Part A

Section-A

Q1 inputs: $[1,2,3] = X$, outputs/labels $= [3,4,5] = Y$

Relu(x) = maximum $(0, x)$ ⇝ Relu's Derivative is 0 in case of $x = 0$.

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

initial Choosen weights:

$w = [0.2, 0.3]$ (w_1, w_2)

$b = [0, 0]$ (b_1, b_2)

η (learning rate ~~= 0.01~~) $= 0.01$

Forward pass:

for $x_1 = 1$,

$\hat{y}_1 = (0.3)[\text{Relu}(0.2 \times 1 + 0)] + 0 = 0.06$

for $x_2 = 2$

$\hat{y}_2 = (0.3)[\text{Relu}(0.2 \times 2 + 0)] + 0 = 0.12$

for $x_3 = 3$,

$\hat{y}_3 = (0.3)[\text{Relu}(0.2 \times 3 + 0)] + 0 = 0.18$

Current loss: $\frac{1}{3}[(0.06 - 3)^2 + (0.12 - 4)^2 + (0.18 - 5)^2]$

$= \frac{1}{3}[5.88 + 15.05 + 23.23]$

$= \frac{1}{3}[44.16] = 14.72$

$\Delta = (y - \hat{y}) \sigma'(y)$ ⟶ There,

$S_{10} = (3 - 0.06) = 2.94$

$S_{20} = (4 - 0.12) = 3.88$

$S_{30} = (5 - 0.18) = 4.82$

weight update:

$$w_2 += \frac{0.01[2.94(0.06) + 3.88(0.12) + 4.82(0.18)]}{3}$$

$$= 0.3 + 0.5032$$

$$= 0.8032$$

$$b_2 = b_2 + (0.01)\frac{[2.94 + 3.88 + 4.82]}{3}$$

$$= 0 + 0.0388$$

$$S_{11} = \text{Relu}'(a) \times 11.64 \times 0.2$$

$$= 1 \times 11.64 \times 0.2$$

$$= 2.32$$

$$w_1 = 0.2 + (0.01) \times (2.32$$

$$= 0.2232$$

$$b_1 = 0 + (0.01)(2.32) = 0.0232$$

loss find =

$$\hat{y}_1 = (0.8032) \times \text{Relu}(0.2232 \times 1 + 2.32) + 0.232$$

$$= 0.4090 + 0.232 = 0.641$$

$$\hat{y}_2 = (0.8032) \times \text{Relu}(0.2232 \times 2 + 0.232) + 0.232$$

$$= 0.8032 \times 0.6784 + 0.232 = 0.776$$

$$\hat{y}_3 = (0.8032) \times \text{Relu}(0.2232 \times 3 + 0.232) + 0.232$$

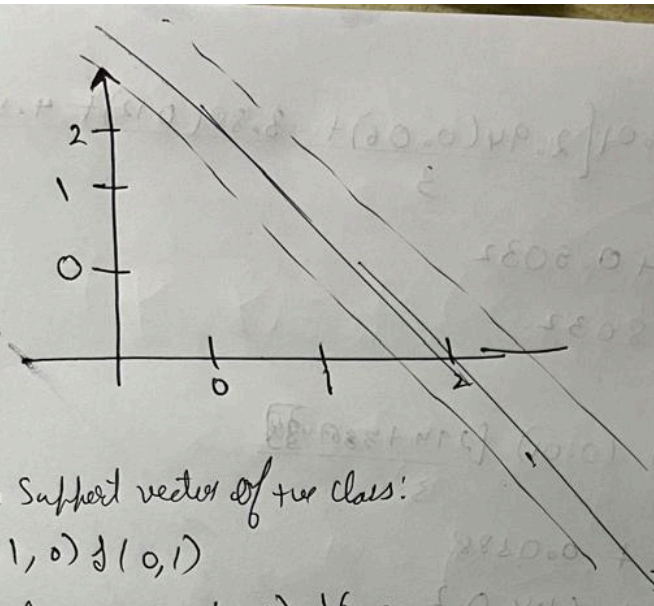
$$= 0.966$$

$$\text{loss} = \frac{1}{3}\sum_{i=1}^{n}(y_i - \hat{y}_i)^2 = \frac{2.359 + 6.448 + 16.33}{3}$$

$$= 8.379$$

Q2

a.)



b.) The support vector of +ve class:

(1, 0) & (0, 1)

& of -ve as (1, 1) & (2, 0)

slope is same for the same

Class of supported vectors, it is -1

for +ve ~~it is~~ +ve class it is $x_1 + x_2 = 1$

for -ve it is $x_1 + x_2 = 2$

obtained is $x_1 + x_2 = 1.5$

weight vector is (1, 1)

Q3 given weights:

$w_1 = -2, w_2 = 0, b = 5$

a.) $\gamma_i = \frac{|w x_i + b|}{\|w\|}$

$\hat{\gamma}_i = |w x_i + b|$

$\hat{\gamma} = \min_{i=0} \hat{\gamma}_i$

$\gamma = \min_{i=0} \gamma_i$

$\gamma_1 = 3/2, \hat{\gamma}_1 = 3$

$\gamma_2 = \frac{1}{2}, \hat{\gamma}_2 = 1$

$\gamma_3 = 1/3, \hat{\gamma}_3 = 1$

$\gamma_4 = 3/2, \hat{\gamma}_4 = 3$

$\therefore \gamma_{min} = 1$

$\hat{\gamma}_{min} = 1/2$

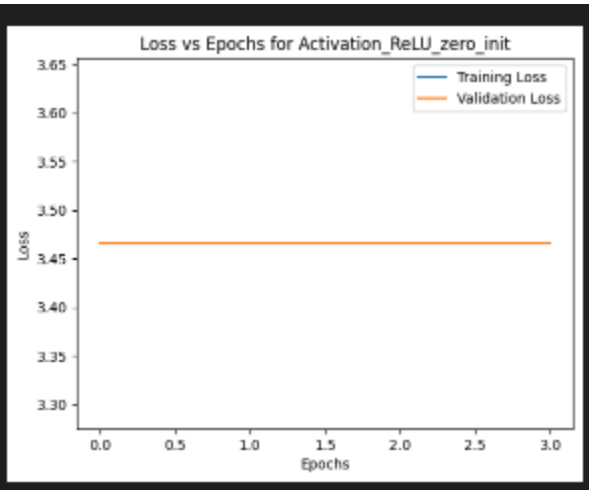
b.) (2,3) & (3,3) are support vectors of +ve & -ve classes. They are both closest & of same margin.

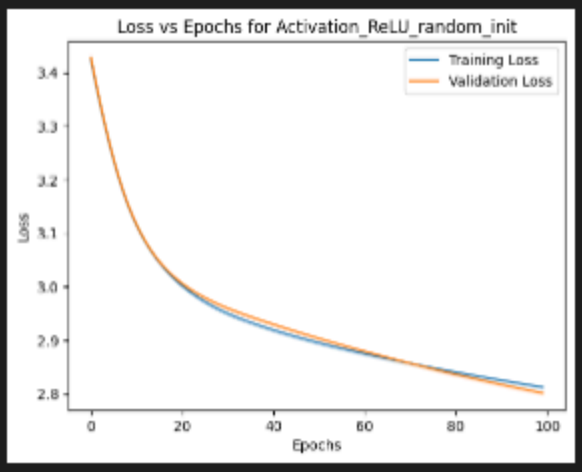
c.) $X_1 = 1, X_2 = 3$

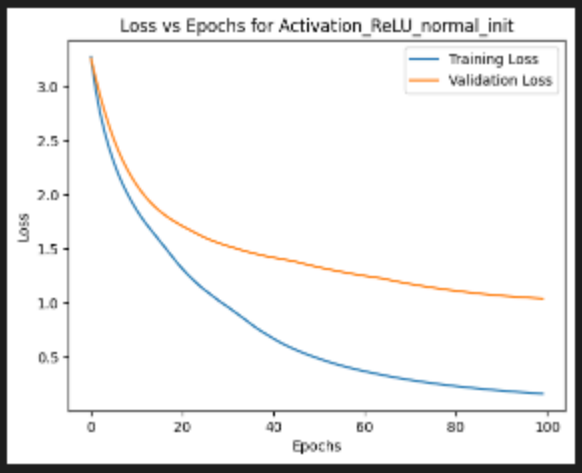
$(-2)(1) + (0)(3) + 5 = 3$

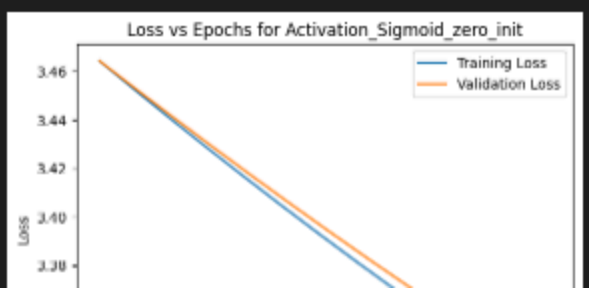
$3 > 0$, hence the sample belongs to +ve class.

