Machine Learning Assignment-1

Pallekonda Naveen Kumar

SR NO : 22915

1 Data

The CIFAR-10 dataset consists of 60000 32x32 colour images in 10 classes, with 6000 images per class. There are 50000 training images and 10000 test images. The classes are completely mutually exclusive.

2 Problem 1 Softmax Regression

2.1 1.1 Logistic Regression

The binary cross-entropy loss function in logistic regression is defined as:

$$\text{Loss} = -\frac{1}{N}\sum_{i=1}^{N}\left(y_i \log(p_i + \epsilon) + (1 - y_i)\log(1 - p_i + \epsilon)\right) \tag{1}$$

where N is the number of samples, y_i is the true label for the ith sample (either 0 or 1), p_i is the predicted probability for the positive class for the ith sample, and ϵ is a small constant added for numerical stability to avoid taking the logarithm of zero. The predicted probability p_i is computed using the sigmoid function:

$$p_i = \frac{1}{1 + e^{-z_i}} \tag{2}$$

where z_i is the linear combination of input features for the ith sample. The linear combination z_i is calculated as:

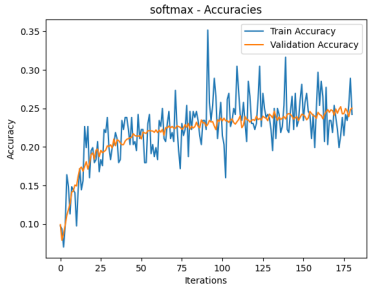
$$z_i = w^T x_i + b \tag{3}$$

where w is the weight vector, x_i is the feature vector for the ith sample, and b is the bias term. Adding ϵ inside the logarithm function ensures numerical stability during computation, preventing issues with taking logarithms of very small values.

As I am using gradient clipping norm and L2 Regularization because without using this I am getting unsual fluctuvating accuracies and losses,After using them I am getting consistently increasing accuracy below in figures you can see it.

2.2 1.2 multi-class Logistic Regression

(a) Softmax Loss for 3000 iterations



(b) Softmax accuracy for 3000 iterations

Softmax Regression is a classification algorithm used to predict probabilities for multiple classes. It extends the Linear Model and is particularly useful for multi-class classification tasks. The Softmax function, denoted as σ, transforms a vector $\mathbf{z}$ into a probability distribution $\mathbf{p}$. For a given input $\mathbf{z}$ and output $\mathbf{p}$, softmax is defined as:

$$p_i = \frac{e^{z_i}}{\sum_{j=1}^{K} e^{z_j}}$$

where K is the number of classes, and z_i is the i-th element of $\mathbf{z}$.

In the forward pass of Softmax Regression, the linear combination $\mathbf{z} = \mathbf{XW}+\mathbf{b}$ is calculated, where $\mathbf{X}$ represents the input data, $\mathbf{W}$ is the weight matrix, and $\mathbf{b}$ is the bias vector. The softmax function is then applied to $\mathbf{z}$ to obtain the predicted probabilities for each class.

The loss in Softmax Regression is computed using the multi-class cross-entropy loss function. For a single sample, the loss L is defined as:

$$\text{Loss} = -\frac{1}{N}\sum_{i=1}^{N}\sum_{k=1}^{K} y_{ik}\log(p_{ik}+\epsilon) \tag{4}$$

Where N represents the number of samples, K signifies the number of classes, y_{ik} denotes the indicator function which equals 1 if sample i belongs to class k, and p_{ik} represents the predicted probability of sample i belonging to class k.

As I stick to default parameters because with changing the parameters there is no significant change in the values.

2.3 Batch Size Selection

The batch size of 256 was chosen for softmax regression to strike a balance between computational efficiency and model stability. This decision considers factors such as training efficiency, memory consumption, and generalization

performance. While smaller batch sizes may aid in convergence by providing noisier updates, larger batch sizes can offer smoother gradients but require more memory. The selected batch size was determined through experimentation to optimize training performance given the available computational resources.

2.4 Criteria for stopping the training

I had taken the stopping criteria,if the accuracy is getting more than 80 percentile I am stopping because as mentioned in the question I am getting more than 80 percent in logistic .

but for the softmax I followed the rigourous method for the softmax as mentioned below The training process utilized early stopping criteria to prevent overfitting and ensure the model's generalization to unseen data. Periodically, typically after every n epochs, the model was assessed on a distinct validation dataset. If the validation loss failed to decrease for a consecutive period of m times, for instance, $m = 10$, training was terminated. This approach safeguards against overfitting by halting training when the model's performance on the validation set stagnates.

