Tokyo University of Agriculture and Technology

Professor Kondo's Laboratory

Final Report

JASSO PROGRAM

Vu Thanh Dang

student from Ho Chi Minh University of Science

2017/09/20

1 Introduction

The purpose of this report is to summarize my project for two months at Tokyo University of Agriculture and Technology. In this report, I concentrate on two main tasks which I have accomplished. The first task is to study Deep Neural Networks (DNNs) and how to use TesorFlow library to model DNNs. Another issue is the Activity Classification (AC) problem in which I built a classifier and produced some results in REALDISP dataset.

2 Deep Neural Networks and TensorFlow

2.1 Multilayer Perceptron

A multilayer perceptron (MLP) (Figure 1) is a class of feedforward artificial neural networks. An MLP consists of more than two layers of units. Except for the input and output layer, MLP has many hidden layers for the purpose of simulating the complex architecture of biological neural networks.

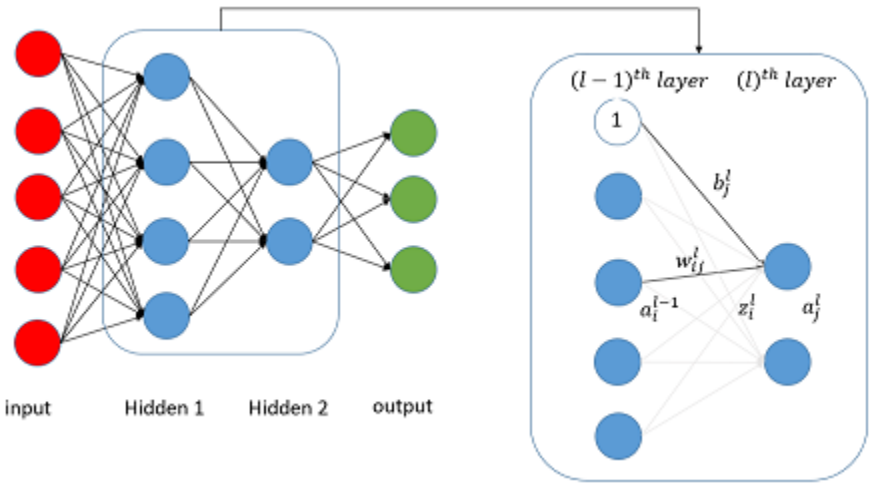

Figure 1: Multilayer perceptron

The units in a layer are the main constituent of DNNs, I explained some notations of them in Figure 1 and below.

$$z_j^l = w_j^l a^{l-1} + b_j^l \tag{2.1.1}$$

$$z^l = W^l a^{l-1} + b^l \tag{2.1.2}$$

$$a^l = f(z^l) \tag{2.1.3}$$

In (2.1.2) equation, W^l is a weight matrix and b^l is a bias vector of l^{th} layer. Besides, an activation function is also important feature in DNNs (f in (2.1.3) equation), this function is a nonlinear function for learning the nonlinear boundaries in supervised learning tasks. Sigmoid, tanh and ReLU functions are some commonly used activation functions (Figure 2).

2.2 Back Propagation Algorithm

The cost function is a measure of the predicted value of models and the correct value. Therefore, the main ideal of learning in DNNs is to find the value of parameters to minimize the cost function. In some machine learning

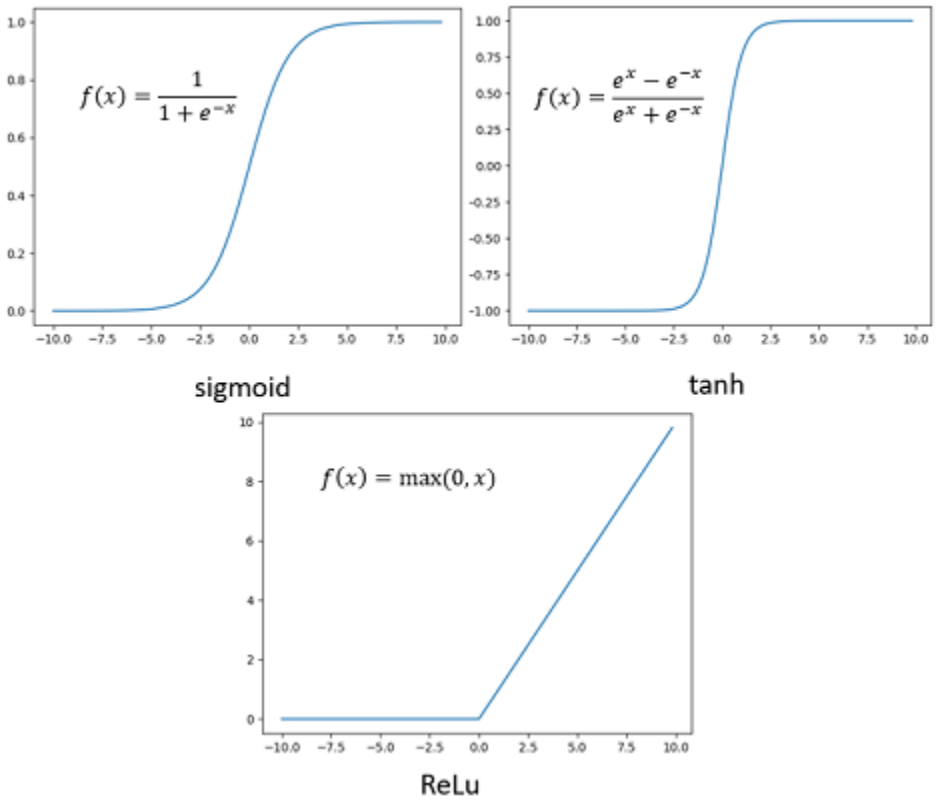

Figure 2: Some activations function

algorithms, Mean Squared Error (MSE) is the most used cost function.

$$J(W, b, X, Y) = \frac{1}{N}\sum_{n=1}^{N} ||y_n - a_n^L||_2^2 \tag{2.2.1}$$

With N is the number of data points and L is the last layer in DNNs.

