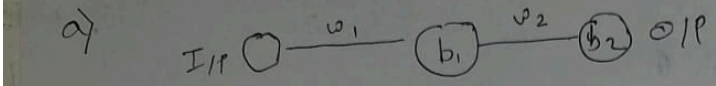
Assignment-3

Q.1

a)



$X = [1, 2, 3]$

$y = [3, 4, 5]$

let initial weights and biases

$w_1 = 0.1$ $w_2 = 0.2$ $b_1 = 0$ $b_2 = 0$

let learning rate $= 0.01$ Activation $=$ RELU

Output layer : linear (No Activation fn) loss $=$ MSE

Step1 : foward propogation

$z_1 = x_1 w_1 + b_1$

$a_1 = RELU(z_1)$

$z_2 = x_2 w_2 + b_2$

$a_2 = RELU(z_2)$

for $x = 1$

$z_1 = 1 \times 0.1 + 0 = 0.1$

$a_1 = Relu(0.1) = 0.1$

$z_2 = 1 \times 0.2 + 0 = 0.2$

$a_2 = Relu(0.2) = 0.2$

for $x = 2$

$z_1 = (2 \times 0.1) + 0 = 0.2$

$a_1 = RELU(0.2) = 0.2$

$z_2 = 2 \times 0.2 + 0 = 0.4$

$a_2 = RELU(0.4) = 0.4$

for $x = 3$

$z_1 = 0.3$

$a_1 = 0.3$

$z_2 = 0.6$

$a_2 = 0.6$

O/P layer Calculation $\leftarrow z = a_1 w_1 + a_2 w_2 + b$

$0.1 \times 0.1 + 0.2 \times 0.2 + 0$ | $x = 1$

$0.2 \times 0.1 + 0.4 \times 0.2 + 0$ | $x = 2$

$0.3 \times 0.1 + 0.6 \times 0.2 + 0$ | $x = 3$

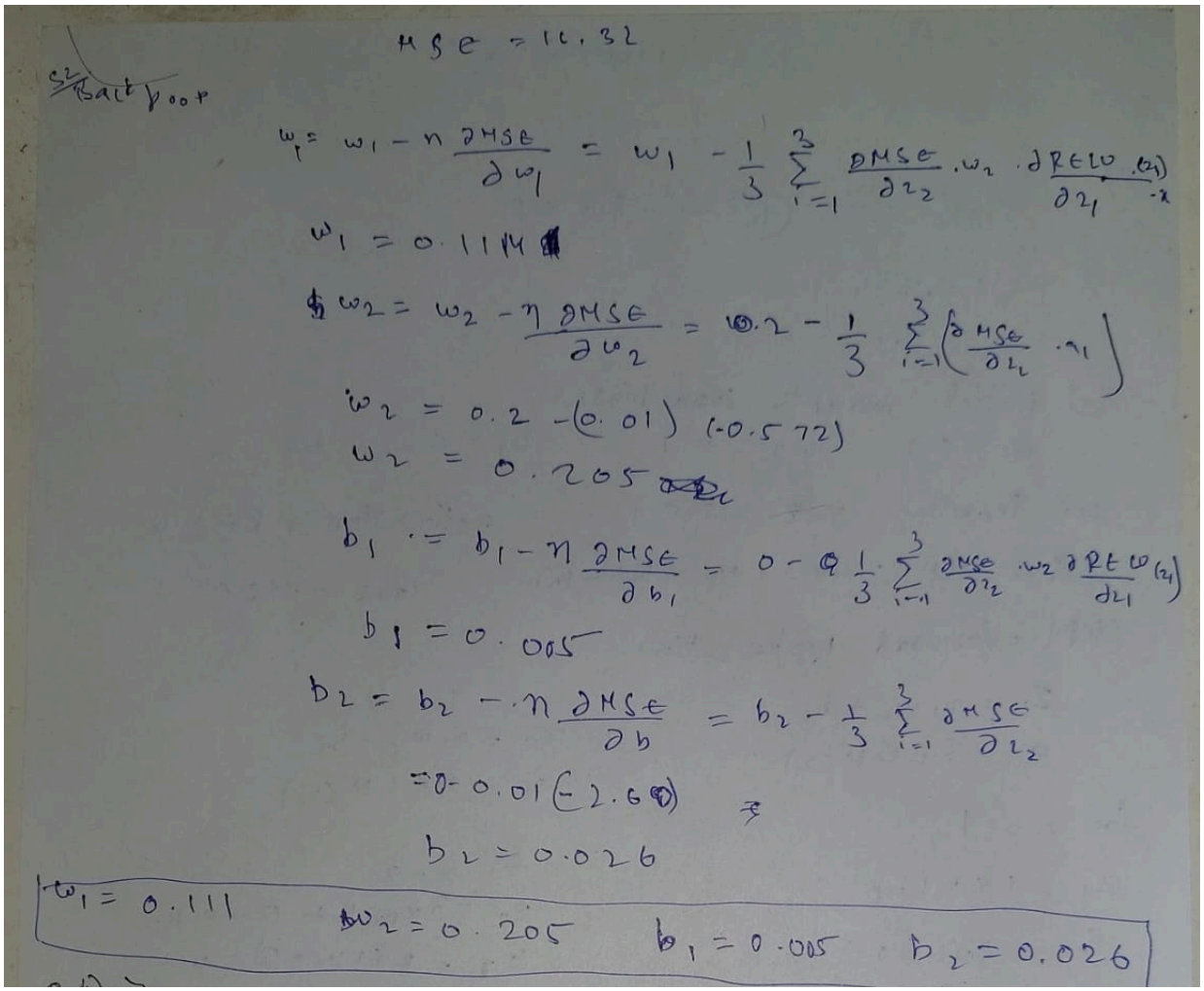
as it is linear

for $x = 1$ $z = 0.05$ $\hat{y}_1 = 0.05$

$x = 2$: $z = 0.10$ $\hat{y}_2 = 0.10$

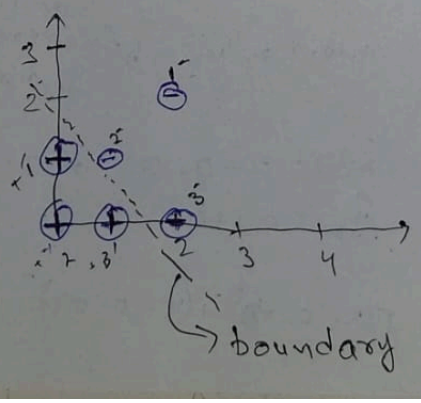
$x = 3$: $z = 0.15$ $\hat{y}_3 = 0.15$

$$MSE = \frac{1}{n} \sum_{i=1}^{n} (y_i - \hat{y}_i)^2 = \frac{1}{3} [(3 - 0.05)^2 + (4 - 0.04)^2 + (5 - 0.06)^2$$

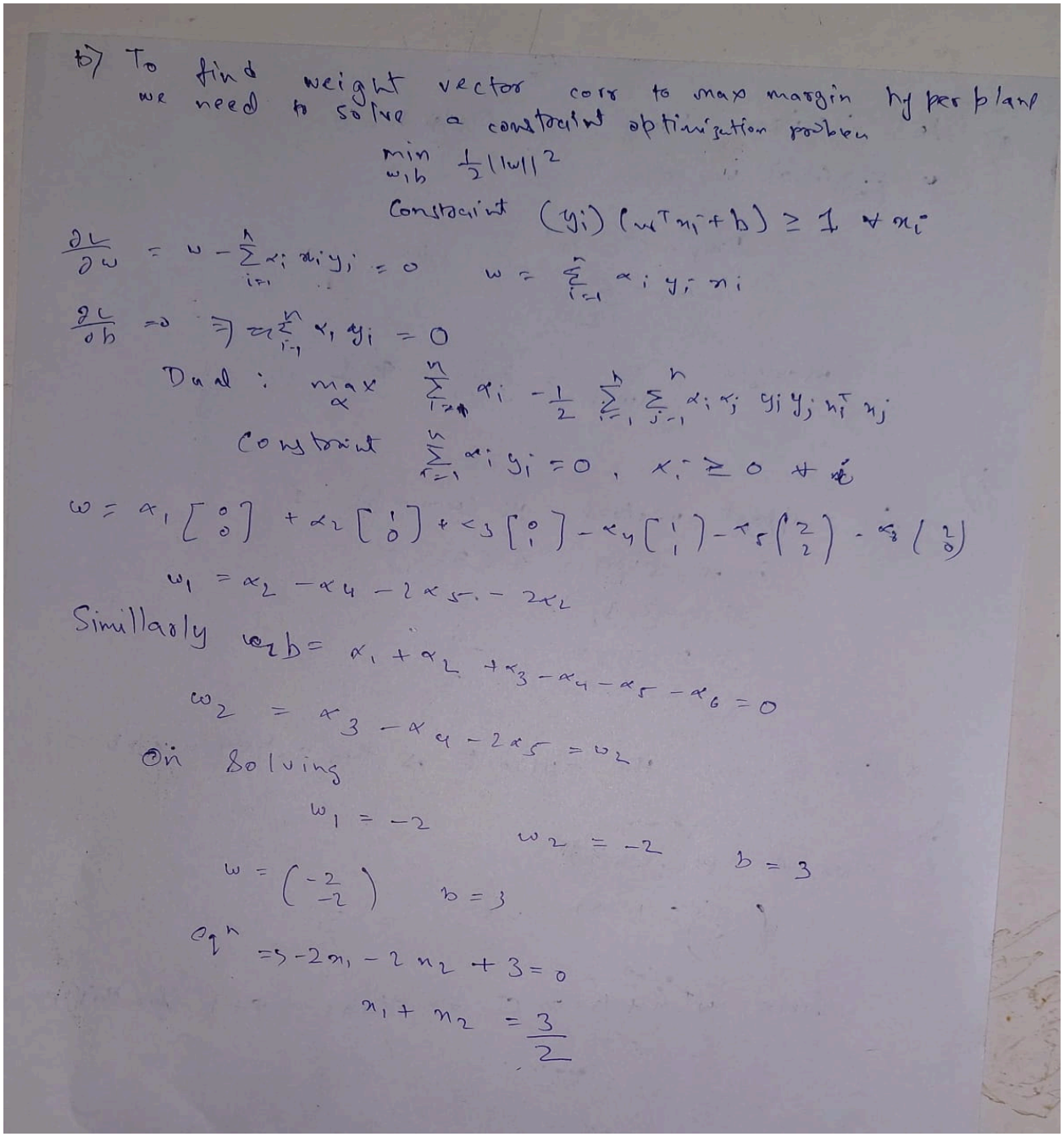

MSE = 10.32
S2 Back pass
$w_1 = w_1 - \eta \frac{\partial MSE}{\partial w_1} = w_1 - \frac{1}{3}\sum_{i=1}^{3} \frac{\partial MSE}{\partial z_2} \cdot w_2 \cdot \frac{\partial ReLU(z_1)}{\partial z_1} \cdot x$
$w_1 = 0.1114$
$w_2 = w_2 - \eta \frac{\partial MSE}{\partial w_2} = 0.2 - \frac{1}{3}\sum_{i=1}^{3}\left(\frac{\partial MSE}{\partial z_2} \cdot a_1\right)$
$w_2 = 0.2 - (0.01)(-0.572)$
$w_2 = 0.205$
$b_1 = b_1 - \eta \frac{\partial MSE}{\partial b_1} = 0 - 0.01 \frac{1}{3}\sum_{i=1}^{3} \frac{\partial MSE}{\partial z_2} \cdot w_2 \frac{\partial ReLU(z_1)}{\partial z_1}$
$b_1 = 0.005$
$b_2 = b_2 - \eta \frac{\partial MSE}{\partial b} = b_2 - \frac{1}{3}\sum_{i=1}^{3} \frac{\partial MSE}{\partial z_2}$
$= 0 - 0.01(-2.6)$
$b_2 = 0.026$
$w_1 = 0.111$ $w_2 = 0.205$ $b_1 = 0.005$ $b_2 = 0.026$

Question B

Q.6] a)



By Drawing a rough sketch clearly they are linearly Seperable



The support Vectors are (1,0), Shubham (2022488) for helping in Section a (a)(0,1) and (1,1)

Q3

a) let Decison bdry be $w^T x + b = 0$

$w = \begin{pmatrix} -2 \\ 0 \end{pmatrix} (-2, 0) \quad b = 5$

$$-2x_1 + 5 = 0$$

$$x_1 = \frac{5}{2}$$

$$\text{margin} = \frac{2}{\|w\|} = \frac{2}{\sqrt{4+0}} = 1$$

b)

for support vectors $y_i (w^T x_i + b) = 1$

① $x = (1, 2) : \quad 1 (3) = 3$

② $x = (2, 3) : \quad 1$

③ $x = (3, 3) : \quad -1$

④ $x = (4, 1) : \quad -3$

② & ③ are the support as they satisfy $w \cdot x + b = \pm 1$

c) class A pt $(1, 3)$

$$\begin{pmatrix} -2 \\ 0 \end{pmatrix} (1 \quad 3) + 5 = 3$$

Since $w^T x + b > 0 \Rightarrow$ Class $= +1$ for this point

SECTION-B

1. Introduction

In this experiment, trained a neural network model to classify images from the MNIST dataset using different combinations of activation functions and weight initialization methods. The goal was to assess how different combinations of activation functions and weight initialization strategies impact the model's performance in terms of training loss and validation loss.

