Deep learning based action recognition with application to dogs

WENYI ZHAO

Exchange in Computer Science
Date: June 13, 2017
Supervisor: Jonas Beskow
Examiner: Olov Engwall
School of Computer Science and Communication
Abstract

The last decade has seen a booming of Human Action Recognition technologies and algorithms. Accurate recognition of human action would impact plenty of areas including medical, security and entertainment. However, the need for pets health increased in the market in the recent years and the industrial application is rare in this area.

In this thesis, we adopt a deep learning based method for dog action recognition which we hope can substitute for well-established analysis techniques that rely on hand-crafted feature extraction and classification methods. In particular, Residual neural network and Long Short Time Memory (LSTM) Recurrent Neural Network based method is tested for recognize the sequential data collected from the accelerometer on the neck of the dog. We achieved the best F1-Score 94.7% on three activities for dog by stacking four recurrent layers that have five residual bi-directional layers inside each of the layer. Our model proved the possibility to use limited data (3-axial acceleration data) to build a complex model, the results are good in practical.

The future work will concentrate on collecting more data and build sub-model for activities that are difficult to distinct.
# Contents

<table>
<thead>
<tr>
<th>Contents</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>1 Introduction</td>
<td>1</td>
</tr>
<tr>
<td>2 Background</td>
<td>3</td>
</tr>
<tr>
<td>2.1 Activity Recognition System</td>
<td>3</td>
</tr>
<tr>
<td>2.1.1 Data Collection</td>
<td>3</td>
</tr>
<tr>
<td>2.1.2 Accelerometer Based Activity Recognition</td>
<td>4</td>
</tr>
<tr>
<td>2.2 Deep Learning</td>
<td>5</td>
</tr>
<tr>
<td>2.2.1 Feed Forward Network</td>
<td>6</td>
</tr>
<tr>
<td>2.2.2 Recurrent Neural Network</td>
<td>11</td>
</tr>
<tr>
<td>2.2.3 Bidirectional Recurrent Neural Network</td>
<td>15</td>
</tr>
<tr>
<td>2.2.4 LSTM Network</td>
<td>17</td>
</tr>
<tr>
<td>2.2.5 Residual Network</td>
<td>22</td>
</tr>
<tr>
<td>2.2.6 Over-fitting Problem</td>
<td>24</td>
</tr>
<tr>
<td>3 Method</td>
<td>26</td>
</tr>
<tr>
<td>3.1 Dataset</td>
<td>26</td>
</tr>
<tr>
<td>3.2 Tensorflow Deep Learning Framework</td>
<td>29</td>
</tr>
<tr>
<td>3.3 LSTM Structure</td>
<td>30</td>
</tr>
<tr>
<td>3.3.1 Stacked Unidirectional LSTM</td>
<td>30</td>
</tr>
<tr>
<td>3.3.2 Stacked Residual Bidirectional LSTM</td>
<td>31</td>
</tr>
<tr>
<td>3.3.3 Configuration</td>
<td>31</td>
</tr>
</tbody>
</table>
4 Experiment

4.1 Environment .................................................. 34

4.2 Results on the UCI human action dataset ......................... 34
  4.2.1 Evaluating .................................................... 34
  4.2.2 Parameter Tuning ............................................. 35
  4.2.3 Result on Stacked Residual Model .......................... 36
  4.2.4 Discussion ................................................... 37

4.3 Results on dog dataset ........................................... 38
  4.3.1 Preprocessing of Data ....................................... 38
  4.3.2 Best Result on Stacked Residual Network .................... 38
  4.3.3 Discussion ................................................... 38

5 Conclusion & Future Work ........................................ 40

5.1 Conclusion on the performance of model ........................ 40

5.2 Future Work ..................................................... 40

Bibliography ......................................................... 42

A Unnecessary Appended Material ................................. 50
Chapter 1

Introduction

Nowadays, more and more people start keeping dogs in their home, there are approximately 65 million dogs in U.S. households back in 2006[1], and the number is continuing increasing over the last decade. The large ownership of dogs stimulates the need of pets health caring service. Nevertheless, people and dogs cannot communicate with each other by human languages, so there is few way available that people can know the health condition of their pets. Without knowing the health condition of pets, diseases of animals may hurt both people and pets. However, if we can the know the health condition of dogs by understanding the of dog activities, pets’ owner can notice the abnormal behavior of their pets and therefore can take action to their pets’ diseases earlier.

In the last decade, deep learning, as a class of machine learning algorithm that take advantages of multi-layer neural network have achieved great success in various areas including computer vision, speech recognition, natural language processing and more. Recurrent neural network(RNN) with long short-time memory(LSTM) implementation is a specific deep neural network model that have the gated form in each units, which have show great advantages in the application of analysis the sequential data like videos, speech audios, text sentences and so on.

Dog activity recognition is a new industrial area therefore not so many academic work have been done exactly for animals activity recognition. However, there have been some papers that focus on the human activity recognition with deep learning methods so far. Hidden Markov Modeling(HMM)[2, 3, 4], Con-
volution Neural Network\[5, 6\] has both shown its advantages over the previous method on the accelerometer human activity data.

Tracy Trackers is a start-up company focus on the dog activity recognition and health care service. They aim to develop a dog activity recognition system based on mobile application and intelligent collar worn on dogs. Their product can collect and analyze the sequential acceleration and gyroscope data collected from the dog, and tell which activity are pets doing. Although there are other competence company that also use the similar product with pets activity, almost all the similar product on the market focus only on monitoring(recording) the pets activity, instead of the analyzing and classifying of data, therefore their products have a great advantages that take the state-of-art method to analysis the dog activity and tell more about the dog health status to the dog owner.

The main purpose of this thesis is to evaluate the performance of using a Residual LSTM model in activity recognition system by experiment on the confusion matrix and testing accuracy. Given the similarity between the form of human activity data and dog activity data, we also examine the model firstly with the UCI human activity dataset\[7\]. The implementation of the code is currently on Python with TensorFlow Framework, and they are working on the on-line model on mobile devices.
Chapter 2

Background

2.1 Activity Recognition System

2.1.1 Data Collection

Accelerometer data play an important role in an activity recognition system. We can either extract the pattern by signal processing method or directly to feed the Neural Network for learning and inferencing of different activity. Typically, there are two class of approaches of data collection for activity recognition system.

The first type of method is based on well designed sensor system including the environmental sensors like locomotion camera and connected accelerometer sensors on every important joint point of the object. This methods can provide a comprehensive data for recognizing the activity, but it also need the object be trained to make sure the sensor all works well, no to mention that the sensor system is usually difficult to maintain.

