Insert here your thesis' task.

FACULTY
OF INFORMATION
TECHNOLOGY
CTU IN PRAGUE

Bachelor's thesis

Gesture detector with Leap Motion sensor

Anh Tran Viet

Department of Theoretical Computer Science
Supervisor: Tomáš Nováček

February 24, 2021

Acknowledgements

THANKS (remove entirely in case you do not with to thank anyone)

Declaration

I hereby declare that the presented thesis is my own work and that I have cited all sources of information in accordance with the Guideline for adhering to ethical principles when elaborating an academic final thesis.

I acknowledge that my thesis is subject to the rights and obligations stipulated by the Act No. 121/2000 Coll., the Copyright Act, as amended, in particular that the Czech Technical University in Prague has the right to conclude a license agreement on the utilization of this thesis as a school work under the provisions of Article 60 (1) of the Act.

In Prague on February 24, 2021 . . . . . .

Czech Technical University in Prague
Faculty of Information Technology

Citation of this thesis

Tran, Anh Viet. *Gesture detector with Leap Motion sensor.* Bachelor's thesis. Czech Technical University in Prague, Faculty of Information Technology, 2021.

Abstrakt

V několika větách shrňte obsah a přínos této práce v českém jazyce.

Klíčová slova Replace with comma-separated list of keywords in Czech.

Abstract

Summarize the contents and contribution of your work in a few sentences in English language.

Keywords Replace with comma-separated list of keywords in English.

Contents

List of Figures

Introduction

Mouse and keyboard are considered to be default devices for human-computer interaction nowadays. But with the maturity in technology, namely virtual and extended reality, the computer's need to understand human's body language is more and more present. Actions such as rotation or grabbing and moving an object in three-dimensional space with a computer mouse are unintuitive. They require a little understanding of the controls to execute the task. The movement is limited to the two-dimensional space of the mouse. Oppose to performing the desired action by hands in our three-dimensional space as we would in real life.

One of the proposed solutions for the issue is gesture recognition, where a general idea is for computers to have the ability to recognize gestures and perform actions based on them. Therefore, several devices, tracking devices, were developed to process an image and yield useful data for gesture recognition.

Our goal is to utilize these tracking devices, specifically Leap Motion controllers, combined with artificial neural networks, creating a simple library with a pre-trained model ready to be used and expanded by other applications.

Chapter 1

Neural Networks

An artificial neural network (ANN) is a mathematical model mimicking biological neural networks, namely their ability to learn and correct errors from previous experience [1], [2].

The ANN subject was first introduced by Warren McCulloch and Walter Pitts in "A logical calculus of the ideas immanent in nervous activity" published in 1943 [3]. But it was not until recent years when ANN has gained popularity with still increasing advancements in technology and availability of training data. ANN had become one of the default solutions for complex tasks which were previously thought to be unsolvable by computers [4].

This chapter will briefly explore different types of neural units and their activation functions, along with some exemplary network architectures.

1.1 Artificial Neuron

As previously mentioned, artificial neurons are units mimicking behavior of biological neurons. Meaning, it can receive as well as pass information between themselves.

1.1.1 Perceptron

Perceptron is the simplest class of artificial neurons developed by Frank Rosenblatt in 1958 [5].

Perceptron takes several binary inputs, vector $\vec{x} = (x_1, x_2, ..., x_n)$, and outputs a single binary number. To express the importance of respected input edges, perceptron uses real numbers called *weights*, assigned to each edge, vector $\vec{w} = (w_1, w_2, ..., w_n)$.

A *step function* calculates the perceptron's output. The function output is either 0 or 1 determined by whether its weighted sum $\alpha = \sum_i x_i w_i$ is less

or greater than its *threshold* value, a real number, usually represented as an incoming edge with a negative weight -1 [6].

$$output = \begin{cases} 1, & \text{if } \alpha \geq threshold \\ 0, & \text{if } \alpha < threshold \end{cases} \tag{1.1}$$

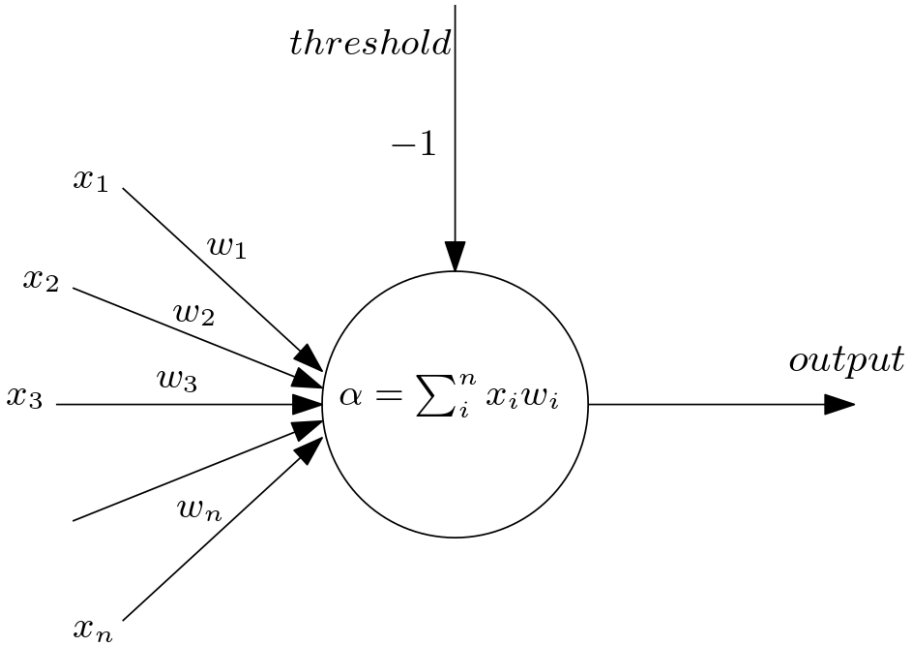


Figure 1.1: Perceptron [6]

1.1.2 Sigmoid Neuron

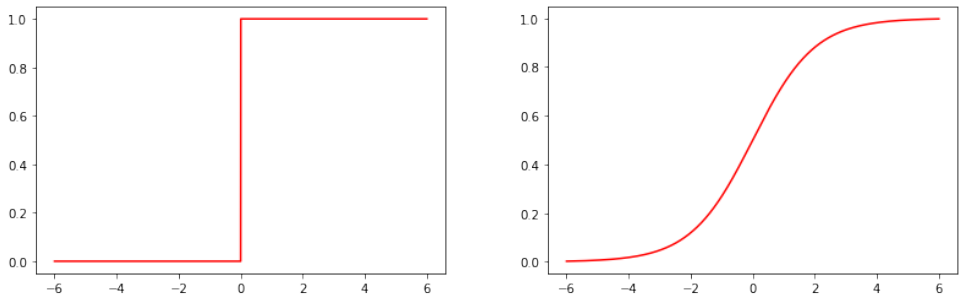
Sigmoid neuron, similarly to perceptron, has inputs $\vec{x}$ and weights. The key difference comes in once we inspect the output value and its calculation. Instead of perceptron's binary output 0 or 1, a sigmoid neuron outputs a real number between 0 and 1 using a *sigmoid function* [7], [8], [6].

$$\sigma(\alpha) = \frac{1}{1 + e^{-\alpha}} \tag{1.2}$$

As shown in Figure 1.1, the sigmoid function(1.1a) is a smoothed-out version of the step function(1.1b).

1.1.3 Activation Function

An artificial neuron's activation function defines that neuron's output value for given inputs, commonly being $f : \mathbb{R} \to \mathbb{R}$ [9]. A significant trait of many activation functions is their differentiability, allowing them to be used for



(a) Step function

(b) Sigmoid function

Figure 1.2: Comparison between step function and sigmoid function

Backpropagation, ANN algorithm for training weights. The activation function needs to have a derivative that does not saturate by heading towards 0, or explode by heading towards inf [6].

For such reasons, the usage of step function or any linear function is unsuitable for ANN.

