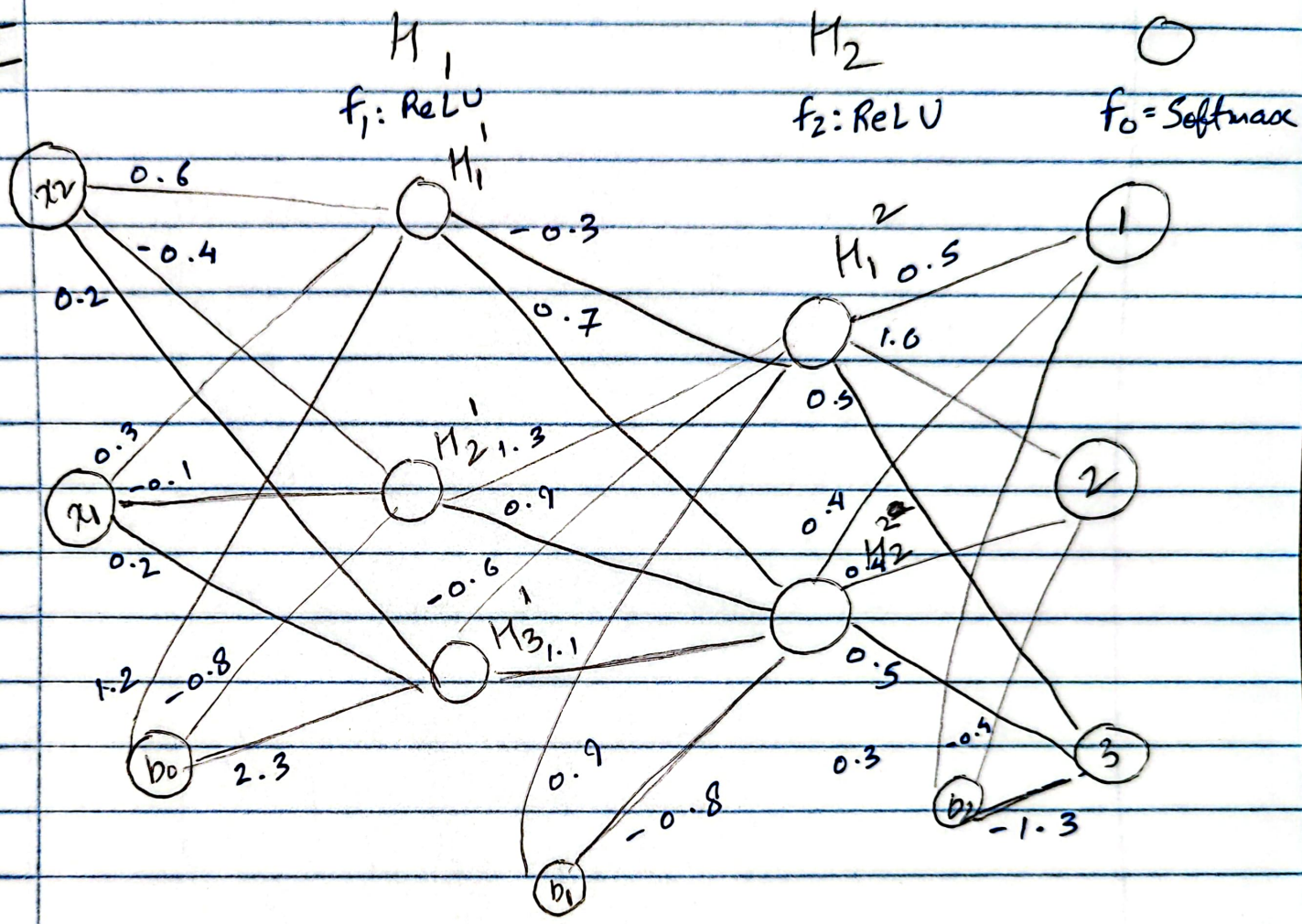
Machine learning & AI
Assignment - 2

Shubh Gaur (23200555)

1.