However, the softmax function (2.2.2) in output layer combines with cross-entropy cost function (2.2.3) commonly used in DNNs.

$$\hat{y}_j = softmax(z_j^L) = \frac{e^{z_j^L}}{\sum_{j=1}^{C} e^{z_j^L}} \tag{2.2.2}$$

$$J(W, b, X, Y) = \frac{1}{N}\sum_{n=1}^{N}\sum_{j=1}^{C} y_{ij} log(\hat{y}_{ij}) \tag{2.2.3}$$

Gradient Descent (GD) is an iterative technique for optimizing cost function, in DNNs, GD is constructed via back propagation (BP) algorithm. BP update parameters based on chain rule to calculate the gradient of cost functions with respect to parameters (weights and biases in DNNS). There are two main steps in BP, the feed forward step calculates all units in DNNs from the input layer to the output layer and the backward step updates all parameters in reverse.

step 1: forward

$$z^l = W^l a^{l-1} + b^l$$
$$a^l = f(z^l)$$

step 2: backward

$$W = W - \alpha \nabla_W(J)$$
$$b = b - \alpha \nabla_b(J)$$

In real applications, GD is rarely used because of the local optimum problem and slow training. Therefore, Stochastic Gradient Descent (SGD) is a commonly used algorithm for training DNNs. Instead of using all data points in one iteration as GD, SGD uses a small subset of data and increases the number of iterations. Moreover, SGD also incorporates some algorithms which have the ability to avoid local optimum such as Momentum GD, Nesterov Momentum or adapt learning rates such as AdaGradient.

2.3 Convolutional Neural Networks

Convolutional Neural Networks (CNN) are very similar to MLP, the main ideal of CNNs is weight sharing. Instead of completely connecting from all units in the previous layer to a unit in the current layer, CNNs use only small units subset for calculating. Therefore, the weight matrix from one layer to the next serves as a kernel in convolution operator. (Figure 3)

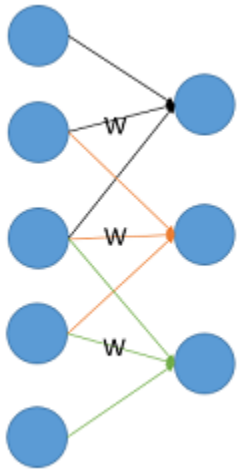

Figure 3: Convolution Layer

The most popular application of CNNs is image classification because the convolution layer is capable of extracting local features in the image. Besides, the weight sharing ideal in CNNs make it easy to train because of fewer parameters to learn.

In CNNs, the hidden layer is constructed from some layers belong to three following types:

1. Convolution layer: extract local features
2. Pooling layer: reduce dimension
3. Fully conected layer: encode featrues into vector

The core component in CNNs is a kernel, which is similar to a filter in signal processing, the kernel generates a meaningful impulse response. However, CNNs must learn a full kernel (7x7, 9x9 parameters) and that is very time-consuming. So instead of learning the whole kernel, we can assume a form of them such as Gaussian (2.3.1) or Laplace kernel. The assumption about forms of kernels makes the model not general, but we can reduce the number of parameters.

$$k(\mu, \sigma) = \frac{1}{\sqrt{2\pi}\sigma} e^{-\frac{(x-\mu)^2}{\sigma^2}} \tag{2.3.1}$$

