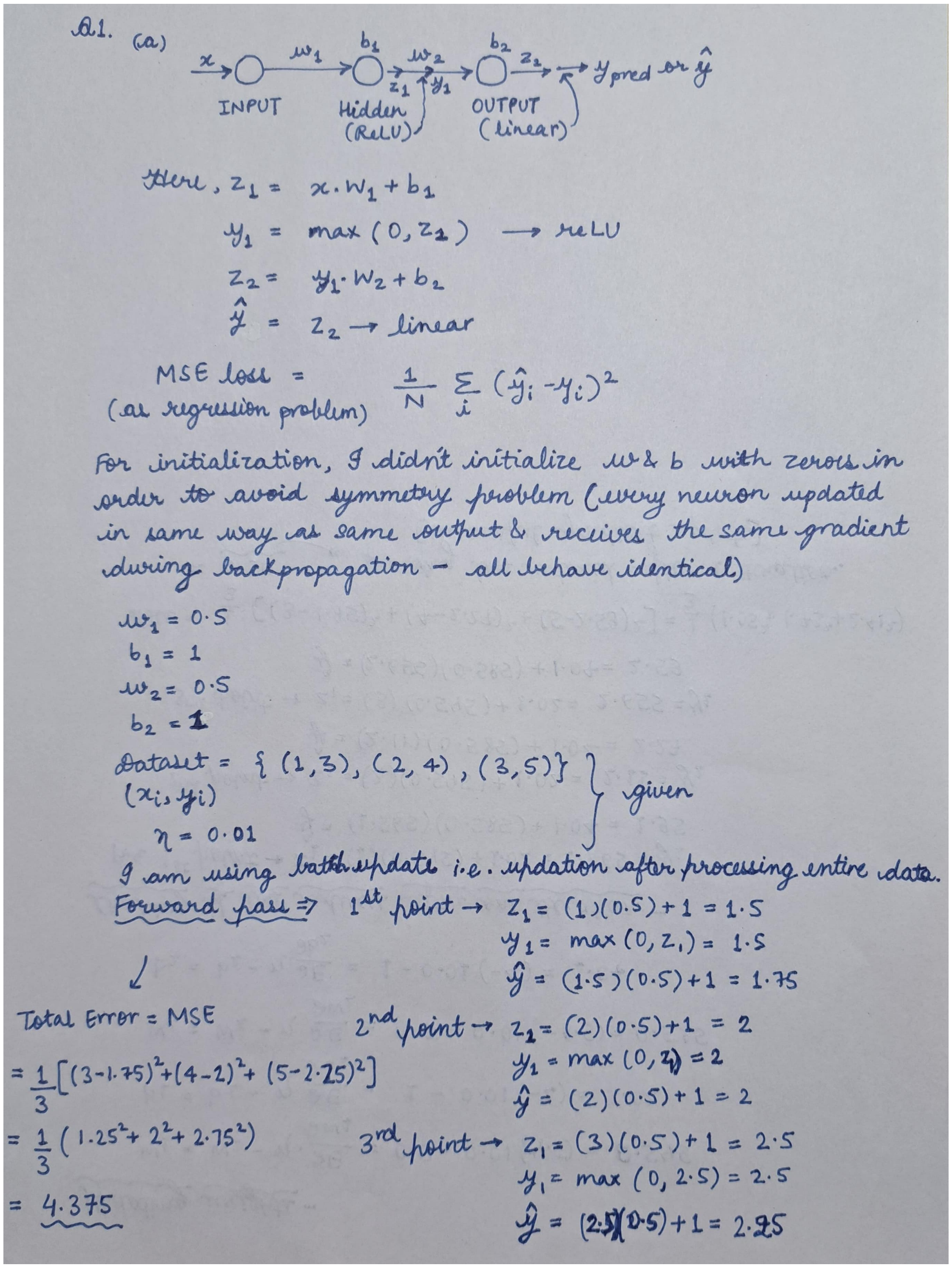
ML Assignment-3

Shobhit Raj (2022482)

1 Section A

1.1 Q1. a)



Q1. (a)

$$\begin{tikzcd} \arrow[r, "x"] & \bigcirc \arrow[r, "w_1"] & \overset{b_1}{\bigcirc} \arrow[r, "w_2"', "z_1 \uparrow y_1"] & \overset{b_2}{\bigcirc} \arrow[r, "z_2"] & y_{pred} \text{ or } \hat{y} \end{tikzcd}$$

INPUT — Hidden (ReLU) — OUTPUT (linear)

Here, $z_1 = x \cdot W_1 + b_1$

$y_1 = \max(0, z_1) \longrightarrow$ reLU

$z_2 = y_1 \cdot W_2 + b_2$

$\hat{y} = z_2 \rightarrow$ linear

MSE loss (as regression problem) $= \frac{1}{N} \sum_i (\hat{y}_i - y_i)^2$

For initialization, I didn't initialize w & b with zeros in order to avoid symmetry problem (every neuron updated in same way as same output & receives the same gradient during backpropagation – all behave identical)

$w_1 = 0.5$

$b_1 = 1$

$w_2 = 0.5$

$b_2 = 1$

Dataset $= \{(1,3), (2,4), (3,5)\}$
(x_i, y_i)
$\eta = 0.01$
} given

I am using batch update i.e. updation after processing entire data.

Forward pass $\Rightarrow$ 1st point $\rightarrow$ $z_1 = (1)(0.5) + 1 = 1.5$
$y_1 = \max(0, z_1) = 1.5$
$\hat{y} = (1.5)(0.5) + 1 = 1.75$

2nd point $\rightarrow$ $z_1 = (2)(0.5) + 1 = 2$
$y_1 = \max(0, 2) = 2$
$\hat{y} = (2)(0.5) + 1 = 2$

3rd point $\rightarrow$ $z_1 = (3)(0.5) + 1 = 2.5$
$y_1 = \max(0, 2.5) = 2.5$
$\hat{y} = (2.5)(0.5) + 1 = 2.25$

