a) Single Training Iteration for an MLP with One Hidden Layer

Problem Setup

You are training a simple MLP with:

- Input $[1, 2, 3]$ and target $[3, 4, 5]$.
- The network has:
  - One hidden layer with ReLU activation.
  - One output layer with linear activation.
- Weights:
  - $W_1 \in \mathbb{R}^{2\times 3}$: from input to hidden layer.
  - $W_2 \in \mathbb{R}^{1\times 2}$: from hidden to output layer.
- Biases:
  - $b_1 \in \mathbb{R}^2$: for hidden layer.
  - $b_2 \in \mathbb{R}$: for output layer.
- Learning rate: $\eta = 0.01$.

Initial Weights and Biases

$$W_1 = \begin{bmatrix} 0.5 & -0.3 & 0.2 \\ 0.8 & 0.1 & -0.5 \end{bmatrix}, \quad b_1 = \begin{bmatrix} 0.1 \\ -0.2 \end{bmatrix}$$
$$W_2 = \begin{bmatrix} 0.7 & -0.4 \end{bmatrix}, \quad b_2 = 0.3$$

Step 1: Forward Pass

Input to hidden layer:

$$z_1 = W_1 \cdot x + b_1 = \begin{bmatrix} 0.5 & -0.3 & 0.2 \\ 0.8 & 0.1 & -0.5 \end{bmatrix} \cdot \begin{bmatrix} 1 \\ 2 \\ 3 \end{bmatrix} + \begin{bmatrix} 0.1 \\ -0.2 \end{bmatrix}$$

$$z_1 = \begin{bmatrix} 0.5 \cdot 1 + (-0.3) \cdot 2 + 0.2 \cdot 3 \\ 0.8 \cdot 1 + 0.1 \cdot 2 + (-0.5) \cdot 3 \end{bmatrix} + \begin{bmatrix} 0.1 \\ -0.2 \end{bmatrix}$$

$$z_1 = \begin{bmatrix} 0.5 - 0.6 + 0.6 \\ 0.8 + 0.2 - 1.5 \end{bmatrix} + \begin{bmatrix} 0.1 \\ -0.2 \end{bmatrix}$$

$$z_1 = \begin{bmatrix} 0.6 \\ -0.7 \end{bmatrix}$$

Apply ReLU activation:

$$h_1 = \max(0, z_1) = \max(0, \begin{bmatrix} 0.6 \\ -0.7 \end{bmatrix}) = \begin{bmatrix} 0.6 \\ 0 \end{bmatrix}$$

Input to output layer:

$$z_2 = W_2 \cdot h_1 + b_2 = \begin{bmatrix} 0.7 & -0.4 \end{bmatrix} \cdot \begin{bmatrix} 0.6 \\ 0 \end{bmatrix} + 0.3$$

$$z_2 = 0.7 \cdot 0.6 + (-0.4) \cdot 0 + 0.3 = 0.42 + 0 + 0.3 = 0.72$$

Step 2: Compute Loss (Mean Squared Error)

Given the target $y = 3$, the prediction $\hat{y} = z_2 = 0.72$, the loss is:

$$\text{MSE} = \frac{1}{2}(\hat{y} - y)^2 = \frac{1}{2}(0.72 - 3)^2 = \frac{1}{2}(-2.28)^2 = \frac{1}{2} \cdot 5.1984 = 2.5992$$

Step 3: Backpropagation

Gradient of the loss with respect to the output:

$$\frac{\partial \text{MSE}}{\partial z_2} = \hat{y} - y = 0.72 - 3 = -2.28$$

Gradients for output layer weights W_2 and bias b_2:

$$\frac{\partial \text{MSE}}{\partial W_2} = \frac{\partial \text{MSE}}{\partial z_2} \cdot \frac{\partial z_2}{\partial W_2} = -2.28 \cdot h_1^T = -2.28 \cdot \begin{bmatrix} 0.6 & 0 \end{bmatrix} = \begin{bmatrix} -1.368 & 0 \end{bmatrix}$$

$$\frac{\partial \text{MSE}}{\partial b_2} = \frac{\partial \text{MSE}}{\partial z_2} = -2.28$$

Gradients for hidden layer weights W_1 and biases b_1:

First, propagate the gradient back through ReLU:

$$\frac{\partial z_1}{\partial h_1} = W_2^T \cdot \frac{\partial \text{MSE}}{\partial z_2} = \begin{bmatrix} 0.7 \\ -0.4 \end{bmatrix} \cdot (-2.28) = \begin{bmatrix} -1.596 \\ 0.912 \end{bmatrix}$$

Since $z_1 = \begin{bmatrix} 0.6 \\ -0.7 \end{bmatrix}$, applying ReLU gives:

$\frac{\partial \text{MSE}}{\partial z_1} = \begin{bmatrix} -1.596 \\ 0 \end{bmatrix}$ (because the gradient through ReLU is 0 for negative inputs)

Now, compute the gradients w.r.t. W_1 and b_1:

$$\frac{\partial \text{MSE}}{\partial W_1} = \frac{\partial z_1}{\partial W_1} = \begin{bmatrix} -1.596 \\ 0 \end{bmatrix} \cdot \begin{bmatrix} 1 & 2 & 3 \end{bmatrix} = \begin{bmatrix} -1.596 & -3.192 & -4.788 \\ 0 & 0 & 0 \end{bmatrix}$$

$$\frac{\partial \text{MSE}}{\partial b_1} = \begin{bmatrix} -1.596 \\ 0 \end{bmatrix}$$

Step 4: Update Weights and Biases

The update rule is:

$$W \leftarrow W - \eta \cdot \frac{\partial \text{MSE}}{\partial W}, \quad b \leftarrow b - \eta \cdot \frac{\partial \text{MSE}}{\partial b}$$

Update W_2 and b_2:

$$W_2 \leftarrow \begin{bmatrix} 0.7 & -0.4 \end{bmatrix} - 0.01 \cdot \begin{bmatrix} -1.368 & 0 \end{bmatrix} = \begin{bmatrix} 0.71368 & -0.4 \end{bmatrix}$$

$$b_2 \leftarrow 0.3 - 0.01 \cdot (-2.28) = 0.3 + 0.0228 = 0.3228$$

Update W_1 and b_1:

$$W_1 \leftarrow W_1 - 0.01 \cdot \begin{bmatrix} -1.596 & -3.192 & -4.788 \\ 0 & 0 & 0 \end{bmatrix}$$

$$= \begin{bmatrix} 0.5 + 0.01596 & -0.3 + 0.03192 & 0.2 + 0.04788 \\ 0.8 & 0.1 & -0.5 \end{bmatrix}$$

$$= \begin{bmatrix} 0.51596 & -0.26808 & 0.24788 \\ 0.8 & 0.1 & -0.5 \end{bmatrix}$$

$$b_1 \leftarrow \begin{bmatrix} 0.1 \\ -0.2 \end{bmatrix} - 0.01 \cdot \begin{bmatrix} -1.596 \\ 0 \end{bmatrix} = \begin{bmatrix} 0.1 + 0.01596 \\ -0.2 \end{bmatrix} = \begin{bmatrix} 0.11596 \\ -0.2 \end{bmatrix}$$

Final Updated Weights and Biases

After one training iteration, the updated parameters are:

$$W_1 = \begin{bmatrix} 0.51596 & -0.26808 & 0.24788 \\ 0.8 & 0.1 & -0.5 \end{bmatrix}, \quad b_1 = \begin{bmatrix} 0.11596 \\ -0.2 \end{bmatrix}$$

$$W_2 = \begin{bmatrix} 0.71368 & -0.4 \end{bmatrix}, \quad b_2 = 0.3228$$

b) SVM Maximum Margin Hyperplane Problem

We are given the following dataset:

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

Step 1: Plot the Points We can visualize the points in a 2D plane. The '+' points represent the positive class, and the '-' points represent the negative class. Step 2: Optimization Problem for SVM

The objective of the SVM is to find the hyperplane that maximizes the margin between the two classes. This involves solving the following optimization problem:

$$\min_{w,b} \frac{1}{2}\|w\|^2$$

subject to:

$$y_i(w \cdot x_i + b) \geq 1, \quad \text{for all } i$$

where y_i is the class label, $x_i = (x_{i1}, x_{i2})$ is the feature vector, $w = (w_1, w_2)$ is the weight vector, and b is the bias.

Step 3: Solution by Inspection

By visual inspection, we identify the support vectors as the points that lie on the margin boundaries. These are:

- Positive class support vectors: $(1, 0)$ and $(0, 1)$ - Negative class support vectors: $(1, 1)$ and $(2, 0)$ Step 4: Constructing the Decision Function

We now solve for the weight vector w and bias term b by setting up the following equations based on the support vectors:

1. For $(1, 0)$, label $+1$:

$$w_1 + b = 1$$

2. For $(0, 1)$, label $+1$:

$$w_2 + b = 1$$

3. For $(1, 1)$, label -1:

$$w_1 + w_2 + b = -1$$

4. For $(2, 0)$, label -1:

$$2w_1 + b = -1$$

Solving this system of equations yields:

$$w_1 = -2, \quad w_2 = -2, \quad b = 3$$

Step 5: Hyperplane Equation

The equation of the decision hyperplane is given by:

$$w_1x_1 + w_2x_2 + b = 0$$

Substituting the values of w_1, w_2, and b, we get:

$$-2x_1 - 2x_2 + 3 = 0$$

which simplifies to:

$$x_1 + x_2 = 1.5$$

Step 6: Plotting the Decision Boundary and Support Vectors

We can plot the data points, the decision boundary, and the support vectors. Below is the LaTeX code to plot this graph using the ‘tikz‘ package:

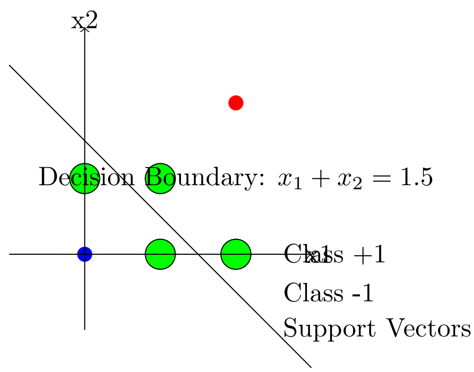


Figure 1: SVM Maximum Margin Hyperplane and Support Vectors

Conclusion

The weight vector for the maximum margin hyperplane is $w = (-2, -2)$, and the bias term is $b = 3$. The support vectors are the points $(1, 0)$, $(0, 1)$, $(1, 1)$, and $(2, 0)$.

c)

We are given the following SVM parameters and dataset:

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

The decision boundary is given by:

$$w \cdot x + b = 0 \quad \text{or} \quad -2 \cdot x_1 + 0 \cdot x_2 + 5 = 0 \quad \Rightarrow \quad x_1 = \frac{5}{2}$$

Part (a): Calculate the margin of the classifier

The margin γ for a linear SVM classifier is calculated as:

$$\gamma = \frac{2}{\|w\|}$$

where $\|w\|$ is the Euclidean norm of the weight vector $w = (w_1, w_2)$. In this case:

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

Now, substitute into the margin formula:

$$\gamma = \frac{2}{\|w\|} = \frac{2}{2} = 1$$

Answer for (a): The margin of the classifier is $\gamma = 1$.

