Assignment 2 , 159.740, 2020 S2

Family Name: JIANG , Given Name: TAO ,

Student ID:20004769

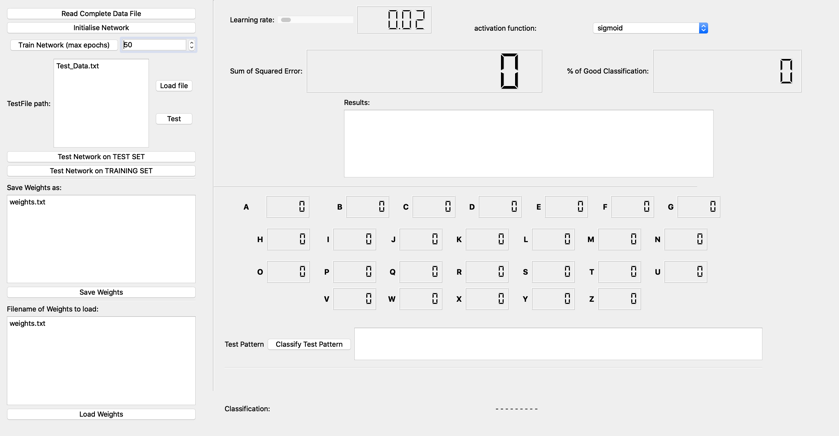
**Ass2 Letter Recognition using Deep Neural Nets with Softmax Units**

The experiment uses a deep neural network to classifier the dataset of **Letter+Recognition** in the UCI.

* **User’s Guide**

**Environment**

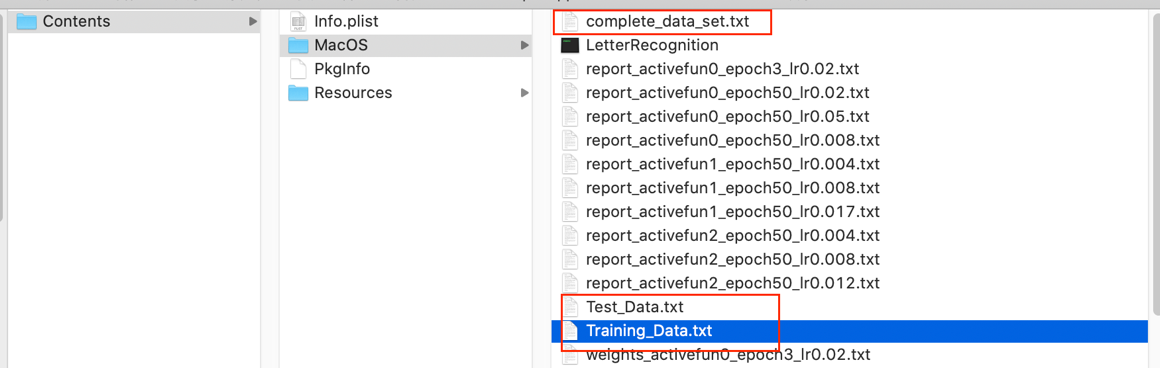
I use macOS as a development platform. And use *qmake* to compile the project in the Qt5.1 integrated environment.



**Preparation**

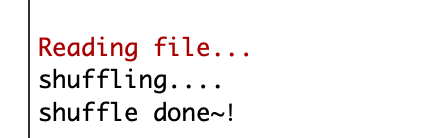
Compile the project first and copy the dataset files to :

*./build-LetterRecognition-Desktop\_Qt\_5\_15\_1\_clang\_64bit-Debug/LetterRecognition.app/Contents/MacOS/*

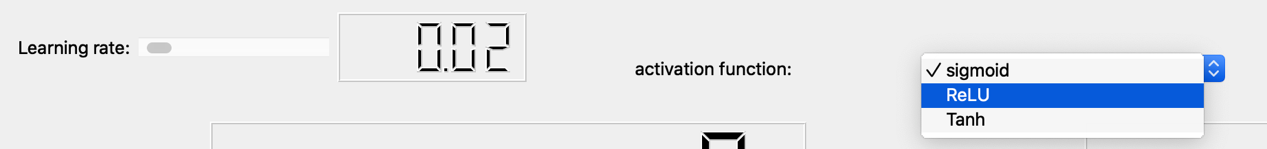


**Training**

Execute the program and click the button of **Read Complete Data File** to read the file of *complete\_data\_set.txt.* The program will automatically perform random shuffle operations for the dataset.

