Deep Learning for Computer Vision (CS776A)
Indian Institute of Technology Kanpur

ASSIGNMENT

1

Member Names: Pradeep Chalotra, Pulkit Sharma
Member Emails: pchalotra22@iitk.ac.in, pulkits22@iitk.ac.in
Member Roll Numbers: 22111045, 22111048
Date: February 3, 2023

---

1 The CIFAR-10 dataset:

The CIFAR-10 dataset is a set of images that is often used to train computer vision and machine learning algorithms.This dataset is a subset of Tiny Images dataset consists of 10,000 test images and 50,000 32x32 color training images that have been labeled into 10 categories.Each training batch contains 10,000 images and the test batch also contains 10,000 images.

2 Data Augmentation:

Image data augmentation is the process of creating new, altered versions of the original images in order to broaden the image dataset.Images is nothing but a 2-dimensional array of numbers to a computer.These numbers represent pixel values, which you can change in a variety of ways to create brand-new, enhanced images.
Below are a few data augmentation techniques we used in this assignment.

1. **Image Enhancement**.
2. **Posterization of Image.**
3. **Random Rotation.**
4. **Contrast Horizontal flipping.**

After data augmentation on the original dataset, the enhanced dataset has a total of 100,000 samples.

3 Feature Extraction:

Feature Extraction is the process of turning raw data into processable numerical features while keeping the original data set's content unchanged.
In this question, for Feature Extraction we use feature_extractor.py file provided with the assignment.In this file class "BBResNet18" that implements a feature extractor using a pre-trained ResNet18 model from "torchvision.models".

4 Implementation of MLP:

To implement MLP for the classification of CIFAR-10 images, the following steps are followed:

- we define the MLP architecture, which consists of two hidden layers with 64 neurons each, and ReLU activation functions at hidden layers and softmax function at output layer. The input layer has the same number of neurons as the size of the 1D vectors and the output layer has 10 neurons, corresponding to the 10 classes in the CIFAR-10 dataset.
- Then initialize the weights and biases of the network randomly.
- Train the network by feeding the 1D vectors through the network, computing the error between the predicted outputs and the true labels, and updating the weights and biases using an optimization algorithm gradient descent(GD).
- Evaluate the performance of the network on the test set, by measuring accuracy.
- Fine-tune the hyperparameters of the network, such as the learning rate, number of epochs, and batch size, to improve its performance on the test set.

5 Model Structure of MLP:

```
1 class MLP():#The MLP class initializes a double hidden-layer perceptron with 64
      neurons at the hidden layers and 10 neurons in output layer. It has
     necessary functions to run the forward pass and backward pass which can be
     used to train the MLP model.
2 def __init__(self, input_size, hidden_size, output_size): #Randomly initialize
     weights and biases.
3 def relu(self, x): # The function returns the input value if it is positive,
     and 0 if it is negative.
4 def softmax(self, x): #The softmax function is to map the input vector of real
     numbers to a probability distribution over the classess.The function works
     by applying the exponential function to each element of the input vector and
      then divide each value by total of the exponential.
5 def forward(self, X): #This function  is used to compute the output for a given
      input data. It is the first step in the prediction process, and is also
     used in the training of the network.In a forward pass, the input data is
     first transformed by applying a series of linear transformations (matrix
     multiplications) and non-linear activation functions to produce the output.
6 def backward(self, X , y ,y_pred ,learning_rate): #The backward function starts
      at the output layer and works backwards through the hidden layers to the
     input layer. The gradients are computed by applying the chain rule of
     calculus to the loss function and the transformations performed by each
     layer in the forward pass and update the weights and biases.
7 def cross_entropy_loss(self, y, y_pred): #This function calculate loss and pass
      to softmax function
```

6 Loss Function:

We use cross entropy as a Loss Function which is used in machine learning for classification problem. It measures the difference between the predicted probability distribution and the true distribution. The cross entropy loss is defined as:

$$CE(p, q) = -\sum i(p_i * log(q_i)) \tag{1}$$

where p is the true distribution and q is the predicted distribution. The loss is calculated for each class in the classification problem, and the sum of the losses is taken over all classes. The goal in training a model is to minimize the cross entropy loss, which can be achieved through gradient descent optimization.

7 Activation Function:

An activation function is a mathematical function used in artificial neural networks to introduce non-linearity into the output of a node or neuron. Activation functions are applied element-wise to the output of each neuron before it is passed on as input to the next layer. For our model we will use the Rectified-Linear Unit (ReLU) activation function. The Relu function returns the input value if it is positive, and 0 if it is negative.

