Assignment 2: Neural Networks

By Ahoo Saeifar ECE421-Winter 2021

March 1, 2021

1. Neural Networks using Numpy

1.1 Helper Functions

1. ReLu:

```python
def relu(x):
    return np.maximum(0.0,x)
```

2. softmax:

```python
def softmax(x):
    maxVal=np.amax(x)
    x=x-maxVal
    sigma=np.exp(x)/np.sum(np.exp(x),axis=1,keepdims=True)
    return sigma
```

3. computeLayer:

```python
def computeLayer(X, W, b):
    return np.matmul(X,W)+b
```

4. CE:

```python
def CE(target, prediction,dataShape):
    averageCE=-np.sum(target * np.log(prediction))/dataShape
    return averageCE
```

5. gradCE:

a. Code Snippet

```python
def gradCE(target, prediction):
    return (prediction - target)/N
```

b. Derivation of Gradient of Cross Entropy Loss

Gradient of softmax function with respect to its input z:

if $i = j$: $\frac{\partial p_i}{\partial z_i} = \frac{\partial \frac{e^{z_i}}{\Sigma e^{z_i}}}{\partial z_i} = \frac{e^{z_i}\Sigma e^{z_i} - e^{z_i}e^{z_i}}{\Sigma(e^{z_i})^2} = \frac{e^{z_i}}{\Sigma e^{z_i}}\frac{\Sigma e^{z_i} - e^{z_i}}{\Sigma e^{z_i}} = \frac{e^{z_i}}{\Sigma e^{z_i}}(1 - \frac{e^{z_i}}{\Sigma e^{z_i}}) = p_i(1 - p_i)$

if $i \neq j$: $\frac{\partial p_i}{\partial z_j} = \frac{\partial \frac{e^{z_i}}{\Sigma e^{z_j}}}{\partial z_j} = \frac{0 - e^{z_i}e^{z_j}}{\Sigma(e^{z_j})^2} = -\frac{e^{z_i}}{\Sigma e^{z_i}}\frac{e^{z_j}}{\Sigma e^{z_j}} = -p_i p_j$

Using the results from above we can find the gradient of cross entropy loss:
(sensitivity of L to layer l input: δ_i^l)

$$
\begin{aligned}
\frac{\partial L}{\partial z_i} &= -\sum_{k=1}^{K}\frac{\partial y_k \log(p_k)}{\partial z_i} = -\sum_{k=1}^{K} y_k \frac{\partial \log(p_k)}{\partial z_i} = -\sum_{k=1}^{K} y_k \frac{1}{p_k}\frac{\partial p_k}{\partial z_i} \\
&= -\frac{y_i}{p_i}\frac{\partial p_i}{\partial z_i} - \sum_{k\neq i}^{K}\frac{y_k}{p_k}\frac{\partial p_k}{\partial z_i} = -\frac{y_i}{p_i}p_i(1 - p_i) - \sum_{k\neq i}^{K}\frac{y_k}{p_k}(-p_k p_i) \\
&= -y_i + y_i p_i + \sum_{k\neq i}^{K} y_k p_i = -y_i + \sum_{k=1}^{K} y_k p_i = -y_i + p_i\sum_{k=1}^{K} y_k \\
&= p_i - y_i
\end{aligned}
$$

1.2 Backpropagation Derivation

We can use the partial derivatives below with chain rule to find the required derivatives:

$$\frac{\partial L}{\partial z_i} = p_i - y_i \text{ (from 1.1.5)} \qquad \frac{\partial z_i}{\partial W_o} = \frac{\partial W_o h + b_o}{\partial W_o} = h$$

$$\frac{\partial z}{\partial b_o} = \frac{\partial W_o x + b_o}{\partial b_o} = 1 \qquad \frac{\partial z}{\partial h} = W_o \qquad \frac{\partial h}{\partial r} = \begin{cases} 1 \text{ if } r>0 \\ 0 \text{ else} \end{cases}$$

$$\frac{\partial r}{\partial W_h} = x \qquad \frac{\partial r}{\partial b_h} = 1$$

*Note: $h = ReLu(r)$ and $p = softmax(z)$

1. $$\frac{\partial L}{\partial W_o} = \frac{\partial L}{\partial z_i}\frac{\partial z_i}{\partial W_o}$$
$$\frac{\partial L}{\partial W_o} = (p_i - y_i)h$$

