Assignment2 Neural Network

Zihe Liu

1831581

1 Introduction

We implement the Neural Network based on Python to solve multi-classification problems, and train and evaluate model on two data sets including iris and abalone.

2 Environment

The version of Python requires as least 3.5. Run the following command in the shell to install third party dependencies.

```
$ pip install -r requirements
```

To train the Neural Network, we can run the command "python train.py" with the arguments in the Table 1. For example, the following command is to train Neural Network on iris data set with specific hyper parameters.

```
$ python train.py \
    --prefix=static/input/iris/iris \
    --hidden_sizes=32 \
    --learning_rate=0.01 \
    --batch_size=8 \
    --activator=tanh
```

The following command is to train Neural Network on abalone data set with specific hyper parameters.

```
$ python train.py \
    --prefix=static/input/abalone/abalone \
    --hidden_sizes=32,64,128 \
    --learning_rate=0.01 \
    --batch_size=8 \
    --activator=tanh
```

3 Data Set and Data Preprocessing

We choose two data sets to train Neural Network, including iris and abalone.The iris data set has 150 instances, 4 features and 3 categories. And the abalone data set has 4117 instances, 8 features, and 29 categories.

Reshuffle data set. For each data set, we reshuffle the data set so that the model can learn more general features from it.

Combine different categories. As for abalone data set, there are 29 classifications and the histogram on classification and its number of instances is show as Figure 1. We can see four ages including 7, 8, 9 and 10 are far higher than others. It is so imbalance between different classification that the neural network can not well classify them. So we combine different categories to three categories: lower than 9, 9 and 10, bigger than 10.

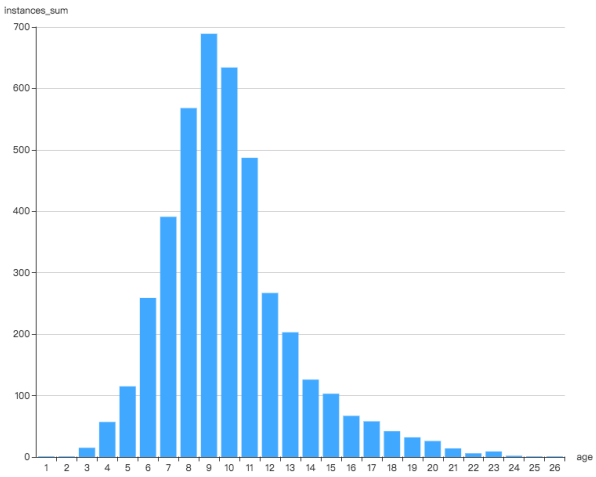

Figure 1: The histogram on classification and its number of instances on abalone data set.

Split train set, validation set and test set. And we divide data set into train set, validation set and test set by 80%, 10% and 10%. As the model optimizes the learnable parameters on the training set, the loss value on the training set is gradually reduced. But when the number iterations is too big, the model will learn the features specific to this training set but not universal features. As a result, the model is overfitting and accuracy decreases.

So we use training set and validation set to train model and decide whether to early stop. At each iteration, we firstly use training data set to optimize the model with SGD algorithm and respectively calculate average loss values of training data and validation data. If the average loss values of training data or validation data increases 5 times, which means loss value on validation data set has converged and will be overfitting if continue training, we early stop the training.

4 Program Modules

There are 8 parts in out program, including *requirements.txt*, *model/*, *preprocess/*, *static/*, *utils/*, *constants.py*, *data.py* and *train.py*.