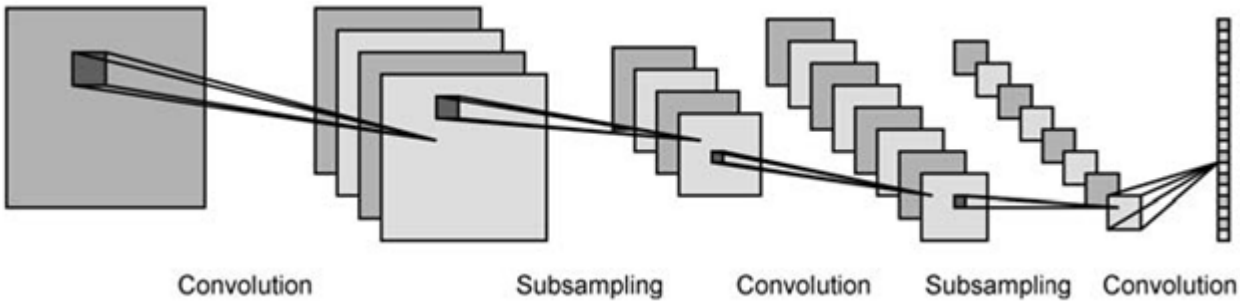

Figure 4: Convolutional Neural Networks

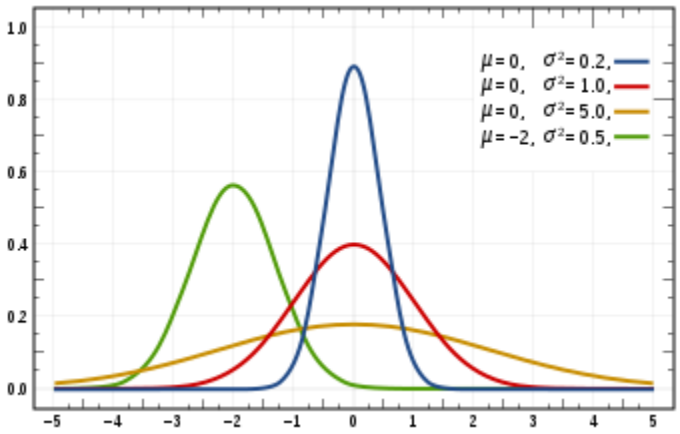

Figure 5: Gaussian distribution

2.4 Recurrent Neural Networks

The ideal behind Recurrent Neural Networks (RNNs) is the use of inherited information. In feedforward NNs, we assume that all inputs are neither dependent of each other nor in order. Another way to think about RNNs is that they have a "memory" which captures the information about what has been calculated so far (Figure 6).

Some notations in RNNs are listed below and in Figure 6.

$$s_t = f(Ux_t + Ws_t - 1) \tag{2.4.1}$$

$$o_t = Vs_t \tag{2.4.2}$$

RNNs also share the same parameters (U, V, W) across all steps. These model treat different inputs with the same task at each step to significantly reduces the total number of parameters we have to learn. Depending on the task, Moreover, the output at each step may not be necessary.

GD is also used to train RNNs model via Back Propagation Through Time (BPTT) algorithm. However, the number of stacked layers in RNNs make them hard to train because of vanish gradient problem. Truncated BPTT is arguably the most practical method for training RNNs and achieves many good results on language modeling [1]. It processes the sequence one timestep at a time, and every k_1 timesteps, it runs BPTT for k_2 timesteps, so a parameter update can be cheap if k_2 is small.

Truncated BPTT:
for t in [1, 2, ..., T]
calculate s_t, o_t

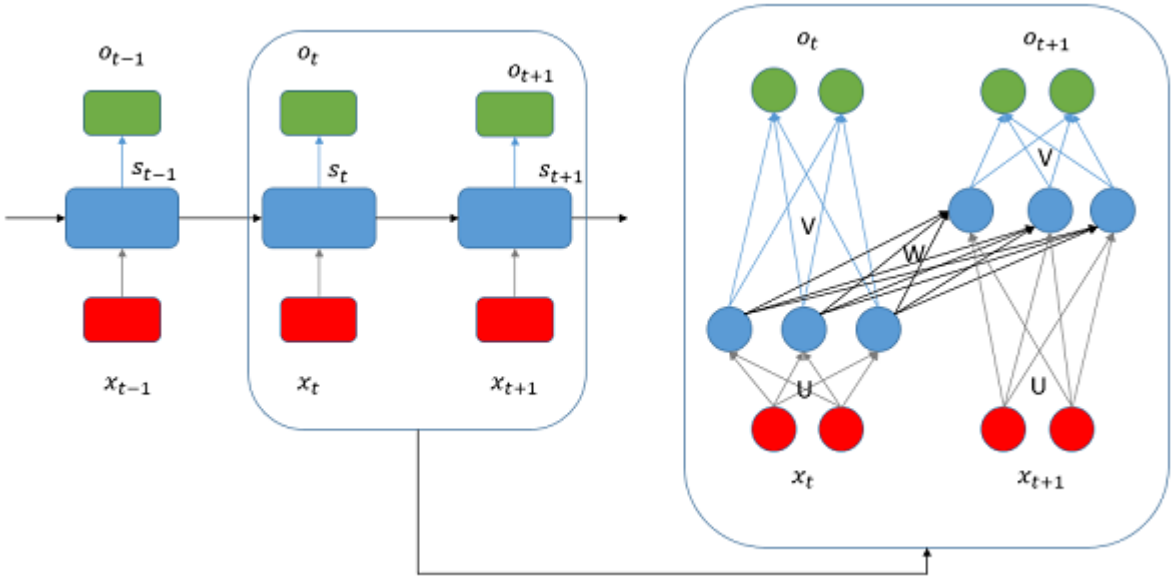

Figure 6: Recurrent Neural Networks

if t divides k_1 then
backward update from t to t - k_2
end if
end for

The disadvantage of ordinary RNNs is that they failed to capture long sequences. Information in the first few layers go through many layers that they no longer meaningful to the current state. Fortunately, we can replace the ordinary RNNs cell with Long short-term memory (LSTMs) cell in recurrent model. LSTMs have a gate system that helps the model to determine what information is stored in its "memory" and what information needs to pass through the next layer.

2.5 TensorFlow

TensorFlow is an open source software library for numerical computation using data flow graphs. Nodes in the graph represent mathematical operations, while the graph edges represent the multidimensional data arrays (tensors) communicated between them. Tensorflow was originally developed by members working on the Brain Team within Google's Machine Intelligence research organization. TensorFlow is a framework for deploying machine learning and deep neural networks model. This library also supports for computers with single or more NVIDIA GPUs.