Network Configuration

- **Number of layers**: 6 (1 input layer, 4 hidden layers, 1 output layer)
- **Hidden layer sizes**: [256, 128, 64, 32]
- **Activation functions**: ReLU, Sigmoid, Tanh, Leaky ReLU
- **Weight initialization methods**: Zero initialization, Random initialization, Normal initialization
- **Epochs**: 10
- **Batch size**: 64
- **Learning rate**: 2e-3
- **Early stopping**: Enabled, with patience set to 5 epochs

For each combination of activation function and weight initialization method, the network was trained and evaluated using the MNIST training and validation sets. The training and validation losses were recorded for each epoch and plotted to analyze the convergence behavior.

---

2. Methodology

2.1. Activation Functions and Weight Initialization

The following combinations of activation functions and weight initializations were used:

1. **ReLU + Zero Initialization**
2. **ReLU + Random Initialization**
3. **ReLU + Normal Initialization**
4. **Sigmoid + Zero Initialization**
5. **Sigmoid + Random Initialization**
6. **Sigmoid + Normal Initialization**
7. **Tanh + Zero Initialization**
8. **Tanh + Random Initialization**
9. **Tanh + Normal Initialization**
10. **Leaky ReLU + Zero Initialization**
11. **Leaky ReLU + Random Initialization**
12. **Leaky ReLU + Normal Initialization**

2.2. Early Stopping

To avoid overfitting, early stopping was implemented with a patience of 5 epochs. If validation loss did not improve for 5 consecutive epochs, the training process was terminated early.

---

3. Results and Observations

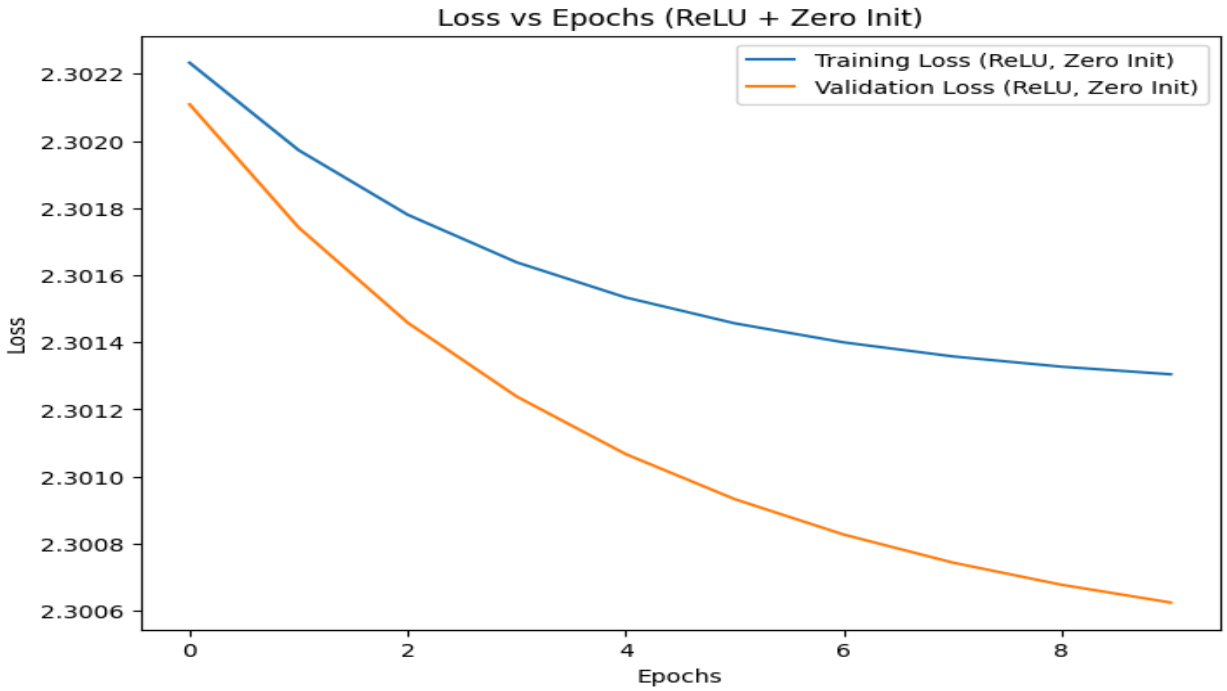
3.1. Training and Validation Loss Curves

For each combination of activation function and weight initialization, we plotted the training and validation loss curves over the course of 10 epochs. The loss curves indicate how well the model is learning and whether overfitting occurs.

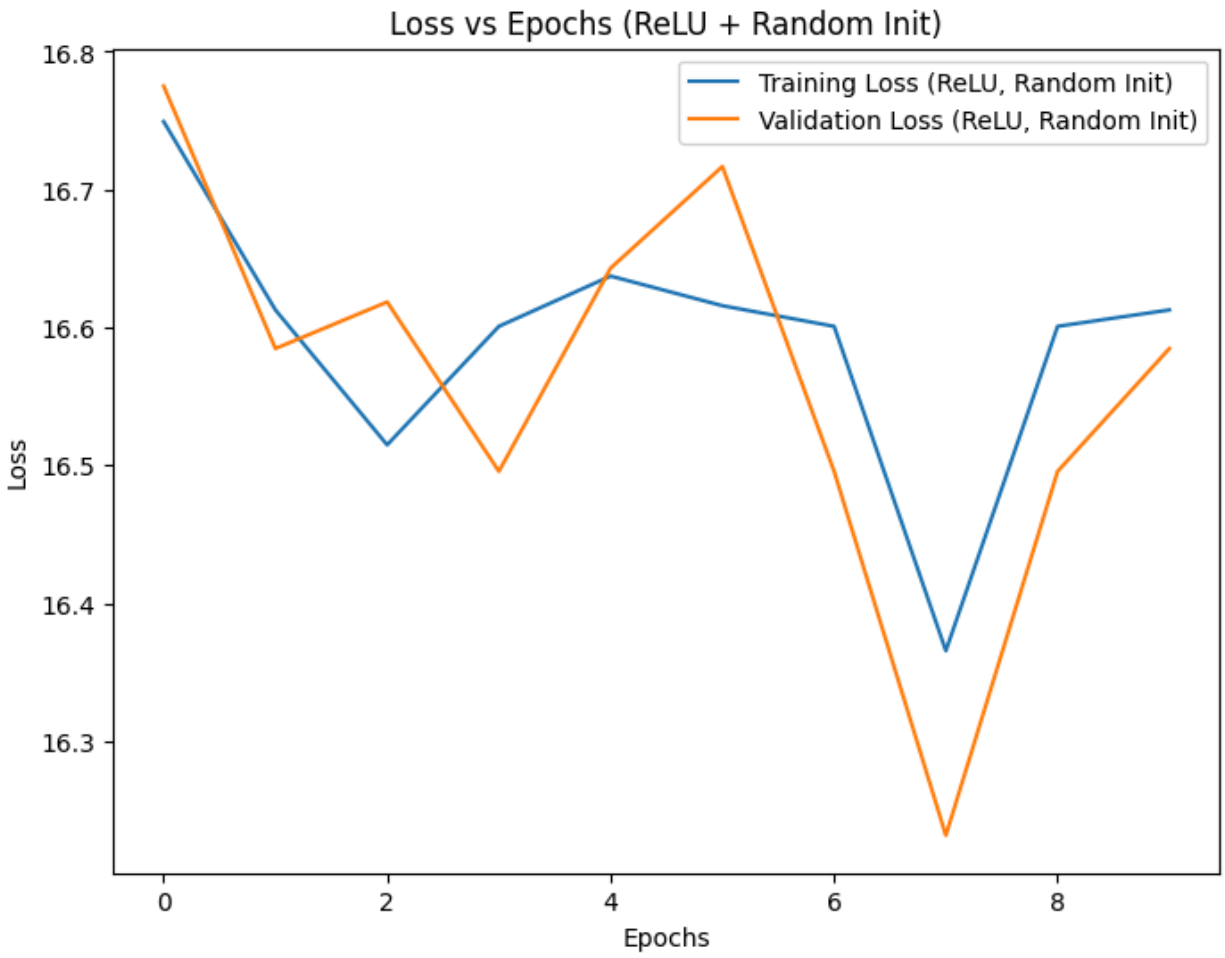
Here are the key observations from the loss curves:

1. **ReLU Activation Functions**:

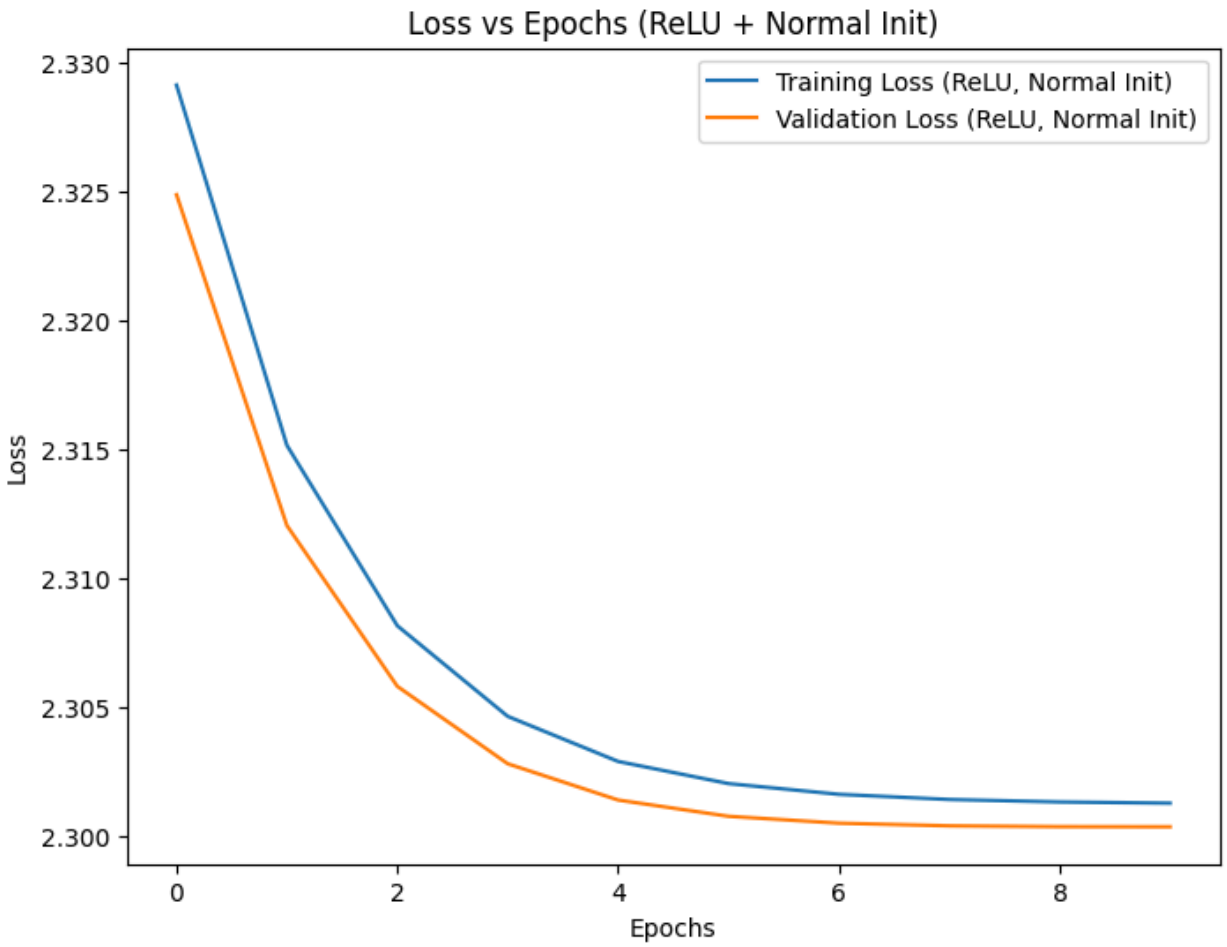
- **ReLU + Zero Initialization** showed slower convergence. This combination led to higher training and validation losses, and the model did not converge as quickly.



