CSE-6363-007 Assignment 3 (Spring 2024)

Solution Report

- **Thanuj Kumar Shivalingaiah (1002136007)**

Problem 1: [80 points]

In this problem, we will implement the feedforward and backpropagation process of the neural networks. We will use digital.mat as our experiment data. Then we can train three layer (data, hidden-relu, loss) neural networks and report test accuracy.

Here is what you need to do:

(i) Finish fullyconnect_feedforward, fullyconnect_backprop, relu_feedforward and relu_backprop part.

(ii) Print loss and accuracy in every training epoch and test.

fullyconnect_feedforward:

```
def fullyconnect_feedforward(in_, weight, bias):
    '''
    The feedward process of fullyconnect
      input parameters:
          in_     : the inputs, shape: [number of images, number of inputs]
          weight  : the weight matrix, shape: [number of inputs, number of outputs]
          bias    : the bias, shape: [number of outputs, 1]

      output parameters:
          out     : the output of this layer, shape: [number of images, number of outputs]
    '''
    # TODO
    # begin answer
    out = np.dot(in_, weight) + bias.T
    # end answer
```

relu_feedforward:

```
def relu_feedforward(in_):
    '''
    The feedward process of relu
      in_:
          in_   : the input, shape: any shape of matrix

      outputs:
          out : the output, shape: same as in
    '''
    # TODO
    # begin answer
    out = np.maximum(0, in_)
    # end answer
    return out
```

relu_backprop:

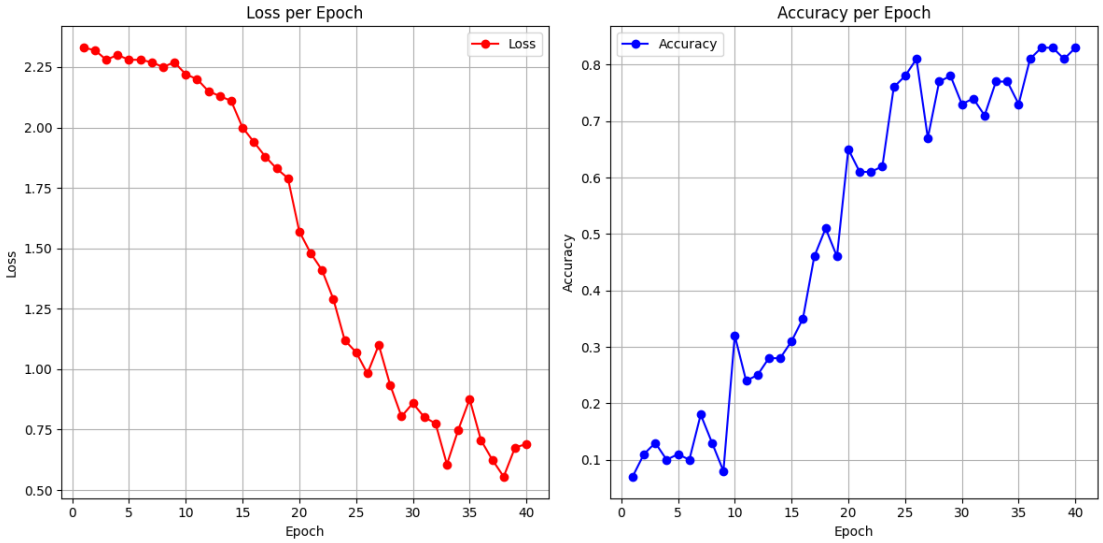
```
def relu_backprop(in_sensitivity, in_):
    '''
    The backpropagation process of relu
      input paramter:
          in_sensitivity  : the sensitivity from the upper layer, shape:
                          : [number of images, number of outputs in feedforward]
          in_             : the input in feedforward process, shape: same as in_sensitivity

      output paramter:
          out_sensitivity : the sensitivity to the lower layer, shape: same as in_sensitivity
    '''
    # TODO

    # begin answer
    out_sensitivity = in_sensitivity * (in_ > 0)
    # end answer
    return out_sensitivity
```

fullyconnect_backprop:

```
def fullyconnect_backprop(in_sensitivity, in_, weight):
    '''
    The backpropagation process of fullyconnect
      input parameter:
          in_sensitivity  : the sensitivity from the upper layer, shape:
                          : [number of images, number of outputs in feedforward]
          in_             : the input in feedforward process, shape:
                          : [number of images, number of inputs in feedforward]
          weight          : the weight matrix of this layer, shape:
                          : [number of inputs in feedforward, number of outputs in feedforward]

      output parameter:
          weight_grad     : the gradient of the weights, shape:
                          : [number of inputs in feedforward, number of outputs in feedforward]
          out_sensitivity : the sensitivity to the lower layer, shape:
                          : [number of images, number of inputs in feedforward]

    Note : remember to divide by number of images in the calculation of gradients.
    '''

    # TODO
    # begin answer
    weight_grad = np.dot(in_.T, in_sensitivity)
    bias_grad = np.sum(in_sensitivity, axis=0, keepdims=True).T
    out_sensitivity = np.dot(in_sensitivity, weight.T)
    # end answer

    return weight_grad, bias_grad, out_sensitivity
```

Output: I have attached in the zip file (output.txt)

Left Graph: "Training Loss per Epoch" | Right Graph: "Training Accuracy per Epoch"



Loss per Epoch (Left Graph)

- This graph shows the loss value of the neural network on the training set as it progresses through the epochs.
- The loss is initially high, starting just above 2.3, which is typical at the beginning of training when the model's weights are still being adjusted.
- As training progresses, the loss steadily decreases, indicating that the model is learning and improving its predictions.
- Around epoch 25, the loss begins to plateau, suggesting that the model might be approaching its minimum loss point or that further learning might require adjustments to the learning rate, additional data, or changes in the model architecture.

Accuracy per Epoch (Right Graph)

- This graph plots the model's accuracy on the training set across epochs.
- Accuracy starts low, which is expected when the model begins training with initial weights.
- There is a sharp increase in accuracy early in training, with the most significant gains in performance occurring before epoch 10.
- After this initial jump, the improvements in accuracy become smaller, and the graph shows some fluctuations. This could indicate the model is facing some difficulty in further refining its predictions, possibly due to the complexity of the data, noise within the dataset, or the need for further tuning.
- The accuracy seems to plateau towards the end of the training, hovering around 0.8 to 0.9, which is quite high and suggests good performance.