MNIST is a simple computer vision dataset. It consists of images of handwritten digits with size 28x28 (Figure 7). For the purpose of learning how to use TensorFlow, I built a simple CNN and RNN model for classification task and experimented in this dataset.

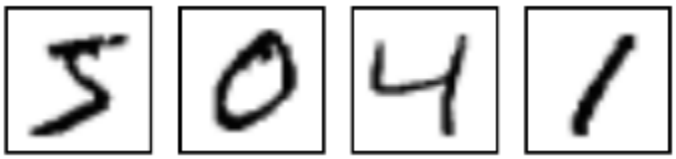

Figure 7: MNIST dataset

In CNN model, I use two convolution layers and two fully connected layers. After each convolution layer, I also add a maxpooling layer for dimension reduction. Units in layers go to ReLU activation function except for the last layer with softmax function (Figure 8).

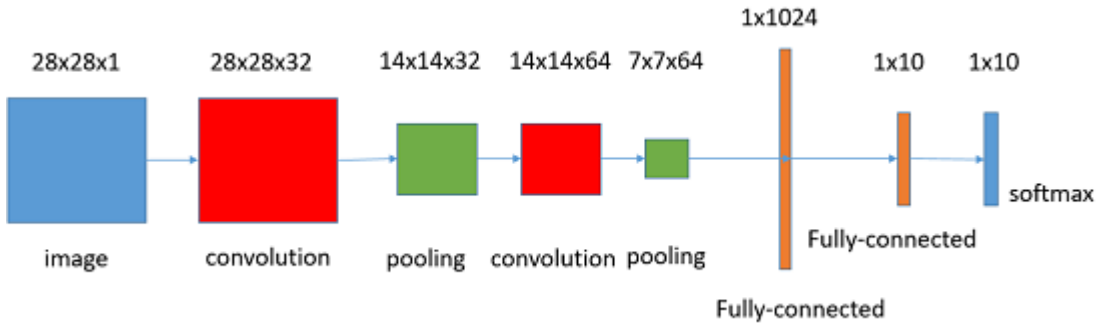

Figure 8: CNN model

Because RNN model is only suitable for sequential data, the image is subdivided in rows and each row goes into RNN cell. In the top of the model, I also use one fully connected layer with the softmax function (Figure 9). The accuracy of two models is shown in Figure 10.

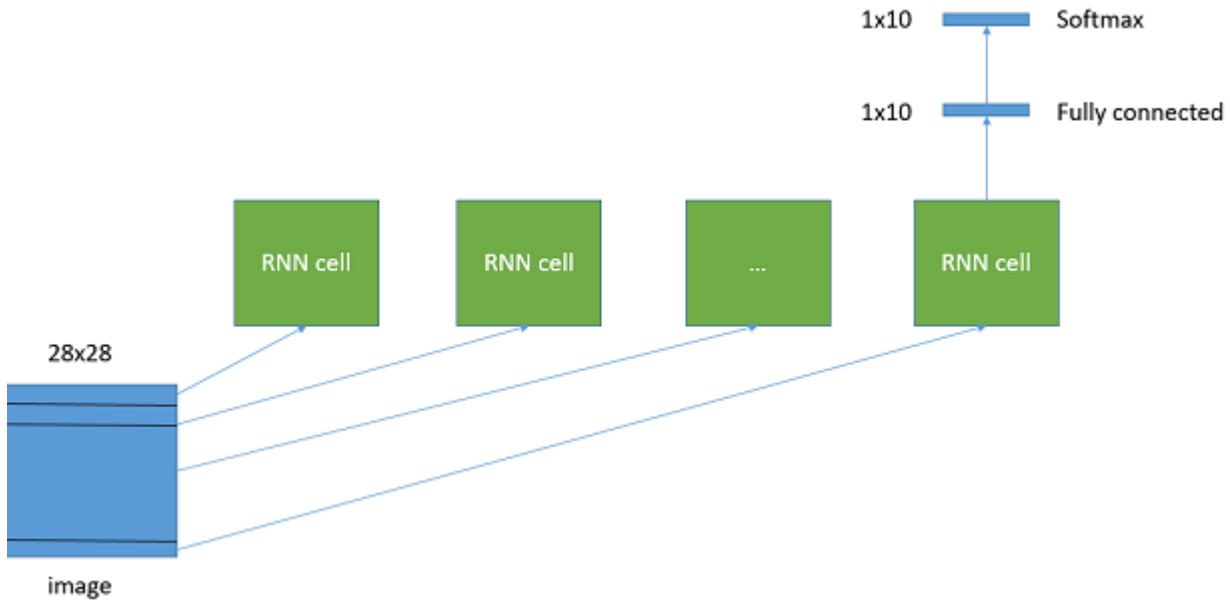

Figure 9: RNN model

3 Activity Classification

The analysis of the daily living human activity has been the foundation for behavioral health applications. Thanks to the rapid development of wearable devices, we can easily gather information about activities. Therefore, the next thing is to focus on understanding activity based on that information. In this report, I concentrate on activity classification (AC) task because of determining these activities is the premise for further processing.