The other type of method is using simple wearable devices combined sensor(like sports bands or smart watches) that can record the acceleration of specific part of the object. Even though it can only provide a limited information comparing to a set of sensors, it is sufficient to differentiate different activity on human body in the previous studies as it is easy to maintain and the data amount is usually larger than previous method.

As is mentioned before, the environmental sensor systems are generally big-
ger in size and more expensive. And our experiment objects are animals that are quite sensitive to the environmental condition and different in size. Additionally, having them physically connected via wires are not easy. Therefore it’s not possible to build a lab for collecting the activity data and we decide to use the off-the-shelf wearable sensor to collect data. As wearable sensors are cheap and easy to use, the sensor is small and will not affect the normal movement of the dogs, which gives use the opportunity to collect data with the minimal invasive to the normal life to the animals.

2.1.2 Accelerometer Based Activity Recognition

There are some previous activity recognitions works that used the multiple accelerometers and other sensors as a recognition system to do the activity recognition, Lester[8] used accelerometers and microphone in a controlled environment to classify 8 activities by static and HMM method. A wavelet-based activity classification method using one or more accelerometers was proposed by Mannini and Sabitini[9], in which the dynamic motion component is separated from the gravity components, leading to a 98.4% of accuracy. Casale[10] use random forest and a wearable device for classifying acceleration data for human activity recognition, proving Minmax and RMS integration as a important feature, they obtain a 94% of accuracy.

Bao and Intille[11] use five biaxial accelerometers worn on the user’s right hip, dominant wrist, non-dominant upper arm, dominant ankle, and non-dominant thigh to monitor 20 types of activities, trained with 20 users using decision tables, instance-based learning, C4.5 and Naive Bayes classifiers. Their results indicate that the accelerometer placed on the thigh was the most powerful for distinguishing between activities. Nishkam et. al[12] use a single tri-axial accelerometer is worn near the pelvic region to recognize eight activities: Standing, Walking, Running, Stairs-Up, Stairs-Down, Sit-Ups, Vacuuming, Brushing Teeth. They have different data collection settings and achieved best accuracy for most of the data collection settings by Plurality Voting Classifier. In [13], smartphone’s accelerometer data is collected from 29 users and they achieved the 90% accuracy on six
activities by multilayer perceptron, except for the case Stairs-Up versus Stairs-Down that poses greater difficulty. They found that neural network classification can best detect Jogging and Stairs-Up.

2.2 Deep Learning

The first theory paper of Artificial Neural Networks was published in the 1980s or earlier[14, 15, 16], which at first was inspired by biology and was trying to mimic human brain mechanism by using multiple levels of nonlinear operations to find the patterns in complex data that are not easy to classify by the traditional pattern recognition methods. The nonlinear operation is crucial since it allows the network to model extremely complex structures. However, it is not under researchers’ attention due to the vanishing gradient problem caused by training many successive non-linear layers of neurons.

Recently, these problems become possible to handled due to the combination of multiple factors, including new structure of network, the increasingly used GPU, new activation functions like ReLUs[19], regularization techniques like dropout[20] and the growing amount of huge labeled database like ImageNet[21]. All these growing technologies not only improve the performance and effectiveness of training a complex neural network, but also stimulate the growth of big-scale neural networks, which is currently known as the Deep Neural Network or Deep Learning.

Starting with the blooming research in computer vision with convolutional neural network in 2012, deep learning has been further successfully applied to many areas including visual object recognition[22, 23, 24], Natural Language Processing[25, 26, 27], Speech recognition[28, 29] and many other domains such as drug discovery and genomics. Among these areas mentioned above, speech recognition are the most similar one to our task as the raw signal form of speech is also a one dimensional data, thus lots of idea in Speech Recognition can be referred and further adapted to our task.
2.2.1 Feed Forward Network

Many variation neural network have been appeared over last decades, one of the most important differences between them is whether the connection between layers form cycles, the neural network that are acyclic are always referred to the Feed-forward network (FNN), and those with cycles are usually called recursive or recurrent neural network which this paper will mainly focused on in section 2.2.2 and later.

There are well known examples of FNN including radial basis function networks[30], and Kohonen maps[31], and the most widely used form of FNN that we are going to introduce is known as Multilayer Perceptron[16, 32].

A Neural Network consists of many simple units, called neurons, connected to each other in layers. Basic structure of a Multilayer Perceptron is as shown in Figure 2.1 below,

![Multilayer Perceptron Diagram](image)

Figure 2.1: A multilayer perception example

2.2.1.1 Forward Pass

Each units in a multilayer are arranged in layers, with the connection between the adjacent layers, the input data can be feed forward layer by layer to the output. The input layer are connected with the raw data, as shown on the left of the figure, then there are usually several levels of hidden layers that propagate the data to the output layer. This process is called forward pass. The number of units
in each layer are depended on different factors, input data determine the units in
the input layer, and the task of the network decide how many units in the output
layer, for example a binary classification typically need one unit in the output
layer.

Each unit in the first hidden layer calculates a weighted sum of the input
units. This sum is denoted by \( a_h \) and then the activation function \( \theta_h \) is applied,
yielding activation \( b_h \) of this unit. And every units in the next layer calculate the
weighted sum from the previous layers similarly, finally reach the output layer.
Denoting the weight from the unit \( i \) to unit \( j \) as \( w_{ij} \), we have the equation for cal-
culating the activation of each unit

\[
b_h = \theta_h(a_h) = \theta_h\left(\sum_{i=1}^{I} w_{ih}x_i\right)
\]  

(2.1)

The most common choice of the activation function are logistic sigmoid and
hyperbolic tangent function, in fact, these two function can be transfer to each
other with linear operation, the only different is the output range of these two
functions. Although linear function can also be the activation function for the
neuron, but since any combination of linear operation is still linear function, it
will make multiple hidden layer exactly the same as only one hidden layer. Nev-
ertheless, with the non-linear activation function theoretically successive layers
can represent any input data.

Normally for a binary classification task the standard configuration of the out-
put layer is a single unit with a logistic sigmoid activation function. With this
setting the output of the output layer’s activation function can be viewed as the
possibility of the current data belongs to a given class \( A \), and one minus this
value can be viewed as the possibility belongs to the other class \( B \).

\[
p(c_0|x) = \alpha(a_{output}) = y \\
p(c_1|x) = 1 - y
\]  

(2.2)

(2.3)

Writing it concisely we can using \( p(c|x) = (1 - c)y + c(1 - y) \) to represent the
likelihood of \( x \) belongs to class \( c \).
(a) Hyperbolic tangent function.  