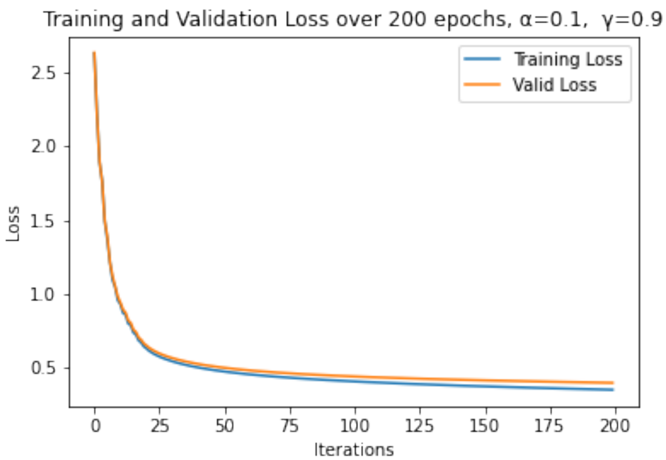
2. $$\frac{\partial L}{\partial b_o} = \frac{\partial L}{\partial p}\frac{\partial p}{\partial z}\frac{\partial z}{\partial b_o}$$
$$\frac{\partial L}{\partial b_o} = (p_i - y_i)$$

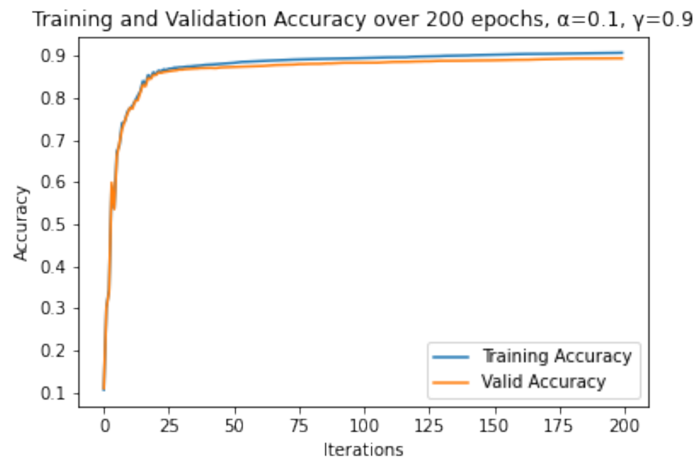
3. $$\frac{\partial L}{\partial W_h} = \frac{\partial L}{\partial p}\frac{\partial p}{\partial z}\frac{\partial z}{\partial h}\frac{\partial h}{\partial r}\frac{\partial r}{\partial W_h}$$
$$\frac{\partial L}{\partial W_h} = (p_i - y_i)(W_o)(h > 0)(x)$$

4. $$\frac{\partial L}{\partial b_h} = \frac{\partial L}{\partial p}\frac{\partial p}{\partial z}\frac{\partial z}{\partial h}\frac{\partial h}{\partial r}\frac{\partial r}{\partial b_h}$$
$$\frac{\partial L}{\partial b_h} = (p_i - y_i)(W_o)(h > 0)$$

<u>*Code snippets are in the Appendix</u>

1.3 Learning


Training and Validation Loss over 200 epochs, α=0.1, γ=0.9
Training Loss
Valid Loss
Loss
Iterations
2.5
2.0
1.5
1.0
0.5
0
25
50
75
100
125
150
175
200


Training and Validation Accuracy over 200 epochs, α=0.1, γ=0.9
Training Accuracy
Valid Accuracy
Accuracy
Iterations
0.9
0.8
0.7
0.6
0.5
0.4
0.3
0.2
0.1
0
25
50
75
100
125
150
175
200

Appendix

1.1 Code Snippet for Gradients required in Backpropagation

```
# the gradient of the loss with respect to the output layer weights
def gradLossW_o(target, prediction,hiddenLayer):
    CE=gradCE(target,prediction)
    return np.matmul(np.transpose(hiddenLayer),CE)


# the gradient of the loss with respect to the output layer bias
def gradLossb_o(target, prediction,hiddenLayer):
    ones=np.ones((1,N))
    CE=np.matmul(ones,gradCE(target,prediction))
    return CE

# the gradient of the loss with respect to the hidden layer weights
def gradLossW_h(target,prediction,hiddenLayer,x_in,W_out): # x_in is train_data
    CE=gradCE(target,prediction)
    loss_W_h=np.matmul(np.transpose(x_in),der_relu(hiddenLayer)*np.matmul(CE,np.transpose(W_out)))
    return loss_W_h

# the gradient of the loss with respect to the hidden layer biases
def gradLossb_h(target,prediction,hiddenLayer,x_in,W_out):
    CE=gradCE(target,prediction)
    loss_W_h=np.matmul(CE,np.transpose(W_out))*der_relu(hiddenLayer)
    ones=np.ones((1,N))
    return  np.matmul(ones,loss_W_h)
```