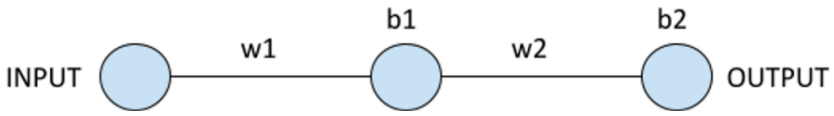
Machine Learning(CSE343)
Assignment-3

Section A - Theoretical

(a) Consider a regression problem where you have an MLP with one hidden layer (ReLU activation) and a linear output (refer to attached Fig 1). Train the network using mean squared error loss. Given a dataset with inputs [1, 2, 3] and corresponding targets [3, 4, 5], perform a single training iteration and update the weights. Assume appropriate initial weights and biases and a learning rate of 0.01.



Solution:

Network Architecture:

- Input layer: 1 input feature.
- Hidden layer: 1 unit with **ReLU activation**.
- Output layer: 1 unit with a **linear activation**.

Given Data:

- **I**nputs: [1,2,3]
- Targets: [3,4,5]
- Learning Rate(η): 0.01

Initial Assumptions:

- w1 = 1, w2 = 2.
- b1 = 1, b1 = 2.

Iteration #1 (Forward Pass):

⇒ Let the input be x_i & target be y_i

⇒ Hidden layer output :-

$$h_i = ReLU(w_1 \cdot x_i + b_1)$$
$$= \max(0, w_1 \cdot x_i + b_1)$$

⇒ Output layer output :-

$$\hat{y}_i = w_2 \cdot h_i + b_2$$

⇒ Forward Pass for Each Input :

1) For $x = 1$ —

→ $h_1 = ReLU(1 \times 1 + 1) = ReLU(2) = 2$

→ $\hat{y}_1 = 2 \cdot 2 + 2 = 6$

2) For $x = 2$ —

→ $h_2 = ReLU(1 \cdot 2 + 1) = ReLU(3) = 3$

→ $\hat{y}_2 = 2 \cdot 3 + 2 = 8$

3) For $x = 3$ —

→ $h_3 = ReLU(1 \cdot 3 + 1) = ReLU(4) = 4$

→ $\hat{y}_3 = 2 \cdot 4 + 2 = 10$

⇒ Outputs :

→ Predicted values : $\hat{y} = [6, 8, 10]$.

⇒ Mean Squarred Error (MSE) loss :

$$MSE(£) = \frac{1}{3} \sum_{i=1}^{3} (\hat{y}_i - y_i)^2$$

→ For each input :

- For $x = 1$: $(\hat{y}_1 - y_1)^2 = (6 - 3)^2 = 9$
- For $x = 2$: $(\hat{y}_2 - y_2)^2 = (8 - 4)^2 = 16$
- For $x = 3$: $(\hat{y}_3 - y_3)^2 = (10 - 5)^2 = 25$

⇒ So, The MSE is :—

$$MSE = \frac{1}{3}(9 + 16 + 25) = \frac{50}{3} \approx 16.67$$

Backward Propagation:

Backward Propagation :—

→ Gradient w.r.t. w_2 :

$$\Rightarrow \frac{\partial L}{\partial w_2} = \frac{\partial L}{\partial \hat{y}_i} \cdot \frac{\partial \hat{y}_i}{\partial w_2} \quad \text{(using the Chain Rule)}$$

$$\Rightarrow \frac{\partial L}{\partial \hat{y}_i} = \frac{1}{3} \frac{\partial \sum (\hat{y}_i - y_i)^2}{\partial \hat{y}_i} = \frac{2}{3} \sum_{i=1}^{3} (\hat{y}_i - y_i)$$

$$\Rightarrow \frac{\partial \hat{y}_i}{\partial w_2} = \frac{\partial (w_2 h_i + b_2)}{\partial w_2} = h_i$$

Thus, $\boxed{\frac{\partial L}{\partial w_2} = \frac{2}{3} \sum_{i=1}^{3} (\hat{y}_i - y_i) \cdot h_i}$

→ Gradient w.r.t. b_2 :

$\because \hat{y}_i = w_2 \cdot h_i + b_2$

$$\Rightarrow \frac{\partial \hat{y}_i}{\partial b_2} = 1$$

⇒ Using the Chain Rule :

$$\frac{\partial L}{\partial b_2} = \sum_{i=1}^{3} \frac{\partial L}{\partial \hat{y}_i} \cdot \frac{\partial \hat{y}_i}{\partial b_2} = \sum_{i=1}^{3} \left(\frac{2}{3} (\hat{y}_i - y_i) \right) \cdot 1$$

⇒ Thus, $\boxed{\frac{\partial L}{\partial b_2} = \frac{2}{3} \sum_{i=1}^{3} (\hat{y}_i - y_i)}$ **

→ Gradient w.r.t. w_1 :

$$\Rightarrow \frac{\partial L}{\partial w_1} = \sum_{i=1}^{3} \frac{\partial L}{\partial \hat{y}_i} \cdot \frac{\partial \hat{y}_i}{\partial h_i} \cdot \frac{\partial h_i}{\partial w_1}$$

$$\Rightarrow \frac{\partial \hat{y}_i}{\partial h_i} = \frac{\partial (w_2 h_i + b_2)}{\partial h_i} = w_2$$

$$\Rightarrow \frac{\partial h_i}{\partial w_1} = \frac{\partial (w_1 x_i + b_1)}{\partial w_1} = x_i \quad \text{(Assuming } w_1 x_i + b_1 > 0 \text{ i.e. } ReLU'(z) = 1\text{)}$$

$$\Rightarrow \frac{\partial L}{\partial w_1} = \sum_{i=1}^{3} \left(\frac{2}{3} (\hat{y}_i - y_i) \right) \cdot w_2 \cdot x_i$$

⇒ Thus, $\boxed{\frac{\partial L}{\partial w_1} = \frac{2}{3}\sum_{i=1}^{3}(\hat{y}_i - y_i)\cdot w_2 \cdot x_i}$

⇒ Gradient w.r.t b_1 :

$$\frac{\partial L}{\partial b_1} = \sum_{i=1}^{3}\frac{\partial L}{\partial \hat{y}_i}\cdot\frac{\partial \hat{y}_i}{\partial h_i}\cdot\frac{\partial h_i}{\partial b_1}$$

$$\Rightarrow \frac{\partial h_i}{\partial b_1} = 1 \quad (\text{for } w_1 \cdot x_i + b_1 > 0)$$

$$\Rightarrow \frac{\partial L}{\partial b_1} = \sum_{i=1}^{3}\left(\frac{2}{3}(\hat{y}_i - y_i)\right)\cdot w_2$$

⇒ Thus, $\boxed{\frac{\partial L}{\partial b_1} = \frac{2}{3}\sum_{i=1}^{3}(\hat{y}_i - y_i)\cdot w_2}$

⇒ Calculations –

• Gradient for w_2 : $\frac{\partial MSE}{\partial w_2} = \frac{\partial L}{\partial w_2}\frac{2}{3}\sum_{i=1}^{3}(\hat{y}_i - y_i)\cdot h_i$

- For $x=1$ — $(6-3)\cdot 2 = 6$
- For $x=2$ — $(8-4)\cdot 3 = 12$
- For $x=3$ — $(10-5)\cdot 4 = 20$

$$\frac{\partial L}{\partial w_2} = \frac{2}{3}(6+12+20) = \frac{2}{3}\cdot 38 = \frac{76}{3} \approx 25.33 //$$

• Gradient for b_2 : $\frac{\partial L}{\partial b_2} = \frac{2}{3}\sum_{i=1}^{3}(\hat{y}_i - y_i)$

$$\rightarrow \frac{\partial L}{\partial b_2} = \frac{2}{3}\left[(6-3)+(8-4)+(10-5)\right]$$

$$= \frac{2}{3}\left[3+4+5\right] = 8 //$$

• Gradient for w_1 : $\frac{\partial L}{\partial w_1} = \frac{2}{3}\sum_{i=1}^{3}(\hat{y}_i - y_i)\cdot w_2 \cdot x_i$

→ For $x=1$ – $(6-3)\cdot 2\cdot 1 = 6$

→ For $x=2$ – $(8-4)\cdot 2\cdot 2 = 16$

→ For $x=3$ – $(10-5)\cdot 2\cdot 3 = 30$

$\Rightarrow \frac{\partial L}{\partial w_1} = \frac{2}{3}(6+16+30) = \frac{2}{3} \cdot 52 = \frac{104}{3} \approx 34.67$

$\Rightarrow$ Gradient for b_1 : $\frac{\partial L}{\partial b_1} = \frac{2}{3}\sum_{i=1}^{3}(\hat{y}_i - y_i) \cdot w_2$

$\frac{\partial L}{\partial b_1} = \frac{2}{3}((6-3)\cdot 2 + (8-4)\cdot 2 + (10-5)\cdot 2) = \frac{2}{3}(6+8+10)$

$= \frac{2}{3}(6+8+10) = 16$

$\Rightarrow$ Updating Weights & Bias :—

$\rightarrow$ Learning Rate $\eta = 0.01$

$\Rightarrow$ Updating w_2 — $w_2 = w_2 - \eta \frac{\partial L}{\partial w_2} = 2 - 0.01 \cdot 25.33$