1.1.3.1 Sigmoid Function

The sigmoid function is commonly used in ANN as an alternative to the step function. A popular choice of the sigmoid function is a *logistic sigmoid*. Its output value is in the range of 0 and 1.

$$\sigma(\alpha) = \frac{1}{1 + e^{-\alpha}} = \frac{e^x}{1 + e^x} \tag{1.3}$$

One of the reasons for its popularity is the simplicity of its derivative calculation:

$$\frac{d}{dx}\sigma(\alpha) = \frac{e^x}{(1 + e^x)^2} = \sigma(x)(1 - \sigma(x)) \tag{1.4}$$

On the other hand one of its disadvantages is the *vanishing gradient*. A problem where for a given very high or very low input values, there would be almost no change in its prediction. Possibly resulting in training complications or performance issues [10], [6].

1.1.3.2 Hyperbolic Tangent

Hyperbolic tangent is similar to logistic sigmoid function with a key difference in its output, ranging between -1 and 1.

$$tanh(x) = \frac{e^x - e^{-x}}{e^x + e^{-x}} \tag{1.5}$$

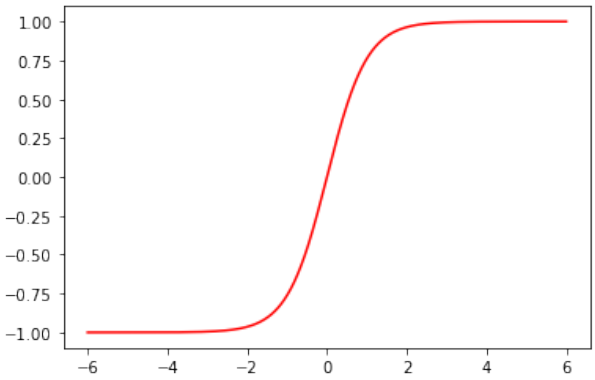


Figure 1.3: Hyperbolic tangent [6]

It shares sigmoid's simple calculation of its derivative.

$$\frac{d}{dx}\tanh(x) = 1 - \frac{(e^x - e^{-x})^2}{(e^x + e^{-x})^2} = 1 - \tanh^2(x) \tag{1.6}$$

By being only moved and scaled version of the sigmoid function, hyperbolic tangent does share sigmoid's advantages but also its disadvantages [9], [6].

1.1.3.3 Rectified Linear Unit

The output of the rectified linear unit (ReLU) is defined as:

$$f(x) = max(0, x) \begin{cases} x, & \text{if } x \geq 0 \\ 0, & \text{if } x < 0 \end{cases} \tag{1.7}$$

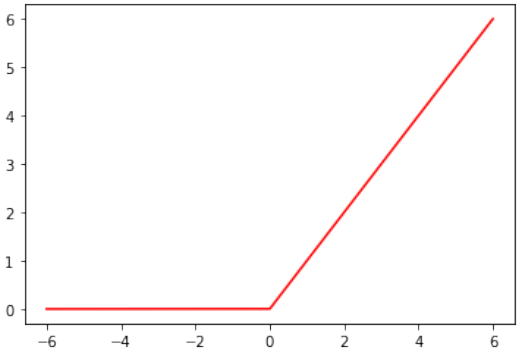


Figure 1.4: Rectified Linear Unit [6]

ReLU's popularity is mainly due to its computational efficiency [10]. Its disadvantages appear when inputs approach zero to or are negative number. Causing the so-called dying ReLu problem, where the network is unable to

learn. There are many variations of ReLu to this date, e.g., Leaky ReLU, Parametric ReLU, ELU, ...

1.1.3.4 Softmax

Softmax separates itself from all the previously mentioned functions by its ability to handle multiple input values in the form of a vector $\vec{x} = (x_1, x_2, ..., x_n)$ and output for each x_i defined as:

$$\sigma(x_i) = \frac{e_i^x}{\sum_{j=1}^{n} e_j^x} \tag{1.8}$$

For output being normalized probability distribution, ensuring $\sum_i \sigma(x_i) = 1$ [11]. It is being used as the last activation function of ANN to normalize the network's output into n probability groups.

1.2 Types of Neural Networks

To this day, there are many types and variations of ANN, each with its structure and use cases. Here we will briefly introduce the most common ones, such as feed-forward networks, convolutional neural networks, or recurrent neural networks.

1.2.1 Feed-forward Networks

Feed-forward network (FFN) was the first ANN to be invented and the simplest form of ANN. Its name comes from the way how the information flows through the network. Its data travels in one direction, oriented from the *input layer* to the *output layer*, without cycles. The input layer takes input data, vector $\vec{x}$, producing $\hat{y}$ at the output layer [12].

FFN may or may not contain several hidden layers of various widths. By having no back-loops, FFN generally minimizes error, computed by *cost function*, in its prediction by using the *backpropagation* algorithm to update its weight values [13], [11].

1.2.1.1 Cost Function

Cost function $C(\vec{w})$ is used in ANN's training process. It takes all weights and biases of an ANN as its input, in the form of a vector $\vec{w}$ and calculates a single real number expressing ANN's incorrectness [14]. The number is high when the ANN performs poorly and gets lower when the ANN's output gets closer to the correct result. The main goal of training is then to minimize the cost function.

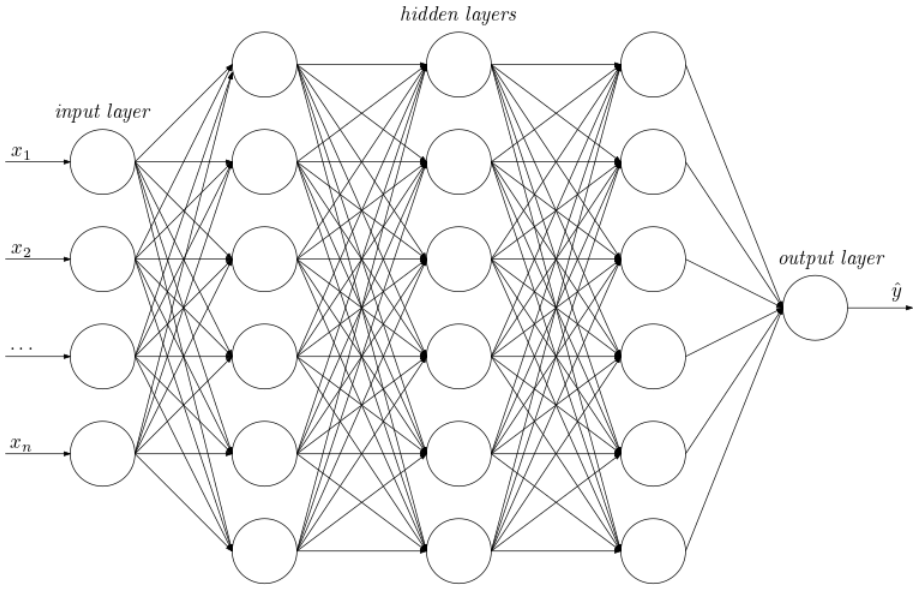


Figure 1.5: Fully connected Feed-forward Neural Network [6]

1.2.1.2 Backpropagation

Backpropagation, short of backward propagation of errors, is a widely used algorithm in training FFN using *gradient descent* to find a local minimum of a cost function and update ANN's weights [15].

A gradient of a function with multiple variables gives us the direction of the steepest gradient ascent, where should we step to increase the output quickly and find the local maximum. Naturally, its negative will point towards a local minimum.

The usual practice is to divide training samples into small *batches* of size n. We will calculate a gradient descent for each sample in the batch and use their average gradient descent to update the network's weights. The average gradient descent tells us which weights should be adjusted for the ANN to get closer to the correct results [15].

$$-\gamma \nabla C(\vec{w}_i) + \vec{w}_i \rightarrow \vec{w}_{i+1} \tag{1.9}$$

Here, $\vec{w}_i$ are weights of the network at the current state (batch), $\vec{w_{i+1}}$ are updated weights, γ is the learning rate and $-\nabla C(\vec{w}_i)$ is the gradient descent.

1.2.2 Convolutional Neural Networks

Convolutional Neural network's (CNN) main goal is to make a computer recognize images and objects. For such, it is primarily used for image classification or object recognition.