→ $(x_2, x_1) = (-0.7, 0.1)$

→ calculating inputs for H_1 neurons:

$$\begin{bmatrix} H_{1i}^1 \\ H_{2i}^1 \\ H_{3i}^1 \end{bmatrix} = \begin{bmatrix} 0.6 & 0.3 \\ -0.4 & -0.1 \\ 0.2 & 0.2 \end{bmatrix} \begin{bmatrix} -0.7 \\ 0.1 \end{bmatrix} + \begin{bmatrix} 1.2 \\ -0.8 \\ 2.3 \end{bmatrix}$$

w_1 x b_0

$$\begin{bmatrix} H_{1i}' \\ H_{2i}' \\ H_{3i}' \end{bmatrix} = \begin{bmatrix} 0.81 \\ -0.53 \\ 2.18 \end{bmatrix}$$

H_{1i}' Applying ReLU activation fxn to H_{1i}'

$$f_1(H_{1i}') = \begin{bmatrix} 0.81 \\ 0 \\ 2.18 \end{bmatrix} = H^1 = \begin{bmatrix} H_1' \\ H_2' \\ H_3' \end{bmatrix}$$

→ Calculating inputs for neurons in H_2

$$\begin{bmatrix} H_{1i}^2 \\ H_{2i}^2 \end{bmatrix} = \begin{bmatrix} -0.3 & 1.3 & -0.6 \\ 0.7 & 0.9 & 1.1 \end{bmatrix} \begin{bmatrix} H_1' \\ H_2' \\ H_3' \end{bmatrix} + \begin{bmatrix} 0.9 \\ -0.8 \end{bmatrix}$$

W_2 H^1 $b1$

$$= \begin{bmatrix} -0.3 & 1.3 & -0.6 \\ 0.7 & 0.9 & 1.1 \end{bmatrix} \begin{bmatrix} 0.81 \\ 0 \\ 2.18 \end{bmatrix} + \begin{bmatrix} 0.9 \\ -0.8 \end{bmatrix}$$

$$\begin{bmatrix} H_{1i}^2 \\ H_{2i}^2 \end{bmatrix} = \begin{bmatrix} -0.651 \\ 2.165 \end{bmatrix}$$

H_{1i}^2

Applying ReLU activation fxn to H^2_i

$$f_2(H^2_i) = \begin{bmatrix} 0 \\ 2.165 \end{bmatrix} = H^2$$

→ calculating inputs for output layer neurons

$$\begin{bmatrix} O_{1i} \\ O_{2i} \\ O_{3i} \end{bmatrix} = \begin{bmatrix} 0.5 & 0.4 \\ 1.0 & 0.4 \\ 0.5 & 0.5 \end{bmatrix} \begin{bmatrix} 0 \\ 2.165 \end{bmatrix} + \begin{bmatrix} 0.3 \\ -0.4 \\ -1.3 \end{bmatrix}$$

O_i = W_O (H^2) + b_2

↳ weights to output layer

$$= \begin{bmatrix} 1.166 \\ 0.466 \\ -0.2175 \end{bmatrix}$$

→ Applying softmax activation fxn to inputs of output layer

$$f_o(O_i) = \begin{bmatrix} 0.572 \\ 0.284 \\ 0.143 \end{bmatrix}$$

2. Since, we have already calculated inputs to H_1, we can just subtract b_0 from them & reapply the activation fxn to the new input

$$H_1^1 i' = H_1^1 i - b_0$$

new input

$$= \begin{bmatrix} 0.81 - 1.2 \\ -0.53 - (-0.8) \\ 2.18 - 2.3 \end{bmatrix}$$

$$H_1^1 i' = \begin{bmatrix} -0.39 \\ 0.27 \\ -0.12 \end{bmatrix}$$

Applying ReLU to $H'i'$

$$f_1(H'i') = \begin{bmatrix} 0 \\ 0.27 \\ 0 \end{bmatrix}$$

H'

→ Calculating inputs for neurons in H_2

$$\underbrace{\begin{bmatrix} H_1^2 i \\ H_2^2 i \end{bmatrix}}_{H_i^2} = \underbrace{\begin{bmatrix} -0.3 & 1.3 & -0.6 \\ 0.7 & 0.9 & 1.1 \end{bmatrix}}_{W_2} \underbrace{\begin{bmatrix} 0 \\ 0.27 \\ 0 \end{bmatrix}}_{H'} + \underbrace{\begin{bmatrix} 0.9 \\ -0.8 \end{bmatrix}}_{b_1}$$

$$\begin{bmatrix} H_1^2 i \\ H_2^2 i \end{bmatrix} = \begin{bmatrix} 1.251 \\ -0.557 \end{bmatrix}$$

