Back Propagation algorithm for Feed Forward Neural Networks Implementation and Applications

Chaoyu Li, chaoyuli@usc.edu, 6641732094

DSCI-552: Machine Learning for Data Science

Instructor: Satish Kumar Thittamaranahalli

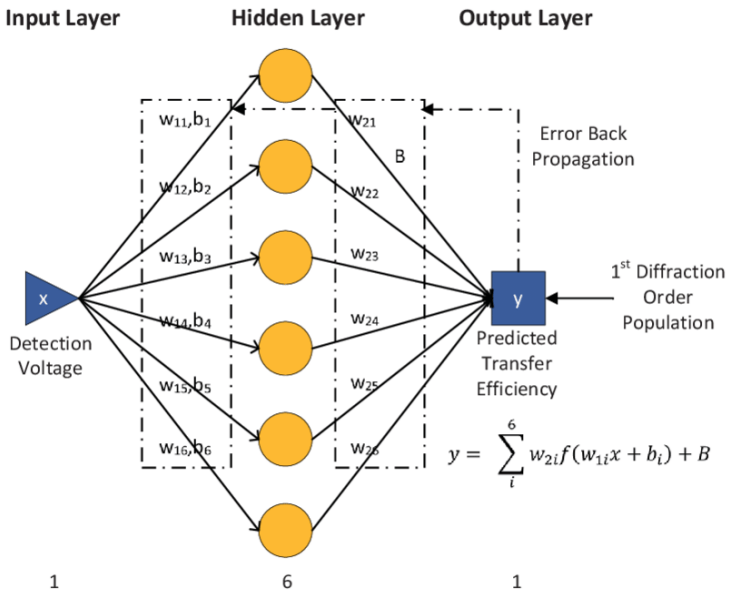
Mar. 30, 2022

Table of Contents

1 Basic Algorithm and Code Functions Description

1.1 Implementation of BPNN

1.1.1 Basic Algorithm of BPNN



1.1.2 Code Functions of BPNN

- **label(filename)**

It read directory list file to extract files with one and zero label. It accepts the name of .list files and it returns two list as : ['file1_label_1', 'file2_label_1' …] and ['file1_label_0', 'file2_label_0'...].

- **readImage(files, label)**

It read an image file using CV2 and changes its type to a 1-D list. Finally, this function will add the label of the image file to the end of the list. It accepts a list of label from function label() and a list of names of the image files, and it returns one list.

- **readImage(files, label)**

It read an image file using CV2 and changes its type to a 1-D list. Finally, this function will add the label of the image file to the end of the list. It accepts a list of label from function label() and a list of names of the image files, and it returns one list.

- **Sigmoid.forward(X)**

  It represents the forward sigmoid formula: sigmoid(z) = s = 1/(1+exp(-z)).

- **Sigmoid.backward(X, dLds)**

  It represents the backward sigmoid formula. Here we know dL/ds and we need to calculate dL/dz. Therefore, the formula should be: dL/dz=dL/ds*ds/dz=dL/ds*s*(1-s).

- **MultiplyGate.forward(W, X)**

  It represents the forward multiply-gate formula: Z = X * W.

- **MultiplyGate.backward(W, X)**

  It represents the backward multiply-gate formula: dL/dW=dL/dZ*dZ/dW=dL/dZ*X.

- **AddGate.forward(W, X)**

  It represents the forward add-gate formula: S = Z+b.

- **AddGate.backward(W, X)**

  It represents the backward add-gate formula:dL/dZ=dL/dS*dS/dZ=dL/dS *Dimension(1).

- **randomizer(n1, n2)**

  It initializes the weight matrix via randomized initiation n1*n2 dimension. Here n1 and n2 are the number of neurons in layer i and i+1 respectively. The value of the weight specified between -0.01 and 0.01.

- **calculate_loss(X, y)**

  It calculates the loss of the forward propagation computation of the neural network. It is used to evaluate how well the model is doing. For the loss function, here I choose LSE.

- **train(X, y, num_passes, lr, regularization)**

  It is the main function for training the neural network model. It accepts the training dataset, the labels corresponding to the dataset, the number of epochs, the learning rate, and the regularization rate. For each epoch, it consists of forward propagation and back propagation. Each forward propagation model predicts each data and calculates the derivative of cumulative error from output layer. Each back propagation modifies the weight matrix according to the calculated derivative. Here the gradient descent method is used to back propagation.