(b) Logistic sigmoid 

Figure 2.2: Some common activation function

When the problem is a multiple class (ex. N > 2 class) task, then it is common to set the number of neurons in the output layer to be N, and using the softmax function[33] to normalize the output activation with to obtain the class probabilities.

\[ p(c_n|x) = y_n = \frac{e^{a_n}}{\sum_{i=1}^{N} e^{a_i}} \]  \hspace{1cm} (2.4)

Where \( a_i \) is the weighted input to the \( i \) output unit. By setting the labeled class to one-hot vector we can get a convenient form of likelihood.

\[ p(c|x) = \prod_{n=1}^{N} y_n^{c_n} \]  \hspace{1cm} (2.5)

### 2.2.1.2 Loss Function

In order to evaluate the prediction of the model to further improve it during the training, we need a function to calculate the difference or error between the predict value and ground-truth value labeled for training. As normally we don’t have prior knowledge of the distribution of the data, we tend to use the loss function derived by using the maximum likelihood(MLE), where we want to
minimize the loss function to maximize the likelihood of $x$ being correctly classified by the model.

$$L(x, c) = -\log p(c|x) \quad (2.6)$$

Substituting the previous likelihood we can have the loss function for our neural network

$$L(x, c) = -\sum_{n=1}^{N} c_n \log y_n \quad (2.7)$$

### 2.2.1.3 Backward Propagation

Learning in a Neural Networks context means finding appropriate weights in order to exhibit desired behavior, such as classification of images etc. An efficient way to learn these weights for supervised learning is called back-propagation[16], which also refer to the backward pass in a neural network. This method was first introduced in 1970s and later in 1980 Rumelhart and some other people bring it to the research attention. Rumelhart’s paper indicates that back-propagation makes several neural network worked much better compare to other learning algorithm. After that the backward pass becomes a standard learning algorithm in neural network and leads the big success of the deep learning later.

The basic idea of back-propagation in repeatedly using the chain rule for partial derivative to calculate the gradient of the loss function of the multiple layer perceptron(MLP) from the output layer back to the input layer. The process of learning the model is called stochastic gradient descent(or steepest descent). By calculate the gradient of the loss function with respect to the weight of each neurons in the network we can we can update the weight towards the opposite direction of the gradient and finally reduce the loss function to a optimized minimum value.

We use $\delta_j$ to represent the partial derivative of Loss function $L(x, c)$ with respect to the input of any units in the network. Since the loss function only depends on each hidden units through the its influence on the output units, there-
fore the derivative with respect to the last hidden layer can be calculated by

\[\delta_h = \frac{\partial L(x, c)}{\partial b_h} = \sum_{n=1}^{N} \frac{\partial L}{\partial a_n} \frac{\partial a_n}{\partial b_h} \]  

(2.8)

we substituting Equation 2.1 to the 2.8 yields

\[\delta_h = \theta_h'(a_h) \sum_{n=1}^{N} \delta_n w_{hn} \]  

(2.9)

And then we can recursively calculate the partial derivative towards the previous hidden layers by

\[\delta_h = \theta_h'(a_h) \sum_{h'=1}^{H_2} \delta_{h'} w_{hh'} \]  

(2.10)

here we use 2.1 as an example so $H_2$ is the last hidden layer. After calculate the derivative towards all sum input $a_i$ of the hidden units, we can finally calculate the derivative with respect to each of the network weights $w_{ij}$,

\[\frac{\partial L(x, c)}{\partial w_{ij}} = \frac{\partial L}{\partial a_j} \frac{\partial a_j}{\partial w_{ij}} = \delta_j b_i \]  

(2.11)

The amount of adjustment $w_j$ by each back-propagation step is controlled by the learning rate $\alpha$, if the learning rate was set correctly, loss function will coverage correctly under a given stop criteria (ex. the loss is not getting smaller in a couple of rounds). However, this method is not vary stable and is easy to get stuck in local minima problem where the loss is not ideal but the algorithm still stop under the criteria, the way to reduce this problem is adding a the previous rounds weight changing $w_{j-1}$ with a momentum parameter $m$, this will increase the coverage speed and therefore may help to avoid local minima[34].

\[\Delta w_{ij} = m \Delta w_{ij-1} - \alpha \frac{\partial L}{\partial w_j} \]  

(2.12)

In fact the loss function is used to calculate the amount of information gain needed to go from the softmax output to the labels’ one-hot “distribution”. The closer the network is into having 100 "belief" in the correct class label, the lower the loss will be, and vice versa. And once the loss can’t be lower the maximum
output of the softmax output layer can have the biggest chance to correctly classify the data.

Therefore the learning process of a feed forward network is first initialize the weight vector on every unit in every layer with a random value (ex. sample from a normal distribution $\mathcal{N}(0, 1)$), and then forward passing every input data vector to the network to get the output of the network, after getting the output likelihood we do the propagation the error to backward through the network to modify the weight on every units from output layer to input. This process is continuing until the given iteration time is meet or the error is under the given threshold.

2.2.2 Recurrent Neural Network

Different from the ordinary neural network which only related the current given data and all inputs and outputs are independent of one another, Recurrent Neural Networks (RNN) are a family of neural networks that make use of sequential data input. Like the Convolutional Network that are good at dealing with the high dimensional matrix data like an image, the recurrent neural network is specialized for for processing a sequence of values $x_1 \ldots x_T$. RNNs are general more computational powerful than Feed Forward Neural Networks, and given the cycles formed between the connection of neurons, the recurrent network can in principle map from the entire history input to the each output. If the number of the hidden units in a recurrent neural network is big enough, it can theoretically approximate any sequential mapping to arbitrary accuracy[35]. The key point is that the recurrent connections allow a ‘memory’ of previous inputs to persist in the network’s internal state, which can then be used to influence the network output.

In RNN, the data on each time step go through same neural network structure that share the same parameter between each other. The reason of sharing parameter in different parts of the model is that it allows us to extend and generalize the model across different length of data. If we have different parameter for each time step, we will not be able to generalize the model to the length of sequence
that are not present in the training set.

Another importance of parameter sharing is that when a specific information appear in multiple position in the sequence, for example if we want the model to find the position content ‘A’ from two different series “B-A-C-D” and “C-D-A-B”, the traditional multilayer perceptron (Feed forward network) would have to build different models on different positions for these two series of data, but the recurrent network sharing the same parameter therefore are requires less parameter to build the model hence are more efficient to train and apply.

![Figure 2.3: A folded RNN example](image)

Above in the figure 2.3 we can see a simple folded recurrent neural network, the hidden units in each time steps and the hidden units of different time steps also connected to form a fully connected layer. The output of current hidden layer not only depends on the current data input but also depends on the previous hidden layers results. For a sequence to sequence mapping problem, one can have each output $y_t$ to for a mapping sequence; and for sequential labeling question as we encounter here, we only need output the final output $y_T$ in the last time step of the input sequence; the output put can also be customized chosen according to the need of the task.