$\downarrow$

Total Error = MSE

$= \frac{1}{3}[(3-1.75)^2 + (4-2)^2 + (5-2.25)^2]$

$= \frac{1}{3}(1.25^2 + 2^2 + 2.75^2)$

$= 4.375$

Backpropagation -

$$E = \frac{1}{N} \sum_i (\hat{y}_i - y)^2$$

$$\frac{\partial E}{\partial \hat{y}} = \frac{2}{N}\sum_i (\hat{y}_i - y) = \frac{2}{3}(6) = -4$$

$$\frac{\partial E}{\partial w_2} = \frac{\partial E}{\partial \hat{y}} \times \frac{\partial \hat{y}}{\partial w_2} = \frac{2}{N}\sum_i (\hat{y}_i - y) \times y_1 = -8.5$$

$$\frac{\partial E}{\partial b_2} = \frac{\partial E}{\partial \hat{y}} \times \frac{\partial \hat{y}}{\partial b_2} = \frac{2}{N}\sum_i (\hat{y}_i - y) \times 1 = -4$$

$$\text{ReLU}'(x) = \begin{cases} 1 & \text{if } x > 0 \\ 0 & \text{else} \end{cases}$$

(Here, $z_1 > 0$)

$$\frac{\partial E}{\partial w_1} = \frac{\partial E}{\partial \hat{y}} \times \frac{\partial \hat{y}}{\partial y_1} \times \frac{\partial y_1}{\partial z_1} \times \frac{\partial z_1}{\partial w_1} = \frac{2}{N}\sum_i (\hat{y}_i - y) \times W_2 \times \underset{\sim}{1} \times (X) = -4.5$$

$$\frac{\partial E}{\partial b_1} = \frac{\partial E}{\partial \hat{y}} \times \frac{\partial \hat{y}}{\partial y_1} \times \frac{\partial y_1}{\partial z_1} \times \frac{\partial z_1}{\partial b_1} = \frac{2}{N}\sum_i (\hat{y}_i - y) \times W_2 \times 1 \times 1 = -2$$

Updating weights -

$$W_1 = W_1 - \eta \cdot \frac{\partial E}{\partial W_1} = 0.5 - 0.01(-4.5) = 0.545$$

$$b_1 = b_1 - \eta \frac{\partial E}{\partial b_1} = 1 - 0.01(-2) = 1.02$$

$$W_2 = W_2 - \eta \frac{\partial E}{\partial W_2} = 0.5 - 0.01(-8.5) = 0.585$$

$$b_2 = b_2 - \eta \frac{\partial E}{\partial b_2} = 1 - 0.01(-4) = 1.04$$

Checking if loss reduces after one iteration -

For 1st point $\rightarrow$ $z_1 = (1)(0.545) + 1.02 = 1.565 = y_1$

$\hat{y} = (1.565)(0.585) + 1.04 = 1.95$

2nd point $\rightarrow$ $z_1 = (2)(0.545) + 1.02 = 2.11 = y_1$

$\hat{y} = (2.11)(0.585) + 1.04 = 2.27$

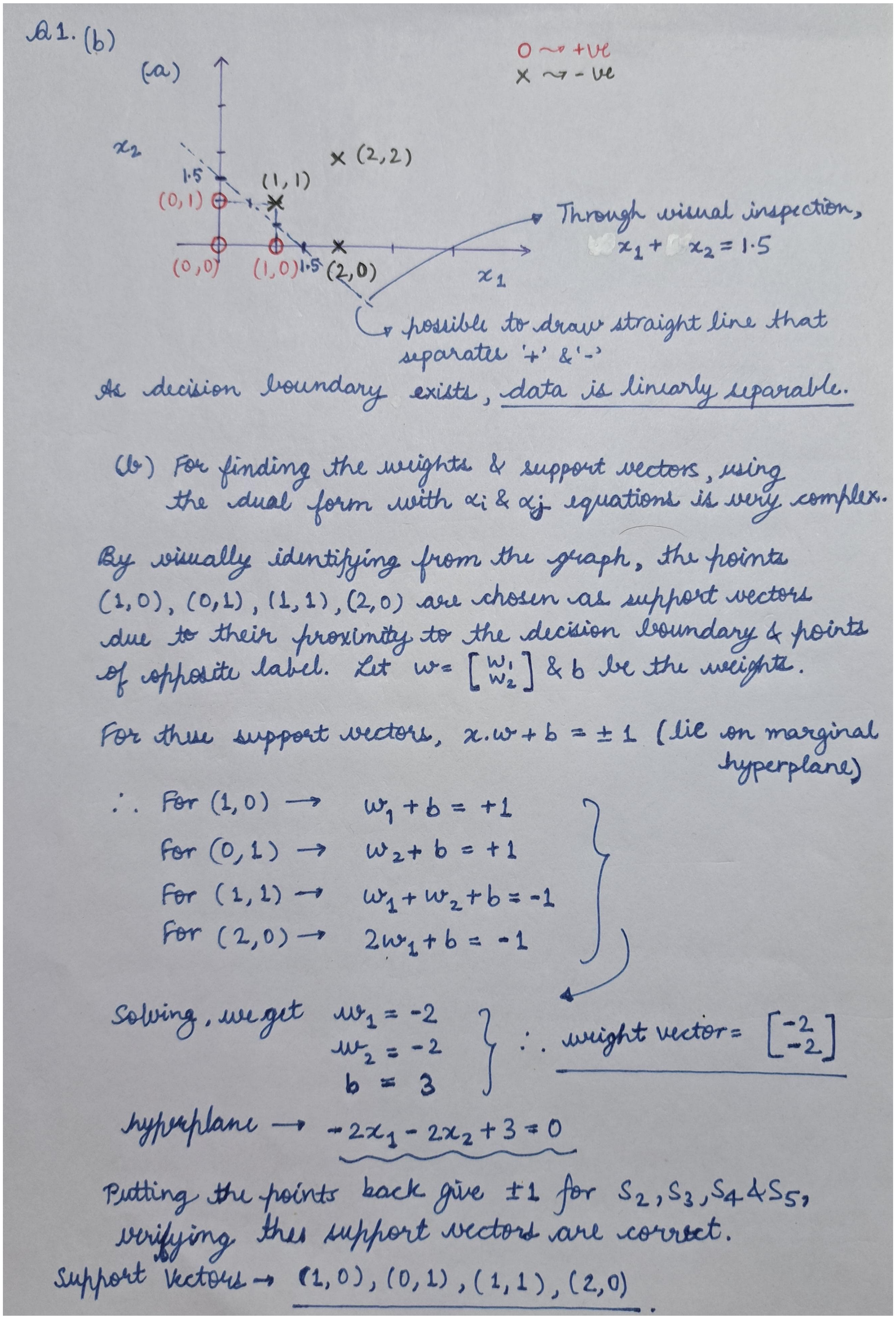
3rd point $\rightarrow$ $z_1 = (3)(0.545) + 1.02 = 2.655 = y_1$

