

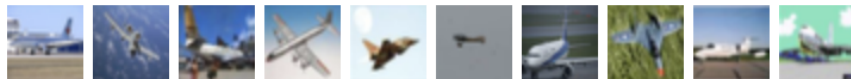
CIFAR10

About the Dataset

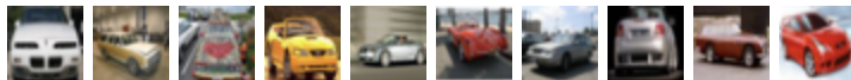
The CIFAR-10 dataset contains 60000 32x32 colour images divided into ten classes, each with 6000 images. There are 50000 photos for training and 10,000 images for testing.

Each of the 10000 photos in the dataset is separated into five training batches and one test batch. The test batch contains exactly 1000 photographs from each class, chosen at random. The remaining photographs are distributed in training batches in a random order, however some batches may contain more images from one class than others. The training batches contain exactly 5000 photos from each class between them.

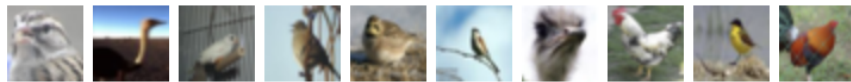
airplane



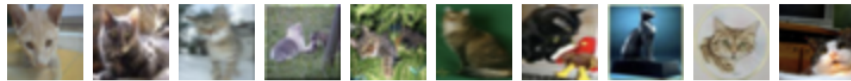
automobile



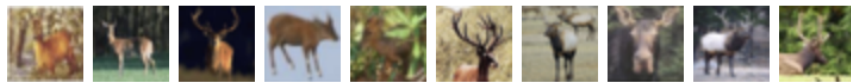
bird



cat



deer



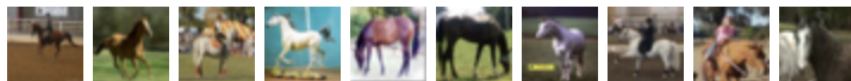
dog



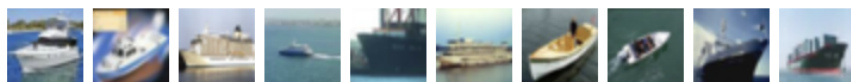
frog



horse



ship



truck



10 random images from each class

Note:

How to Run the program? -- Download the code and run "python <file_name.py>

Python version Used -- Python 3.7.12

Loading the Dataset

The torchvision.datasets module contains Dataset objects for many real-world vision data like CIFAR, COCO.

We load the CIFAR-10 Dataset with the following parameters:

The root defines where the train/test data is stored, train determines whether the dataset is for training or testing, and download=True downloads the data from the internet if it is not accessible at root.

The feature and label transformations are specified by transform and target transform.

Tensors are a type of data structure that looks a lot like arrays and matrices. Tensors are used in PyTorch to encapsulate a model's inputs and outputs, as well as its parameters.

We use data and targets methods to get numpy.ndarray data and labels. The images in CIFAR-10 are of size 3x32x32, i.e. 3-channel color images of 32x32 pixels in size. The train dataset contains 50000 images, the test dataset contains 10000 images.

KMeans Clustering Algorithm

The K-means clustering method aims to generate clusters out of comparable elements. The number of groups is denoted by the letter K. The k-means clustering algorithm attempts to organise comparable things into groups. It compares the objects and divides them into clusters based on their similarity.

The K-means clustering algorithm is a well-known unsupervised clustering algorithm. Based on the distance between data points and the cluster mean, K-means will usually yield K clusters. The knn clustering algorithm, on the other hand, often returns clusters with k samples per cluster.

The method alternates between two steps, starting with an initial set of k means:

We first assign each observation to the cluster with the nearest mean that with the least squared Euclidean distance

$$S_i^{(t)} = \left\{ x_p : \|x_p - m_i^{(t)}\|^2 \leq \|x_p - m_j^{(t)}\|^2 \forall j, 1 \leq j \leq k \right\}$$

Even if it may be assigned to two or more $S(t)$, each x_p is assigned to exactly one of them. Then we perform the update step, recalculate means (centroids) for observations assigned to each cluster.

$$m_i^{(t+1)} = \frac{1}{|S_i^{(t)}|} \sum_{x_j \in S_i^{(t)}} x_j$$

The procedure is frequently described as assigning items to the cluster closest to them based on their distance.

Initialization : The Random Partition technique assigns a cluster to each observation at random before moving on to the update step, computing the initial mean as the centroid of the cluster's randomly allocated points.

