BLG 506E Computer Vision 2022-2023 Spring Assignment III

Resul Dagdanov - 511211135

***Abstract*— This assignment focuses on implementing and using a two-layer neural network classifier for image classification. The development and use of this neural network classifier along with the evaluation of the classifier's performance using numeric gradient checking are the main goals of this assignment. The neural network classifier is put into practice and used to categorize the CIFAR10 public dataset. Implementing vectorized gradient code and vectorized forward and backward propagation techniques, evaluating it against basic implementations, and utilizing numeric gradient testing to verify the code's accuracy is part of the assignment. The code is developed in the two-layer-net.py file and the guidelines are in the two-layer-net.ipynb which is given a notebook in order to complete the assignment. Also the instructions for testing and evaluating the classifier's performance as well as guidance on how to incorporate them in the notebook are received. This assignment's overall goal is to increase your knowledge of and practical proficiency in creating and using a neural network for image classification tasks.**

***Index Terms*— Image Classifier, Deep Learning, Neural Network, ReLU, Cross-Entropy Loss, CIFAR10**

I. Introduction

Neural networks are a type of machine learning model inspired by the structure and function of the human brain. They are used for tasks such as image [1] and speech recognition, natural language processing [2], and prediction. At a high level, a neural network consists of layers of interconnected nodes, or "neurons," which receive input, process that input, and produce output. Each neuron has a set of weights that determine how it responds to different inputs. These weights are adjusted during training in order to improve the network's accuracy. Neural networks can be trained using a variety of algorithms, including backpropagation and gradient descent. They can also be designed with various architectures, such as feedforward, recurrent, and convolutional networks. One of the advantages of neural networks is their ability to learn complex patterns in data and generalize to new examples. However, they can be computationally expensive and require large amounts of data for training.

The CIFAR10 dataset is a popular benchmark dataset for image classification tasks in machine learning, which contains 60,000 color images divided into 10 classes, each having 6,000 images. The image size of the photos in the dataset is relatively modest at 32x32 pixels, compared to other similar datasets like ImageNet. The dataset comprises ten classes, including car, truck, boat, cat, deer, dog, frog, horse, ship, and airplane, with an equal number of images in each class in the training and test sets. The training set contains 50,000 images, while the test set contains 10,000 images. Due to the small image size and significant similarities between some classes, such as dogs and cats or ships and airplanes, the CIFAR10 dataset poses a significant challenge for classification tasks. The dataset has been widely used to evaluate the performance of various machine learning and deep learning models and has played a crucial role in the development of state-of-the-art computer vision methods.

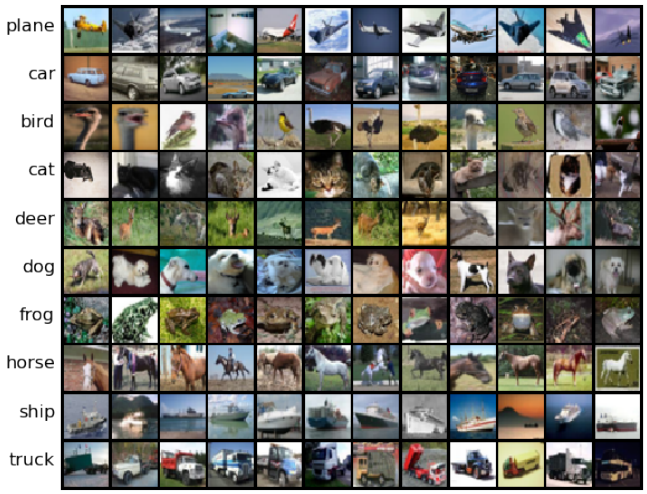

Fig. 1. CIFAR10 dataset example class label images [3].

ReLU (Rectified Linear Unit) is a popular activation function used in neural networks, particularly in deep learning. The function applies a simple rule: it returns the input value if it is positive, and 0 otherwise. ReLU has become a popular choice for activation functions due to its simplicity and effectiveness in deep neural networks. It is computationally efficient and can speed up training by avoiding the vanishing gradient problem. This can occur when using other activation functions, such as the sigmoid or hyperbolic tangent functions, which can cause the gradients to become very small as they propagate through many layers of a neural network.

