ECE421 PA2 Report- Sanjana

1.1 Helper Functions

1. relu()
   a.
```python
def relu(x):
    # TODO
    # x is an array so we need to check element wise if x[i] > 0 or not
    return np.maximum(x, 0)
```

2. softmax()
   a.
```python
def softmax(x):
    # TODO
    # prevent overflow
    x = x - (np.max(x, axis=1)).reshape(x.shape[0], 1)
    # x = x - x.max(axis=1, keepdims=True)
    #return np.exp(x)/np.sum(np.exp(x), axis=1, keepdims=True)
    return np.exp(x) / (np.sum(np.exp(x), axis=1)).reshape(x.shape[0],1)
```

3. compute()
   a.
```python
def compute_layer(x, w, b):
    # TODO
    # return x@w + b
    # x: N, F
    # w: F, H
    # b: 1, H
    # return N, H
    #s = np.dot(x, w) + np.transpose(b)
    s = np.matmul(x, w) + b
    return s
```

4. average()
   a.
```python
def average_ce(target, prediction):
    # TODO
    N = target.shape[0]
    #avg_ce = -np.sum(np.multiply(target, np.log(prediction))) / N
    avg_ce = -np.sum(target * np.log(prediction + 0.00001)) / N
    return avg_ce
```

5. grad_ce()

a.

need $\frac{\partial L}{\partial D} = \frac{\partial L}{\partial f} \cdot \frac{\partial f}{\partial o}$

$f = \text{Softmax}(o)$

$L = \sum_{n=1}^{N} \sum_{k=1}^{K} y_k^{(n)} \log(p_k^{(n)})$

$\frac{\partial L}{\partial p_o} = \sum_{n=1}^{N} \sum_{k=1}^{K} \frac{\partial y_k^{(n)} \log(p_k^{(n)})}{\partial p_o}$

$= -y^T \frac{1}{p}$

$\frac{\partial p}{\partial o} = \frac{e^o (\sum_{k=1} e^{o_k}) - (e^o)^T e^o}{(\sum_{k=1} e^{o_k})^2}$ } Quotient rule

$= \frac{e^o}{\sum_{k=1}^{K} e^{o_k}} \cdot \frac{\sum_{k=1} e^{o_k} - e^o}{\sum_{k=1}^{K} e^{o_k}}$

$= p(1-p)$

$\frac{\partial L}{\partial o} = \frac{\partial L}{\partial f} \cdot \frac{\partial f}{\partial o} = -y^T \frac{1}{p} \cdot p(1-p)$

$= \underline{p} - \underline{y}$

b.

```python
def grad_ce(target, logits):
    # TODO
    #prediction = softmax(logits)
    # return (prediction-target).mean(axis=0)
    return (logits - target)
```

1.2 Backpropagation derivation

1. dL/dw_o

$$\frac{\partial L}{\partial w_o} = \frac{\partial L}{\partial p_o} \cdot \frac{\partial p_o}{\partial z_o} \cdot \frac{\partial z_o}{\partial w_o}$$ (use chain rule)

$$\frac{\partial p_o}{\partial z_o} = p_o(1-p_o)$$

derived previously

$$\frac{\partial L}{\partial p_o} = -y \frac{1}{p_o}$$

$$\frac{\partial z_o}{\partial w_o} = \frac{\partial}{\partial w_o}(x_h w_o + b_o) = x_h$$

$$\Rightarrow \frac{\partial L}{\partial w_o} = x_h (p-y)^T$$ $H \times 1$ 1×1 $H \times 1$

a.

2. dL/db_o

$$\frac{\partial L}{\partial b_o} = \frac{\partial L}{\partial p_o} \cdot \frac{\partial p_o}{\partial z_o} \cdot \frac{\partial z_o}{\partial b_o}$$

$$= (p-y)$$

a.

b.

