E0270: MACHINE LEARNING

Assignment #1

Submitted By:
Rajkamal Ingle(SR No. 22892)
CSA, IISc
April 7, 2024

Contents

1 Problem 1

1.1 Binary Classification with Logistic Regression

The cost function for logistic regression is the binary cross-entropy loss, given by:

$$-\frac{1}{N}\sum_{i=1}^{N}[y^{(i)}\log(\hat{y}^{(i)}) + (1-y^{(i)})\log(1-\hat{y}^{(i)})]$$

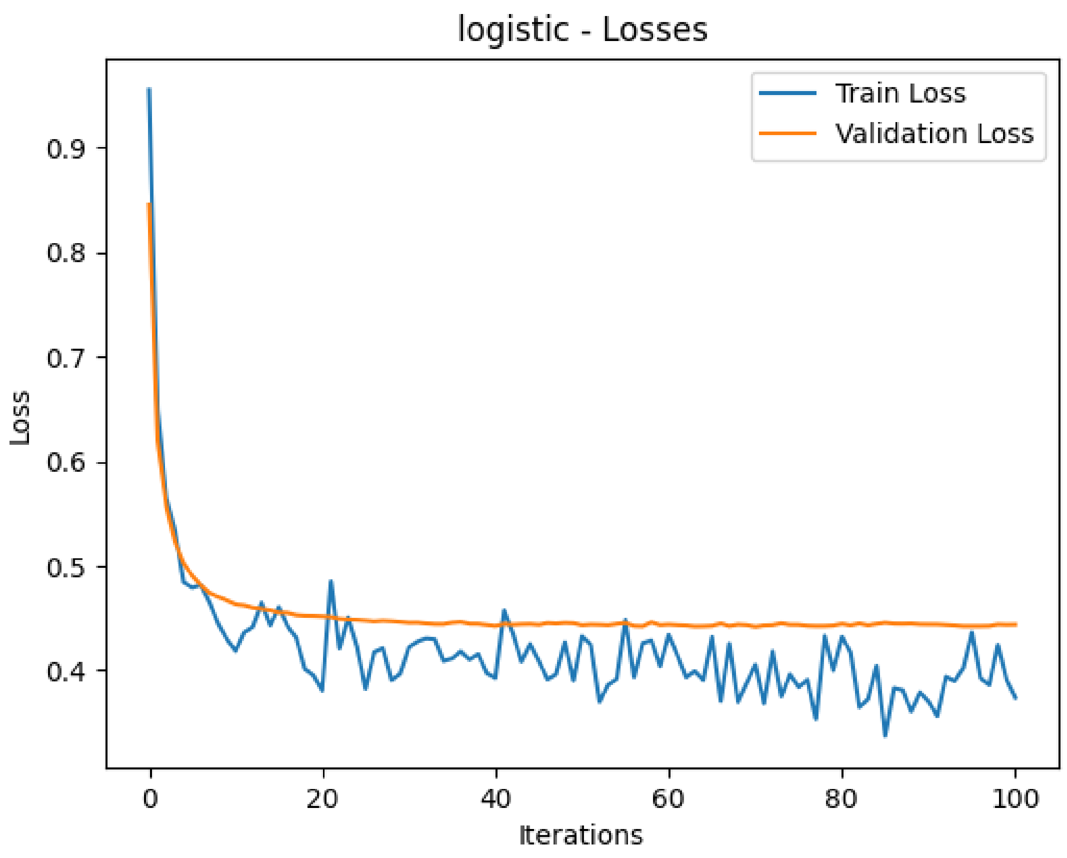

Figure 1: Loss

NOTE: Please ignore the x-axis of the graphs (i.e. the number of iteratios), while constructing the graph, they weren't correctly labelled and due to shortage of time, I couldn't modify them.The number of iterations are explicitly mentioned in the report for each of them.

Batch_size = 512 was chosen simply based on empirical analysis, where it was giving slightly better results.

Rest all parameters were the same as default.

