Assignment for Deep Learning Course Day 2

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: Convolutional Neural Networks

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

1.1 Model and Data

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

1. 2D convolutional layer with 1 input feature, 32 output features, kernel size 5×5, stride 1×1, and padding 0×0.
2. Activation function rectified linear unit (ReLU).
3. 2D max-pooling layer with kernel size 2×2, stride 2×2, and padding 0×0.
4. 2D convolutional layer with 32 input feature, 16 output features, kernel size 5×5, stride 1×1, and padding 0×0.
5. Activation function ReLU.
6. 2D max-pooling layer with kernel size 2×2, stride 2×2, and padding 0×0.
7. Flatten the data into a vector with $16 \times 22 \times 7 = 2464$ elements.
8. Fully-connected linear layer with 2464 input features and 128 output features.
9. Activation function ReLU.
10. Fully-connected linear layer with 128 input features and 128 output features.
11. Activation function ReLU.
12. Output linear layer with 128 input features and 3 output features (corresponding to 3 classes).
13. Activation function softmax.

1.1.1 The size of the input layer

The principle of choosing the first convolutional layer is that:

> If each data sample consists of X Y-dimensional matrices (X: number of matrices, Y: number of dimensions in each matrix), the first convolutional layer should be YD, with X input features.

In this assignment, each data consists of 1 2-dimensional matrix, so the first convolutional layer (input layer) should be 2D, with 1 input feature. In order to make our data fit the 1 input feature, we use data.view(-1, 1, 101, 40) to reshape our data from (n_sample, 101, 40) to (n_sample, 1, 101, 40).

1.1.2 The variable "flattened_size"

The role of the convolutional layers is to reduce the number of features of the data passed to the fully connected layers. In this assignment, each sample of data is a 2D matrix, the convolutional layers along with the max-pooling layers can reduce the size of the data matrices, but before we passed the data to the first fully-connected layer, we should flatten the data into a vector.

To calculate the number of elements in the flattened vector, we should know that for a convolutional layer or a max-pooling layer, if the input data matrices size is $I_x \times I_y$, the output data matrices size is $O_x \times O_y$, the kernel size is $K_x \times K_y$, the stride is $S_x \times S_y$, the padding is $P_x \times P_y$. O_x and O_y can be calculated by (1).

$$\begin{aligned} O_x &= \lfloor \frac{I_x - K_x + 2P_x}{S_x} + 1 \rfloor \\ O_y &= \lfloor \frac{I_y - K_y + 2P_y}{S_y} + 1 \rfloor \end{aligned} \tag{1}$$

In this assignment, the original data matrices size is 101×40, so we can use (1) to calculate that the output matrices size of the first convolutional layer, first max-pooling layer, second convolutional layer, and second max-pooling layer are 97×36, 48×18, 44×14, and 22×7, respectively. The second convolutional layer has 16 output channels, so the output data of the second max-pooling layer consists of 16 22×7 matrices. Hence, the number of elements in the flattened vector is $16 \times 22 \times 7 = 2464$.

1.2 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, normalize the speech feature matrices by $X\_train = (X\_train – np.mean(X\_train))/np.std(X\_train)$, and choose cross-entropy as the training loss function. We use two optimizers: stochastic gradient descent (SGD) with a learning rate of 0.01, a momentum of 0.9, and a weight decay of 0.00001 and Adam with default parameters.

1.3 Result

The test accuracy with optimizer SGD is 0.8340, and the test accuracy with optimizer Adam is 0.9007 better than the former.

For SGD and Adam, 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. When the model only experiences a small number of iterations, Adam performs better than SGD obviously, with much higher accuracies and lower