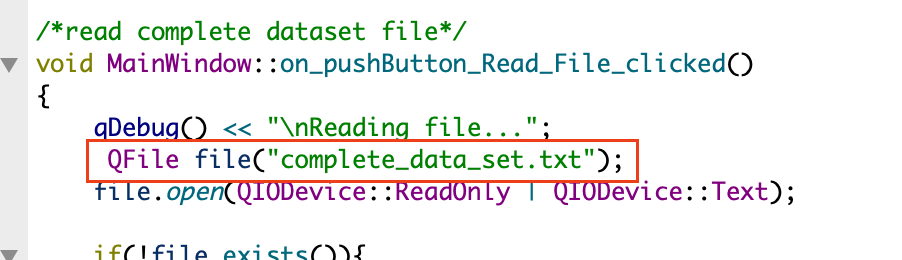


Next, click the button of **Initialize Network** to initialize the network and weight. After that, adjust the required parameter **Learning rate** and the **Activation function** of the model.



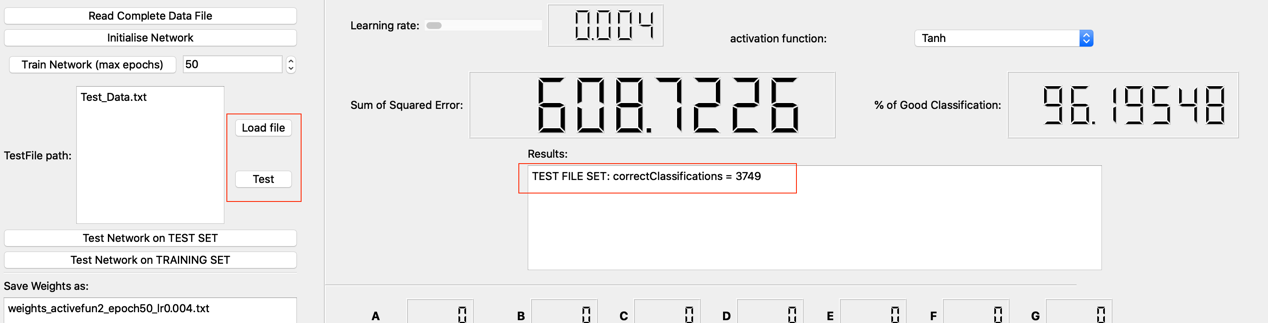
The last selected the size of **Max epochs** and click the button of **Train Network**. And then the Network will training.

Changing the training dataset requires changing the path of the data set.



**Testing**

The program implements the function of verifying a separate test dataset. Enter test dataset file name and click the button of **Load file**. Click the **Test** button will verify the model by test dataset as below.