Part (b): Identify the support vectors

The support vectors are the points that lie on the margin boundaries, where the decision function satisfies:

$$|w \cdot x + b| = 1$$

Substituting the given values, the decision function is:

$$w \cdot x + b = -2 \cdot x_1 + 5$$

Now, we check each sample to see if $|w \cdot x + b| = 1$.

- Sample 1: $x_1 = 1$, $x_2 = 2$

  $$w \cdot x + b = -2 \cdot 1 + 5 = -2 + 5 = 3 \quad \text{(not a support vector)}$$

- Sample 2: $x_1 = 2$, $x_2 = 3$

  $$w \cdot x + b = -2 \cdot 2 + 5 = -4 + 5 = 1 \quad \text{(on the margin boundary, support vector)}$$

  The equation for this support vector is:

  $$-2 \cdot 2 + 5 = 1$$

- Sample 3: $x_1 = 3$, $x_2 = 3$

  $w \cdot x + b = -2 \cdot 3 + 5 = -6 + 5 = -1$ (on the margin boundary, support vector)

  The equation for this support vector is:

$$-2 \cdot 3 + 5 = -1$$

- Sample 4: $x_1 = 4$, $x_2 = 1$

$$w \cdot x + b = -2 \cdot 4 + 5 = -8 + 5 = -3 \quad \text{(not a support vector)}$$

Answer for (b): The support vectors are Sample 2 and Sample 3. The equation for the support vector for Sample 2 is:

$$-2 \cdot 2 + 5 = 1$$

The equation for the support vector for Sample 3 is:

$$-2 \cdot 3 + 5 = -1$$

Part (c): Predict the class of a new point $x_1 = 1$, $x_2 = 3$

To predict the class of the new point, we calculate the decision function $w \cdot x + b$ for the point $x_1 = 1$, $x_2 = 3$:

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

Since $w \cdot x + b = 3 > 0$, the point is classified as $+1$.
Answer for (c): The class of the new point $(x_1 = 1, x_2 = 3)$ is $+1$.

SECTION-B

1)

```python
# Neural Network Class
class NeuralNetwork:
    def __init__(self, N, layer_sizes, lr, activation_func, weight_init_func, epochs, batch_size):
        self.N = N
        self.layer_sizes = layer_sizes
        self.lr = lr
        self.epochs = epochs
        self.batch_size = batch_size
        self.activation_func = activation_func
        self.weight_init_func = weight_init_func

        # Initialize weights and biases
        self.weights = [weight_init_func((layer_sizes[i-1], layer_sizes[i])) for i in range(1, N)]
        self.biases = [np.zeros((1, layer_sizes[i])) for i in range(1, N)]

    def forward(self, X):
        self.a = [X]
        self.z = []

        for i in range(self.N - 1):
            z = self.a[-1] @ self.weights[i] + self.biases[i]
            self.z.append(z)

            if i == self.N - 2:  # Output layer
                a = ActivationFunctions.softmax(z)
            else:
                a = self.activation_func(z)

            self.a.append(a)

        return self.a[-1]

    def backward(self, X, Y):
        m = X.shape[0]
        grads_w = [None] * (self.N - 1)
        grads_b = [None] * (self.N - 1)

        dz = self.a[-1] - Y
        for i in reversed(range(self.N - 1)):
            grads_w[i] = (self.a[i].T @ dz) / m
            grads_b[i] = np.sum(dz, axis=0, keepdims=True) / m

            if i != 0:
                da = dz @ self.weights[i].T
                dz = da * ActivationFunctions.relu_derivative(self.z[i-1])

        return grads_w, grads_b

```

```python
def update_weights(self, grads_w, grads_b):
    for i in range(self.N - 1):
        self.weights[i] -= self.lr * grads_w[i]
        self.biases[i] -= self.lr * grads_b[i]

def fit(self, X, Y, early_stopping=False, patience=10):
    Y_one_hot = np.eye(self.layer_sizes[-1])[Y]
    X_train, X_val, Y_train, Y_val = train_test_split(X, Y_one_hot, test_size=0.1)
    train_losses = []
    val_losses = []
    best_val_loss = float("inf")
    wait = 0

    for epoch in range(self.epochs):
        for i in range(0, X_train.shape[0], self.batch_size):
            X_batch = X_train[i:i + self.batch_size]
            Y_batch = Y_train[i:i + self.batch_size]

            self.forward(X_batch)
            grads_w, grads_b = self.backward(X_batch, Y_batch)
            self.update_weights(grads_w, grads_b)

        # Calculate and store train and validation loss
        train_loss = -np.sum(Y_train * np.log(self.forward(X_train))) / X_train.shape[0]
        val_loss = -np.sum(Y_val * np.log(self.forward(X_val))) / X_val.shape[0]
        train_losses.append(train_loss)
        val_losses.append(val_loss)

        if early_stopping:
            if val_loss < best_val_loss:
                best_val_loss = val_loss
                wait = 0
            else:
                wait += 1
                if wait >= patience:
                    print(f"Early stopping at epoch {epoch+1}")
                    break

        print(f"Epoch {epoch+1}/{self.epochs}, Train Loss: {train_loss:.4f}, Val Loss: {val_loss:.4f}")

    # Plot training and validation loss vs. epochs
    plt.plot(range(len(train_losses)), train_losses, label="Train Loss")
    plt.plot(range(len(val_losses)), val_losses, label="Validation Loss")
    plt.xlabel("Epochs")
    plt.ylabel("Loss")
    plt.legend()
    plt.title("Loss vs. Epochs")
    plt.show()

def predict_proba(self, X):
    return self.forward(X)

def predict(self, X):
    return np.argmax(self.predict_proba(X), axis=1)

def score(self, X, Y):
    Y_pred = self.predict(X)
    return accuracy_score(Y, Y_pred)
```

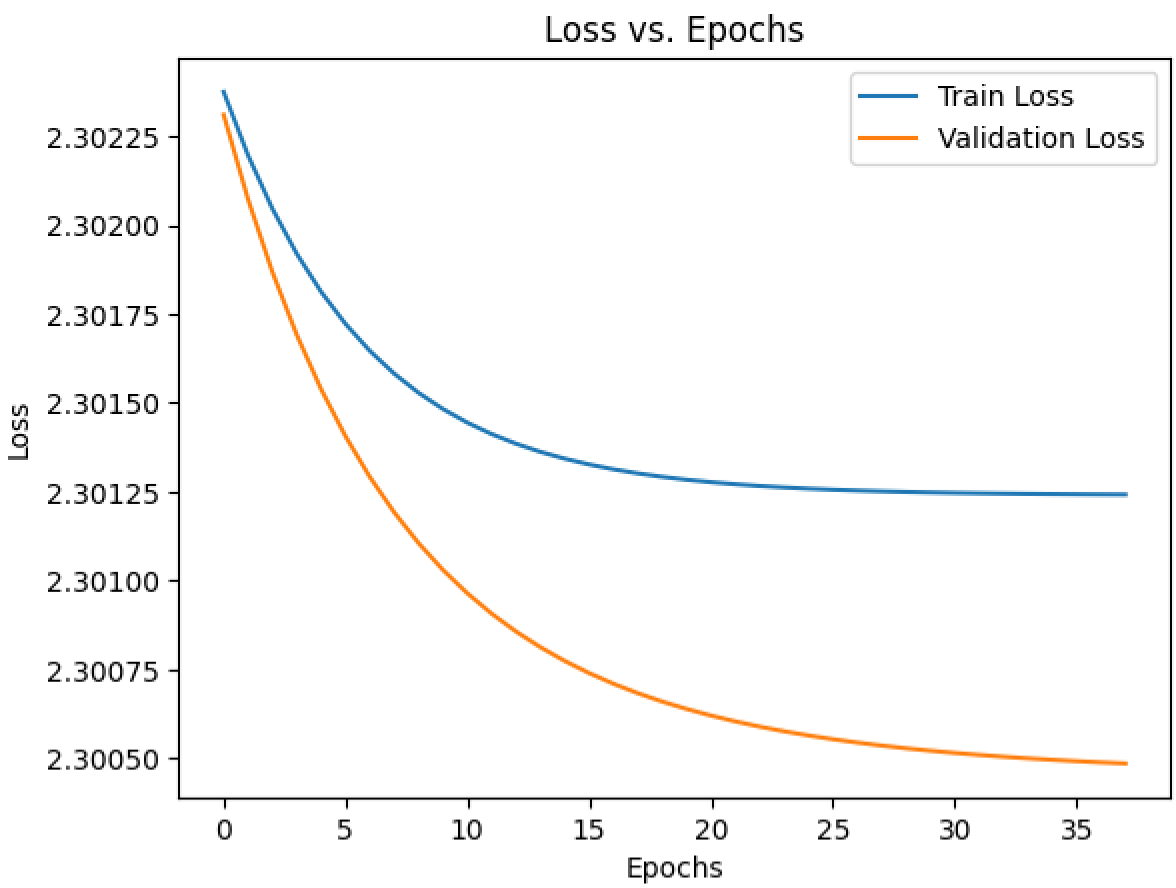
2)

```python
# Activation Functions Class
class ActivationFunctions:
    @staticmethod
    def sigmoid(x):
        return 1 / (1 + np.exp(-x))

    @staticmethod
    def sigmoid_derivative(x):
        s = ActivationFunctions.sigmoid(x)
        return s * (1 - s)

    @staticmethod
    def tanh(x):
        return np.tanh(x)

    @staticmethod
    def tanh_derivative(x):
        return 1 - np.tanh(x) ** 2

    @staticmethod
    def relu(x):
        return np.maximum(0, x)

    @staticmethod
    def relu_derivative(x):
        return np.where(x > 0, 1, 0)

    @staticmethod
    def leaky_relu(x, alpha=0.01):
        return np.where(x > 0, x, alpha * x)

    @staticmethod
    def leaky_relu_derivative(x, alpha=0.01):
        return np.where(x > 0, 1, alpha)

    @staticmethod
    def softmax(x):
        exps = np.exp(x - np.max(x, axis=1, keepdims=True))
        return exps / np.sum(exps, axis=1, keepdims=True)
```

