ECSE 6965 Deep Learning
Programming Assignment 4

Xiangyang Mou
moux4@rpi.edu

April 20, 2018

1 Theory

For this programming assignment, we are supposed to implement a recurrent neural network and apply the model to gesture pose estimation. A recurrent neural network(RNN), at its most fundamental level, is another type of densely connected neural network. The idea behind RNNs is to make use of sequential information. In a traditional neural network we assume that all inputs and outputs are independent of each other.

1.1 Basic RNN

Fig.(1) shows a very basic structure of RNN with one layer of cells in a length of 3.

- x_t is the input at time t. The input could be a vector or a feature map extracted from a CNN.
- s_t is the hidden state at time t. It's the "memory" of the network. It is calculated based on the previous hidden state. The information from previous states would be accumulated through W
- o_t is the output of the cell at time t. It is also regarded as the output of the model if the cell is the last "vertical" layer of the model. If there are other layers on the top, o_t would be the input of the cell in the higher layer.
- U and W are the input weights for each cell. They follow the equation $s_t = tanh(Ux_t + Ws_{t-1} + b_0)$, where b_0 is the bias.
- V is the output weights for each cell. For regression problem, $s_t = tanh(Vs_t + b_1)$; for classification problem, $s_t = softmax(Vs_t + b_1)$. b_1 is the bias.
- U, W and V are all shared for all samples in each sequence.

1.2 Long Short Term Memory (LSTM)

Fig.(2) shows a detailed structure of the cell of LSTM. Different from basic RNN, LSTM add an extra pathway of memory, C. Note that h_{t-1} in Fig.(2) is equivalent to s_{t-1} in Fig.(1).

- $f_t = \sigma(W_{hf}h_{t-1} + W_{if}x_t + b_f)$ is the forget gate.