- **predict(X)**

  It is a prediction function to calculate the output of the network. It accepts the dataset and does forward propagation as defined above. It returns the class with the highest probability.

- **eval(Y_pred, Y_true)**

  It calculates the accuracy of predicted labels list. It accepts a array of predicted labels and groundtruth and returns the accuracy.

2 Data Structure Used

2.1 BPNN data structure

A. The name of train and test files that belong to different labels are stored in the data structure of List. The format is:

train1: ['gestures/A/A_down_4.pgm', …, 'gestures/M/M_down_8.pgm']

train0: ['gestures/A/A_hold_3.pgm', …, 'gestures/M/M_up_6.pgm']

test1:['gestures/A/A_down_1.pgm', …, 'gestures/K/K_down_3.pgm']

test0: ['gestures/A/A_hold_1.pgm', …, 'gestures/K/K_stop_2.pgm']

B. The pgm image files are stored in the data structure of List. The format is:

train_image1 = [960 px, 1]

train_image0 = [960 px, 0]

C. The layer dimensions are stored in the data structure of NpArray. The format is:

[dimension of training data, number of epoches, number of output layer]

D. The output of output layer is stored in the data structure of NpArray.

E. The weights and biases are stored in the data structure of NpArray.

3 Challenges Faced and Optimizations

3.1 Challenges

A. It is needed to add a regularization term, otherwise the accuracy is not high.

B. The neural network can be improved with minibatch stochastic gradient descent. Each epoch does not have to scan the whole training data.

3.2 Optimizations

A. Regularization parameter lambda is added to the weight calculation to avoid overfitting.

B. Another popular activation function “tanh” can be considered to improve the accuracy.

4 Execution of my algorithm and results

- Print of result:

Loss after iteration 0: 0.208670

Loss after iteration 100: 0.105323

Loss after iteration 200: 0.088921

Loss after iteration 300: 0.086943

Loss after iteration 400: 0.086551

Loss after iteration 500: 0.085147

Loss after iteration 600: 0.079845

Loss after iteration 700: 0.086455

Loss after iteration 800: 0.085338

Loss after iteration 900: 0.087218

training accuracy:

0.8858695652173914

testing accuracy:

0.8674698795180723

5 Comparison between my algorithm and scikit-learn

- The code of scikit-learn in BPNN is:

```
from sklearn.neural_network import MLPClassifier
def main(train_List_Dir, test_List_Dir):
    # initialize lists of images and labels
    images = []
    labels = []
    # import training data
    with open(train_List_Dir) as f:
        for line in f.readlines():
            # get file directory
            train_img_dir = line.strip()
            # import the images
            images.append(read_PGM_img(train_img_dir))
            # assign label based on file name: 1 for down gesture, 0 for otherwise
            if 'down' not in train_img_dir:
                labels.append(0)
            else:
                labels.append(1)

    nn = MLPClassifier(solver='sgd', tol=1e-3, alpha=0.1, learning_rate='adaptive',
                       hidden_layer_sizes=(100,), activation='logistic', learning_rate_init=0.1,
                       max_iter=1000, verbose=False, warm_start=True, early_stopping=False,
validation_fraction=0.1)

    nn.fit(images, labels)
    f.close()
    # predict testing data
    total_count = 0
    correct_count = 0
    with open(test_List_Dir) as f:
        for line in f.readlines():
            total_count += 1
            test_image = line.strip()
            pred = nn.predict([read_PGM_img(test_image), ])[0]
            if (pred == 1) == ('down' in test_image):
                correct_count += 1
```

```
    print('correct_count rate on test data: {}'.format(correct_count / total_count))
```

- The result is:
  Accuracy: 0.8674698795180723

  The accuracy is exactly the same as mine.

6 Application of BPNN

- Handwriting Recognition – The two common applications of handwriting recognition are: Optical character recognition for data entry and Validation of signatures on a bank cheque.
- Traveling Salesman Problem – Neural networks can also solve the traveling salesman problem. But this is to a certain degree of approximation only.
- Image Compression – Neural networks receive and process large amounts of information at once. This makes them useful in image compression. With the explosion of the Internet and more websites using more images on their sites, image compression using neural networks is worth a look.
- Stock Exchange Prediction – The daily business of the stock market is very complex. Many factors affect whether a given stock will rise or fall on any given day. As a result, neural networks can quickly examine a large amount of information and sort it all out. So we can use them to predict stock prices.

7 Contributions

All works were completed by Chaoyu Li.