Applying ReLU to $H^2 i$

$$f_2(H^2 i) = \begin{bmatrix} 1.251 \\ 0 \end{bmatrix}$$

H^2

→ Calculating inputs for output layer

$$\begin{bmatrix} O_1 i \\ O_2 i \\ O_3 i \end{bmatrix} = \begin{bmatrix} 0.5 & 0.4 \\ 1.0 & 0.4 \\ 0.5 & 0.5 \end{bmatrix} \begin{bmatrix} 1.251 \\ 0 \end{bmatrix} + \begin{bmatrix} 0.3 \\ -0.4 \\ -1.3 \end{bmatrix}$$

O_i $\quad$ W_0 $\quad$ H^2 $\quad$ b_2

$$= \begin{bmatrix} 0.9255 \\ 0.851 \\ -0.6745 \end{bmatrix}$$

Applying softmax activation fxn to inputs of output layer.

$$f_o(Oi) = \begin{bmatrix} 0.469 \\ 0.436 \\ 0.095 \end{bmatrix}$$

Assignment 2

Machine Learning & AI

Shubh Gaur - 23200555

Exercise 1

2

Case 1: bias term included in input layer

The Probabilities pertaining to each class label in this case: $\begin{bmatrix}\Pr(A)\\ \Pr(B)\\ \Pr(C)\end{bmatrix} = \begin{bmatrix}0.572\\ 0.284\\ 0.143\end{bmatrix}$

In this case the probability for class **A** is the highest.

Case 2: bias term excluded from input layer

The Probabilities pertaining to each class label in this case: $\begin{bmatrix}\Pr(A)\\ \Pr(B)\\ \Pr(C)\end{bmatrix} = \begin{bmatrix}0.469\\ 0.436\\ 0.095\end{bmatrix}$

In this case as well the probability for class **A** is the highest.

However, in case 1 it was far easier to distinguish between classes A and B as clear from the difference in probabilities than case 2 where the probabilities for classes A and B are closer.

Since, bias was removed from the input layer, the ability of classification model to learn complex patterns in data was hindered because bias terms are significant in shifting activation functions which is crucial for modelling the data more flexibly leading to better generalization to unseen data.

3

Complexity hyperparameter is being tuned in the model which is usually associated with L1 or L2 regularization.

It is evident from the first graph that the model has achieved a good fit(ideal) as loss vs epoch curve shows a smooth and aligned reduction in loss between train and validation sets as the number of epochs are increased.

From the second graph, the model seems underfitted because even though the loss vs epoch curve for training and validation sets are aligned,some fluctuations(non-smooth curve) can be observed as number of epochs increase.

In the third graph, it is evident that the model is overfitted because the training loss curve is not in alignment with the validation loss curve which which seems to increase after reaching a minimum.

Complexity parameter is used to control the fit of the model. As the complexity parameter increases, the penalization increases which leads to simpler models and the capability to fit the training data decreases. Moreover, as the complexity parameter decreases, the penalization decreases and the model is able to fit the training data well resulting in a complex model. Therefor, a balanced value for the complexity hyperparameter is preferred which can be deduced through hyperparameter tuning using cross validation techniques and analysing the loss curves on training and validation sets.

4

We will be calculating the classification accuracy using the given four observations.

Observation 1: True Label = **1** ; Predicted Label = **1**

Observation 2: True Label = **2** ; Predicted Label = **3**

Observation 3: True Label = **2** ; Predicted Label = **3**

Observation 4: True Label = **3** ; Predicted Label = **3**

Classification Accuracy $= \frac{2}{4} \times 100 = 50\%$

The average loss can be computed by calculating the cross entropy loss for each observation and averaging it out. The formula for cross entropy loss is given by: $L_i = -\sum_{j=1}^{X} y_{ij} \log(\hat{y}_{ij})$ where, X: number of classes $y_{ij} = 0, 1$: if true class label then value is taken as 1 otherwise 0. $\hat{y}_{ij}$: predicted probability of class j in the ith observation.