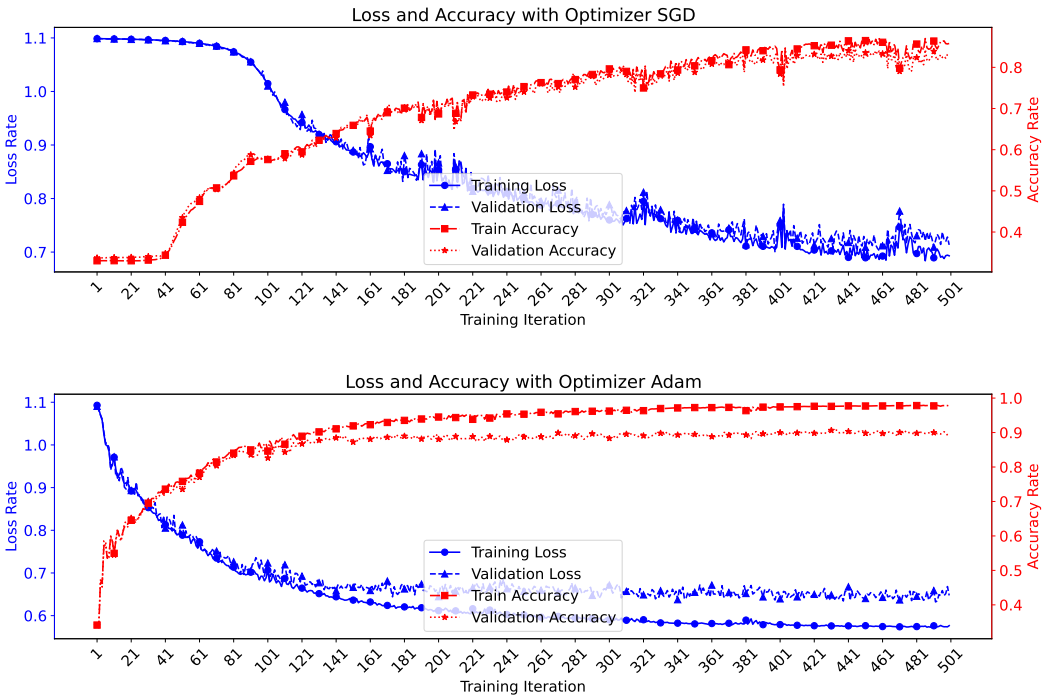

Figure 1: Losses and accuracies.

Table 1: CNN vs FFNN.

neural network type	test accuracy	training time
FFNN with SGD	0.7828	0:04:31.276868
CNN with SGD	0.8340	6:09:55.469010

losses, but as the number of iterations increases, their difference is decreasing, although Adam still has better values at the end of the training. Besides, overfitting does not occur in the SGD experiment, in which the training and validation losses are close as well as the accuracies, while the model with Adam has overfitting, as the training and validation accuracies have a gap of about 0.1, as does the losses.

1.3.1 Comparation with the feedforward neural network

In the previous assignment, we implemented a FFNN with optimizer SGD, so we compare it with the CNN with optimizer SGD, as shown in Table 1. The test accuracy of CNN is 0.05 higher than that of FFNN, but its training time is 6 hours and 10 minutes, compared with only 4.5 minutes of FFNN. Thus, CNN's performance is better but the computational complexity is much higher than FFNN.

Table 2: Convolutional layers in my model.

name	kernel size	number of input channels	number of output channels
convolution layer 1	5×5	1	32
convolution layer 2	5×5	32	16

Table 3: Linear layers in my model.

name	number of inputs	number of outputs
fully-connected layer 1	2464	128
fully-connected layer 2	128	128
output layer	128	3

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 a CNN.

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 convolutional layer with the kernel size $K_x \times K_y$, C_{in} input channels, and C_{out} output channels has $K_x \times K_y \times C_{in} \times C_{out} + C_{out}$ parameters.
4. Max-pooling layers do not have parameters that need learning.

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

We should know that **each output channel of a convolutional layer has a kernel, an output, and a bias**. We have a convolutional layer with C_{in} input channels and C_{out} output channels. For the $No.j$ output channel, its kernel is ke_j, its output is O_j, and its bias is B_j. We use ch_l to represent the $No.l$ input channel. The weight of ch_l in ke_j is W_{jl}. O_j can be calculated by (2).

$$O_j = \sum_{l=1}^{C_{in}} ke_j \times ch_l \times W_{jl} + B_j \tag{2}$$

In (2), O_j and ch_l are variables, B_j is a parameter, and for every $1 \leq l \leq C_{in}$, W_{jl} is a parameter. We define the kernel size of this convolutional layer as $K_x \times K_y$, so ke_j consists of $K_x \times K_y$ parameters. Therefore, (2) has $K_x \times K_y \times C_{in} + 1$ parameters, and for every output channel of this convolutional layer, there is an equation like (2), so this convolutional layer has
$(K_x \times K_y \times C_{in} + 1) \times C_{out} = K_x \times K_y \times C_{in} \times C_{out} + C_{out}$ parameters.

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 (3).

$$
\begin{aligned}
&number\ of\ parameters\ in\ convolution\ layer\ 1: \\
&5 \times 5 \times 1 \times 32 + 32 = 832 \\
&number\ of\ parameters\ in\ convolution\ layer\ 2: \\
&5 \times 5 \times 32 \times 16 + 16 = 12816 \\
&number\ of\ parameters\ in\ fully-connected\ layer\ 1: \\
&(2464 + 1) \times 128 = 315520 \\
&number\ of\ parameters\ in\ fully-connected\ layer\ 2: \\
&(128 + 1) \times 128 = 16512 \\
&number\ of\ parameters\ in\ output\ layer: \\
&(128 + 1) \times 3 = 387 \\
&number\ of\ parameters\ in\ the\ convolutional\ neural\ network: \\
&832 + 12816 + 315520 + 16512 + 387 = 346067
\end{aligned}
\tag{3}
$$