$= 2 - 0.2533 = 1.7467$

$\Rightarrow$ Updating b_2 —

$b_2 = b_2 - \eta \frac{\partial L}{\partial b_2}$

$= 2 - 0.01 \cdot 8$

$= 2 - 0.08$

$= 1.92$

$\Rightarrow$ Updating w_1 —

$w_1 = w_1 - \eta \frac{\partial L}{\partial w_1}$

$= 1 - 0.01 \cdot 34.67$

$= 1 - 0.3467$

$= 0.6533$

$\Rightarrow$ Update b_1 —

$b_1 = b_1 - \eta \frac{\partial L}{\partial b_1} = 1 - 0.01 \cdot 16 = 1 - 0.16 = 0.84$

$\Rightarrow$ Thus, after a single training iteration, this is the value of weights & biases:

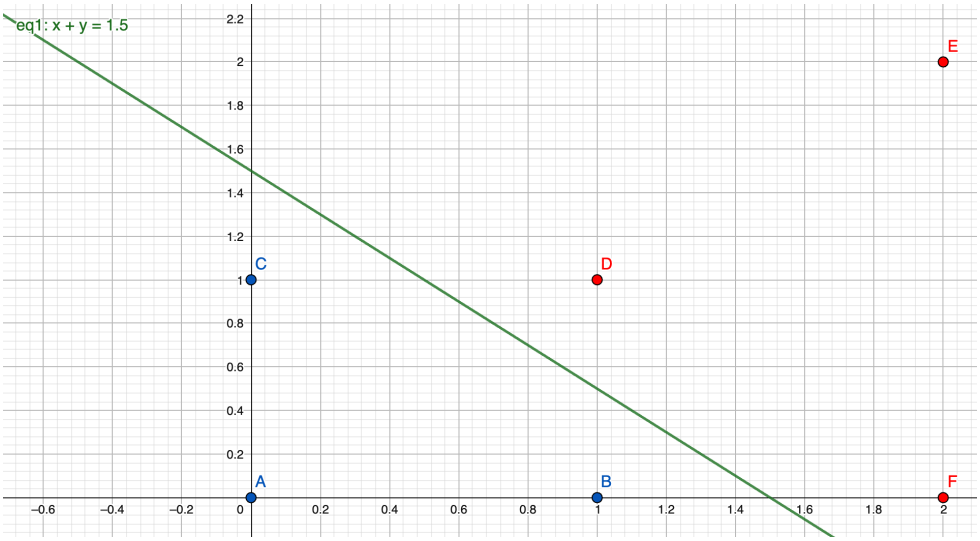
- $w_1 = 0.6533$
- $b_1 = 0.84$
- $w_2 = 1.7467$
- $b_2 = 1.92$

(b) You have the following dataset:

Class	x_1	x_2	Label
+	0	0	+
+	1	0	+
+	0	1	+
-	1	1	-
-	2	2	-
-	2	0	-

(a) Are the points linearly separable? Support your answer by plotting the points.

Solution:



Based on the visualization, it is visible that the points are linearly separable. We can see that a straight line can be drawn that divides the blue circles (class +1) from the red squares (class −1) without overlap.

Example: a line along $\mathbf{x_1 + x_2 = 1.5}$ or another similar boundary could serve as a separating line between these two classes. Thus, we conclude that this dataset is indeed linearly separable.

(b) Find out the weight vector corresponding to the maximum margin hyperplane. Also find the support vectors present.

Solution:

To find the weight vector corresponding to the maximum margin hyperplane, we will first find out the equation of the maximum margin hyperplane.

As evident from the dataset diagram, the points A and E cannot be support vectors as they are the farthest from the possible margin hyperplane.

⇒ The points that can be closest to the decision boundary based on their proximity & are most likely to be the support vectors are :

- B — (1,0) with label +.
- C — (0,1) with label +.
- D — (1,1) with label −.

⇒ We want to find the weight vector $w = (w_1, w_2)$ & the bias b for the maximum margin hyperplane.

⇒ The points B, C & D are most likely to be the support vectors because they are on the boundary between classes. Therefore, we will use these points to define the margin.

⇒ For a point $x_i = \begin{bmatrix} x_{1i} \\ x_{2i} \end{bmatrix}$ with label y_i, the margin, or the support vector condition is :—

$$\boxed{y_i(w^T x_i + b) = 1} \quad ***$$

⇒ For the Point (1,0) with label ⊕ :

$$\Rightarrow y_1(w_1 x_{11} + w_2 x_{21} + b) = 1$$
$$\Rightarrow 1 \cdot (w_1(1) + w_2(0) + b) = 1$$
$$\Rightarrow w_1 + b = 1 \quad \text{——— (1)}$$

⇒ For the Point (0,1) with label ⊕ :

$$\Rightarrow y_2(w_1 x_{12} + w_2 x_{22} + b) = 1$$
$$\Rightarrow 1 \cdot (w_1 \cdot (0) + w_2(1) + b) = 1$$
$$\Rightarrow w_2 + b = 1 \quad \text{——— (2)}$$

⇒ For the Point (1,1) with label ⊖ :

$$\Rightarrow y_3(w_1 \cdot x_{13} + w_2 \cdot x_{23} + b) = 1$$
$$\Rightarrow -1(w_1(1) + w_2(1) + b) = 1$$
$$\Rightarrow w_1 + w_2 + b = -1 \quad \text{——— (3)}$$

→ Now, we have the system of three equations:

(i) $w_1 + b = 1$

(ii) $w_2 + b = 1$

(iii) $w_1 + w_2 + b = -1$

⇒ From (i) — $w_1 + b = 1 \Rightarrow w_1 = 1 - b$

⇒ From (ii) — $w_2 + b = 1 \Rightarrow w_2 = 1 - b$

⇒ Substituting these into equation (3):

$(1-b) + (1-b) + b = -1$

$2 - b = -1 \Rightarrow \boxed{b = 3}$

⇒ Hence, $w_1 = 1 - 3 = -2$

$w_2 = 1 - 3 = -2$

⇒ Thus, the weight vector corresponding to the maximum margin hyperplane is: $w = \begin{bmatrix} -2 \\ -2 \end{bmatrix}$

⇒ The bias is: $b = 3$.

Now check the support vectors other than the point B, C and D.

⇒ For the sample point A (0,0) with label ⊕ :—

$y_i (w^T x_i + b) = (+1) \cdot (w_1 \cdot 0 + w_2 \cdot 0 + b)$

$= +1 \cdot (3) = 3 \neq 1$

→ Hence, it is not a support vector.

⇒ For the sample point E (2,2) with label ⊖ :—

$y_i (w^T x_i + b) = (-1) \cdot (w_1 \cdot 2 + w_2 \cdot 2 + b)$

$= (-1) \cdot (-2 \cdot 2 + (-2) \cdot 2 + 3)$

$= (-1) \cdot (-4 - 4 + 3)$

$= (-1) \cdot (-5) = 5 \neq 1$

→ Hence, it is not a support vector.

⇒ For the sample point F (2,0) with label ⊖ :—

$y_i (w^T x_i + b) = (-1) \cdot [w_1 \cdot 2 + w_2 \cdot 0 + b]$

$= (-1) \cdot [-2 \cdot 2 + 0 + 3]$

$= (-1) \cdot (-1) = 1$

→ Hence, it is a support vector.

∴ The support vectors are:

⇒ B → (1,0) with label ⊕

⇒ C → (0,1) " " ⊕

⇒ D → (1,1) with label ⊖

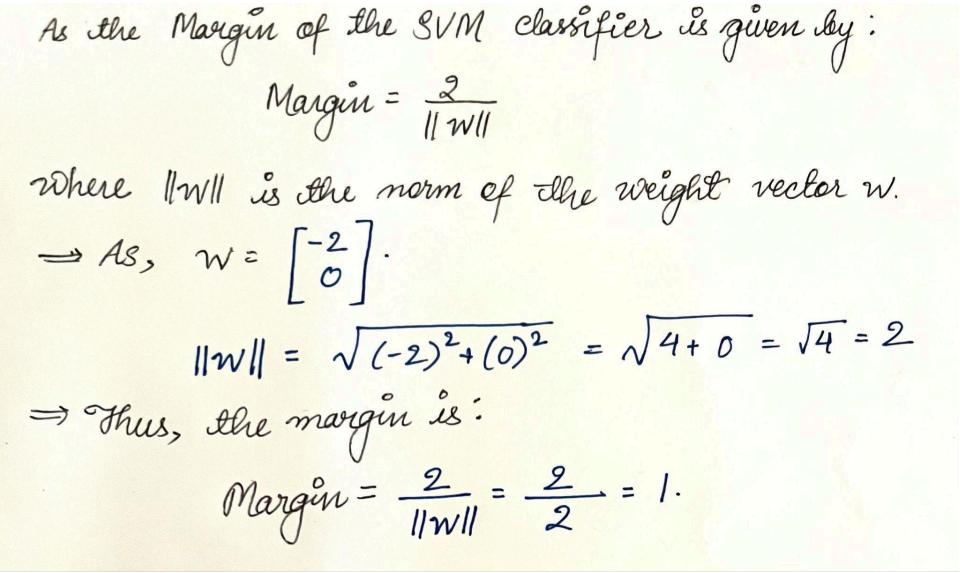
⇒ F → (2,0) with label ⊖

(c) Consider the dataset with features x1 and x2 and 2 classes y=−1 and y = +1.
Training Dataset:

Sample No.	x_1	x_2	y
1	1	2	+1
2	2	3	+1
3	3	3	-1
4	4	1	-1

The SVM formulates a decision boundary of format: w.x + b = 0 Given the values: w1 = -2, w2 = 0, b = 5. Solve the following parts:

(a) Calculate the margin of the classifier.
Solution:



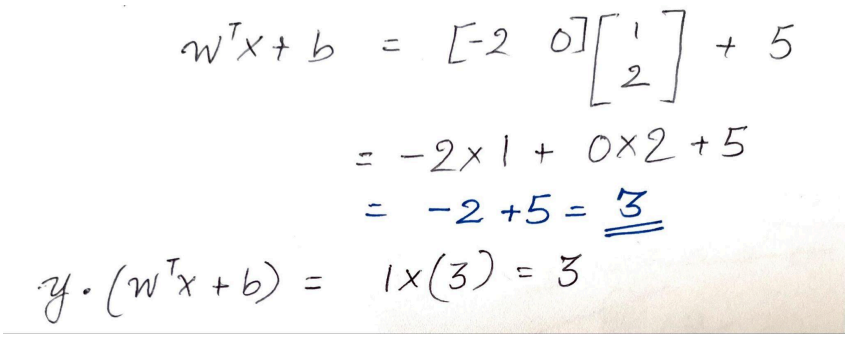
(b) Identify the support vectors. (given samples 1 to 4).
Solution: Support vectors are the points that lie on the margin boundary, satisfying the equation:

$$y(w \cdot x + b) = 1 \qquad \ldots(i)$$

If the samples in the dataset satisfy condition (i), they are considered

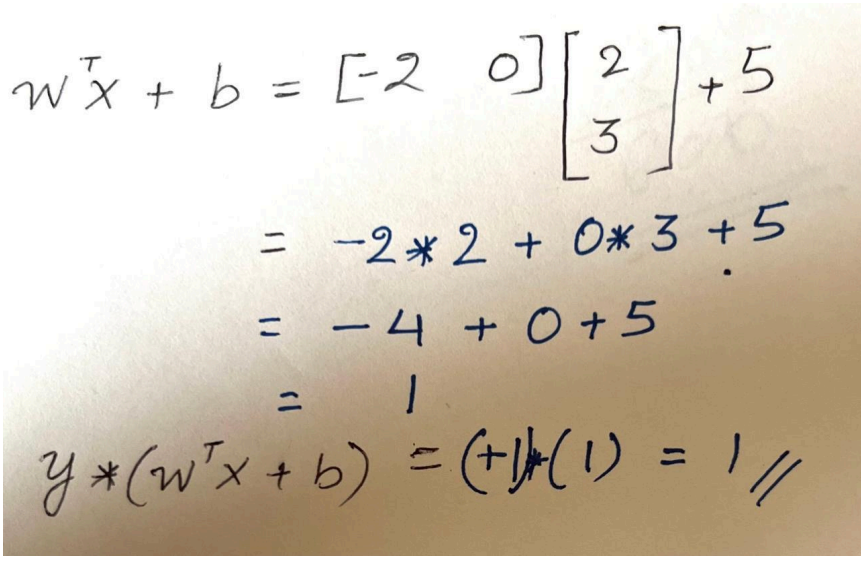
Support Vectors.

For Sample Point 1:
$x_1 = 1, x_2 = 2, y = +1$



Because, $y(w \cdot x + b) = 3$. This sample point is **not** a support vector.

For Sample Point 2:
$x_1 = 2, x_2 = 3, y = +1$



Because, $y(w \cdot x + b) = 1$. Hence, this sample point is **a** support vector.

For Sample Point 3:
$x_1 = 3, x_2 = 3, y = -1$

$$w^T x + b = [-2 \quad 0]\begin{bmatrix}3\\3\end{bmatrix} + 5$$
$$= -2*3 + 0*3 + 5$$
$$= -6 + 0 + 5$$
$$= -1$$
$$\Rightarrow y*(w^T x + b) = -1 \times (-1) = 1$$

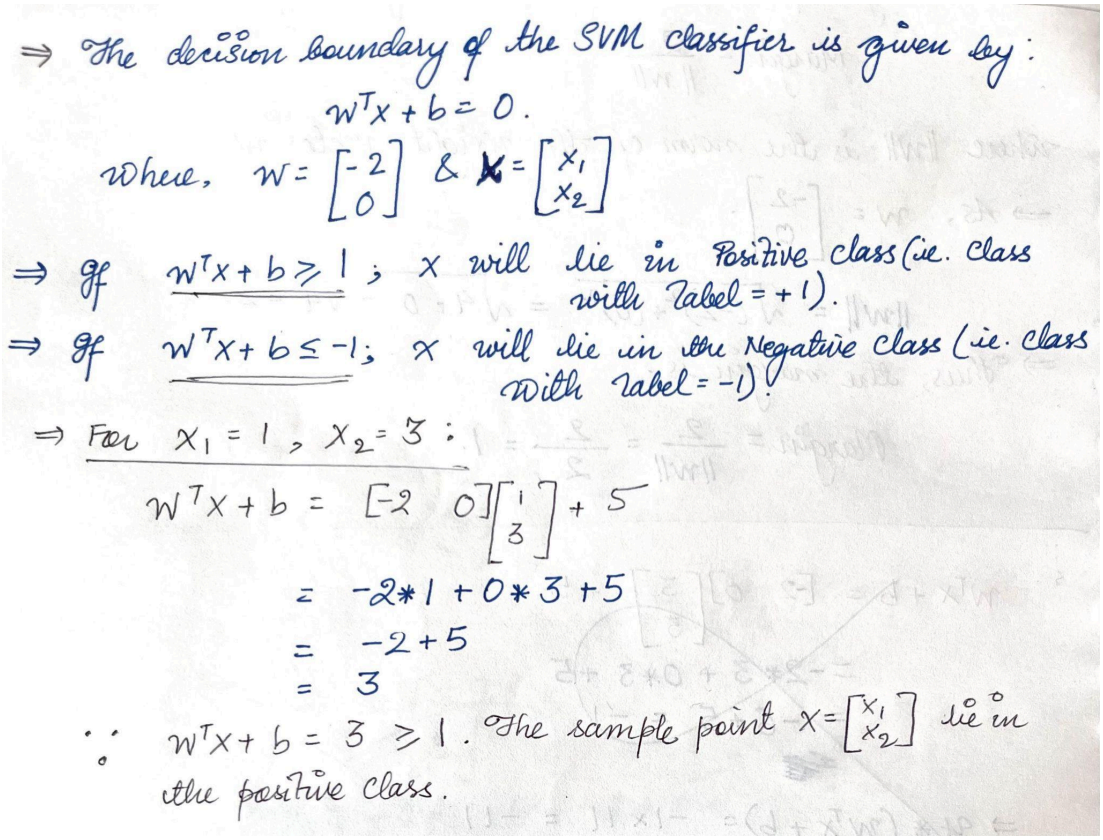
Because, $y(w \cdot x + b) = 1$. Hence, this sample point is **a** support vector.

For Sample Point 4:
$x_1 = 4$, $x_2 = 1$, $y = -1$

$$\Rightarrow w^T x + b = [-2 \quad 0]\begin{bmatrix}4\\1\end{bmatrix} + 5$$
$$= -2*4 + 0*1 + 5$$
$$= -8 + 5$$
$$= -3$$
$$\Rightarrow y*(w^T x + b) = -1*(-3) = 3$$

Because, $y(w \cdot x + b) = 3$. This sample point is **not** a support vector.

(c) Predict the class of a new point : x1=1, x2=3.
Solution:



Section B - Scratch Implementation

Solution 1:

Implemented a class named **NeuralNetwork** in a modular fashion from scratch with the following parameters during initialization:

- **N**: Number of layers in the network.
- **layer_sizes**: A list of size N specifying the number of neurons in each layer.
- **lr**: Learning rate.
- **activation**: Activation function.
- **weight_init**: Weight initialization function.
- **epochs**: Number of epochs.
- **batch_size**: Batch size.

The **NeuralNetwork** class also implemented the following functions:

- **fit(X, Y):** trains a model on input data X and labels Y.
- **predict(X):** gives the prediction for input X.
- **predict_proba(X)**: gives the class-wise probability for input X.
- **score(X, Y)**: gives the accuracy of the trained model on input X and labels Y.

- **forward(X)**: Performs Forward propagation on the input data and returns the result i.e. the output based on the specific value of activation function and weight initialization.
- **backward(X,Y)**: Performs backward propagation on the input data, based on the Mean Squared Error loss function. Parameters are also updated in this step.

Solution 2:
Implemented the following activation functions inside the class **NeuralNetwork**, along with their gradient function:

- Sigmoid
- Tanh
- Rectified Linear Unit(ReLU).
- Leaky ReLU.
- Softmax(used only in the last/output layer)

Solution 3:
Implemented the following initialization methods in the class **NeuralNetwork**:

- **Zero Initialization**: Initializing the weights and biases with 0.
- **Random Initialization**: Initializing the weights and biases with random values.
- **Normal(0,1) Initialization**: Initializing weights with values drawn from a normal (Gaussian) distribution with a mean of 0 and standard deviation of 1.