3. dL/dw_h

$$\frac{\partial L}{\partial w_h} = \frac{\partial L}{\partial p_o} \cdot \frac{\partial p_o}{\partial z_o} \cdot \frac{\partial z_o}{\partial x_h} \cdot \frac{\partial x_h}{\partial z_h} \cdot \frac{\partial z_h}{\partial w_h}$$

$$\frac{\partial z_o}{\partial x_h} = \frac{\partial x_h w_o + b_o}{\partial x_h} = w_o$$

$$\frac{\partial x_h}{\partial z_h} = \begin{cases} 0 & \text{if } z_h \le 0 \\ 1 & \text{if } z_h > 0 \end{cases} = (x_h > 0)$$ indicator fcn

$$\frac{\partial z_h}{\partial w_h} = \frac{\partial}{\partial w_h}(x_{in} w_h + b_h) = x_{in}$$

$$\frac{\partial L}{\partial w_h} = (p-y)\, w_o\, \mathbb{I}(x_h > 0)\, x_{in}$$

a.

4. dL/db_h

$$\frac{\partial L}{\partial b_h} = \frac{dL}{x_o} \cdot \frac{\partial x_o}{\partial z_o} \cdot \frac{\partial z_o}{\partial x_h} \cdot \frac{dx_h}{dz_h} \cdot \frac{dz_h}{db_h}$$

$$\frac{\partial z_h}{db_h} = \frac{\partial x_{in} w_h + b_h}{\partial b_h} = 1$$

$$\Rightarrow \frac{dL}{db_h} = (p - y) w_o \, I(x_h > 0)$$

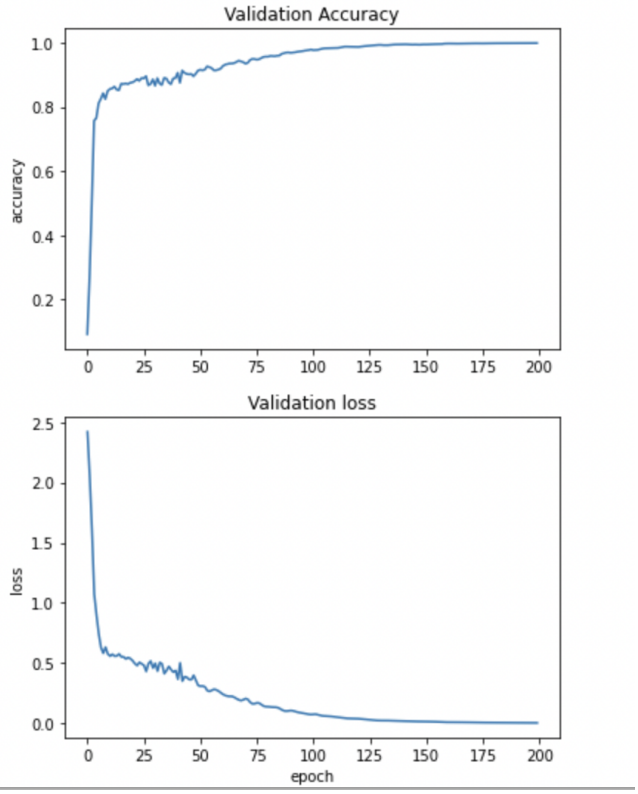
a.