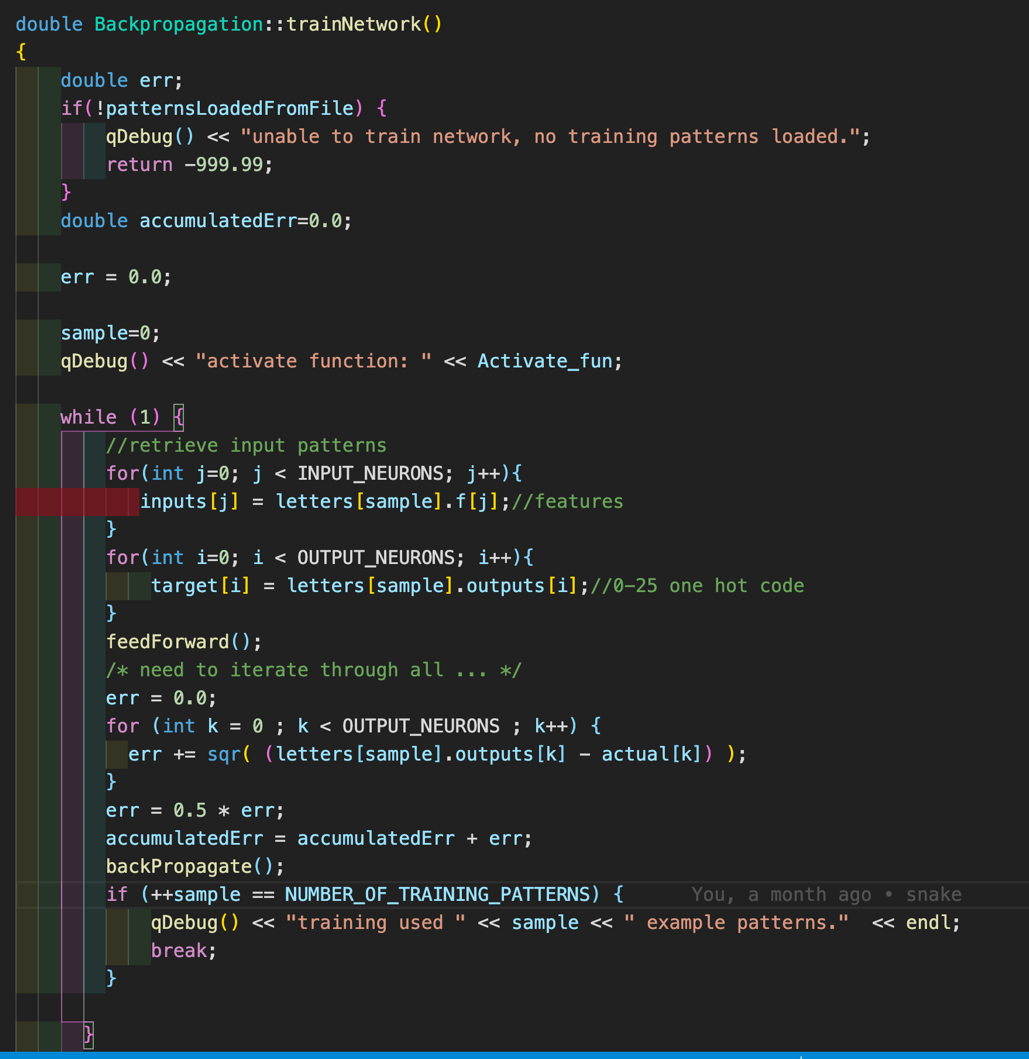


**Save and load weight**

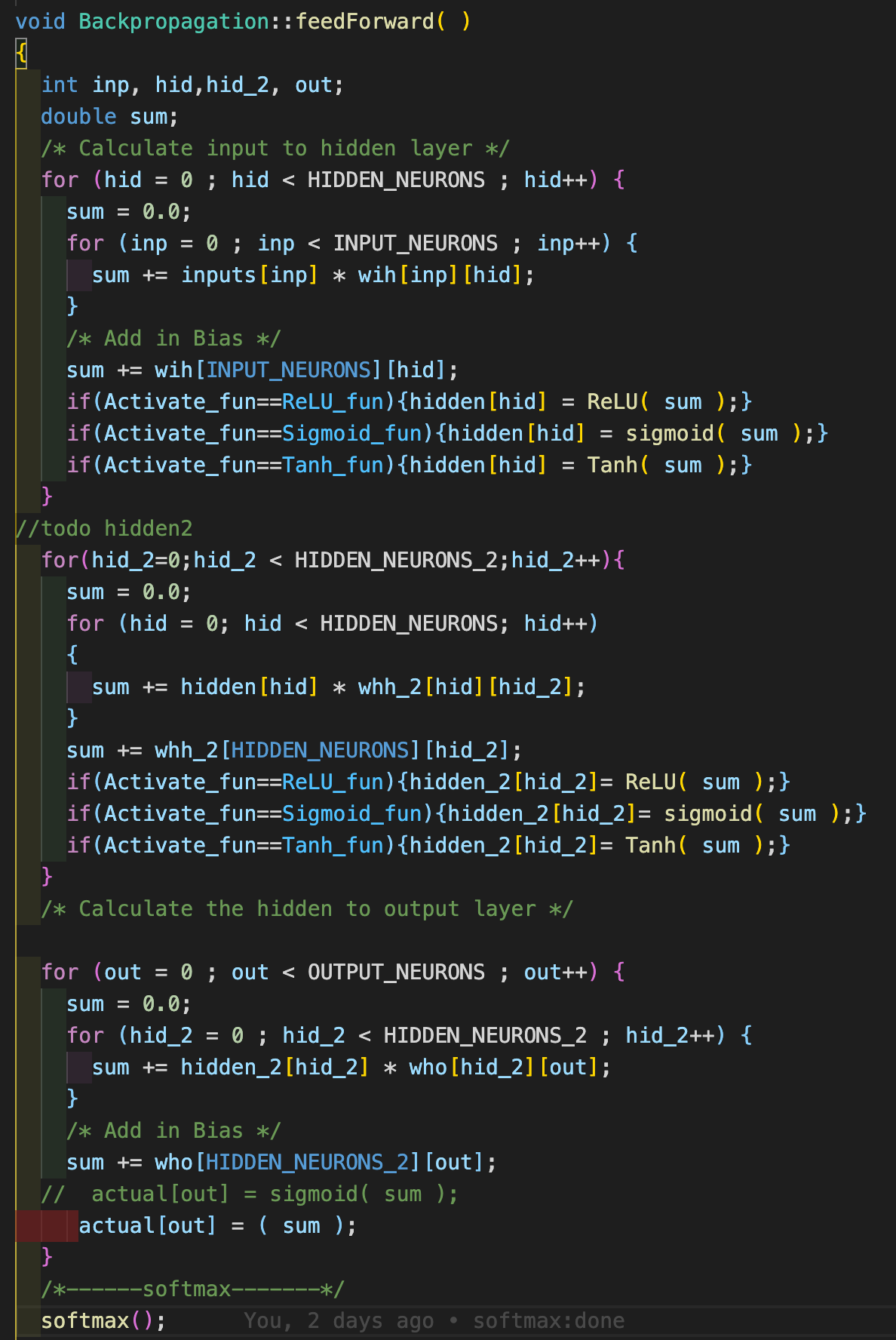
Save and load weight functions implement properly as well. The files will save at *./build-LetterRecognition-Desktop\_Qt\_5\_15\_1\_clang\_64bit-Debug/LetterRecognition.app/Contents/MacOS/* . At the same time, Each the model of epoch SSE and MSE will record.