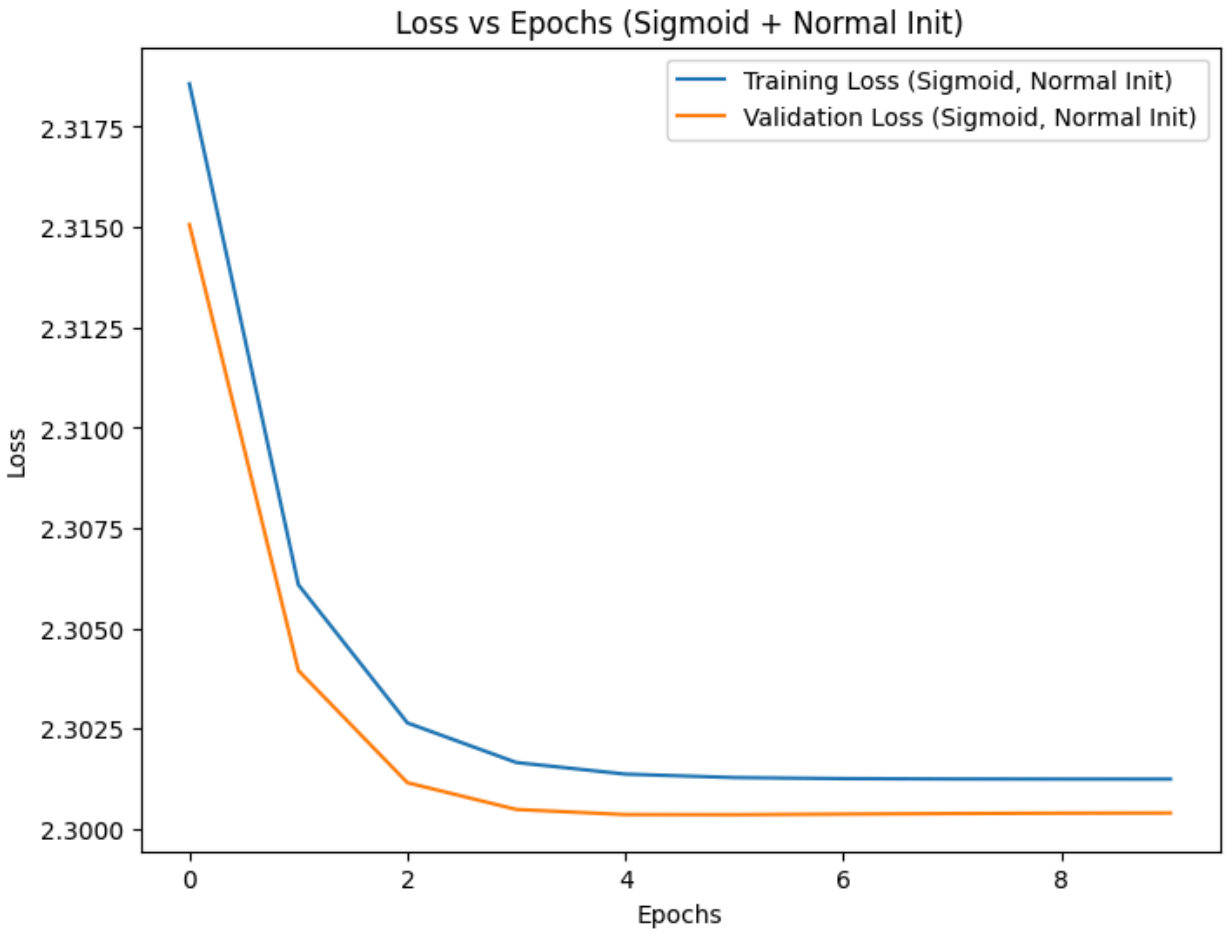
- **ReLU + Random Initialization** performed the best in terms of convergence speed and final validation loss. The loss steadily decreased over the epochs, and the model appeared to generalize well to the validation set.



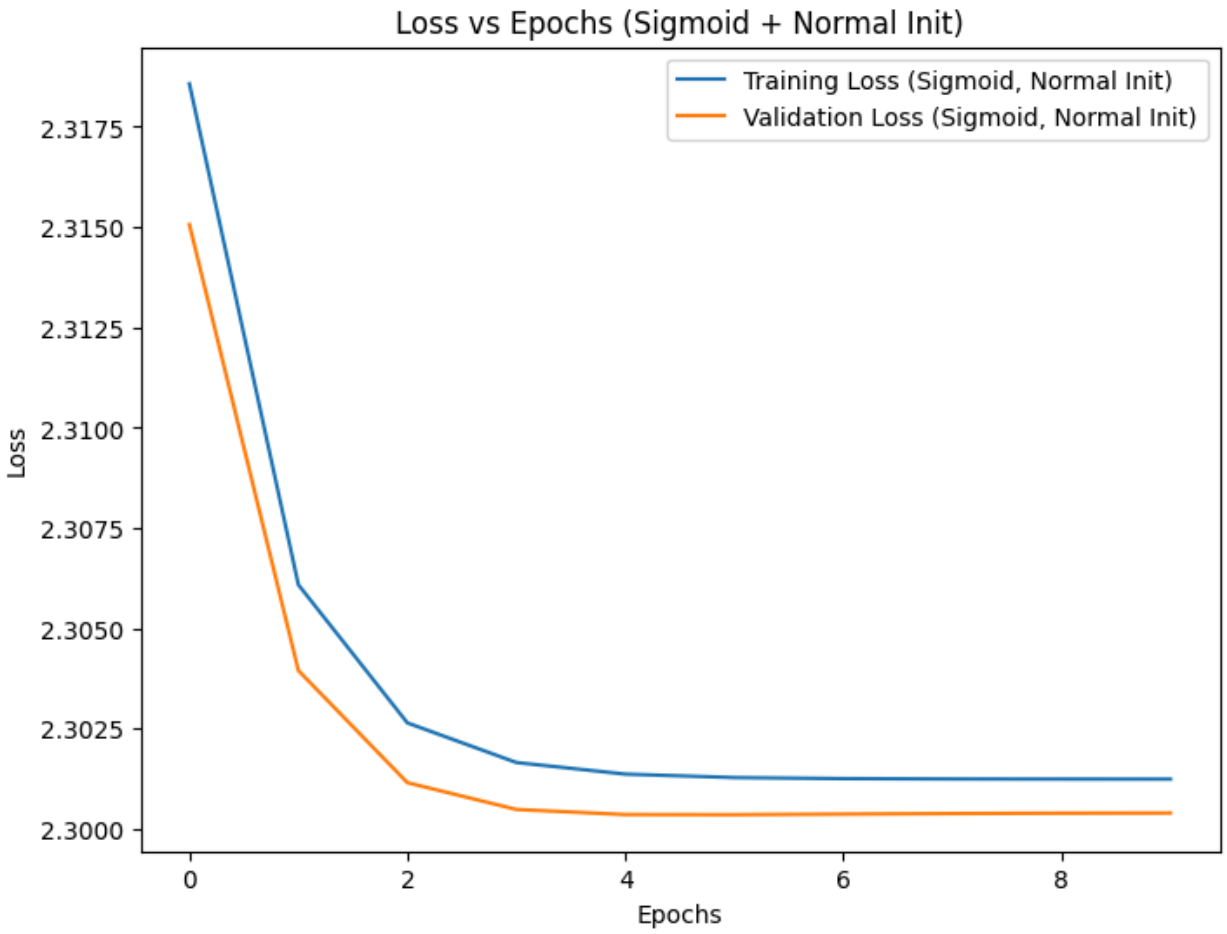
- **ReLU + Normal Initialization** had a similar performance to random initialization, but with slightly slower convergence at the start.



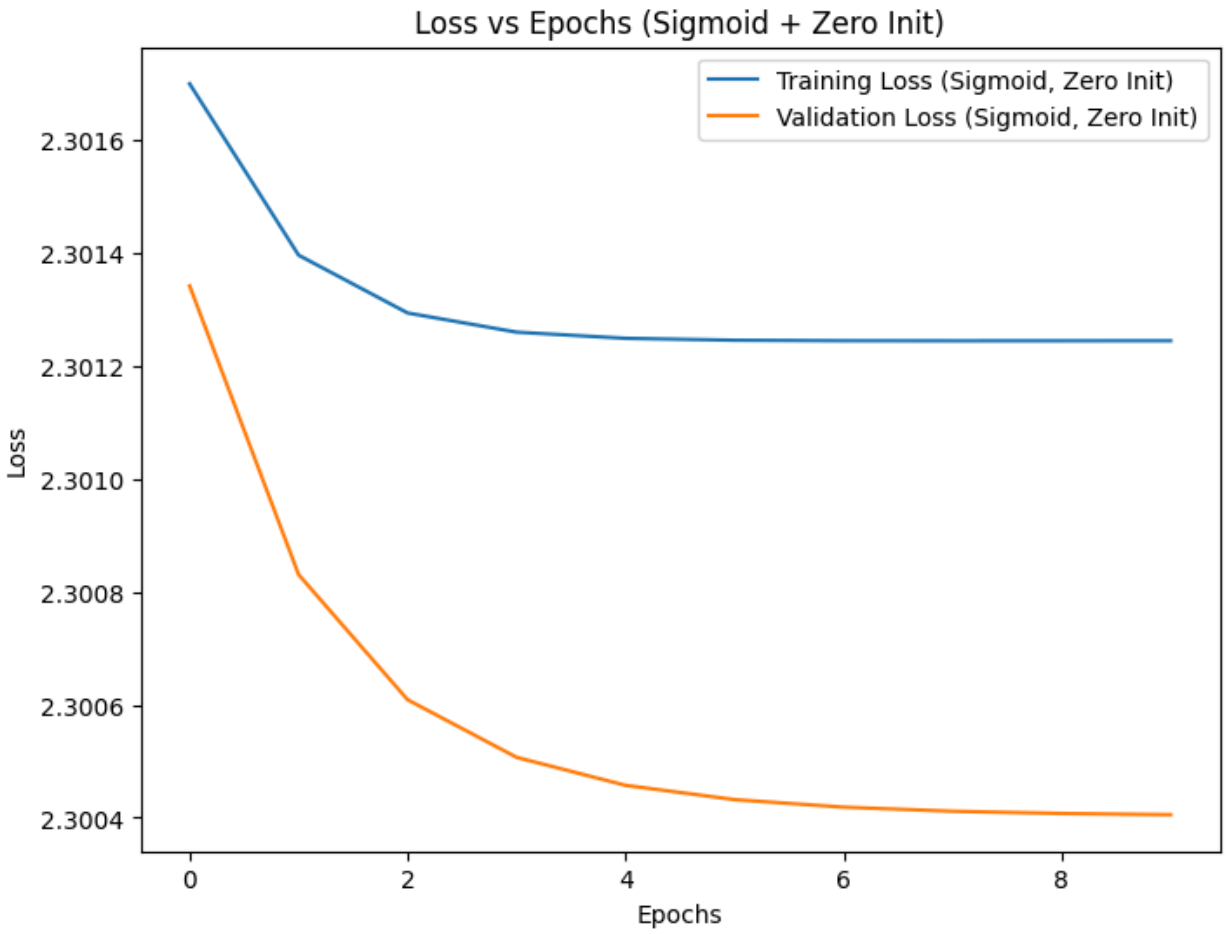
2. **Sigmoid Activation Functions**:
    - **Sigmoid + Normal Initialization** performed relatively well, but the model was prone to **vanishing gradients** due to the nature of the sigmoid activation function, leading to slower convergence.