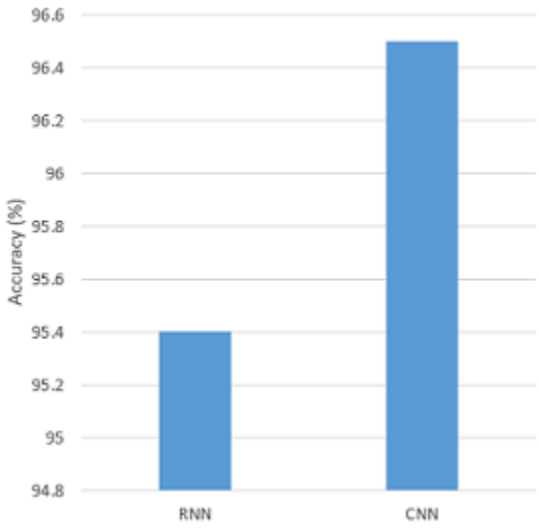

Figure 10: Accuracy result from the CNN and RNN models for the MNIST dataset

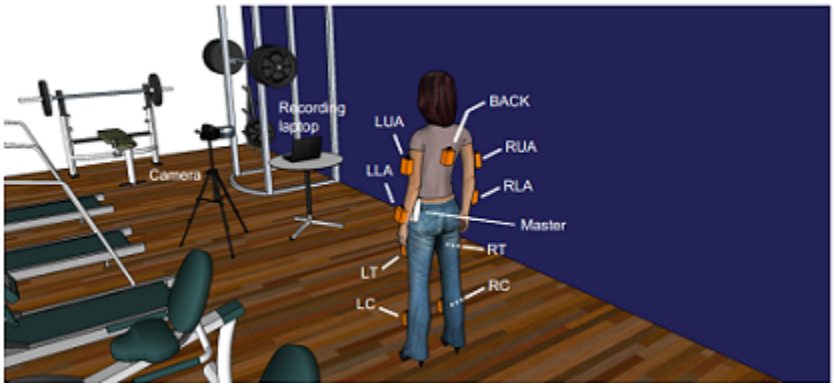

Figure 11: The position of sensors in REALDISP dataset

3.1 Realistic sensor displacement dataset

The REALDISP (REAListic sensor DISPlacement) dataset has been originally collected to investigate the effects of sensor displacement in the activity recognition process in real-world settings. The dataset includes a wide range of physical activities (33 activities), sensor modalities (9 sensors) and participants (17 subjects). [2] The dataset includes 3 scenarios:

1. Ideal-placement: The sensors are positioned by the instructor to predefined locations within each body part.
2. Self-placement: The user is asked to position sensors himself on the body part.
3. Mutual-displacement: An intentional de-positioning of sensors using rotation and translations with respect to the ideal-placement.

9 sensors are attached to 9 body parts (Figure 11), each sensor measures 3D acceleration (x, y, z), gyroscope (x, y, z), magnetic field measurements (x, y, z) and orientation quaternion (q1, q2, q3, q4). Records are sampled at 50 Hz. The list of activities is shown in Figure 12.

Activity set		
L1: Walking (1 min)	L12: Waist rotation (20x)	L23: Shoulders high amplitude rotation (20x)
L2: Jogging (1 min)	L13: Waist bends (reach foot with opposite hand) (20x)	L24: Shoulders low amplitude rotation (20x)
L3: Running (1 min)	L14: Reach heels backwards (20x)	L25: Arms inner rotation (20x)
L4: Jump up (20x)	L15: Lateral bend (10x to the left + 10x to the right)	L26: Knees (alternatively) to the breast (20x)
L5: Jump front & back (20x)	L16: Lateral bend arm up (10x to the left + 10x to the right)	L27: Heels (alternatively) to the backside (20x)
L6: Jump sideways (20x)	L17: Repetitive forward stretching (20x)	L28: Knees bending (crouching) (20x)
L7: Jump leg/arms open/closed (20x)	L18: Upper trunk and lower body opposite twist (20x)	L29: Knees (alternatively) bend forward (20x)
L8: Jump rope (20x)	L19: Arms lateral elevation (20x)	L30: Rotation on the knees (20x)
L9: Trunk twist (arms outstretched) (20x)	L20: Arms frontal elevation (20x)	L31: Rowing (1 min)
L10: Trunk twist (elbows bended) (20x)	L21: Frontal hand claps (20x)	L32: Elliptic bike (1 min)
L11: Waist bends forward (20x)	L22: Arms frontal crossing (20x)	L33: Cycling (1 min)

Figure 12: All activities in REALDISP dataset

3.2 Classification Model

In AC task, I built Neural Network models to classify 10 whole body activities with single sensor (RC/3D accelerations) and 33 activities with 4 sensors (LLA, RLA, LC, RC). Activities are measured over a long period, so I decide to divide the activity at 4 seconds (Figure13). That not only increases the number of data points (activities) for the model but also normalizes the size of data points.