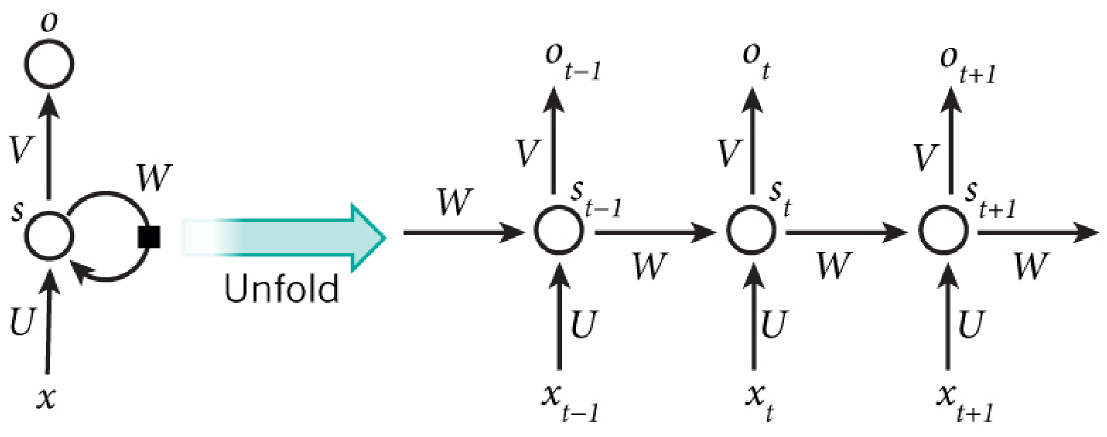

Figure 1: A Basic Structure of RNN

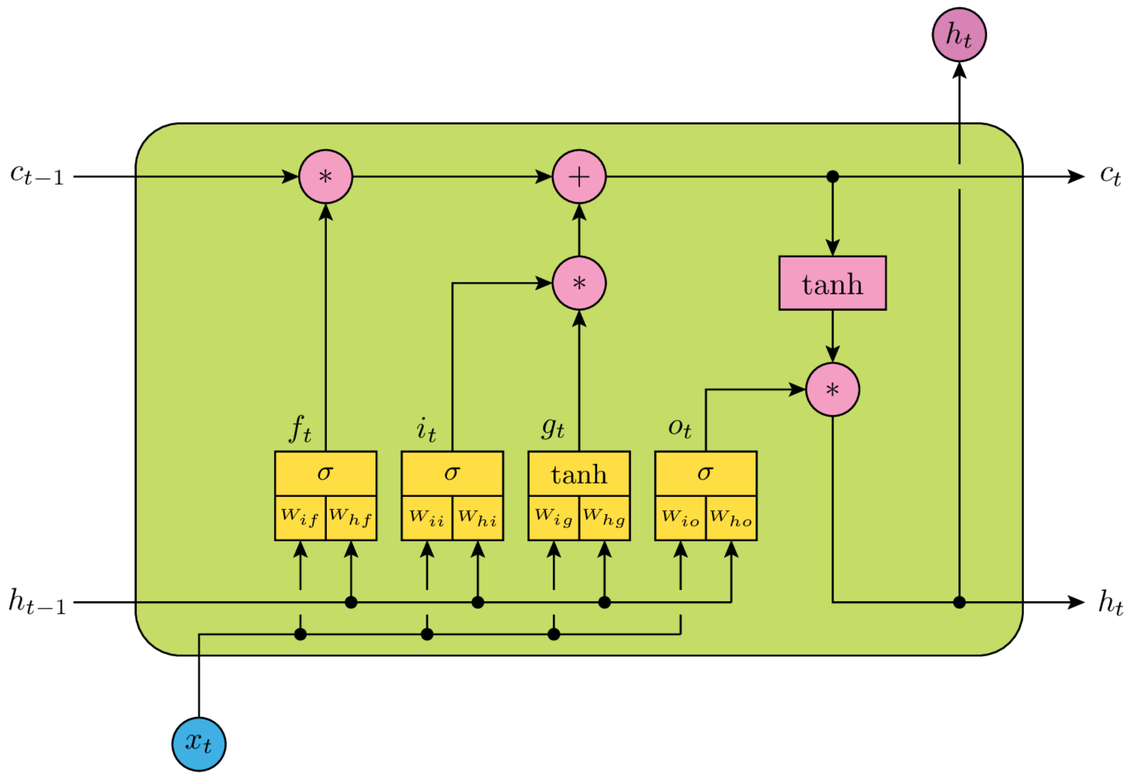

Figure 2: A Basic Structure of each cell in LSTM

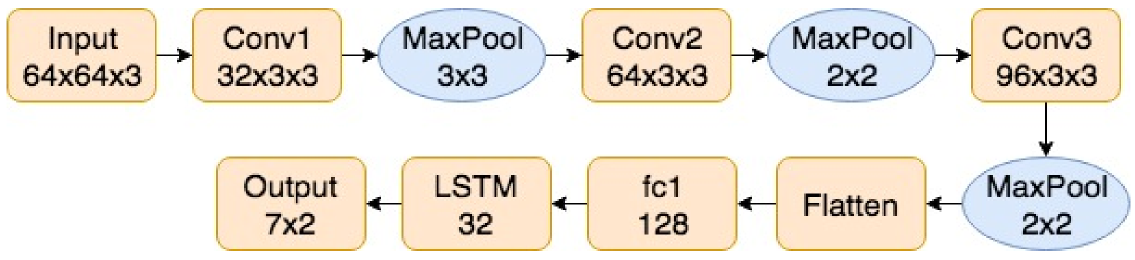

Figure 3: Model Structure

- $i_t = \sigma(W_{hi}h_{t-1} + W_{ii}x_t + b_i)$ is the memory gate.
- $g_t = \tanh(W_{hg}h_{t-1} + W_{ig}x_t + b_g)$ is the intermediate memory content
- $o_t = \sigma(W_{ho}h_{t-1} + W_{io}x_t + b_o)$ is the ouput gate
- $c_t = c_{t-1} \circ f_t + i_t \circ g_t$ is the current accumulative memory
- $h_t = \tanh(c_t) \circ o_t$ is the output of each cell

2 Experiment Settings

1. Preprocessing:
   Before feeding the data into the model, I first normalized the original images to remove the interfere over the original images from different context. The normalization equations follow the instructions:

$$data = data - np.mean(data, axis = (2, 3, 4), keepdims = True) \tag{1}$$
$$data = data/np.std(data, axis = (2, 3, 4), keepdims = True) \tag{2}$$

2. Validation:
   I split the data into two parts, one for the training and the other for validation. The training size is 7000 sequencies and the validation size is 1000 sequencies. The 1000 validation data is picked from the whole data with a fixed interval.