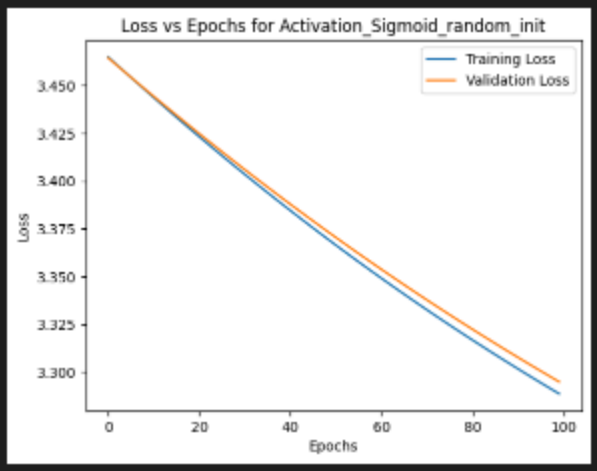
Part B

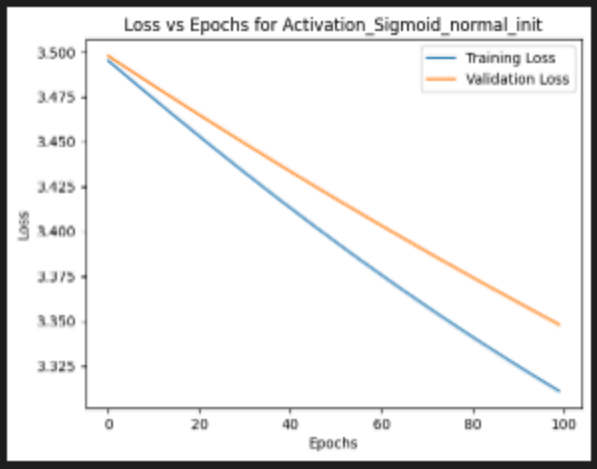

Loss vs Epochs for Activation_ReLU_zero_init
Training Loss
Validation Loss
Loss
Epochs
3.65
3.60
3.55
3.50
3.45
3.40
3.35
3.30
0.0
0.5
1.0
1.5
2.0
2.5
3.0

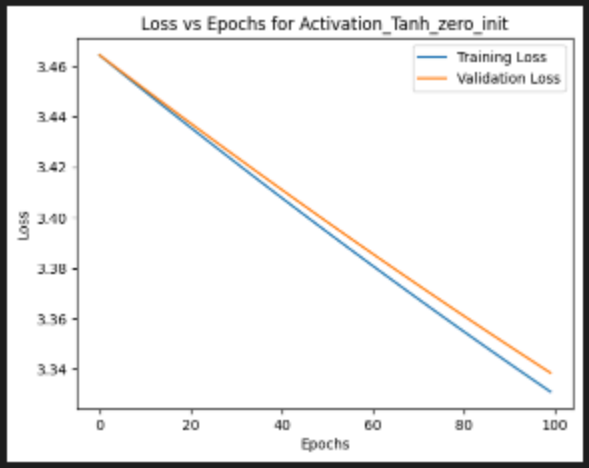

Loss vs Epochs for Activation_ReLU_random_init
Training Loss
Validation Loss
Loss
Epochs
3.4
3.3
3.2
3.1
3.0
2.9
2.8
0
20
40
60
80
100

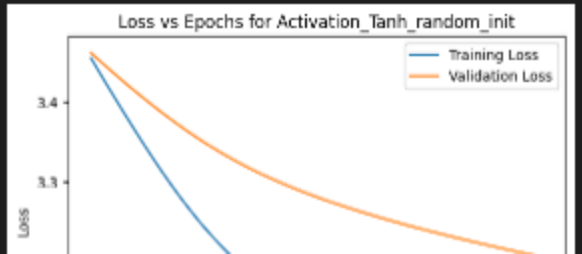

Loss vs Epochs for Activation_ReLU_normal_init
Training Loss
Validation Loss
Loss
Epochs
3.0
2.5
2.0
1.5
1.0
0.5
0
20
40
60
80
100

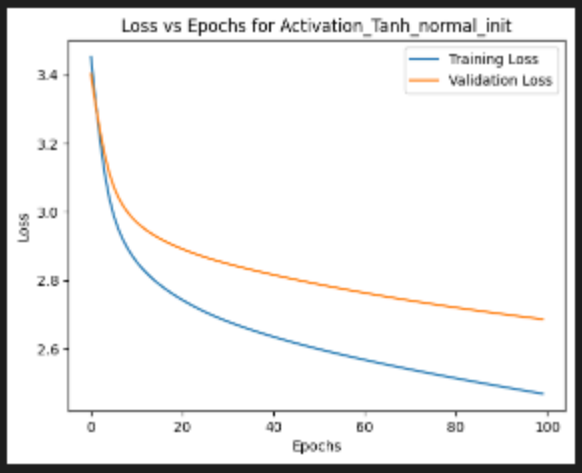

Loss vs Epochs for Activation_Sigmoid_zero_init
Training Loss
Validation Loss
Loss
3.46
3.44
3.42
3.40

