CSE343
Machine Learning
Monsoon 2024

Report for Assignment 3

Shamik Sinha - 2022468

Section A

Part 1.

Given an MLP with one hidden layer (ReLU activation) and a linear output, we will perform a single training iteration using the Mean Squared Error (MSE) loss function. The network has the following parameters:

$$W_1 = 1, \quad W_2 = 2, \quad b_1 = 3, \quad b_2 = 6$$

$$\text{Inputs: } x = [1, 2, 3], \quad \text{Targets: } y = [3, 4, 5]$$

$$\text{Learning rate: } \eta = 0.01$$

Step 1: Forward Pass

1. Compute the hidden layer input z using the weights and bias for W_1 and b_1:

$$z = W_1 \cdot x + b_1$$

For $x = [1, 2, 3]$, we compute:

$$z = 1 \cdot x + 3 = [1 + 3, 2 + 3, 3 + 3] = [4, 5, 6]$$

2. Apply the ReLU activation function:

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

For $z = [4, 5, 6]$, we get:

$$h = [4, 5, 6]$$

3. Compute the output layer:

$$\hat{y} = W_2 \cdot h + b_2$$

For $W_2 = 2$ and $b_2 = 6$, we compute:

$$\hat{y} = 2 \cdot [4, 5, 6] + 6 = [8 + 6, 10 + 6, 12 + 6] = [14, 16, 18]$$

Step 2: Loss Calculation

Calculate the Mean Squared Error (MSE) loss:

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

For $y = [3, 4, 5]$ and $\hat{y} = [14, 16, 18]$, we compute:

$$\text{MSE} = \frac{1}{3} \left[(14 - 3)^2 + (16 - 4)^2 + (18 - 5)^2 \right]$$

$$\text{MSE} = \frac{1}{3} \left[121 + 144 + 169 \right] = \frac{1}{3} \cdot 434 = 144.67$$

Step 3: Backward Pass

1. Compute the gradient of the loss with respect to the output $\hat{y}$:

$$\frac{\partial \text{MSE}}{\partial \hat{y}} = \frac{2}{3}(\hat{y} - y) = \frac{2}{3} \cdot [14 - 3, 16 - 4, 18 - 5] = \frac{2}{3} \cdot [11, 12, 13]$$

$$\frac{\partial \text{MSE}}{\partial \hat{y}} = \left[\frac{22}{3}, \frac{24}{3}, \frac{26}{3}\right] = [7.33, 8, 8.67]$$

2. Compute the gradients for W_2 and b_2:

$$\frac{\partial \text{MSE}}{\partial W_2} = \frac{\partial \text{MSE}}{\partial \hat{y}} \cdot h = [7.33, 8, 8.67] \cdot [4, 5, 6] = 7.33 \cdot 4 + 8 \cdot 5 + 8.67 \cdot 6$$

$$\frac{\partial \text{MSE}}{\partial W_2} = 29.32 + 40 + 52.02 = 121.34$$

$$\frac{\partial \text{MSE}}{\partial b_2} = \sum_{i=1}^{3} \frac{\partial \text{MSE}}{\partial \hat{y}} = 7.33 + 8 + 8.67 = 24$$

3. Compute the gradient for h (the hidden layer output):

$$\frac{\partial \text{MSE}}{\partial h} = \frac{\partial \text{MSE}}{\partial \hat{y}} \cdot W_2 = [7.33, 8, 8.67] \cdot 2 = [14.67, 16, 17.34]$$

4. Compute the gradient with respect to z using the derivative of the ReLU activation:

$$\frac{\partial \text{MSE}}{\partial z} = \frac{\partial \text{MSE}}{\partial h} \cdot \text{ReLU}'(z)$$

Since $\text{ReLU}'(z) = 1$ for all values of $z = [4, 5, 6]$ (as they are all positive), we have:

$$\frac{\partial \text{MSE}}{\partial z} = [14.67, 16, 17.34]$$

5. Compute the gradients for W_1 and b_1:

$$\frac{\partial \text{MSE}}{\partial W_1} = \frac{\partial \text{MSE}}{\partial z} \cdot x = [14.67, 16, 17.34] \cdot [1, 2, 3] = 14.67 \cdot 1 + 16 \cdot 2 + 17.34 \cdot 3$$

$$\frac{\partial \text{MSE}}{\partial W_1} = 14.67 + 32 + 52.02 = 98.69$$

$$\frac{\partial \text{MSE}}{\partial b_1} = \sum_{i=1}^{3} \frac{\partial \text{MSE}}{\partial z} = 14.67 + 16 + 17.34 = 48.01$$

Step 4: Update Weights

Now we update the weights and biases using the gradients and learning rate $\eta = 0.01$:

$$W_2 = W_2 - \eta \cdot \frac{\partial \text{MSE}}{\partial W_2} = 2 - 0.01 \cdot 121.34 = 2 - 1.2134 = 0.7866$$

$$b_2 = b_2 - \eta \cdot \frac{\partial \text{MSE}}{\partial b_2} = 6 - 0.01 \cdot 24 = 6 - 0.24 = 5.76$$

$$W_1 = W_1 - \eta \cdot \frac{\partial \text{MSE}}{\partial W_1} = 1 - 0.01 \cdot 98.69 = 1 - 0.9869 = 0.0131$$

$$b_1 = b_1 - \eta \cdot \frac{\partial \text{MSE}}{\partial b_1} = 3 - 0.01 \cdot 48.01 = 3 - 0.4801 = 2.5199$$