3. Model Structure:
   The model consists of two parts: CNN which is used to extract some features of each image and RNN which is used to improve the accuracy of the predictions. The hyperparameters are shown in Fig.(3).

4. Initializer:
   For CNN kernals, I used *he_normal* initializer for all three layers. *he_normal* initializer is

designed to be used with ReLU activation function. For CNN bias, zero initializers are applied. For the output layer of RNN, I initialized the bias to be zeros and the weight by a Gaussian distribution with the mean of 0 and the standard deviation of 0.001.

5. Batch Size:
   As recommended in the instruction, I set 5 for batch size. Actually, I tried many different batch size, ranging from 5 to 50. Regardless of the training speed, I found that a batch size of 5 generally leads to a quicker convergence, though theoretically it should be the opposite. Hence, I used a batch size of 5.

6. Loss Function:
   I use average pixel distance error as my loss. It is defined as

$$err = \sqrt{(x_i - x_{gt})^2 + (y_i - y_{gt})^2} \tag{3}$$

7. Optimizer and Learning Rate:
   AdamOptimizer is used in this assignment. It is also recommended in the instruction. To have a faster traininig speed, I adjusted the AdamOptimizer's parameters, setting the learning rate 0.002 (the default is 0.001) and beta2 0.99 (the default is 0.999).

3 Result

1. Speed:
   For this assignment, the speed is much slower than all the previous programming assignments, because I used a model more complicated than ever. It took around 70 seconds for each 200 iterations, which is equivalent to $\frac{1}{7}$ of an epoch.

   The training converges very fast. After 1600 irations or so, the algorithm converges to 9.2 distance error in pixel. The speed is affected by the batch size as well as the total number of parameters of the model.

2. Accuracy:
   Since this is a regression problem rather than a classification problem, there shouldn't be an accurate rate. But to supervise the quality of the training and predictions, I defined another measurement in addition to training loss.

   The accuracy is measuered by computing the percentage of predictions that locate within a 15-pixel range in terms of the ground truth, as Fig.(5). The accuracy remains unchanged since it reached 85. From Fig.(5), we can see that the overall predictions are getting closer to the ground truth.

   Fig.(6) shows the quality of the predictions for each landmarks. This is required in the instruction. From the Fig.(6), we can see that the head and shoulders can be predicted most accurately, which also matches our intuitions that head and shoulders are easier to predict due to their big size and less geometrical variations.

3. Loss Error:
   The loss (Fig.4) also seemingly makes sense. Each number on the plot is the average distance error in pixels. The optimal loss stablized at around 9.2.

4. Imposition of Predictions:
   Though not required, I am interested in knowing that how well explicitly my predictions are. Fig.(7) shows that the predictions are not as good as I expected. The reasons might be that, since there are only around $40\% - 60\%$ percentage of predictions within 10-pixel range, it is highly possible that I just randomly picked a sample image with poor predictions.

5. Summary:
   After tens of trials with different combination of parameters, *e.g.*, different models, learning rate and batch size, the final loss always stays at around 9.1 or 9.2. It doesn't make many difference to use a larger and more complicated model. It is highly because this task is relatively simple comparing to a task with much larger datatset.

