# ECE421 - Winter 2022 Assignment 2: Neural Networks

Mingyu Zheng 1003797661

Due date: February 28

## 1 Neural Networks using Numpy

### 1.1 Helper Functions

1.$relu()$ :

```
def relu(x):
    # TODO
    relu_x = np.maximum(x, 0)
    return relu_x
    # pass
```

Figure 1: Screenshot of relu() Function

2.$softmax()$ :

```
def softmax(x):
    # TODO
    x = x - np.max(x, axis=1, keepdims=True)
    softmax_x = np.exp(x)/np.sum(np.exp(x), axis=1, keepdims=True)
    return softmax_x
    # pass
```

Figure 2: Screenshot of softmax() Function

3.$compute_layer()$ :

```
def compute_layer(x, w, b):
    # TODO
    compute_layer = np.dot(x, w) + b
    return compute_layer
    # pass
```

Figure 3: Screenshot of compute$_l$*ayer*()*Function*

4.*averagece*() :

```
def average_ce(target, prediction):
    # TODO
    average_ce = - np.mean(np.multiply(target, np.log(prediction)))
    return average_ce
    # pass
```

Figure 4: Screenshot of average$_c$*e*()*Function*

5.*gradce*() :

```
def grad_ce(target, logits):
    # TODO
    grad_ce = logits - target
    return grad_ce
    # pass
```

Figure 5: Screenshot of grad$_c$*e*()*Function*

Derive softmax():

$$\frac{\partial \mathbf{L}}{\partial \mathbf{o}} = \frac{\partial \mathbf{L}}{\partial \mathbf{s}} * \frac{\partial \mathbf{s}}{\partial \mathbf{o}}$$

$$\frac{\partial \mathbf{L}}{\partial \mathbf{s}} = -\frac{1}{N}\sum_{k=1}^{10} \frac{y_k}{s_k}$$

Since:

$$s(o_j) = \frac{exp(o_j)}{\sum_{k=1}^{10} exp(o_k)}$$

And:

$$\frac{\partial \mathbf{L}}{\partial \mathbf{o}}$$
$$= s_j * (s_j(1 - o_j)) \text{when i equals j}$$
$$= -s_j \text{when i not equals j}$$

Apply chain rule:

$$\frac{\partial L}{\partial \mathbf{o}} = (1/N) * (-y_i + S_i \sum_{k=1}^{10} y_k S_i)$$
$$= (1/N) * (S_i - y_k)$$

Therefore:

$$\frac{\partial \mathbf{L}}{\partial \mathbf{o}} = \text{logits - target}$$

## 1.2 Backpropagation Derivation

1.

$$\frac{\partial L}{\partial W_o} = \frac{\partial L}{\partial o} * \frac{\partial o}{\partial W_o} = (outputofhiddenlayer).Tdot(gradce)$$

shape = (number of hidden units, 10)

```
# the gradient of the loss with respect to the output layer weights.
def dL_dWo(target, logits, hidden_layer_out):
    grad = grad_ce(target, logits)
    dL_dWo = np.dot(hidden_layer_out.T, grad)
    return dL_dWo
```

Figure 6: Screenshot of dLdWo() Function

2.

$$\frac{\partial L}{\partial b_o} = \frac{\partial L}{\partial o} * \frac{\partial o}{\partial b_o} = (vector1s).Tdot(gradce)$$

shape = (1, 10)

```
# the gradient of the loss with respect to the output layer biases.
def dL_dbo(target, logits):
    ones = np.ones((1, target.shape[0]))
    grad = grad_ce(target, logits)
    dL_dbo = np.dot(ones, grad)
    return dL_dbo
```

Figure 7: Screenshot of dLdbo() Function

3.

$$\frac{\partial L}{\partial W_h} = (input_data.T)dot(dRelu * (gradcedotWo.T))$$

shape = (784, 10000) dot ((10000, number of hidden units) * (10000, 10) dot (10, number of hidden units))

```
# the gradient of the loss with respect to the hidden layer weights.
def dL_dWh(d_relu, input, target, logits, Wout):
    d_relu[d_relu >= 0] = 1
    d_relu[d_relu < 0] = 0
    grad = grad_ce(target, logits)
    dL_dWh = np.dot(input.T, (d_relu * np.dot(grad, Wout.T)))
    return dL_dWh
```

Figure 8: Screenshot of dLdWh() Function

4.