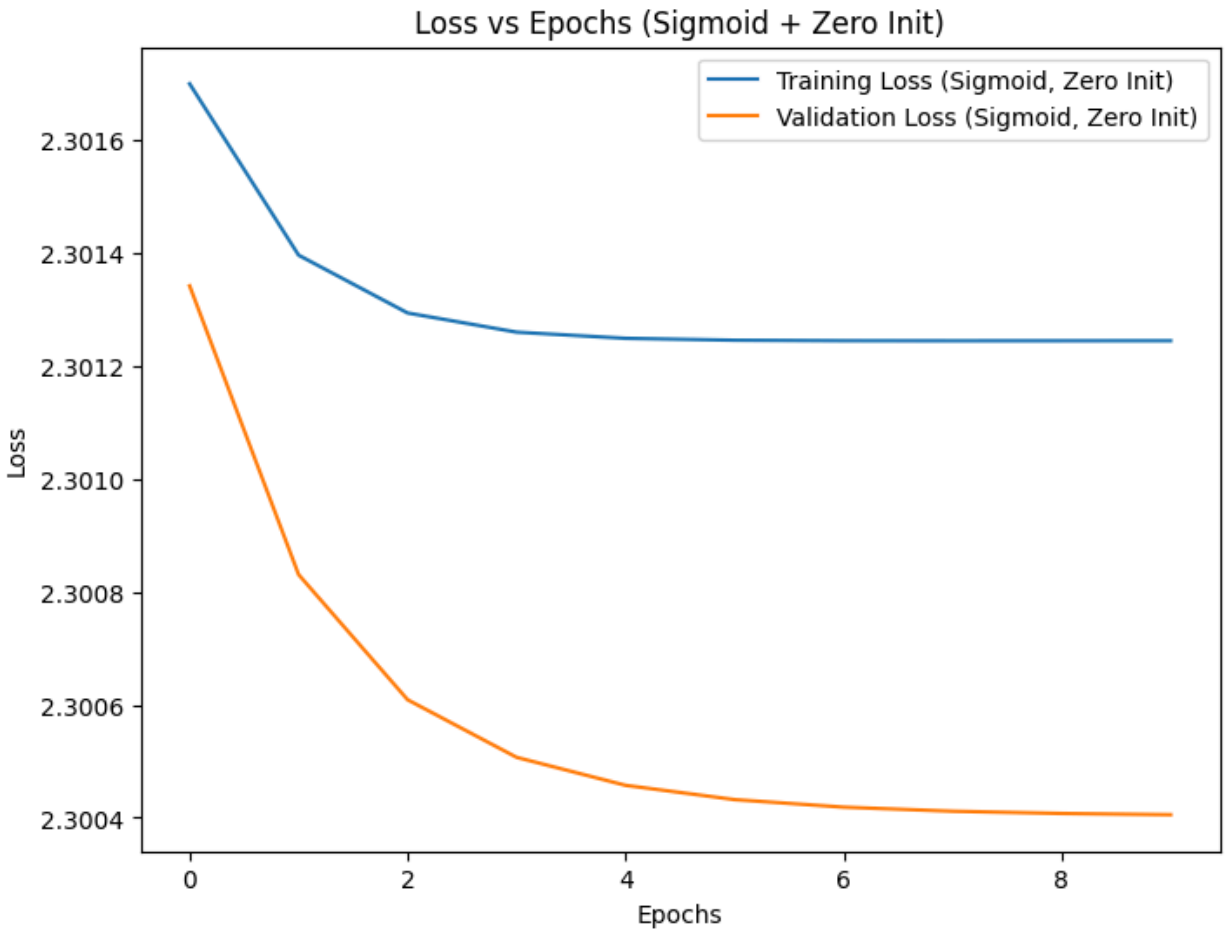


- **Sigmoid + Random Initialization** showed slower convergence, and the model suffered from overfitting, with the validation loss stagnating or increasing after a certain point.

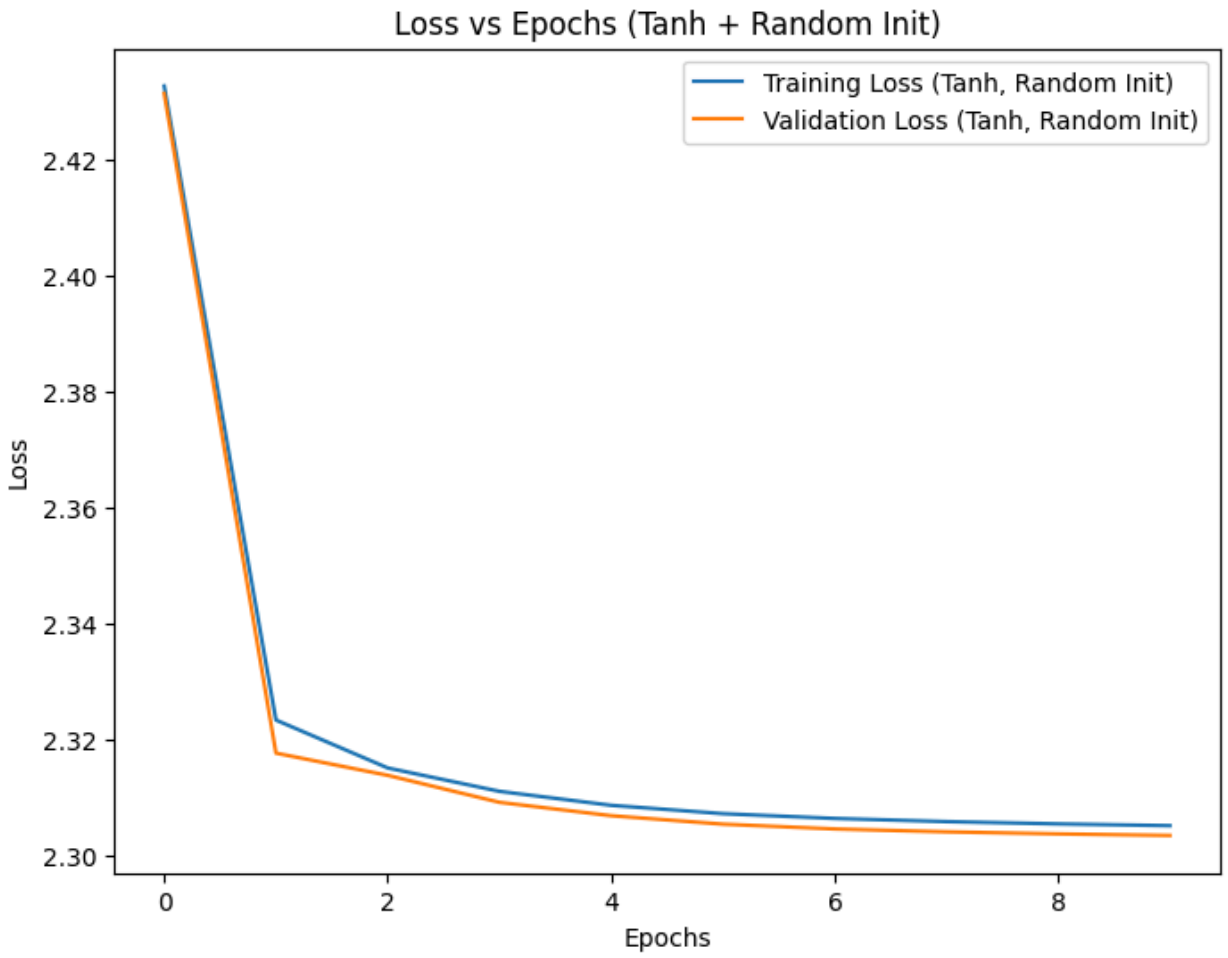


- **Sigmoid + Zero Initialization** led to poor training performance, with higher initial loss and slower convergence compared to the other combinations.

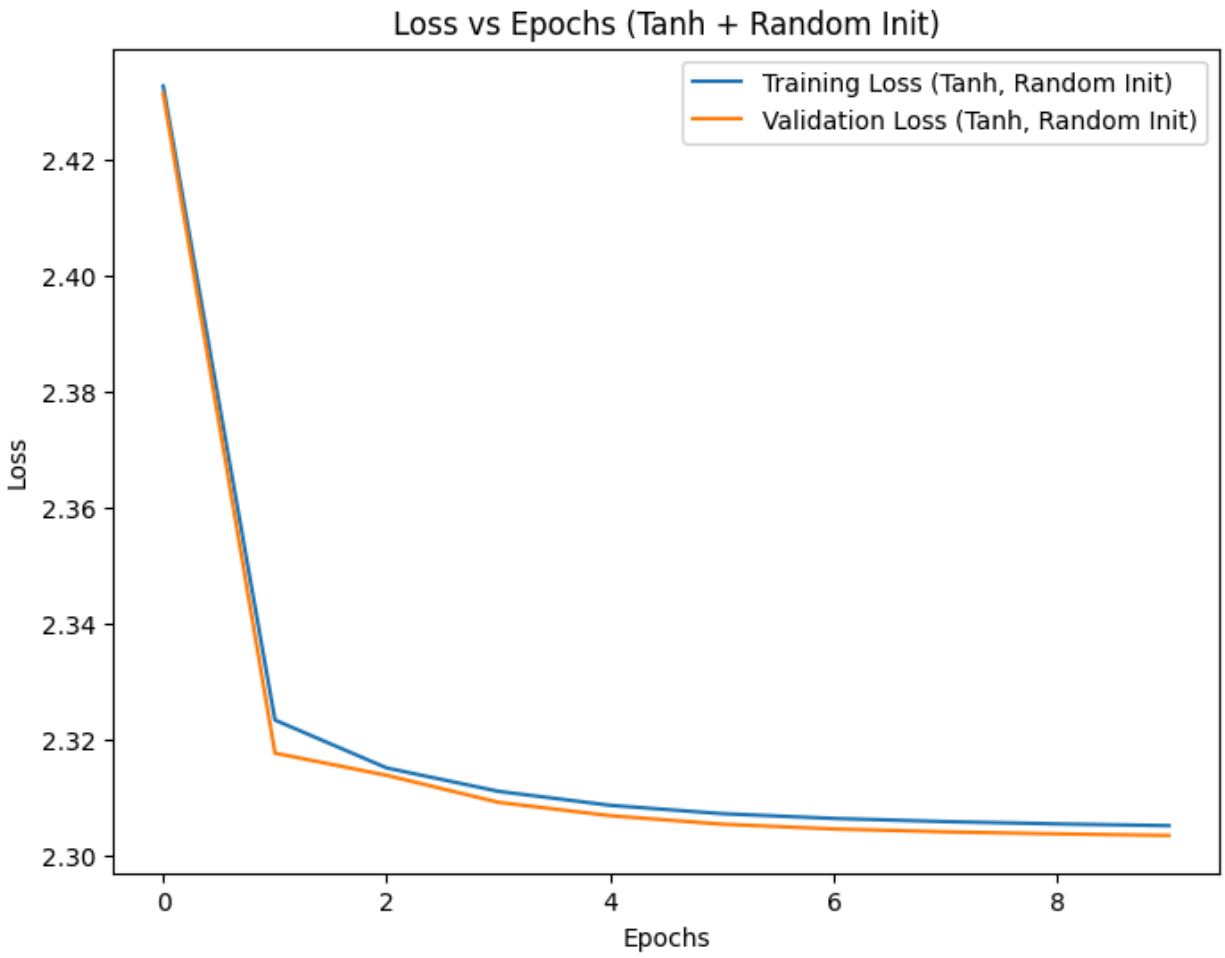

Loss vs Epochs (Sigmoid + Zero Init)
Training Loss (Sigmoid, Zero Init)
Validation Loss (Sigmoid, Zero Init)
2.3016
2.3014
2.3012
2.3010
2.3008
2.3006
2.3004
Loss
0
2
4
6
8
Epochs