* **pseudo code**

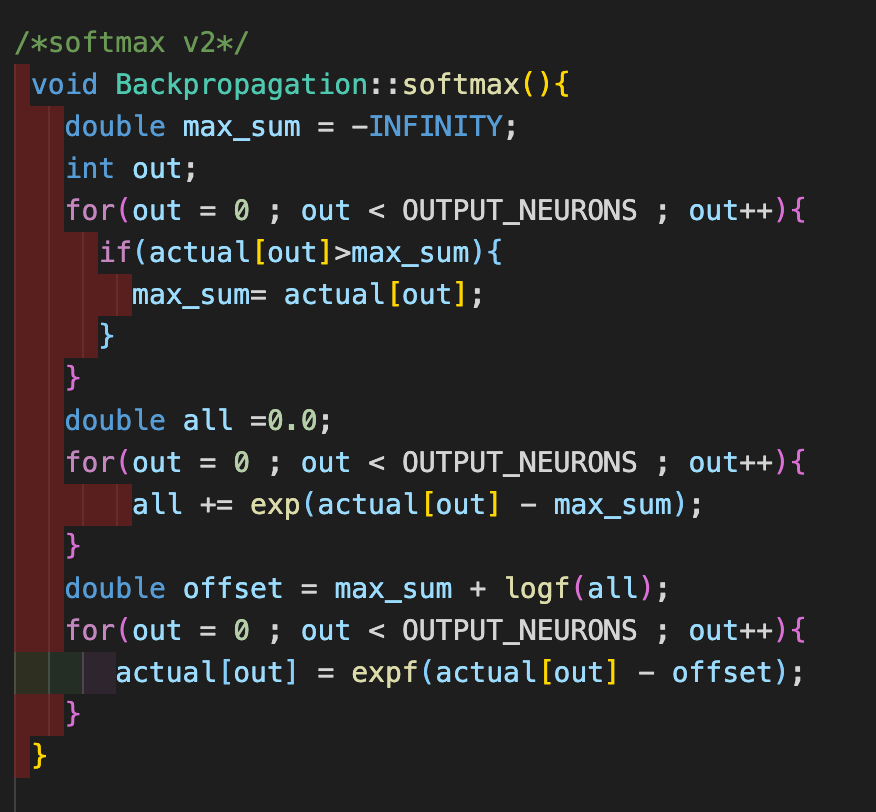
In the function *trainNetwork*(), feedforward and backpropagation are performed for each sample.