Thus, after one training iteration, the updated parameters are:

$$W_2 = 0.7866, \quad b_2 = 5.76, \quad W_1 = 0.0131, \quad b_1 = 2.5199$$

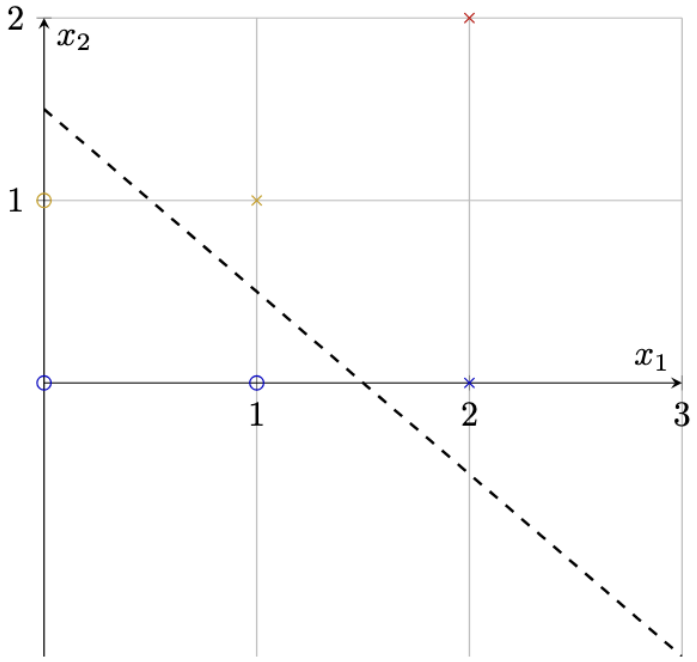
2.

From the dataset, we observe the following points:

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

We will plot the points on a 2D plane and determine the weight vector corresponding to the maximum margin hyperplane, as well as identify the support vectors.

Scatter plot of dataset points with Decision Boundary



Step 1: Linearly Separable
We can see from the plot that the points are linearly separable.
Step 2: Defining Support Vectors and Classifying Points
Assume that the linear decision boundary is defined by the equation:

$$w_0 x_1 + w_1 x_2 + b = 0$$

For the $+1$ class, we use the following points:

$$w_0 x_1 + w_1 x_2 + b = 1$$

Substituting points A and B:

$$\begin{aligned} w_2 + b &= 1 \quad \text{(Equation 1)} \\ w_0 + w_2 + b &= 1 \quad \text{(Equation 2)} \\ w_0 + w_2 + b &= 1 \quad \text{(Equation 3)} \end{aligned}$$

Step 3: Solving the system
Solving these equations for $w_0 = w_2 = -2$ and $b = 3$:

$$\text{Weight vector:} \quad w = \begin{bmatrix} -2 \\ -2 \end{bmatrix}, \quad b = 3$$

Step 4: Support Vectors
The points $(0, 1)$, $(1, 0)$, $(1, 1)$, and $(2, 0)$ are the support vectors.

The decision boundary equation is given by:

$$x + y = 1.5$$

which can be rewritten as:

$$x + y - 1.5 = 0$$

Thus, we have the weight vector and bias as:

$$\text{Weight vector} = \begin{bmatrix} 1 \\ 1 \end{bmatrix}, \quad \text{Bias} = -1.5$$

The support vectors are identified as:

$$(0, 1), (1, 0), (1, 1), (2, 0)$$

Calculation for Distance Between Support Vector Hyperplanes

The equations for the support vector hyperplanes are: 1. $x + y = 1$ 2. $x + y = 2$

The distance between these two hyperplanes is calculated as:

$$\text{Distance} = \frac{|2 - 1|}{\sqrt{1^2 + 1^2}} = \frac{1}{\sqrt{2}}$$

Calculation for Decision Boundary

For the decision boundary, the calculation is as follows:

$$\frac{|1 - c|}{\sqrt{2}} = \frac{1}{2} \times \frac{1}{\sqrt{2}}$$

Solving for c:

$$c = \frac{3}{2}$$

Part 3.

Given that w_1 = -2, $w_2 = 0$, $b = 5$ and the classes as y=+1 and y=-1:

$$||w|| = \sqrt{w_1^2 + w_2^2} = \sqrt{(-2)^2 + (0)^2} = \sqrt{4} = 2$$

$$\text{Margin of SVM} = \frac{2}{||w||} = \frac{2}{2} = 1$$

For a point to be a support vector, $w \cdot x + b = \pm 1$

$$\begin{aligned}
&(1,2)\text{: } w \cdot x + b = (-2)(1) + (0)(2) + 5 = -2 + 5 = 3 \\
&(2,3)\text{: } w \cdot x + b = (-2)(2) + (0)(3) + 5 = -4 + 5 = 1 \\
&(3,3)\text{: } w \cdot x + b = (-2)(3) + (0)(3) + 5 = -6 + 5 = -1 \\
&(4,1)\text{: } w \cdot x + b = (-2)(4) + (0)(1) + 5 = -8 + 5 = -3
\end{aligned}$$

Therefore, samples 2 and 3, i.e., (2,3) and (3,3) are the support vectors.