Loss Function is given by,

$$\begin{aligned} L &= \sum_{i=1}^m \sum_{k=1}^K 1\{c_i = k\} \|x_i - \mu_k\|^2 \\ &= \sum_{i=1}^m \|x^i - \mu_{c^i}\|^2 \end{aligned}$$

To perform the above functions we have created the following functions

- Initialization : `initiate_clusters`
- Assignment Function : `fit`
- Update Step : `update_centroids, reshape_cluster`

- Convergence : `converged`
- Calculating Loss : `calculate_loss`
- Calculating Accuracy : `calculate_accuracy`

Parameter Tuned: Number of clusters

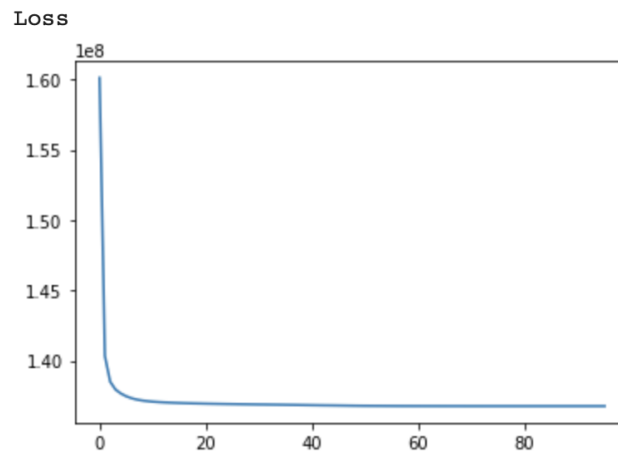
Analysis of the Results

The training accuracy is as follows,

cluster_label	no_occurence_of_label	total_samples_in_cluster	cluster_accuracy
9	1533	4746	0.323
4	908	4547	0.1997
7	817	4649	0.176
5	976	5407	0.181
0	1212	3473	0.349

Accuracy: 0.24128944557174833

The loss decreases with increase in number of iterations

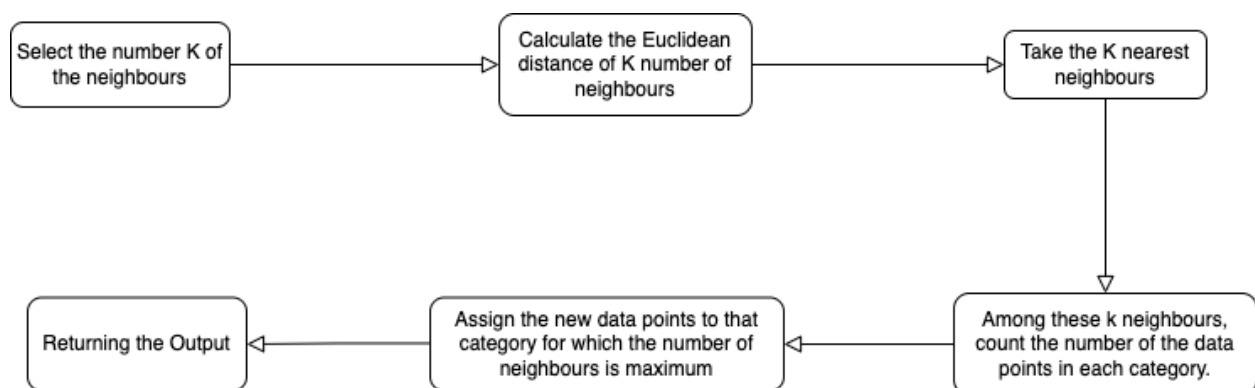


K-NN Algorithm

The K-NN method implies that the new case/data and previous cases are comparable, and places the new case in the category that is the most similar to the previous categories. The K-NN approach saves all available data and categorises new data points depending on how similar they are to the current data. This means that utilising the K-NN approach, fresh data can be swiftly sorted into a well-defined category. Although the K-NN method can be used for both regression and classification, it is most typically employed for classification. The K-NN algorithm is a non-parametric algorithm, meaning it doesn't make any assumptions about the data.

A projected class for the new data point is the majority of K Nearest Neighbors in the training dataset were allocated a class label.

There is overfitting of data/high variance at low K levels. As a result, the test error is significant while the training error is low. Because the nearest neighbour to that point is that point itself, the error is always zero in train data when $K=1$. As a result, with smaller K values, even while training error is minimal, test error is considerable. This is referred to as overfitting. The test error decreases when the value for K is increased. However, after a certain K value, bias/underfitting occurs, and test error increases. So we may say that the test data error is high at first (due to variation), then it drops and stabilises, and with a higher K value, it rises again. When the test error stabilises and is low, the K value is regarded to be optimal.