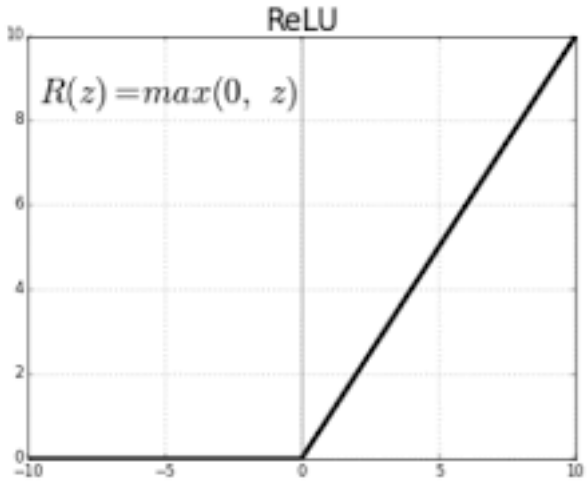

Figure 1: Rectified-Linear Unit

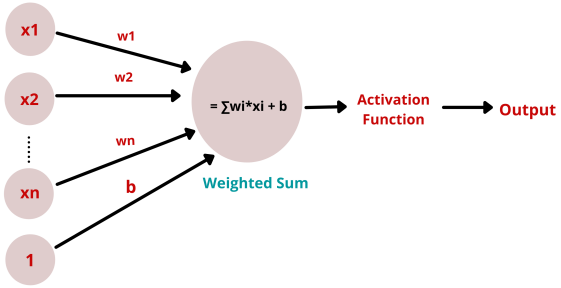

Figure 2: Activation Function

8 Back Propagation:

Backpropagation is an algorithm for training artificial neural networks. Backpropagation works by computing the gradient of the loss with respect to each weights in the network.The gradients is then used to update the weights in an iterative manner. This process continues until the network has learned the relationships between the inputs and outputs of the training data.

8.1 Derivation of Gradient

The derivative of softmax activation has been computed as follows:

$$\begin{aligned}
\frac{dL}{dZ_3^i} &= \frac{d}{dZ_3^i}\left[-\sum_{k=1}^{c} Y^k \log\left(A_3^k\right)\right] = -\sum_{k=1}^{C} Y^k \frac{d\left(\log\left(A_3^k\right)\right)}{dZ_3^i} \\
&= -\sum_{k=1}^{C} Y^k \frac{d\left(\log\left(A_3^k\right)\right)}{dA_3^k} \cdot \frac{dA_3^k}{dZ_3^i} \\
&= -\sum_{k=1}^{C} \frac{Y^k}{A_3^k} \cdot \frac{dA_3^k}{dZ_3^i} \\
&= -\left[\frac{Y^i}{A_3^i} \cdot \frac{dA_3^i}{dZ_3^i} + \sum_{k=1,k\neq i}^{C} \frac{Y^k}{A_3^k} \frac{dA_3^k}{dZ_3^i}\right] \\
&= -\frac{Y^i}{A_3^i} \cdot A_3^i\left(1 - A_3^i\right) - \sum_{k=1,k\neq i}^{C} \frac{Y^k}{A_3^k} \cdot \left(A_3^k A_3^i\right) \\
&= -Y^i + Y^i A_3^i + \sum_{k=1,k\neq i}^{C} Y^k A_3^i \\
&= A_3^i\left(Y^i + \sum_{k=1,k\neq i}^{C} Y^k\right) - Y^i = A_3^i \cdot \sum_{k=1}^{C} Y^k - Y^i \\
&= A_3^i \cdot 1 - Y^i, \text{since} \sum_{k=1}^{C} Y^k = 1 \\
&= A_3^i - Y^i = A_3 - Y
\end{aligned}$$

The gradient of ReLU function is 1 if the input is positive else zero.

The gradient of loss with respect to the weights and biases of the network layers can be derived as follows

Calculating the gradients between the output layer and second hidden layer

$$dZ_3 = \frac{\partial L}{\partial Z_3} = dZ_3 = A_3 - Y$$
$$dW_3 = \frac{\partial L}{\partial W_3} = \frac{\partial L}{\partial Z_3}\frac{\partial Z_3}{\partial W_3} = \frac{1}{m}A_2^T dZ_3$$
$$dB_3 = \frac{\partial L}{\partial B_3} = \frac{\partial L}{\partial Z_3}\frac{\partial Z_3}{\partial B_3} = \frac{1}{m}np \cdot \text{sum}\,(dZ_3,\ \text{axis}\ = 0)$$
$$g_1' = \frac{\partial A_2}{\partial Z_2} = 1 \text{ if } A_2^i > 0 \text{ otherwise } 0$$

