Assignment 6

Durbasmriti Saha
EE708: Fundamentals of Data Science and Machine Intelligence

April 25, 2025

1) An odd integer times an odd integer results in an odd integer. The logistic regression classifier is usually preferred over the classic perceptron for the following reasons:

Probabilistic Output:

Logistic regression outputs a probability between 0 and 1 using the sigmoid function, making it easy to interpret and threshold as needed.

The perceptron gives binary outputs (0 or 1), offering no confidence score.

Differentiability and Optimization:

Logistic regression uses a smooth, convex loss function (cross-entropy or log-loss), which allows for gradient-based optimization techniques like stochastic gradient descent (SGD).

Perceptron uses a non-differentiable step function, which makes it less amenable to standard gradient descent.

Convergence Guarantees:

The perceptron algorithm only converges if the data is linearly separable; otherwise, it may oscillate or fail.

Logistic regression doesn't need perfect separability and still finds a best-fit solution using likelihood maximization.

Better Generalization:

Logistic regression naturally incorporates regularization (L1 or L2), helping it generalize better and avoid overfitting.

The classic perceptron lacks such mechanisms.

To make a perceptron behave like a logistic regression classifier:

1.Replace the Step Function with a Sigmoid Activation Function: $y = \sigma(w^T x + b) = \frac{1}{1+e^{-(w^T x + b)}}$ This gives probabilistic output like logistic regression.

2. Change the Loss Function:

Use cross-entropy loss instead of the perceptron loss: $L(y, \hat{y}) = -y \log(\hat{y}) - (1-y) \log(1-\hat{y})$

3. Train Using Gradient Descent:

Update weights using the gradient of the cross-entropy loss with respect to the weights, just like in logistic regression.

2) 1. Email Classification (Spam or Ham):
Number of output neurons: 1

It's a binary classification problem (spam = 1, ham = 0), so only one neuron is needed to output the probability of one class (e.g., spam).

Activation function: Sigmoid

The sigmoid activation squashes the output to a range between 0 and 1, which can be interpreted as the probability of the email being spam.

2.MNIST Digit Classification:
Number of output neurons: 10

MNIST has 10 classes (digits 0 through 9), so you need one output neuron per class.

Activation function: Softmax

The softmax function ensures that the outputs are non-negative and sum to 1, making them interpretable as a probability distribution over the 10 digits.

3) a. Shape of the input matrix X:
Each input has 10 features, and there are m samples. Hence, X has shape (m, 10)
b. Shapes of the hidden layer's weight matrix Wh and bias vector bh:
Wh maps input (10) → hidden (50)

Wh shape: (10, 50)

bh is added to each row of the output: shape is (1, 50)

c. Shapes of the output layer's weight matrix Wo and bias vector bo:
Wo maps hidden (50) → output (3)

Wo shape: (50, 3)

bo shape: (1, 3)

d. Shape of the network's output matrix Y:
Final output = 3 neurons

For m samples, the output matrix has shape (m, 3)

e. Equation to compute the network's output Y:

Let's use ReLU as the activation function:

$$\mathrm{ReLU}(z) = \max(0, z)$$

Then the forward pass is:

$$H = \mathrm{ReLU}(X \cdot W_h + b_h)$$

$$Y = \mathrm{ReLU}(H \cdot W_o + b_o)$$

4) The cross-entropy error function for a logistic sigmoid output neuron is:

$$E(a) = -\sum_{k=1}^{n} \left[t_k \ln y_k + (1 - t_k) \ln(1 - y_k) \right] \tag{1}$$

Where:

- t_k is the target label (0 or 1)
- y_k is the predicted output of the neuron for sample k
- $y_k = \sigma(a_k) = \frac{1}{1+e^{-a_k}}$ — the logistic sigmoid activation function
- a_k is the net input to the sigmoid neuron (i.e., before activation)

Chain Rule:

$$\frac{\partial E}{\partial a_k} = \frac{\partial E}{\partial y_k} \cdot \frac{\partial y_k}{\partial a_k}$$