Distance computation is a method of comparing two photographs that may be quantified. Pixel-wise disparities are frequently used to calculate distance. Distance is

computed "pixel by pixel" in this method, and then the elements of the appropriate matrix are combined together. Photos that are very similar will have lower values, whereas images that are very distinct will have higher values. Furthermore, photos that are identical have a 0 distance. The choice of distance that was used in the assignment is the L2 distance:

$$d_2(I_1, I_2) = \sqrt{\sum_p (I_1^p - I_2^p)^2}$$

The no-loop computation is a fully-vectorized implementation of the L2 distance. This eliminates the loops entirely, replacing a cell by cell (or row by row) implementation into a single matrix solution.

Performing Cross-Validation :

Cross-validation is a good way to figure out the best value of k, or the best number of neighbours who will do the "majority vote" to identify the class. The accuracies will be computed with respect to an array of k choices once we divide the training set into 5 folds. We may then arrange the accuracies and find the best k value. After that, we run cross-validation. As a result, we'll run the k-NN algorithm num folds times for each k value. All but one fold will be used as training data, and the final one will be used as validation data. After that, we run cross-validation. As a result, we'll run the k-NN algorithm num folds times for each k value. All but one fold will be used as training data, and the final one will be used as validation data. The accuracies of each fold, as well as all values of k, are then stored in the k to accuracies dictionary.

Parameters Tuned : Number of neighbours

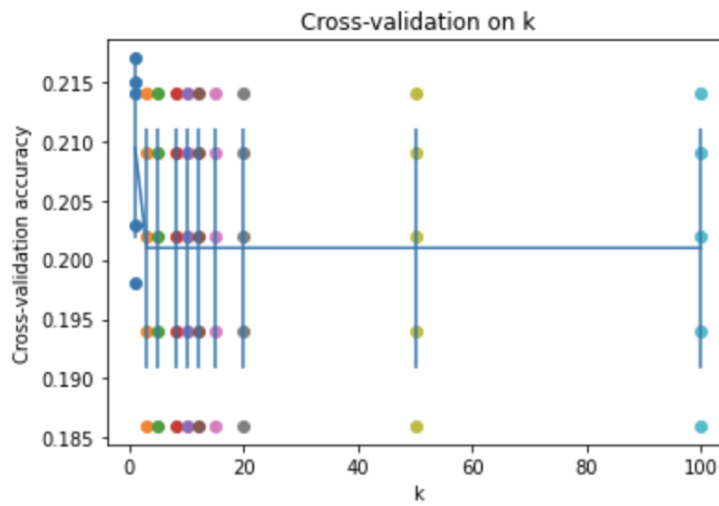
Values of the parameters : [1, 3, 5, 8, 10, 12, 15, 20, 50, 100]

Analysis of the Results:

```

(1, [0.203, 0.217, 0.214, 0.198, 0.215])
(3, [0.202, 0.194, 0.214, 0.186, 0.209])
(5, [0.202, 0.194, 0.214, 0.186, 0.209])
(8, [0.202, 0.194, 0.214, 0.186, 0.209])
(10, [0.202, 0.194, 0.214, 0.186, 0.209])
(12, [0.202, 0.194, 0.214, 0.186, 0.209])
(15, [0.202, 0.194, 0.214, 0.186, 0.209])
(20, [0.202, 0.194, 0.214, 0.186, 0.209])
(50, [0.202, 0.194, 0.214, 0.186, 0.209])
(100, [0.202, 0.194, 0.214, 0.186, 0.209])

```



Best parameter setting: k=10, Accuracy on Test Dataset = 0.282000

Softmax Classifier:

The cross-entropy loss is used by the Softmax classifier. The Softmax classifier derives its name from the softmax function, which is used to squash raw class scores into normalized positive values that total to one in order to apply the cross-entropy loss.

The Softmax function turns an N-dimensional vector of real values into a range of real numbers (0,1). Instead of a single maximum value, the softmax function returns a probability distribution, making it suited for probabilistic interpretation in classification applications. The softmax function is given by

$$p_i = \frac{e^{a_i}}{\sum_{k=1}^N e^{a_k}}$$

However, with computer calculations, exponential quantities, such as e^{1000} , can readily burst to an infinitely huge value. The max method is used to solve this problem by limiting each exponential value to one. For implementation, the maximum of the exponential values is subtracted from the total.