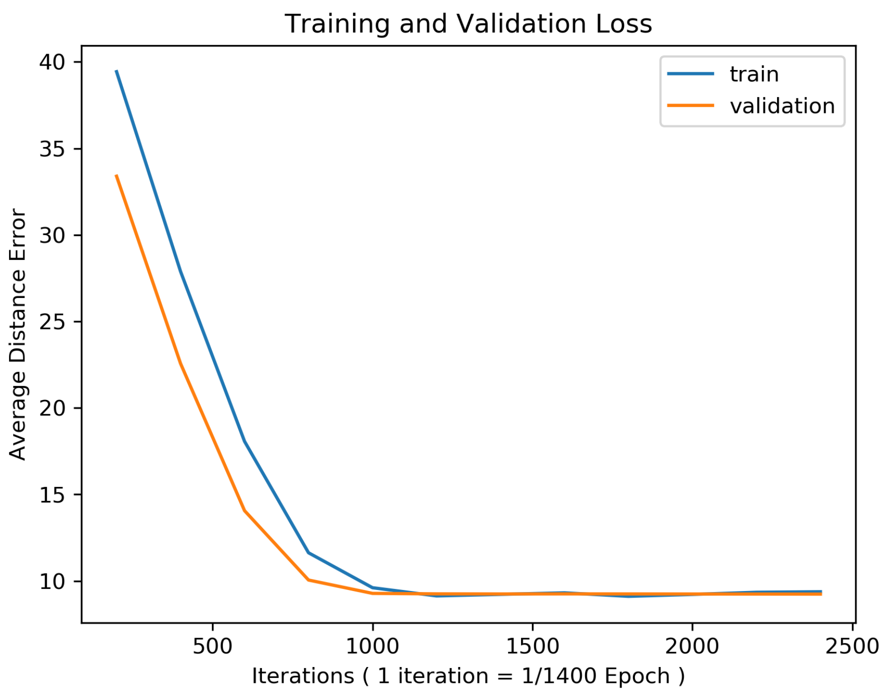

Figure 4: The Loss of training and validtaion

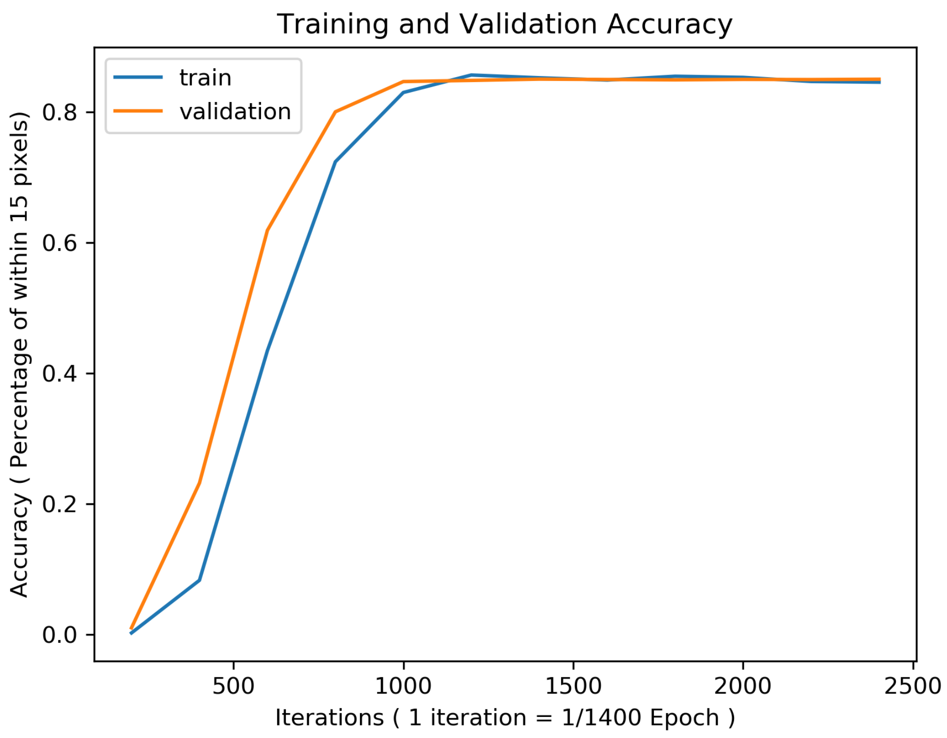

Figure 5: The Accuracy of training and validtaion