CNN was inspired by the biological processes of the human brain. Its connectivity patterns resemble the human's visual cortex. But an image is perceived differently by a human brain than by a computer. To a computer, an image is interpreted as an array of numbers. Thus CNN is designed to work with two-dimensional image arrays, although it is possible to work with one-dimensional or three-dimensional arrays too [16].

CNN is a variation of FNN [14]. It usually consists of the input layer followed by multiple hidden layers, typically several *convolutional layers* with standard *pooling layers*, and ending with the output layer.

1.2.2.1 Convolutional Layer

The convolutional layer's objective is to extract key features from the input image by passing a matrix known as a *kernel* over the input image abstracted into a matrix [17].

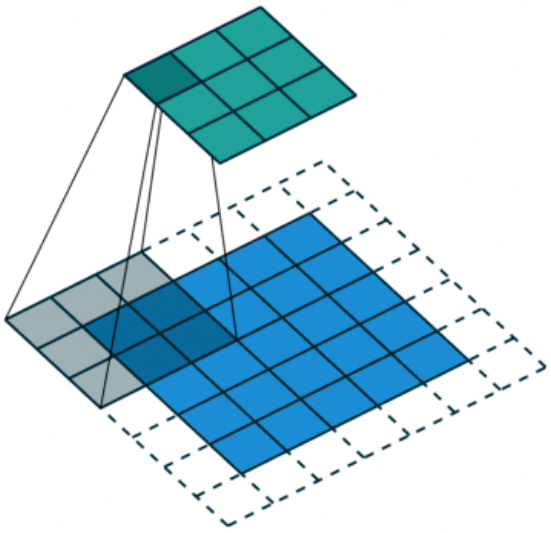


Figure 1.6: Convolution of an 5x5x1 image with 3x3x1 kernel [18]

The convolution result can be of two types depending on their size. One being the convolved feature is reduced in dimensions compared to the input, *valid padding*. For example, an input image of dimensions 8x8 being reduced to 6x6 after convolution operation, and the other type being where dimensions are either increased or remain the same, *same padding* [18].

1.2.2.2 Pooling Layer

Similar to the previously mentioned convolutional layer, the pooling layer reduces the convolved feature's spatial size to decrease the computational power

required for data processing. Furthermore, being useful by extracting dominant features, which are rotational and positional invariant, thus maintaining the process of effectively training the model [18].

There are two types of pooling: *max pooling* and *average pooling*. Max pooling returns the maximum value from the portion of the image covered by the kernel. It performs as a noise suppressant, discarding the noisy activations altogether and performing de-noising and dimensionality reduction. Where average pooling returns the average of all the values from the same covered portion, performing dimensionality reduction as a noise suppressing mechanism. Hence, it is possible to note that max-pooling performs better [18].

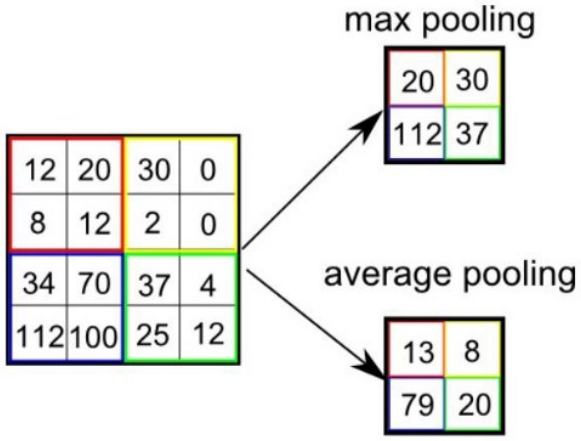


Figure 1.7: Types of pooling [18]

1.2.3 Recurrent Neural Networks

Recurrent Neural Network (RNN) is distinct by its memory, taking input sequence with no predetermined size. Its past predictions influence currently generated output. Thus for the same input, RNN could produce different results depending on previous inputs in the sequence [19].

RNNs features make it commonly used in fields such as speech recognition, image captioning, natural language processing, or language translation. Some of the popular being, for example, Siri, Google Translate or Google Voice search [20].

As previously mentioned, RNN takes into consideration information from previous inputs. Let us look at the idiom ”feeling under the weather”, where for it to make sense, words have to be in a specific order. RNN needs to account for each word’s positions and use its information to predict the next word in the sequence. Each timestep represents a single word. In our case, the

third timestep represents ”the”. Its hidden state holds information of previous inputs, ”feeling” and ”under” [20].

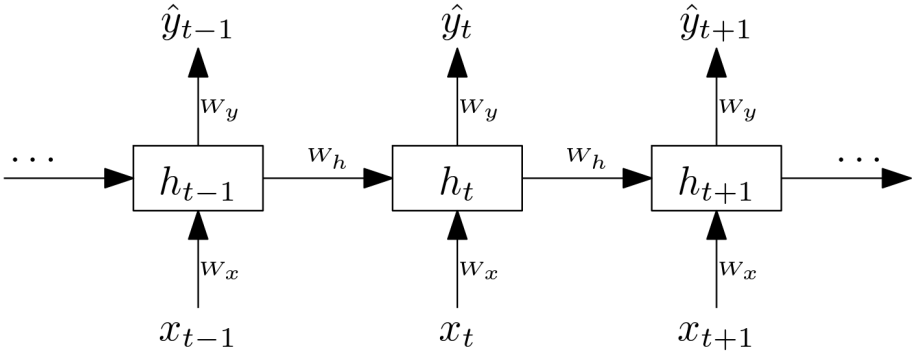


Figure 1.8: Unrolled structure of RNN [6]

Figure 1.4 shows the network for each timestep, i.e., at time t, the input $\vec{x_t}$ goes into the network to produce output $\hat{y}_t$, the next timestep of the input is x_{t+1} with additional input from the previous time step from the hidden state h_t. This way, the neural network looks at the current input and has the context from the previous inputs. With this structure, recurrent units hold the past values, referred to as memory. Making it possible to work with a context in the data [21].

The recurrent unit is calculated as follows:

$$h_t = f(W_x x_t + W_h h_{t-1} + \vec{b_h}) \tag{1.10}$$

$f()$ being the activation function, W_x, W_h are weight matrixes, x_t is the input, and $\vec{b_h}$ is the vector of bias parameters. Unit at time step $t = 0$ is initialized to $(0, 0, ..., 0)$. The output $\hat{y}_t$ is then calculated as:

$$\hat{y}_t = g(W_y h_t + \vec{b_y}) \tag{1.11}$$

$g()$ also being an activation function, usually being softmax to ensure the output is in the desired class range. W_y is the weight matrix and $\vec{b_y}$ being a vector of biases determined during the learning process.

Training RNNs uses a modified version of the backpropagation algorithm called *backpropagation through time* (BPTT), working by unrolling the RNN [14], calculating the losses across time steps, then updating the weights with the backpropagation algorithm. More on RNN in [11] by Lipton et al.

1.2.3.1 Bidirection Recurrent Neural Networks

Bidirectional Recurrent Neural Networks (BRNN) allow training the network using all available input information in the past and future of a specific time frame. Oppose to regular RNN, where its hidden state is determined only by

the prior states. The idea behind BRNN is splitting the hidden state into two. One is responsible for the positive time direction, *forward states*, and the other for the negative time direction, *backward states*.

BRNN's training generally starts with processing forward, and backward states before output neurons are passed, *forward pass*. Following with *backward pass*, where output neurons are processed first, and forward and backward states after. Weights are then updated after completing forward pass and backward pass [22].