Key Change made to Repository: Introduced Normalization of input data in *get_data*

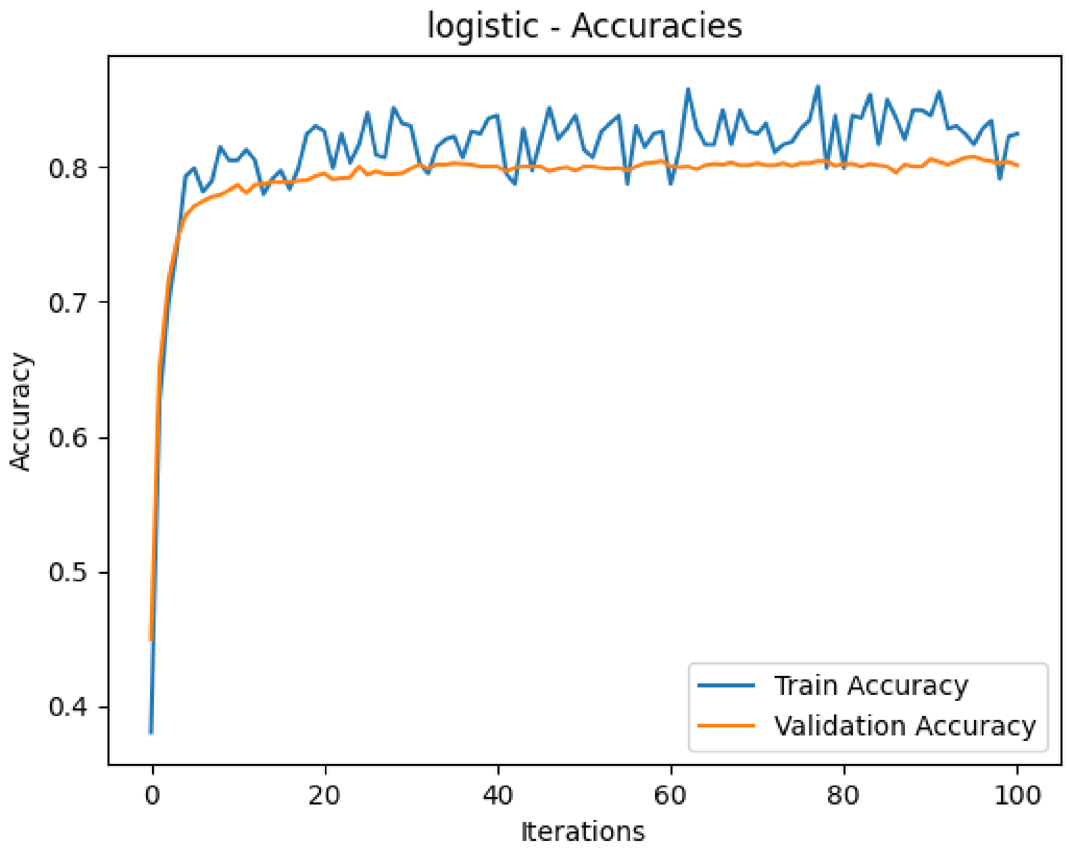

Figure 2: Accuracy

function and updated other necessary files.
Valdiation Accuracy : 80.40 %

1.2 Multi-class Classification with Softmax Regression

For softmax regression, the cross-entropy loss is used, which is:

$$-\frac{1}{N}\sum_{i=1}^{N}\sum_{k=1}^{K} y_k^{(i)} \log(\hat{y}_k^{(i)})$$

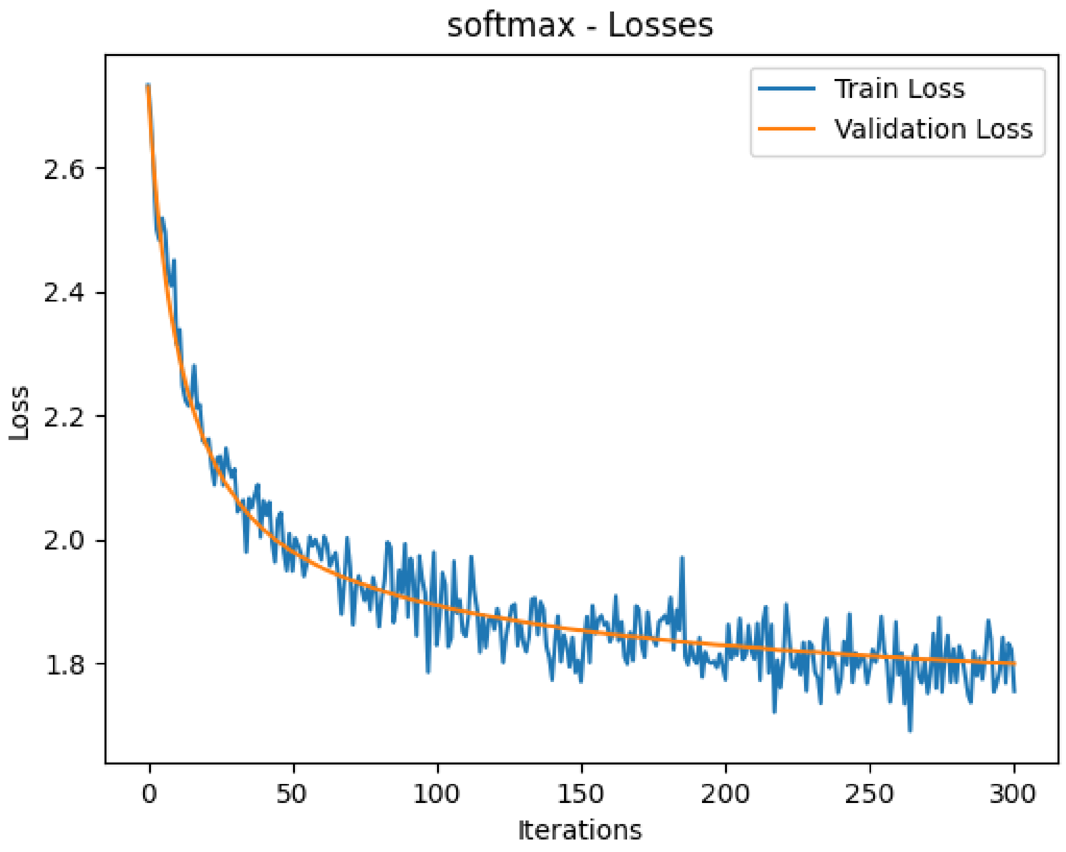

Figure 3: Loss

Just like the Binary Classification problem, batch_size was modified to 512, number of iterations used were 3000 and rest all parameters were used default.
No criteria for stopping the training was introduced.
Validation Accuracy obtained: 38.40 %

Training Strategy:

- Initialization: The training process begins by initializing empty lists to store training and validation losses and accuracies. It also sets up a loop control variable continue training and tracks the number of validation loss increases (number of violations) to potentially implement early stopping.
- Batch Processing: Within each iteration of the training loop, a batch of data is sampled

