Homework 4

Submitted By → Jyotsna Rajaraman.

Q1.

Given:

3 layer neural network

- x = input vector $\longrightarrow (1 \times d_i)$
- W^1, W^2 = weight matrices $\rightarrow d_i \times d_h, d_h \times d_o$
- b^1, b^2 = bias vectors
- $\hat{y} = \text{softmax}(\tilde{y}) = \left[\frac{e^{\tilde{y}_1}}{\sum_j e^{\tilde{y}_j}}, \frac{e^{\tilde{y}_2}}{\sum_j e^{\tilde{y}_j}}, \dots \right]$

$$\tilde{y} = hW^2 + b^2$$

$$h = \sigma(z) = \frac{1}{1+e^{-z}}$$

$$z = xW^1 + b^1$$

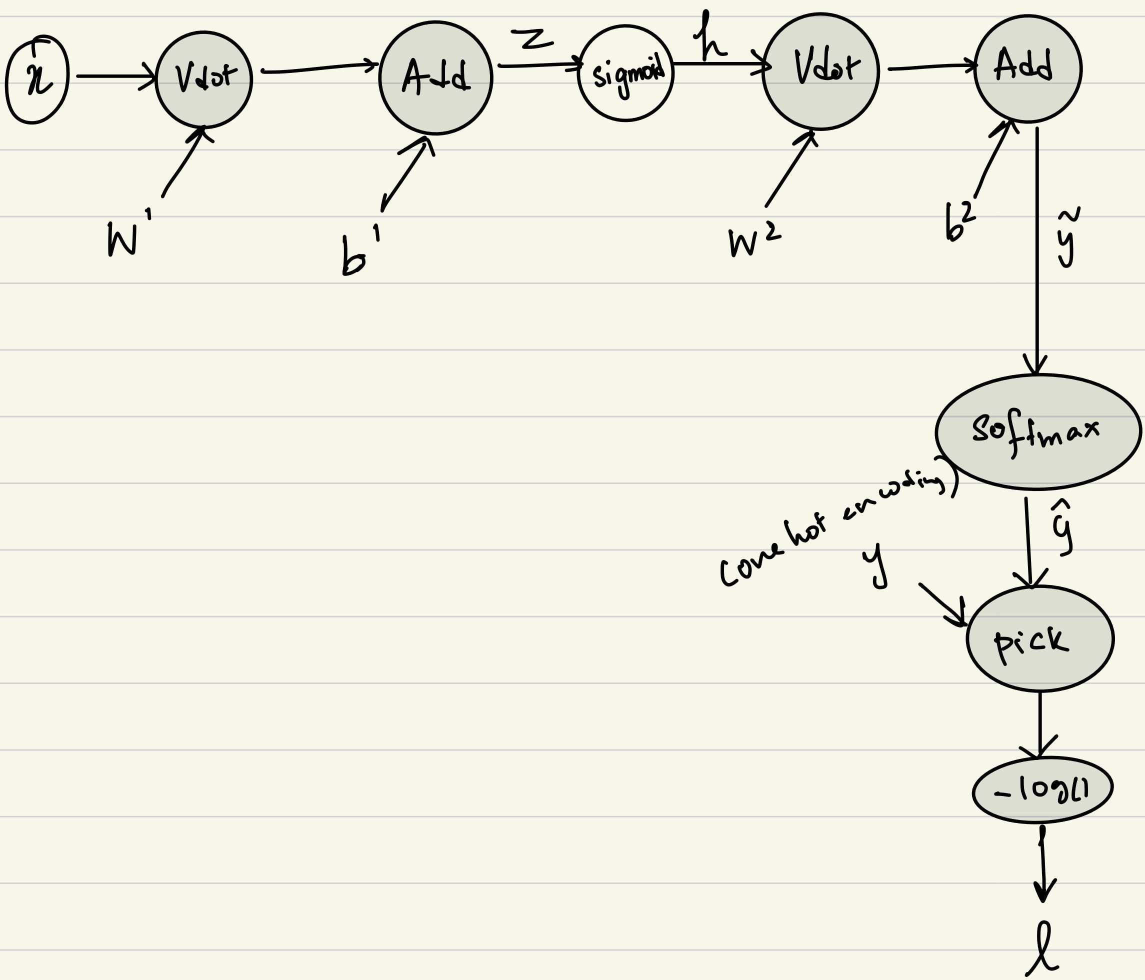
$y \rightarrow$ one hot vector encoding the correct class