Table 1: The arguments for train.py

Arguments	Introduction	Type
prefix	The prefix of data set path .	str, 'static/input/iris/iris' or 'static/input/abalone/abalone'
hidden_sizes	The hidden size of each hidden layer.	multiple numbers spitted by ','. Eg: '12,12,12'
batch_size	The size of batch.	int, default is 8
learning_rate	The learning rate	float, default is 0.01
activator	The activator of hidden size	'sigmoid' or 'tanh'

4.1 requirements.txt

The content of *requirements.txt* is the third-party dependencies and their version. This file is generated by the following command.

```
$ pip freeze > requirements
```

4.2 preprocess/

The module *preprocess/* is to preprocess the raw data set into fixed formal file. Firstly it reshuffle the data set and then divide the raw data set into train, validation and test set by 80%, 10% and 10%.

4.3 data.py

data.py is to read the preprocessed data set and store *train_source_data*, *train_target_data*, *eval_source_data*, *eval_target_data*, *test_source_data* and *test_eval_data* in the instance of Class *Data*. Each instance of target data is an one hot vector full of zero except the position whose index is the relevant category. For example, if the category of a data instance is 2 with 4 categories in all instances, then the target data of this instance is [0, 0, 1, 0].

What's more, the class *Data* also provide methods to get batch data of train, validation and test sequentially. And there are also some methods to provide the common values such as *input_size*, *output_size* and *train_batches_sum*.

4.4 constants.py

constants.py stores the constant values used in project, including paths static files and suffix of fixed formal data set file.

4.5 utils/

utils/ has various tool methods. *file_utils.py* is to transform between file and string, transform between JSON file and dict, and make directory if it does not exist. *log_utils.py* is to output logs to the console and file. And *shell_args.py* is to define the formate of arguments used in shell.

4.6 model/

The module *model/* defines the Neural Network model, with three parts including *activator/*, *fully_connected_layer.py* and *neural_network.py*.

All calculations in the model use matrix operations by $numpy$ instead of loops in $python$. Because matrix operations by $numpy$ underlying layer is implemented in C++, which is much faster than $python$.

4.6.1 Neural Network Model

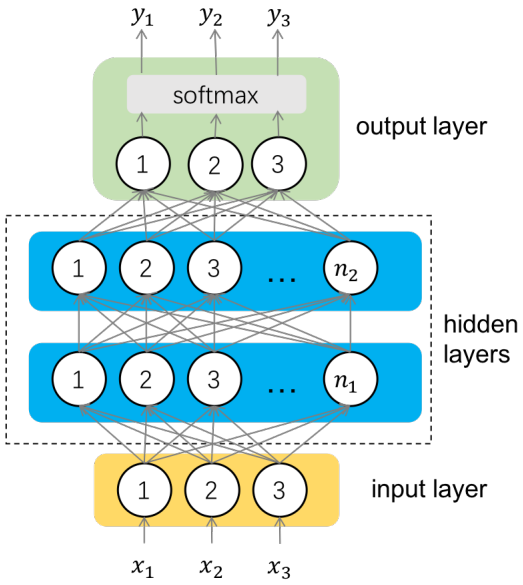

Figure 2: The overall structure of full connected Neural Network.

We implement the full connected Neural Network shown as Figure 2. Neural networks are actually multiple neurons connected according to certain rules. There are one input layer, output layer, and multiple hidden layers between them.

The number of neurons in input layer is the number of features in source data. The number of neurons in output layer is the number of categories in target data. Each output of neuron in output layer represents the probability of getting this category based on the input features, which is between 0 and 1 and the sum is equal to 1.

For the i-th layer, the output vector is calculated by the following equation according to the input vector:

$$\boldsymbol{y_i} = f(\boldsymbol{x_i}\boldsymbol{w_i} + \boldsymbol{b_i}) \tag{1}$$

where $\boldsymbol{y_i}$ is the output vector, $\boldsymbol{x_i}$ is the input vector, $\boldsymbol{w_i}$ and

$\boldsymbol{b_i}$ is the learnable parameters, and f is the activator function. The activator in output layer is $softmax$ as follows:

$$softmax(z_k) = \frac{e^{z_k}}{\sum_{i=1}^{n} e^{z_i}} \tag{2}$$

where n is the output size. Except output layer, the activator in each layer is $sigmoid$ or $tanh$ as follows.

$$sigmoid(z) = \frac{1}{1 + e^{-z}} \tag{3}$$

$$tanh(x) = \frac{e^{x} - e^{-x}}{e^{x} + e^{-x}} \tag{4}$$

The loss function is the cross entropy calculated by the negative log likelihood:

$$Loss = \frac{1}{m} \sum_{i=1}^{m} -t_i \log y_i \tag{5}$$

where m is the number of categories, t_i is the target value and y_i is the predicted value for the category i.