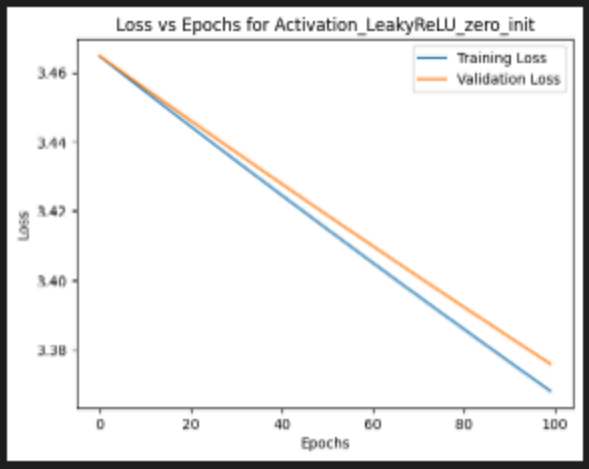

Loss vs Epochs for Activation_Sigmoid_random_init
Training Loss
Validation Loss
Loss
Epochs

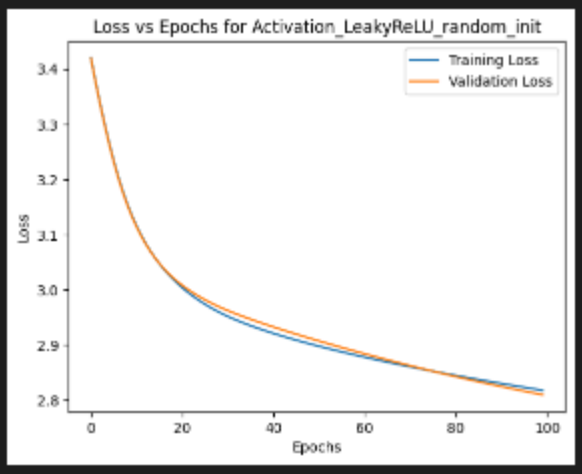

Loss vs Epochs for Activation_Sigmoid_normal_init
Training Loss
Validation Loss
Loss
Epochs

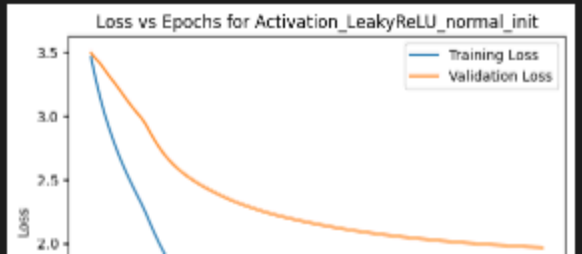

Loss vs Epochs for Activation_Tanh_zero_init
Training Loss
Validation Loss
Loss
Epochs


Loss vs Epochs for Activation_Tanh_random_init
Training Loss
Validation Loss
Loss


Loss vs Epochs for Activation_Tanh_normal_init
Training Loss
Validation Loss
Loss
Epochs


Loss vs Epochs for Activation_LeakyReLU_zero_init
Training Loss
Validation Loss
Loss
Epochs


Loss vs Epochs for Activation_LeakyReLU_random_init
Training Loss
Validation Loss
Loss
Epochs


Loss vs Epochs for Activation_LeakyReLU_normal_init
Training Loss
Validation Loss
Loss

the results of training an MLPClassifier using different activation functions (`relu`, `sigmoid`, `tanh`, `leakyrelu`) with three different weight initialization strategies (`zero`, `random`, `normal`). The performance is assessed by plotting the training loss vs. epochs and validation loss vs. epochs for each combination of activation and initialization. Below are the detailed observations and findings:

---

1. ReLU Activation

- **Zero Initialization**:
    - **Observation**: The training and validation loss remained constant throughout the training process.
    - **Explanation**: Initializing weights to zero for ReLU activation caused the network to fail to learn, as all neurons remained inactive. This resulted in no change in loss.
- **Random Initialization**:
    - **Observation**: The loss curves showed good convergence, with both training and validation loss decreasing. However, the validation loss was slightly lower than the training loss towards the end.
    - **Explanation**: Random initialization allowed the network to learn effectively. The slightly lower validation loss suggests that the model was able to generalize well, possibly benefiting from some regularization effects.
- **Normal Initialization**:
    - **Observation**: The model converged, but there was a noticeable gap between the training and validation loss after convergence, with the training loss being much lower.
    - **Explanation**: The large gap indicates overfitting, where the model learned the training data well but failed to generalize to the validation set.