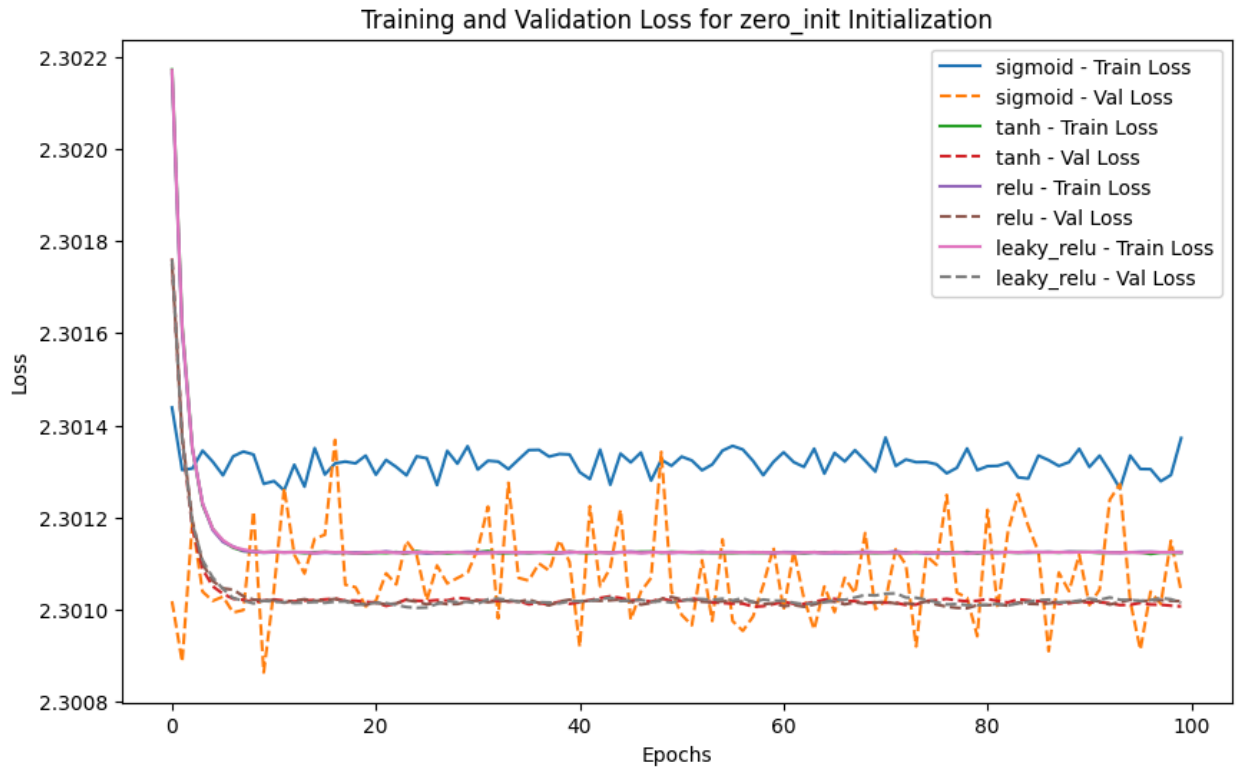
For predicting the class of (1,3):

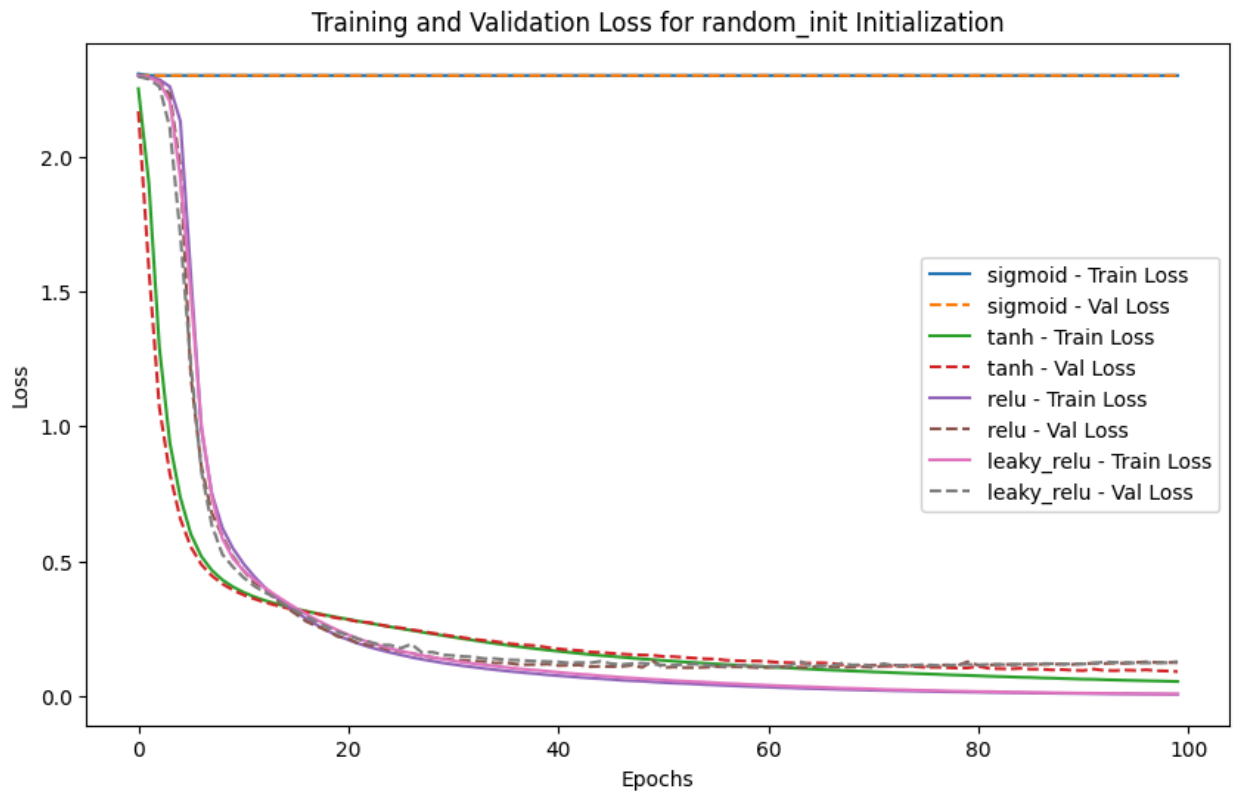
$$w \cdot x + b = (-2)(1) + (0)(3) + 5 = -2 + 5 = 3$$