Solution 4:

Loading the dataset-

- I first downloaded the MNIST dataset from Kaggle.
- Extracted both the training and testing images and labels from the provided IDX-format files.
- The following are the dimensions of training and testing dataset:

```
X_train shape before reshaping: (60000, 28, 28)
X_test shape before reshaping: (10000, 28, 28)
y_train shape: (60000,)
y_test shape: (10000,)
```

Data Preprocessing:

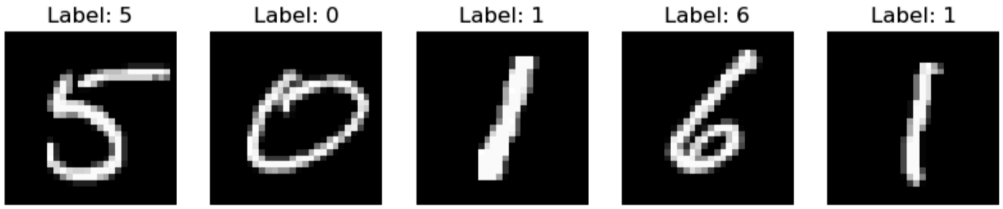
1. **Flattening Images**:
   Reshaped each image from its original **28x28** matrix into a **1D** array of **784 pixels** to simplify the input format for the neural network.
2. **Normalizing Images**:

Normalized pixel values to the range **[0, 1]** by dividing each pixel value by **255**, converting all pixel intensities to float values for consistent scaling.

3. **One Hot Encoding:**
   Applied one-hot encoding to the labels present in training and testing dataset, so that we can use it for the multiclass classification in the neural network.

Data Splitting:
According to the question, the training dataset is split into the ratio of **80:10:10** to form new training, validation and testing dataset.
Accordingly, labels are also split in the ratio of **80:10:10**.

Data Visualization:
As a sanity check, we visualize 5 random samples from the training dataset along with their labels.



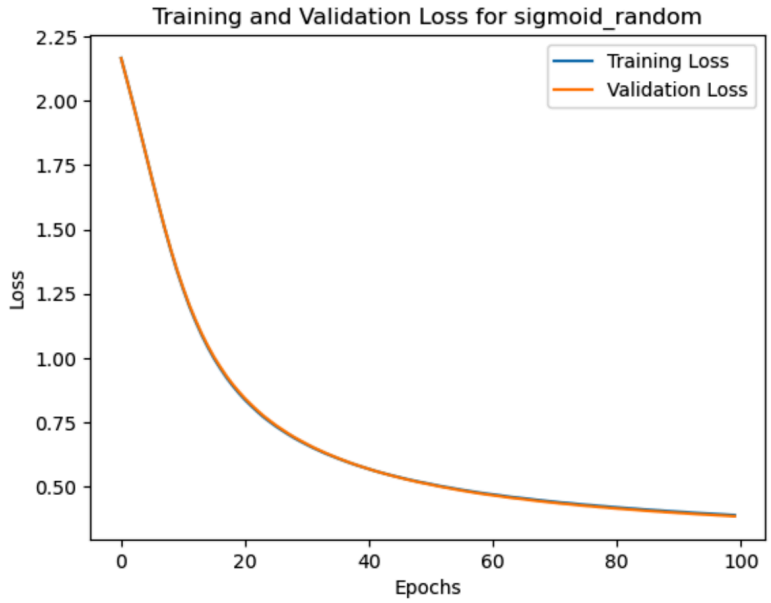
Model Training:
I trained the model using each activation function and weight initialization method across the following configurations:

- Number of hidden layers = 4
- Layer sizes = [256,128,64,32]
- Number of epochs = 100
- Batch size = 64
- Learning Rate = 2e-3
- Input Neurons = 784
- Output Neurons = 10

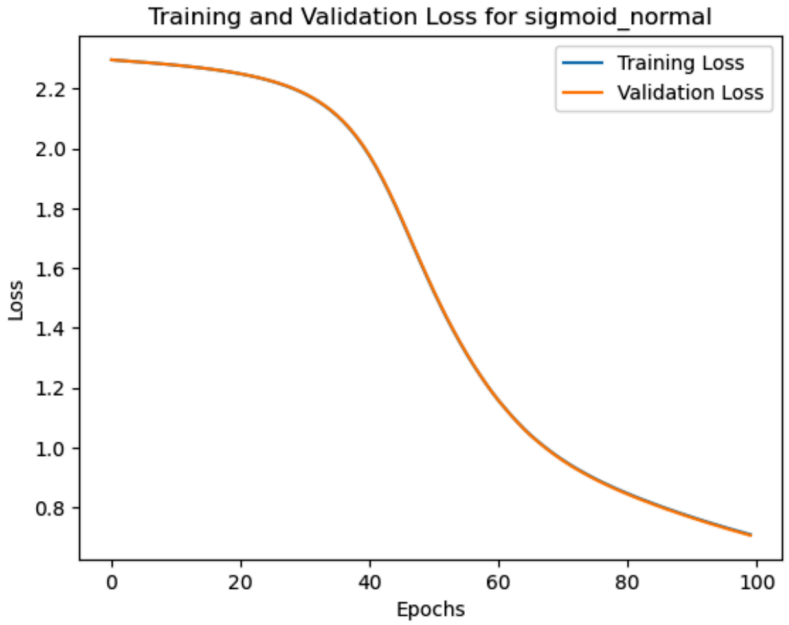
Training Loss and Validation Loss vs Epochs Curves for each model:

1. NeuralNetwork Model with Activation Function = '**sigmoid**' and Weight Initialization = '**zero**':

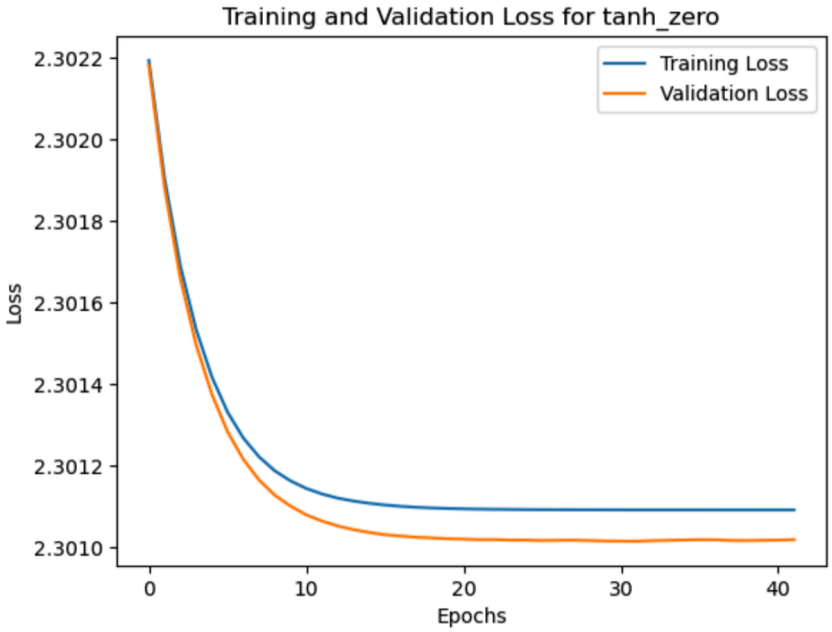
2. NeuralNetwork Model with Activation Function = '**sigmoid**' and Weight Initialization = '**random**':



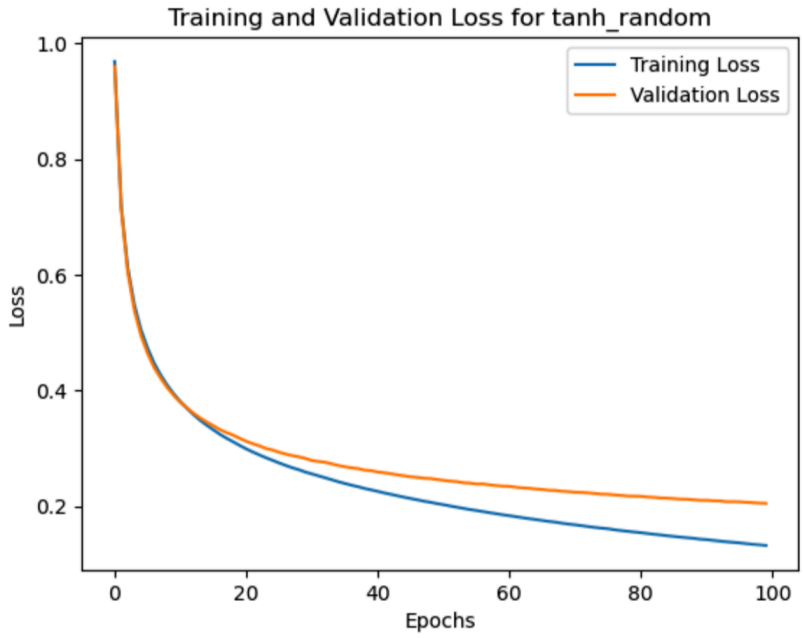
3. NeuralNetwork Model with Activation Function = '**sigmoid**' and Weight Initialization = '**normal**':



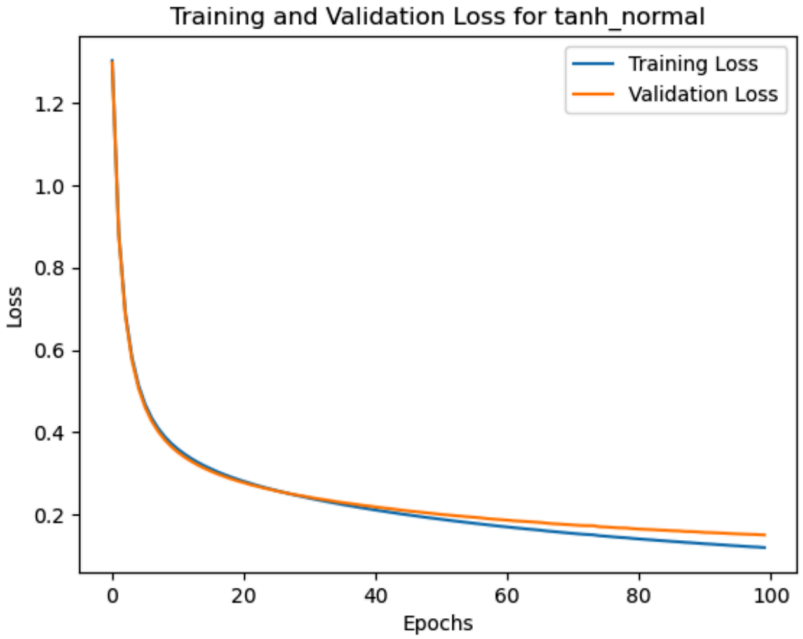
4. NeuralNetwork Model with Activation Function = '**tanh**' and Weight Initialization = '**zero**':