### 2.2.2.1 Forward Pass

Similar to the multilayer perceptron, learning a RNN model also include the forward and backward pass, the different is that previous and future data also in-
fluence the current process. Concerning a simple recurrent network structure like figure 2.3, a length T input sequence is fed to the network with I inputs units, H hidden units and N output units. We represent the current input i with \( x^t_i \), and let \( a^t_h \) and \( b^t_h \) be respectively the network input to hidden unit h at time t and activation of unit h at time t. Unlike the standard feed forward equation, the activations of the current hidden units is related to both the weighted sum of current input and also prior network’s hidden layer activation. Therefore we can have the function of calculating the activation.

\[
b^t_h = \theta_h(a^t_h) = \theta_h\left(\sum_{i=1}^{I} w_{ih} x^t_i + \sum_{h'=1}^{H} w_{h'x} x^t_i \right)
\]

(2.13)

In order to calculate the activation for every time step the value of the first time step’s activation need to be initialized before the data is received. Normally the \( b^0_0 \) can be set to zero for simplicity, but researchers have found in some cases, the stability and robustness of RNN can be improved by using non-zero initial activation[36].

The network inputs to the output layer units \( a^t_n \) is calculated at the same time as hidden activations as standard FNN.

\[
a^t_n = \sum_{h=1}^{H} w_{hk} b^t_n
\]

(2.14)

Since the data of our collected data are processed before feed into the network and the classification targets are presented at the end of the sequence, we can reuse the softmax activation function for the output layer, and the loss function can also be reused[37]. However, if we want to classify the temporal data which means on-line classify the data while it is being collected, we can not reuse the previous output layers’ setting since we don’t know the position of the target output.

2.2.2.2 Backward Propagation

In order to calculate the partial derivative of the loss function with respect to the sum input of each hidden units, we apply the chain rule as before, the only
different is that the loss function depends on the activation of the hidden layer $b_h$ not only through its influence on the output layer but also through the influence on the hidden layer on the next time step. Thus the $\delta^t_h$ is calculated with the following equation,

$$
\delta^t_h = \theta'_h(a^t_h)(\sum_{n=1}^{N} \delta^t_k w_{hk} + \sum_{h'=1}^{H} \delta^{t+1}_{h'} w_{hh'})
$$ (2.15)

Since each time step share the weights so we only need to calculate the derivative at time T and sum over each time step to get the derivative with respect to the weights, We set the $\delta^{T+1}_h$ to zero as the training set are all segmented sequence and no data is out-bounded.

$$
\frac{\partial L(x,c)}{\partial w_{ij}} = \sum_{t=1}^{T} \frac{\partial L}{\partial a^t_j} \frac{\partial a^t_j}{\partial w_{ij}} = \sum_{t=1}^{T} \delta^t_j b^t_i
$$ (2.16)

### 2.2.2.3 Unfolding

One way of understanding cycles in recurrent neural networks is by unfolding the loop. In figure 2.4, we can see every previous network pass the hidden value to its successor, and the whole network output with a single value $y$ at the end of the sequence, unlike the other sequence to sequence mapping, this network can be used to summarize a sequence and produce a fixed-sized representation used as the input for further processing. For example, in our activity recognition problem, the output $y$ can be viewed as the estimated class of activity and can be used to calculate the loss function with a given ground-truth.

### 2.2.2.4 Long-Term Dependence

This unfolding chain-like structure reveals why recurrent neural networks are so well suited for understanding sequences and lists. However, the biggest problem with standard RNNs is the back-propagation process of training a RNN is sensitive to the gradient value. The gradient will be either small or too big as it is multiplied with the weights of every time step, only the suitable gradient value can lead to the ideal coverage. If the weights are too small then the learning and
coverage will become very slow and even stop. Otherwise If the weights are too big the learning may diverges in the end. That will turn into the fact that RNN is very difficult to store information for long periods of time as more weights is included, which is called the long-term dependence problem. One of the most popular methods of solving this gradient problems is called Long Short-Term Memory (LSTM) architecture[38]. Which is one of the most popular type of RNN that has shown great success in the various areas like Speech Language Modeling[39], Sentiment Classification and Machine Translation problem. These success indicates the capability of using LSTM to solve the long-term dependency and has made LSTMs be a standard choice of solving this type of problem for a couple of years.

2.2.3 Bidirectional Recurrent Neural Network

In some cases, if we want to classify a series of input $x$ at the end of the sequence $t$, its better to delay the input for an small amount time $t_{delay}$ to include some future input data for better performance, but it’s not easy to determine the optimal delay time and the prediction result will drop if the delaying time is too big. In order to deal with this problem Bidirectional Recurrent Neural Network...
is proposed as an elegant solution[40]. BRNNs have previously given improved results in various domains, notably Speech Recognition (Combined with Long short-term memory)[29], Translation[41] and Handwritten Recognition[42].

The basic structure of an unfolded BRNN is shown in Figure 2.5, where we include three time steps. Each square represent the whole layer in its time step, the basic idea of the BRNN is to present the input sequence to two separate hidden layers, namely forward layer and backward layer, both of these two layer is connected a shared output layer. Note that any of these hidden layer can be used for prediction as the normal RNN. This two layer will give the network with a complete, symmetrical past and future context for every point in the input sequence, without displacing the inputs from the relevant targets.

The forward pass for the BRNN hidden layers is the same as for an unidirectional RNN, except that the input sequence is presented in opposite directions to the two hidden layers, and the output layer is not updated until both hidden layers have processed the entire input sequence. Similarly, the backward pass proceeds as for a standard RNN trained with BPTT, except that all the output layer $\delta$ terms are calculated first, then fed back to the two hidden layers in opposite directions.
2.2.4 LSTM Network

Normal LSTM Structure  As is shown in Figure 2.6. A LSTM network is formed exactly like a simple RNN, instead of using the regular nonlinear units it uses the memory cells in the hidden layer. In fact the memory cell is a well designed structure that have an internal recurrence in addition to the outer recurrence of the RNN, it take the same input and output as the standard recurrent network’s hidden layer, but has more parameter and gating units that control the flow of information going through the network.

Cells are connected recurrently to each other as the hidden unit of the ordinary recurrent neural network. An input feature is computed with a regular artificial neuron unit input gate, this gate can either allow or block incoming signals that would alter the state of the memory cell. The state unit has a linear self-loop whose weight is controlled by the forget gate, allowing it to forget its previous state if needed. The output gate can decide if the state of the memory cell will be allowed to affect other memory cells. All the gating units have a sigmoid nonlinearity, while the input unit can have any squashing nonlinearity including hyperbolic tangent and others. The output of a LSTM network can also be attached to
any type of output layer like any other neural network depending on the task at hand, whether that task is classification or regression or else.