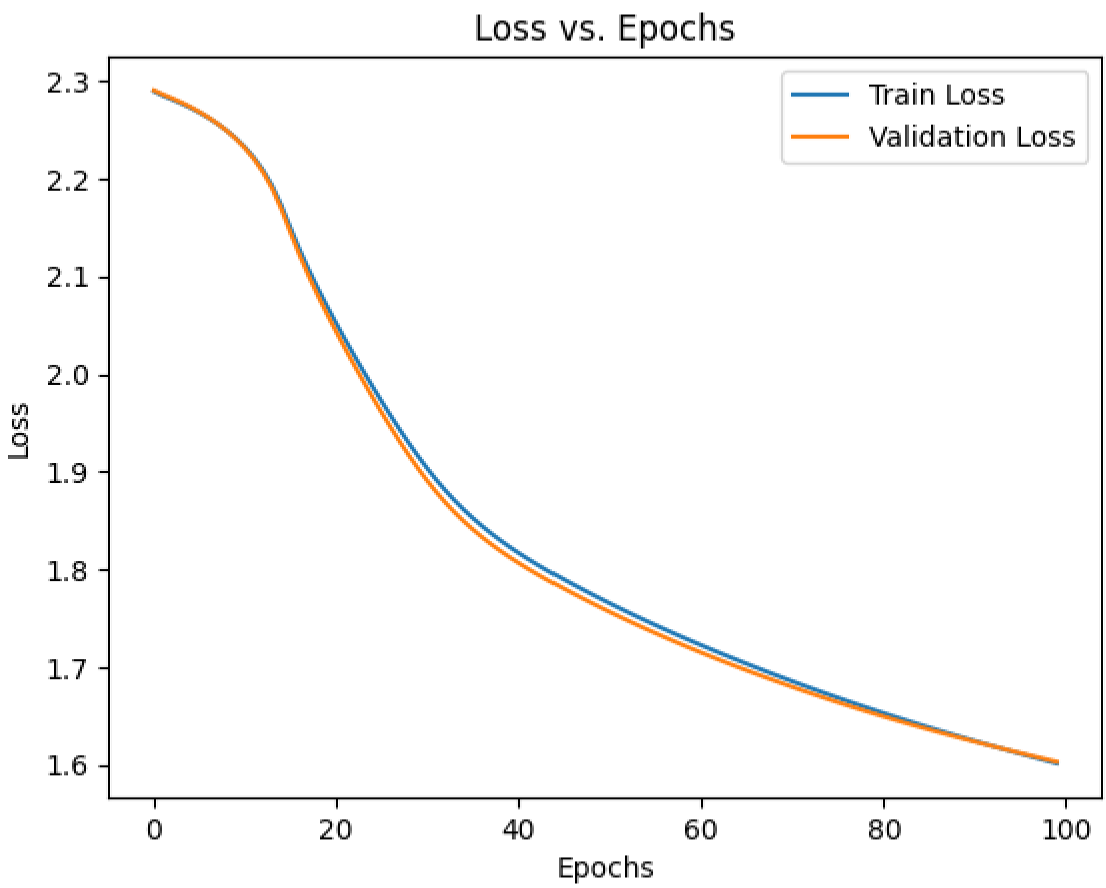
3)

```python
# Weight Initializations Class
class WeightInitializations:
    @staticmethod
    def zero_init(shape):
        return np.zeros(shape)

    @staticmethod
    def random_init(shape):
        return np.random.rand(*shape) * 0.01  # Small random values for stability

    @staticmethod
    def normal_init(shape):
        return np.random.randn(*shape) * np.sqrt(2 / shape[0])  # initialization for better convergence

```

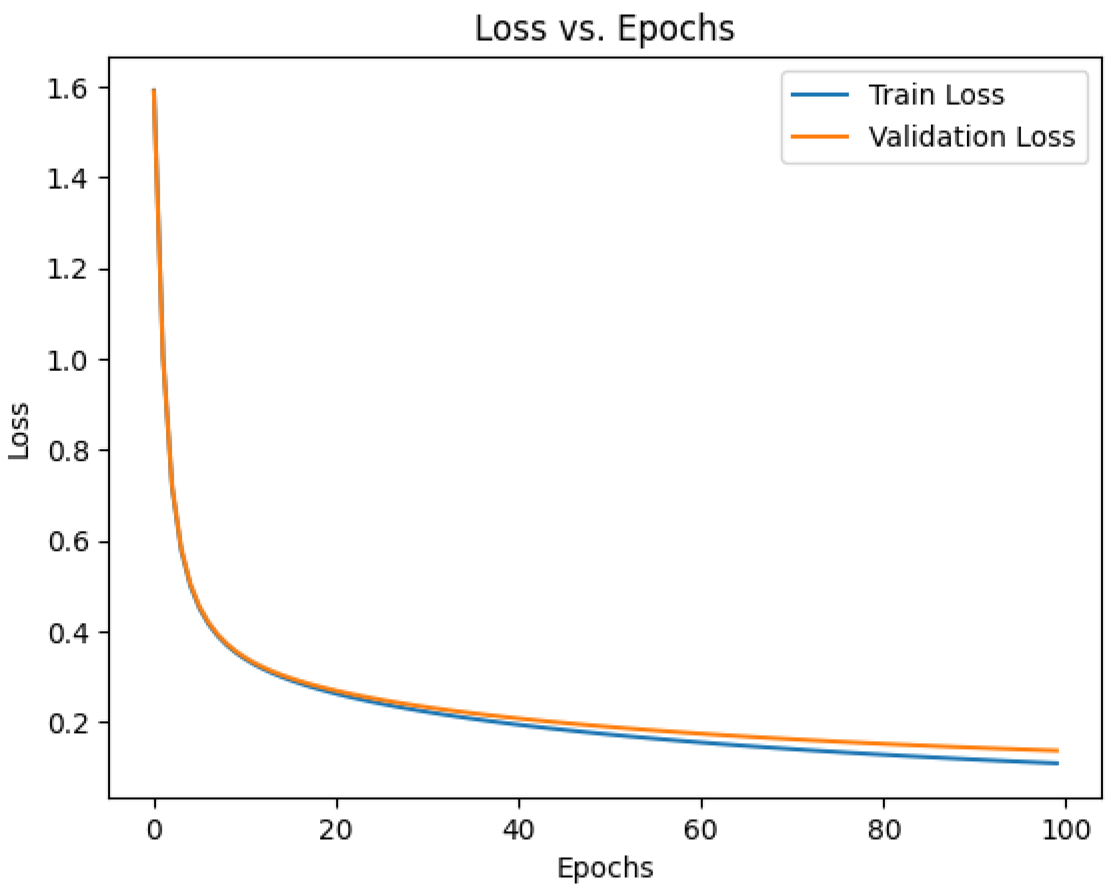
4) Training model with relu and zero_init



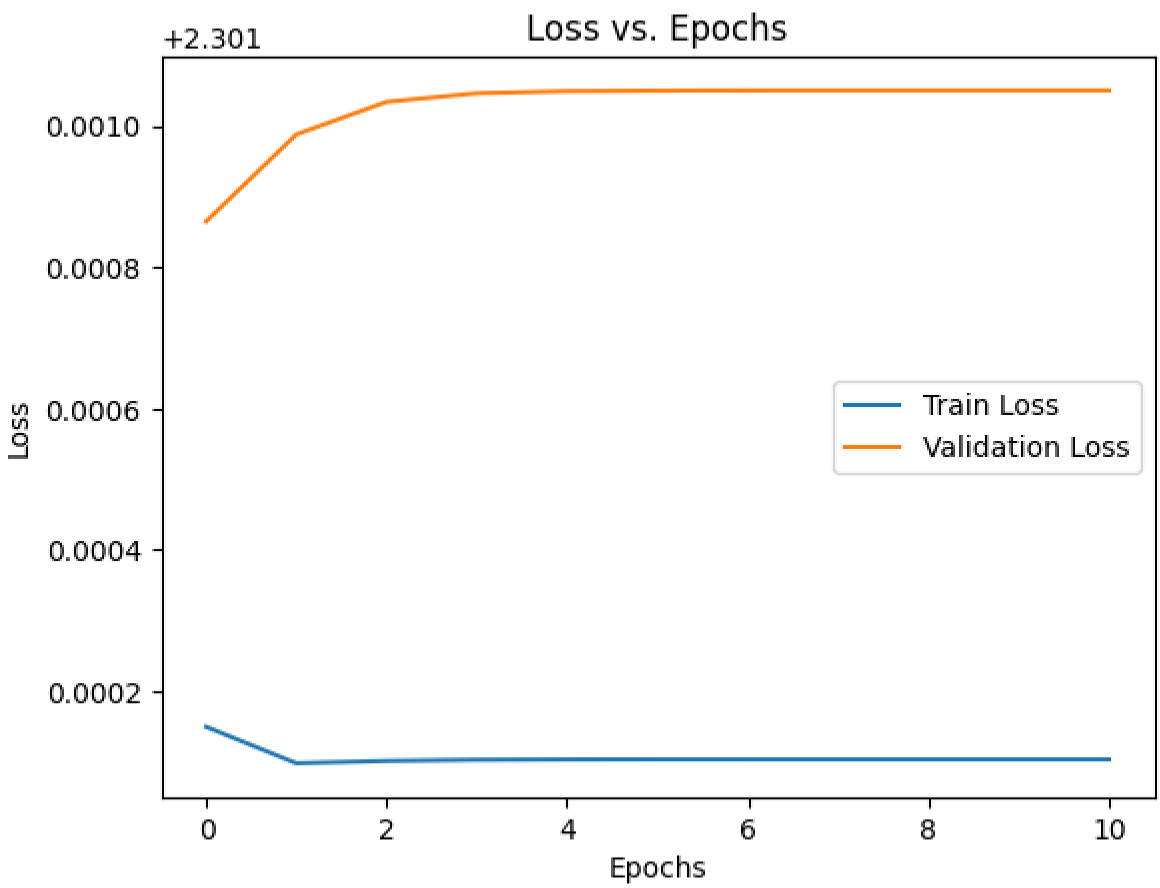
Training model with relu and random_init



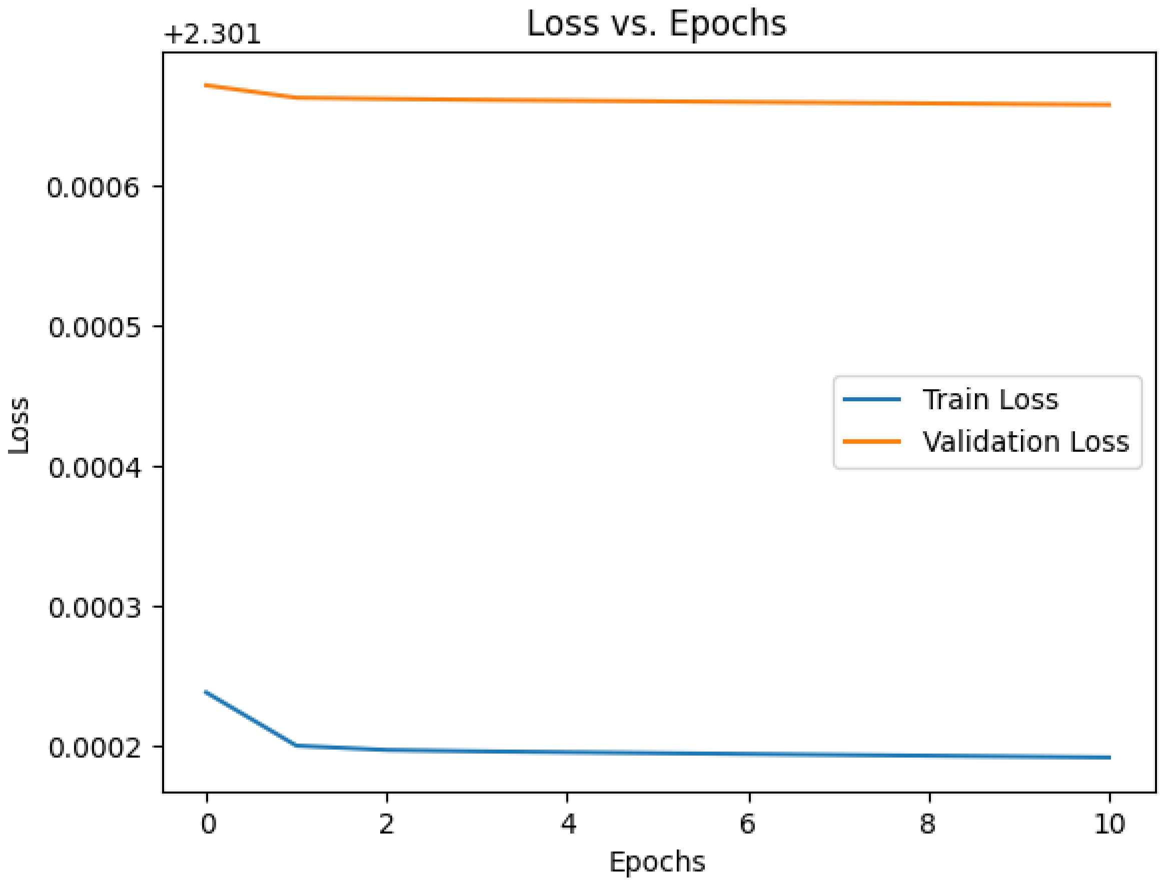
Training model with relu and normal_init