Calculating the gradients between second hidden layer and first hidden layer

$$dZ_2 = \frac{\partial L}{\partial Z_2} = \frac{\partial L}{\partial Z_3}\frac{\partial Z_3}{\partial A_2}\frac{\partial A_2}{\partial Z_2} = dZ_3 W_3^T * g_1'$$
$$dW_2 = \frac{\partial L}{\partial W_2} = \frac{\partial L}{\partial Z_3}\frac{\partial Z_3}{\partial A_2}\frac{\partial A_2}{\partial Z_2}\frac{\partial Z_2}{\partial W_2} = \frac{1}{m}A_1^T dZ_2$$
$$dB_2 = \frac{\partial L}{\partial B_2} = \frac{\partial L}{\partial Z_3}\frac{\partial Z_3}{\partial A_2}\frac{\partial A_2}{\partial Z_2}\frac{\partial Z_2}{\partial B_2} = \frac{1}{m}\,\text{np.sum}\,(dZ_2, axis = 0)$$
$$g_0' = \frac{\partial A_1}{\partial Z_1} = 1 \text{ if } A_1^i > 0 \text{ otherwise } 0$$

Calculating the gradients between first hidden layer and input layer

$$dZ_1 = \frac{\partial L}{\partial Z_1} = \frac{\partial L}{\partial Z_3}\frac{\partial Z_3}{\partial A_2}\frac{\partial A_2}{\partial Z_2}\frac{\partial Z_2}{\partial A_1}\frac{\partial A_1}{\partial Z_1} = dZ_2 W_2^T * g_0$$
$$dW_1 = \frac{\partial L}{\partial W_1} = \frac{\partial L}{\partial Z_3}\frac{\partial Z_3}{\partial A_2}\frac{\partial A_2}{\partial Z_2}\frac{\partial Z_2}{\partial A_1}\frac{\partial A_1}{\partial Z_1}\frac{\partial Z_1}{\partial W_1} = \frac{1}{m}x_0^T dZ_1$$
$$dB_1 = \frac{\partial L}{\partial B_1} = \frac{\partial L}{\partial Z_3}\frac{\partial Z_3}{\partial A_2}\frac{\partial A_2}{\partial Z_2}\frac{\partial Z_2}{\partial A_1}\frac{\partial A_1}{\partial Z_1}\frac{\partial Z_1}{\partial B_1} = \frac{1}{m}\,\text{np.sum}\,(dZ_1, axis = 0)$$

We use the Gradient Descent for the optimization and equation of gradient descent are as follows:

$$w_i = w_i - \eta \cdot dw_i \tag{2}$$
$$b_i = b_i - \eta \cdot db_i \tag{3}$$

where eta is the learning rate which impacts the size of step taken in the opposite direction of gradient.

9 Training:

Training of MLP model on original and augmented dataset.

- **Learning Rate**: 0.1
- Number of **epochs**: 1000
- **Batch size**: 1000
- Initializes the weights with "Xavier Initialization" for the layers of Multi-level Perceptron (MLP) using random normal distribution with mean 0 and standard deviation (np.sqrt(2/input_size)).

10 Evaluation of performance:

10.1 Table for test accuracy on top five classification of classifiers:

Model	Original Training Set	Augmented Training Set
SVM Classifier	90.86%	88.16%
KNN Classifier	84.76%	84.08%
Logistic Regression Classifier	89.89%	87.14%
Decision Tree Classifier	56.93%	56.31%

10.2 Table for accuracy on top five classification of MLP model:

Model	Test Accuracy
Original Dataset	90.84%
Augmented Dataset	89.78%

11 References:

1. The CIFAR-10 dataset: https://www.cs.toronto.edu/~kriz/cifar.html.
2. Backpropagation: https://towardsdatascience.com/understanding-backpropagation-algorithm-
3. Implementation of Neural Network with SoftMax in Python from scratch : - https://www.adeveloperdiary.com/data-science/deep-learning/neural-network-with-softmax-in-python/.
4. Image Transformation : https://shrishailsgajbhar.github.io/post/Image-Processing-Image-Rot
5. Weight Initialization Techniques in Neural Networks: https://towardsdatascience.com/weight-initialization-techniques-in-neural-networks-26c649eb3b78
6. Normalization Techniques: https://developers.google.com/machine-learning/data-prep/transform/normalization