The associate time constant of the self-recurrent connection is set between 0 and 1 via a sigmoid unit, which is controlled by the **Forget gate** unit $f_t$. Setting this time constant to 1 means the memory cells state will remain constant between different timesteps without any outside interference.

![Figure 2.7: The structure of a LSTM Memory Unit(Cell)](image)

**LSTM with Peephole**  What we have described so far is pretty normal LSTM cell, but not all LSTMs are the same as the above. In 2002 Gers[44] adds the “peephole” connecting the gates to the memory cell, as is shown in Figure 2.7, peepholes are gate layers that look at the cell state, which improved the LSTM’s ability to learn tasks that require precise timing and counting of the internal states.

**Bidirectional Long Short-Term Memory** Combining the Bidirectional Recurrent Neural Network discussed in Previous Chapter we can have the bidirectional LSTM[45], which can provide access to long range context in both forward and backward directions.

---

1The figure is taken from [43]
2.2.4.1 Gradient Calculation

Like the normal RNNs discussed before, LSTM is also a differentiable function approximator that can be typically trained by gradient descent methods. However, the first LSTM that was brought out in 1997 only used the combination of Real Time Recurrent Learning and Back Propagation Through Time (BPTT) to approximate the error gradient. The BPTT part was removed after one time step, because it was felt that long time dependencies would be dealt with by the memory blocks instead of the vanishing gradient of the activation between the recurrent connections. Truncating the gradient has the benefit of making the algorithm completely online, in the sense that weight updates can be made after every time step. This is an important property for tasks such as continuous control or time-series prediction. Additionally, it could only be proven with the truncated gradient that the error flow through the memory cells was constant. However, it is also possible to calculate the exact LSTM gradient with BPTT. As well as being more accurate than the truncated gradient, the exact gradient has the advantage of being easier to debug, since it can be checked numerically by using the method called symmetrical finite differences. This can be done by adding positive and negative perturbations to each weight and calculating the changes in the objective function:

$$\frac{\partial O}{\partial w_{ij}} = \frac{O(w_{ij} + \epsilon) - O(w_{ij} - \epsilon)}{2\epsilon} + O(\epsilon^2)$$  \hspace{1cm} (2.17)

The optimal value of $\epsilon$ depends on the computer architecture and floating point accuracy of a given implementation. And if the $\epsilon$ is too small will lead to numerical underflows and decreased accuracy.

2.2.4.2 LSTM Equations

In this section we will have a brief view of the forward pass and backward pass of a LSTM with peephole and forget gate. These equations are cited from a book focusing on RNNs and LSTMs by Alex Graves[37] As before, $w_{ij}$ is the weight of the connection from unit i to unit j, the weighted in put to unit i at time t will
denoted by $a^t_j$, and the corresponding activation value is $b^t_j$. The LSTM equations are given for a single memory block, and repeated calculating will yield the multiple blocks equations. We use $i$, $\phi$, $\omega$ to refer input gate, forget gate and output gate respectively, and $c$ refers to one of the $C$ memory cells. So the the peephole weights from cell $c$ to the input, forget and output gates are denoted $w_{ci}$, $w_{cb}$ and $w_{co}$ respectively. The state of the cell $c$ at time $t$ is denoted by $s^t_c$, $f$ is the activation function of the gates, and $g$ and $h$ are respectively the cell input and output activation functions.

Let $I$ be the number of inputs, $N$ be the number of outputs and $H$ be the number of cells in the hidden layer. Only the cell outputs $b^t_c$ are connected to the other blocks in the layer. The other LSTM activations, including the cell states, cell inputs, or other gate activations, are not visible to other blocks. We use the subscript $h$ to refer to cell outputs from other blocks in the hidden layer, exactly as for standard hidden units. We define $G$ as the total number of inputs to the hidden layer, including cells and gates and we use the subscript $g$ to refer to these inputs when we don’t need to distinguish between the input types.

The forward pass is calculated for an input sequence $x$ of length $T$ where you start at $t = 1$ and recursively using the update functions whilst incrementing $t$ after every step. The BPTT backward pass is calculated by starting at $t = T$ and recursively calculating the unit derivatives while decreasing $t$ after every step. By letting the $\delta^t_j$ to represent the partial derivative of Loss function with respect to the weighted sum input of any unit at time $t$, The final weight derivatives are found by summing over the derivatives at each time step.

### 2.2.4.3 Forward Pass

**Input Gates**

\[
a^t_i = \sum_{i=1}^{I} w_{ii}x^t_i + \sum_{h=1}^{H} w_{hi}b^{t-1}_h + \sum_{c=1}^{C} w_{ci}s^{t-1}_c \tag{2.18}
\]

\[
b^t_i = f(a^t_i) \tag{2.19}
\]
Forget Gates
\[
a^t_\phi = \sum_{i=1}^{I} w_{i\phi}x_i^t + \sum_{h=1}^{H} w_{h\phi}b^{t-1}_h + \sum_{c=1}^{C} w_{c\phi}s^{t-1}_c \tag{2.20}
\]

\[
b^t_\phi = f(a^t_\phi) \tag{2.21}
\]

Cells
\[
a^t_c = \sum_{i=1}^{I} w_{ic}x_i^t + \sum_{h=1}^{H} w_{hc}b^{t-1}_h \tag{2.22}
\]

\[
s^t_c = b^t_\phi s^{t-1}_c + b^t_\phi g(a^t_c) \tag{2.23}
\]

Output Gates
\[
a^t_\omega = \sum_{i=1}^{I} w_{i\omega}x_i^t + \sum_{h=1}^{H} w_{h\omega}b^{t-1}_h + \sum_{c=1}^{C} w_{c\omega}s^t_c \tag{2.24}
\]

\[
b^t_\omega = f(a^t_\omega) \tag{2.25}
\]

Cell Outputs
\[
b^t_c = b^t_\omega h(s^t_c) \tag{2.26}
\]

2.2.4.4 Backward Pass

We first define \(\epsilon^t_c\) and \(\epsilon^t_s\) as the partial derivative of the loss function \(L\) with respect to the cell output \(b^t_c\) and cell state \(s^t_c\).