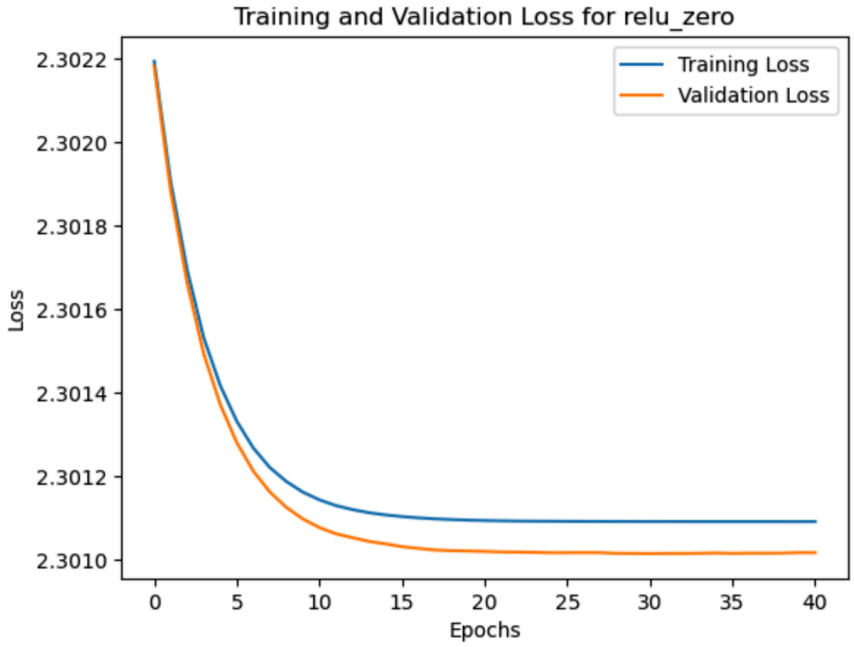
5. NeuralNetwork Model with Activation Function = '**tanh**' and Weight Initialization = '**random**':



6. NeuralNetwork Model with Activation Function = '**tanh**' and Weight Initialization = '**normal**':

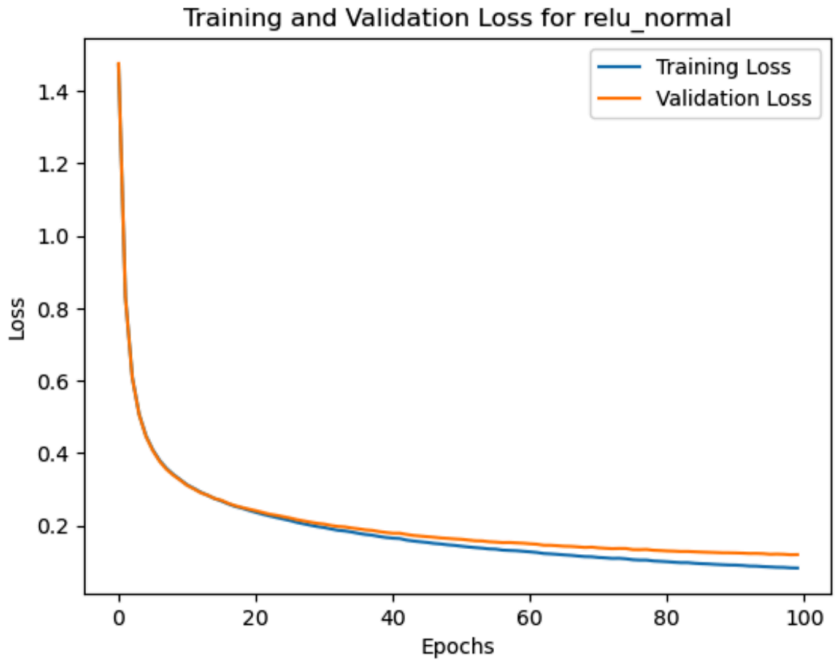


7. NeuralNetwork Model with Activation Function = '**relu**' and Weight Initialization = '**zero**':

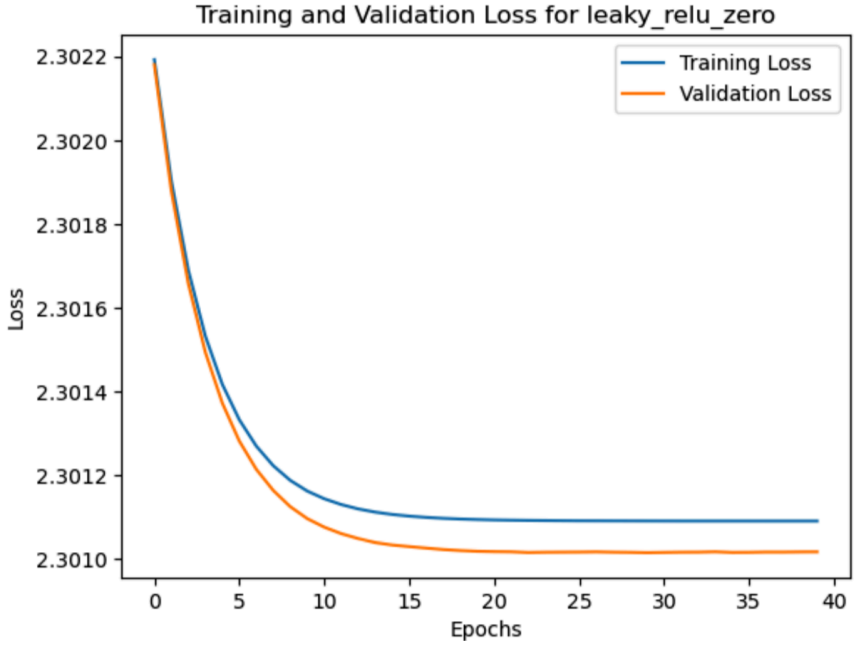


8. NeuralNetwork Model with Activation Function = '**relu**' and Weight Initialization = '**random**':

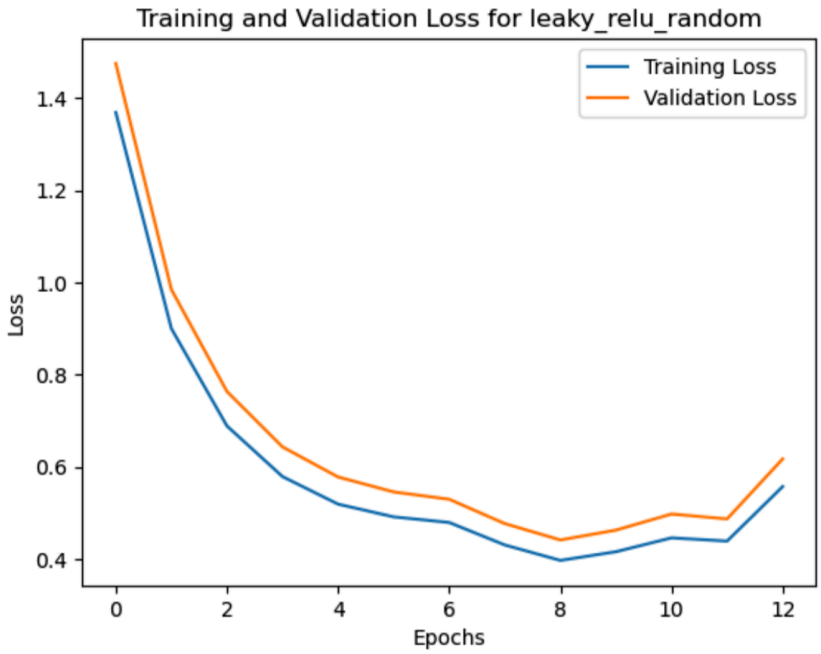
9. NeuralNetwork Model with Activation Function = '**relu**' and Weight Initialization = '**normal**':



