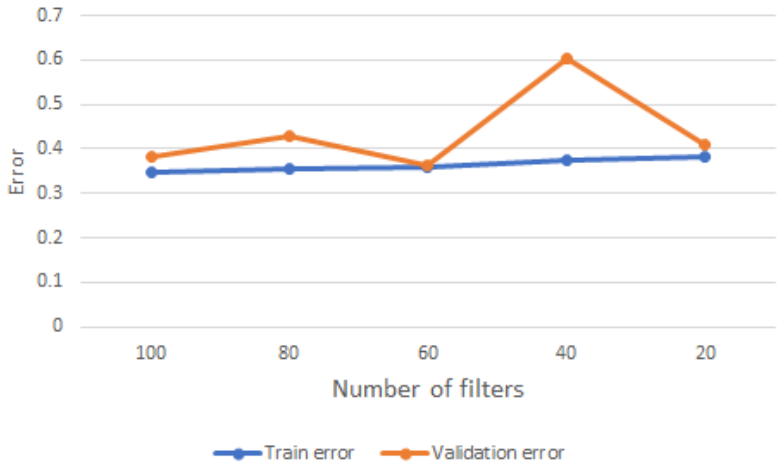

Figure 4.8: Training and validation error depending on various numbers of filters in CNNs on the spectrogram set.

5 Conclusion

In this thesis, three different structures of neural networks are investigated on three different input types, respectively, to compare the performance of an end-to-end time-continuous emotion recognition task. Performances are measured by concordance correlation coefficient. Therefore, experiments are conducted to find the optimal hyperparameters that generalize the performance in terms of concordance correlation. First, The recurrent neural networks are investigated on the feature set that includes 65 acoustic Low-Level Descriptors and their first order derivate same as the RECOLA database. The z-score normalization provides better performance than mean-standard deviation. The z-score normalization decreases 99% of the training error on the raw feature set. Softsign function is considered as the optimal activation function of the recurrent neural network, since it has the smallest gap between the training and validation error among ReLU, Tanh, and softsign. The Gaussian input noise with standard deviation of 0.1 and dropout with probability 0.2 minimize the gap between the training and validation error. generalization. With the chosen hyperparameters, the best result shows the CCC of 0.765 and 0.7259 on the validation set and the test set, respectively. Although the best CCC on the feature set of this thesis is relatively smaller than the CCC of 0.788 in [2], it can not be directly compared since the dataset used in this thesis includes fewer data points than the gold standard.

The one-dimensional convolutional network is also investigated to find the optimal network on the raw waveform. Due to hardware limitations, the experiments in this thesis conduct a way of reducing the number of parameters to be computed and improving performance in terms of CCC. Therefore, not a filter size of the first convolutional layer but a filter size of the second convolutional layer is investigated. The first convolutional layer with filter size of 3 frames, the first max pooling layer with pool size of 32 frames, the second convolutional with filter size of 15, and the second max pooling layer with pool size of 8 frames provide the best result of 0.7284 and 0.7517 in CCC on the training waveform set and the validation waveform set, respectively. The CCC on the validation set and test set are 0.741 and 0.686 in recent research [4]. Although less sampling rate of the raw speech signal and fewer number of data points are used in this thesis, than the research [4], the network on the waveform set in this thesis performs better.

The two-dimensional convolutional network is investigated on the spectrogram input set. Experiments are conducted to find the optimal filter size of the second convolutional layer with the fixed filter size of the first convolutional layer. The optimal network on the spectrogram input set includes two convolutional layers and a max pooling layer. The first convolutional layer with 60 filters that has filter size of 3×3 frames, the max pooling layer with pooling size of 4×4 frames, and the second convolutional layer with 60 filters that has filter size of 15×3 provide the best CCC $0.7194, 0.6555, 0.6999$ on the training

set, the validation set, the test set, respectively. Since most of the research for emotion recognition on the spectrogram performs to recognize the discrete category of emotions [9] and [10], no research is found to be compared.
Although different sizes of the RECOLA database were used in this thesis and the existing one [4], models for the waveform set and the spectrogram set in this thesis achieved compatible performance.

Bibliography

[1] Rosalind W. Picard. *Affective computing*. MIT Press, 2000.

[2] Fabien Ringeval, Florian Eyben, Eleni Kroupi, Anil Yuce, Jean-Philippe Thiran, Touradj Ebrahimi, Denis Lalanne, and Björn Schuller. Prediction of Asynchronous Dimensional Emotion Ratings from Audiovisual and Physiological Data. *Pattern Recogn. Lett.*, 66(C):22–30, November 2015.

[3] Florian Eyben, Martin Wöllmer, and Björn Schuller. Opensmile: The munich versatile and fast open-source audio feature extractor. In *Proceedings of the 18th ACM International Conference on Multimedia*, MM '10, pages 1459–1462, New York, NY, USA, 2010. ACM.