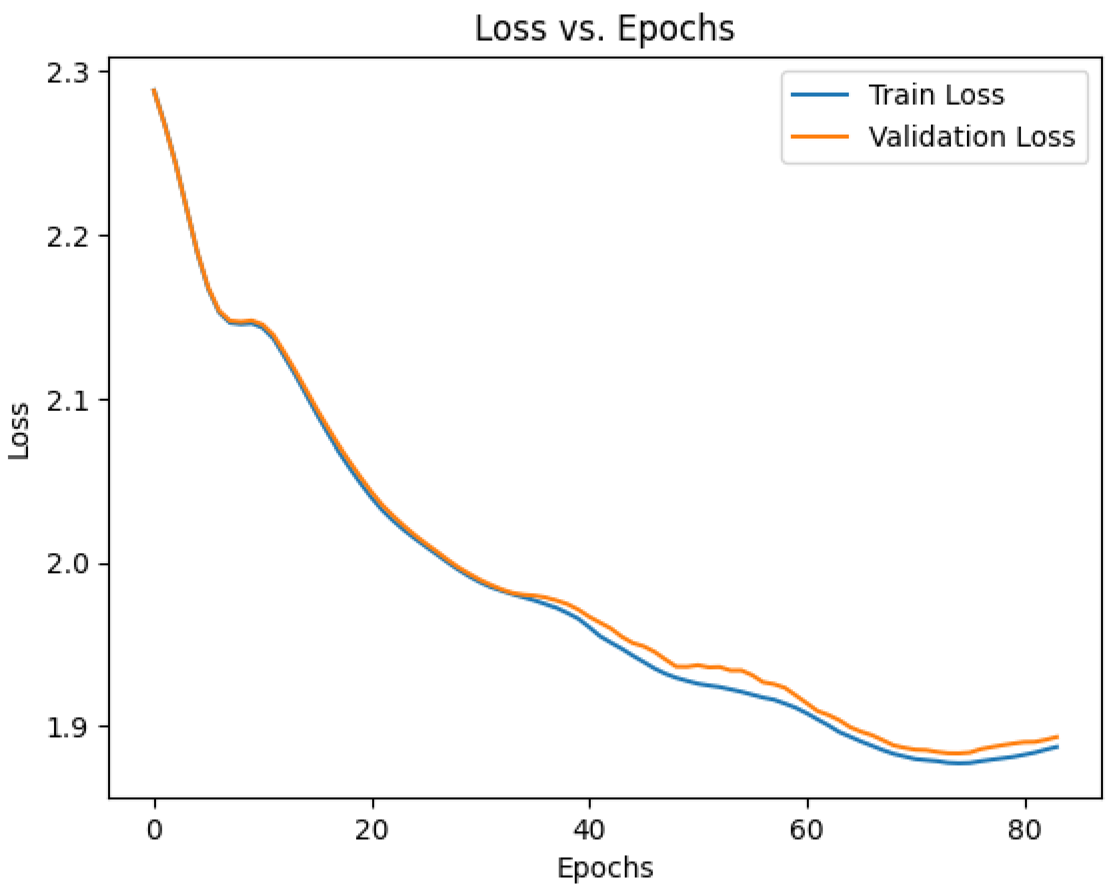
Training model with sigmoid and zero_init



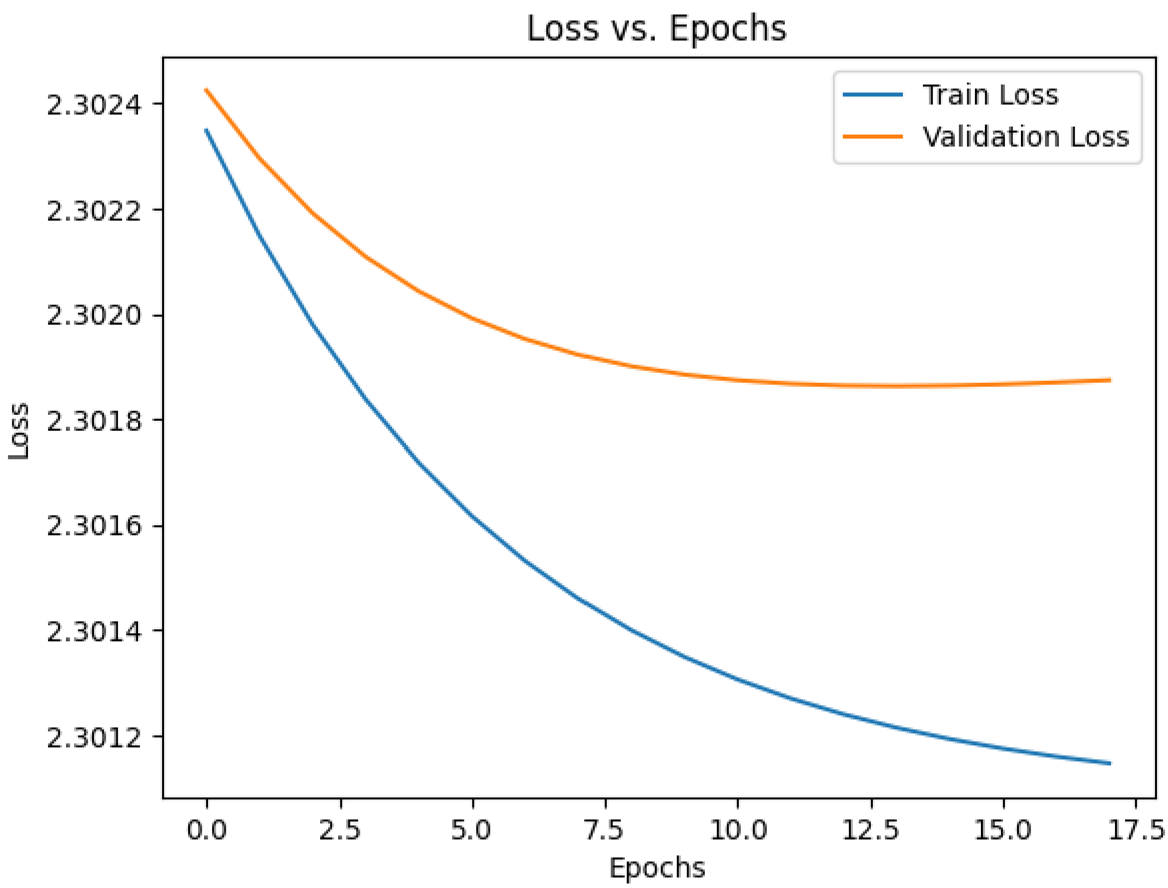
Training model with sigmoid and random_init



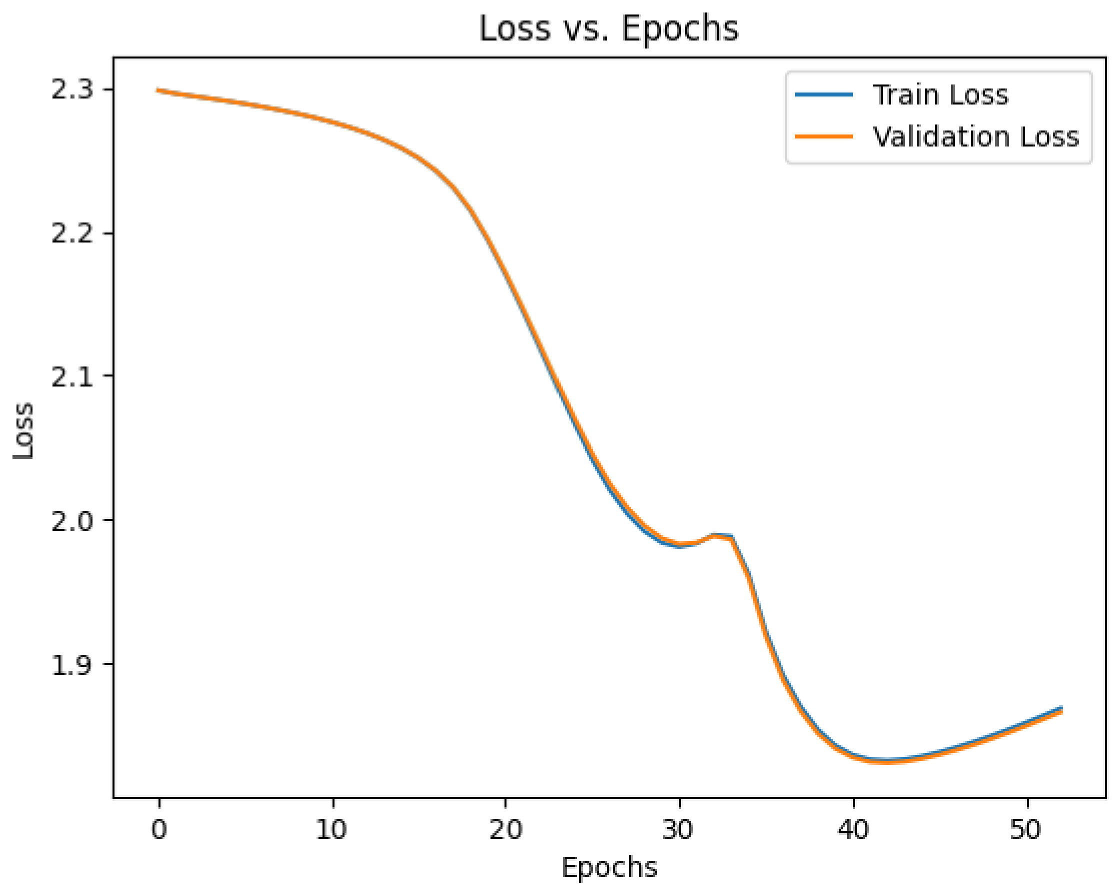
Training model with sigmoid and normal_init