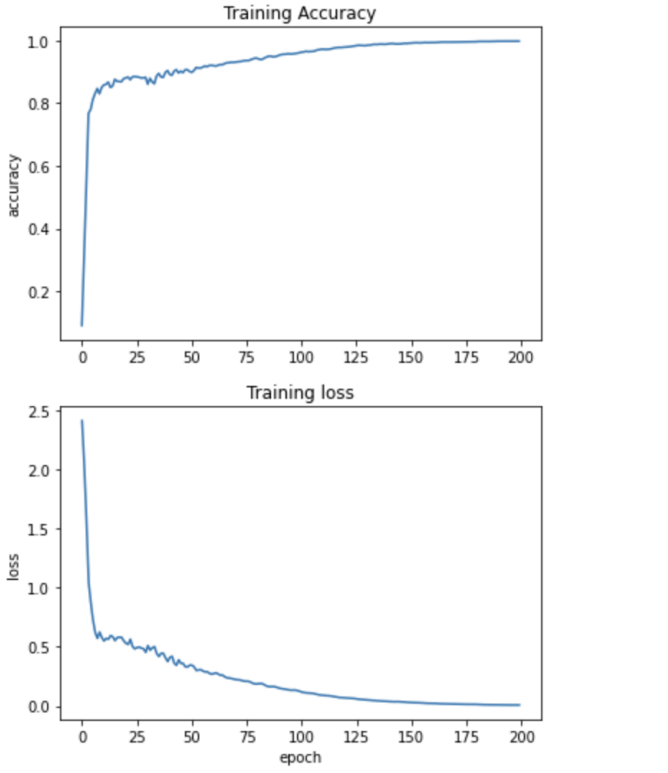
1.3 Learning

1. Code

```
for epoch in range(200):
  #if epoch % 10 == 0:
    #print("Iteration ", epoch)
  print("Iteration ", epoch)
  # forward step
  x_h, x_o, z_o, z_h = forward_prop(x_i, b_h, w_h, b_o, w_o)
  prediction = np.argmax(x_o, axis = 1)
  target = np.argmax(train_target, axis=1)

  # find accuracy
  accuracy = (prediction == target).mean()
  accuracies.append(accuracy)

  # find loss
  loss = average_ce(train_target, x_o)
  losses.append(loss)

  print("Accuracy ", accuracy)
  print("Loss ", loss)

  #print("Point C")
  # backward step
  d_wo, d_bo, d_wh, d_bh = back_prop(x_h, x_o, x_i, w_o, z_o, train_target, z_h)

  vw_h = gamma*vw_h + learning_rate*d_wh
  vb_h = gamma*vb_h + learning_rate*d_bh
  vw_o = gamma*vw_o + learning_rate*d_wo
  vb_o = gamma*vb_o + learning_rate*d_bo

  w_h = w_h - vw_h
  b_h = b_h - vb_h
  w_o = w_o - vw_o
  b_o = b_o - vb_o

  # validation
  x_h_valid, x_o_valid, z_o_valid, z_h_valid = forward_prop(x_i_valid, b_h_valid, w_h_valid, b_o_valid, w_o_valid)
  prediction_valid = np.argmax(x_o_valid, axis = 1)
  target_valid = np.argmax(valid_target, axis=1)

  # find accuracy
  accuracy_valid = (prediction_valid == target_valid).mean()
  accuracies_valid.append(accuracy_valid)

  # find loss
  loss_valid = average_ce(valid_target, x_o_valid)
  losses_valid.append(loss_valid)

  d_wo_valid, d_bo_valid, d_wh_valid, d_bh_valid = back_prop(x_h_valid, x_o_valid, x_i_valid, w_o_valid, z_o_valid, valid_target, z_h_valid)

  vw_h_valid = gamma*vw_h_valid + learning_rate*d_wh_valid
  vb_h_valid = gamma*vb_h_valid + learning_rate*d_bh_valid
  vw_o_valid = gamma*vw_o_valid + learning_rate*d_wo_valid
  vb_o_valid = gamma*vb_o_valid + learning_rate*d_bo_valid

  w_h_valid = w_h_valid - vw_h_valid
  b_h_valid = b_h_valid - vb_h_valid
  w_o_valid = w_o_valid - vw_o_valid
  b_o_valid = b_o_valid - vb_o_valid
```

2. Graphs


Validation Accuracy
accuracy
1.0
0.8
0.6
0.4
0.2
0
25
50
75
100
125
150
175
200
Validation loss
loss
2.5
2.0
1.5
1.0
0.5
0.0
0
25
50
75
100
125
150
175
200
epoch


Training Accuracy
accuracy
1.0
0.8
0.6
0.4
0.2
0
25
50
75
100
125
150
175
200
Training loss
loss
2.5
2.0
1.5
1.0
0.5
0.0
0
25
50
75
100
125
150
175
200
epoch