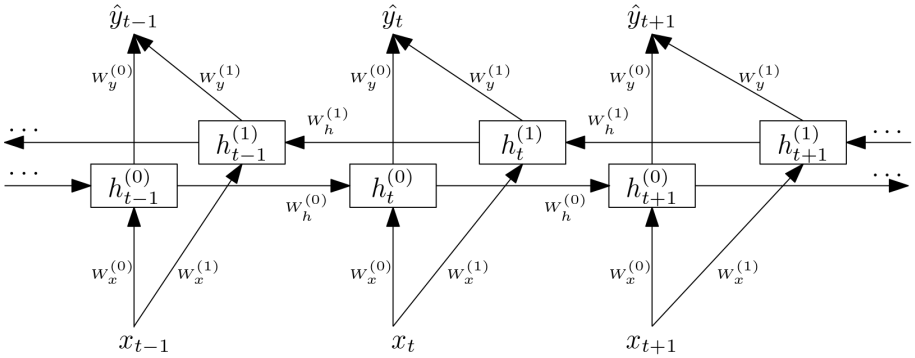


Figure 1.9: Unrolled structure of BRNN [6]

Both hidden states are updated identically as the hidden state in RNN.

$$h_t^{(0)} = f(W_x^{(0)} x_t + W_h^{(0)} h_{t-1} + \vec{b_{h^{(0)}}}) \tag{1.12}$$

$$h_t^{(1)} = f(W_x^{(1)} x_t + W_h^{(1)} h_{t-1} + \vec{b_{h^{(1)}}}) \tag{1.13}$$

The output is then computed in combination of both hidden states.

$$\hat{y}_t = g(W_y^{(0)} h_t^{(0)} + W_y^{(1)} h_t^{(1)} + \vec{b_y}) \tag{1.14}$$

All the activation functions and parameters remain the same as they were in RNN.

1.2.4 Long Short-Term Memory

Consider a task where we are trying to predict the last word in "The clouds are in the *sky*". It is fairly obvious the last word is meant to be "*sky*". The gap between the relevant information and the prediction place is small, and RNN can learn to utilize past information and predict the last word. However, if we consider "I grew up in Spain... I speak fluent *Spanish*", the gap between the relevant information and predicting word can become large. As the gap grows, RNNs are unable to handle the task. Such problem is called *long-term dependencies* [23].

Long Short Term Memory networks (LSTM) are RNN architecture first introduced by Hochreiter S. and Schmidhuber J. [24] with the ability to handle long-term dependencies. Its core idea is to replace RNN's hidden states with so-called **LSTM Cells** and add connections between cells, called cell states or c_t. Each LSTM Cell consists of three gates, regulating the input and output of the cell. The calculation in each cell runs as follows:

1. **Forget Gate**: Controls which information should be discarded and which kept. *Sigmoid function* outputs a value between 0 and 1 base on the information from the previous hidden state and from the current input. The value closer to 0 means discard, and closer to 1 means keep.

$$f_t = \sigma(W_{x_f} x_t + W_{h_f} h_{t-1} + \vec{b_f}) \tag{1.15}$$

2. **Input Gate**: Decides which information should be updated. The sigmoid function outputs a value between 0 and 1 base on the previous hidden state and current input state. Closer to 0 means not important, and closer to 1 means important.

$$i_t = \sigma(W_{x_i} x_t + W_{h_i} h_{t-1} + \vec{b_i}) \tag{1.16}$$

The information from the previous hidden state and current input state is also passed into a *tanh function*, getting values between -1 and 1.

$$g_t = \tanh(W_{x_g} x_t + W_{h_g} h_{t-1} + \vec{b_g}) \tag{1.17}$$

The decision on how to update the cell is obtained by multiplying sigmoid output and tanh output. With all the required values available, we can now calculate the **cell state** as follows:

$$c_t = i_t \odot g_t + f_t \odot c_{t-1} \tag{1.18}$$

3. **Output Gate**: Determines what information should the next hidden state contain. The previous hidden state and the current input are passed into a sigmoid function.

$$o_t = \sigma(W_{x_o} x_t + W_{h_o} h_{t-1} + \vec{b_o}) \tag{1.19}$$

Then passing the newly modified cell state into a tanh function, and multiplying its output with the sigmout ouput, we get the hidden state [25].

$$h_t = o_t \odot tanh(c_t) \tag{1.20}$$

The computation of the output $\hat{y}_t$ proceeds the same way as regular RNN [6].

$$\hat{y}_t = g(W_y h_t + \vec{b_y}) \tag{1.21}$$

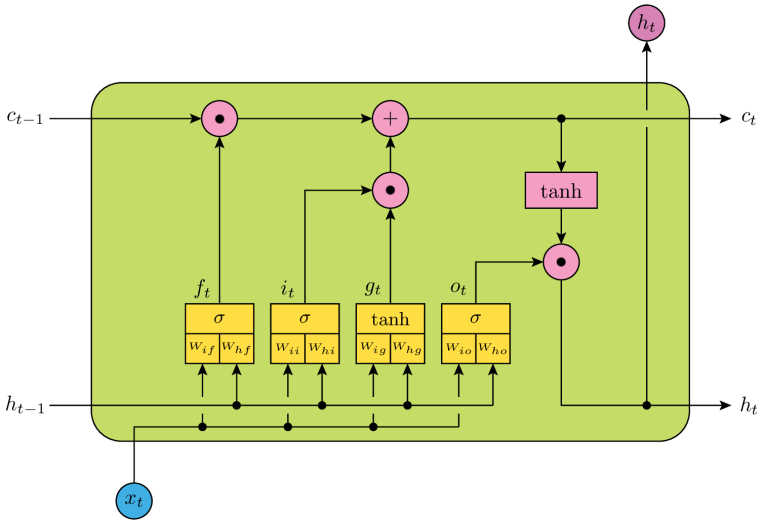


Figure 1.10: LSTM cell [26]

1.2.4.1 Bidirectional Long Short-Term Memory

Similarly, as previously described in BRNN (1.2.3.1), Bidirectional Long Short-Term Memory (BLSTM) has its hidden state split into two, forward states and backward states. Such modification allows the network to gain context from past and future alike. BLSTM, in comparison with BRNN, handles better the information storage across the timeline with large time gaps from either past or future.

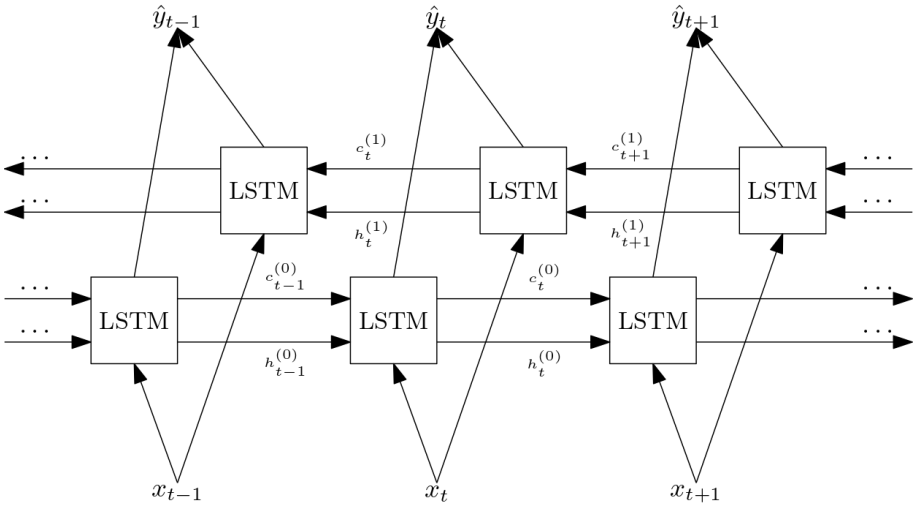


Figure 1.11: Unrolled structure of BLSTM [6]

1.2.4.2 Deep Long Short-Term Memory

Deep Long Short-Term Memory (DLSTM), or stacked LSTM, is now considered to be a stable technique for challenging sequence prediction tasks. It was first introduced by Graves, et al. [27], where it was found that the depth of the network has greater importance than the number of memory cells in a given layer. DLSTM architecture can be described as an LSTM model consisting of multiple LSTM layers.

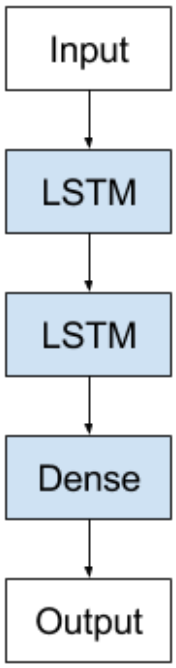


Figure 1.12: Deep Long Short-Term memory architecture [28]

The LSTM layer above outputs a sequence rather than a single value for the LSTM layer below [28].