$\hat{y} = (2.655)(0.585) + 1.04 = 2.59$

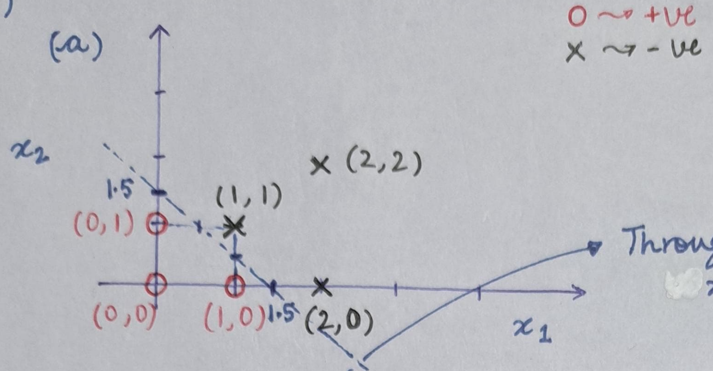
$$\text{Loss} = \frac{1}{3}\left[(3-1.95)^2 + (4-2.27)^2 + (5-2.59)^2\right] = \frac{1}{3}(1.05^2 + 1.73^2 + 2.41^2)$$

$\approx \underset{\sim}{3.30}$ $\rightarrow$ showing loss reduced after 1 iteration. [successful gradient update]

1.2 Q1. b)



Q1. (b)



Through visual inspection, $x_1 + x_2 = 1.5$

possible to draw straight line that separates '+' & '-'

As decision boundary exists, data is linearly separable.

(b) For finding the weights & support vectors, using the dual form with α_i & α_j equations is very complex.

By visually identifying from the graph, the points (1,0), (0,1), (1,1), (2,0) are chosen as support vectors due to their proximity to the decision boundary & points of opposite label. Let $w = \begin{bmatrix} w_1 \\ w_2 \end{bmatrix}$ & b be the weights.

For these support vectors, $x.w + b = \pm 1$ (lie on marginal hyperplane)

$$\left.\begin{aligned} \therefore \text{ For } (1,0) &\rightarrow w_1 + b = +1 \\ \text{For } (0,1) &\rightarrow w_2 + b = +1 \\ \text{For } (1,1) &\rightarrow w_1 + w_2 + b = -1 \\ \text{For } (2,0) &\rightarrow 2w_1 + b = -1 \end{aligned}\right\}$$

$$\text{Solving, we get } \left.\begin{aligned} w_1 &= -2 \\ w_2 &= -2 \\ b &= 3 \end{aligned}\right\} \therefore \underline{\text{weight vector} = \begin{bmatrix} -2 \\ -2 \end{bmatrix}}$$

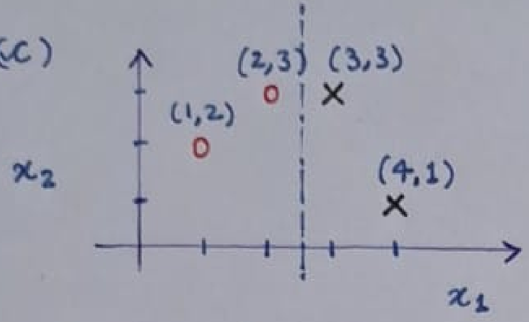
hyperplane $\rightarrow$ $-2x_1 - 2x_2 + 3 = 0$

Putting the points back give ± 1 for S_2, S_3, S_4 & S_5, verifying the support vectors are correct.

Support Vectors $\rightarrow$ (1,0), (0,1), (1,1), (2,0).

1.3 Q1. c)

Q1. (c)



$W = \begin{bmatrix} -2 \\ 0 \end{bmatrix}$, $b = 5$

$\swarrow$

$-2x_1 + 5 = 0 \Rightarrow \underline{x_1 = 2.5}$.

Decision boundary $\Rightarrow$ $w.x + b = 0$

a) Margin (separating '+' & '-') $= \dfrac{2}{\|w\|} = \dfrac{2}{\sqrt{(-2)^2+0^2}} = \dfrac{2}{2} = \underset{\sim\sim}{1 \text{ unit}}$.

[Margin of separation (distance to closest example) $\simeq 0.5$ unit]

b) Support vectors are points that lie on marginal hyperplane, where

$$wx + b = \pm 1$$

point 1 $\rightarrow$ $\begin{pmatrix} -2 \\ 0 \end{pmatrix} \cdot \begin{pmatrix} 1 \\ 2 \end{pmatrix} + 5 = 3$

point 2 $\rightarrow$ $\begin{pmatrix} -2 \\ 0 \end{pmatrix} \cdot \begin{pmatrix} 2 \\ 3 \end{pmatrix} + 5 = 1$ $\Big\}$ $\underline{\text{support vectors}}$.

point 3 $\rightarrow$ $\begin{pmatrix} -2 \\ 0 \end{pmatrix} \cdot \begin{pmatrix} 3 \\ 3 \end{pmatrix} + 5 = -1$

point 4 $\rightarrow$ $\begin{pmatrix} -2 \\ 0 \end{pmatrix} \cdot \begin{pmatrix} 4 \\ 1 \end{pmatrix} + 5 = -3$

$\therefore$ $\underline{\text{Sample 2 \& 3}}$ are support vectors.

c) For new sample, we predict sign ($X_{test} \cdot w + b$)

$X_{test} = \begin{bmatrix} 1 \\ 3 \end{bmatrix}$

$\therefore$ $\text{sign}\left(\begin{pmatrix} 1 \\ 3 \end{pmatrix} \cdot \begin{pmatrix} -2 \\ 0 \end{pmatrix} + 5\right) = \text{sign}(+3) \rightarrow \underline{\text{predict '+1' class}}$.

$\downarrow$

$\underset{\sim\sim}{\geq 0}$

2 Section B

We have been given the "MNIST" dataset, which is a collection of 70,000 grayscale images, each 28x28 pixels (784 features) representing 10 handwritten digits from 0-9. It includes 60,000 training images and 10,000 testing images, with each image labeled from 0 to 9 corresponding to different digits.