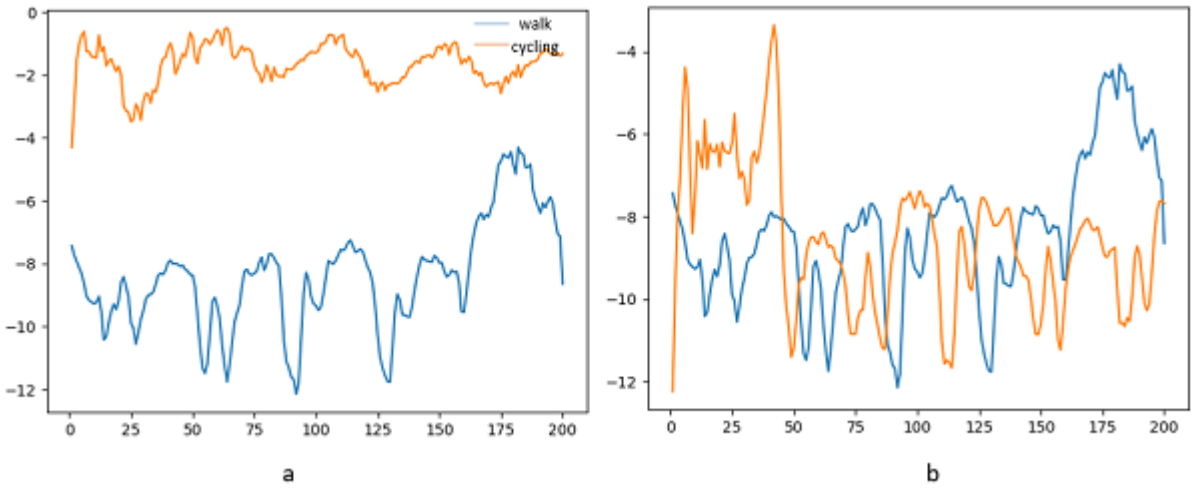

Figure 13: X-axis acceleration signal of RC sensor: a. walking-cycling, b. walking-walking

Because data points are time series data, RNN is the most suitable model for this AC task. However, the raw data is inappropriate for the model to learn, so I added some convolution layers for extracting local features of signal. My model for classifying 10 whole body activities is summarized in Figure 14.

In 10 whole body activities classification model, I constructed two convolution layers and two pooling layers at the bottom of model. After extracting features from raw signals, I concatenated data into a matrix whose row represent for time and column for features. Then, a matrix is broken down over time and pass through LSTM cell. At the top of model, I used one fully connected with softmax layer for encoding feature into probability vector. I also used cross-entropy cost function with one-hot coding label for 10 activities. Finally, the model was trained with Adagrad algorithm[3].

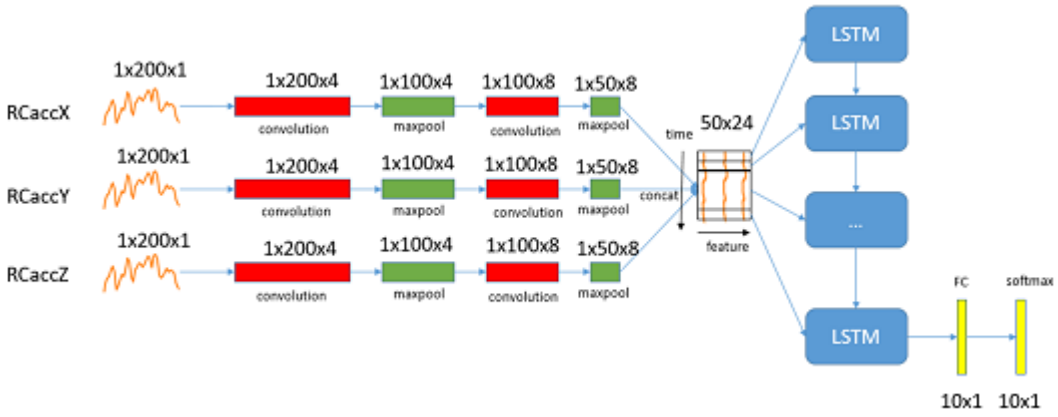

Figure 14: Classification model for 10 whole body activities

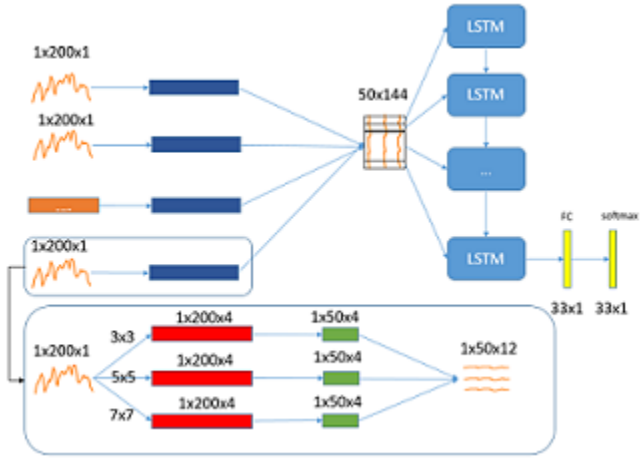

Figure 15: Classification wide model for 33 activities

Activities used for 10 whole body activities classification are listed below:

1. Walking
2. Jogging
3. Running
4. Jump front back
5. Jump sideways
6. Jump leg/arms open/closed
7. Jump rope
8. Rowing
9. Elliptic bike
10. Cycling

The model for classifying 33 activities have similar architecture to above model but with more signals (12 signals) from RC, LC, LLA, RLA sensors. Thus the matrix for RNN part size is 50x96. Inspired by the ideal of Inception module in GoogleNet, I also constructed a model with the ability to extract small and large features (Figure 15). In this model, I replace sequential convolutional and pooling layers by the wide module. In this module, original signals are convoluted with 3x3, 5x5 and 7x7 kernel. They then go through pooling layer for calculating the max value and reducing dimension. 12 signals produced by the wide module are concatenated and pass through the recurrent part. I also used drop out technique [4] in fully connected layer to avoid overfitting.