Chapter 2

Gesture Recognition

2.1 Gesture Categories

Gestures are categorized into *static gestures* and *dynamic gestures*. A Group of static gestures consists of fixed gestures, where they are not relative to time. A group of dynamic gestures, on the other hand, are time-varying. These classes can be further subdivided into a set of gestures distinct by their purpose.

- **Deictic gestures** involve pointing to establish the identity or spatial location of an object within the context of the application domain [29].
- **Manipulative gestures** mimic manipulation of a physical object, such as scaling, moving, or rotating.
- **Gesticulation** is commonly used along with the language group. These hand gestures are difficult to analyse.
- **Language group of hand gestures** form a grammatical structure for conversational style interfaces.
- **Semaphoric hand gestures** also may be referred to as communicative gestures, are a group of hand gestures serving as a set of symbols/commands used to interact with machines. The group consists of static hand gestures as well as dynamic hand gestures.

2.2 Tracking devices

Hand and body gesture recognition had followed a conventional scheme of extracting key features via one or multiple preprocessing sensors and applying machine learning techniques on them [30]. The field of gesture recognition gave birth to several image processing devices yielding useful data.

2.2.1 Microsoft Kinect

One of them being Micorosoft Kinect, a device first released in 2010. Originally developed for gaming but eventually finding more success in academics and commercial applications, such as robotics, medicine, and health care, it led Microsoft to discontinue production of its Xbox version in 2018 and release Azure Kinect in March 2020, incorporating Microsoft Azure cloud computing functionalities.

Figure 2.1: Azure Kinect [31]

Azure Kinect contains a depth sensor, spatial microphone array with a video camera, and orientation sensor as a small all-in-one device with multiple modes, options, and software development kits [32].

With all that said, the primary purpose of the Kinect device overall is to interpret whole-body movement. For such, it lacks in required accuracy for hand gesture recognition, thus making it insufficient for our uses.

2.2.2 Leap Motion Controller

Another option would be using a Leap Motion Controller (LMC), developed specifically to track hand movements and extract its features, such as positions of fingers, hand rotation, and others.

LMC consists of two monochromatic IR cameras and three IR LEDs (emitters).

The LMC's current API, Leap Motion Service, yields positions of extracted hand features. All the positional data about the hand and its features are represented in the coordinate system relative to the LMC's center point, positioned at top of the controller [33]. The x- and z-axes lie in the camera sensors plane, with the x-axis running along the camera baseline. The y-axis is vertical, with positive values increasing upwards (in contrast to the downward orientation of most computer graphics coordinate systems). The z-axis has positive values increasing toward the user [34].

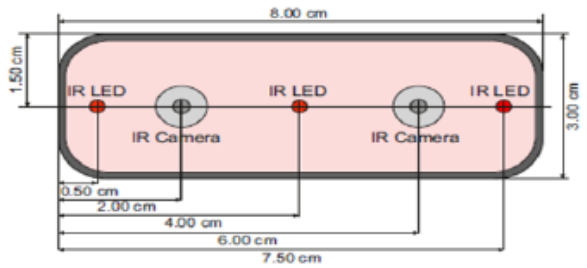


Figure 2.2: Schematic View of Leap Motion Controller [33]

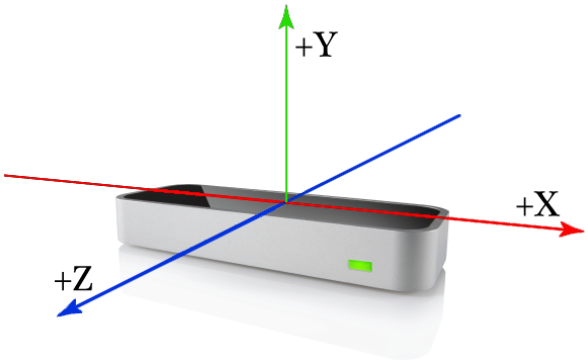


Figure 2.3: Leap Motion Controller Axes [34]

Unfortunately, Leap Motion Controller has no official library for gesture recognition, limiting developers from utilizing the controller for its key features. Leap Motion provided tracking software built for virtual reality, used to have a gesture detector with its 3.0 version, but the detector is absent with the release of more accurate version 4.0.

2.2.3 Ultraleap Stereo IR 170

Ultraleap Stereo IR 170, formerly known as the Leap Motion Rigel, is the successor to the Leap Motion controller.

The Stereo IR inherits Leap Motions key features but improves with a wider 170-degree field of view, more-powerful LED illuminators providing more extended tracking range, and a higher framerate when used with USB 3.0 [35], [36].

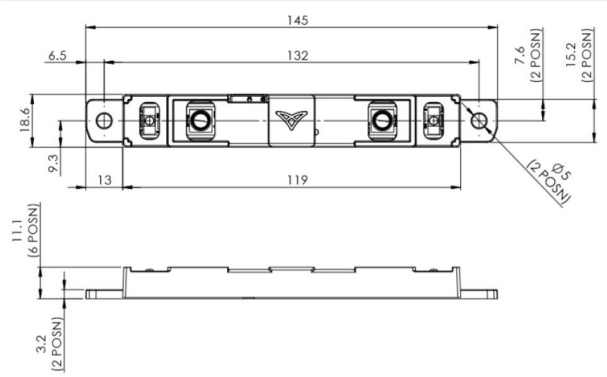


Figure 2.4: Schematic View of Ultraleap Stereo IR 170 [35]

2.3 Gesture Recognition Methods

Gestures group classification should be taken into account when choosing appropriate methods due to their time-varying properties. As previously mentioned, gestures are classified into static and dynamic groups.

2.3.1 Static Gesture Recognition

One of the commonly used methods for static gesture recognition is *Support Vector Machine* (SVM), an algorithm used for both regression and classification tasks. But overall, it is widely used in classifications. SVM's goal is to find a *hyperplane* in N-dimension space, N being the number of features, that distinctively classifies data points [37]. *Hyperplanes* are decision boundaries between data points and optimal *hyperplane* is the one with maximal separation, *margin*, between classes [38].

Chen and Tseng [39] presented an SVM solution for multi-angle hand gesture recognition for rock paper scissors using images from a web camera. The training dataset consisted of 420 images and a testing set of 120 images. Datasets were collected from 5 different people for the right hand only and achieving 95%. The classifier still managed to recognize left-hand gestures with 90% accuracy.

Domino et al. [40] utilized SVM with Microsoft Kinect sensors. Extracting hand features, fingertips, and center of the hand, from the depth map and feeding the data into SVM. Achieving 99.5% recognition rate on the dataset provided by Ren et Al. [41]. The dataset consists of 10 different gestures performed by ten different people repeatedly each ten times, a total of 1000 different depth maps.

Mapari and Kharat [42] on the other hand, proposed a method to recognize American Sign Language (ASL) with an *Feed-forward network* using *Multilayer Perceptron (MLP)*, extracting data from LMC and computing 48

features (18 positional values, 15 distance values, and 15 angle values) for 4672 collected signs (146 users for 32 signs). The average classification accuracy is 90%.

2.3.2 Dynamic Gesture Recognition

Katia et al. [43] proposed a method classifying dynamic gestures acquired through LMC with a CNN. Adopting a modified version of ResNet-50 architecture, a 50 layers deep CNN, removing the last fully connected layer and adding a new layer with as many neurons as the considered collection of gesture classes. The acquired gesture information is converted into hand joints color images. The variation of hand joint positions during the gesture is projected on a plane, and temporal information is represented with the color intensity of the projected points. The trained model achieved 91% classification accuracy on the LMDHG dataset [44].

Yang L., Chen J., and Zhu W. [45] used two-layer Bidirectional RNN in combination with an LMC to classify dynamic hand gestures represented by sets of feature vectors (fingertip distance, angle, height, the angle of adjacent fingertips and the coordinates of the palm). The proposed method has been tested on the American Sign Language (ASL) datasets with 360 samples and the Handicraft-Gesture dataset with 480 samples, achieving 90% and 92% accuracy [45].