From equation (1):

$$E_k = -\left[t_k \ln y_k + (1 - t_k)\ln(1 - y_k)\right]$$

Now, the derivative of the error with respect to the output y_k is:

$$\frac{\partial E_k}{\partial y_k} = -\left[\frac{t_k}{y_k} - \frac{1 - t_k}{1 - y_k}\right]$$

Derivative of sigmoid with respect to input a_k:

$$\frac{dy_k}{da_k} = y_k(1 - y_k)$$

Combining using the chain rule:

$$\frac{\partial E_k}{\partial a_k} = \left[-\left(\frac{t_k}{y_k} - \frac{1 - t_k}{1 - y_k}\right)\right] \cdot y_k(1 - y_k)$$

Now,

$$\frac{\partial E_k}{\partial a_k} = y_k - t_k \tag{2}$$

– This is the derivative of the cross-entropy error wrt the input of a sigmoid neuron.

5) Applying the filter to each 2×2 submatrix of the input and compute the dot product:

Position (0,0):

$$\begin{bmatrix} 2 & 5 \\ 0 & 6 \end{bmatrix} \Rightarrow (-2)(2) + (0)(5) + (4)(0) + (6)(6) = -4 + 0 + 0 + 36 = 32$$

Position (0,1):

$$\begin{bmatrix} 5 & -3 \\ 6 & 0 \end{bmatrix} \Rightarrow (-2)(5) + (0)(-3) + (4)(6) + (6)(0) = -10 + 0 + 24 + 0 = 14$$

Position (0,2):

$$\begin{bmatrix} -3 & 0 \\ 0 & -4 \end{bmatrix} \Rightarrow (-2)(-3) + (0)(0) + (4)(0) + (6)(-4) = 6 + 0 + 0 - 24 = -18$$

Position (1,0):

$$\begin{bmatrix} 0 & 6 \\ -1 & -3 \end{bmatrix} \Rightarrow (-2)(0) + (0)(6) + (4)(-1) + (6)(-3) = 0 + 0 - 4 - 18 = -22$$

Position (1,1):

$$\begin{bmatrix} 6 & 0 \\ -3 & 0 \end{bmatrix} \Rightarrow (-2)(6) + (0)(0) + (4)(-3) + (6)(0) = -12 + 0 - 12 + 0 = -24$$

Position (1,2):

$$\begin{bmatrix} 0 & -4 \\ 0 & 2 \end{bmatrix} \Rightarrow (-2)(0) + (0)(-4) + (4)(0) + (6)(2) = 0 + 0 + 0 + 12 = 12$$

Position (2,0):

$$\begin{bmatrix} -1 & -3 \\ 5 & 0 \end{bmatrix} \Rightarrow (-2)(-1) + (0)(-3) + (4)(5) + (6)(0) = 2 + 0 + 20 + 0 = 22$$

Position (2,1):

$$\begin{bmatrix} -3 & 0 \\ 0 & 0 \end{bmatrix} \Rightarrow (-2)(-3) + (0)(0) + (4)(0) + (6)(0) = 6 + 0 + 0 + 0 = 6$$

Position (2,2):

$$\begin{bmatrix} 0 & 2 \\ 0 & 3 \end{bmatrix} \Rightarrow (-2)(0) + (0)(2) + (4)(0) + (6)(3) = 0 + 0 + 0 + 18 = 18$$

Final Output (3x3):

$$\begin{bmatrix} 32 & 14 & -18 \\ -22 & -24 & 12 \\ 22 & 6 & 18 \end{bmatrix}$$

6) Input: Original activations:

$$(x_1, x_2, x_3, x_4)$$

After zero-padding on both sides (1 zero on each side):

$$(0, x_1, x_2, x_3, x_4, 0)$$

Let the filter (length 3) be:

$$(w_1, w_2, w_3)$$

Stride = 2 This means the filter moves 2 positions at a time.