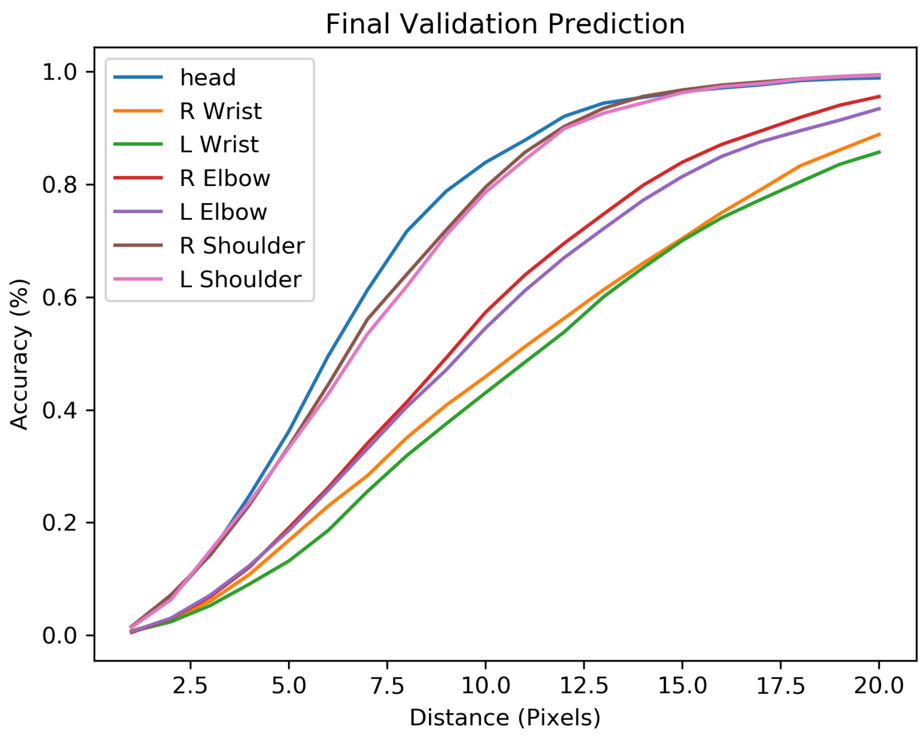

Figure 6: The accuracy within a certain area

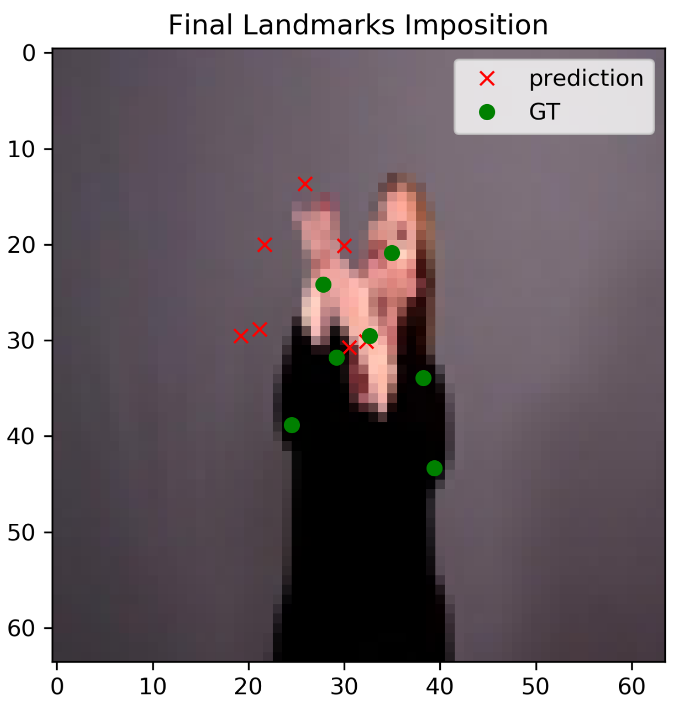

Figure 7: Imposition