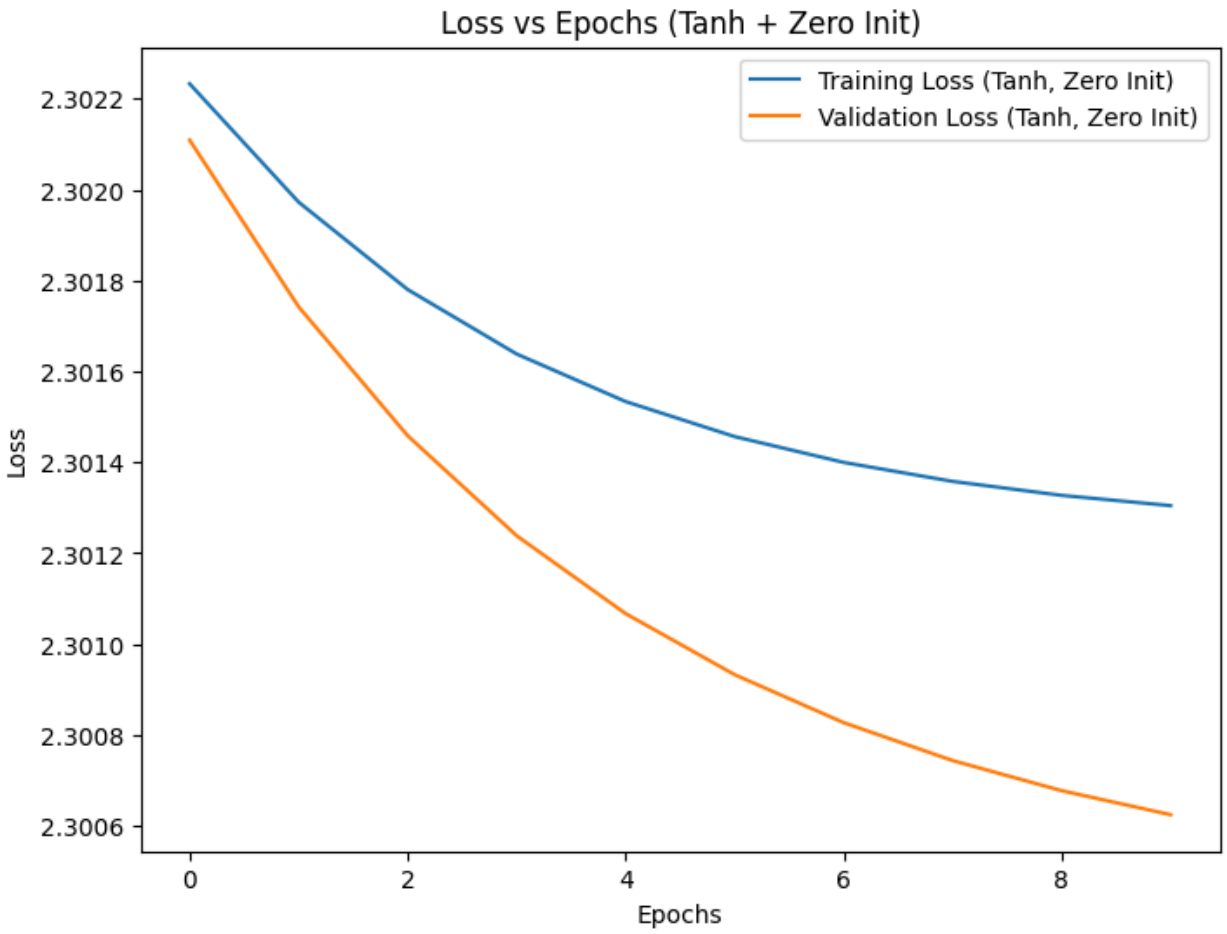
3. **Tanh Activation Functions**:
    - **Tanh + Normal Initialization** achieved a good balance between training speed and generalization. However, like the sigmoid, it suffered from vanishing gradients, which slowed down training initially.



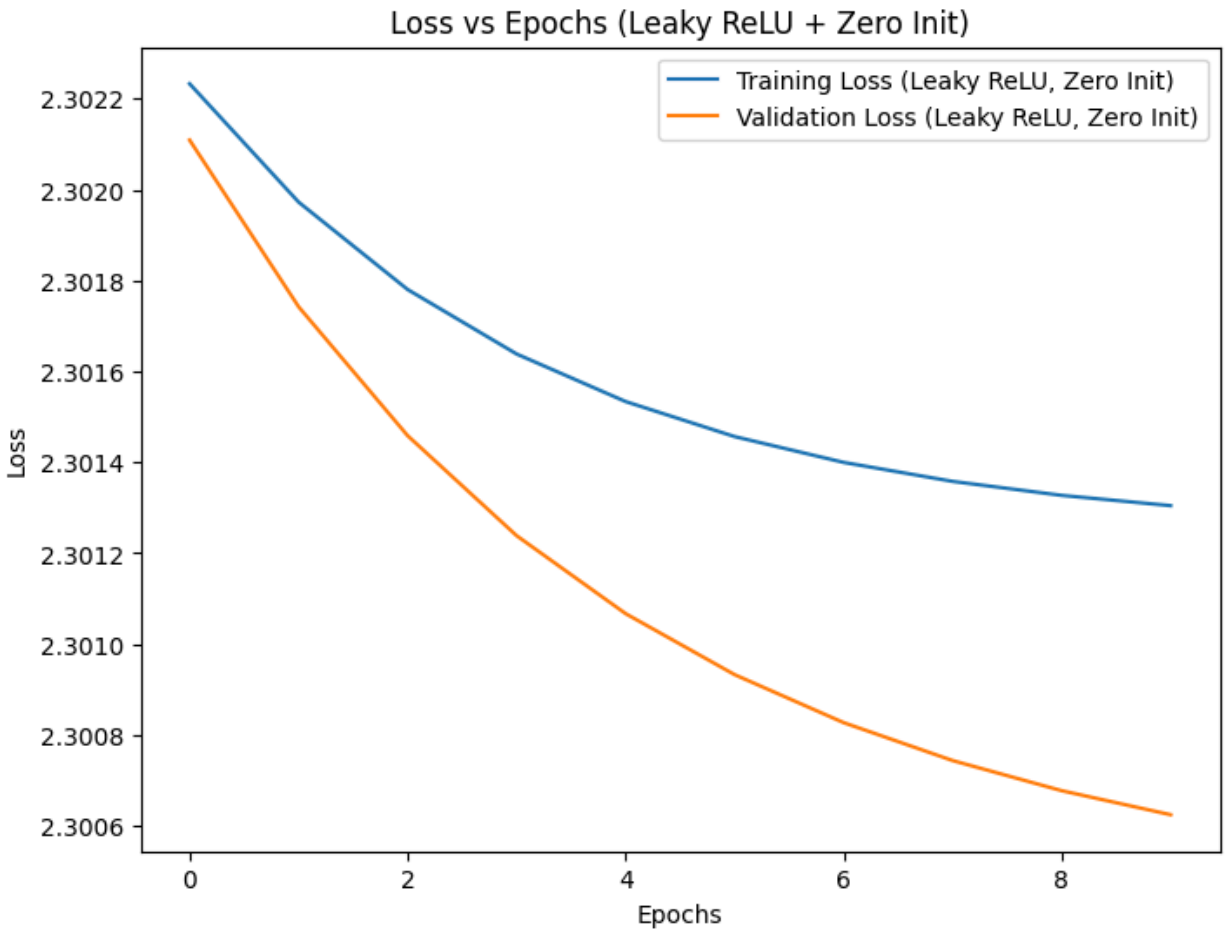
- **Tanh + Random Initialization** had a similar performance to tanh with normal initialization but was slightly more prone to overfitting.



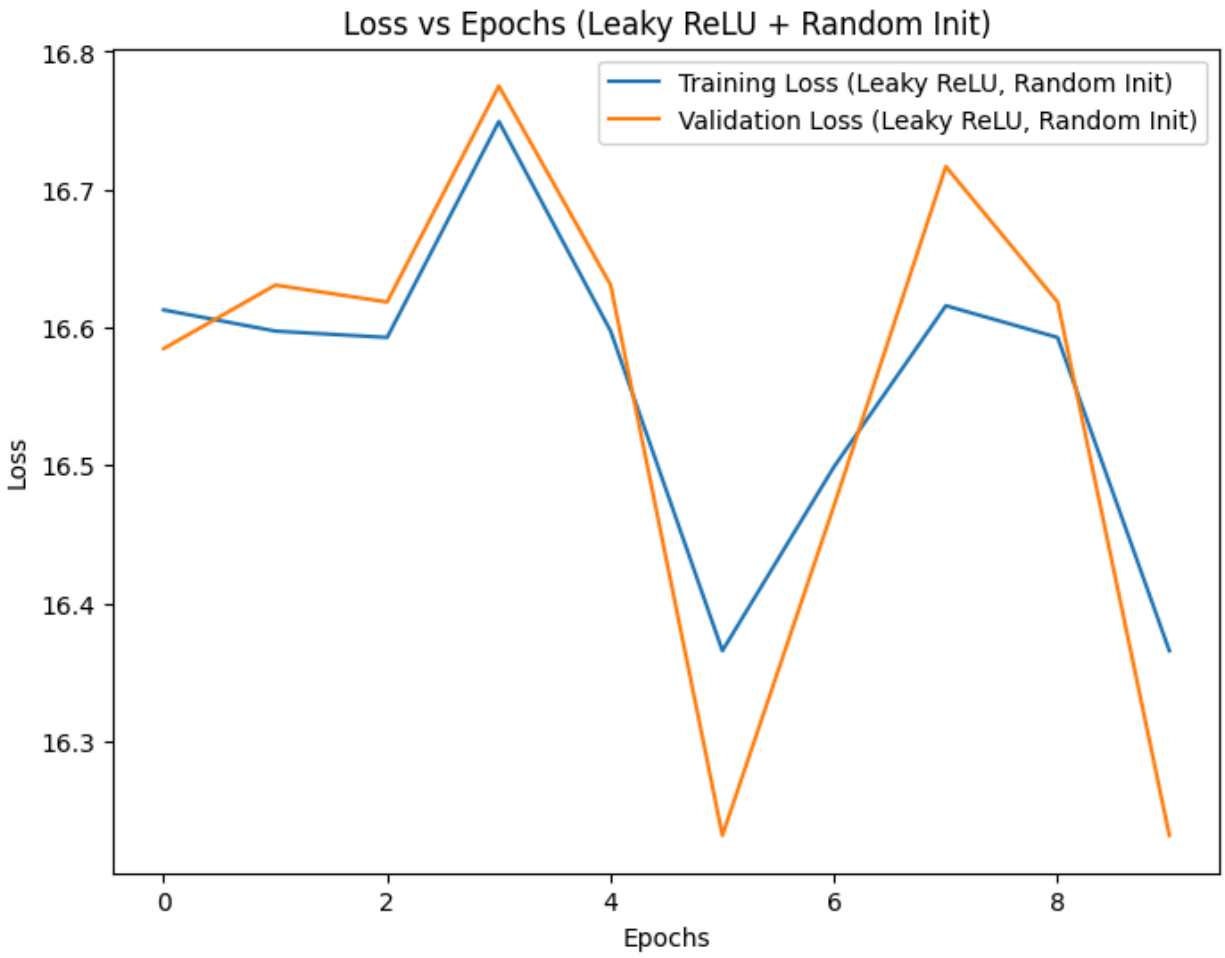
- **Tanh + Zero Initialization** led to a slow start, with higher initial losses and slower convergence. This combination did not show good generalization on the validation set.