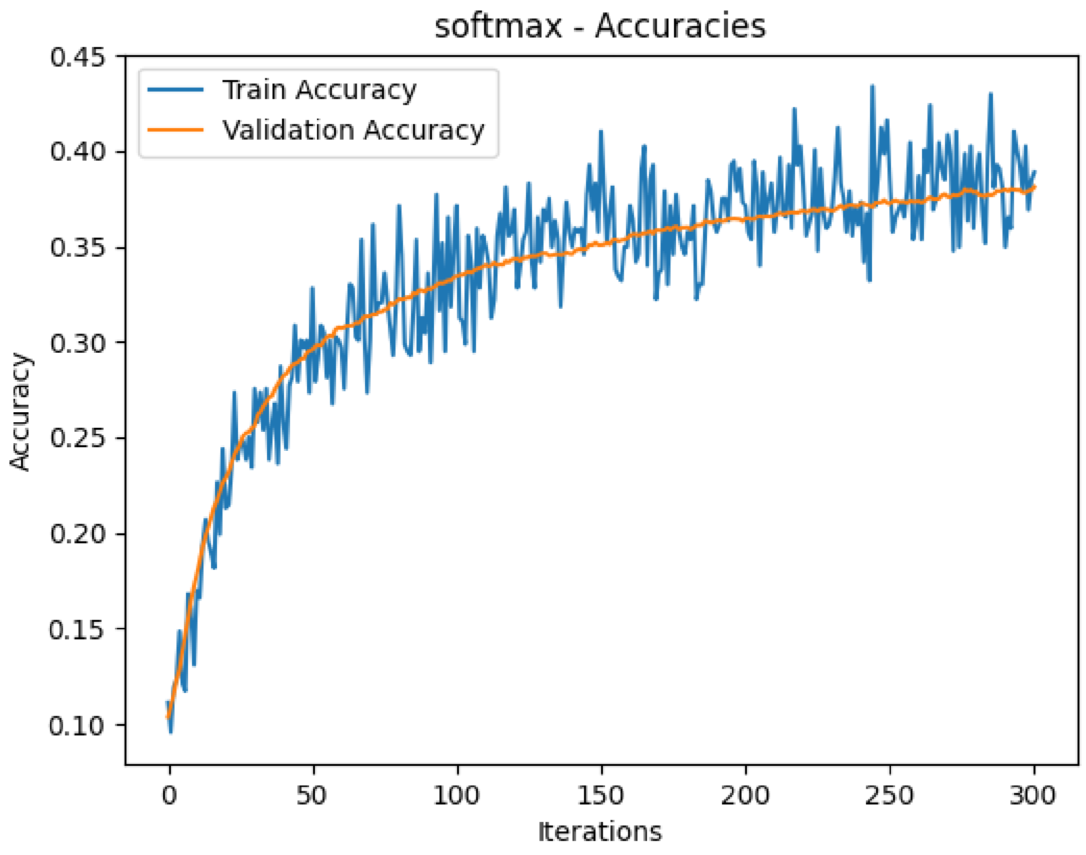

Figure 4: Accuracy

from the training set using the get_data_batch function. This approach allows the model to update its parameters iteratively on smaller subsets of the data, which is efficient and scalable for large datasets.

- Prediction and Loss Calculation: For each batch, the model makes predictions (y_preds) based on the current batch of features (X_batch). The loss is then calculated using these predictions and the true labels (y_batch). The calculate_loss function likely incorporates a loss function suitable for the task (binary or multi-class classification).
- Accuracy Measurement: The accuracy for each batch is calculated by comparing the predictions against the true labels. This provides a measure of the model's performance on the training data during the learning process.
- Gradient Computation and Regularization: The gradient of the loss with respect to the model's parameters (grad_W and grad_b) is computed. L2 regularization is applied to these gradients to prevent overfitting by penalizing large weights.
- Gradient Clipping: The gradients are clipped based on a predefined norm value (grad_norm_clip) to prevent exploding gradients, a common issue in training deep neural networks. This ensures the stability of the training process.
- Model Update: The model parameters are updated using a simple gradient descent step. The learning rate (lr) controls the size of the update step.

2 Problem 2

2.1 Model Architecture

This model architecture was inspired by VGG-11 and its implementation was referred from this article. **Dimension of the Output Vector**

The dimension of the output vector is determined by the final layer of the network used to define the Encoder Class. In the provided code, the final layer of the encoder class, which is a linear layer maps the input to an output embedding of dimension 1000. This output vector size typically corresponds to the output vector dimension in Encoder Class.

Overall Architecture of the Network The provided network architecture is a convolutional neural network (CNN) inspired by the VGG11 model but with some modifications. The network is divided into two main parts: an encoder (Encoder class) and a classifier (Classifier class).

Encoder Architecture:

- Input Specification: The encoder expects an input tensor of shape [N,C,H,W], where N is the batch size, C=3 for RGB image channels, and H and W are the height and width of the images, respectively. The CIFAR-10 dataset we used have 3*32*32 images.
- Convolutional Layers: The encoder comprises a sequence of convolutional layers, each followed by batch normalization and ReLU activation. Specific layer configurations include:
  - Convolutional layer with 64 filters, followed by batch normalization, ReLU activation, and max-pooling.
  - Convolutional layer with 128 filters, followed by the same sequence of batch normalization, ReLU, and max-pooling.
  - Two convolutional layers with 256 filters, each followed by batch normalization and ReLU, with max-pooling after the second convolutional layer.
  - Four convolutional layers with 512 filters, each followed by batch normalization and ReLU, with max-pooling after the fourth convolutional layer.
- Flattening: The output from the last convolutional layer is flattened into a 1D tensor.
- Linear Layer: A fully connected layer that maps the flattened features to a 1000-dimensional vector.

