Assignment for Deep Learning Course Day 1

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: Feedforward Neural Networks

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

1.1 Model

My neural network has 1 input layer, 4 hidden layers (required by the assignment), and 1 output layer. The input layer has 101*40 inputs and 128 outputs. Every hidden layer has 128 inputs and 128 outputs. The output layer has 128 inputs and 3 outputs. Therefore, the input data of my neural network have 101*40 dimensions and the output data have 3 dimensions.

- The value 128 is required by the assignment.
- The value 101*40 is because the shape of every sample in our data is 101*40, which means it has 101*40 features. The 101*40 dimensions of the input data map the 101*40 features.
- The value 3 is because the data have 3 labels (classes), "yes", "no", and "other words". The 3 dimensions of the output data map the 3 different labels.

The input layer does not have an activation function because the purpose of the input layer is to simply receive the input data and pass it on to the next layer. As required by the assignment, all hidden layers use the activation function rectified linear unit (ReLU). The output layer uses the activation function softmax (I will explain the reason later). We use stochastic gradient descent (SGD) with a learning rate of 0.01, a momentum of 0.9, and a weight decay of 0.00001 as the optimizer, which is required by the assignment. We choose cross-entropy as the training loss function.

The reason why I choose softmax as the output layer activation function and cross-entropy as the training loss function is that, ChatGPT tells me the following principle:

> For binary classification problems, sigmoid is usually used as the output layer activation function, and binary cross-entropy is used as the training loss function.
>
> For multiclass classification problems, softmax is typically used as the output layer activation function, and cross-entropy is used as the training loss function.

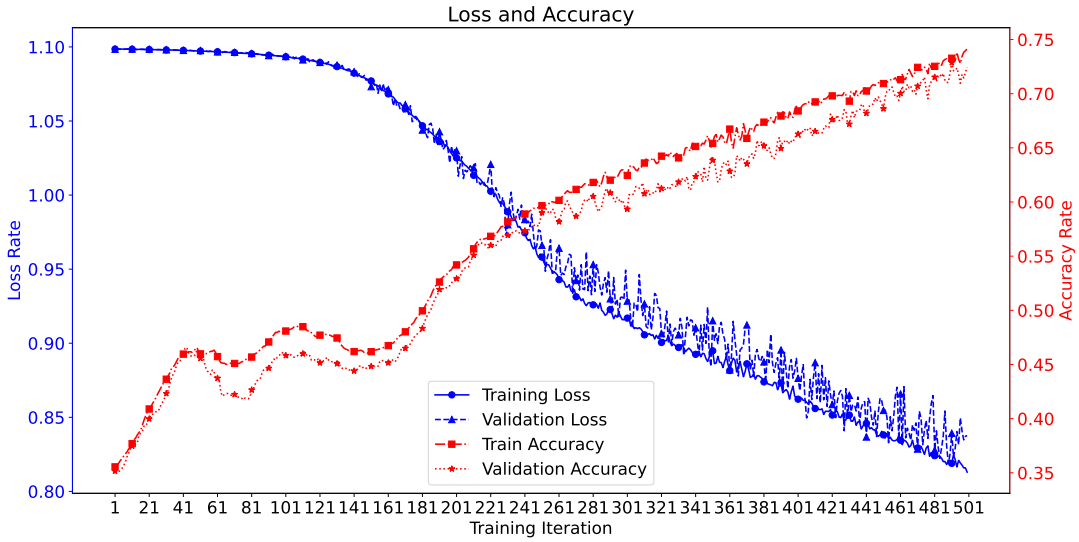

Figure 1: Losses and accuracies.

> For regression problems, no output layer activation function is typically used, and an appropriate training loss function is chosen depending on the specific scenario, such as mean squared error (MSE) or mean absolute error (MAE).

This assignment is to classify data into 3 classes, which is a multiclass classification problem, so I choose the output layer activation function and training loss function.

1.2 Data and Others

After getting data from the "pickle" files, I use "data.view(-1, 101 * 40)" to flatten the data from shape (n_samples, 101, 40) to shape (n_samples, 101 * 40). I use the tools *from torch.utils.data import DataLoader, TensorDataset* to preprocess all data, setting the minibatch size as 1024 samples and shuffling the training data and validation data (required by the assignment).

Also following the instruction of the assignment, I train my model for 50 epochs, use "torch.manual_seed(0)" to fix the random seed to make the results reproducible, and normalize the speech feature matrices by $X\_train = (X\_train – np.mean(X\_train))/np.std(X\_train)$.

1.3 Result

The test accuracy is 0.6941.

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 the overfitting has not occurred, because there are no big differences between training and validation losses or between training and validation accuracies. If overfitting occurs, I think I can adjust the parameters in the SGD optimizer, reducing the learning rate and the momentum, and increasing the weight decay.

Table 1: Layers in my model.

layer type	number of this type of layers	number of inputs in each layer	number of outputs in each layer
input	1	101*40	128
hidden	4	128	128
output	1	128	3

1.4 Total Number of Parameters

1.4.1 Method of Calculation

If we have a layer with 3 inputs x_1, x_2, and x_3, and 2 outputs y_1 and y_2, because every layer in a neural network is linear, this layer can certainly be represented as (1):

$$\begin{aligned} y_1 &= a_{11} \times x_1 + a_{21} \times x_2 + a_{31} \times x_3 + b_1 \\ y_2 &= a_{12} \times x_1 + a_{22} \times x_2 + a_{32} \times x_3 + b_2 \end{aligned} \tag{1}$$

In the above example, a_{ij} is the **weight** and b_j is the **bias**. From this example, we can understand the method to calculate the number of parameters in a neural network:

1. The "number of parameters in a neural network" is calculated by the *sum* of "the number of parameters in every layer".
2. A 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.

1.4.2 Calculation for My Model

All layers in my model are listed in Table 1. According to the calculation method in Section 1.4.1, we can calculate the number of the parameters in my model by (2).

$$\begin{aligned} &number\ of\ parameters\ in\ the\ input\ layer = (101 \times 40 + 1) \times 128 = 517248 \\ &number\ of\ parameters\ in\ hidden\ layers = (128 + 1) \times 128 \times 4 = 66048 \\ &number\ of\ parameters\ in\ the\ output\ layer = (128 + 1) \times 3 = 387 \\ &number\ of\ parameters\ in\ the\ neural\ network = 517248 + 66048 + 387 = 583683 \end{aligned} \tag{2}$$