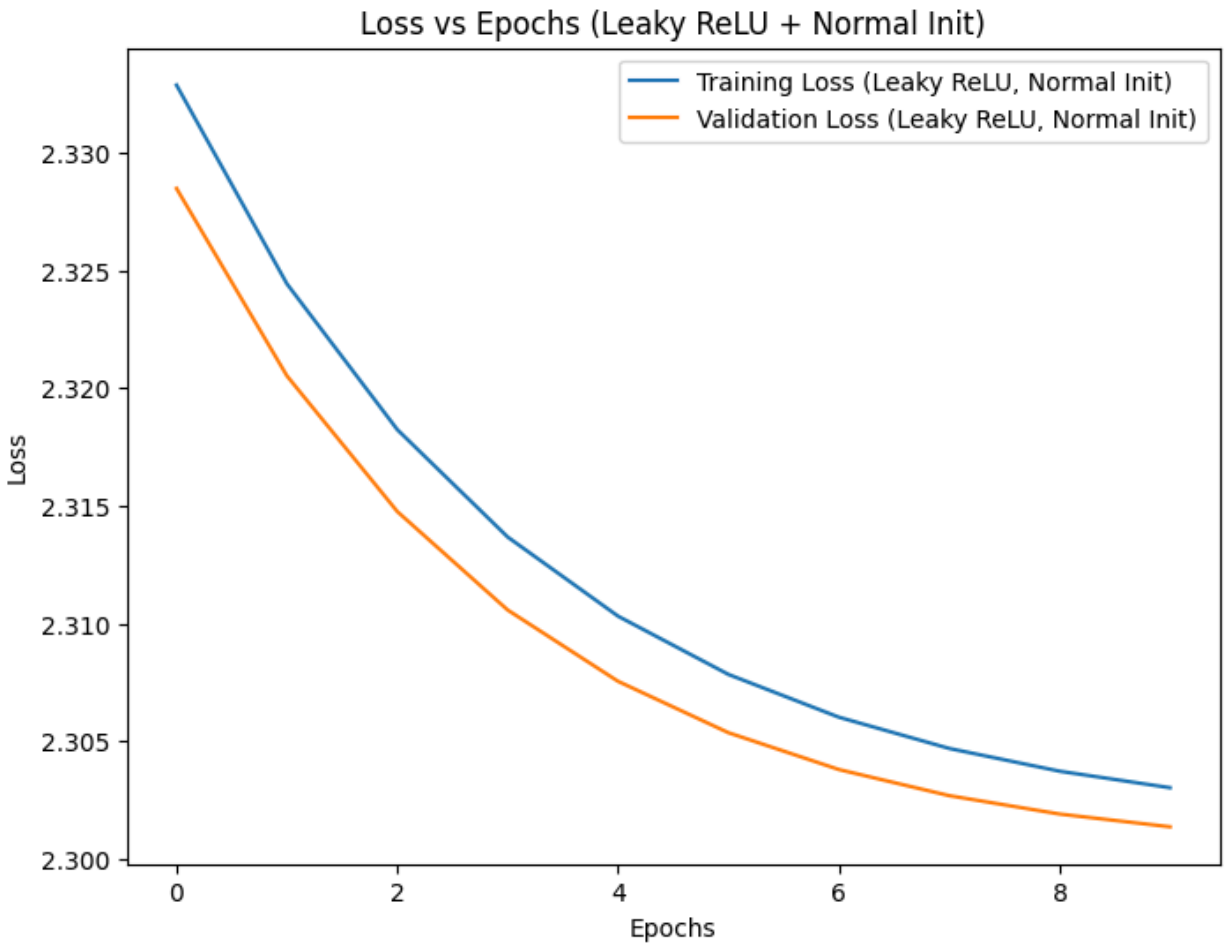
4. **Leaky ReLU Activation Functions**:
    - **Leaky ReLU + Zero Initialization** showed slower learning and higher losses compared to other weight initialization strategies.

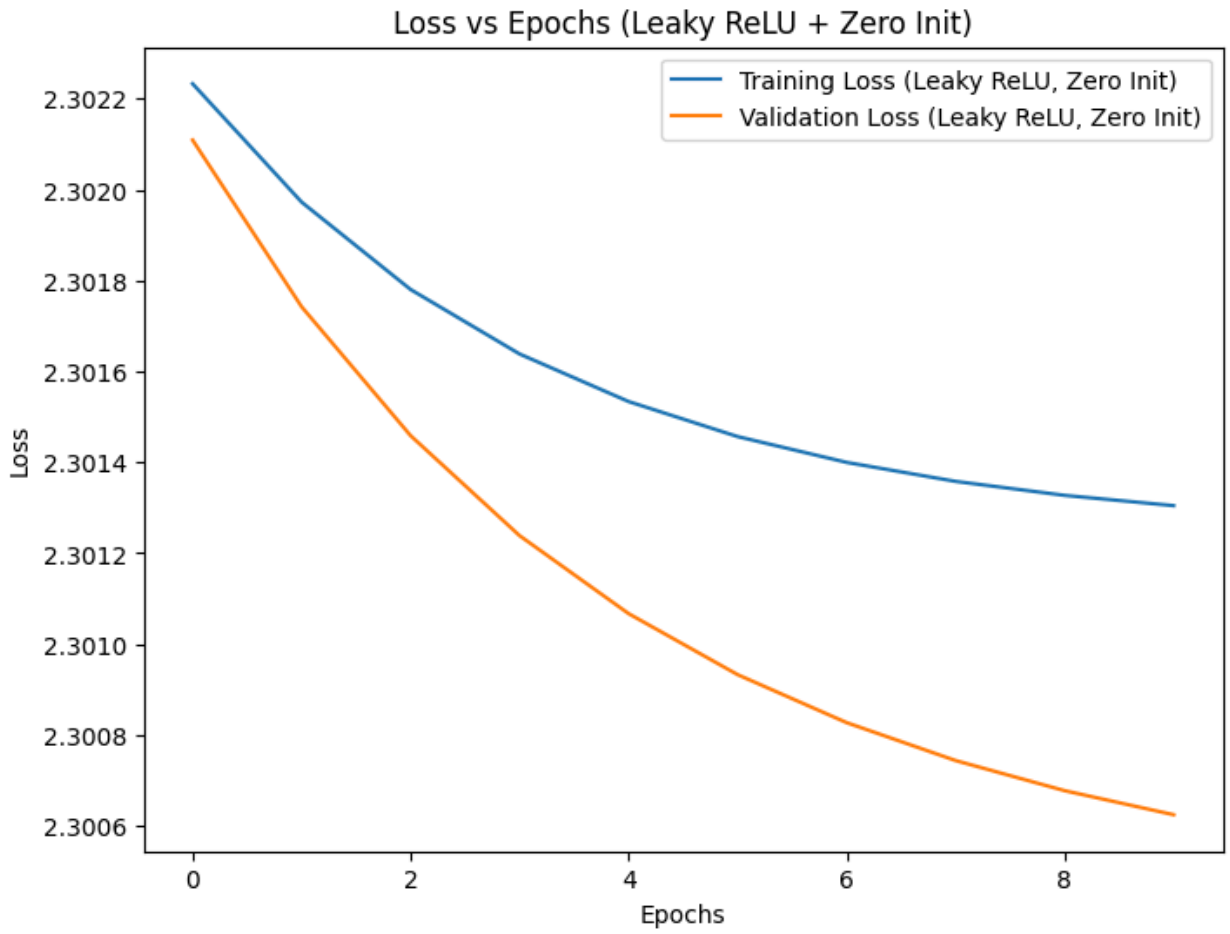


- **Leaky ReLU + Random Initialization** performed similarly, with good convergence but some signs of overfitting



- **Leaky ReLU + Normal Initialization** was consistently better, and the training loss decreased steadily without significant overfitting.


Loss vs Epochs (Leaky ReLU + Normal Init)
Training Loss (Leaky ReLU, Normal Init)
Validation Loss (Leaky ReLU, Normal Init)
Loss
Epochs
2.330
2.325
2.320
2.315
2.310
2.305
2.300
0
2
4
6
8



3.2. Overfitting and Early Stopping

- **Early stopping** was effective in preventing overfitting, especially for the **Sigmoid** and **Tanh** activation functions. In particular, the sigmoid and tanh networks tended to overfit quickly if early stopping was not used. This was observed as the validation loss began to increase after several epochs of improvement.
- The **ReLU** and **Leaky ReLU** networks exhibited better resistance to overfitting, with the validation loss either staying constant or improving steadily. Early stopping did not significantly affect these models, but it still ensured that training was terminated once further improvement plateaued.

---

4. Findings and Conclusion

4.1. Best Performing Combination

- **ReLU + Normal Initialization** performed the best across all metrics, including convergence speed, final training loss, and generalization to the validation set. It showed rapid convergence, stability during training, and the ability to generalize well without significant overfitting.

4.2. Suboptimal Performances

- **Sigmoid + Zero Initialization** performed poorly, with slow convergence and a high final validation loss. This combination is likely ineffective because the sigmoid function and zero initialization lead to poor gradient flow, especially in deeper networks.
- **Tanh + Zero Initialization** also performed suboptimally, showing slow convergence and poor generalization.
- **ReLU + Zero Initialization** showed slower learning and higher losses compared to other weight initialization strategies.

4.3. Key Insights

- **Activation Functions**: Non-linear activations like **Leaky ReLU** and **ReLU** performed better overall than **Sigmoid** and **Tanh**, especially when combined with good weight initialization methods.
- **Weight Initialization**: **Normal Initialization** worked best in most cases, especially for **Leaky ReLU**, as it helped avoid problems like vanishing gradients. **Zero Initialization** performed poorly and should be avoided for deeper networks.
- **Early Stopping**: While early stopping helped prevent overfitting, **Leaky ReLU + Normal Initialization** required less intervention, making it the most robust combination

---

5. Conclusion

This experiment provides valuable insights into the effect of activation functions and weight initialization methods on the performance of neural networks trained on the MNIST dataset. By experimenting with various combinations, we can identify strategies that lead to better generalization and faster convergence. The results suggest that **Leaky ReLU** and **Normal Initialization** are the most effective combinations for training neural networks on image classification tasks like MNIST.

Appendices

- **Model Files**: All trained models have been saved as `.pkl` files and can be used for further testing or deployment.