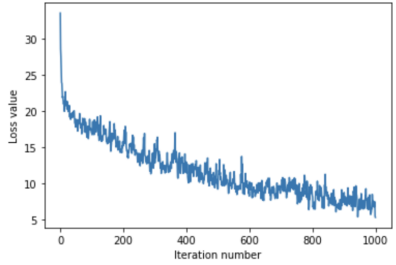
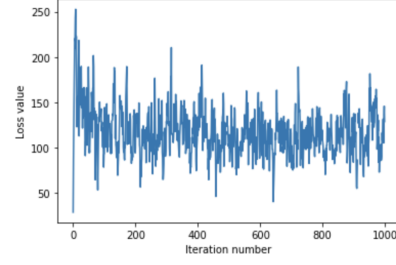
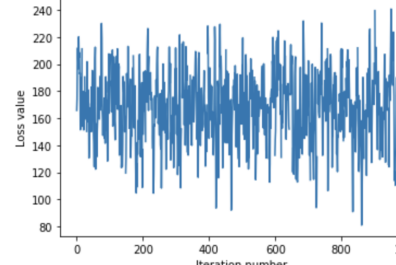
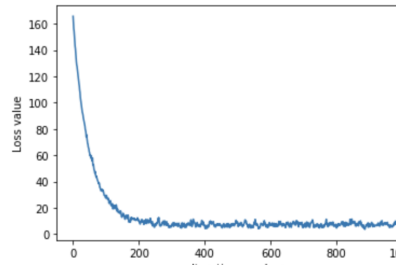
The cross entropy loss is defined as,

$$L = -\frac{1}{N} \sum_{i=1}^N \log(p_i^{y_i})$$

The derivatives of the softmax function and cross entropy loss are required to compute the analytical equation for the gradient of the loss function. y_{oh} is the one-hot representation of the labels in the derivative of the cross entropy loss. A regularisation term $R(W)$ multiplied by the regularisation parameter λ is also included in the implementation. By computing the gradient w.r.t. a randomly selected batch from the training set, mini-batch Stochastic Gradient Descent (SGD) is utilised to minimise the loss. This strategy is faster than computing the gradient across the entire training set before doing each update.

Parameters tuned: Learning Rate, Regularisation

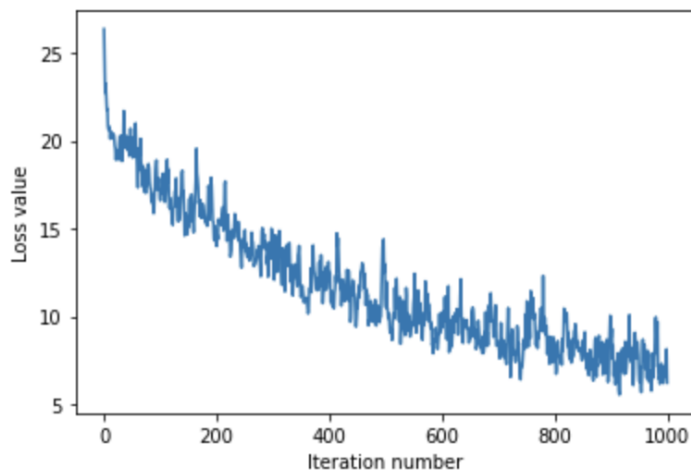
Values:

Learning Rate	Regularisation	Accuracy	Loss
1e-6	1e3	0.252900	
1e-5	1e3	0.100200	
1e-5	1e4	0.100100	
1e-6	1e4	0.171400	

Analysis of the Best Parameters:

Gradient Check:

```
numerical: -13.829246 analytic: -13.829246, relative error: 1.559290e-09  
numerical: -18.088293 analytic: -18.088293, relative error: 2.534986e-10  
numerical: -5.664483 analytic: -5.664483, relative error: 4.024907e-08  
numerical: -11.063876 analytic: -11.063876, relative error: 2.703260e-08  
numerical: -5.468891 analytic: -5.468891, relative error: 5.737546e-08  
numerical: -15.730250 analytic: -15.730250, relative error: 3.138167e-11  
numerical: -4.585340 analytic: -4.585341, relative error: 5.775083e-08  
numerical: -6.284711 analytic: -6.284712, relative error: 3.731485e-08  
numerical: 29.759698 analytic: 29.759698, relative error: 1.177262e-08  
numerical: -6.523219 analytic: -6.523220, relative error: 4.058500e-08
```



Accuracy of the Softmax classifier on the test set: 0.274200

Best Values for the parameters:

Learning Rate = $1e-6$, Regularisation = $1e3$, Accuracy= 0.274200