2.5 Images Sampled

The 'get data batch' function defined in the provided code efficiently fetches a batch of images and their associated labels from a given dataset. It employs random index sampling without replacement, determined by the specified batch size, ensuring diversity within the batch. This random selection method helps mitigate bias and enhances the model's capacity to generalize during training. The function returns two numpy arrays, 'X batch' containing the batch of images, and 'y batch' containing their corresponding labels, seamlessly facilitating their integration into the training process.

3 Problem 2.1

The Encoder Model is shown in Table 1

Linet5 architecture is used for the encoder part I tried with that and after tuning some hyperparameters.I got 57 percent accuracy.
VGG11 architecture fixing the z_dim (below specified the reason) and after tuning the hyperparameters I got the accuracy of 76 percent.
I want to decrease the training loss of the encoder so that the fine-tuned linear and fine-tuned neural network will get good accuracy. So I upgraded to VGG16 architecture
VGG16 Architecture I got an accuracy of nearly 85.88 accuracy after tuning the parameter, the losses and accuracy here are consistently decreasing and increasing correspondingly

Here I have chosen the dout(out dimension) as 64, as default the size is 32. representing the image of 3*32*32 into 32-dimensional will lose more information

I had tried to use the size of 128 but the loss is increasing so after several trials. I finally chose the dout as the 64, which for me consistently decreased the loss and increased the accuracy.

I used the architectures which are available online from standard websites.

4 Problem 2.2

I had used the Margin Triplet loss during the training of the encoder model. This loss function is in-built in pytorch, and used that for the loss function for the model training and evaluation

$$L(a, p, n) = \max\{d(a_i, p_i) - d(a_i, n_i) + \text{margin}, 0\}$$

$$d(x_i, y_i) = \|x_i - y_i\|_p$$

This is used for measuring a relative similarity between samples. A triplet is composed of an anchor a, positive examples p, and negative examples n, where a, p, and n are input tensors x_1, x_2, and x_3 respectively, and the margin has a value greater than 0.

margin - A non negative margin representing the minimum difference between the positive and negative distances required for the loss to be 0. Larger margins penalize cases where the negative examples are not distant enough from the anchors, relative to the positives by Default:1 (used 1 for implemtation also) **p** – The norm degree for pairwise distance. Default:2(used same 2 for implementation)

The images are sampled to facilitate contrastive learning, where the model is trained to distinguish between similar (from the same class) and dissimilar (from different classes) images.

1. The dataset is shuffled to ensure the generation of random batches without any inherent order from the original dataset.This allows you to produce better results that are more representative.

2. Batches of images are produced from the shuffled dataset, with the batch size determined by the parameter *batch_size*.

3. **Anchor, Positive, and Negative Sampling**: For each image in the batch:Where Anchor represents the current image in the batch. Postive sample is randomly selected from images in the batch sharing the same class label as the anchor.Negative sample is randomly chosen from images in the batch having different class labels from the anchor. This sampling process utilizes numpy functions such as `np.where` for finding indices and `np.random.choice` for random selection.

   The process yields three arrays: X_a (anchor samples), X_p (positive samples), and X_n (negative samples), which are then utilized for training the model.

Layer	Type
1	Conv2d (3, 64, kernel_size=3, padding="same") BatchNorm2d (64) ReLU (inplace=True)
2	Conv2d (64, 64, kernel_size=3, padding="same") BatchNorm2d (64) ReLU (inplace=True) MaxPool2d (kernel_size=2, stride=2)
3	Conv2d (64, 128, kernel_size=3, padding="same") BatchNorm2d (128) ReLU (inplace=True)
4	Conv2d (128, 128, kernel_size=3, padding="same") BatchNorm2d (128) ReLU (inplace=True) MaxPool2d (kernel_size=2, stride=2)
5	Conv2d (128, 256, kernel_size=3, padding="same") BatchNorm2d (256) ReLU (inplace=True)
6	Conv2d (256, 256, kernel_size=3, padding="same") BatchNorm2d (256) ReLU (inplace=True)
7	Conv2d (256, 256, kernel_size=3, padding="same") BatchNorm2d (256) ReLU (inplace=True) MaxPool2d (kernel_size=2, stride=2)
8	Conv2d (256, 512, kernel_size=3, padding="same") BatchNorm2d (512) ReLU (inplace=True)
9	Conv2d (512, 512, kernel_size=3, padding="same") BatchNorm2d (512) ReLU (inplace=True)
10	Conv2d (512, 512, kernel_size=3, padding="same") BatchNorm2d (512) ReLU (inplace=True) MaxPool2d (kernel_size=2, stride=2)
11	Conv2d (512, 512, kernel_size=3, padding="same") BatchNorm2d (512) ReLU (inplace=True)
12	Conv2d (512, 512, kernel_size=3, padding="same") BatchNorm2d (512) ReLU (inplace=True)
13	Conv2d (512, 512, kernel_size=3, padding="same") BatchNorm2d (512) ReLU (inplace=True) MaxPool2d (kernel_size=2, stride=2)

Table 1: Encoder Architecture

1. A data batch was created, consisting of n anchors along with their corresponding positive and negative samples. Anchors were randomly sampled from the dataset.