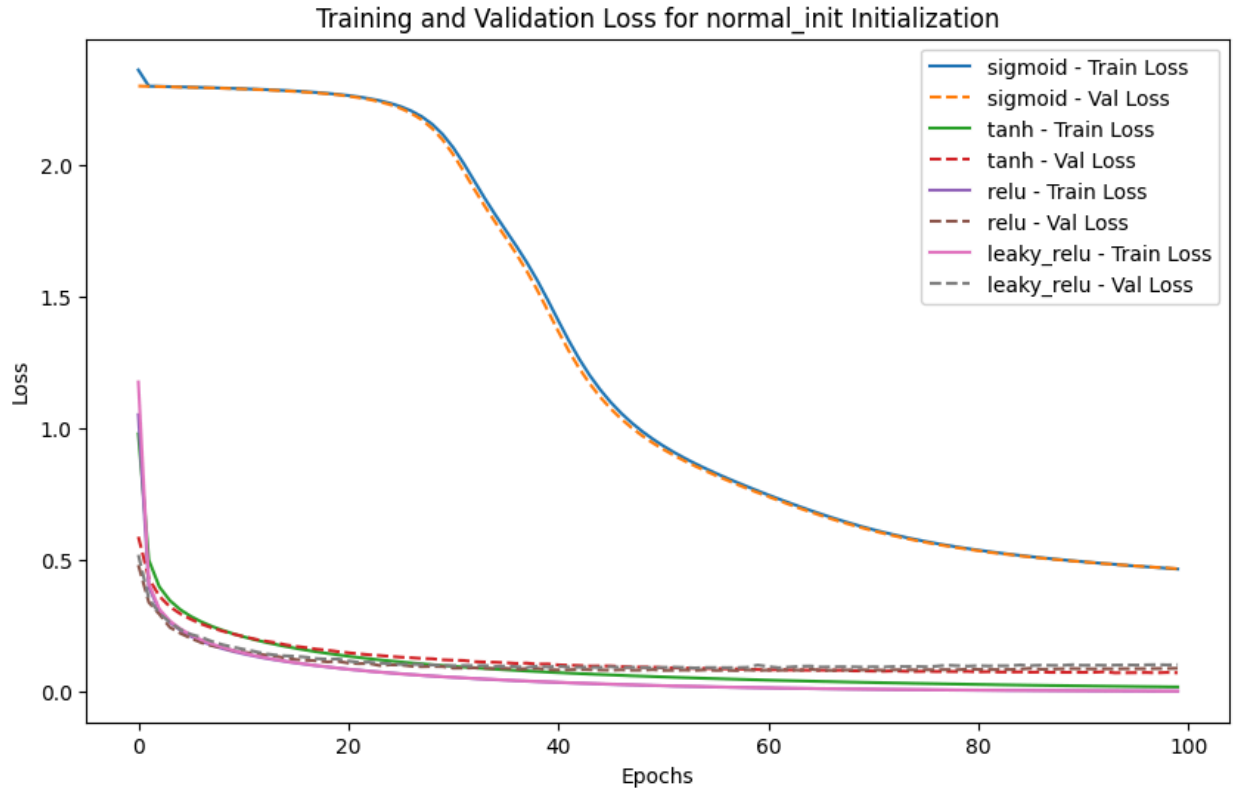
Since $w \cdot x + b = 3 > 0$, we predict $y = +1$ for (1,3).

Section B

Test Accuracy for sigmoid with zero_init: 10.67%
Test Accuracy for tanh with zero_init: 10.67%
Test Accuracy for relu with zero_init: 10.67%
Test Accuracy for leaky_relu with zero_init: 10.67%
Test Accuracy for sigmoid with random_init: 10.67%
Test Accuracy for tanh with random_init: 96.70%
Test Accuracy for relu with random_init: 97.00%
Test Accuracy for leaky_relu with random_init: 96.73%
Test Accuracy for sigmoid with normal_init: 87.40%
Test Accuracy for tanh with normal_init: 97.28%
Test Accuracy for relu with normal_init: 97.53%
Test Accuracy for leaky_relu with normal_init: 97.27%







Observations:

Zero Initialization:
Across all activation functions (sigmoid, tanh, ReLU, leaky ReLU), test accuracy remains low (~10.67%), indicating little learning due to symmetry issues. Loss plots show negligible improvement, confirming that zero initialization is unsuitable.