---

2. Sigmoid Activation

- **Zero Initialization**:
    - **Observation**: The loss curve was a straight line downwards.
    - **Explanation**: Sigmoid activation with zero initialization faced gradient issues, leading to ineffective learning. The network's neurons did not update properly.
- **Random Initialization**:
    - **Observation**: Similar to zero initialization, the loss curve was a straight line.
    - **Explanation**: Sigmoid activation suffers from gradient vanishing problems, which were exacerbated by poor initialization strategies like zero and random.
- **Normal Initialization**:

    - **Observation**: The model showed a small gap between training and validation loss, with both decreasing steadily, similar to the trends seen with zero and random initialization.
    - **Explanation**: Normal initialization provided some improvement in learning compared to zero and random initialization. However, the gap between training and validation loss suggests that sigmoid continues to struggle with deep networks due to the gradient vanishing issue, limiting the model's performance.
    -

---

3. Tanh Activation

- **Zero Initialization**:
    - **Observation**: The loss curve was a straight line, similar to sigmoid.
    - **Explanation**: Like sigmoid, tanh activation with zero initialization failed to learn properly due to vanishing gradient issues.
- **Random Initialization**:
    - **Observation**: Both training and validation losses converged, but the loss values were large, indicating a high overall loss.
    - **Explanation**: Random initialization allowed the network to train, but the large loss values suggest that it struggled to reduce the loss effectively. The model converged, but it was not optimized well.
- **Normal Initialization**:
    - **Observation**: The model showed better convergence with normal initialization, with a more balanced loss curve and smaller gap between training and validation loss.
    - **Explanation**: Normal initialization performed the best with tanh, as it avoided issues like vanishing gradients, allowing for better learning and generalization.

---

4. Leaky ReLU Activation

- **Zero Initialization**:
    - **Observation**: The loss curve was a straight line, similar to ReLU with zero initialization.
    - **Explanation**: Leaky ReLU also suffered from inactive neurons when initialized with zero weights, preventing any learning.
- **Random Initialization**:
    - **Observation**: The model showed good convergence, with the validation loss being slightly lower than the training loss at the end.
    - **Explanation**: Random initialization worked well with Leaky ReLU, enabling the network to learn effectively and generalize well.
- **Normal Initialization**:

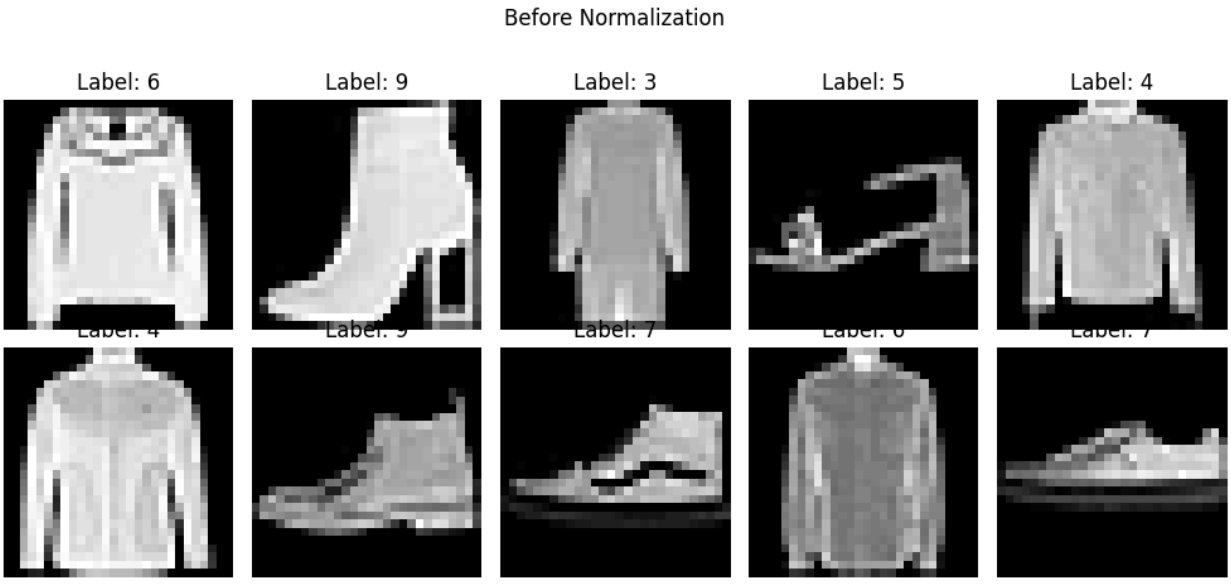
- **Observation**: The model converged, but both training and validation loss were large, indicating suboptimal performance.
- **Explanation**: Normal initialization performed poorly with Leaky ReLU, likely due to the larger initial weight values leading to slower convergence and larger final losses.

