Assignment for Deep Learning Course Day 3

Weifan Zhang

Note:

> **The moodel requires *Maximum file size: 100MB*, but the data set is too big, so in my zip file, the folder "data" is empty.**

1 Assignment: Recurrent Neural Networks

In this assignment, I implement a Recurrent Neural Network (RNN) that solves the speech classification task. I use the Python library "torch" (known as "PyTorch") to make the neural network.

1.1 Model

My RNN has these layers or operations in order as follows:

1. Gated Recurrent Unit (GRU) layer with input size 40, hidden size 64.
2. GRU layer with input size 64, hidden size 64.
3. Only keep the last time frame of data and discard other time frames. The details are described in Section 1.2.2.
4. Fully-connected linear layer with 64 input features and 128 output features.
5. Activation function rectified linear unit (ReLU).
6. Fully-connected linear layer with 128 input features and 128 output features.
7. Activation function ReLU.
8. Output linear layer with 128 input features and 3 output features (corresponding to 3 classes).
9. Activation function softmax.

1.2 Data

1.2.1 Input data

The function *data.shape* in Python gets $(n\_sample, 101, 40)$, which means that the data in each sample is a 101×40 matrix. The input layer in our model is a GRU layer with the parameter $batch\_first = True$. This parameter is to tell the GRU layer, the first dimension (101) of the input data is the time frame and the second dimension (40) is the true data, which means in each sample of data, there are 101 time frames, and the data at each time frame has 40 features.

1.2.2 Data operation in the middle steps

Because the second GRU layer has 64-dimensional hidden states, the output data of the second GRU layer is with the shape $(n\_sample, 101, 64)$, which means each sample of data has the shape $(101, 64)$. However, the fully-connected layer requires that the input data should be a vector instead of a matrix, so before passing it to the first fully-connected layer, we need to transform the output data of the second GRU layer into a vector. There are more than one methods to do it:

a We can flatten the data using $data.view(-1, 101 * 64)$, and set the input feature number of the first fully-connected layer as $101 * 64$.

b We can use $data = torch.mean(data, dim = 1)$ to calculate the calculate average values along the second dimension (time frame) to eliminate this dimension. With this method, the input feature number of the first fully-connected layer should be 64.

c We can use $data = data[:, -1, :]$ to choose the data at the last time frame to eliminate this dimension. With this method, the input feature number of the first fully-connected layer should be 64.

As required by the assignment, I use the method c.

1.3 Training, Validation, Testing, and Others

In our program, we use "torch.manual_seed(0)" to make our results reproducible, train our model for 50 epochs, use the data minibatch size of 1024 samples, shuffle the data, normalize the speech feature matrices by $X\_train = (X\_train – np.mean(X\_train))/np.std(X\_train)$, choose cross-entropy as the training loss function, and use Adam with default parameters as the optimizer.

1.4 Number of Parameters in My Model

1.4.1 Method of Calculation

We can use the following principles to calculate the number of parameters in an RNN.

1. The "number of parameters in a neural network" is calculated by the *sum* of "the number of parameters in every layer".
2. A linear layer with m inputs and n outputs has $(m + 1) \times n$ parameters, because for every output there are m **weight** parameters and 1 **bias** parameter.
3. A GRU layer with the input size m, and hidden size n has $(m + n + 2) \times 3 \times n$ parameters.

We have discussed item 1 and 2 in a previous assignment. Then, we discuss item 3.

According to this link[1], we define the symbols in Table 1, and h_t, the hidden state at the time frame t, can be calculated by (1), (2), (3), and (4).

$$h_t = (1 - z_t) \odot h_{t-1} + z_t \odot \tilde{h}_t \tag{1}$$

$$\tilde{h}_t = g(W_h x_t + b_{hW} + U_h(r_t \odot h_{t-1}) + b_{hU}) \tag{2}$$

[1] https://stats.stackexchange.com/questions/328926/how-many-parameters-are-in-a-gated-recurrent-unit-gru-recurrent-neural-network

Table 1: The symbols used to calculate the number of parameters in a GRU layer.

Symbol	**Description**
m	input size of this GRU layer
n	The size of hidden states in this GRU layer
x_t	The input data at the time frame t, which is an $m \times 1$ vector
h_t	The hidden state at the time frame t, which is an $n \times 1$ vector
h_{t-1}	The hidden state at the time frame $t-1$, which is an $n \times 1$ vector
$\tilde{h}_t$	The candidate hidden state at the time frame t
z_t	The update gate at the time frame t, which is an $n \times 1$ vector
r_t	The reset gate at the time frame t, which is an $n \times 1$ vector
W	Weight matrix with $n \times m$ elements
U	Weight matrix with $n \times n$ elements
b	Bias, which is an $n \times 1$ vector
g, σ	Functions with no parameters that need training

Table 2: GRU layers in my model.

name	input size	hidden state size
GRU layer 1	40	64
GRU layer 2	64	64

$$z_t = \sigma(W_z x_t + b_{zW} + U_z h_{t-1} + b_{zU}) \tag{3}$$

$$r_t = \sigma(W_r x_t + b_{rW} + U_r h_{t-1} + b_{rU}) \tag{4}$$

(1) does not have parameters. As W is an $n \times m$ matrix, U is an $n \times n$ matrix, and b is an $n \times 1$ vector, the number of parameters in each of (2), (3), or (4) is $(n \times m + n \times 1 + n \times n + n \times 1)$, so the total number of parameters in (1), (2), (3), and (4) is $(n \times m + n \times 1 + n \times n + n \times 1) \times 3 = (m + n + 2) \times 3 \times n$, which are the parameters for the time frame t. We should also know that, in a GRU layer, the parameters for every time frame are the same, so the number of parameters in a GRU layer can be calculated by $(m + n + 2) \times 3 \times n$.