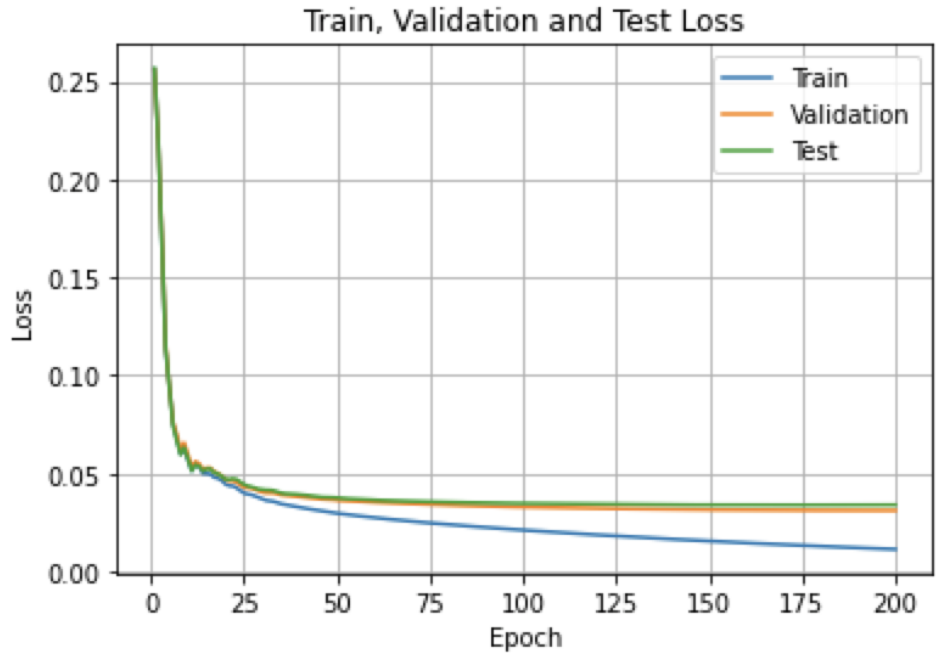
$$\frac{\partial L}{\partial b_h} = (vector1s)dot(dRelu * (gradcedotWout.T))$$

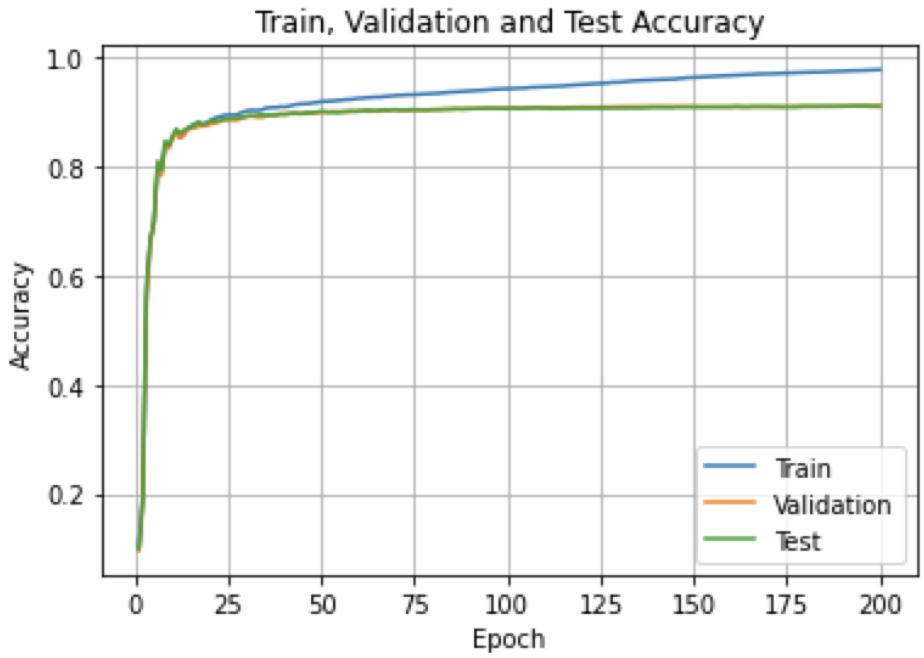
shape = (1, 10000) dot ((10000, number of hidden units) * (10000, 10) dot (10, number of hidden units))

```
# the gradient of the loss with respect to the hidden layer biases.
def dL_dbh(target, logits, d_relu, Wout):
    d_relu[d_relu >= 0] = 1
    d_relu[d_relu < 0] = 0
    ones = np.ones((1, d_relu.shape[0]))
    grad = grad_ce(target, logits)
    dL_dbh = np.dot(ones, (d_relu * np.dot(grad, Wout.T)))
    return dL_dbh
```

Figure 9: Screenshot of dLdbh() Function

### 1.3 Learning




```
Final train accuracy is: 0.9769
Final validation accuracy is: 0.9111666666666667
Final test accuracy is: 0.9093245227606461
```

Figure 10: Screenshot of learning with gamma=0.9,H=1000,alpha=1e-5

The first plot shows the loss of training, validation and test. The second plot shows the accuracy of training, validation and test. The final training accuracy is 0.977. The final validation accuracy is 0.911. The final test accuracy is 0.909.