Ameur et al. [46] presented a solution using an SVM classifier used with LMC acquired data, (X, Y, Z) coordinates of fingertips and palm center. The experimental results show an accuracy of 81% on a dataset containing 11 actions, performed by ten different subjects, having in total 550 samples.

2.3.3 Proposed LSTM solution

Many of the proposed methods focus either on static gesture recognition or dynamic gesture recognition, but very few of them are actually utilized for both types at the same time.

Avola D., Bernardi M. et al. proposed a method in [30] utilizing LSTM, specifically Deep LSTM (DLSTM), and LMC to recognize sign language and semaphoric hand gestures. It uses a hand skeleton extracted by an LMC and considers angles formed by a specific subset of hand joints. The presented method reached 96% accuracy in its predictions.

Consider each hand gesture to be represtend as set $X = \{x_0, x1, ..., x_{T-1}\}$ of feature vectors, in predetermined interval Θ size T, T being the number of time instances, in which features are extraxted by LMC. DLSTM is applied to obtain series of output probability vectors $Y = \{y_0, y1, ..., y_{T-1}\}$. At last the gesture classification is performed by a *softmax* layer using $n = |C|$, where C being the set of considered hand gestures [30].

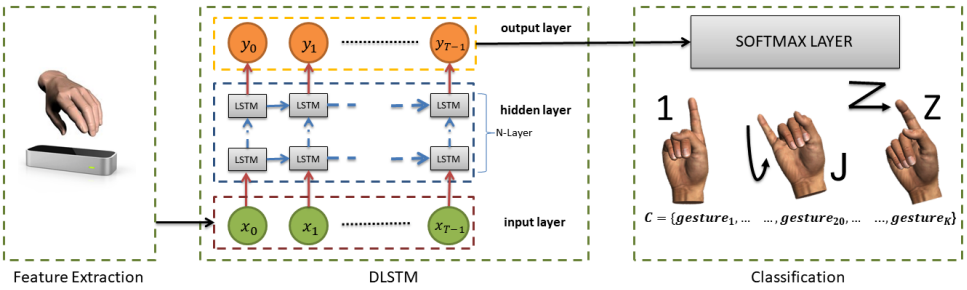


Figure 2.5: Logical structure of the proposed method [30]

2.3.3.1 Feature Extraction

A hand gesture can be considered to be composed of different poses, where particular angles characterize each pose. Each feature vector $x_t \in X$ consists mostly of internal angles, finger segments, palm position, and fingertip positions.

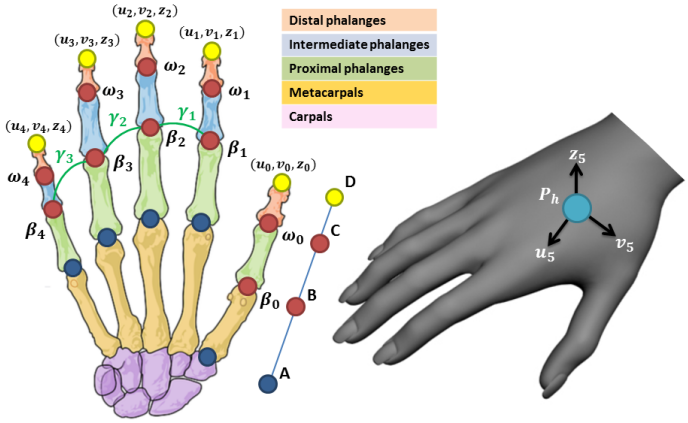


Figure 2.6: Internal angles of hand joints [30]

As seen in Figure 2.6, each finger can be represented as set of segments:

- $\overline{AB}$, proximal phalax, or metacarpal in case of thumb
- $\overline{BC}$, intermediate phalanx, or proximal phalanx in case of thumb
- $\overline{CD}$, distal phalanx

These set of segments are then used to calculate internal angles of considered finger:

- internal angles $\omega_1, \omega_2, \omega_3, \omega_4$ between distal phalanges and intermediate phalanges. Internal angle ω_0 of the thumb is calculated between distal phalanx and proximal phalanx.

$$\omega_{j\in\{0,...,4\}} = \frac{\overline{BC} \cdot \overline{CD}}{|\overline{BC}| \cdot |\overline{CD}|} \tag{2.1}$$

- internal angles $\beta_1, \beta_2, \beta_3, \beta_4$ between intermediate phalanges and proximal phalanges. Internal angle β_0 of the thumb is calculated between proximal phalanx and metacarpal.

$$\beta_{j\in\{0,...,4\}} = \frac{\overline{AB} \cdot \overline{BC}}{|\overline{AB}| \cdot |\overline{BC}|} \tag{2.2}$$

- intra-finger angles $\gamma_1, \gamma_2, \gamma_3$ are angles between two neighboring fingers, where considered fingers are: the pointer finger between middle finger, the middle finger and the ring finger, and the ring finger with a pinky finger. The infra-finger angles are used to handle special static gestures, for example, an open palm and a pop culture "Spock" greeting.

3D displacements of palm and fingertip positions serve to help classify dynamic hand gestures, where the movement is performed in 3D space.

- palm central point coordinates $P_h = (u_5, v_5, z_5)$ help to track the hand transition in the 3D space.
- finger tip positions u_l, v_l, z_l, $l \in 0, ..., 4$ help to track the hand rotation in 3D space.

All above features form the input vector x_t passed to DLSTM at time t.

$$x_t = \{\omega_0, ..., \omega_4, \beta_0, ..., \beta_4, u_0, v_0, z_0, ..., u_5, v_5, z_5, \gamma_1, \gamma_2, \gamma_3\} \tag{2.3}$$

2.3.3.2 Optimal Number of Stacked LSTMs

Several tests were performed to find the optimal number of stacked LSTMs. The results showed that having 4 LSTM layers proved to achieve the best accuracy by using 800 *epochs*, the number of times the learning algorithm goes through the complete training dataset. Although it was possible to get the same results with 5 or 6 stacked LSTM layers, only due to using 1600 and 1800 epochs, thus increasing the training time [30].

The *learning rate* was set to 0.0001 after large empirical tests. The learning rate determines how much the newly acquired information about the weights will influence their updating. If the learning rate is too low, it will require more time to converge towards the local minimum, while if the rate is too large, it may overstep the local minimum [30].

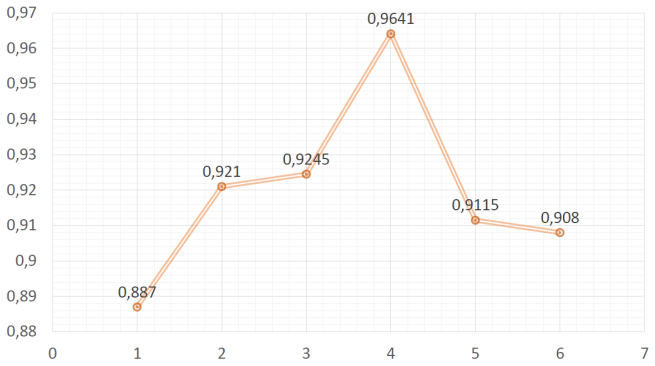


Figure 2.7: Model accuracy by using 800 epochs [30]