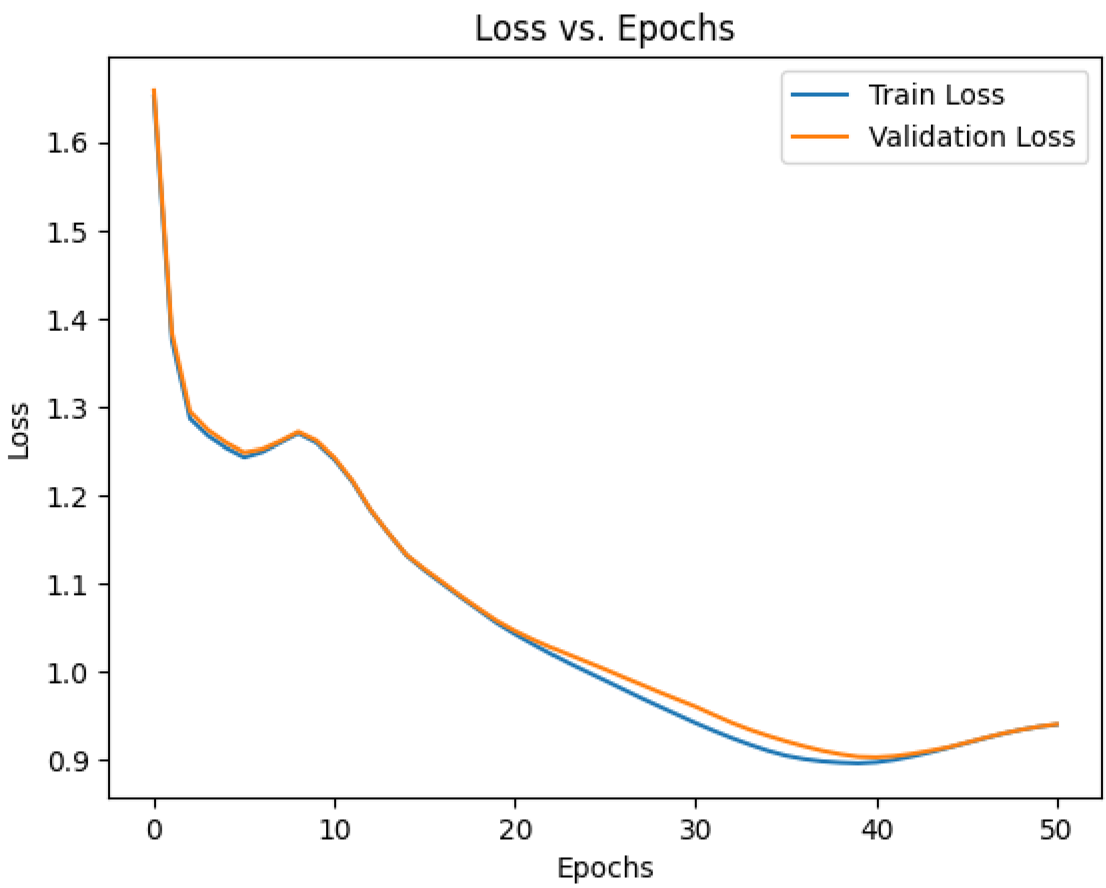
Training model with tanh and zero_init



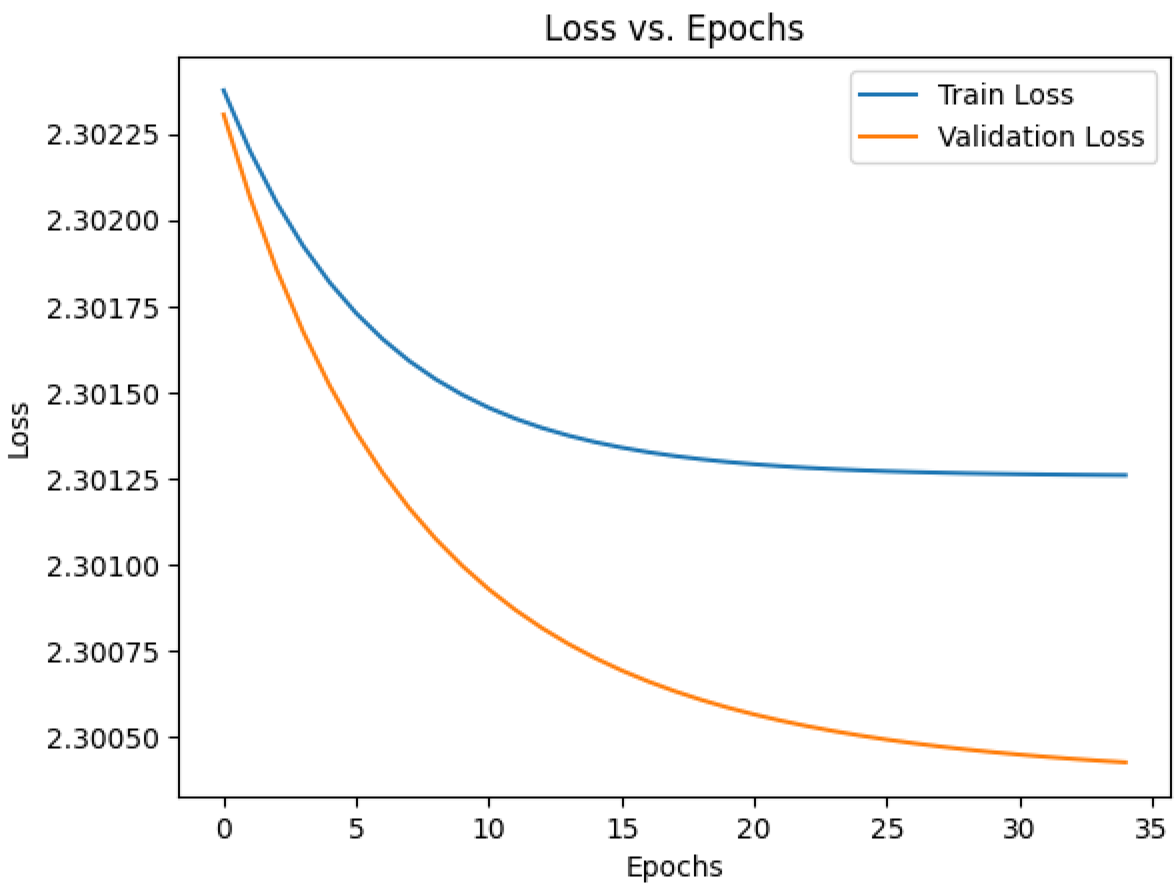
Training model with tanh and random_init



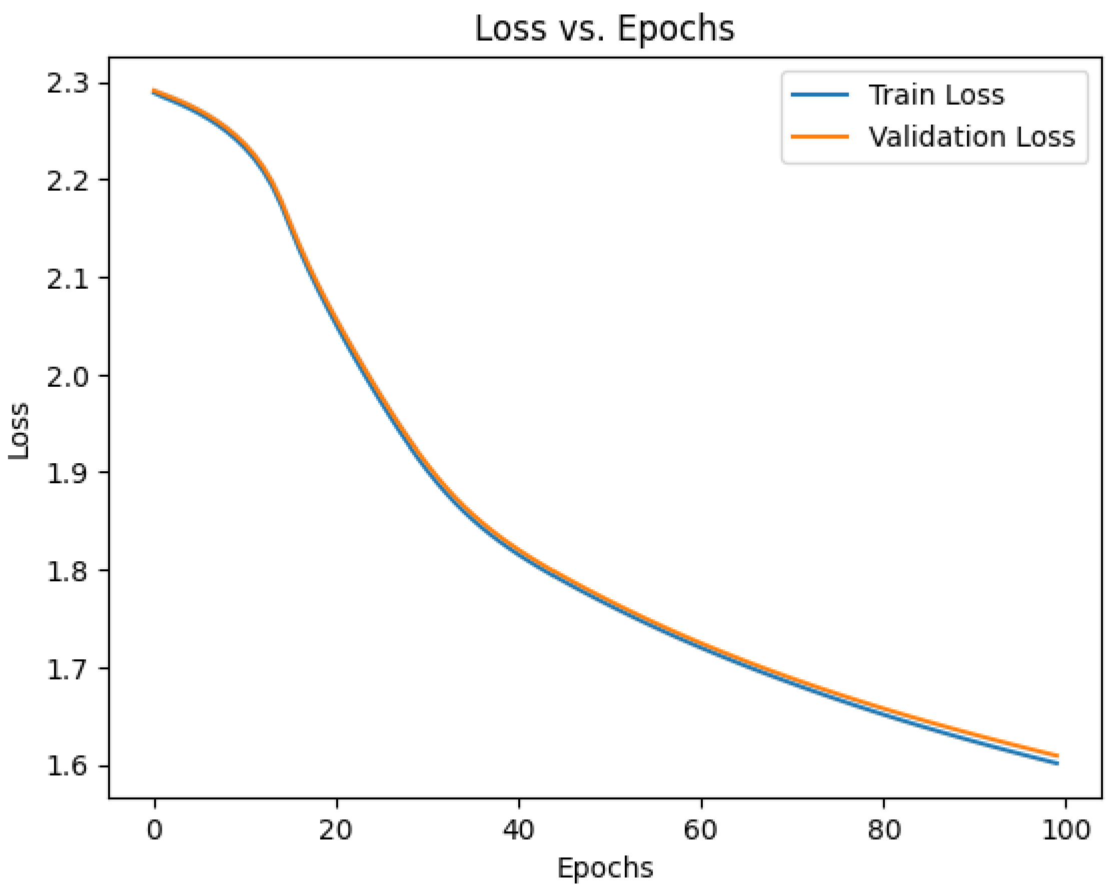
Training model with tanh and normal_init



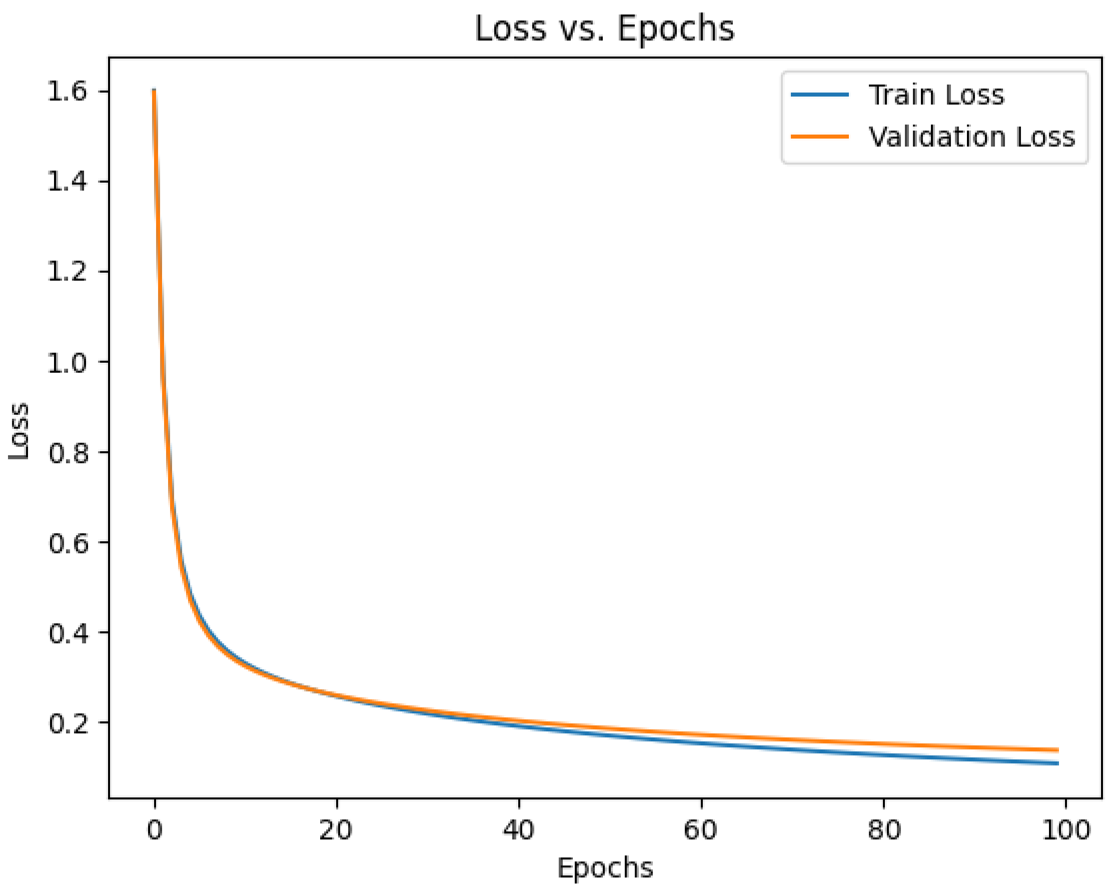
Training model with leaky_relu and zero_init