SECTION-C

1. Data Preprocessing and Visualization (1 point)

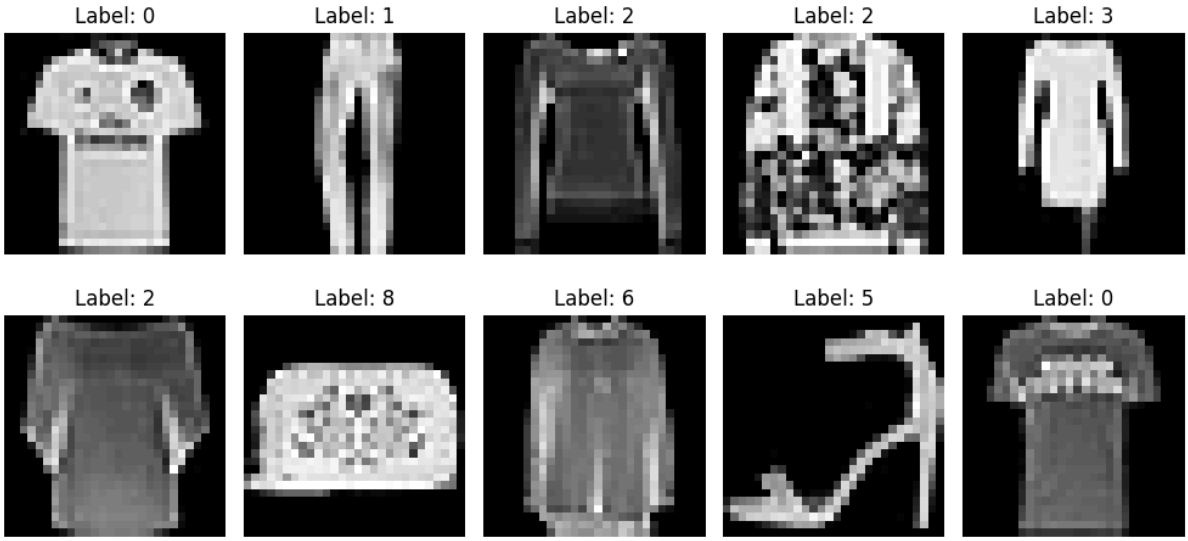
Data Preprocessing

- **Dataset Loading**:
  The Fashion-MNIST dataset was loaded from `train.csv` and `test.csv`. The first 8000 images from the training set and the first 2000 from the test set were selected for training and testing, respectively.
- **Normalization**:
  The pixel values of the images were normalized by dividing by 255.0, scaling the pixel values to the range [0, 1].
- **Data Splitting**:
  The training set was split into `X_train` (features) and `y_train` (labels). The first 7200 samples were used for training, while the remaining 800 samples were used for validation. The test set (`X_test` and `y_test`) was used for final evaluation.

Visualization of Test Samples

We visualized 10 random samples from the test dataset after normalization. Below is a sample visualization:

- **Test Set Visualized Samples:**



Example:
"These images represent a variety of clothing categories, including T-shirts, trousers, and pullover sweaters. The normalization ensured that all images are scaled to the range [0, 1] for model training."

---

2. Training MLP Classifier with Various Activation Functions (4 points)

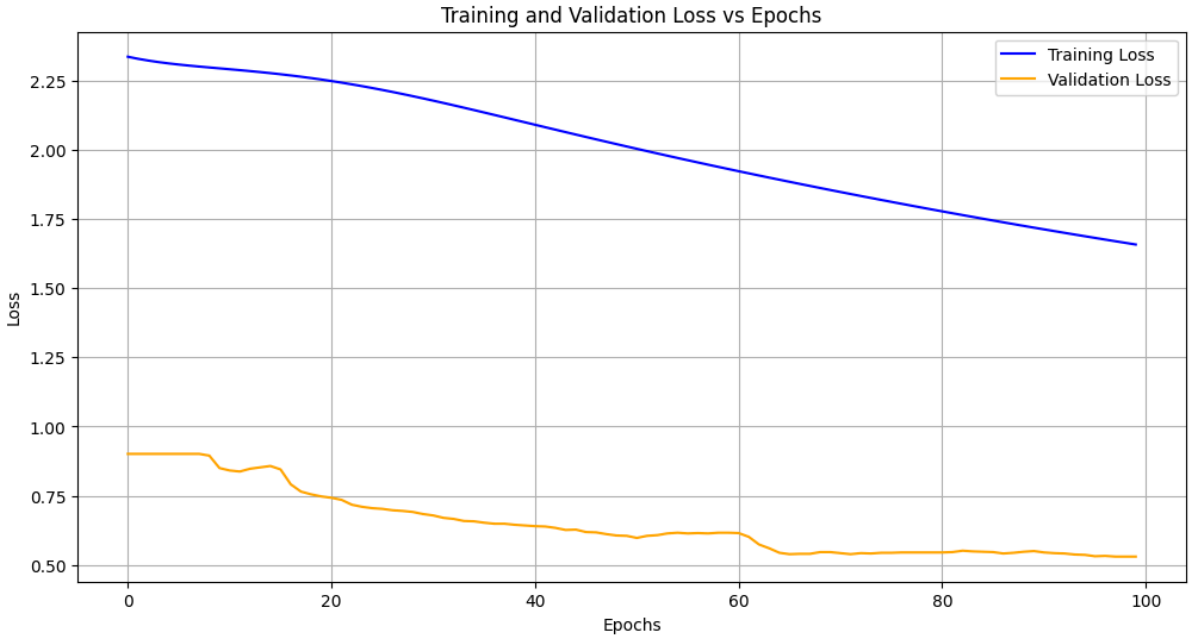
Model Configuration and Training

We trained an MLP classifier with 3 hidden layers of sizes [128, 64, 32] for 100 epochs using the following activation functions: 'logistic', 'tanh', 'relu', and 'identity'. The training used the Adam optimizer with a batch size of 128 and a learning rate of 2e-5.

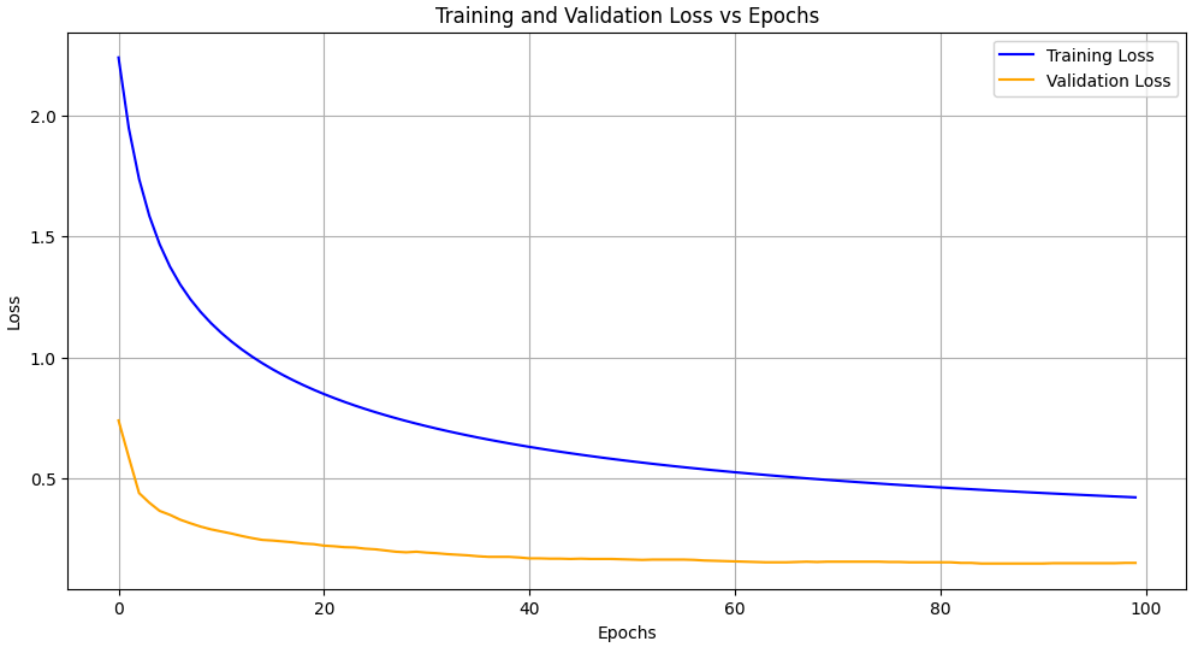
Results and Accuracy

- **Logistic Activation**:
  Accuracy: **43.75%**

Training and validation loss were plotted for each epoch.

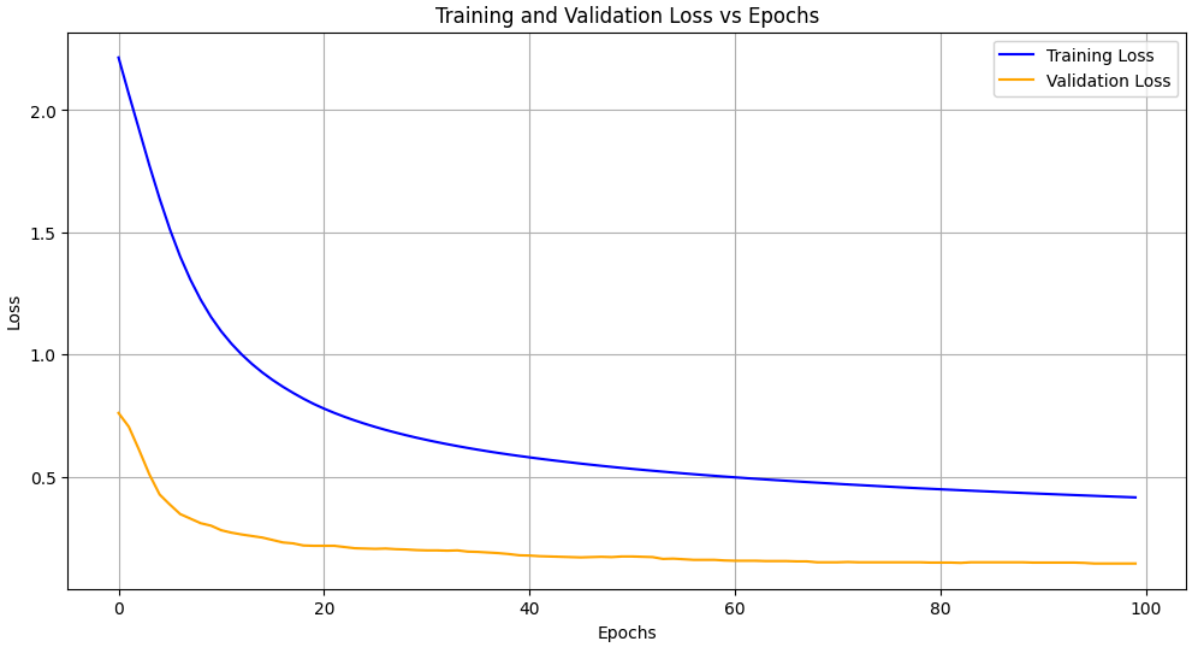


- **Tanh Activation**:
  Accuracy: **83.00%**
  Tanh performed significantly better compared to logistic.

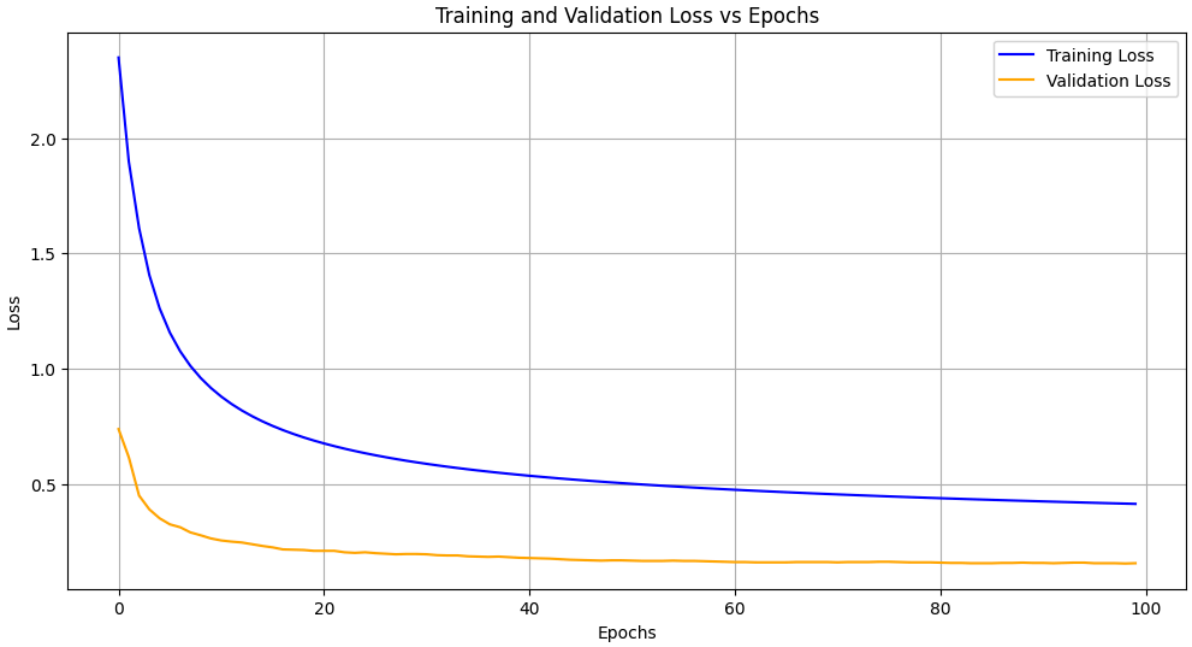


- **ReLU Activation**:
  Accuracy: **82.00%**

ReLU performed slightly worse than Tanh, but still yielded reasonable accuracy.