Cell Outputs
\[
\epsilon^t_c = \sum_{n=1}^{N} w_{ck}\delta^t_k + \sum_{g=1}^{G} w_{cg}\delta^{t+1}_g \tag{2.27}
\]

Output Gates
\[
\delta^t_\omega = f'(a^t_\omega) \sum_{c=1}^{C} h(s^t_c)\epsilon^t_c \tag{2.28}
\]

Cell States
\[
\epsilon^t_s = b^t_\omega h'(s^t_c)\epsilon^t_c + b^t_\phi \epsilon^{t+1}_s + w_{cs}\delta^t_{i+1} + w_{cs}\delta^{t+1}_\phi + w_{cs}\delta^t_\omega \tag{2.29}
\]
Cells

\[ \delta^t_\omega = b^t g^t(a^t_c) \epsilon^t_s \]  

(2.30)

Forget Gates

\[ \delta^t_\phi = f^t(a^t_\phi) \sum_{c=1}^{C} s_{c}^{t-1} \epsilon^t_s \]  

(2.31)

Input Gates

\[ \delta^t_\iota = f^t(a^t_\iota) \sum_{c=1}^{C} g(a^t_c) \epsilon^t_s \]  

(2.32)

The original memory cell contained only input gates and output gates. The forget gates [30] and peephole weights [31] connecting the gates to the memory cell were added later on by other researchers. The purpose of the forget gate is to give memory cells a way to reset their states. To give an example as to why the ability to forget is useful, imagine that you’re analyzing a large collection of playlists. Once you reach the end of a playlist you know that the next playlist is unrelated to the one you just finished looking at, and therefore the memory cell should be reset. Peephole connections improve the LSTM’s ability to learn more difficult tasks that might require precise timing. In this thesis we are using the LSTM with forget gates and peephole connections added.

2.2.5 Residual Network

Though Long Time Short-Term Memory is one of the most effective structures that is used to control the information flow to solve the gradient vanishing problem in long term dependency in vanilla RNN. However, recent experiment has shown that training a LSTM RNN needs the carefully designed optimization procedure(Hochreiter et al., 2001; Pascanu et al., 2013; Dai and Le, 2015; Laurent et al., 2015; He et al., 2016; Arjovsky et al., 2015), especially when faced with unfolded very deep architectures for fairly long sequences (Dai and Le, 2015). While recently the residual network(ResNet)(He et al., 2015) has proved its success in several computer vision task combine with the convolution network, by learning a residual mapping between layers with the identity skip connections(Jaeger
et al., 2007). ResNet ensures a fluent information flow, leading to efficient optimization for very deep structures (e.g., with hundreds of layers). And there are also several works on the combination of ResNet and Recurrent Network with LSTM (Wang), where they achieved similar performance to vanilla LSTM in text classification but takes much less model parameter, which makes the training of the network easier.

ResNet (He et al., 2015) is composed of several stacked residual units, in which the \( l^{th} \) unit takes the following transformation:

\[
h_{l+1} = f(g(h_l) + F(h_l; W_l))
\]  

where \( h_l \) and \( h_{l+1} \) are the input and output for the \( l^{th} \) unit respectively. \( F \) is the residual function that have the weight \( W_l \), \( f \) is the ReLU function[19]. \( g \) is set as the identity function where \( g(h_l) = h_l \), which guarantees the direct propagation of signals among different layers, thereby avoids gradient vanishing (Wang). The recent paper (Liao and Poggio, 2016) also indicates the possibility of using shared weights in ResNet, as the normal RNN does.

![Figure 2.8: The structure of Recurrent Residual Network](image)

The basic structure of Recurrent Residual Network is shown in Figure 2.8, in which orange arrows indicate the identity connections from each \( h_{t-1} \) to \( h_t \), and blue arrows represent the recurrent transformations taking both \( h_t \) and \( x_t \) as input. RRN forces the direct propagation of hidden state signals between every two consecutive time steps with identity connections \( g \). In addition, the multiple non-linear transformations in \( F \) guarantees its capability in modeling complicated recurrent relationship.
2.2.6 Over-fitting Problem

As the data we collected from the user is still not big enough and the model we are using is quite complicated, it is important to address the problem of over-fitting during the training procedure. We introduced two methods we use in the implementation to deal with the problem in this section

2.2.6.1 Dropout

Dropout is a technique that was proposed in 2014[20], The idea of dropout is to randomly drop units (along with their connections but set their output to 0) from the neural network during training. This prevents units from co-adapting too much. During training, dropout samples from an exponential number of different “thinned” networks. At test time, it is easy to approximate the effect of averaging the predictions of all these thinned networks by simply using a single un-thinned network that has smaller weights. This significantly reduces over-fitting and gives major improvements over other regularization methods.

![Standard Neural Net](a) Standard Neural Net ![After applying dropout](b) After applying dropout.

Figure 2.9: An example of a thinned net produced by applying dropout to the network on the left. Crossed units have been dropped.

2.2.6.2 Cross Validation

In order to avoid the over-fitting problem in machine learning, we apply the K-fold Cross Validation[46] in the training. Cross validation is the a statistical method that used for examining the performance of the classifier, the idea is to separate
the data into training set and validation set, using the training set to train the model and then use the testing set to test the model’s performance. K-fold Cross Validation is one of the cross validation method that first divide the data in to K sets, and repeat the training K times to obtain K models, in each training process, each subset perform as the testing set once. The overall classification rate of the K model will be used as the performance of this K-CV classifier. Usually K is bigger than 2, we here did 10 experiments which mean K equals to 10. K-fold Cross Validation can avoid over-fitting and under-fitting.
Chapter 3

Method

3.1 Dataset

Dataset of Human Activity  The dataset we used for testing human activity is Human Activity Recognition Using Smart phones Dataset[7] released by UCI, where the experiments are carried out with a group of 30 volunteers within an age bracket of 19-48 years. Each person performed six activities (walking, walking upstairs, walking downstairs, sitting, standing, laying) wearing a smartphone (Samsung Galaxy S II) on the waist. We decided to use this dataset because it also use the accelerometer data as we do and it has quite lot of data. The raw data was being segmented by 50% overlapping sliding window of 2.56 second. As the data is sampled with 50Hz the so each window contains 128 data point. In total there are 10299 windows and each window contains 9 series of data i.e. 3-axial total acceleration, 3-axial body acceleration (filtered on the total acceleration) and 3-axial angular velocity. Figure 3.1 shows how the six activities are distributed in the dataset.

In Figure 3.2, we can see how the raw total acceleration data looks like, we only use this 3 axial data since it ease the comparison between the human dog. In each sub-figure a single window of activity is plotted. One serious problem is that the raw data of the dataset has some transition windows as shown in the Lower-Left figure(mainly in Sitting and Standing), this transition process is also
labeled in the dataset and this will affect the results if we are using the raw data to training and evaluate the model.