Training model with leaky_relu and random_init



Training model with leaky_relu and normal_init



Evaluating model: leaky_relu_normal_init.pkl

Test Accuracy for leaky_relu_normal_init.pkl: 96.15%

Classification Report:

	precision	recall	f1-score	support
0	0.97	0.98	0.98	980
1	0.98	0.99	0.98	1135
2	0.97	0.96	0.96	1032
3	0.94	0.96	0.95	1010
4	0.96	0.96	0.96	982
5	0.96	0.94	0.95	892
6	0.96	0.97	0.96	958

7	0.97	0.96	0.96	1028
8	0.95	0.95	0.95	974
9	0.96	0.94	0.95	1009
accuracy			0.96	10000
macro avg	0.96	0.96	0.96	10000
weighted avg	0.96	0.96	0.96	10000

Confusion Matrix:

```
[[ 964    0    2    2    0    4    6    1    1    0]
 [   0 1118    3    2    0    1    3    2    6    0]
 [   5    4  993    4    4    1    5    6   10    0]
 [   0    1    8  969    0   15    1    8    5    3]
 [   1    0    5    0  945    0    8    2    3   18]
 [   8    1    0   18    3  840    8    1   10    3]
 [   6    3    2    0    6    7  929    0    5    0]
 [   1   10   11    5    2    0    0  982    1   16]
 [   4    2    1   17    3    5    7    7  925    3]
 [   6    6    2    9   18    3    1    8    6  950]]
```

Evaluating model: leaky_relu_random_init.pkl

Test Accuracy for leaky_relu_random_init.pkl: 33.33%

Classification Report:

	precision	recall	f1-score	support
0	0.71	0.94	0.80	980
1	0.27	0.94	0.42	1135
2	0.30	0.33	0.31	1032
3	0.22	0.13	0.17	1010
4	0.19	0.05	0.08	982

```
           5       0.00      0.00      0.00       892
           6       0.23      0.36      0.28       958
           7       0.00      0.00      0.00      1028
           8       0.39      0.49      0.44       974
           9       0.00      0.00      0.00      1009

    accuracy                           0.33     10000
   macro avg       0.23      0.32      0.25     10000
weighted avg       0.23      0.33      0.25     10000
```

Confusion Matrix:

```
[[ 917    0   41    5    0    0   11    0    6    0]
 [   0 1072    1    2   30    0    6    0   24    0]
 [ 181   24  336   81   19    0  270    0  121    0]
 [  46   24  194  136   20    0  387    0  203    0]
 [   2  874    1    4   53    0    4    0   44    0]
 [  53   13  197  128   28    0  296    0  177    0]
 [  84   10  293  119    8    0  344    0  100    0]
 [   0  937    4    6   41    0    5    0   35    0]
 [  12   42   58  140   57    0  190    0  475    0]
 [   5  935    2    7   30    0    7    0   23    0]]
```

Evaluating model: leaky_relu_zero_init.pkl

Test Accuracy for leaky_relu_zero_init.pkl: 11.35%

Classification Report:

```
              precision    recall  f1-score   support

           0       0.00      0.00      0.00       980
           1       0.11      1.00      0.20      1135
           2       0.00      0.00      0.00      1032
```

3	0.00	0.00	0.00	1010
4	0.00	0.00	0.00	982
5	0.00	0.00	0.00	892
6	0.00	0.00	0.00	958
7	0.00	0.00	0.00	1028
8	0.00	0.00	0.00	974
9	0.00	0.00	0.00	1009
accuracy			0.11	10000
macro avg	0.01	0.10	0.02	10000
weighted avg	0.01	0.11	0.02	10000

Confusion Matrix:

```
[[   0  980    0    0    0    0    0    0    0    0]
 [   0 1135    0    0    0    0    0    0    0    0]
 [   0 1032    0    0    0    0    0    0    0    0]
 [   0 1010    0    0    0    0    0    0    0    0]
 [   0  982    0    0    0    0    0    0    0    0]
 [   0  892    0    0    0    0    0    0    0    0]
 [   0  958    0    0    0    0    0    0    0    0]
 [   0 1028    0    0    0    0    0    0    0    0]
 [   0  974    0    0    0    0    0    0    0    0]
 [   0 1009    0    0    0    0    0    0    0    0]]
```

Evaluating model: relu_normal_init.pkl

Test Accuracy for relu_normal_init.pkl: 96.13%

Classification Report:

	precision	recall	f1-score	support
0	0.97	0.98	0.98	980

1	0.98	0.98	0.98	1135
2	0.96	0.96	0.96	1032
3	0.95	0.95	0.95	1010
4	0.97	0.96	0.96	982
5	0.96	0.95	0.95	892
6	0.96	0.97	0.97	958
7	0.97	0.95	0.96	1028
8	0.95	0.95	0.95	974
9	0.95	0.95	0.95	1009

accuracy			0.96	10000
macro avg	0.96	0.96	0.96	10000
weighted avg	0.96	0.96	0.96	10000

Confusion Matrix:

```
[[ 963    0    1    1    0    6    4    2    2    1]
 [   0 1117    2    2    0    1    5    1    7    0]
 [   5    1  990    9    5    0    7    6    8    1]
 [   1    1   11  964    0   12    0    7   11    3]
 [   1    1    8    0  939    1    5    2    3   22]
 [   6    1    1   13    2  847    9    1    9    3]
 [   7    3    0    1    5    8  929    0    5    0]
 [   2    9   13    6    2    0    0  979    2   15]
 [   5    1    6   12    4    4    5    5  926    6]
 [   4    5    2    8   13    4    1    8    5  959]]
```

Evaluating model: relu_random_init.pkl

Test Accuracy for relu_random_init.pkl: 33.20%

Classification Report:

precision recall f1-score support

0	0.71	0.93	0.81	980
1	0.27	0.95	0.42	1135
2	0.29	0.40	0.33	1032
3	0.23	0.06	0.10	1010
4	0.17	0.04	0.06	982
5	0.00	0.00	0.00	892
6	0.21	0.34	0.26	958
7	0.00	0.00	0.00	1028
8	0.39	0.50	0.44	974
9	0.00	0.00	0.00	1009
accuracy			0.33	10000
macro avg	0.23	0.32	0.24	10000
weighted avg	0.23	0.33	0.24	10000

Confusion Matrix:

```
[[ 912    0   51    0    0    0   11    0    6    0]
 [   0 1081    2    1   19    0    5    0   27    0]
 [ 176   28  411   34   12    0  238    0  133    0]
 [  42   28  255   61   16    0  395    0  213    0]
 [   2  894    1    3   38    0    5    0   39    0]
 [  49   17  241   49   26    0  315    0  195    0]
 [  84   11  376   47    7    0  329    0  104    0]
 [   0  950    5    1   30    0    7    0   35    0]
 [  10   47   87   70   52    0  220    0  488    0]
 [   5  943    2    3   23    0    9    0   24    0]]
```

Evaluating model: relu_zero_init.pkl

Test Accuracy for relu_zero_init.pkl: 11.35%

Classification Report:

	precision	recall	f1-score	support
0	0.00	0.00	0.00	980
1	0.11	1.00	0.20	1135
2	0.00	0.00	0.00	1032
3	0.00	0.00	0.00	1010
4	0.00	0.00	0.00	982
5	0.00	0.00	0.00	892
6	0.00	0.00	0.00	958
7	0.00	0.00	0.00	1028
8	0.00	0.00	0.00	974
9	0.00	0.00	0.00	1009
accuracy			0.11	10000
macro avg	0.01	0.10	0.02	10000
weighted avg	0.01	0.11	0.02	10000

Confusion Matrix:

```
[[   0  980    0    0    0    0    0    0    0    0]
 [   0 1135    0    0    0    0    0    0    0    0]
 [   0 1032    0    0    0    0    0    0    0    0]
 [   0 1010    0    0    0    0    0    0    0    0]
 [   0  982    0    0    0    0    0    0    0    0]
 [   0  892    0    0    0    0    0    0    0    0]
 [   0  958    0    0    0    0    0    0    0    0]
 [   0 1028    0    0    0    0    0    0    0    0]
 [   0  974    0    0    0    0    0    0    0    0]
 [   0 1009    0    0    0    0    0    0    0    0]]
```

Evaluating model: sigmoid_normal_init.pkl

Test Accuracy for sigmoid_normal_init.pkl: 26.77%

Classification Report:

	precision	recall	f1-score	support
0	0.96	0.36	0.53	980
1	0.21	0.95	0.34	1135
2	0.00	0.00	0.00	1032
3	0.00	0.00	0.00	1010
4	0.00	0.00	0.00	982
5	0.00	0.00	0.00	892
6	0.14	0.00	0.01	958
7	0.43	0.45	0.44	1028
8	0.00	0.00	0.00	974
9	0.24	0.77	0.36	1009
accuracy			0.27	10000
macro avg	0.20	0.25	0.17	10000
weighted avg	0.20	0.27	0.17	10000

Confusion Matrix:

```
[[ 356  533    0    0    2    0    2    0    0   87]
 [    0 1080    0    0    0    0    2    0    0   53]
 [    0  886    0    0    0    0    3    0    0  143]
 [    0  865    0    0    1    0    2    1    0  141]
 [    1    7    0    0    0    0    0  390    0  584]
 [    0  569    0    0    0    0    7    0    1  315]
 [   13  656    0    0    1    0    4    1    0  283]
 [    0   23    0    0    0    0    1  465    0  539]
```

```
 [   0  619    0    0    0    0    8    0    0  347]
 [   0   18    0    0    0    0    0  219    0  772]]
Evaluating model: sigmoid_random_init.pkl
Test Accuracy for sigmoid_random_init.pkl: 11.35%
Classification Report:
              precision    recall  f1-score   support

           0       0.00      0.00      0.00       980
           1       0.11      1.00      0.20      1135
           2       0.00      0.00      0.00      1032
           3       0.00      0.00      0.00      1010
           4       0.00      0.00      0.00       982
           5       0.00      0.00      0.00       892
           6       0.00      0.00      0.00       958
           7       0.00      0.00      0.00      1028
           8       0.00      0.00      0.00       974
           9       0.00      0.00      0.00      1009

    accuracy                           0.11     10000
   macro avg       0.01      0.10      0.02     10000
weighted avg       0.01      0.11      0.02     10000

Confusion Matrix:
 [[   0  980    0    0    0    0    0    0    0    0]
 [   0 1135    0    0    0    0    0    0    0    0]
 [   0 1032    0    0    0    0    0    0    0    0]
 [   0 1010    0    0    0    0    0    0    0    0]
 [   0  982    0    0    0    0    0    0    0    0]
 [   0  892    0    0    0    0    0    0    0    0]
```

```
[   0  958    0    0    0    0    0    0    0    0]
[   0 1028    0    0    0    0    0    0    0    0]
[   0  974    0    0    0    0    0    0    0    0]
[   0 1009    0    0    0    0    0    0    0    0]]
```