- $l = -y \cdot \log \hat{y} \rightarrow l$ = cross entropy loss

$u_{m \times n} \rightarrow$ matrix $\Rightarrow u = \begin{bmatrix} u_{11} & u_{12} & \cdots & u_{1n} \\ \vdots & & & \vdots \\ u_{m1} & \cdots & & u_{mn} \end{bmatrix}$

gradient of loss wrt u $\Rightarrow d(u) = \nabla_u l = \begin{bmatrix} \partial l/\partial u_{11} & \cdots & \partial l/\partial u_{1n} \\ \vdots & & \\ \partial l/\partial u_{m1} & \cdots & \partial l/\partial u_{mn} \end{bmatrix}$

a) computation graph for n/w



b) Derive an expression for $d(\hat{y}_i)$

$d(\hat{y})_i \rightarrow$ gradient of loss wrt $\hat{y}_i$

$$\hat{y} = \text{softmax}(\tilde{y}) = \left[\frac{\exp(\tilde{y}_1)}{\sum_j \exp(\tilde{y}_j)}, \dots \right]$$

$d(\hat{y}_i) = \nabla_{\hat{y}}$ loss for some i^{th} node.

$$l = -y \log \hat{y} = -\log \hat{y}_t$$

$\hookrightarrow$ where t = correct class.

$$\therefore \quad d(\hat{y})_i = \begin{cases} 0 & \text{if } i \neq t \\ \dfrac{\partial(-\log \hat{y}_t)}{\partial(\hat{y}_t)} & \text{if } i = t \end{cases}$$

here $t \rightarrow$ actual class as in y's one hot encoding

c) $d(\tilde{y})_i \rightarrow \nabla_{\tilde{y}}$ loss for node 'i'

$\tilde{y} = hW^2 + b^2$

$h = \frac{1}{1+e^{-z}}$

$z = xW^1 + b^1$

$\hat{y} = \text{softmax}(\tilde{y}) = \left[\frac{e^{\tilde{y}_1}}{\sum_j e^{y_j}} \cdots \right]$

$l = -y \cdot \log(\hat{y})$

$\because \; y = [0\;0\;0 \cdots 1 \cdots 0\;0\;0]$

$\uparrow$

for the actual class. [say 't']

$\therefore \; l = -\log(\hat{y}_t) = -\log\left(\frac{e^{\tilde{y}_t}}{\sum_j e^{\tilde{y}_j}}\right)$

we need to diff $A(z_i) = \frac{e^{z_i}}{\sum_j e^{z_j}} \longrightarrow \frac{\partial A(z_i)}{\partial x}$

where x = some arbitary variable

$$\frac{\partial A(z_i)}{\partial x} = A(z_i)\left(1 - A(z_i)\right)\frac{\partial z_i}{\partial x} - A(z_i)\sum_j A(z_j)\frac{\partial z_j}{\partial x}$$

here $\rightarrow A(\tilde{y}_t) \rightarrow \dfrac{e^{\tilde{y}_t}}{\sum_j e^{\tilde{y}_j}} = \hat{y}_t$

$$\frac{\partial \hat{y}_t}{\partial x} = \hat{y}_t(1-\hat{y}_t)\frac{\partial \tilde{y}_t}{\partial x} - \hat{y}_t \sum \hat{y}_j \frac{\partial \tilde{y}_j}{\partial x}$$