1.4.2 Calculation for My Model

The information of all layers in my model is listed in Table 2 and 3. According to the calculation method in Section 1.4.1, we can calculate the number of the parameters in my model by (5). The result 70531 is

Table 3: Linear layers in my model.

name	number of inputs	number of outputs
fully-connected layer 1	64	128
fully-connected layer 2	128	128
output layer	128	3

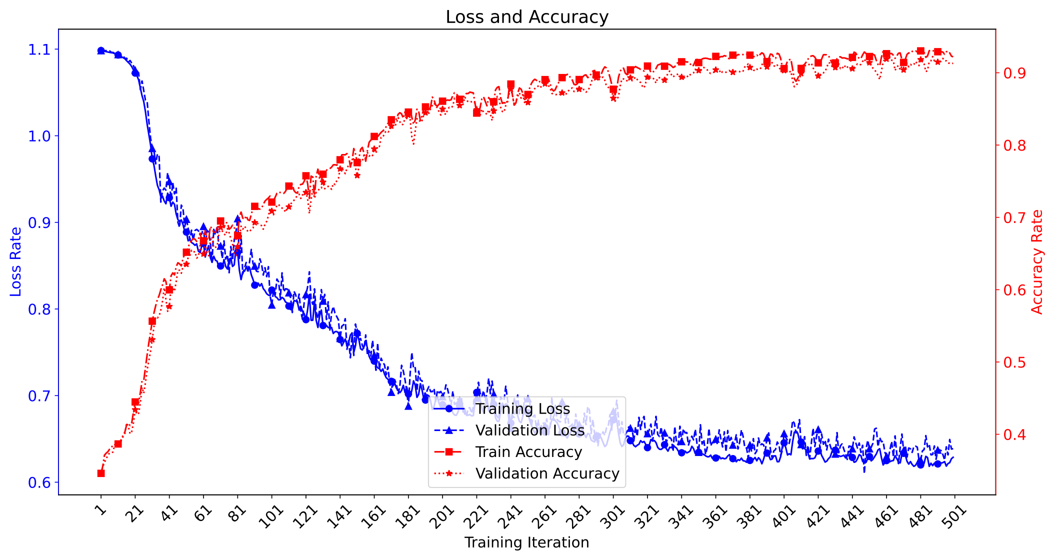

Figure 1: Losses and accuracies.

the same as that calculated by Pytorch.

$$\begin{aligned}
&number\ of\ parameters\ in\ GRU\ layer\ 1:\\
&(40+64+2)\times 3\times 64 = 20352\\
\\
&number\ of\ parameters\ in\ GRU\ layer\ 2:\\
&(64+64+2)\times 3\times 64 = 24960\\
\\
&number\ of\ parameters\ in\ fully-connected\ layer\ 1:\\
&(64+1)\times 128 = 8320\\
\\
&number\ of\ parameters\ in\ fully-connected\ layer\ 2:\\
&(128+1)\times 128 = 16512\\
\\
&number\ of\ parameters\ in\ output\ layer:\\
&(128+1)\times 3 = 387\\
\\
&number\ of\ parameters\ in\ the\ convolutional\ neural\ network:\\
&20352+24960+8320+16512+387 = 70531
\end{aligned} \tag{5}$$

1.5 Result

The test accuracy is 0.9194. The training and validation losses, as well as the training and validation accuracies, as a function of the training iteration, are shown in Figure 1. I think overfitting does not occur in this experiment because there are no obvious gaps between training and validation losses or training and validation accuracies.

Table 4 compares Feedforward Neural Network (FFNN), Convolutional Neural Network (CNN), and RNN in the assignments. From the highest test accuracy, we can know that RNN has the best performance. I

Table 4: RNN vs. CNN vs. FFNN.

neural network type	test accuracy	training time	number of parameters
FFNN with SGD	0.7828	0:04:31.276868	567171
CNN with SGD	0.8340	6:09:55.469010	346067
CNN with Adam	0.9007	6:10:34.715379	346067
RNN with Adam	0.9194	1:16:21.486403	70531

think this is because RNN makes use of the relationship between different time frames of data, but the other two do not do it. Training time can show the computational complexity, and FFNN has the shortest training time, followed by RNN, while CNN needs the longest training time. I think this is because the network structure and computation operations of FFNN are much simpler than the other two networks. RNN's parameters are much fewer than that of CNN, which may be the reason for the gap in training time between the two types of networks.