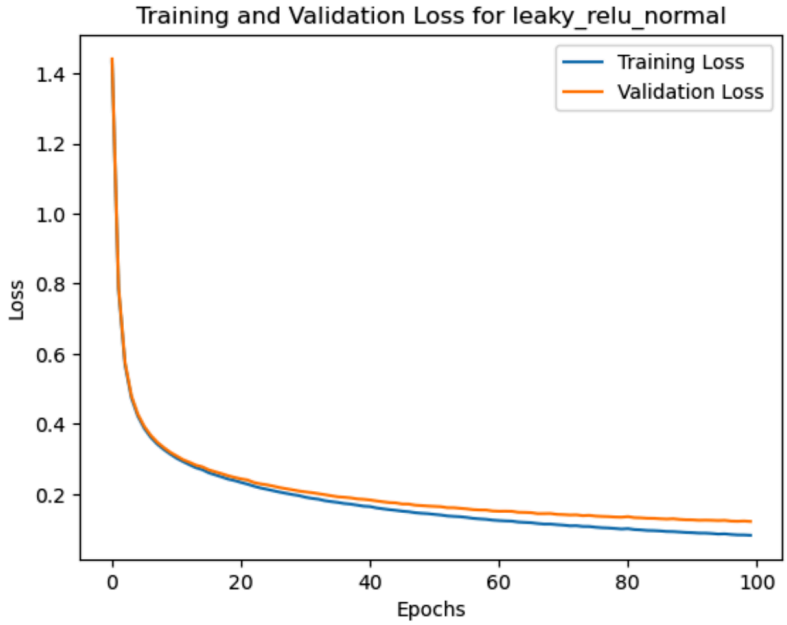
10. NeuralNetwork Model with Activation Function = '**leaky_relu**' and Weight Initialization = '**zero**':



11. NeuralNetwork Model with Activation Function = '**leaky_relu**' and Weight Initialization = '**random**':



12. NeuralNetwork Model with Activation Function = '**leaky_relu**' and Weight Initialization = '**normal**':



After testing each model on the test dataset, we get the following accuracies:

```
Test Accuracies Summary:
sigmoid_zero: 0.1067
sigmoid_random: 0.8880
sigmoid_normal: 0.7862
tanh_zero: 0.1067
tanh_random: 0.9327
tanh_normal: 0.9540
relu_zero: 0.1067
relu_random: 0.1007
relu_normal: 0.9615
leaky_relu_zero: 0.1067
leaky_relu_random: 0.1007
leaky_relu_normal: 0.9628
```

Analysis:
From the results, we can see that:

- Best-Performing Combination:
    - Model with activation function as **ReLU** and weight initialization as **normal** and model with activation function as **Leaky ReLU** and weight initialization as **normal** performs best, achieving **accuracy** of **96.15 %** and **96.28 %** on the test set, respectively.
    - Both models showed steady and consistent learning without **NaN** or early stopping issues. They successfully minimized training and validation loss over **100** epochs, indicating robust convergence and model stability.
- Suboptimal Combinations:
    - All the models (Sigmoid, Tanh, ReLU, Leaky ReLU) initialized with zero performed poorly, with each configuration having a **test accuracy of approximately 10.67%**.
    - Zero initialization often leads to symmetry problems, especially with activation functions like ReLU and Leaky ReLU, as all neurons learn the same features. This caused models to stagnate around the same loss and accuracy values, never improving across epochs.
- Moderate Combinations:
    - **Tanh with Normal Initialization (95.40% accuracy) and Tanh with Random Initialization (93.27% accuracy):** The tanh activated functions also performed and converged well, with its initialization with Normal and Random weights and biases.

- **Sigmoid with Normal Initialization achieved 78.62% test accuracy:** Sigmoid activation's limitations with deeper architectures or larger datasets likely caused it to underperform compared to ReLU and Leaky ReLU activations.
- **Sigmoid with Random Initialization achieved 88.80% test accuracy:** Although, this combination shows well convergence, it is likely to suffer from the potential vanishing gradient from Sigmoidal activation function.

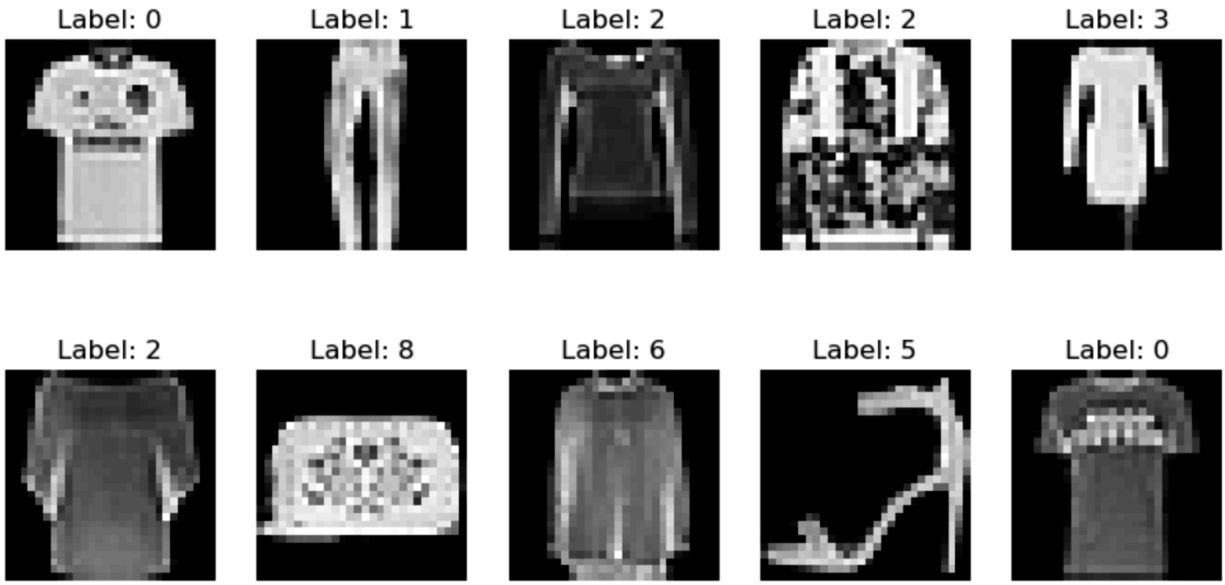
Section C - Algorithm implementation using packages

For this question, you would need to download the Fashion-MNISTdataset. It contains a train.csv and a test.csv. You need to take the first 8000 images from the train data and the first 2000 from the test data. These will be your training and testing splits.

Q1. Perform appropriate preprocessing on the data (for eg: normalization) and visualize any 10 samples from the test dataset.
Solution:
For the process of data preprocessing, I proceed in two steps:

1. The first step of data preprocessing includes **handling of any type of missing values**. This is an important step with regards to the dataset because missing values can often lead to inaccurate or misleading results. This is because models trained on incomplete data might learn patterns that do not generalize well to the unseen data. After observing the dataset, there are no missing values or data present in the dataset. Hence, we proceed to the second step.
2. The next step of data processing includes **normalizing or scaling** the input feature values, which originally ranged from **1 to 256**. This normalization ensures that all features are scaled consistently, improving the performance and stability of the machine learning model.

Data Visualization on the **10 samples** from the test dataset are given below:



Q2. Train a MLP Classifier from sklearn's neural network module on the training dataset. The network should have 3 layers of size [128, 64, 32], should be trained for 100 iterations using an 'adam' solver with a batch size of 128 and learning rate of 2e-5. Train it using all the 4 activation functions i.e. 'logistic', 'tanh', 'relu' and 'identity'. For each activation function, plot the training loss vs epochs and validation loss vs epochs curves and comment on which activation function gave the best performance on the test set in the report.
Solution:

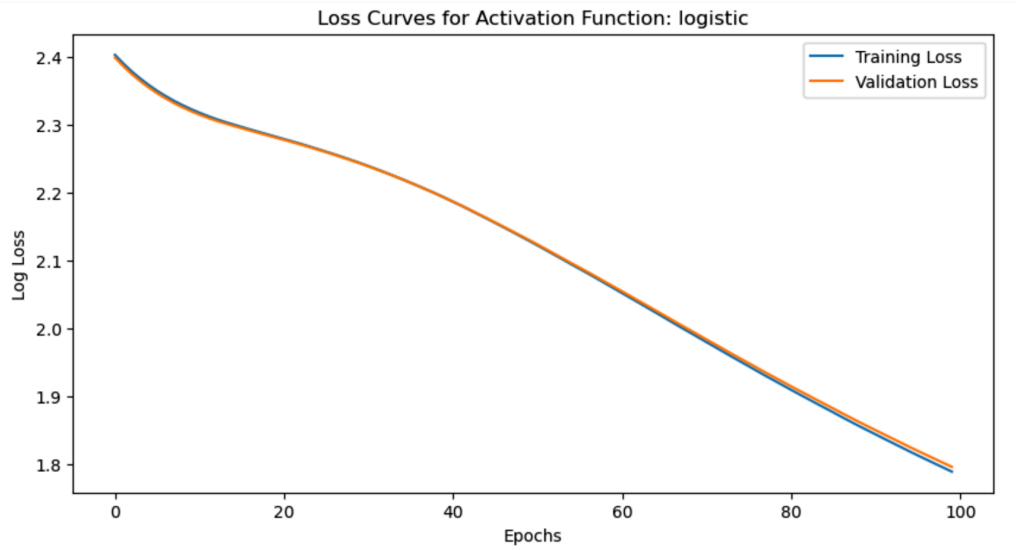
The Original Training Set is divided in a ratio of **80:20** to form a new Training Set and Validation Set.