There are no null values in the dataset.

2.1 Q2. 1) Initialization, Parameters & Functions

The NeuralNetwork class is initialized with flexible parameters, including the number of layers, neurons per layer, learning rate, activation function, weight initialization method, epochs, and batch size. Core methods include fit for training, predict for label prediction, predict_proba for class probability output, and score for accuracy evaluation. These functions cover essential tasks for training and testing a neural network model. The network employs a forward pass for prediction, backpropagation to calculate and propagate gradients, and mini-batch gradient descent to update weights. For understanding the methodology, I referred to this video.

2.2 Q2. 2) Activation Functions and Gradients

The class supports multiple activation functions: sigmoid, tanh, ReLU, Leaky ReLU, and softmax. Each has its own gradient function, which facilitates effective backpropagation and enables non-linear learning across different types of data.

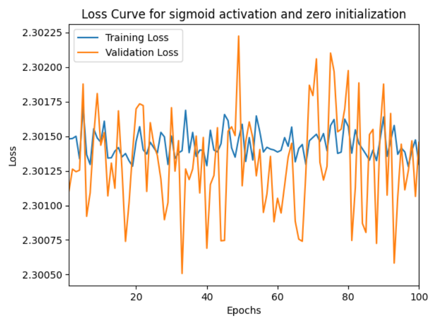
2.3 Q2. 3) Weight Initialization

The class includes three initialization schemes: zero, random, and normal (Gaussian). These options, especially the scaled random & normal initialization, help prevent issues like vanishing or exploding gradients, ensuring efficient training and better convergence. I referred to this article on weight initialization techniques.

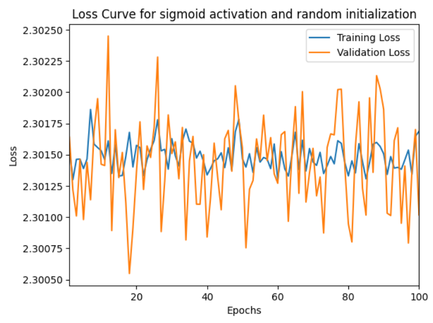
2.4 Q2. 4) Model Training & Grid Search

I trained the custom neural network on the MNIST dataset using a four-layer architecture with hidden layer sizes set to [256, 128, 64, 32]. The dataset was split into training, validation, and test sets with an 80:10:10 ratio. I implemented a grid search over activation functions (sigmoid, tanh, relu, and leaky relu) and weight initializations (zero, random, and normal), yielding a total of 12 different model configurations. Each model was trained with a batch size of 128, a learning rate of 0.02, and for up to 100 epochs (stopping early if necessary to prevent overfitting).

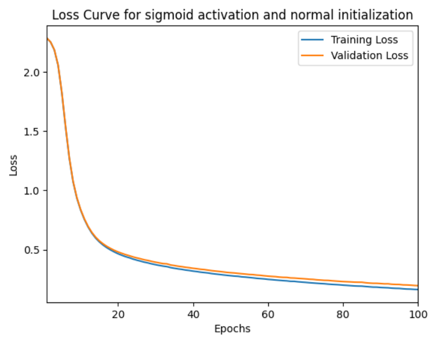
Training and validation losses were recorded across epochs for each configuration to assess performance, and each trained model was saved as a .pkl file for reproducibility. The plots & observations are given below :-