Computing Convolution: We slide the filter over the padded input and apply dot product at each position.

1st window:

$$(0, x_1, x_2) \cdot (w_1, w_2, w_3) = 0 \cdot w_1 + x_1 \cdot w_2 + x_2 \cdot w_3$$

2nd window:

$$(x_2, x_3, x_4) \cdot (w_1, w_2, w_3) = x_2 \cdot w_1 + x_3 \cdot w_2 + x_4 \cdot w_3$$

3rd window:

$$(x_4, 0, 0) \cdot (w_1, w_2, w_3) = x_4 \cdot w_1 + 0 \cdot w_2 + 0 \cdot w_3$$

Final output vector:

$$\begin{bmatrix} x_1w_2 + x_2w_3 \\ x_2w_1 + x_3w_2 + x_4w_3 \\ x_4w_1 \end{bmatrix}$$

Let the padded input vector be:

$$x = \begin{bmatrix} 0 \\ x_1 \\ x_2 \\ x_3 \\ x_4 \\ 0 \end{bmatrix}$$

We can express the convolution output as a matrix-vector multiplication:

$$y = A \cdot x$$

Where the matrix A is:

$$A = \begin{bmatrix} 0 & w_2 & w_3 & 0 & 0 & 0 \\ 0 & 0 & w_1 & w_2 & w_3 & 0 \\ 0 & 0 & 0 & 0 & w_1 & 0 \end{bmatrix}$$

Then,

$$y = \begin{bmatrix} 0 & w_2 & w_3 & 0 & 0 & 0 \\ 0 & 0 & w_1 & w_2 & w_3 & 0 \\ 0 & 0 & 0 & 0 & w_1 & 0 \end{bmatrix} \cdot \begin{bmatrix} 0 \\ x_1 \\ x_2 \\ x_3 \\ x_4 \\ 0 \end{bmatrix} = \begin{bmatrix} x_1w_2 + x_2w_3 \\ x_2w_1 + x_3w_2 + x_4w_3 \\ x_4w_1 \end{bmatrix}$$

2. Input:

$$z = \begin{bmatrix} x_1 \\ x_2 \end{bmatrix}$$

Filter:

$$(w_1, w_2, w_3)$$

Stride: 2 Activation Function: Identity (linear)

Transposed convolution:
Each input value spreads over the output vector with stride 2:

- x_1 spreads over positions 0, 1, 2:

$$\begin{aligned} x_1 w_1 &\to \text{position } 0 \\ x_1 w_2 &\to \text{position } 1 \\ x_1 w_3 &\to \text{position } 2 \end{aligned}$$

- x_2 spreads over positions 2, 3, 4:

$$\begin{aligned} x_2 w_1 &\to \text{position 2 (added to previous)} \\ x_2 w_2 &\to \text{position } 3 \\ x_2 w_3 &\to \text{position } 4 \end{aligned}$$

- Position 5 gets 0.

Final Output:

$$\begin{bmatrix} x_1 w_1 \\ x_1 w_2 \\ x_1 w_3 + x_2 w_1 \\ x_2 w_2 \\ x_2 w_3 \\ 0 \end{bmatrix}$$

Matrix Formulation:
Recall the convolution matrix A:

$$A = \begin{bmatrix} 0 & w_2 & w_3 & 0 & 0 & 0 \\ 0 & 0 & w_1 & w_2 & w_3 & 0 \\ 0 & 0 & 0 & 0 & w_1 & 0 \end{bmatrix}$$

Then, the transposed convolution uses A^T:

$$A^T = \begin{bmatrix} 0 & 0 & 0 \\ w_2 & 0 & 0 \\ w_3 & w_1 & 0 \\ 0 & w_2 & 0 \\ 0 & w_3 & w_1 \\ 0 & 0 & 0 \end{bmatrix}$$