Evaluating model: sigmoid_zero_init.pkl

Test Accuracy for sigmoid_zero_init.pkl: 11.35%

Classification Report:

```
              precision    recall  f1-score   support

           0       0.00      0.00      0.00       980
           1       0.11      1.00      0.20      1135
           2       0.00      0.00      0.00      1032
           3       0.00      0.00      0.00      1010
           4       0.00      0.00      0.00       982
           5       0.00      0.00      0.00       892
           6       0.00      0.00      0.00       958
           7       0.00      0.00      0.00      1028
           8       0.00      0.00      0.00       974
           9       0.00      0.00      0.00      1009

    accuracy                           0.11     10000
   macro avg       0.01      0.10      0.02     10000
weighted avg       0.01      0.11      0.02     10000
```

Confusion Matrix:

```
[[   0  980    0    0    0    0    0    0    0    0]
 [   0 1135    0    0    0    0    0    0    0    0]
 [   0 1032    0    0    0    0    0    0    0    0]
 [   0 1010    0    0    0    0    0    0    0    0]
```

[0 982   0   0   0   0   0   0   0   0]

[0 892   0   0   0   0   0   0   0   0]

[0 958   0   0   0   0   0   0   0   0]

[0 1028   0   0   0   0   0   0   0   0]

[0 974   0   0   0   0   0   0   0   0]

[0 1009   0   0   0   0   0   0   0   0]]

Evaluating model: tanh_normal_init.pkl

Test Accuracy for tanh_normal_init.pkl: 68.63%

Classification Report:

	precision	recall	f1-score	support
0	0.91	0.90	0.91	980
1	0.95	0.88	0.91	1135
2	0.82	0.68	0.74	1032
3	0.79	0.64	0.71	1010
4	0.78	0.51	0.61	982
5	0.72	0.37	0.49	892
6	0.86	0.71	0.78	958
7	0.94	0.78	0.85	1028
8	0.30	0.90	0.46	974
9	0.65	0.45	0.53	1009
accuracy			0.69	10000
macro avg	0.77	0.68	0.70	10000
weighted avg	0.78	0.69	0.70	10000

Confusion Matrix:

[[879   0   2 11   1 22   3   1 43 18]

[0 996 38   4   3   1   0   0 92   1]

```
[ 16  34 697  39  16   4  54   5 158   9]
[  6   8  12 645   0  64   1   9 259   6]
[  0   0   1   0 496   0  28   1 365  91]
[ 23   0  14  81   1 331  13   2 416  11]
[ 23   2  33   1  49  23 682   0 134  11]
[  3   9  11   5  19   0   0 804  85  92]
[  6   3  36  25   1  12   8   0 881   2]
[  5   1   3   1  49   1   2  35 460 452]]
```

Evaluating model: tanh_random_init.pkl

Test Accuracy for tanh_random_init.pkl: 21.73%

Classification Report:

	precision	recall	f1-score	support
0	0.19	1.00	0.32	980
1	0.25	0.96	0.39	1135
2	0.00	0.00	0.00	1032
3	0.00	0.00	0.00	1010
4	0.00	0.00	0.00	982
5	0.00	0.00	0.00	892
6	0.15	0.05	0.07	958
7	0.00	0.00	0.00	1028
8	0.34	0.06	0.11	974
9	0.00	0.00	0.00	1009
accuracy			0.22	10000
macro avg	0.09	0.21	0.09	10000
weighted avg	0.09	0.22	0.09	10000

Confusion Matrix:

```
[[ 979    0    0    0    0    0    0    0    1    0]
 [  21 1087    0    0    0    0   16    0   11    0]
 [ 915   73    0    0    0    0   24    0   20    0]
 [ 868  104    0    0    0    0   27    0   11    0]
 [  56  883    0    0    0    0   21    0   22    0]
 [ 793   49    0    0    0    0   31    0   19    0]
 [ 840   60    0    0    0    0   44    0   14    0]
 [  52  947    0    0    0    0   16    0   13    0]
 [ 559  251    0    0    0    0  101    0   63    0]
 [  56  928    0    0    0    0   14    0   11    0]]
```

Evaluating model: tanh_zero_init.pkl

Test Accuracy for tanh_zero_init.pkl: 11.35%

Classification Report:

	precision	recall	f1-score	support
0	0.00	0.00	0.00	980
1	0.11	1.00	0.20	1135
2	0.00	0.00	0.00	1032
3	0.00	0.00	0.00	1010
4	0.00	0.00	0.00	982
5	0.00	0.00	0.00	892
6	0.00	0.00	0.00	958
7	0.00	0.00	0.00	1028
8	0.00	0.00	0.00	974
9	0.00	0.00	0.00	1009
accuracy			0.11	10000
macro avg	0.01	0.10	0.02	10000
weighted avg	0.01	0.11	0.02	10000

Confusion Matrix:

```
[[   0  980    0    0    0    0    0    0    0    0]
 [   0 1135    0    0    0    0    0    0    0    0]
 [   0 1032    0    0    0    0    0    0    0    0]
 [   0 1010    0    0    0    0    0    0    0    0]
 [   0  982    0    0    0    0    0    0    0    0]
 [   0  892    0    0    0    0    0    0    0    0]
 [   0  958    0    0    0    0    0    0    0    0]
 [   0 1028    0    0    0    0    0    0    0    0]
 [   0  974    0    0    0    0    0    0    0    0]
 [   0 1009    0    0    0    0    0    0    0    0]]
```

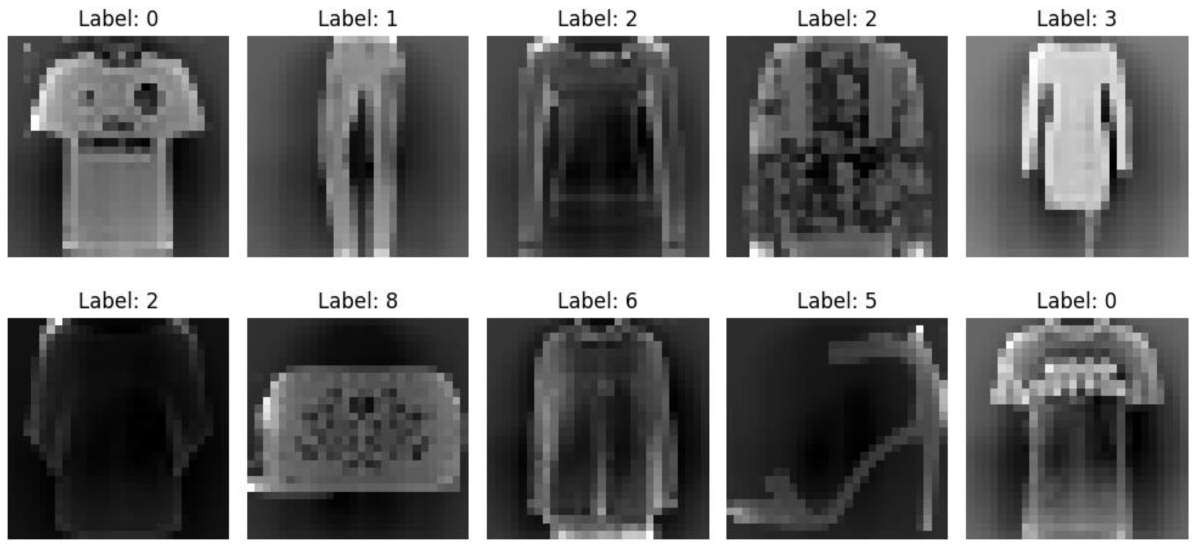
The **Leaky ReLU with Normal Initialization** and **ReLU with Normal Initialization** models performed best, achieving about 96% accuracy, with well-balanced classification results.

Random Initialization and **Zero Initialization** models underperformed significantly, with the zero-initialization model being particularly ineffective (close to random guessing).

Suboptimal combination: Discuss **Sigmoid + Heuristic** and **Tanh + Heuristic** showing poor convergence and higher validation loss, leading to overfitting

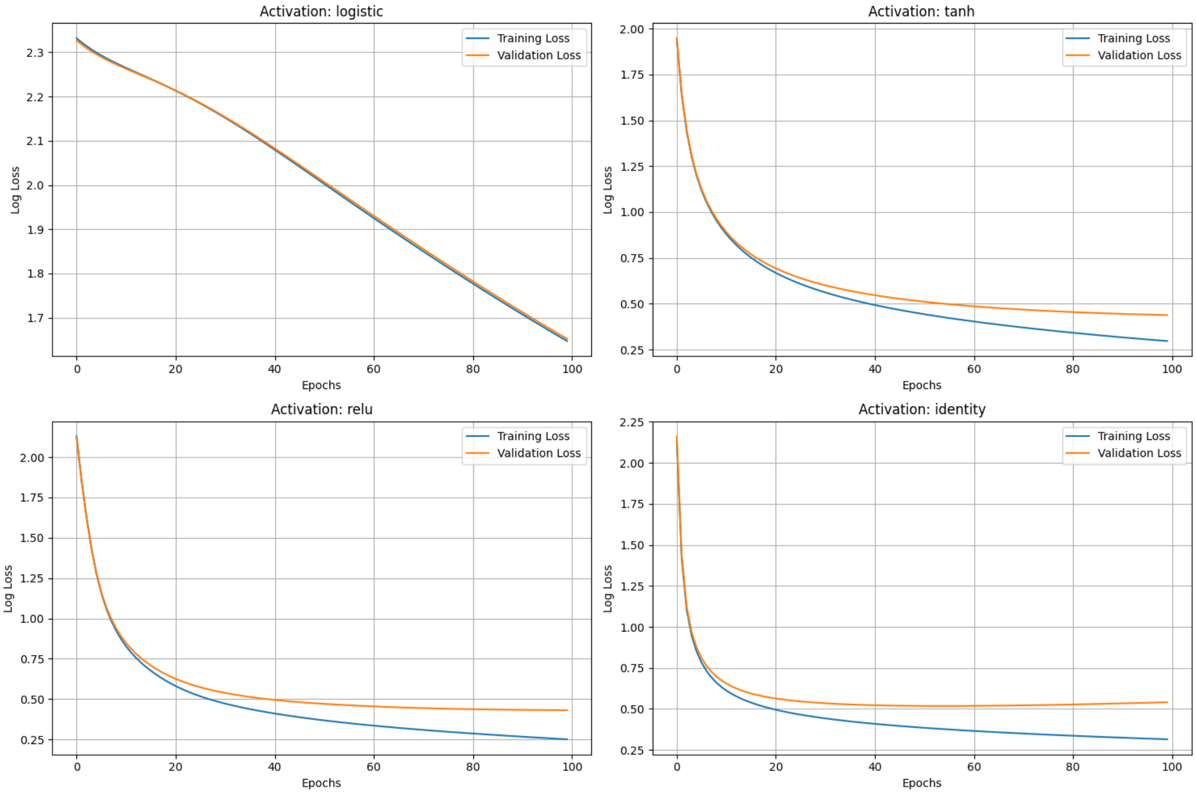
SECTION- C

1)