Cross-entropy loss [4] is a commonly used loss function in machine learning, particularly in classification tasks. It measures the difference between the predicted probability distribution and the actual probability distribution of the target variable. In simpler terms, cross-entropy loss measures how well a predicted probability distribution matches the true probability distribution. It is commonly used in multi-class classification tasks, where there are more than two possible outcomes. In this case, the cross-entropy loss is calculated for each possible outcome and then summed together to get the total loss.

R. Dagdanov is with Department of Aeronautical and Astonautical Engineering, Istanbul Technical University, Turkey dagdanov21 at itu.edu.tr

II. Methodology

In this section, a detailed inspection of the image classification with a neural network approach is carried out with related equations. It is possible to formulate the image classification as follows:

- The input images are preprocessed to ensure that they are in a format suitable for the neural network. This may involve resizing, normalization, or other transformations.
- The neural network is trained on a set of labeled images, typically using a variant of supervised learning. The network learns to map input images to their corresponding labels by adjusting the weights of the connections between neurons in the network. The objective is to minimize a loss function that measures the difference between the predicted labels and the true labels.
- Once the neural network has been trained, it is evaluated on a set of unseen test images to measure its performance.
- The trained neural network can be used to predict the label of new, unseen images. The network takes an input image and produces a probability distribution over the possible labels. The label with the highest probability is typically chosen as the predicted label.

Given a training dataset of N feature vectors $x_1, x_2, ..., x_N$, with corresponding true labels $y_1, y_2, ..., y_N$, where $x_i \in \mathbb{R}^D$ and $y_i \in 1, 2, ..., K$ for a K-class problem, we want to learn a weight vector $w \in \mathbb{R}^D$ and a bias term $b \in \mathbb{R}$ that can be used to predict the class labels of new, unseen feature vectors.

In the assignment, it is expected to construct a two-layer neural network module with PyTorch.

The forward pass of this two-layer neural network:

$$h = XW_1 + b_1 \tag{1}$$

$$a = \max(0, h) \tag{2}$$

$$scores = aW_2 + b_2 \tag{3}$$

where X is the input data, W_1 and W_2 are the weights of the two layers, b_1 and b_2 are the biases of the two layers, and h and a are the intermediate outputs of the first and second layers, respectively.

The Softmax classifier loss function:

$$L_i = -\log\left(\frac{e^{f_{y_i}}}{\sum_j e^{f_j}}\right) \tag{4}$$

where f_j is the j-th element of the output scores $scores$, and y_i is the correct class label for the i-th input X_i.

The regularization term added to the loss function:

$$R = \frac{1}{2}\lambda\left(\sum_{k=1}^{H}\sum_{d=1}^{D} W_{1,k,d}^2 + \sum_{k=1}^{C}\sum_{d=1}^{H} W_{2,k,d}^2\right) \tag{5}$$

where λ is the regularization strength, H is the number of hidden units, and C is the number of classes.

The gradients of the weights and biases computed during backpropagation:

$$\frac{\partial L}{\partial scores} = \frac{1}{N}(a_2 - y) \tag{6}$$

$$\frac{\partial L}{\partial W_2} = a_1^T \frac{\partial L}{\partial scores} + \lambda W_2 \tag{7}$$

$$\frac{\partial L}{\partial b_2} = \sum_{i=1}^{N} \frac{\partial L}{\partial scores_i} \tag{8}$$

$$\frac{\partial L}{\partial a_1} = \frac{\partial L}{\partial scores} W_2^T \tag{9}$$

$$\frac{\partial L}{\partial h} = \frac{\partial L}{\partial a_1} \odot (h > 0) \tag{10}$$

$$\frac{\partial L}{\partial W_1} = X^T \frac{\partial L}{\partial h} + \lambda W_1 \tag{11}$$

$$\frac{\partial L}{\partial b_1} = \sum_{i=1}^{N} \frac{\partial L}{\partial h_i} \tag{12}$$

where N is the batch size, $\odot$ denotes element-wise multiplication, and a_1 and a_2 are the intermediate outputs of the first and second layers, respectively.