Let the input be padded as:

$$z = \begin{bmatrix} x_1 \\ x_2 \\ 0 \end{bmatrix}$$

Output:

$$A^T \cdot z = \begin{bmatrix} 0 \cdot x_1 + 0 \cdot x_2 + 0 \cdot 0 \\ w_2 \cdot x_1 + 0 \cdot x_2 + 0 \cdot 0 \\ w_3 \cdot x_1 + w_1 \cdot x_2 + 0 \cdot 0 \\ 0 \cdot x_1 + w_2 \cdot x_2 + 0 \cdot 0 \\ 0 \cdot x_1 + w_3 \cdot x_2 + w_1 \cdot 0 \\ 0 \end{bmatrix} = \begin{bmatrix} 0 \\ x_1 w_2 \\ x_1 w_3 + x_2 w_1 \\ x_2 w_2 \\ x_2 w_3 \\ 0 \end{bmatrix}$$

This is same as earlier transposed convolution output.

7) Given input,

$$X = \begin{bmatrix} 1 & 1 & 2 & 1 \\ 2 & 1 & 2 & 0 \\ 3 & 2 & 4 & 1 \\ 1 & 1 & 2 & 1 \end{bmatrix}$$

The target Y' must also be the same as the X.
And for the encoder part, the output at any step can be written as a matrix of order mxn, where m = number of rows in kernel and n = number of kernel. and the matrix will be calculated as Relu(WX + b) where W = weight matrix and X is the input matrix, b is the bias vector.

Encoded output :

$$\begin{bmatrix} 1 & 0 & 0 & 1 & 0 \\ 0 & 1 & 1 & 0 & 0 \\ -1 & 0 & 1 & 0 & -1 \end{bmatrix} (Relu) \quad \begin{bmatrix} 2 & 3 & 4 & 2 \\ 5 & 3 & 6 & 1 \\ 1 & 0 & 1 & 0 \end{bmatrix}$$

$$\begin{bmatrix} 1 & 0 & 1 & 0 \\ -1 & 1 & 0 & 0 \end{bmatrix} (Relu) \quad \begin{bmatrix} 3 & 2 & 5 & 2 \\ 3 & 1 & 2 & 0 \end{bmatrix} (Bottleneck)$$

Here the element Q_i is basically: $Q_i = \text{ReLU}\left(\sum W_i \times I_i + b_i\right) = \text{ReLU}(1 \times 1 + 2 \times 0 + 3 \times 0 + 1 \times 1 + 0) = \text{ReLU}(2) = 2$

For decoding part, :

$$\begin{bmatrix} 1 & 0 & 0 \\ 0 & 1 & 1 \\ 1 & -1 & 0 \end{bmatrix} (ReLu) \quad \begin{bmatrix} 3 & 2 & 5 & 2 \\ 4 & 2 & 3 & 1 \\ 0 & 1 & 3 & 2 \end{bmatrix} (Decodeoutput)$$

$$\begin{bmatrix} 1 & 0 & -1 & 0 \\ 1 & -1 & 0 & 0 \\ 0 & 0 & 1 & 1 \\ 0 & 1 & 1 & -3 \end{bmatrix} (ReLu) \quad \begin{bmatrix} 3 & 1 & 2 & 0 \\ 1 & 0 & 2 & 1 \\ 1 & 2 & 4 & 3 \\ 1 & 0 & 3 & 0 \end{bmatrix} (Reconstructed \quad output)$$

and we know target Y' = X.
So, :

$$\text{Y - Y'} = \begin{bmatrix} 2 & 0 & 0 & -1 \\ -3 & -1 & 0 & 1 \\ -2 & 0 & 0 & 2 \\ 0 & -1 & 1 & -1 \end{bmatrix} \text{ (Reconstruction Loss)}$$

$$\text{and } X_2 = \begin{bmatrix} 4 & 0 & 0 & -2 \\ -6 & -2 & 0 & 2 \\ -4 & 0 & 0 & 4 \\ 0 & -2 & 2 & -2 \end{bmatrix} = \frac{\partial \mathcal{L}}{\partial Y}$$