![Figure 3.1: Human Activity Data](image)

![Figure 3.2: Human Activity Signals](image)

**Dataset of Dog Activity**  The current data includes 7 volunteer dogs, each dogs perform activities in a normal life environment. The data we are using is collected from the off-the-shelf sensor with the mobile application. The mobile application as shown in Figure 3.3 can communicate with the sensor on the neck of the dog and send the collected data by email. Although the collected data have the accelerometer and gyroscope data and temperature and more, we only use
the 3-axial total acceleration with 28Hz frame rate to do the training and testing since the gyroscope should be calibrate but currently the users of our product do not calibrate them.

As shown in Figure 3.4, we have collected several type of data including walking, running eating and more. In total there are 2155 windows of data. One problem is that we didn’t collect same amount of data for different activity, some ac-
tivity (like running) are too little for training the model. Therefore we only used the Walking (389 windows), Eating (755 windows), and Resting (337 windows) to train and evaluate the model.

![Figure 3.5: Dog Activity Signals](image)

Figure 3.5 shows the raw 3 axial signals we used for training and testing the model. But since our volunteer dogs are not same in size; therefore, even for the same activity the wave form may various a lot among different dogs.

### 3.2 Tensorflow Deep Learning Framework

The implementation of the algorithm in this thesis is based on the Tensorflow Deep Learning Framework, which is one of the most popular open source deep learning framework developed by Google, it allow the user to quickly and efficiently implement machine learning algorithm especially the neural networks model, given the sufficient well coded functions already available in the library, as well as the google’s community supports.

Comparing to other deep learning framework, TensorFlow is developed for efficient parallelized computations on multiple devices[47], this feature is the main reason why we choose this library, this flexibility gives us further opportunity to improve the product after this thesis. Including building on-line self learning network across the users mobile application with little computational cost on the individual device. It also supports the cloud service with Docker technology.

Tensors is the basic data form in the frame work, which is defined as the multi-linear maps from vector spaces to real numbers, thereby all the scalars and vectors, matrices are different instance from a tensor, which can be regarded as the
multi-dimensional arrays.

### 3.3 LSTM Structure

For experiment we implement two LSTM model, the first one is simple a stacked unidirectional LSTM networks, and the second is a Residual LSTM model that stacked Residual Layers which contains bidirectional LSTM Layers inside each.

#### 3.3.1 Stacked Unidirectional LSTM

![Figure 3.6: Stacked Two Unidirectional LSTM](image)

The sample structure of the first network is shown in Figure 3.6, where two unidirectional LSTM layer is being stacked, inside each LSTM layer there are numbers of cells which is not shown in the picture. The choice of how many LSTM layer to stack and how many units in each layer is depends on the task and the complexity of the model. The parameter tuning part is discussed in the later Chapter.
### 3.3.2 Stacked Residual Bidirectional LSTM

Inspired by the structure proposed in a recent paper [48] by Google that uses residual LSTM networks to handle machine translation problems, we further stack layers which have residual hidden bidirectional LSTM layers inside, the basic structure is shown in Figure 3.7. There are total 32 LSTM units in each Bi-directional Layer which means 16 units in each direction.

#### 3.3.3 Configuration

**Activation** The activation function we used is rectifier function (R Hahnloser, 2000), which is defined as follows in Figure 3.8: This activation function has been proved more effectively (Xavier, 2011) and practical (Yann LeCun, 1998) than logistic sigmoid and hyperbolic tangent in convolutional network. The rectifier is also the most popular activation in deep neural network in 2015 (LeCun, 2015).

**Loss function** We choose the cross-entropy loss function (logistic loss function), the loss function is to measure the distance between the probability our model predicts to the true label (every ground-truth is an one-hot vector which has 1 on
the labeled class and 0 for other classes). The cross-entropy between two discrete probability distribution is defined as below:

\[ H(p, q) = -\sum_x p(x) \log q(x) \]  

(3.1)

where \( p(x) \) is the ground-truth and \( q(x) \) is the prediction. We also add the l2 with parameter lambda as the regularization term in order to prevent the coefficients to fit so perfectly to over-fitted. The reason we use l2 is to avoid the sparse weights.

**Dropout**  For the first model we have have input dropout for 15% and inter dropout rate 50% during the training part, and the second model we have both same input dropout and the inter dropout is performed between hidden LSTM layer but not between the stacked residual layers. Adding dropout will not only avoid the over-fitting but also increase the training speed, as the units that need to be update is less.

**Output**  We take the last output of LSTM layer at time T as the output class of the sequence, the initial output of the LSTM layer is a probability distribution over the total class we used for training, which is calculated by softmax function mentioned before in Equation 2.4, the function takes N dimensional vector and squashes it to a probability distribution that sums up to 1.
**Optimizer**  Instead of using Stochastic Gradient Descent (SGD) as the optimization methods, We use Adaptive Moment Estimation (Adam) optimizer (2014 Diederik). Adam is the variant version of RMSProp with momentum (Tieleman & Hinton, 2012; Graves, 2013), which computes individual adaptive learning rates for different parameters from estimates of first and second moments of the gradients. The advantage of Adam is that the learning rate will have a certain range after each iteration, which result in a more stable parameter during the training. By combine the strength of Adagrad in dealing with the sparse gradient and the pros of RMSprop in optimizing the RNN, Adam is therefore very suitable for nonconvex optimization i.e. good for big dataset (2014 Diederik)
Chapter 4

Experiment

4.1 Environment

We implement our algorithm by TensorFlow r1.0 with Python 3.5. The experiment is carried on a azure virtual machine that have a single NVIDIA Tesla K80 GPU with 12GB of GPU Memory.

4.2 Results on the UCI human action dataset

4.2.1 Evaluating

We evaluate the model by the test data after every training epoch, we later plot the accuracy and loss along the training procedure. The ideal model should not only have a high recognition rate on training data but also should have same performance on the validation data. Which mean the the loss and accuracy curve of training and validating should be as near as possible. If the training curve has a very good performance and the validation curve is diverge from the training, then we can say the model is over-fitting, since it only focus on the training data and does not perform on the new data it never seen before.
4.2.2 Parameter Tuning

As the training of the complex neural network model is a time consuming process even with the use of GPU, we need to find the configuration parameter for the training with some experiment on simple model. Therefore, we tune the three parameter by a stacking 3 stack unidirectional and the results has been shown below, we test for different learning rate 0.001, 0.0025 and 0.007, different clip gradient (used to clip the gradient and avoid vanishing (exploring)) 5, 10, 15, and two lambda 0.0025 and 0.005.