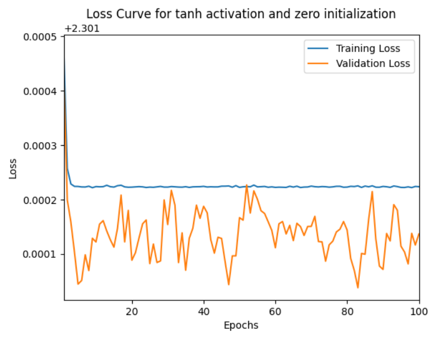
(a) Sigmoid - Zero



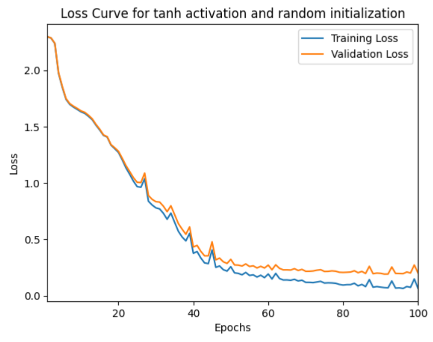
(b) Sigmoid - Random



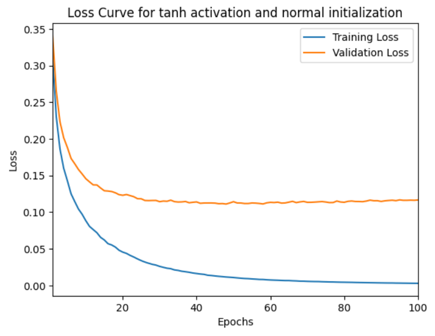
(c) Sigmoid - Normal



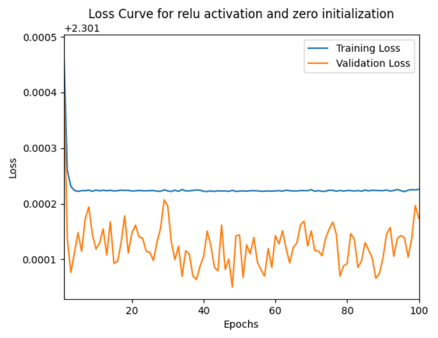
(d) Tanh - Zero



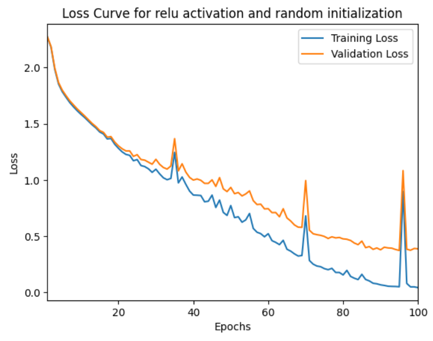
(e) Tanh - Random



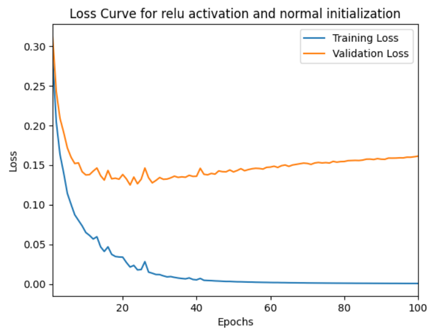
(f) Tanh - Normal



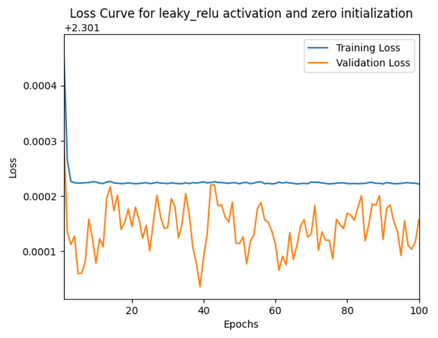
(g) ReLU - Zero



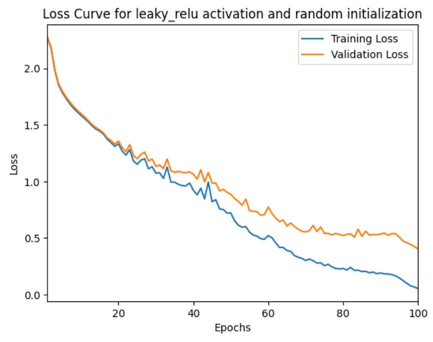
(h) ReLU - Random



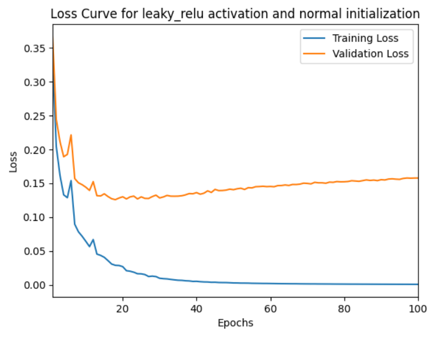
(i) ReLU - Normal



(j) Leaky ReLU - Zero



(k) Leaky ReLU - Random



(l) Leaky ReLU - Normal

Figure 5: Training and Validation Losses for Different Activation Functions and Weight Initializations

Sigmoid Activation

- **Random Initialization**: Very unstable with high spikes in both training and validation losses, yielding very low accuracy. This behavior is likely due to slow convergence and the need for a higher learning rate with the sigmoid function. **Test Accuracy**: 11.71%
- **Normal Initialization**: Converged smoothly with high accuracy, demonstrating that normal initialization supports stability and effective learning with sigmoid. **Test Accuracy**: 95.11%
- **Zero Initialization**: Did not converge, showing instability and high spikes throughout training and validation losses, leading to very low accuracy. **Test Accuracy**: 11.71%

Tanh Activation

- **Random Initialization**: Converged gracefully with minimal spikes and achieved high accuracy, indicating that the tanh function benefits from scaled random initialization for stability. **Test Accuracy**: 95.43%
- **Normal Initialization**: Showed the smoothest convergence and the highest accuracy across all configurations, suggesting that the combination of tanh and normal initialization is optimal for convergence and performance. **Test Accuracy**: 97.33%
- **Zero Initialization**: Training loss converged quickly, but validation loss was unstable with frequent spikes, leading to low accuracy. **Test Accuracy**: 11.71%

ReLU Activation

- **Random Initialization**: Converged gracefully with minor spikes in loss, achieving high accuracy. Training and validation losses indicate strong convergence. **Test Accuracy**: 94.16%
- **Normal Initialization**: Completely converged but showed slight overfitting, indicated by an increase in validation loss. Achieved one of the highest accuracies. **Test Accuracy**: 97.26%
- **Zero Initialization**: Training loss converged quickly, but validation loss was unstable with frequent spikes, resulting in low accuracy. **Test Accuracy**: 11.71%

Leaky ReLU Activation

- **Random Initialization**: Converged gracefully with few spikes and achieved high accuracy, showcasing the ability of leaky ReLU to support stable convergence. **Test Accuracy**: 94.60%
- **Normal Initialization**: Completely converged with slight overfitting, similar to the behavior seen in ReLU, resulting in high accuracy. This configuration was one of the best performers. **Test Accuracy**: 97.03%
- **Zero Initialization**: Training loss converged quickly, but validation loss remained unstable, resulting in low accuracy. This outcome aligns with the poor performance observed in other zero-initialized models. **Test Accuracy**: 11.71%

Weight Initialization

- **Normal Initialization (He Initialization):** Performed best across all activation functions, achieving high accuracy and smooth convergence in training and validation losses. It provided consistent stability and prevented vanishing/exploding gradient issues, particularly with ReLU, Tanh, and Leaky ReLU.
- **Random Initialization (0.01 Scaling):** Showed good performance with minimal spikes and smooth convergence. However, it generally yielded slightly lower accuracy compared to normal initialization.
- **Zero Initialization:** Demonstrated the poorest performance across all activation functions, resulting in frequent spikes in validation loss and significantly low accuracy. This initialization was especially problematic due to symmetry-breaking issues that hindered effective weight updates.

Activation Functions

- **ReLU and Leaky ReLU:** Both activation functions performed exceptionally well with normal and random initializations, achieving high accuracy and smooth convergence. Minor differences in loss spikes suggested that Leaky ReLU might provide marginal improvements in convergence stability.
- **Tanh:** Showed the smoothest convergence with normal initialization and achieved high accuracy, marking it as one of the most effective activation functions in this configuration.
- **Sigmoid:** Encountered significant convergence issues and low accuracy, likely due to the low learning rate and gradient vanishing problems. It was less suited for this model setup compared to ReLU, Leaky ReLU, and Tanh.

In summary, the best combination was **Tanh with Normal Initialization**, which achieved high accuracy and smooth convergence without frequent loss spikes. Other effective combinations included **ReLU/Leaky ReLU with Normal or Random Initialization**. These findings emphasize the importance of proper initialization (such as **He Initialization** for ReLU/Leaky ReLU) for stable convergence. Zero initialization should be avoided due to its inability to break symmetry in weight updates, resulting in poor performance.

3 Section C

We have been given the "Fashion MNIST" dataset, which is a collection of 70,000 grayscale images, each 28x28 pixels (784 features) representing 10 different categories of clothing. It includes 60,000 training images and 10,000 testing images, with each image labeled from 0 to 9 corresponding to different classes.