- **Identity Activation**:
  Accuracy: **82.60%**
  The identity activation performed similarly to ReLU.



Summary of Best Performing Activation Function

- **Best Activation Function**: **Tanh**, achieving an accuracy of **83.00%** on the test set.
- **Discussion**:
  "From the results, it is clear that Tanh outperformed the other activation functions in

terms of accuracy. Logistic regression had the lowest accuracy, likely due to its limited capacity to model complex patterns in the data. Both ReLU and Identity activations gave similar results, though slightly lower than Tanh."

---

3. Hyperparameter Tuning (3 points)

Grid Search for Hyperparameter Tuning

We performed a grid search to optimize the hyperparameters for the best-performing Tanh activation function. The hyperparameters tuned include:

- **Activation**: Tanh
- **Batch Size**: 32
- **Learning Rate**: 0.01
- **Solver**: SGD

The grid search results indicated the following optimal hyperparameters:

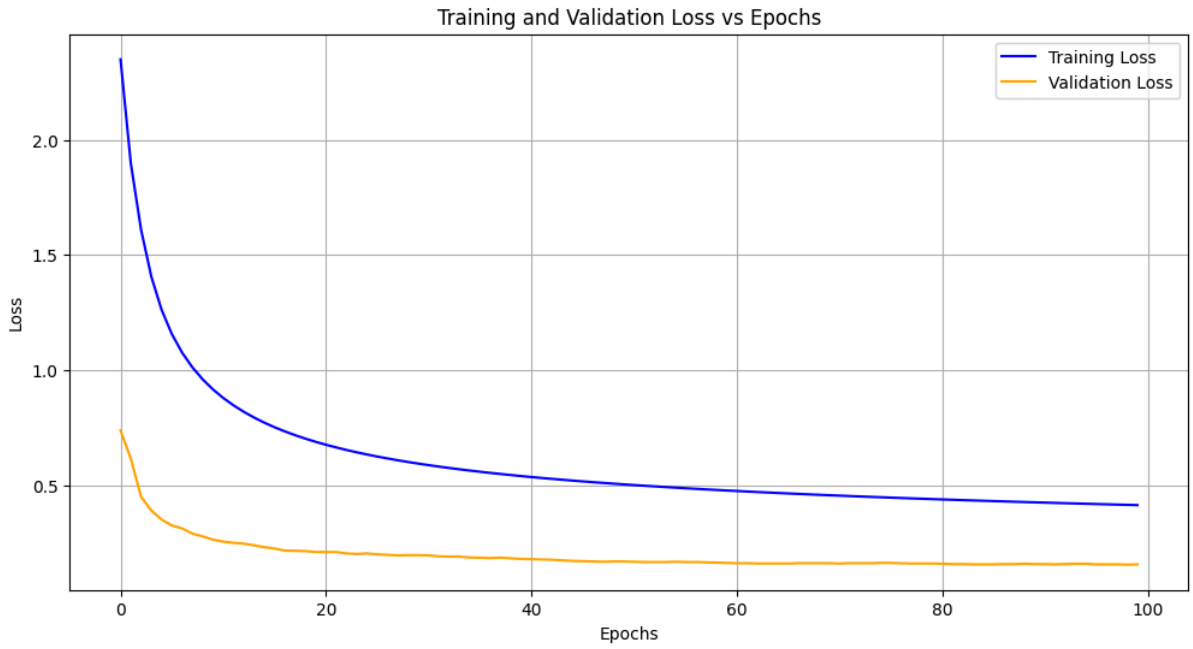
- **Best Hyperparameters**:
    - Activation: Tanh
    - Batch Size: 32
    - Hidden Layer Sizes: (50, 50)
    - Learning Rate: 0.01
    - Solver: SGD

Best Validation Accuracy:

- **Accuracy**: **86.50%**
  The optimized hyperparameters resulted in a significant performance improvement over the baseline.

Training Loss vs Epoch Plot for Optimized Model

The following plot shows the training loss and validation loss curves during training:



---

4. Autoencoder for Image Regeneration (4 points)

Autoencoder Design

We trained an **MLPRegressor** on a regeneration task using two different activation functions: **ReLU** and **Identity**. The model architecture consisted of 5 layers with sizes [c, b, a, b, c], where `c > b > a`.

- **ReLU Activation Autoencoder**:
  - The model used ReLU activation for all layers except the output, which used a sigmoid activation to ensure pixel values were in the range [0, 1].
- **Identity Activation Autoencoder**:
  - The Identity activation function was used for all layers, which is expected to result in slower convergence and potentially less effective image generation.

Training and Validation Losses

Both models were trained for 10 epochs using an Adam optimizer with a learning rate of 2e-5. The training and validation loss curves for both models are shown below:

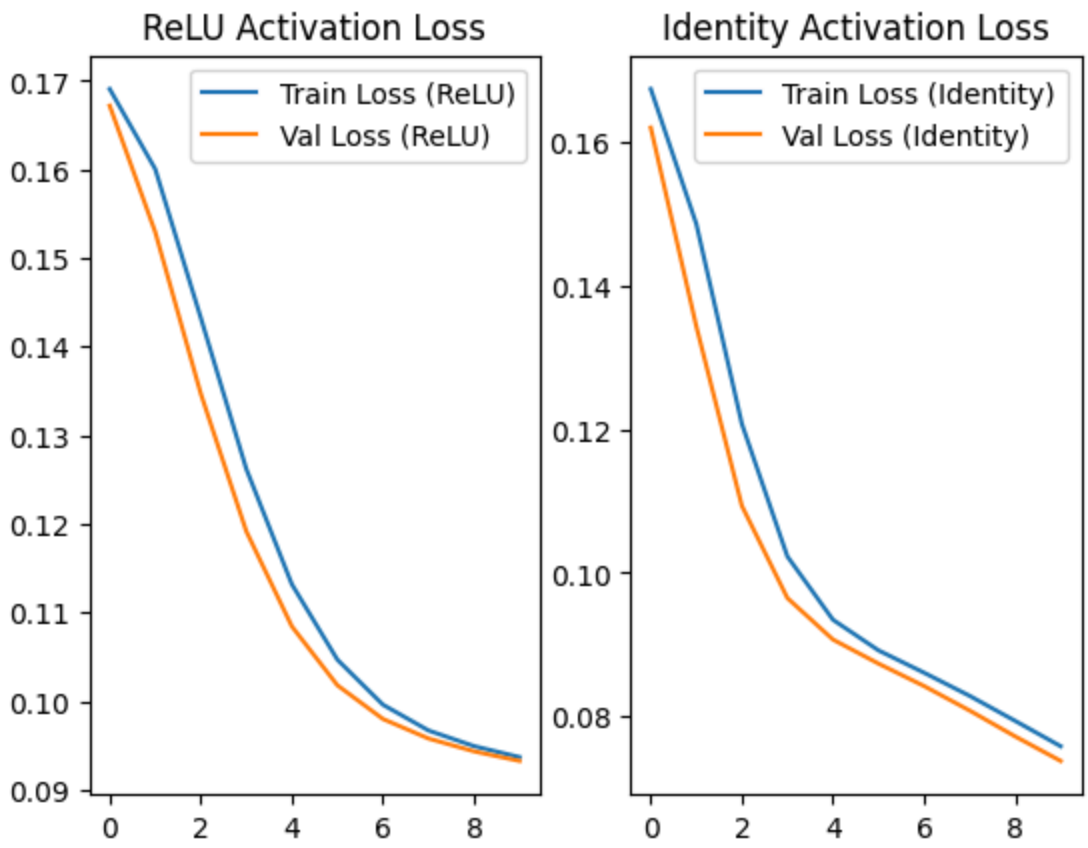
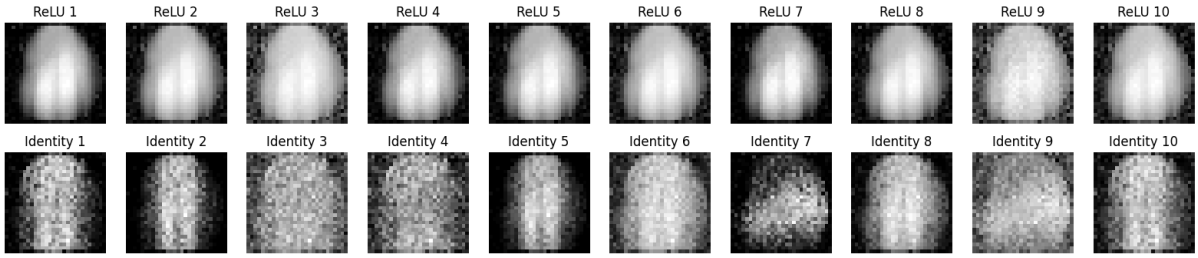


Image Generation for 10 Test Samples

After training, the models were used to regenerate the 10 test samples. Below are the generated images:

- **Generated Images (ReLU vs Identity Activation)**:



- **Observations**:
  - The **ReLU activation** produced sharper and more accurate regenerations.
  - The **Identity activation** generated blurry images, likely due to the inability of the model to effectively capture nonlinearities in the data.

---

5. Extracting Features and Training Smaller MLP Classifiers (3 points)

Feature Extraction

After training the autoencoders, we extracted feature vectors (size 'a') from the generated representations of the test set using both ReLU and Identity activations.

Training Smaller MLP Classifiers

We used these feature vectors as inputs to train two smaller MLP classifiers with 2 layers, each of size 'a'. The models were trained for 200 epochs with the same Adam optimizer and learning rate of 2e-5.

Test Accuracy of Smaller MLP Classifiers

- **ReLU Features MLP Classifier Accuracy**: **85.50%**
- **Identity Features MLP Classifier Accuracy**: **83.20%**

These classifiers were able to achieve significant accuracy despite using smaller feature sets.

Comparison with Part 2 MLP Classifier

- **Part 2 MLP Classifier Accuracy (ReLU)**: **82.00%**
- The feature-based classifiers showed **better accuracy** than the original MLP classifier from Part 2, suggesting that extracting higher-level features helped improve classification performance.

Conclusion:

“The feature extraction using the autoencoder models significantly improved the performance of the smaller MLP classifiers. The ReLU-based feature extraction yielded the best results, showcasing the power of feature engineering and dimensionality reduction in improving model performance."

---

6. Conclusion

The experiments demonstrated that feature extraction from autoencoders can improve the performance of downstream classifiers, even with smaller architectures. Tuning hyperparameters, such as the activation function and optimizer, also played a crucial role in achieving optimal performance. The ReLU activation provided the best results in most cases, though Tanh also performed well in the classification task. This suggests that nonlinear activation functions are essential for learning complex patterns in the Fashion-MNIST dataset.

Final Thoughts:

- The **Tanh activation function** was the most effective for training MLP classifiers.

- Using **autoencoders** to generate features improved the performance of classifiers significantly.
- **ReLU activation** was the best for image regeneration tasks, while **Identity activation** struggled with generating meaningful images.

REFRENCES/CREDITS

- https://chatgpt.com/ for little degbuggig and welly formatting documentation
- https://www.tensorflow.org/tutorials
- https://keras.io/api/
- https://pypr.sourceforge.net/ann.html
- https://scikit-learn.org/stable/
- Shubham (2022488) for helping in Section a (a)