Label: 0
Label: 1
Label: 2
Label: 2
Label: 3
Label: 2
Label: 8
Label: 6
Label: 5
Label: 0

```python
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
from sklearn.preprocessing import StandardScaler
from sklearn.neural_network import MLPClassifier, MLPRegressor
from sklearn.metrics import log_loss
from sklearn.model_selection import GridSearchCV
import warnings

# Load the dataset
train_data = pd.read_csv('fashion-mnist_train.csv')
test_data = pd.read_csv('fashion-mnist_test.csv')

# Take the first 8000 images from the train data and the first 2000 from the test data
train_data_subset = train_data.iloc[:8000]
test_data_subset = test_data.iloc[:2000]

# Separate features and labels
X_train = train_data_subset.drop('label', axis=1).values
y_train = train_data_subset['label'].values
X_test = test_data_subset.drop('label', axis=1).values
y_test = test_data_subset['label'].values

# Normalize the data
scaler = StandardScaler()
X_train = scaler.fit_transform(X_train)
X_test = scaler.transform(X_test)

# Visualize 10 samples from the test dataset
plt.figure(figsize=(10, 5))
for i in range(10):
    plt.subplot(2, 5, i + 1)
    plt.imshow(X_test[i].reshape(28, 28), cmap='gray')
    plt.title(f'Label: {y_test[i]}')
    plt.axis('off')
plt.tight_layout()
plt.show()
```

2) Training with activation: logistic
Training with activation: tanh
Training with activation: relu
Training with activation: identity



Test Accuracy with activation 'logistic': 0.5490
Test Accuracy with activation 'tanh': 0.8425
Test Accuracy with activation 'relu': 0.8375
Test Accuracy with activation 'identity': 0.8310

Best performing activation function on the test set: tanh

Here tanh performs slightly better than relu and identity.

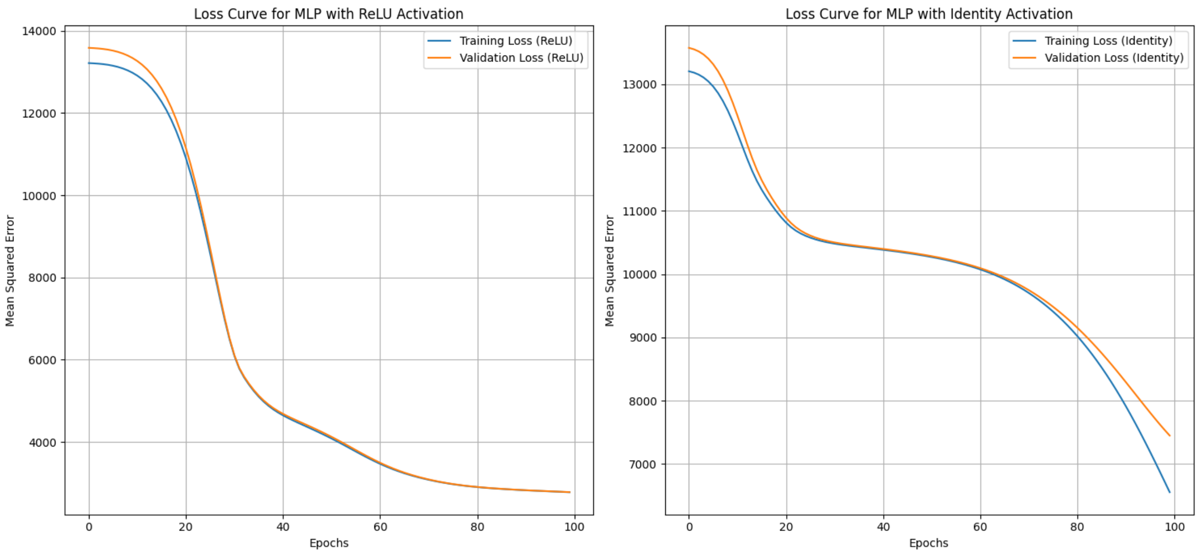
3)

Fitting 3 folds for each of 243 candidates, totalling 729 fits
B0est Hyperparameters: {'alpha': 0.0001, 'batch_size': 32, 'hidden_layer_sizes': (128, 64, 32), 'learning_rate_init': 0.001, 'solver': 'adam'}
Best Cross-validation Accuracy: 0.8209
Test Accuracy with best hyperparameters: 0.8125

4) A)

```python
hidden_layer_sizes = (128, 64, 32, 64, 128)

# Initialize two MLPRegressors with different activations
mlp_relu = MLPRegressor(hidden_layer_sizes=hidden_layer_sizes,
                        activation='relu',
                        solver='adam',
                        learning_rate_init=2e-5,
                        max_iter=1,
                        warm_start=True,  # So we can train one epoch at a time
                        random_state=42,
                        verbose=True)

mlp_identity = MLPRegressor(hidden_layer_sizes=hidden_layer_sizes,
                            activation='identity',
                            solver='adam',
                            learning_rate_init=2e-5,
                            max_iter=1,
                            warm_start=True,
                            random_state=42,
                            verbose=True)

```

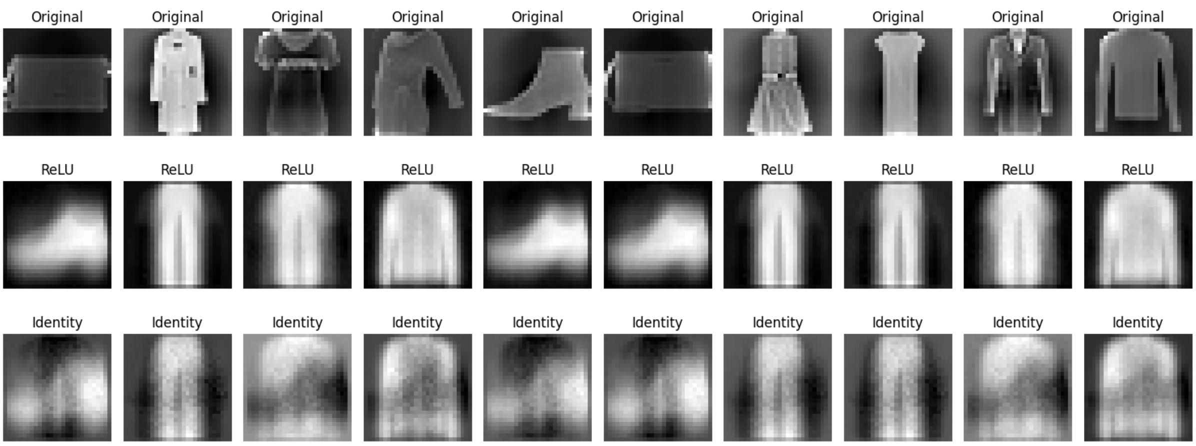
B)



c)

```python

# Training loop for both networks
max_epochs = 100
for epoch in range(max_epochs):
    # Train with ReLU activation
    mlp_relu.fit(X_train, y_train)
    train_pred_relu = mlp_relu.predict(X_train)
    val_pred_relu = mlp_relu.predict(X_test)

    train_loss_relu = mean_squared_error(y_train, train_pred_relu)
    val_loss_relu = mean_squared_error(y_test, val_pred_relu)

    loss_history_relu['train_loss'].append(train_loss_relu)
    loss_history_relu['val_loss'].append(val_loss_relu)

    # Train with Identity activation
    mlp_identity.fit(X_train, y_train)
    train_pred_identity = mlp_identity.predict(X_train)
    val_pred_identity = mlp_identity.predict(X_test)

    train_loss_identity = mean_squared_error(y_train, train_pred_identity)
    val_loss_identity = mean_squared_error(y_test, val_pred_identity)

    loss_history_identity['train_loss'].append(train_loss_identity)
    loss_history_identity['val_loss'].append(val_loss_identity)

    if epoch % 10 == 0:
        print(f"Epoch {epoch+1}/{max_epochs} completed")

```

d)



5) Accuracy for classifier trained on ReLU-extracted features: 0.555

Classification Report for ReLU-extracted features:

	precision	recall	f1-score	support
0	0.57	0.77	0.65	195
1	0.63	0.85	0.73	189
2	0.43	0.53	0.47	205
3	0.52	0.38	0.44	200
4	0.40	0.35	0.38	199
5	0.82	0.31	0.45	202
6	0.32	0.17	0.22	213
7	0.58	0.89	0.70	204
8	0.74	0.71	0.73	188
9	0.57	0.64	0.61	205
accuracy			0.56	2000
macro avg	0.56	0.56	0.54	2000
weighted avg	0.56	0.56	0.53	2000

Accuracy for classifier trained on Identity-extracted features: 0.589

Classification Report for Identity-extracted features:

	precision	recall	f1-score	support
0	0.52	0.72	0.60	195
1	0.62	0.85	0.72	189
2	0.43	0.53	0.47	205
3	0.62	0.48	0.54	200
4	0.47	0.41	0.44	199
5	0.77	0.70	0.74	202
6	0.40	0.22	0.28	213
7	0.70	0.67	0.69	204
8	0.89	0.49	0.63	188
9	0.60	0.84	0.70	205
accuracy			0.59	2000
macro avg	0.60	0.59	0.58	2000
weighted avg	0.60	0.59	0.58	2000

Contrast with Original MLP Classifier from Part 2

Input Dimensionality:

In part 2, the original MLP Classifier directly used the high-dimensional raw pixel data from the images as input, which contained thousands of features (one per pixel).
In this method, the input to the new classifiers is a much smaller feature vector of size a, extracted from the third hidden layer of the pre-trained MLPRegressors. This greatly reduces the dimensionality from the raw image data.

Feature Representation:

The original classifier learned the features from scratch using the raw pixel values. This means it had to learn to identify meaningful patterns across all pixels without any prior information.
The extracted features already capture significant information about the image, as the MLPRegressors trained for image regeneration learned compact, useful representations of each image. These representations, or feature vectors, are already optimized to retain essential characteristics of the image, making them more meaningful and easier for the classifier to process.

Training Complexity:

Training a classifier on high-dimensional data, as in part 2, is challenging due to the "curse of dimensionality." The model must learn useful patterns while filtering out noise from the thousands of pixel values.
By using the extracted feature vectors, the new classifiers in this part focus on fewer, more compact features. This reduces the training complexity and noise, making it easier for the model to converge to a good solution.

Generalization Capability:

The feature extraction process acts as a form of regularization. Because the extracted features are a condensed representation, the new classifiers benefit from a degree of generalization already present in these features, which the MLPRegressor learned while performing the regeneration task.
This method can generalize well, as it leverages learned representations that capture fundamental aspects of the images, rather than overfitting to specific pixel patterns in the raw data.

Reasons for Decent Classifier Performance

Efficient Encoding of Information: The extracted feature vector of size a from the hidden layer contains high-level patterns and structural information, representing each image in a more concise and informative way.

Transfer Learning Benefit: This approach resembles transfer learning, where the pre-trained network's hidden layers serve as feature extractors. The classifiers trained on these features perform well because the network has already learned to capture the core attributes of the images during the regeneration task.

Reduced Dimensionality and Noise: The reduced dimensionality of the input makes the classifier less prone to overfitting and noise, as it relies on the distilled, meaningful features rather than raw pixel data.