Classifier Architecture:

- Fully Connected Layers: The classifier component consists of three fully connected layers, each followed by batch normalization, ReLU activation, and dropout for regularization, with configurations as follows:
  - First layer: Maps 1000-dimensional input to 4096 units, followed by batch normalization, ReLU, and dropout.
  - Second layer: Maps 4096 units to 2048 units, with batch normalization, ReLU, and dropout.

- Output layer: Maps 2048 units to 10 units, corresponding to the 10 classes. In this way, the classifier outputs a 10-dimensional vector representing the class scores.

Design Choices and Architectural Notes:

- Activation Function: ReLU is used throughout the network due to its effectiveness in non-linear transformation and alleviating the vanishing gradient problem.
- Dropout: Included in the classifier to prevent overfitting by randomly omitting a subset of features during training.
- Batch Normalization: Applied after every convolutional and fully connected layer except the output layer to stabilize learning and improve convergence speed.
- Pooling: Max pooling is used to reduce the spatial dimensions of feature maps, decreasing the number of parameters and computation in the network.

2.2 Training Encoder

Cost Function used: Triplet loss is a loss function used to train a neural network to closely embed features of positive pairs (anchor and positive example) while pushing apart the features of negative pairs (anchor and negative example). The anchor is a reference input, the positive example is a different input of the same class as the anchor, and the negative example is an input from a different class.

$$L_{\text{triplet}} = \max\{\|f(a) - f(p)\|_2^2 - \|f(a) - f(n)\|_2^2 + \alpha, 0\}$$

Where:

- $f(\cdot)$ represents the embedding function (the output of the neural network).
- a represents the anchor example.
- p represents the positive example (similar to the anchor).
- n represents the negative example (dissimilar to the anchor).
- $\|\cdot\|_2$ denotes the Euclidean distance.
- α is a margin hyperparameter that controls the minimum difference between the distances of the anchor-positive pair and the anchor-negative pair.

The objective is to make the distance between the anchor and the positive example smaller than the distance between the anchor and the negative example by at least the margin. This encourages the network to place positive pairs closer together and negative pairs farther apart in the embedding space.

- Batch_size = 512 was chosen simply based on empirical analysis, where it was giving slightly better results.

- **Sampling Strategy**: The same strategy specified in Assignment description of anchor samples, positive sample(same label, different images) and a negative sample (different sample), adhering to the principles of contrastive learning.
  - Random selection plays a crucial role in this strategy, introducing variability and robustness in the training process by presenting the model with a wide range of triplets.
  - This sampling strategy facilitates learning an embedding space where images of the same class (anchor and positive pairs) are closer together, while images from different classes (anchor and negative pairs) are farther apart.