There are four activation function that we have to use to train the model:

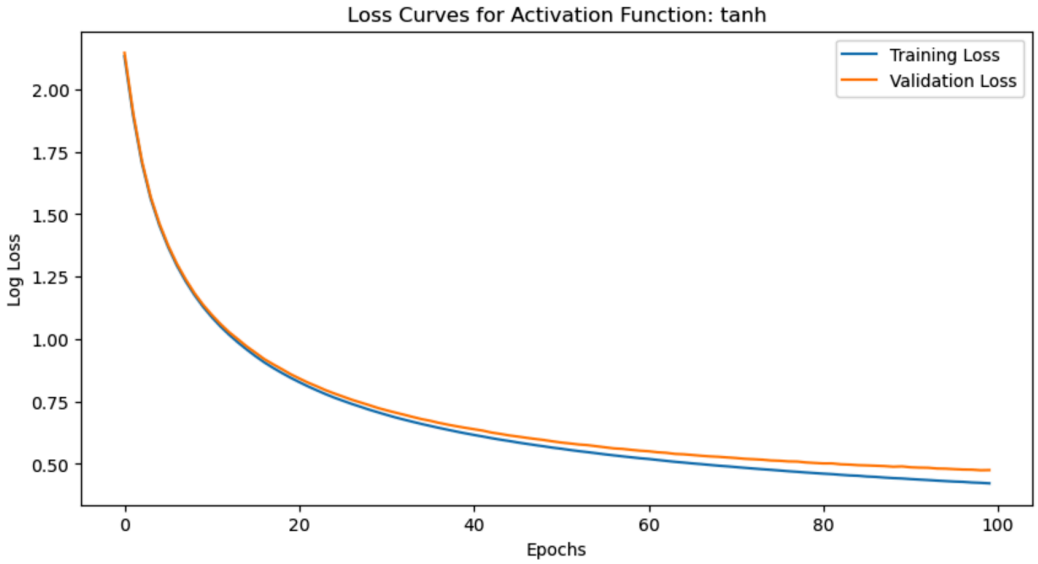
1. Logistic
2. tanh
3. ReLU(Rectified Linear Unit)
4. Identity

Training Loss and Validation Loss curves for each type of activation functions are provided below:

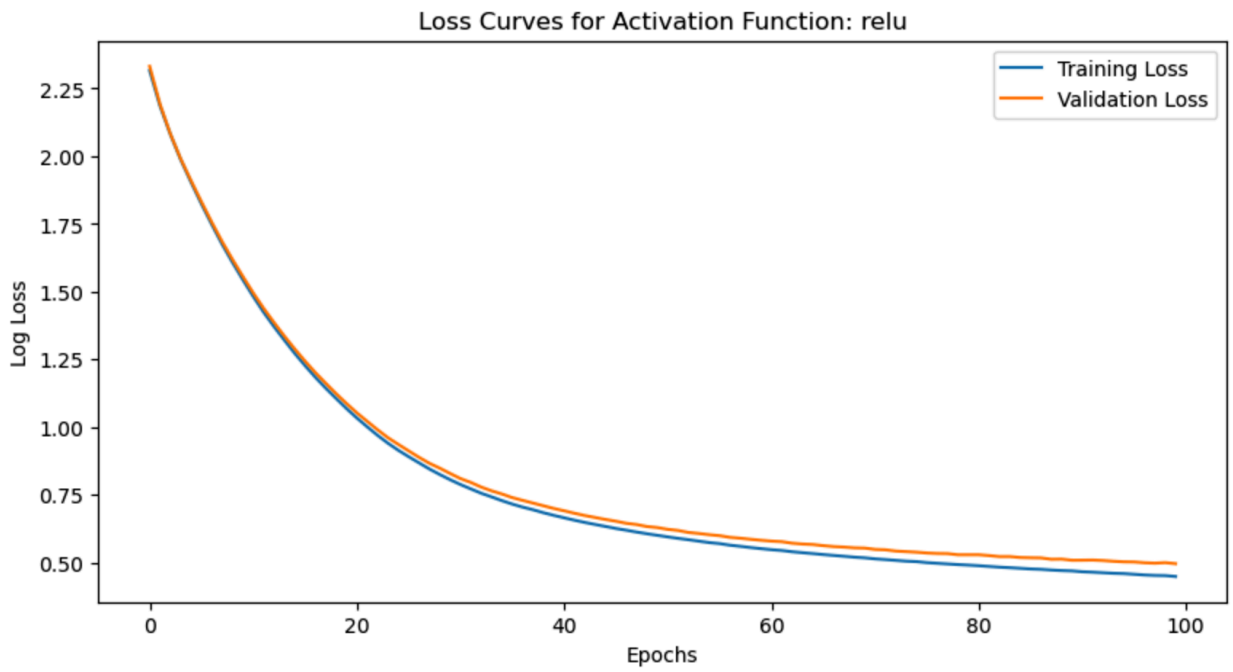
1. **For 'logistic' activation function:**



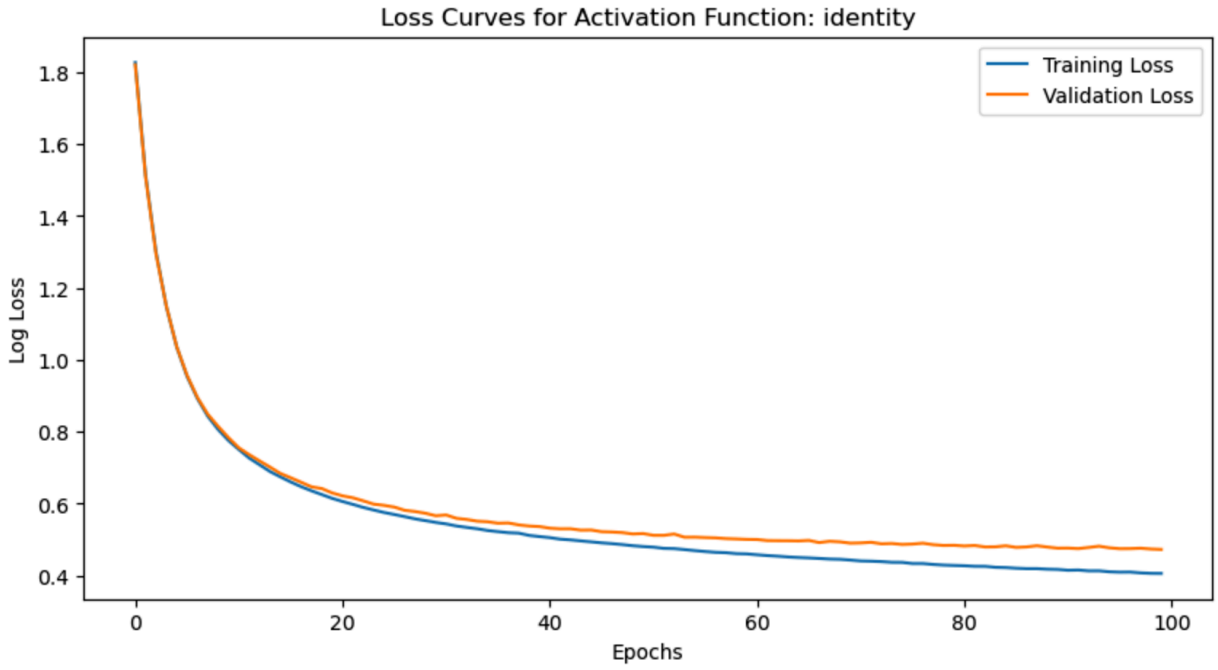
2. **For ‘tanh’ activation function:**



3. **For ‘relu’ activation function:**



4. **For 'identity' activation function:**



After evaluating each model on the test set, we get the following results:

```
Test Set Accuracies for Each Activation Function:
logistic: 0.2910
tanh: 0.8365
relu: 0.8260
identity: 0.8275

The best activation function based on test set performance is: tanh
```

Out of all the activation functions, tanh function performs best as it has the highest accuracy score on the test set.

Q3. Perform grid search using the best activation function from part 2 to find the best hyperparameters (eg: solver, learning rate, batch size) for the MLP classifier and report them in the report.
Solution:

In this part, I performed the grid search on the best activation function from part 2 i.e. tanh activation function.
The following are the choices of parameters out of which, we need to select the best hyperparameters for the MLP classifier:

- **Solver**: adam, sgd, lbfgs.
- **Learning Rate**: 0.0001, 0.0005, 0.001
- **Batch Size**: 64, 128, 256
- **Hidden Layer Size**: (128, 64, 32)
- **Maximum Iterations**: 100

Out of all the choices for each parameter, performing Grid Search CV yields the following best hyperparameters for the MLP classifier:

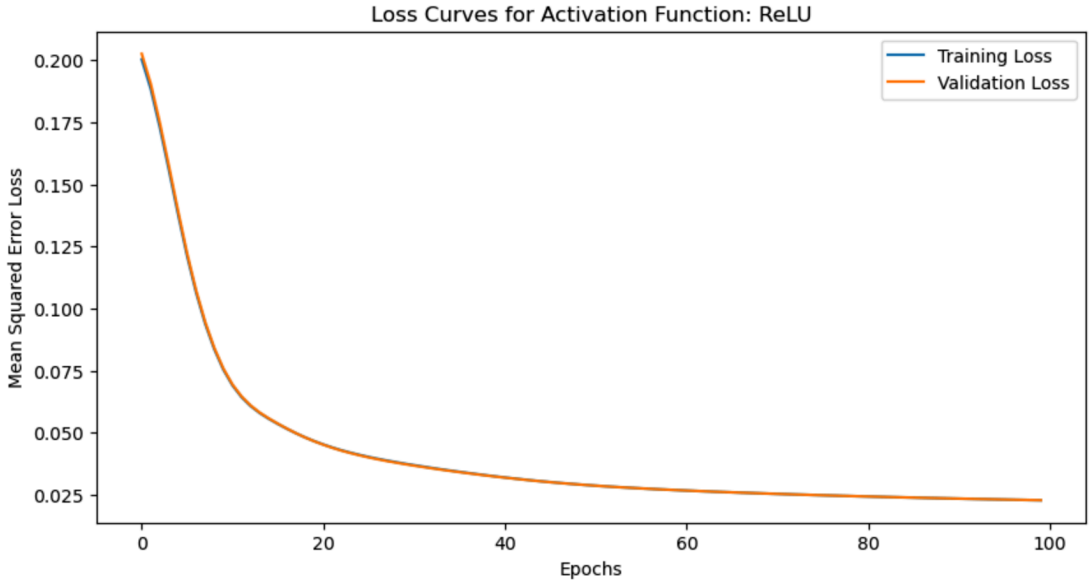
- **Solver**: adam
- **Learning Rate**: 0.001
- **Batch Size**: 64
- **Hidden Layer Size**: (128, 64, 32)
- **Maximum Iterations**: 100