4.6.2 The Module activator/

There is a abstract class $Activator$ with two abstract methods: $forward$ which is used to calculate output by the input, $backward$ which is used to calculate the derivative of the activator according to the output calculated by the $forward$. And the subclass $SoftmaxActivator$, $SigmoidActivator$, $TanhActivator$ respectively implement their respective $backward$ and $forward$.

4.6.3 The Module fully_connected_layer.py

The class $FullyConnectedLayer$ is represented a layer in Neural Network. It stores and updates the learnable vector $\boldsymbol{weights}$ and $\boldsymbol{bia}$. It also provides how to initial the learnable vector such as full of zero, random uniform and random normal.

It provides methods $forward$ to calculate the output vector according to input, weights, bia and activator, and $backward$ to calculate the gradient delta according to the gradient of previous layer.

4.6.4 The Module neural_network.py

The class NeuralNetwork define the structure of the model, including the input size, output size, number of hidden layer, the size of each hidden layer and the activator of each layer.

It also provides methods to train model with batch source data and relevant target data, predict category by the source data, and calculate the loss value according to source data and target data.

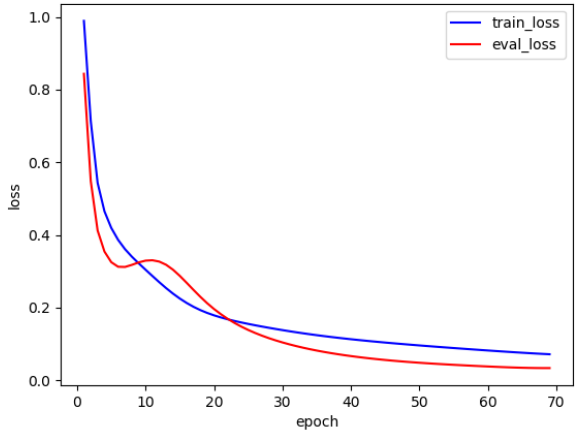

Figure 3: The epoch-loss curves for train data and validation data on iris data set.

4.7 train.py

In this module, we train the Neural Network model using different data sets and hyper parameters. We use batch data to train the model, and use batch Stochastic Gradient Descent(SGD) algorithm to optimize the parameters of the model. At each iteration, we firstly optimize the model on train data set and calculate the average loss value of it. Then we calculate the average loss value on validation data set without optimizing model. And if the loss value of train data or validation data increases more than 5 times, we early stop training.

As for learning rate, after each training iteration, if the loss value decreases, then the learning rate is increased 5%. Otherwise the learning rate is decreased 50%.

After finishing training, respectively calculate accuracy rate in train data set, validation data set and test data set.

4.8 static/

The static directory stores the input data set, and output logs and figures.

5 Evaluation of Results

We train the Neural Network model on iris data set and wine-quality data set.

5.1 Accuracy Rate of Two Data Set

For iris data set, the hidden size is 64 with one hidden layer, batch size is 8, activator is $tanh$ and learning rate is 0.01. The curve between iterations and loss value of train data set and validation set is show as Figure 3. It is early stopped after training 69 iterations. The accuracy rate of train set, validation set and test set is respectively 98.33%, 100% and 100%.