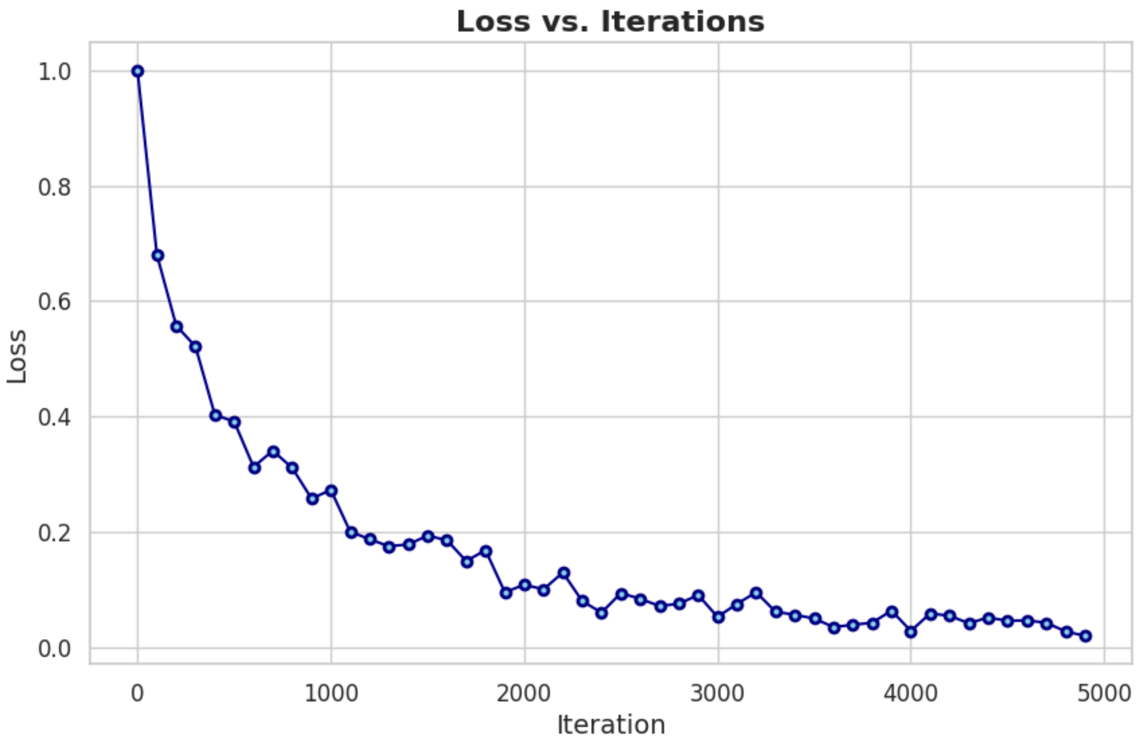

Figure 5: Loss(Cost) vs Iterations while training Encoder

2.3 tSNE plot

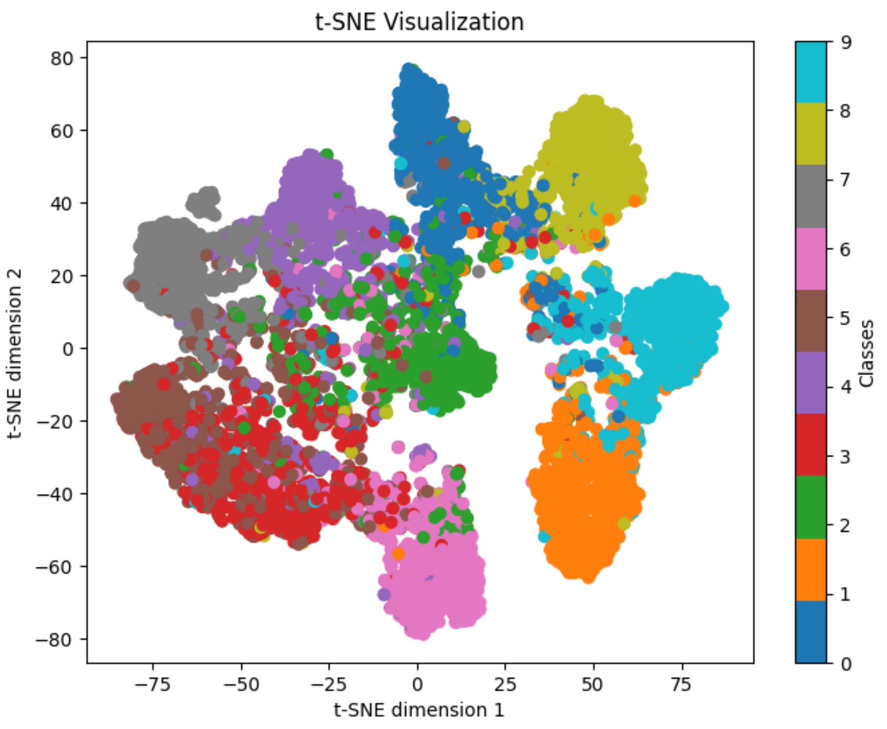

Figure 6: tSNE plot

Observations from the t-SNE plot:

- Cluster Formation: Different colors indicate different classes, and we can see distinct clusters forming, which suggests that instances of the same class are grouping together in the 2D t-SNE space. This implies that t-SNE is effective in capturing the inherent structure of the data and separating different classes.
- Class Separation: Some classes (like the ones in orange and blue) are well-separated from others, indicating good class discriminability for these classes. However, other classes (like the ones in grey and green) seem to intermingle, suggesting similarities between these classes or insufficient separation in the high-dimensional space.
- Data Distribution: The spread of data points shows how each class varies within itself. Some classes form tight clusters (like the pink and yellow clusters), indicating low intra-class variance, while others (like the blue cluster) are more spread out, indicating higher intra-class variance.
- Potential Overlap and Misclassification: Areas where different colors mix or where clusters are close to each other might represent regions where the model may be more likely to misclassify instances. These areas might benefit from further feature engineering or more complex models to better distinguish between classes.

- Anomaly Detection: Isolated points far from their main cluster might be considered anomalies or outliers, which could be interesting for further investigation

2.4 Fine Tuned LinearClassifier(Softmax Regression)

Accuracy on Training Set: 93.28
Validation accuracy : 77.34
Number of Iterations : 5000
z_dimension (Dimension of Embedding/Output Vector from Encoder) : 1000
Batch Size : 512
Number Of Iterations: 5000
Rest all parameters were default.