Random Initialization:
With random initialization, the model performs significantly better, achieving ~96.7% accuracy with tanh, ReLU, and leaky ReLU. However, sigmoid remains ineffective (10.67%), likely due to vanishing gradient issues. Loss curves for tanh, ReLU, and leaky ReLU display smooth declines, suggesting effective learning.
Normal Initialization:

Normal initialization yields the highest test accuracies, especially with ReLU (97.53%) and tanh or leaky ReLU (96.7%+). Sigmoid shows some improvement (87.4%) but lags behind due to gradient issues. Loss plots for tanh, ReLU, and leaky ReLU show rapid convergence.
Best Combinations:

The highest accuracies are achieved with ReLU (97.53%) or leaky ReLU and tanh (96.7%+) under normal initialization.
Suboptimal Combinations:

Zero initialization fails across all activations, while sigmoid performs poorly with all initializations.

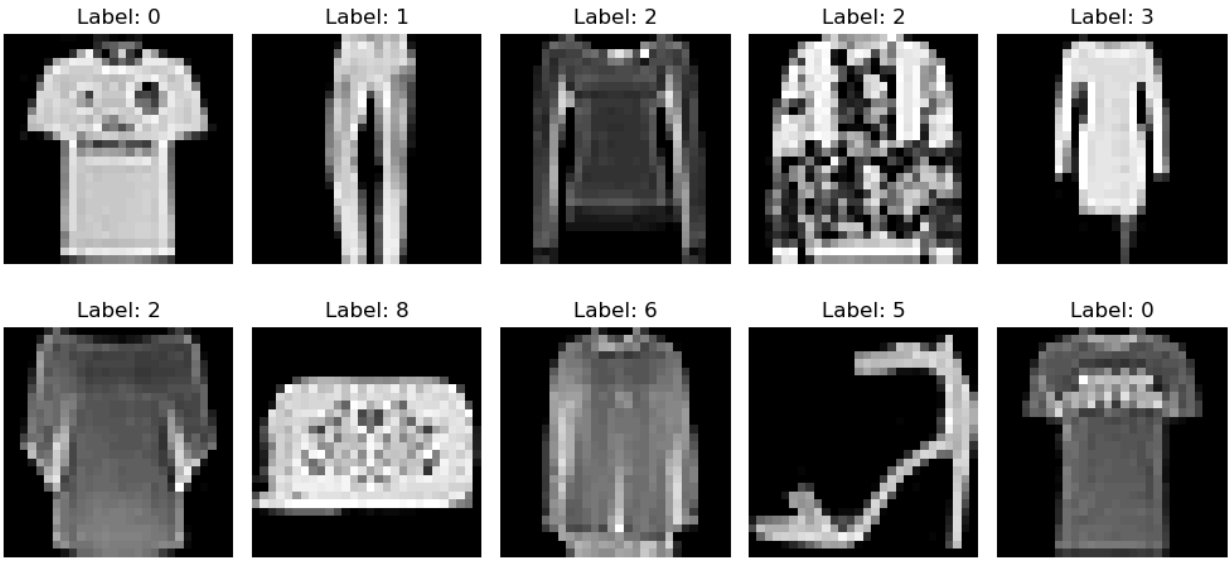
The low accuracy in the zero-initialization case is primarily due to symmetry-breaking issues. When all weights are initialized to zero, every neuron in each layer produces the same output, resulting in identical gradients during backpropagation. This prevents the network from learning any meaningful features, as the updates for each weight are the same, effectively negating any benefit from the network's depth or structure. As a result, the model fails to learn and stays at a baseline accuracy level, which in the case of MNIST (10 classes) is around 10%.

The high accuracy with ReLU (and similarly good results with leaky ReLU) under normal initialization is because ReLU activation functions mitigate the vanishing gradient problem that affects deep networks using activation functions like sigmoid. With sigmoid activations, gradients tend to diminish across layers, especially for deeper networks, slowing down learning and leading to suboptimal performance. However, ReLU's gradient is constant (1 for positive values), preventing gradient shrinkage and enabling efficient learning. This stability in gradient flow helps models with ReLU activation achieve higher accuracy, as observed in your results, where ReLU with normal initialization yields 97.53% accuracy.

Section C

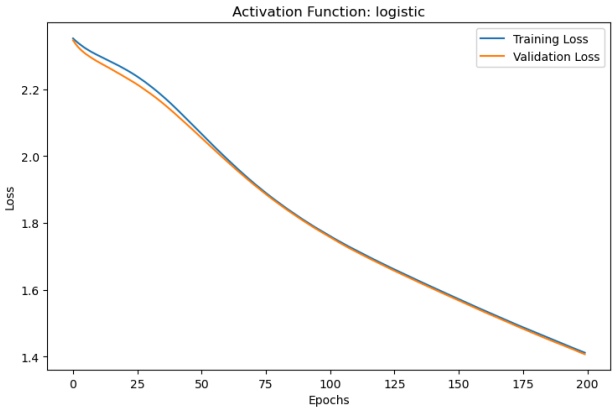
Part 1.

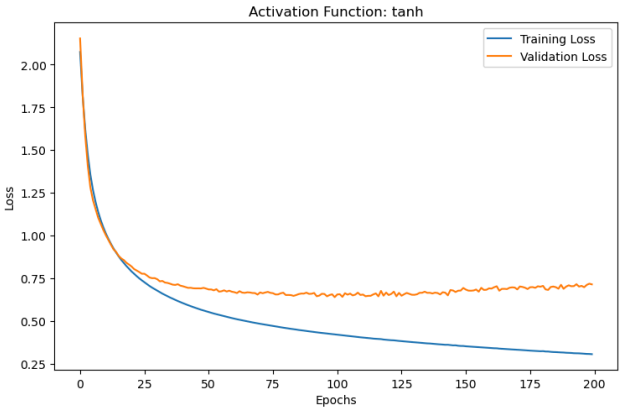
The code preprocesses the MNIST dataset by loading the training and test data, selecting subsets for training and validation, and normalising the pixel values to a 0-1 range. It splits the training data into training and validation sets using train_test_split from scikit-learn. Below are 10 sample images we printed thereafter.