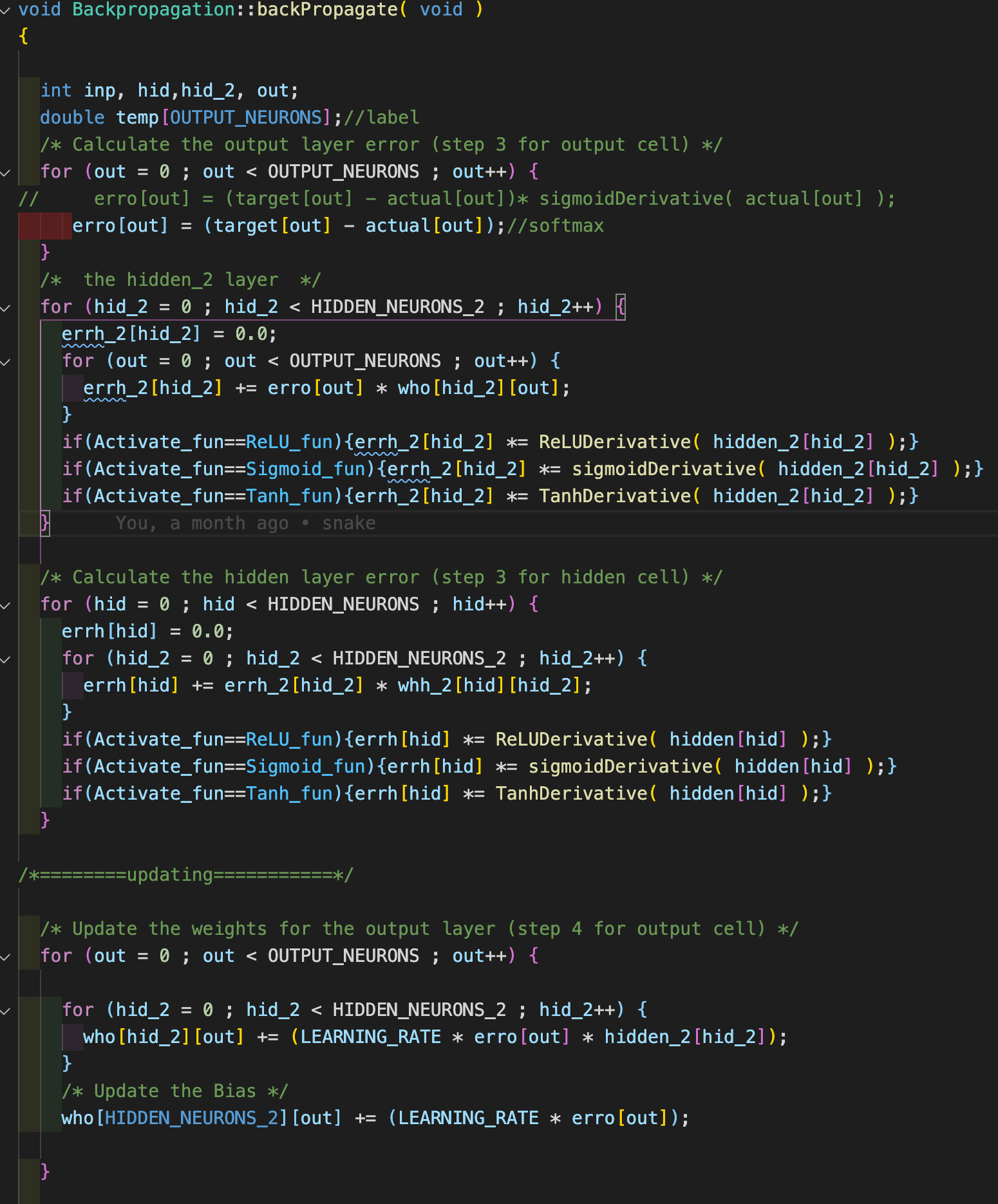
In feedforward, each layer uses full connections and uses activation functions for nonlinear activation. The final output layer uses the *softmax()* activation function.



I didn't use the traditional SoftMax activation function while optimized it here. The offset was added while normalizing so that the model finally converged faster.



The *backPropagate ()* function calculates the loss, and the weight is updated according to the loss.



* **Experiment results**

The experiment uses different activation functions and learning rates to train and verify the model. The input size of the model is a vector of 16, and the output layer is 26 neurons and passes through the SoftMax classifier. Among them, the model uses 2 hidden layers, each with 128 and 256 neurons connected. I trained the models by using different learning rates and activation functions. Get the three best performing models which use randomly initialized weight parameters. The results are as follows:



Here, I only show the data analysis of the best model. For more data analysis results, please refer to the **data.xlsx** file.

The best-performing model currently uses the sigmoid activation function, the learning rate is 0.008, and the model after 50 iterations. The figure below shows the convergence of the model on the training set.

The figure below shows the final accuracy comparison between the training set and the test set of the model.

It can be seen from the confusion matrix by test dataset in the figure below that most characters can be correctly recognized. Among them, 6 characters G are recognized as C, and 6 of H characters are recognized as R. There are 5 I and J identification confusions. It makes sense that those handwritten characters are indeed easy to recognize errors in daily life.



* **Conclusion**

Through the above experiment, it can be seen that using different nonlinear activation functions and learning rates can get different results. The sigmoid activation function can get a relatively good model under the premise of a larger learning rate. If the learning rate becomes smaller, the model can learn more details of the feature value, which results in a model with higher accuracy. At the same time, if the learning rate is too low, there is a risk of model overfitting. Using more neurons allows the model to learn more detailed features, which makes the model more accurate.

After the data is loaded, random shuffle operations are performed on the data. This operation uses a timestamp as a random seed, which results in slightly different results for each training and testing under the same hyperparameter.