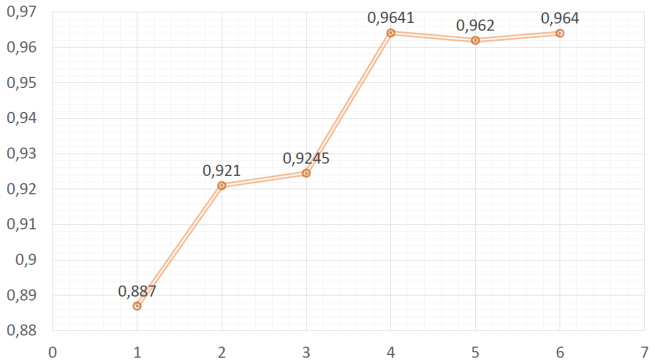


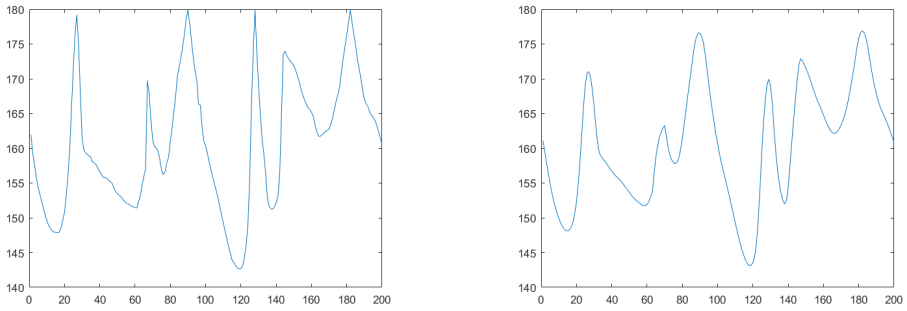
Figure 2.8: Model accuracy by using 1600 epochs for 5 LSTM layers and 1800 epochs for 6 LSTM layers [30]

2.3.3.3 Sampling Process

One gesture can be performed differently by each person, and all collected frame sequences must be composed of the same number of T samples. The proposed solution would collect data only in most significant T time instances, $t \in \Theta$ is considered significant if the joint angle and the central palm point coordinate P_h differs substantially between t and $t+1$.

To explain more specifically, let $f_{\omega_i}(t)$, $f_{\beta_i}(t)$, $f_{\gamma_j}(t)$ be functions representing values of ω_i, β_i, γ_j angles at time t, where $0 \leq i \leq 4$ and $1 \leq j \leq 3$. Coordinates of P_h may be ϕ and coordinates at time t may be represented as $f_\phi(t)$. Then the Savitzky-Golay filter [47] is applied on each of the named functions, $f_g(t)$, $g \in G = \{\omega_i, \beta_i, \gamma_j, \phi\}$. Svaitzky-Golay is a digital filter used to smooth a set of digital data in order to increase the signal-to-noise ratio without distorting the signal itself. Local extremes of each $f_g(t)$ are to be indentified as significant time variations and all time instances t, associated with at least one of these local maximum and minimum of feature g, form a

new set Θ^*, representing candidates of possible important time instances to be sampled.



(a) Feature ω_1

(b) Feature ω_1 after Savitzky-Golay filtering

Figure 2.9: Sampling example of feature ω_1

Depending on the cardinality of the newly acquired set Θ^*, the following cases must be considered:

- $|\Theta^*| < T$, the remaining samples ($|\Theta^*| - T$) are picked randomly from the original set Θ
- $|\Theta^*| > T$, only some of significant time instances for each g feature are picked to be sampled. Let $\Theta_g \subseteq \Theta^*$ be a set of significant time instances for feature g. The number of instances T_g to be sampled is chosen according to the ratio $|\Theta_g| : |\Theta^*| = T_g : T$, where the sum $\sum_{g \in G} T_g = T$ must be preserved [30].

Chapter 3

Implementation

As briefly mentioned in the Introduction chapter, our goal is to utilize Leap Motion controllers combined with the pre-trained ANN model.

We picked Python to be our primary language for training the ANN model, with the proposed DLSTM architecture by Avola D., Bernardi M. et al. [30], along with the web-based interactive development environment Jupyter Notebook. One of the main reasons to pick Python was its wide range of libraries and scientific packages supporting machine learning tasks. Most importantly, Keras, a high-level deep learning API integrated with TensorFlow, enabling the user to create and train model structures in very few steps.

3.1 Dataset Description

Training and testing have been performed on a combination of two gathered datasets, ASL Dataset [30], and SHREC 2017 dataset created in conjunction with [48].

3.1.1 SHREC 2017 Dataset

The SHREC dataset contains sequences of 14 dynamic hand gestures (grab, tap, expand, pinch, rotation clockwise, rotation counterclockwise, swipe right, swipe left, swipe up, swipe down, swipe X, swipe +, swipe V, shake). Each gesture was performed between 1 and 10 times by 28 participants in two ways, using one finger and the whole hand. All participants were right-handed. The length of sample gestures varies between 20 to 170 frames, making some samples too short. We solved this by using the padding technique to an acceptable value of $T = 100$ and discarding samples where more than 50 frames have to be padded [48].

3.1.2 ASL Dataset

ASL Dataset has been created by Avola D., Bernardi M. et al. [30] as the result of lack of public datasets holding necessary information about hand joints. The dataset consists of 30 hand gestures, 18 static gestures (1, 2-V, 3, 4, 5, 6-W, 7, 8, 9, A, B, C, D, H, I, L, X, and Y), and 12 dynamic gestures (bathroom, blue, finish, green, hungry, milk, past, pig, store, and where). Gestures were collected from 20 different people. 13 were used to form the training set, while the remaining 7 formed a test set. Each person performed 30 hand gestures twice, once for each hand, and each gesture is composed of fixed 200 frames as oppose to frame varying SHREC dataset [30]. Small modifications had to be made since we wanted to utilize both datasets at the same time. We stripped ASL Dataset of its dynamic gestures and split frames of static gesture sample in half, acquiring 2 samples. Static gestures are, in theory, one frame stretched out through time. Therefore such modification should not have a negative impact on our trained model.

Bibliography

[1] Chen, Y.-Y.; Lin, Y.-H.; et al. Design and Implementation of Cloud Analytics-Assisted Smart Power Meters Considering Advanced Artificial Intelligence as Edge Analytics in Demand-Side Management for Smart Homes. *Sensors*, 05 2019, doi:10.3390/s19092047.

[2] Bengio, Y.; Goodfellow, I.; et al. *Deep learning*, volume 1. Citeseer, 2017, ISBN 0262035618, 166–485 pp.

[3] McCulloch, W. S.; Pitts, W. A logical calculus of the ideas immanent in nervous activity. *The bulletin of mathematical biophysics*, volume 5, no. 4, 1943: pp. 115–133, ISSN 0007-4985.

[4] Krishtopa. What Are Neural Networks, Why They Are So Popular And What Problems Can Solve. 2016. Available from: `https://steemit.com/academia/@krishtopa/what-are-neural-networks-why-they-are-so-popular-and-what-problems-can-solve`

[5] Rosenblatt, F. The Perceptron: A Probabilistic ModelFor Information Storage And Organization In The Brain. *Psychological Re-view*, 1958: p. 2047, doi:0.1037/h0042519.

[6] Kozák, M. Static malware detection using recurrent neural networks. [cit. 2020-12-28]. Available from: `https://dspace.cvut.cz/bitstream/handle/10467/88342/F8-BP-2020-Kozak-Matous-thesis.pdf?sequence=-1&isAllowed=y`

[7] Nielsen, M. A. *Neural Networks and Deep Learning*. Determination Press, 2015.

[8] Rojas, R. *Neural networks: a systematic introduction*. Springer Science & Business Media, 2013, ISBN 9783642610684, 37–99 pp.

[9] Leskovec, J.; Rajaraman, A.; et al. *Mining of massive data sets*. Cambridge university press, 2020, ISBN 9781108476348, 523–569 pp.

[10] Maladkar, K. 6 Types of Artificial Neural Networks Currently Being Used in ML. Available from: https://analyticsindiamag.com/6-types-of-artificial-neural-networks-currently-being-used-in-todays-technology/

[11] Lipton, Z. C.; Berkowitz, J.; et al. A critical review of recurrent neural networks for sequence learning. *arXiv preprint arXiv:1506.00019*, 2015: pp. 5–25, ISSN 2331-8422. Available from: https://arxiv.org/pdf/1506.00019.pdf

[12] Feedforward Neural Networks. Available from: https://brilliant.org/wiki/feedforward-neural-networks/

[13] Team, T. A. Main Types of Neural Networks and its Applications-Tutorial. Aug 2020. Available from: https://medium.com/towards-artificial-intelligence/main-types-of-neural-networks-and-its-applications-tutorial-734480d7ec8e

[14] Goodfellow, I.; Bengio, Y.; et al. *Deep Learning*. MIT Press, 2016, http://www.deeplearningbook.org.