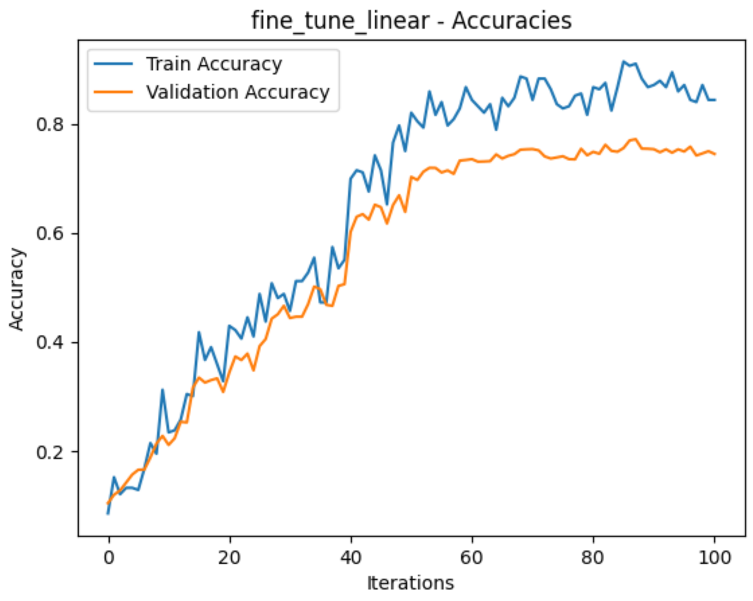

Figure 7: Accuracy of Fine Tuned Linear CLassifier

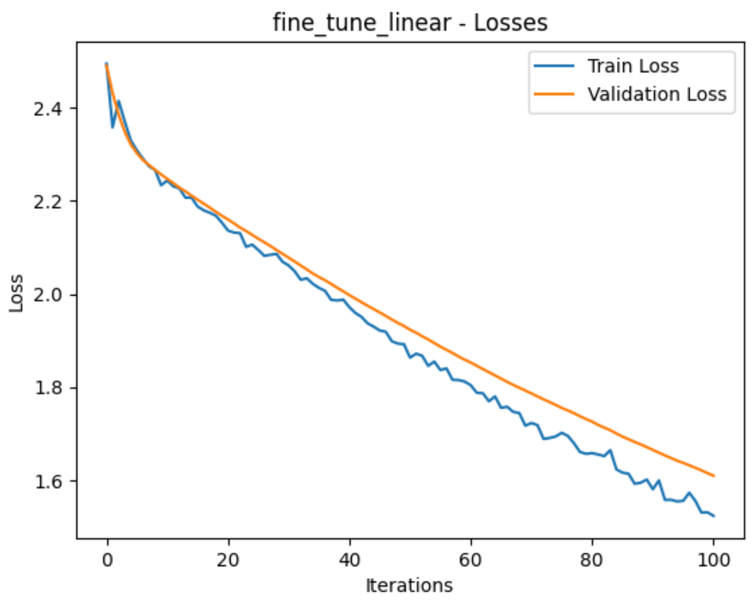

Figure 8: Loss of Fine Tuned Linear Classifier

2.5 Fine Tune Neural Network Classifier

Train Accuracy: 94.80 %
Validation Accuracy: 78.40 %
Number of Iterations : 1000
z_dimension (Dimension of Embedding/Output Vector from Encoder) : 1000
Batch Size : 512
Number Of Iterations: 5000
Rest all parameters were default.

Obtained Slightly better results than fine tune Linear classifier. Fine tune linear was slowly converging while Fine tune Neural network classifier was converging very fast within initial 100 iterations.

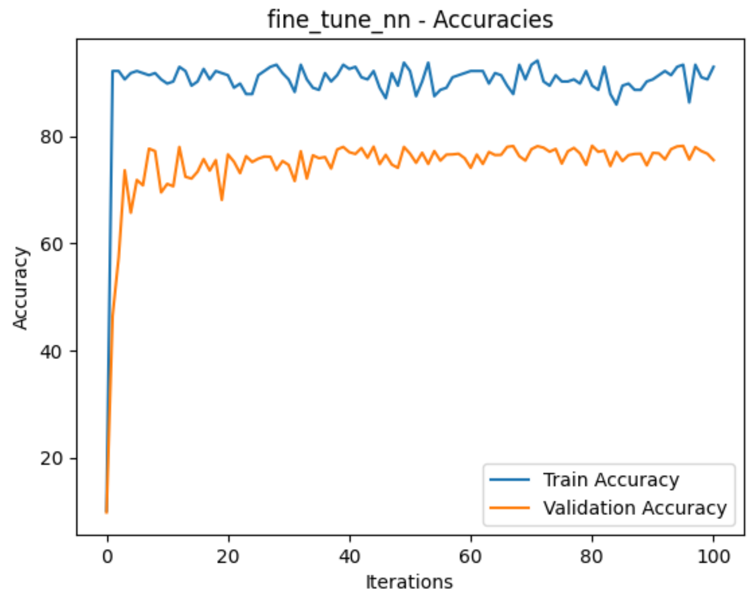

Figure 9: Accuracy of Fine Tuned Neural Network Classifier

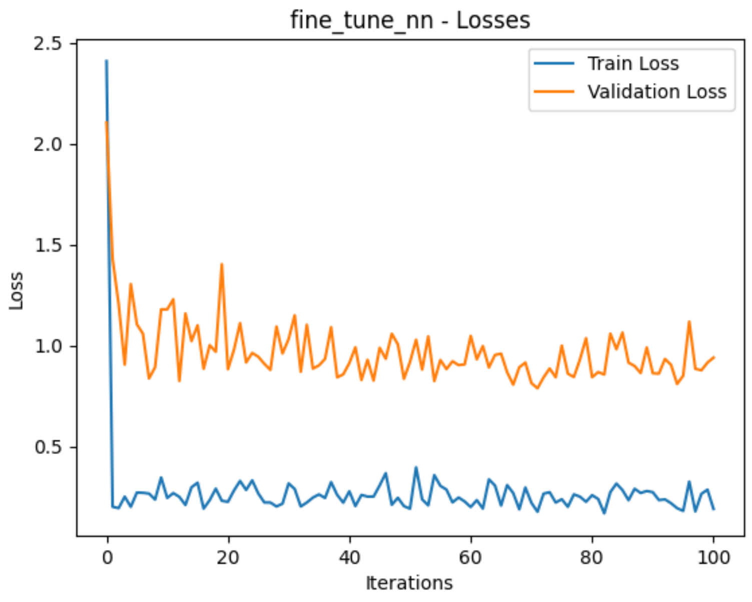

Figure 10: Loss of Fine Tuned Neural Network Classifier

Final Accuracy: 78.02 %