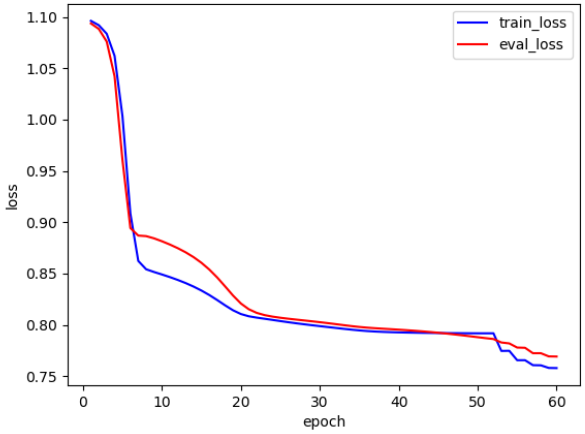

Figure 4: The epoch-loss curves for train data and validation data on abalone data set.

For abalone data set, the hidden layers have 4 layers with 16, 32, 64 neurons, batch size is 16, activator is $tanh$ and learning rate is 0.01. The curve between iterations and loss value of train data set and validation set is show as Figure 4. It is early stopped after training 36 iterations. The accuracy rate of train set, validation set and test set is respectively 63.78%, 62.26% and 66.82%.

We can see the fully connected Neural Network model performs well on nonlinear multi-classification problems, when the activator of output layer is $softmax$ and the loss function is cross entropy. With proper depth of hidden layer and size of hidden layer, the model performs well and the accuracy is low.

However if we do not combine catalogs on abalone data set and use 29 catalogs to train the model. The accuracy is only around 25%. Because the 29 catalogs are imbalance from which the neural network can not learn how to classify them well.

5.2 Accuracy Rate of Different Hyper Parameters

There are some hyper parameters affecting the performances of result, including depth of hidden layer, size of each hidden layer, batch size, activator and initialization of learnable parameters. We choose various values for each hyper parameter to evaluate the performances.

Initialization method. If the initial weights are full of zero. The accuracy rate on iris data is only around 50%. The loss value decreases very slowly until number of iterations is around 500. Because when all weights are zero, the output of each layer is zero and the gradient delta of each layer is also zero except the bias of output layer. So there is only a learnable parameter changing in the training which is the bias of the output layer. We can use random number in [-0.1, 0.1] or random number based normal distribution to initial parameters.

Hidden layer size. When the number of neurons of hidden layer increase, the accuracy increases first and then decreases. Because the model can not learn enough features from data set when the hidden size is too small. In comparison that, the model is easily overfitting and learn the features which is not universal but specific to this data set. So the hidden layer size should not be too big or too small.

Batch size. When the bigger is the batch size from 8 to 128, the faster is the speed of each iteration. But more and more epochs are required to achieve the similar accuracy and even it can not achieve the similar accuracy, when the batch size is increasing. So the batch size should also not be too big or too small.

6 Conclusion

We implement the fully connected Neural Network based on Python and numpy, and train and evaluate model on two data sets including iris and wine-quality. The model has one input layer, one output layer, and multiple hidden layers. We use $softmax$ activator and cross entropy cost function to solve the muti-classification problems. The accuracies on two data set are both bigger than 90%.

But there are also some limitations in the current model. We only use the fully connected layer. But when the numbers of neurons and features is much larger, the speed of calculations will be very slow. We can use convolutional neural network when the number of neurons are larger.

What's more, if the depth of hidden layer is larger, the problems of gradient vanishing and gradient exploding will be easier to occur. We can use drop out mechanism to avoid them, that is, the model drops out randomly some neurons at each batch training and only optimize weights of the rest neurons. When predicting target value, the model use all the neurons to calculate the output.

The classification distribution for data set is only important. If the data set is not imbalance between different categories, the neural network model can not learn how to classify well. To solve it, we combine close categories. The better way may be using weighted cross entropy loss function instead of standard cross entropy loss function, or over-sampling and under-sampling data set.