[15] Backpropagation. Available from: https://brilliant.org/wiki/backpropagation/

[16] How Do Convolutional Layers Work in Deep Learning Neural Networks? April 2020. Available from: https://machinelearningmastery.com/convolutional-layers-for-deep-learning-neural-networks/

[17] Convolutional Neural Network. Available from: https://www.mathworks.com/solutions/deep-learning/convolutional-neural-network.html

[18] A Comprehensive Guide to Convolutional Neural Networks-the ELI5 way. Dec 2018. Available from: https://towardsdatascience.com/a-comprehensive-guide-to-convolutional-neural-networks-the-eli5-way-3bd2b1164a53

[19] Recurrent Neural Networks. Jun 2019. Available from: https://towardsdatascience.com/recurrent-neural-networks-d4642c9bc7ce

[20] What are Recurrent Neural Networks? Available from: https://www.ibm.com/cloud/learn/recurrent-neural-networks

[21] Understanding Recurrent Neural Networks in 6 Minutes. Sep 2019. Available from: https://medium.com/x8-the-ai-community/understanding-recurrent-neural-networks-in-6-minutes-967ab51b94fe

[22] Schuster, M.; Paliwal, K. Bidirectional recurrent neural networks. *Signal Processing, IEEE Transactions on*, volume 45, 12 1997: pp. 2673 – 2681, doi:10.1109/78.650093.

[23] Olah, C. Understanding LSTM Networks [online]. [cit. 2020-12-28]. Available from: https://colah.github.io/posts/2015-08-Understanding-LSTMs/

[24] Hochreiter, S.; Schmidhuber, J. Long Short-term Memory. *Neural computation*, volume 9, 12 1997: pp. 1735–80, doi:10.1162/neco.1997.9.8.1735.

[25] Phi, M. Illustrated Guide to LSTM's and GRU's: A step by step explanation [online]. [cit. 2020-12-28]. Available from: https://towardsdatascience.com/illustrated-guide-to-lstms-and-gru-s-a-step-by-step-explanation-44e9eb85bf21

[26] Holzner, A. LSTM cells in PyTorch [online]. Oct 2017, [cit. 2020-12-28]. Available from: https://medium.com/@andre.holzner/lstm-cells-in-pytorch-fab924a78b1c

[27] Graves, A.; Mohamed, A.; et al. Speech Recognition with Deep Recurrent Neural Networks. *CoRR*, volume abs/1303.5778, 2013, 1303.5778. Available from: http://arxiv.org/abs/1303.5778

[28] Brownlee, J. Stacked Long Short-Term Memory Networks [online]. Aug 2019, [cit. 2020-12-28]. Available from: https://machinelearningmastery.com/stacked-long-short-term-memory-networks/

[29] Vafaei, F.; Slator, B.; et al. TAXONOMY OF GESTURES IN HUMAN COMPUTER INTERACTION. 12 2013.

[30] Avola, D.; Bernardi, M.; et al. Exploiting Recurrent Neural Networks and Leap Motion Controller for the Recognition of Sign Language and Semaphoric Hand Gestures. *IEEE Transactions on Multimedia*, volume 21, no. 1, Jan 2019: p. 234–245, ISSN 1941-0077, doi:10.1109/tmm.2018.2856094. Available from: http://dx.doi.org/10.1109/TMM.2018.2856094

[31] Microsoft. Azure Kinect [online]. [cit. 2020-12-25]. Available from: https://img-prod-cms-rt-microsoft-com.akamaized.net/cms/api/am/imageFileData/RWqOsq?ver=2e37

[32] Microsoft. Azure Kinect DK documentation [online]. [cit. 2020-12-25]. Available from: https://docs.microsoft.com/en-us/azure/Kinect-dk/

[33] Weichert, F.; Bachmann, D.; et al. Analysis of the Accuracy and Robustness of the Leap Motion Controller. *Sensors (Basel, Switzerland)*, volume 13, 05 2013: pp. 6380–6393, doi:10.3390/s130506380.

[34] Tomas Novacek, M. J., Christian Marty. Project MultiLeap: Fusing data from multiple Leap Motion sensors. *ACM Trans. Graph*, volume 37, 08 2020: pp. 1–5.

[35] Ultraleap. Tracking: Ultraleap Stereo IR 170 Evaluation Kit [online]. [cit. 2020-12-26]. Available from: https://www.ultraleap.com/product/stereo-ir-170/

[36] Prototyping, S. Ultraleap Stereo IR 170 [online]. [cit. 2020-12-26]. Available from: https://www.smart-prototyping.com/Ultraleap-Stereo-IR-170

[37] Microsoft. Support Vector Machine - Introduction to Machine Learning Algorithms [online]. 7 2018, [cit. 2020-12-25]. Available from: https://towardsdatascience.com/support-vector-machine-introduction-to-machine-learning-algorithms-934a444fca47

[38] Alexandre Savaris, A. v. W. A. Comparative evaluation of static gesture recognition techniques based on nearest neighbor, neural networks and support vector machines. *J Braz Comput Soc*, volume 16, 2010: p. 147–162, doi:10.1007/s13173-010-0009-z.

[39] Chen, Y.; Tseng, K. Developing a Multiple-angle Hand Gesture Recognition System for Human Machine Interactions. 2007: pp. 489–492, doi:10.1109/IECON.2007.4460049.

[40] Dominio, F.; Donadeo, M.; et al. Hand Gesture Recognition with Depth Data. 2013: p. 9–16, doi:10.1145/2510650.2510651. Available from: https://doi.org/10.1145/2510650.2510651

[41] Ren, Z.; Yuan, J.; et al. Robust hand gesture recognition based on finger-earth mover's distance with a commodity depth camera. *MM'11 - Proceedings of the 2011 ACM Multimedia Conference and Co-Located Workshops*, 11 2011: pp. 1093–1096, doi:10.1145/2072298.2071946.

[42] Mapari, R. B.; Kharat, G. American Static Signs Recognition Using Leap Motion Sensor. 2016, doi:10.1145/2905055.2905125. Available from: https://doi.org/10.1145/2905055.2905125

[43] Lupinetti, K.; Ranieri, A.; et al. 3D Dynamic Hand Gestures Recognition Using the Leap Motion Sensor and Convolutional Neural Networks. 2020: pp. 420–439.

[44] Boulahia, S. Y.; Anquetil, E.; et al. Dynamic hand gesture recognition based on 3D pattern assembled trajectories. 2017: pp. 1–6, doi:10.1109/IPTA.2017.8310146.

[45] Yang, L.; Chen, J.; et al. Dynamic Hand Gesture Recognition Based on a Leap Motion Controller and Two-Layer Bidirectional Recurrent Neural Network. *Sensors*, volume 20, 04 2020: p. 2106, doi:10.3390/s20072106.

[46] Ameur, S.; Khalifa, A. B.; et al. A comprehensive leap motion database for hand gesture recognition. 2016: pp. 514–519, doi:10.1109/SETIT.2016.7939924.

[47] Savitzky, A.; Golay, M. J. E. Smoothing and Differentiation of Data by Simplified Least Squares Procedures. *Anal. Chem.*, volume 36, no. 8, July 1964: pp. 1627–1639, doi:10.1021/ac60214a047. Available from: `http://dx.doi.org/10.1021/ac60214a047`

[48] De Smedt, Q.; Wannous, H.; et al. SHREC'17 Track: 3D Hand Gesture Recognition Using a Depth and Skeletal Dataset. Apr. 2017: pp. 1–6, doi:10.2312/3dor.20171049. Available from: `https://hal.archives-ouvertes.fr/hal-01563505`

Appendix A

Acronyms

GUI Graphical user interface

XML Extensible markup language

Appendix B

Contents of enclosed CD

- `readme.txt` ..... the file with CD contents description
- `exe` ..... the directory with executables
- `src` ..... the directory of source codes
  - `wbdcm` ..... implementation sources
  - `thesis` ..... the directory of LaTeX source codes of the thesis
- `text` ..... the thesis text directory
  - `thesis.pdf` ..... the thesis text in PDF format
  - `thesis.ps` ..... the thesis text in PS format