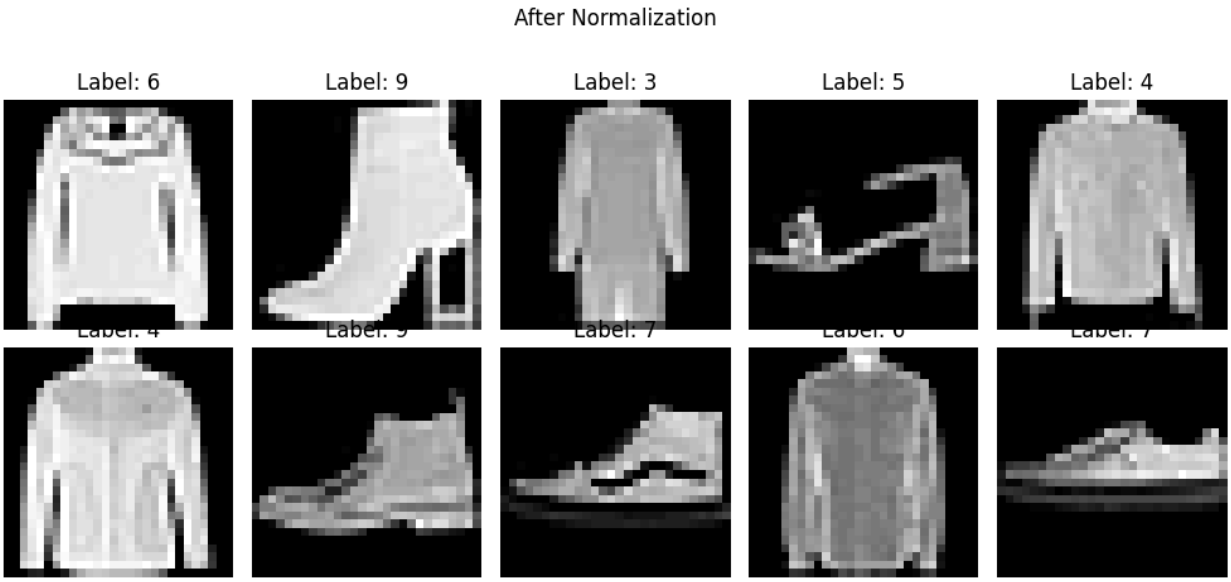
Part C

a) **Data Preprocessing and Visualization**

We used the Fashion-MNIST dataset for this task, selecting the first 8000 samples for training and the first 2000 for testing. The data was preprocessed through normalization, which scales pixel values from [0, 255] to [0, 1] for more efficient model training.

Visualization: 10 randomly selected samples from the test dataset were visualized. These images represent different clothing items such as t-shirts, pants, and shoes. They offer a quick insight into the variety of items within the Fashion-MNIST dataset.





b)

MLP Classifier Training

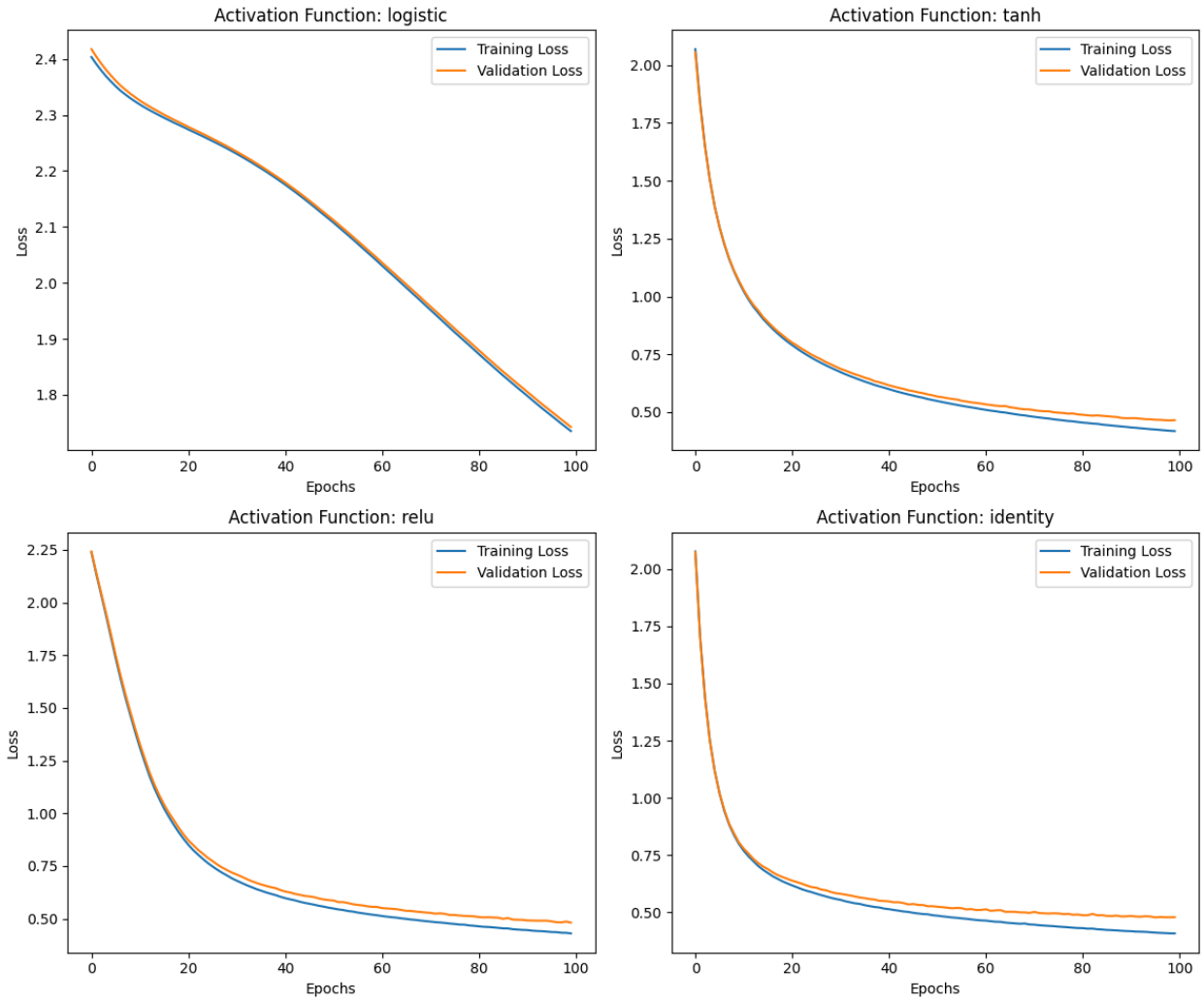
We trained an MLPClassifier with the following parameters:

- **Layers**: [128, 64, 32]
- **Iterations**: 100
- **Solver**: ada
- **Batch Size**: 128
- **Learning Rate**: 2e-5
- **Activation Functions Tested**: logistic, tanh, relu, identity

For each activation function, training and validation loss curves were plotted over the epochs.

Findings:

- The **tanh** activation function showed the best performance on the test set.
- **relu** showed high validation loss fluctuations, and **identity** struggled to capture complex patterns.
- Logistic activation also converged but slower than **tanh**.




```
Summary of Final Training and Validation Losses:
Activation: logistic - Final Train Loss: 1.7685 - Final Val Loss: 1.7785
Activation: tanh - Final Train Loss: 0.4260 - Final Val Loss: 0.4756
Activation: relu - Final Train Loss: 0.4400 - Final Val Loss: 0.4817
Activation: identity - Final Train Loss: 0.4190 - Final Val Loss: 0.4718
```

c)

Using the best activation function (**tanh**), we conducted a grid search over the following hyperparameters:

- **Solvers**: adam, sgd, lbfgs
- **Learning Rate Init**: [1e-4, 5e-5, 2e-5, 1e-5]
- **Batch Size**: [64, 128, 256]

Best Hyperparameters:

- **Batch Size**: 128
- **Learning Rate Init**: 0.0001
- **Solver**: adam

These hyperparameters achieved the lowest negative log loss, indicating improved model performance.

```
[CV] END batch_size=256, learning_rate_init=1e-05, solver=adam; total time=  38.2s
Best Hyperparameters: {'batch_size': 128, 'learning_rate_init': 0.0001, 'solver': 'adam'}
Best Validation Score (Negative Log Loss): -0.43840764667335175
```

d). MLPRegressor for Image Regeneration

We designed two MLPRegressor models with 5 layers of size [c, b, a, b, c], where c > b > a, specifically:

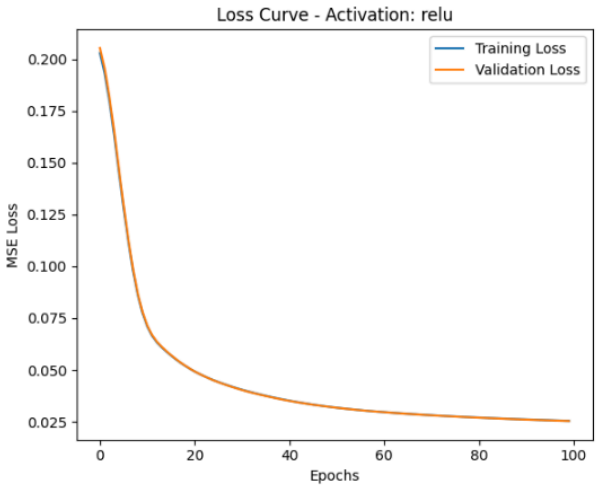
- [128, 64, 32, 64, 128]

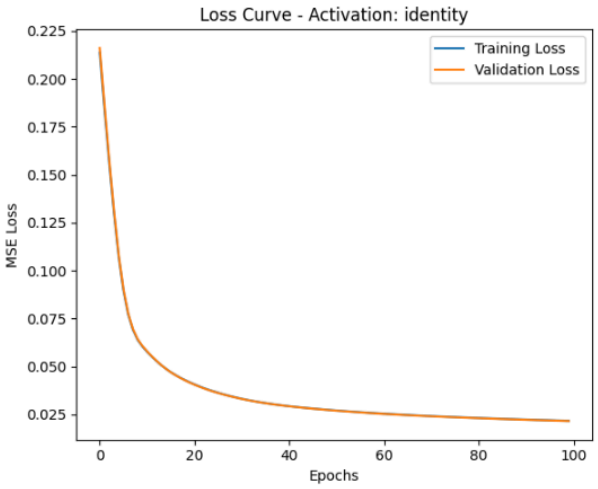
Two models were trained with:

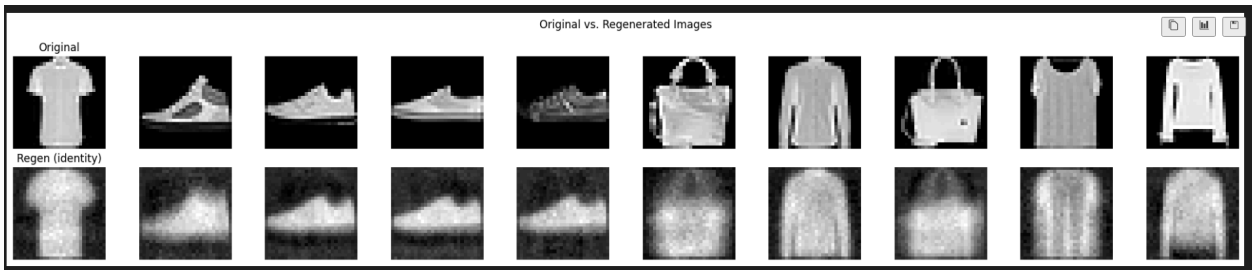
- **Activation Functions**: relu, identity
- **Solver**: adam
- **Learning Rate**: 2e-5

Training and Validation Loss: For both networks, the loss curves showed that the models were learning and converging correctly. The **relu** network converged faster than **identity**, but both were able to regenerate input images effectively.

Visualization: Post-training, we visualized 10 regenerated test samples from both networks. The **relu** network produced sharper regenerated images compared to **identity**, which appeared more blurred.







e)

Feature Extraction and New MLP Classifiers

From the two MLPRegressor networks, we extracted the feature vector of size a for the train and test data. Using this feature vector as the new image representation, we trained two new smaller MLP classifiers, each with 2 layers of size a.

- **Iterations**: 200
- **Solver**: adam
- **Learning Rate**: 2e-5

Accuracy Metrics: These smaller classifiers performed decently compared to the original classifier. Even though the input features were compressed into a lower-dimensional space, the new classifiers were able to retain essential information, thanks to the representative nature of the feature vectors extracted by the MLPRegressor.

Performance Comparison:

- The classifier in part 2 was directly trained on high-dimensional raw pixel data (784 features per image).
- In part 5, we used feature vectors of reduced dimensionality (size a, which was much smaller than 784) extracted from the MLPRegressor network, and trained two smaller MLP classifiers on these compressed features.

Reasons for Decent Performance in Part 5:

1. **Dimensionality Reduction**: The MLPRegressor in part 4 served as a form of dimensionality reduction. By forcing the network to regenerate images through a bottleneck layer of size a, it learned compact, essential representations of the images. These representations capture important features while discarding noise or redundant information, similar to how autoencoders work. As a result, the classifier in part 5 was able to use this distilled information effectively.
2. **Feature Learning**: The MLPRegressor learned an internal structure of the data through its hidden layers. The feature vectors extracted from the bottleneck layer contain rich representations that are more abstract and descriptive than raw pixel data. The smaller classifiers in part 5 used this distilled feature set, benefiting from the earlier "pre-training" done by the MLPRegressor.
3. **Model Complexity**: The classifiers in part 5 were smaller and less complex than the classifier in part 2, but due to the feature extraction performed by the MLPRegressor, they didn't need to learn complex relationships from raw pixel data. Instead, they worked on features that already contained significant information about the input images. This pre-processing made it easier for the smaller classifiers to perform well, even with fewer parameters.
4. **Regularization Effect**: The compression to a lower-dimensional space acted as a regularizer, reducing the risk of overfitting that might occur when training directly on raw pixel data. This reduced the model complexity and made the classifier in part 5 more robust, even with a smaller feature set.

Conclusion

- The **tanh** activation function was the best choice for the MLPClassifier.
- Hyperparameter tuning further improved the performance of the classifier.
- The feature extraction via MLPRegressor yielded a compressed but effective feature space, allowing smaller classifiers to still perform reasonably well. The feature extraction likely retained essential image characteristics, explaining the good performance.

```
Accuracy of smaller MLP Classifier with ReLU activation: 0.6940
Accuracy of smaller MLP Classifier with Identity activation: 0.6940
```

Reference:https://www.youtube.com/playlist?list=PLPTV0NXA_ZSj6tNyn_UadmUeU3Q3oR-hu