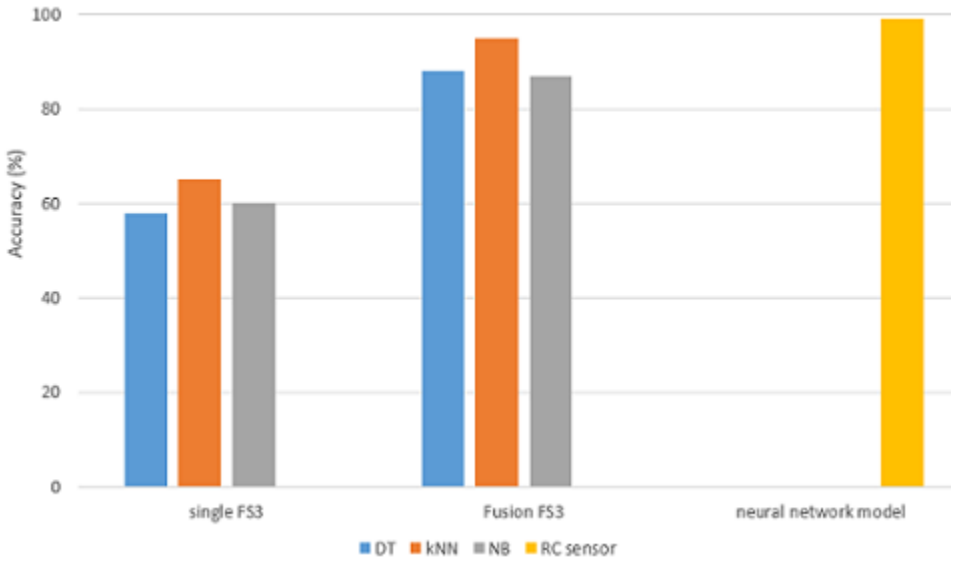

Figure 16: 10 activities classificaition experiment's accuracy

3.3 Experiment and Result

In my experiment, I used cross-validation to evaluate the accuracy of models. I also compare the result of my method with a single sensor (LLA) and decision fusion model for self-placement scenario [5] [6]. In these model, authors use some classifiers (kNN, Navie Bayes, decision tree) with new feature sets such as FS1 (mean), FS2 (mean, standard deviation) and FS3 (mean, standard deviation, maximum, minimum, mean crossing rate). Activities from ideal-placement and self-placement scenarios are used for my models. In this dataset, I used 90 percent for training and 10 percent for validating.

In 10 activities classification task, I achieved very high accuracy (about 98%) compare to the highest accuracy in single sensor model (65 %) and fusion sensors model (95 %) (Figure 16). I also achieved high accuracy when using 12 signals from 4 sensors LC, LLA, RC and RLA for 33 activities classification task with the ordinary model (93 %) and the wide model (92 %) (Figure 17).

My Neural Networks models converge quickly (about 1000 iterations) (Figure 18) and that help to reduce the training time. Due to less complexity of models, the process of recognizing activity is very fast, about 4s to record data and 1s to calculate in the model.

4 Conclusion

In this report, I summarized about my work for two months when I have taken part in JASSO program at Tokyo University of Agriculture and Technology. Studying about DNNs and TensorFlow in the first month help me with the background for AC project in the second month. My models achieve good performance with REALDISP dataset