2. The encoder was utilized to obtain unlearned representations in the z-dimensional space for the anchor, positive, and negative samples.

3. The MarginTriplet loss function, as described previously, was employed to calculate the loss using these three representations.

4. The encoder parameters were updated based on the computed loss using the Adam optimizer.

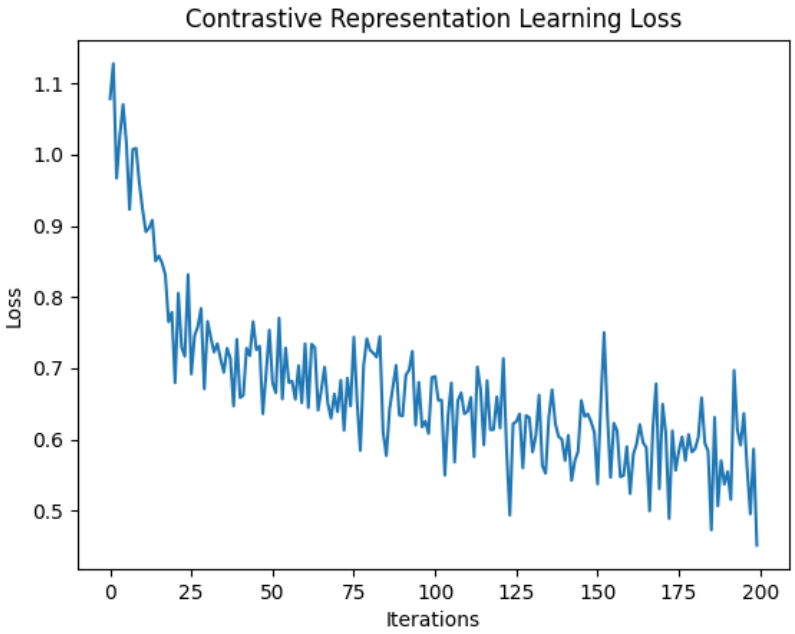


Figure 2: Comparison of Training loss and iterations of Encoder

5 Problem 2.3

Observing the figure 3, it's shows that the clusters exhibit distinct separation, indicating that images belonging to the same class are tightly grouped, while those from different classes are effectively isolated. This suggests that our encoder architecture adeptly encoded the images, leading to robust learning within the model.

Actually the t-SNE plot will also depend on the hyperparameter margin.I keep the margin of the 1.I am getting the good seperating clusters when I am using the margin of 10 and 100 but as far as I do reasearch I came to know that

the margin 1 is good and it is suitable based ont the above triplet loss formulae. so I choose the margin as 1.even the clusters not exactly custered but the 10 classes were seen in the figure 3.