[4] George Trigeorgis, Fabien Ringeval, Raymond Brueckner, Erik Marchi, Mihalis A Nicolaou, Björn Schuller, and Stefanos Zafeiriou. Adieu features? end-to-end speech emotion recognition using a deep convolutional recurrent network. In *Acoustics, Speech and Signal Processing (ICASSP), 2016 IEEE International Conference on*, pages 5200–5204. IEEE, 2016.

[5] Michel Valstar, Jonathan Gratch, Björn Schuller, Fabien Ringeval, Denis Lalanne, Mercedes Torres Torres, Stefan Scherer, Giota Stratou, Roddy Cowie, and Maja Pantic. Avec 2016: Depression, mood, and emotion recognition workshop and challenge. In *Proceedings of the 6th International Workshop on Audio/Visual Emotion Challenge*, AVEC '16, pages 3–10, New York, NY, USA, 2016. ACM.

[6] J. Sauer F. Ringeval, A. Sonderegger and D. Lalanne. Introducing the RECOLA Multimodal Corpus of Remote Collaborative and Affective Interactions. *2nd International Workshop on Emotion Representation, Analysis and Synthesis in Continouous Time and Space(EmoSPACE)*, 2013.

[7] Fabien Ringeval, Björn Schuller, Michel Valstar, Shashank Jaiswal, Erik Marchi, Denis Lalanne, Roddy Cowie, and Maja Pantic. Av+ec 2015: The first affect recognition challenge bridging across audio, video, and physiological data. In *Proceedings of the 5th International Workshop on Audio/Visual Emotion Challenge*, AVEC '15, pages 3–8, New York, NY, USA, 2015. ACM.

[8] Tara N Sainath, Ron J Weiss, Andrew Senior, Kevin W Wilson, and Oriol Vinyals. Learning the speech front-end with raw waveform cldnns. In *Sixteenth Annual Conference of the International Speech Communication Association*, 2015.

[9] Jonathan Chang and Stefan Scherer. Learning representations of emotional speech with deep convolutional generative adversarial networks. *CoRR*, abs/1705.02394, 2017.

[10] Jaebok Kim, Khiet P. Truong, Gwenn Englebienne, and Vanessa Evers. Learning spectro-temporal features with 3d cnns for speech emotion recognition. *CoRR*, abs/1708.05071, 2017.

[11] Qirong Mao, Ming Dong, Zhengwei Huang, and Yongzhao Zhan. Learning salient features for speech emotion recognition using convolutional neural networks. *IEEE Transactions on Multimedia*, 16(8):2203–2213, 2014.

[12] Ian Goodgellow, Yoshua Bengio, and Aaron Courville. *Deep Learining*. MIT Press, 2016. `http://www.deeplearningbook.org`.

[13] Andrew Ng. Machine learning. Available at http://cs229.stanford.edu/notes/cs229-notes1.pdf, 2012.

[14] Simon Haykin. *Neural networks:a comprehensive foundation*. Prentice Hall, 2nd edition, 1999.

[15] Alex Graves. Supervised sequence labelling with recurrent neural networks. In *Studies in Computational Intelligence*. Springer, 2012.

[16] Xavier Glorot and Yoshua Bengio. Understanding the difficulty of training deep feedforward neural networks. In Yee Whye Teh and Mike Titterington, editors, *Proceedings of the Thirteenth International Conference on Artificial Intelligence and Statistics*, volume 9 of *Proceedings of Machine Learning Research*, pages 249–256, Chia Laguna Resort, Sardinia, Italy, 13–15 May 2010. PMLR.

[17] Christopher M. Bishop. *Neural Networks for Pattern Recognition*. Oxford University Press, Inc., 1995.

[18] Richard O. Duda, Peter E. Hart, and David G. Stork. *Pattern Classification*. Wiley-Interscience, 2nd edition, 2000.

[19] Nitish Srivastava, Geoffrey Hinton, Alex Krizhevsky, Ilya Sutskever, and Ruslan Salakhutdinov. Dropout: A simple way to prevent neural networks from overfitting. *The Journal of Machine Learning Research*, 15(1):1929–1958, 2014.

[20] T Jayalakshmi and A Santhakumaran. Statistical normalization and back propagationfor classification. *International Journal of Computer Theory and Engineering*, 3(1):89, 2011.

[21] Jonathan Posner, James A Russell, and Bradley S Peterson. The circumplex model of affect: An integrative approach to affective neuroscience, cognitive development, and psychopathology. *Development and psychopathology*, 17(3):715–734, 2005.

[22] Felix Weninger, Florian Eyben, Björn Schuller, Marcello Mortillaro, and Klaus Scherer. On the acoustics of emotion in audio: What speech, music, and sound have in common. *Frontiers in Psychology*, 4:292, 2013.

[23] Jo-Anne Bachorowski Michael J. Owren. Measuring emotion-related vocal acoustics. In *Handbook of emotion elicitation and assessment*, Series in affective science, pages 239–266, New York, 2007. Oxford University Press.

[24] Geoffrey Hinton Tijmen Tieleman. Lecture 6.5-rmsprop: Divide the gradient by a running average of its recent magnitude. *COURSERA: Neural networks for machine learning*, 4(2):26–31, 2012.