<table>
<thead>
<tr>
<th>Learning rate</th>
<th>Clip gradient</th>
<th>5</th>
<th>10</th>
<th>15</th>
</tr>
</thead>
<tbody>
<tr>
<td>0.001</td>
<td></td>
<td>92.63</td>
<td>92.80</td>
<td>92.60</td>
</tr>
<tr>
<td>0.0025</td>
<td></td>
<td>92.50</td>
<td>92.26</td>
<td>91.51</td>
</tr>
<tr>
<td>0.007</td>
<td></td>
<td>36.27</td>
<td>91.75</td>
<td>37.08</td>
</tr>
</tbody>
</table>

Table 4.1: Best Testing Accuracy With Lambda 0.0025

<table>
<thead>
<tr>
<th>Learning rate</th>
<th>Clip gradient</th>
<th>5</th>
<th>10</th>
<th>15</th>
</tr>
</thead>
<tbody>
<tr>
<td>0.001</td>
<td></td>
<td>91.51</td>
<td>91.41</td>
<td>92.50</td>
</tr>
<tr>
<td>0.0025</td>
<td></td>
<td>91.99</td>
<td>41.36</td>
<td>90.12</td>
</tr>
<tr>
<td>0.007</td>
<td></td>
<td>91.24</td>
<td>18.22</td>
<td>62.91</td>
</tr>
</tbody>
</table>

Table 4.2: Best Testing Accuracy With Lambda 0.005

Learning Rate As what we can see from the results, bigger learning rate tend to make the learning curve to be not robust and result in a worse accuracy (indicated by the red box in the table 4.1 and 4.2), therefore we choose the learning rate 0.001 in the later experiment. And then we find the optimal lambda and the clip gradient value by analyzing the training and testing curve.
**L2 Parameter Lambda**  From the Figure 4.1 and 4.2 we can see, though they both have a almost same curve for the accuracy, the loss curve when $\lambda$ is 0.0025 shows a salient gap between Training and Validating. Therefore we choose the $\lambda = 0.005$ and $clip - gradient = 15$ as the parameter we used later.

### 4.2.3 Result on Stacked Residual Model

We further use the sophisticated residual model to test on the data, among all we test for several configuration and found three stacked layer with three residual bidirectional layers inside each has the best result on overall accuracy at 93.1%. We plot the confusion matrix as below:
4.2.4 Discussion

We can see from the Precision Matrix 4.3b that the recall drop occurred on Sitting and Standing, with the sitting activity having lowest recall rate equal to 80%. The noticeable misclassification between these two activity are largely attributed to the physical location of the device and its difficulty to categorize them, as mentioned in Paragraph 3.1 and Figure 3.2, the raw data may not be correctly labeled. The precision rate for walking, walking upstairs, laying are pretty high and almost 100%, though there is a small drop on the recall rate for walking downstairs. The overall F1 score is 93.7% which shows the model is a good fit over the five activities. Although comparing to the results in the original paper of this dataset[7], where they reach the best overall accuracy 96% by calculating features vectors and use multi-class SVM to learn the model, we are not as good as them. However, since we only use the raw data without using signal processing and pattern extraction, the results are are still good in practice. Since the human data set is not big enough, and our final product will be on our own dog dataset, we can expect a better results once more data is aggregated.
4.3 Results on dog dataset

4.3.1 Preprocessing of Data

The raw data is 6 rows of 3-axial linear acceleration and 3-axial angular velocity, we divide the data to 2 set for the model to learn, where 30% is for testing and 70% for training.

We first apply the de-noising filters with the signal and then sampled with fixed width sliding window of 2.56 second and 50% overlapping. The sensor acceleration signal, which has the gravitational and the body motion components was separated using a Butterworth low-pass filter in to body acceleration and gravity. The gravitational force is assumed to have only low frequency components, therefore a filter with 0.3 Hz cutoff frequency was used.

4.3.2 Best Result on Stacked Residual Network

Our best results of the current dog dataset with three activity (Resting, Running, Eating) is 94.6%. With the F1-score 94.6%. The confusion matrix is shown below in Figure. 4.4 and the loss and accuracy curves are shown in Figure 4.5.

4.3.3 Discussion

As we can see in Figure 4.5, the best testing accuracy is at 197 epoch, where the corresponding training accuracy is 97%, the gap between training and testing is not big, therefore we can conclude the model is not over-fitting a lot. One problem with the result is that although we only use the data that has enough amount for training and evaluating the model, the data is still not evenly distributed, the model always tend to predict the data to the where most of the data is located, as shown in Figure 4.4a, which will make the overall accuracy not exactly reflect the performance of the model. This unbalanced data problem is frequently appears in the machine learning problem, and one of the most frequently used solution is to use F1 Score(or F-score) to evaluate the performance. Here we have
reached the best test F-score also at epoch 197, and the best F1-score is 94.6%, which proved our model is good in practice.
Chapter 5

Conclusion & Future Work

5.1 Conclusion on the performance of model

After evaluating the model in the previous chapter, it shows that our model is capable of reference different of activities with limited data. By saying limited data we means that comparing our input to the model to the other pattern recognition method where hundreds of patterns are calculated for training the model, we only use 9 series of data on human activities and 3 series of data for dog activities. Although the final dog results are not super good in academic perspective, but it is enough for the product to serve the customers.

5.2 Future Work

The current dog dataset is not evenly distributed as mentioned in the previous chapter, as the final goal of the product is to distinct all the activities shown in Figure3.4, more data should be collected. As we can see in the results on human dataset, the current model is still not perform well between some activities that have similar waveform, this problem may also appears in the future dog model. Therefore, the building sub-model for those activities that are ease to be confused could improve the model a lot. My suggestion is to build a simple sub model especially for those activities like SVM(Support Vector Machine), and set a criteria of which model should be choose by conducting experiments on the future data.
And TracyTrackers are also works on the immigration of the code to IOS and other mobile application. And by using the distributed computing power on TensorFlow network, the training rate can be faster and the online model can also be built.
Bibliography


[27] Karl Moritz Hermann, Tomás Kociský, Edward Grefenstette, Lasse Espeholt, Will Kay, Mustafa Suleyman, and Phil Blunsom. Teaching ma-


[48] Yonghui Wu, Mike Schuster, Zhifeng Chen, Quoc V. Le, Mohammad Norouzi, Wolfgang Macherey, Maxim Krikun, Yuan Cao, Qin Gao, Klaus Macherey, Jeff Klingner, Apurva Shah, Melvin Johnson, Xiaobing Liu, Lukasz Kaiser, Stephan Gouws, Yoshikiyo Kato, Taku Kudo, Hideto Kazawa, Keith Stevens, George Kurian, Nishant Patil, Wei Wang, Cliff Young, Jason Smith, Jason Riesa, Alex Rudnick, Oriol Vinyals, Greg Corrado, Macduff Hughes, and Jeffrey Dean. Google’s neural machine translation system: Bridging the
Appendix A

Unnecessary Appended Material