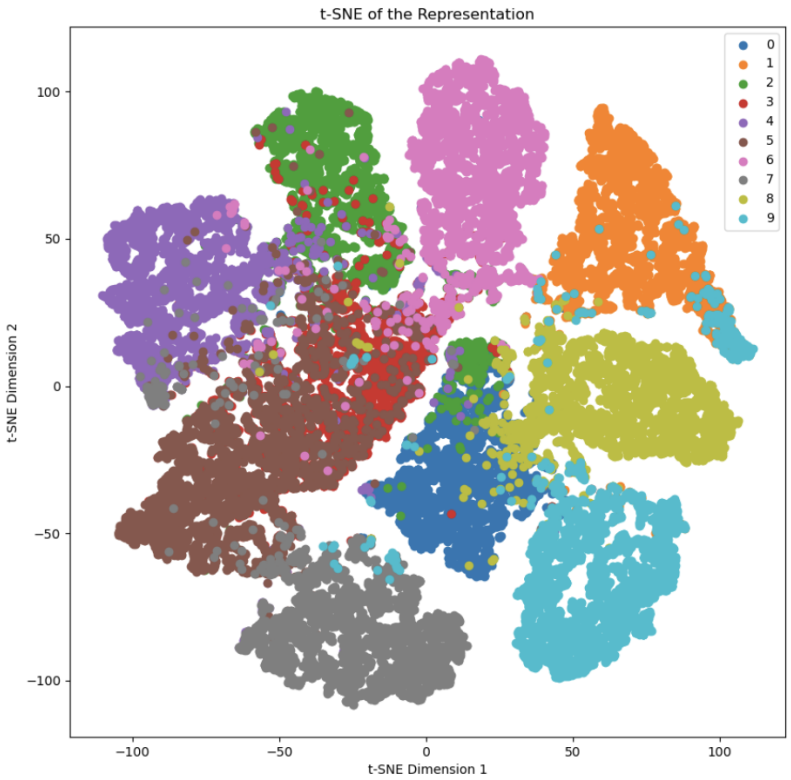


Figure 3: t-SNE plot for X train from encoder

6 Problem 2.4

6.1 Hyperparameters

- Batch size: 2048 (As I observed increasing the batch size increases the accuracy so I tried the different batches and fixed it to 2048)
- Number of iterations: 3000 (As I initially set for 1000 iterations but even at the end of 1000 accuracy increases after gradually increasing a=at 2500

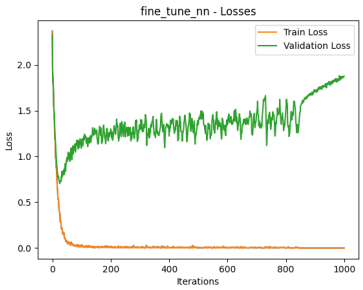
iterations it stabilizes and just want to observe the pattern I extend to 3000 iterations)

- Learning rate: 0.001 (I used the default learning because as increasing the learning rate is overstepping and decreasing it is not showing good results)
- z dimension: 64 (Reason mentioned above)

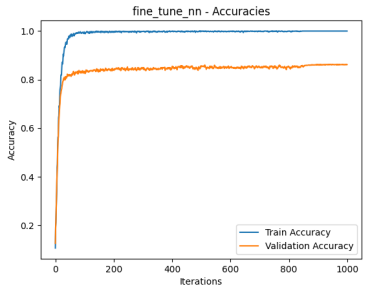
Training Accuracy of 78% and validation accuracy threshold of 52

7 Problem 2.5

In exploring the neural network classifier, I conducted a thorough series of experiments, testing various architectures(As mentioned above at the encoder model how I used the different architectures, the architectures are commented in the model.py of the file with specified name). Ultimately, I found that a configuration featuring two linear layers giving good results.



(a) Comparison of Training loss and validation losses and iterations for fine tune nn



(b) Comparison of Training accuracy and validation accuracy and iterations for fine tune nn

Throughout the training process, I employed fine-tuning, updating both the pretrained Encoder parameters and the newly initialized classifier.

During training, I fine-tuned the neural network classifier using a set of hyperparameters:

- Batch size: 2048 (As I observed increasing the batch size is increasing the accuracy so I tried the diferent batches and fixed to 2048)
- Number of iterations: 5000 (As I intially set for 1000 iterations but even at the end of 1000 accuracy is increasing after gradually increasing a=at 9000 iterations it is overfitting and just want to observe the pattern I extend to 10000 iterations showed in the graph)
- Learning rate: 0.001 (I used the default learning beacuse as increasing the learning rate it is overstepping and decreasing it is not showing good results)

- z dimension: 64 (Reason mentioned above)
- Adam Optimizer is used.

I experimented with various classifier models, each offering its own unique accuracy, with the most successful configurations being...

During fine-tuning for 1000 iterations, the nn validation accuracy was approximately 78%, which notably increased to 86% for the VGG16 encoder and 10000 ierations. The superior performance of fine-tuning the neural network's linear layer using projections from the encoder.

Furthermore, the use of the Adam optimizer facilitated parameter updates for both the linear layer and the encoder.

Iter: 5000, Train Loss: 0.0082, Train Acc: 0.9680, Val Loss: 1.3342, Val Acc: 0.82

Iterations: 10000, Train Loss: 0.0032, Train Acc: 0.9959, Val Loss: 1.7513, Val Acc: 0.86

7.0.1 Comparision of the methods

After comparing both methods, we noticed a big increase in accuracy when we fine-tuned the pretrained Encoder instead of just freezing it. This improvement happened because the Encoder got to learn more about our current training data, making it perform better. Also, fine-tuning lets the Encoder adjust its weights a bit based on the classifier used downstream. So, instead of only depending on the Encoder for representations, fine-tuning is like using both the Encoder and classifier together, which helps boost performance.

8 Problem 2.6

Leaderboard Score : 85.88%

(As I am Trying to increase further more the accuracy I am trying different configurations may be tht plots may not be the same as in report please do consider but overall code and architectures are same)