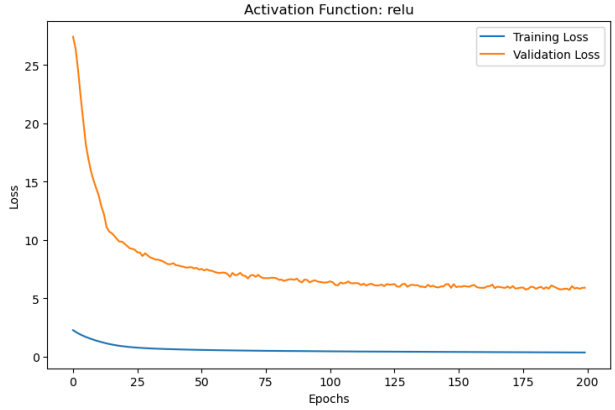


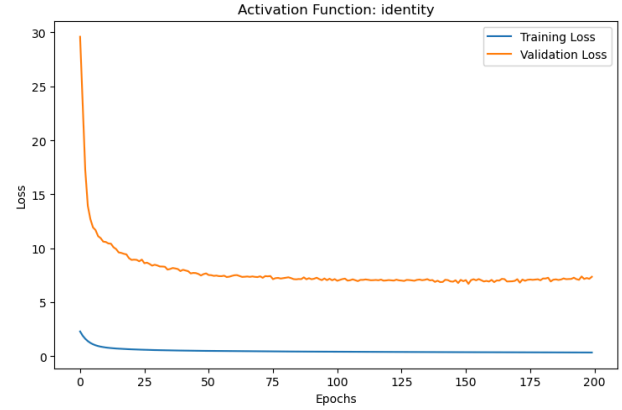
Part 2.

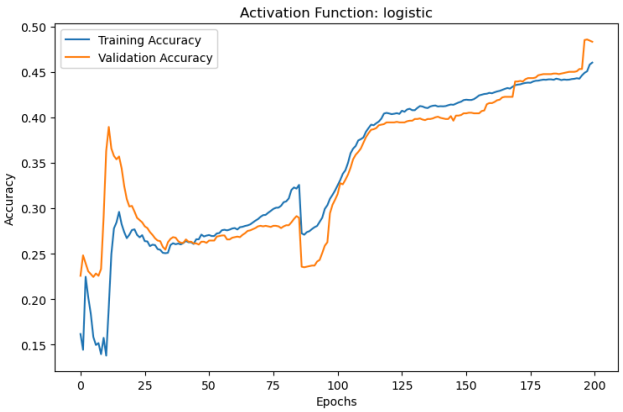
We visualise the training and validation loss and accuracy over epochs for different activation functions. It generates plots for each activation function (logistic, tanh, relu, identity), showing how the loss and accuracy change during training. Additionally, it prints the final validation accuracy for each activation function, with relu achieving the highest accuracy at 0.8455.