Observation 1: True Label = **1** ; Loss $= -\log(1 \times 0.62 + 0 \times 0.35 + 0 \times 0.03) = -\log(0.62)$

Observation 2: True Label = **2** ; Loss $= -\log(0 \times 0.12 + 1 \times 0.08 + 0 \times 0.80) = -\log(0.08)$

Observation 3: True Label = **2** ; Loss $= -\log(0 \times 0.07 + 1 \times 0.40 + 0 \times 0.07) = -\log(0.40)$

Observation 4: True Label = **3** ; Loss $= -\log(0 \times 0.22 + 0 \times 0.07 + 1 \times 0.71) = -\log(0.71)$

Average Loss $= -\frac{\log(0.62)+\log(0.08)+\log(0.40)+\log(0.71)}{4} = -\frac{\log(0.0140864)}{4} = 1.0656$

Exercise 2

1

A

Since the **input_shape** in the first convolutional layer is $(256, 256, 3)$ we can deduce that the images have three color channels which are most likely RGB. Also, it is evident that images are of size 256×256 pixels.

B

The depth of convolutional layers can be evaluated by looking at the number of filters for each convolutional layer.

For convolutional layer 1: $depth = 64$

For convolutional layer 2: $depth = 128$

For convolutional layer 3: $depth = 128$

C

The size of the filters can be deduced by looking at the **kernel_size**.

For convolutional layer 1: $size_{filter} = 6 \times 6$

For convolutional layer 2: $size_{filter} = 4 \times 4$

For convolutional layer 3: $size_{filter} = 2 \times 2$

D

The number of batches processed in each training epoch can be calculated using the **number of training instances** and **batch size** as follows: $N = \frac{Total\ Training\ Instances}{Batch\ Size} = \frac{15725}{185} = 85$

Therefore, **85** batches are processed in each training epoch.

E

ReLU activation function is being used for all the convolutional layers. On the other hand, **Softmax** activation function is being used for the output layer.

F

L2 regularization is applied to the first dense layer(fully connected) with weight **0.2**.

2

A

To calculate the number of parameters for the second convolutional layer, we utilize the formula: $N_{params} = (k \times k \times D_{l-1} + 1) \times D_l$

Here, k represents the kernel size, D_{l-1} is the depth of the previous layer, and D_l is the depth of the current layer.

Given:

- Kernel size, $k = 4$
- Depth of previous layer, $D_{l-1} = 64$

- Depth of current layer, $D_l = 128$
- Plugging in the values, we get: $N_{params} = (4 \times 4 \times 64 + 1) \times 128 = 131200$

Hence, the number of parameters for the **2nd** convolutional layer = 131200

B

For the first dense layer, we need to calculate the total number of parameters for which we'll be using the formula: $P = n \times (m + 1)$

Here, m is the number of neurons in the previous layer and n is the number of neurons in the current layer.

First, let's calculate the output shape for each convolutional layer using the formula:

Output Shape $= \frac{\text{Input Shape} - \text{Filter Shape} + 2 \times \text{Padding}}{\text{Stride}} + 1$

Assuming zero padding and a stride of 1 and applying the specified formula on:

1. **Convolutional Layer 1**
   - $OutputShape = 256 - 6 + 2 \times 0 + 1 = 251 \rightarrow (251 \times 251 \times 64)$
   - After max pooling (2×2), the output shape reduces to half and becomes $125 \times 125 \times 64$.
2. **Convolutional Layer 2**:
   - We have a filter depth of 128 for this layer.
   - $OutputShape = 125 - 4 + 2 \times 0 + 1 = 122 \rightarrow (122 \times 122 \times 128)$
   - After max pooling (2×2), the output shape again reduces to half and becomes $61 \times 61 \times 128$.
3. **Convolutional Layer 3**:
   - We again have a filter depth of 128 for this layer.
   - $OutputShape = 61 - 2 + 2 \times 0 + 1 = 60 \rightarrow (60 \times 60 \times 128)$
   - After max pooling (2×2), the output shape decreases by half and becomes $30 \times 30 \times 128$.

Note:Padding and stride are assumed to be 0 and 1 because of the absence of explicit definition in the code segment, therefore, defaulting to commonly used configurations for simplicity and consistency.

Finally, the output from the last convolutional layer goes through the flatten layer, resulting in an output shape of $30 \times 30 \times 128 = 115200 = m$.

Now, we can calculate the total number of parameters for the first dense layer:

$n = 64$ (given)

$P = 64 \times (115200 + 1) = 64 \times (115201) = 7372864$

Hence, the number of parameters for the **1st** dense layer = 7372864