Programming Question 1

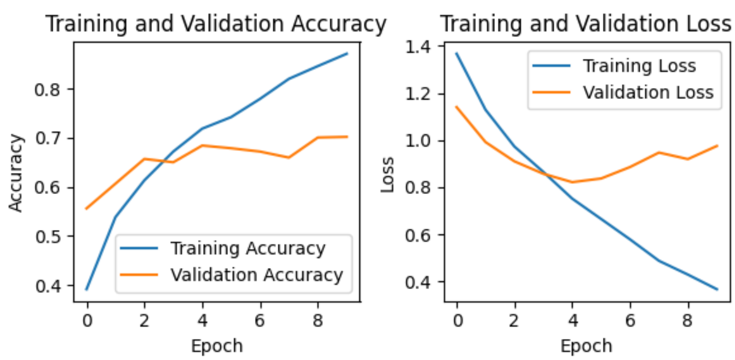

Figure 1: training and testing accuracy over epochs

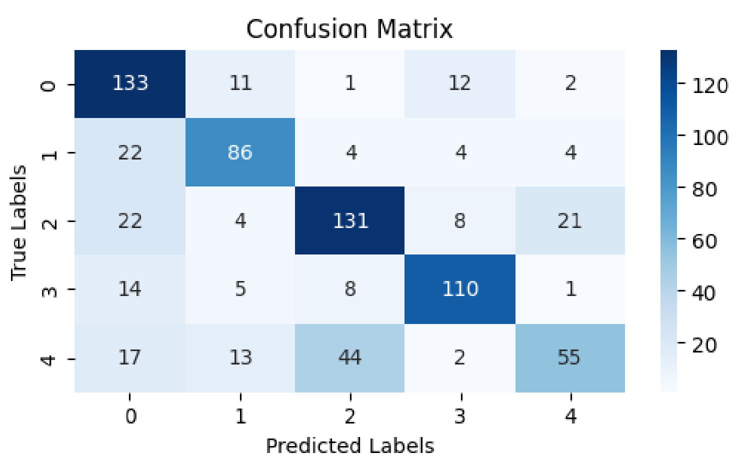

Figure 2: Confusion matrix, Question 1

Programming Question 2

Classification Report:

	precision	recall	f1-score	support
0	0.64	0.84	0.72	159
1	0.72	0.72	0.72	120
2	0.70	0.70	0.70	186
3	0.81	0.80	0.80	138
4	0.66	0.42	0.51	131
accuracy			0.70	734
macro avg	0.71	0.69	0.69	734
weighted avg	0.70	0.70	0.69	734

Figure 3: Classification Report, Question 1

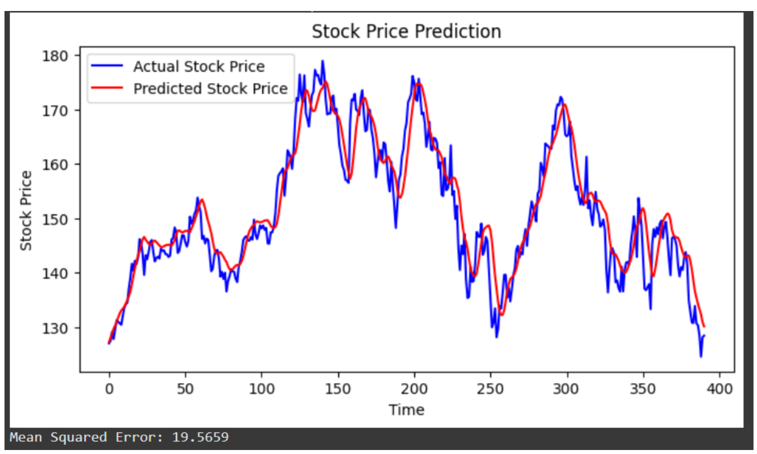

Figure 4: the actual and predicted stock prices, Question 2