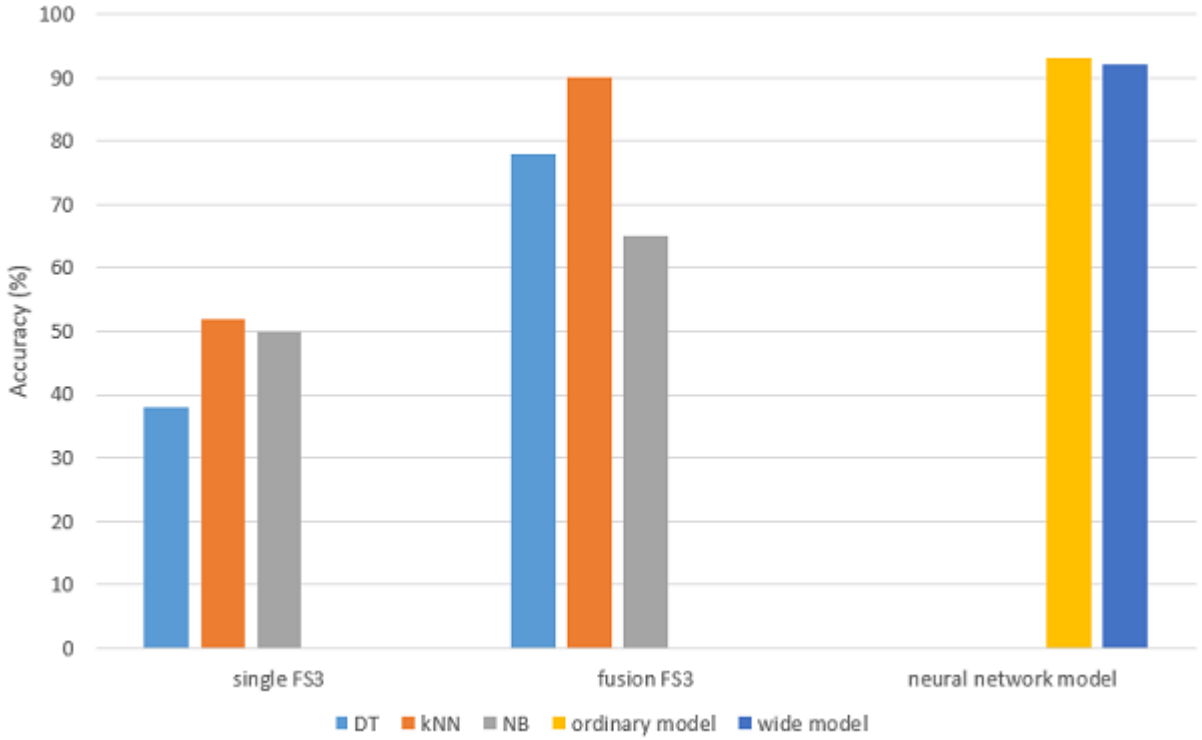

Figure 17: 33 activities classificaition experiment's accuracy

and they also easily extends to multiple sensors. Using convolution layers in the bottom of model help to extract signal features without regard to the difficulty of the preprocessing step. Convolution layers also do not affect the chronological order of signal. However, building the classifier on a small dataset (17 participants) can not draw any conclusions in real life applications. We have to get more data to increase the reliability of classifiers. One way to increase the amount of data is to divide the time interval of activity overlap. The wide module has the ability to combine small features and large features in one layer, so I think we can get the better result when using more data into the model has more wide modules.

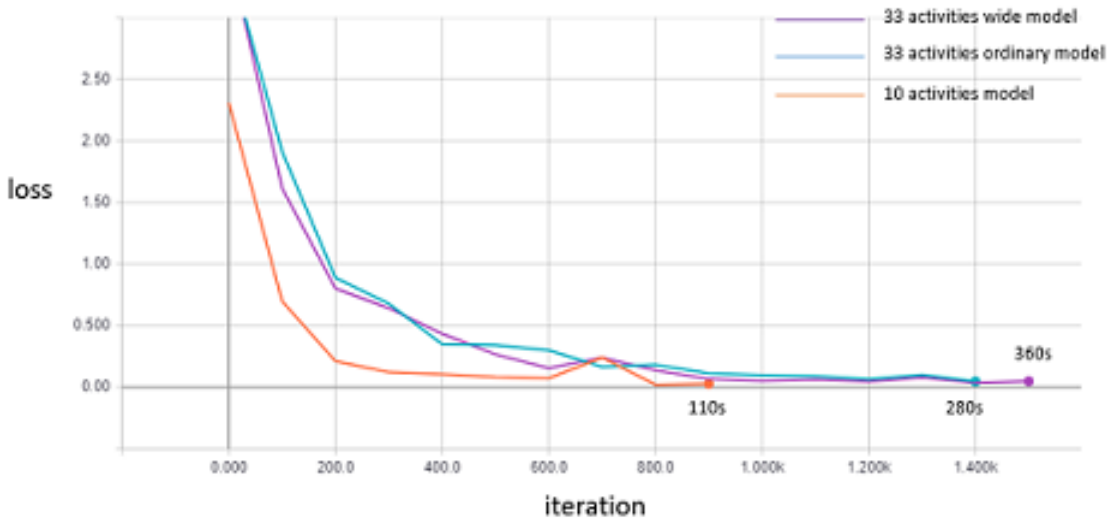

Figure 18: Total time and loss in Neural Network models

References

[1] Ilya Sutskever. Training recurrent neural networks. *University of Toronto, Toronto, Ont., Canada*, 2013.

[2] Oresti Banos, Mate Toth, Miguel Damas, Hector Pomares, and Ignacio Rojas. Dealing with the Effects of Sensor Displacement in Wearable Activity Recognition. *Sensors*, 14(6):9995–10023, June 2014.

[3] John Duchi, Elad Hazan, and Yoram Singer. Adaptive subgradient methods for online learning and stochastic optimization. *Journal of Machine Learning Research*, 12(Jul):2121–2159, 2011.

[4] Nitish Srivastava, Geoffrey E. Hinton, Alex Krizhevsky, Ilya Sutskever, and Ruslan Salakhutdinov. Dropout: a simple way to prevent neural networks from overfitting. *Journal of machine learning research*, 15(1):1929–1958, 2014.

[5] Oresti Baños, Juan Manuel Galvez, Miguel Damas, Alberto Guillén, Luis Javier Herrera, Héctor Pomares, and Ignacio Rojas. Evaluating the effects of signal segmentation on activity recognition. In *IWBBIO*, pages 759–765, 2014.

[6] Oresti Banos, Miguel Damas, Hector Pomares, Fernando Rojas, Blanca Delgado-Marquez, and Olga Valenzuela. Human activity recognition based on a sensor weighting hierarchical classifier. *Soft Computing*, 17(2):333–343, February 2013.