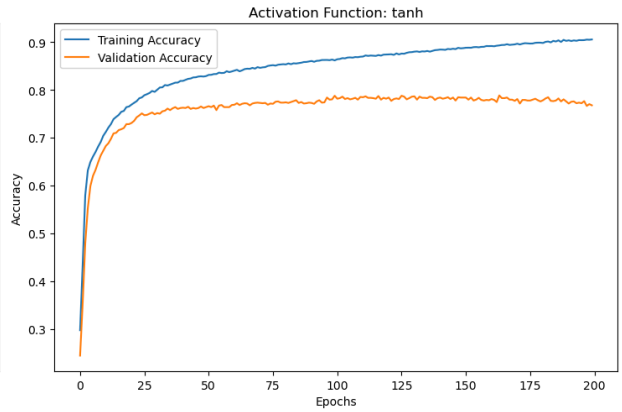

Activation Function: logistic
Training Loss
Validation Loss
Loss
Epochs

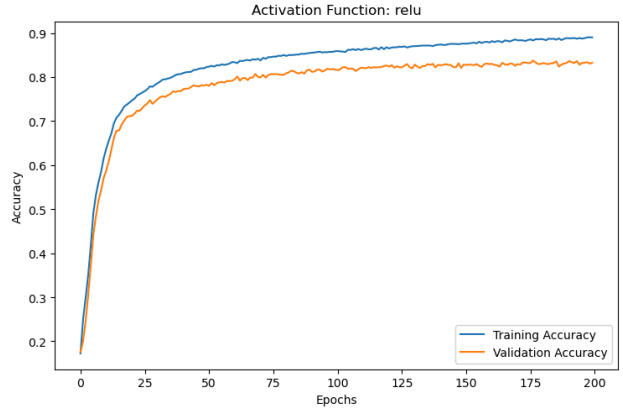

Activation Function: tanh
Training Loss
Validation Loss
Loss
Epochs

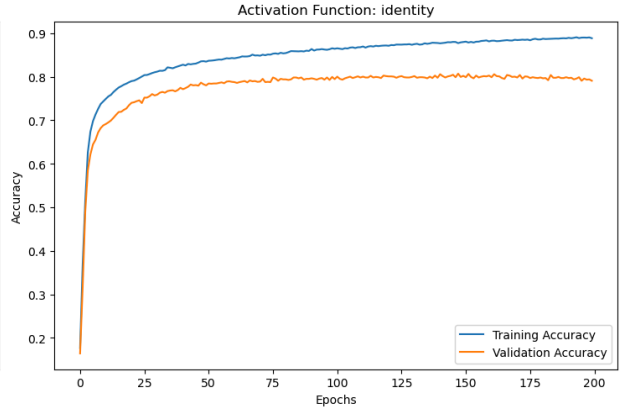

Activation Function: relu
Training Loss
Validation Loss
Loss
Epochs


Activation Function: identity
Training Loss
Validation Loss
Loss
Epochs


Activation Function: logistic
Training Accuracy
Validation Accuracy
Accuracy
Epochs


Activation Function: tanh
Training Accuracy
Validation Accuracy
Accuracy
Epochs


Activation Function: relu
Training Accuracy
Validation Accuracy
Accuracy
Epochs


Activation Function: identity
Training Accuracy
Validation Accuracy
Accuracy
Epochs

The logistic and tanh activation functions show a similar trend, with training and validation loss decreasing steadily. ReLU performs better, converging faster. Identity function struggles, indicating its limitations for non-linear problems.

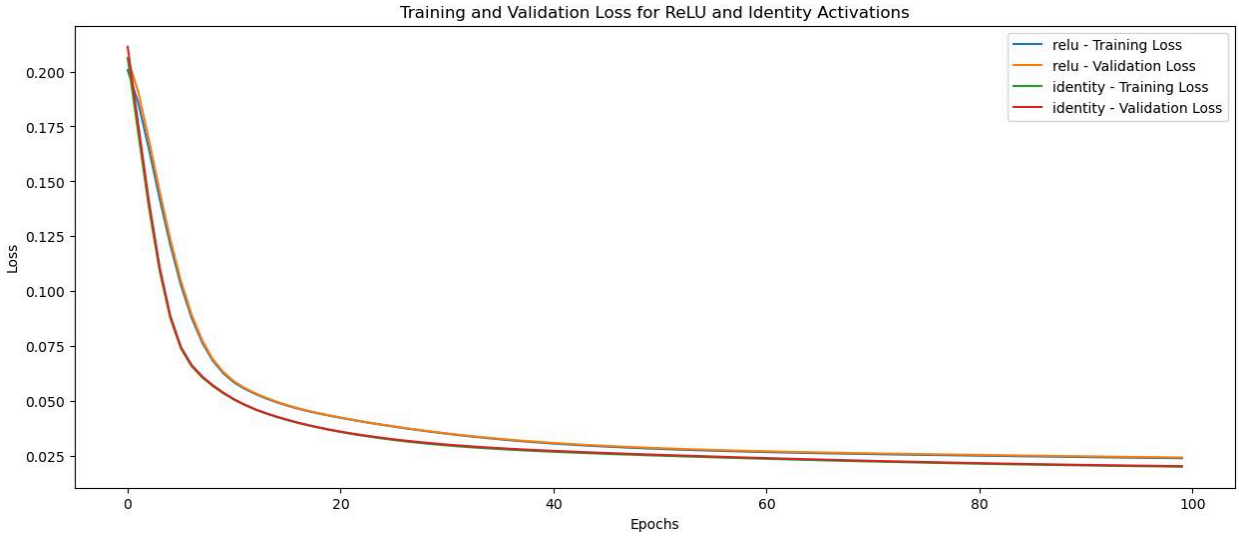
The ReLU activation function appears to be the most effective, with training and validation accuracy converging to high values. The identity function struggles, indicating its limitations for non-linear problems. The logistic and tanh functions show similar performance, with some fluctuations in accuracy.

Part 3.

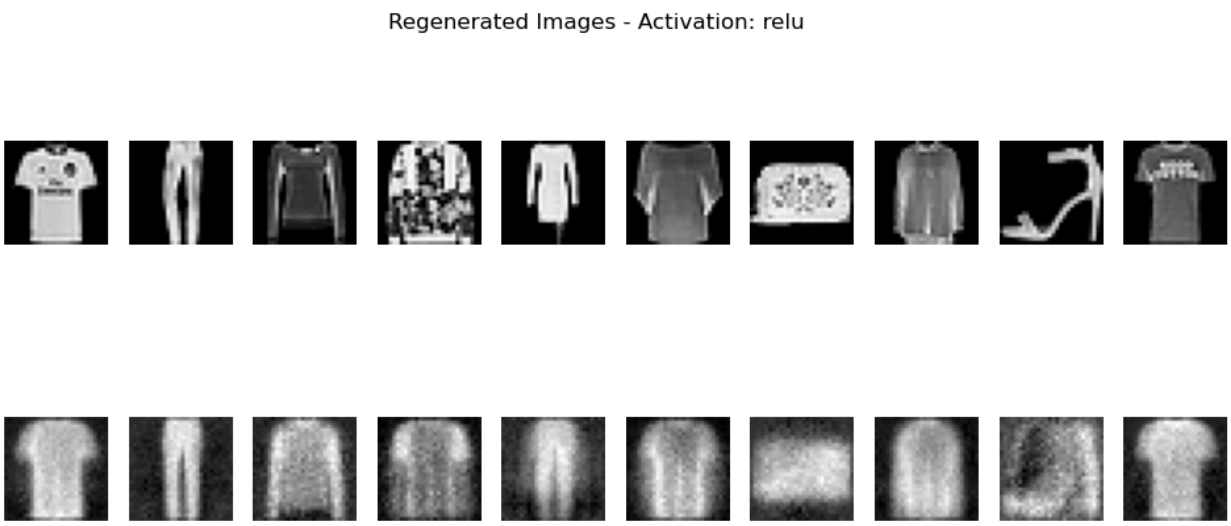
Grid search optimization was performed for the relu activation function since it gave the best results. The best hyperparameters are batch_size=128, learning_rate_init=0.0001, and solver='adam', achieving the best cross-validation accuracy of 0.8578. This suggests improved performance with these settings for the model.