There are no null values in the dataset.

The dataset is split into training & testing sets by selecting the first 8000 samples from the training data and the first 2000 samples from the testing data. Separate features and labels for both training and testing sets were created.

3.1 Q3. 1) Preprocessing & Normalization

I normalized the pixel values of the dataset by dividing each pixel value by 255. This scales the pixel intensity values from their original range (0 to 255) to a range of 0 to 1. As the pixel values are in a well defined range [0, 255], dividing by 255 is both effective and straightforward. Normalizing helps in improving model performance and reach convergence faster.

I visualized 10 random samples from the test dataset shown below :-

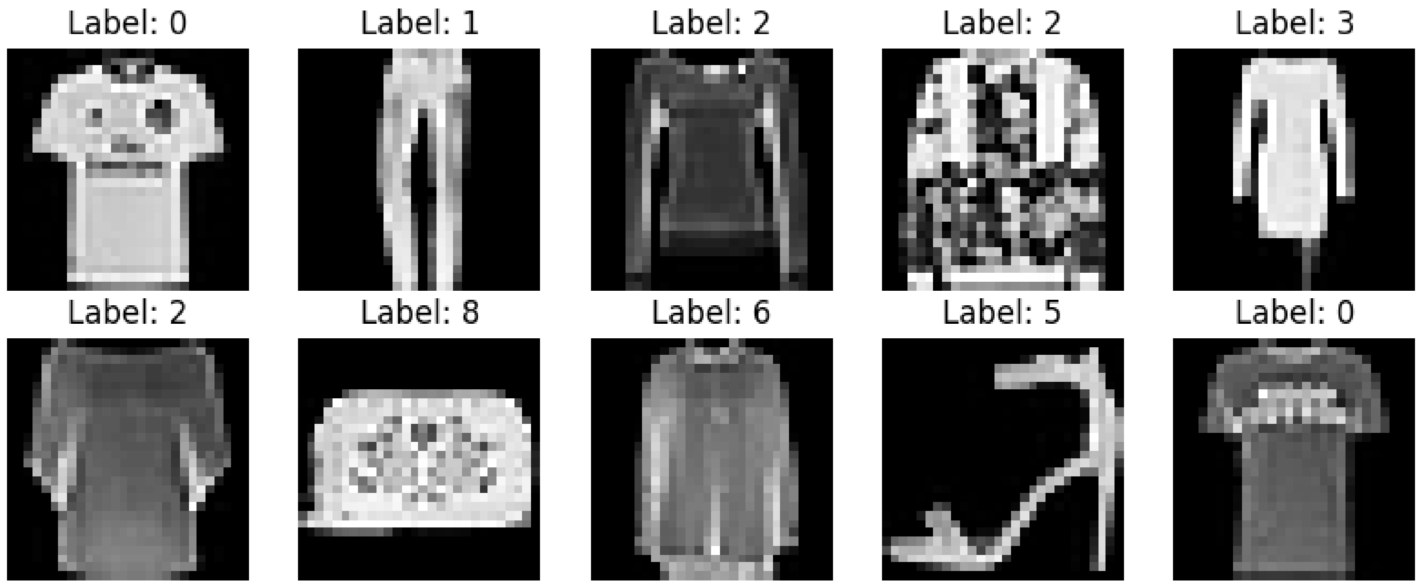
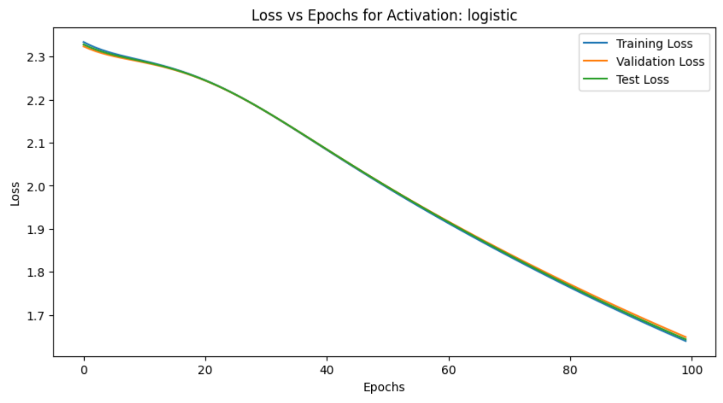


Figure 6: 10 Random Samples from the Test Dataset

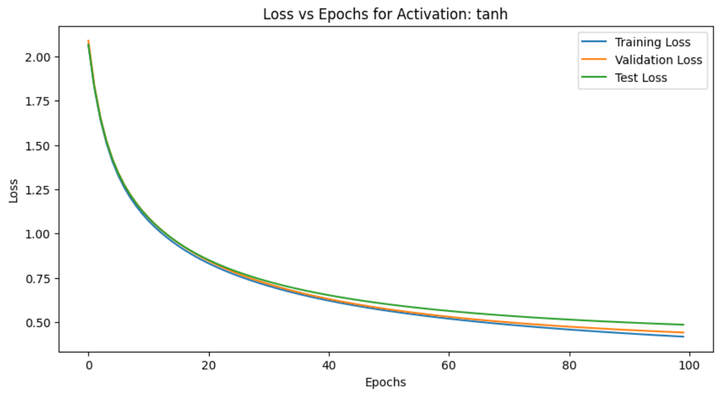
3.2 Q3. 2) Training MLP Classifier

I trained a Multi Layer Perceptron (MLP) classifier using scikitlearn's MLPClassifier. The neural network consisted of three hidden layers with sizes [128, 64, 32], trained for 100 iterations with an adam solver, batch size of 128, and learning rate of 2e-5.