$\therefore$

$$\frac{\partial L}{\partial \tilde{y}_i} = d(\hat{y})_i\,\hat{y}_i - \sum_j d(\hat{y})_j\,\hat{y}_j\,\hat{y}_i$$

d) derive $d(h)_i$ $\qquad z = xW^1 + b^1$

$$h = \sigma(z) = \frac{1}{1+\exp(-z)} = \frac{1}{1+e^{-z}}$$

$d(h)_i \longrightarrow \nabla_h$ loss for some i

Using chain rule

$$\underbrace{\frac{\partial L}{\partial \tilde{y}_i}}_{\text{from part (c)}} \cdot \frac{\partial \tilde{y}_i}{\partial h_i} = \frac{\partial L}{\partial h_i}$$

$$\tilde{y}_i = h_i W^2 + b^2 \rightarrow \therefore \frac{\partial \tilde{y}_i}{h_i} = W^2$$

$$d(h)_i = \left[d(\hat{y})_i \, \hat{y}_i - \textstyle\sum_j d(\hat{y})_j \, \hat{y}_j \, \hat{y}_i \right] W^2$$

(e) $d(z_i) = \frac{\partial L}{\partial h_i} \cdot \frac{\partial h_i}{\partial z_i}$ (Chain rule)

$\frac{\partial h_i}{\partial z_i} \Rightarrow h = \frac{1}{1+e^{-z_i}} = (1+e^{-z_i})^{-1}$

$\Rightarrow \frac{e^{-z_i}}{(e^{z_i}+1)^2}$

$\therefore d(z_i) = \frac{e^{-z_i} \cdot W^2}{(e^{z_i}+1)^2} \left[d(\hat{y})_i \, \hat{y}_i - \sum_j d(\hat{y})_j \, \hat{y}_j \, \hat{y}_i \right]$

Q2) Python code

Q2) Final training:

Accuracy **= 0.9648**

Loss **= 0.1137**

```
Append <VDot> to the computational graph
Append <Add> to the computational graph
Append <Sigmoid> to the computational graph
Append <VDot> to the computational graph
Append <Add> to the computational graph
Append <SoftMax> to the computational graph
Append <Aref> to the computational graph
Append <Log> to the computational graph
Append <Mul> to the computational graph
Append <Accuracy> to the computational graph

100% 30000/30000 [00:43<00:00, 727.56it/s]

Epoch 0: train loss = 0.3166, accy = 0.9089, [51.579 secs]

100% 30000/30000 [00:44<00:00, 1113.19it/s]

Epoch 1: train loss = 0.2438, accy = 0.9297, [53.278 secs]

100% 30000/30000 [00:43<00:00, 660.52it/s]

Epoch 2: train loss = 0.1952, accy = 0.9431, [50.191 secs]

100% 30000/30000 [00:43<00:00, 877.23it/s]

Epoch 3: train loss = 0.1607, accy = 0.9534, [51.942 secs]

100% 30000/30000 [00:42<00:00, 713.60it/s]

Epoch 4: train loss = 0.1353, accy = 0.9611, [51.263 secs]

100% 30000/30000 [00:42<00:00, 655.62it/s]

Epoch 5: train loss = 0.1162, accy = 0.9667, [51.927 secs]

100% 30000/30000 [00:44<00:00, 682.43it/s]

Epoch 6: train loss = 0.1013, accy = 0.9715, [53.083 secs]

100% 30000/30000 [00:43<00:00, 705.79it/s]

Epoch 7: train loss = 0.0892, accy = 0.9747, [52.195 secs]

100% 30000/30000 [00:43<00:00, 696.35it/s]

Epoch 8: train loss = 0.0791, accy = 0.9779, [51.905 secs]

100% 30000/30000 [00:43<00:00, 726.13it/s]

Epoch 9: train loss = 0.0706, accy = 0.9807, [52.436 secs]
```

```
# After 10 epochs of training, you should expect an accuracy over 95% and loss around 0.1
accy, loss = model.eval(test_x, test_y)
print("Test accuracy = %.4f, Loss = %.4f" % (accy, loss))
```

```
Test accuracy = 0.9648, Loss = 0.1137
```