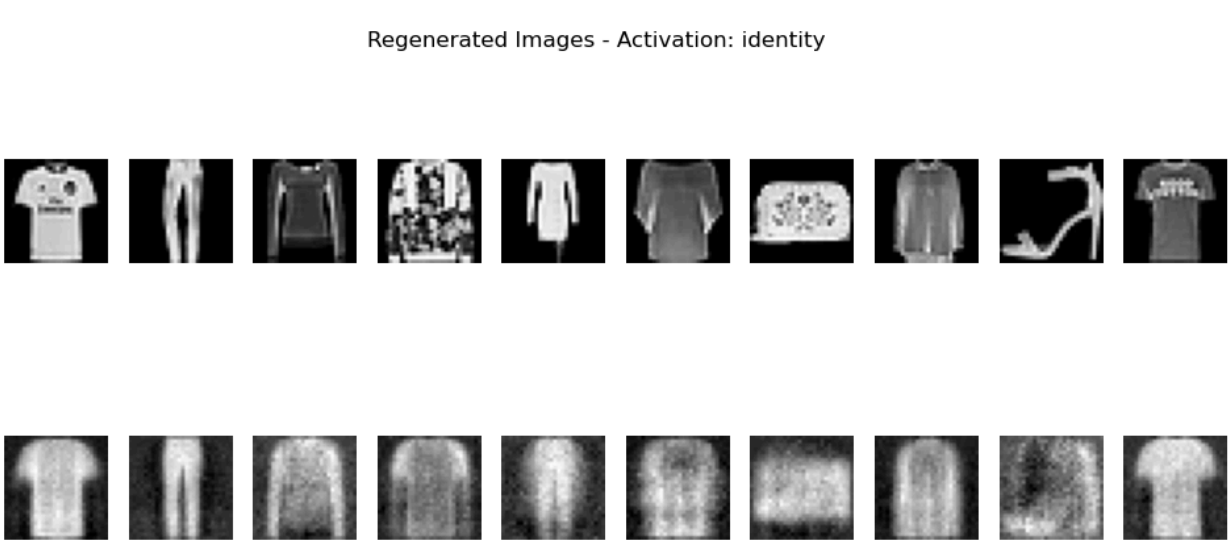
Part 4.

For validation set = test set



The ReLU activation function outperforms the identity function in terms of both training and validation loss. The ReLU's loss converges faster and reaches lower values, indicating better model performance. The identity function struggles to capture complex patterns in the data, leading to higher loss.





Observations Summary:

Image Reconstruction Quality: The ReLU network yields blurry reconstructions, losing detail, likely due to its non-linearity, while the Identity network retains more structural detail, closer to the originals.

Loss Curves and Training Dynamics: Both networks show smooth loss reductions, with ReLU initially optimizing faster, though at the cost of image detail accuracy. Identity activation better preserves fine details.

Conclusion: Identity activation outperforms ReLU for visual fidelity in image reconstruction, highlighting activation choice's importance in preserving detail in MLP-based tasks.

- Network Design: The 5-layer MLP with layer sizes [128, 64, 32, 64, 128] effectively compresses and then reconstructs the input, which provides a structured bottleneck for image regeneration.
- Training and Validation Loss: The training and validation loss curves decrease smoothly over epochs for both ReLU and Identity networks, indicating successful convergence and effective model training for the regeneration task.
- Activation Comparison: ReLU-activated networks initially reduce loss faster, yet the Identity-activated network yields better image fidelity, as it preserves structural details more effectively in the reconstructed outputs.
- Image Quality: Identity activation provides regenerated images that are visually closer to the original test samples, highlighting its suitability for tasks requiring high visual accuracy over ReLU, which produces blurrier outputs.

Part 4.

Observation on Feature Vector Extraction: The middle layer feature vector (size `a=32`) captures essential image representations, allowing a smaller, simpler classifier to be trained effectively on this reduced data, achieving reasonable accuracy.

Classifier Performance: The new 2-layer MLP classifiers, trained on the extracted features, achieve test accuracies of 70.40% (ReLU) and 71.35% (Identity), which are comparable to the previous classifier trained on the full image dataset. This shows that the feature vector retains most of the information needed for classification.

Comparison to Part 2: Although the classifiers here use fewer parameters and simpler networks, they perform similarly to the full MLP classifier from part 2. This indicates that the feature extraction at the middle layer captures enough discriminative information for classification.

Possible Explanation: By compressing the images to feature vectors of size `a`, the network likely retains essential patterns or abstract features crucial for classification. This helps reduce computational complexity while maintaining good accuracy, as the distilled features still represent the images well.