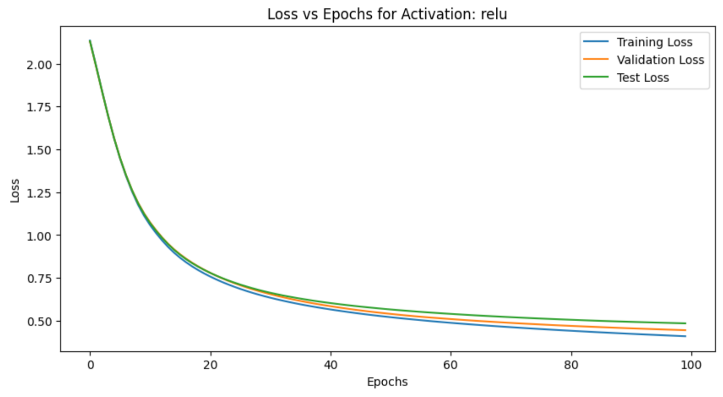
I explored four activation functions for the hidden layers: logistic, tanh, relu, and identity. The performance of each activation function was evaluated based on the training and validation loss over epochs, providing insight into convergence and generalization. The plots are given below :-



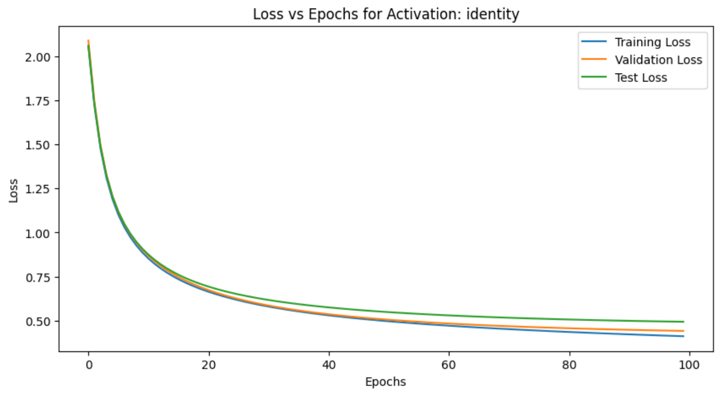
(a) Loss v/s Epochs using Logistic Loss



(b) Loss v/s Epochs using tanh Loss



(c) Loss v/s Epochs using ReLU Loss



(d) Loss v/s Epochs using Identity Loss

Figure 7: Loss v/s Epochs using Different Loss Functions

Validation Loss (logistic): 1.6488
Validation Loss (tanh): 0.4419
Validation Loss (relu): 0.4452
Validation Loss (identity): 0.4412
Test Loss (logistic): 1.6435
Test Loss (tanh): 0.4859
Test Loss (relu): 0.4851
Test Loss (identity): 0.4931
Test Accuracy (logistic): 0.4475
Test Accuracy (tanh): 0.8355
Test Accuracy (relu): 0.8340
Test Accuracy (identity): 0.8250

Based on the results, the logistic activation function showed poor performance across all metrics, while tanh, relu, and identity performed comparably well. Among these, tanh achieved the best test accuracy (83.55%) and high validation loss (0.4419), making it the most balanced choice overall.

3.3 Q3. 3) Hyperparameter Tuning

For hyperparameter tuning, a grid search was conducted using the tanh activation function, optimizing the solver, learning rate, and batch size, evaluating 36 configurations across 3-fold cross-validation (108 total fits). The best-performing combination was found to be:

- **Solver**: **adam**
- **Learning Rate**: **0.001**
- **Batch Size**: **64**

This configuration achieved a cross-validation accuracy of **86.33%**, demonstrating improved model performance with these hyperparameters.

3.4 Q3. 4) Regeneration Task with MLPRegressor

For the regeneration task, I trained a 5-layer MLPRegressor, the hidden layers were set to sizes [128, 64, 32, 64, 128]. We experimented with two activation functions, relu and identity, and used adam as the solver with a learning rate of 2e-5.

The hidden layer sizes were set to 128, 64, and 32 to create a bottleneck structure that compresses and then reconstructs the input images. This design helps the model learn the most important features in the compressed center layers
Since this is a regeneration task, the model aims to reproduce each input image rather than predict a separate target. Therefore, both the input and target are X_train_main, and the Mean Squared Error (MSE) between these evaluates the accuracy of the reconstruction.

I plotted the training and validation loss per epoch for both activation functions :-