Best Cross Validation Score, after performing the Grid Search CV came out to be:
0.86537551148192

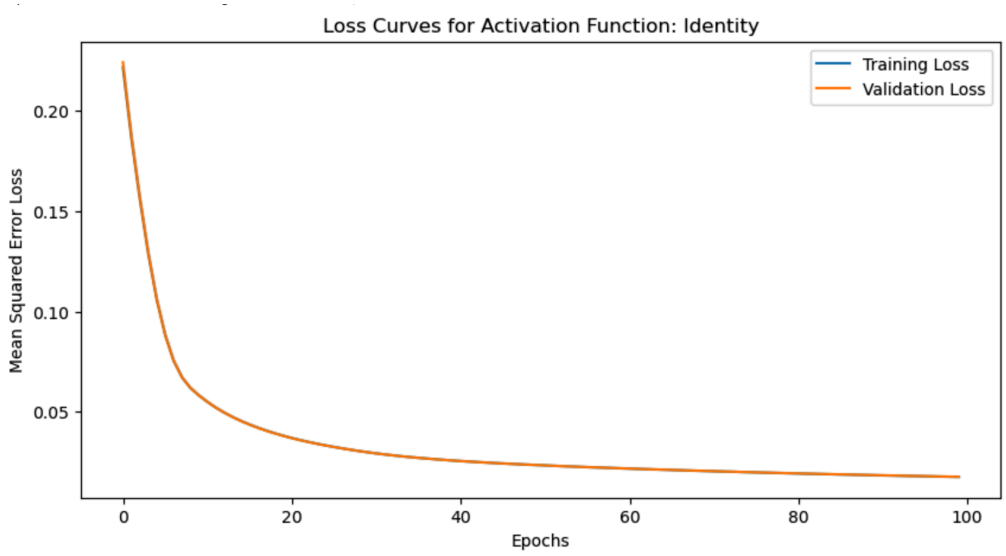
```
Best Hyperparameters: {'batch_size': 64, 'hidden_layer_sizes': (128, 64, 32), 'learning_rate_init': 0.001,
'max_iter': 100, 'solver': 'adam'}
Best Cross-Validation Score: 0.86537551148192
```

Q4. For this part, you need to train a MLPRegressor from sklearn neural network module on a regeneration task:
(a) This means you will need to design a 5 layer neural network with layer sizes following the format: [c, b, a, b, c] where c > b > a.
(b)By regeneration task, it means that you will try to regenerate the input image using your designed neural network and plot the training and validation losses per epoch to see if your model is training correctly.
(c) Train 2 neural networks on the task above. One using a 'relu' activation and the other using the 'identity' activation function. Set the solver as adam and use a constant learning rate of 2e-5.
(d) Post training both the networks, visualize the generations for the 10 test samples you visualized in part 1 and describe your observations in the report.
Solution:

Training and Validation Losses Per Epoch for ReLU Model:

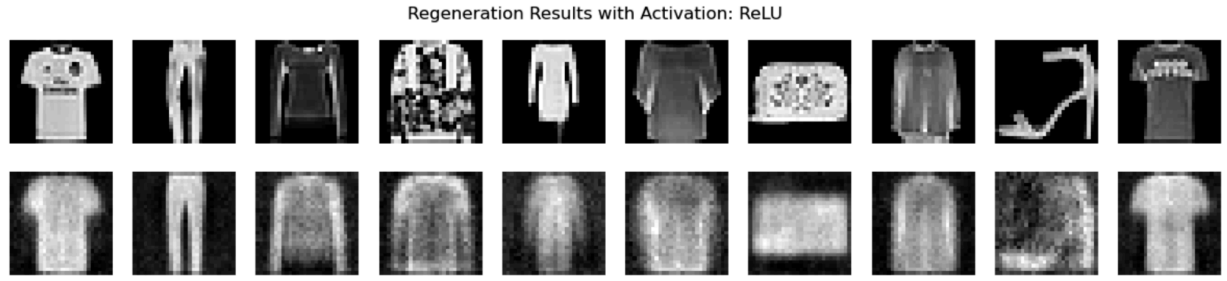


Training and Validation Losses Per Epoch for Identity Model:

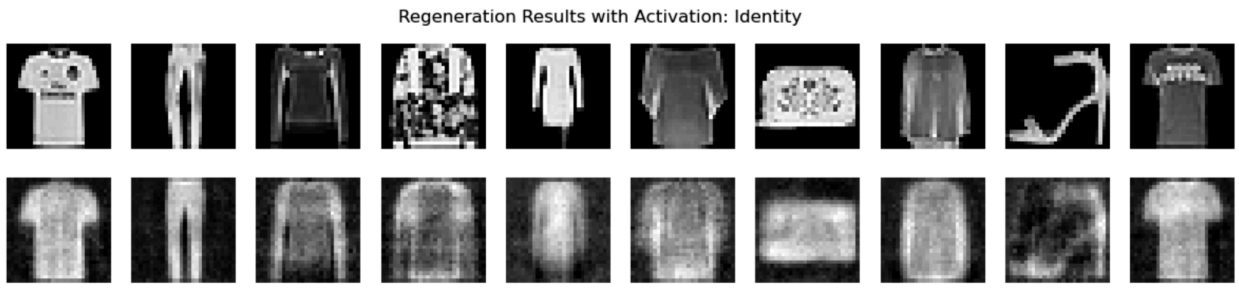


Visualizations of Test Samples:

Regenerated results for Model with Activation: ReLU



Regenerated results for Model with Activation: Identity



Observations:

- **Regeneration with ReLU Activation:**
    - The regenerated images, although less sharp, maintain clear structural outlines of the original samples.

    - There is a noticeable amount of detail retained, showing that the ReLU activation model is capable of approximating the original images closely.
    - However, certain regenerated items, such as shirts and dresses, may appear blurred or lacking finer details.

- **Regeneration with Identity Activation:**
    - The regenerated images using the Identity activation also capture the general structure but are somewhat fuzzier than those generated by the ReLU-based model.
    - Despite some blurriness, the Identity activation's output suggests better handling of continuous, gradient-based adjustments in pixel intensity, providing smoother transitions.
    - The results may exhibit more even gray tones, which could be beneficial for maintaining soft details but less effective for clear, high-contrast edges.
- **New Models Trained:**
    - The model with ReLU Activation Function shows constant reduction in the loss for both the training and validation set. The loss curve starts higher but reduces steadily as the epoch progresses.
    - The model with Identity Activation Function also shows similar behavior with initially starting at a higher point and then decreasing steadily.

Q5. Lastly, from the two neural networks trained above extract the feature vector of size 'a' for the train and test data samples. Using this vector as your new set of image features, train two new smaller MLP Classifiers with 2 layers, each of size 'a' on the training dataset and report accuracy metrics for both these classifiers. Train it for 200 iterations with the same solver and learning rate as part 2. Contrast this with the MLP Classifier you trained in part 2 and report possible reasons why this method still gives you a decent classifier?
Solution:

The extracted features from the earlier trained neural network yield:

```
Train features shape (ReLU): (8000, 64)
Test features shape (ReLU): (2000, 64)
Train features shape (Identity): (8000, 64)
Test features shape (Identity): (2000, 64)
```

Configurations for training new two smaller MLP Classifier models:

- Number of layers: 2
- Size of each layer: 2
- Number of iterations: 200
- Solver: adam
- Learning rate: 2e-5

The accuracy of the two new smaller MLP Classifiers with 2 layers, each of the size ‘a’ on the training dataset yields the accuracy as:

```
Test accuracy with ReLU features: 0.7480
Test accuracy with Identity features: 0.8035
```

Observations:

- In Part 2, we trained different activation functions directly on the image data. The performance varied significantly depending on the chosen activation function, with **tanh** yielding the best test accuracy (83.65%), followed by **identity** (82.75%), **ReLU** (82.60%), and **logistic** (29.10%).
- In contrast, the current method involves extracting feature vectors from the already trained neural networks (with ReLU and Identity activations) and using these features as inputs to train new, smaller MLP classifiers.
- This approach reuses learned representations, potentially making it easier for the new classifiers to converge and perform well.

Possible reasons for the decent performance of the classifier performance can be:

- The new classifiers use pre-learned features(As these are the features that are extracted by the already trained ReLU and Identity activated models). By starting with the inputs that are highly informative, it might have led to better performance compared to training from scratch.
- Training a classifier on extracted feature vectors simplifies the problem by reducing the complexity of learning from raw pixel data. The MLPs can focus on finding optimal decision boundaries within this feature space instead of having to learn the initial feature extraction themselves.
- The extracted features have a lower dimensionality as compared to the original input space with only 64 extracted features as compared to originally present 784

columns, it simplified the learning process for the new MLP classifiers. This would have led to more efficient training and better generalization.