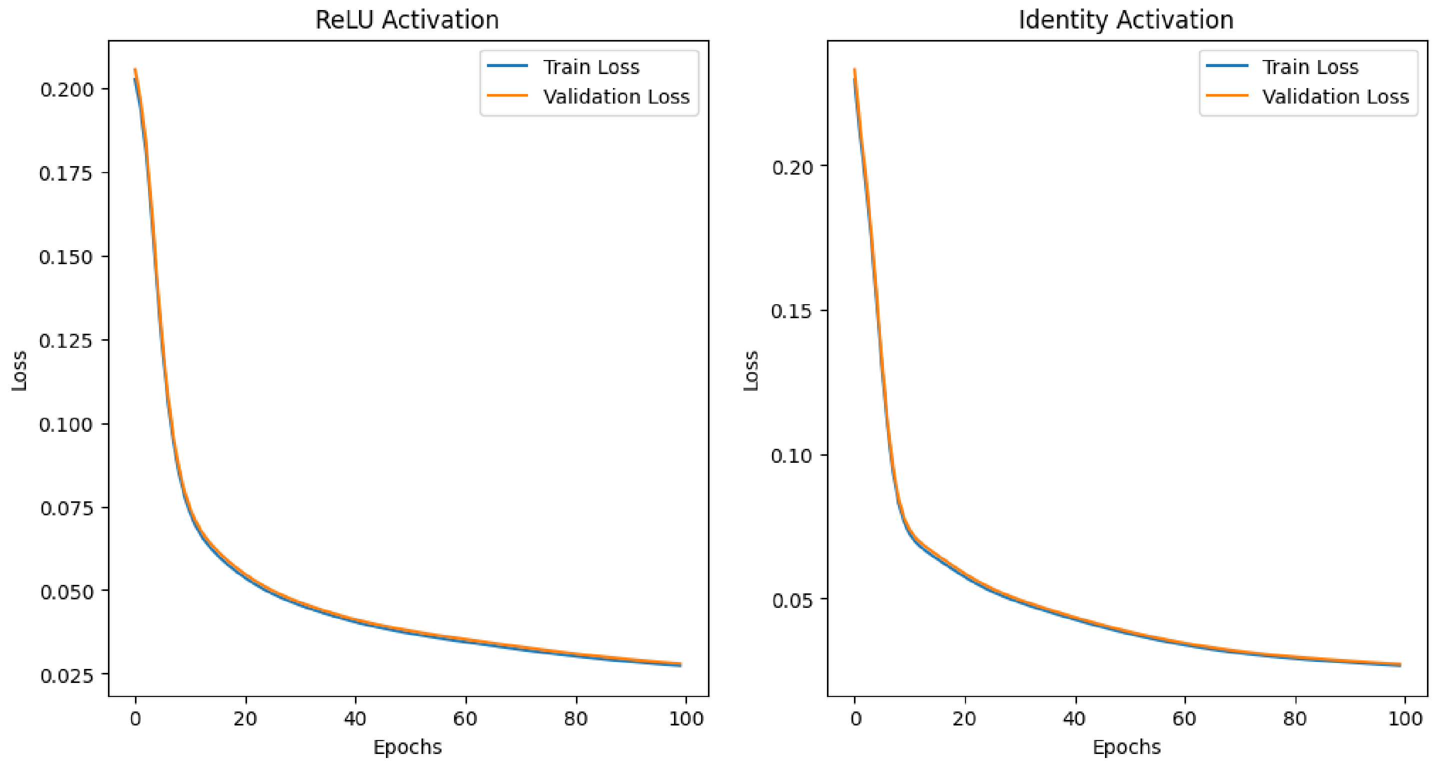


Figure 8: Loss v/s Epochs after regeneration

After training, I visualized 10 test samples alongside their regenerated versions :-

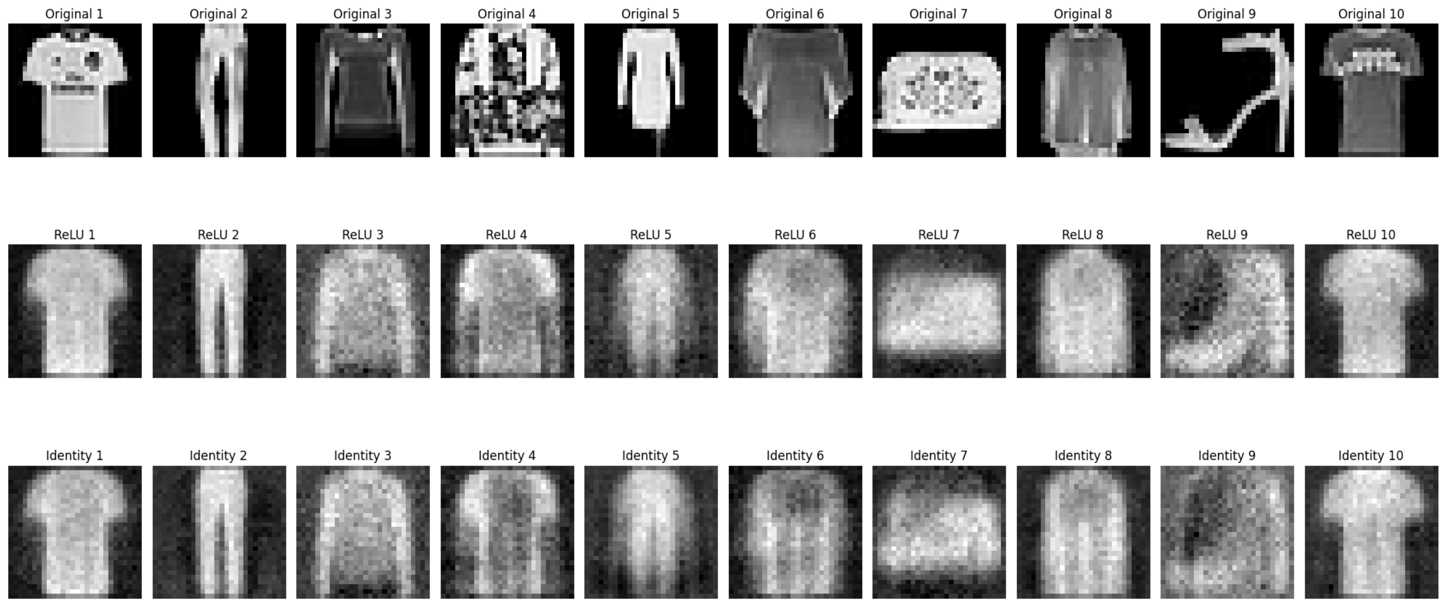


Figure 9: Visualization of Regenerated Images

The comparison revealed that :-

The regenerated images using the ReLU activation function appear to capture the general shapes and outlines of the original items but lack finer details. The images look somewhat blurry and less defined, particularly around edges and textures, which may be due to the tendency of ReLU to produce sparse activations.

The regenerated images with the Identity activation function show slightly better preservation of pixel intensity and smoother gradients, but still lack some finer details and sharpness. Overall, the Identity activation provides a slightly better reconstruction, possibly because it allows gradients to pass through more directly, helping maintain subtle variations.

In summary, while neither ReLU nor Identity perfectly recreates the originals, Identity seems to offer a marginally better visual reconstruction quality.

3.5 Q3. 5) Feature Extraction and Classifier Training

From the two trained neural networks (using ReLU and Identity activations), I extracted a feature vector of size a = 32. This was done by performing a forward pass through the networks and taking the output from the last hidden layer as the extracted features. Using the extracted feature vectors as the new representation of the images, I trained two MLP classifiers, each with two hidden layers of size 32. The MLPs were trained with 200 iterations using the Adam solver and a learning rate of 2e-5, with the tanh activation, as this performed best in part 2 of this question. Classifiers were evaluated on train and test sets, yielding train and test accuracies for both ReLU and Identity activations :-

- **ReLU Activation**:
  - Train Accuracy: 73.0%
  - Test Accuracy: 72.0%
- **Identity Activation**:
  - Train Accuracy: 73.0%
  - Test Accuracy: 73.7%

In comparison to the classifier accuracies in part 2 :-

- **Tanh Activation**: 83.6% test accuracy
- **ReLU Activation**: 83.4% test accuracy
- **Identity Activation**: 82.5% test accuracy

Despite using reduced features (a 32-dimensional feature vector) instead of the full 784-dimensional image vector, the classifiers still achieved reasonably high test accuracies (72–73.7%), which is close to the performance in part 2. This can be because of :-

Dimensionality Reduction with Relevant Features: The feature extraction process in the bottleneck layer of the neural networks likely captured essential patterns and structures in the images, reducing the data size while retaining a substantial amount of discriminative power, allowing for effective classification.

Generalization from Reduced Representation: Training on reduced features often helps with generalization by reducing overfitting. This simplification helps the model to ignore the noise in the high-dimensional data.