III. Experiments

A. Play with a Toy Data

The input to the neural network is a batch of N D-dimensional vectors, where N represents the number of inputs and D represents the input size. The hidden layer consists of H hidden units, where H represents the hidden size. The network is designed to predict classification scores for C categories, where C represents the number of classes. Therefore, the learnable weights and biases of the network have the following shapes:

W_1: First layer weights with shape (D, H)
b_1: First layer biases with shape $(H,)$
W_2: Second layer weights with shape (H, C)
b_2: Second layer biases with shape $(C,)$

The task is to implement a function that calculates the loss and the gradient of the loss with respect to each model parameter given the model weights and a batch of images and labels, similar to what was done in the Linear Classifiers exercise. To achieve this, a staged approach will be taken where the full forward and backward pass of the network will be implemented one step at a time. The distance gap is 2.24×10^{-11} which is smaller than 1×10^{-10}, indicating that the implemented forward pass of the network using the given weights and biases correctly computes scores for all inputs in the nn-forward-pass function.

The implementations of the data and regularization loss using softmax loss and L2 regularization on W1 and W2 weights and excluded b1 and b2 biases from regularization loss are done. It is checked that the implementation using the provided toy data, and the computed loss matches the expected answer within the given tolerance of less than 1×10^{-4}.

The gradient of the loss with respect to the variables W_1, W_2, b_1, and b_2 are given in the Table I below.

TABLE I

GRADIENT LOSSES

Parameter	Max Relative Error
W1	1.101100×10^{-2}
b1	8.239303×10^{-6}
W2	7.306626×10^{-3}
b2	3.122771×10^{-9}

The network training procedure has been successfully implemented as per the instructions given in the assignment. The method nn-train function has been filled in with the necessary code and nn-predict function has been implemented to keep track of accuracy during training. Running the code provided trains a two-layer network on toy data and the final training loss is 0.5211752 which is less than 1.0, indicating the successful training of the network. The training loss is visualized in Figure 2.

Fig. 2. Toy Data Training Iteration Loss.

Additionally, the classification accuracy on training and validation data sets are visualized for each epoch of training in Figure 3.

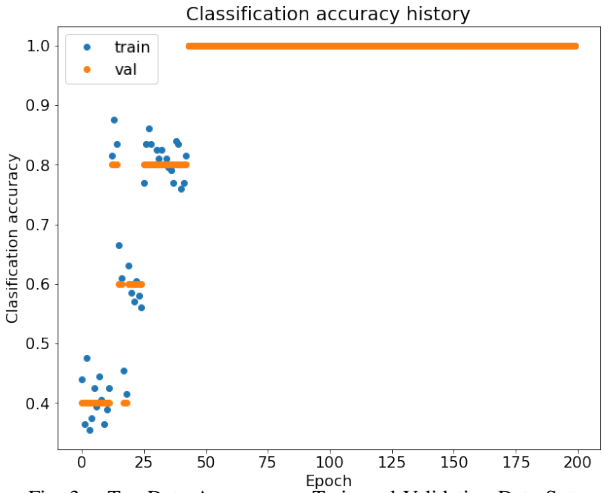

Fig. 3. Toy Data Accuracy on Train and Validation Data Sets.

B. Training Neural Network on CIFAR10 Dataset

By directly applying two-layer neural network training on the CIFAR10 dataset without any hyperparameter tuning, the results are not good as shown in Table II. It is expected that the validation accuracy will be below $\%10$ and with the implementation, $\%9.77$ validation accuracy is obtained.

TABLE II

VALIDATION ACCURACY ON CIFAR10 DATASET WITHOUT HYPERPARAMETER TUNING

Iteration	Loss
0 / 500	2.302862
100 / 500	2.302759
200 / 500	2.302736
300 / 500	2.302609
400 / 500	2.302618
Validation Accuracy: 9.77	

Without hyperparameter tuning, it can be observed from Figure 4 that the model is underfitting since the validation accuracy remains constant throughout the training epochs and the training accuracy deviates significantly.

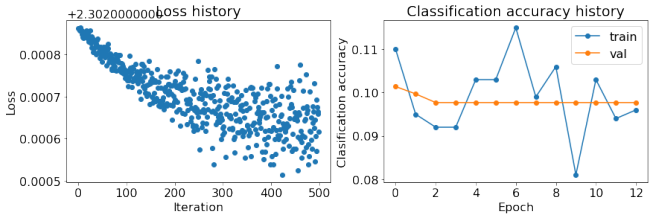

Fig. 4. CIFAR10 Data Accuracy on Train and Validation Data Sets without Hyperparameter Tuning.

The weights of the trained model could be visualized as shown in Figure 5. As it could be observed, the model did not learn anything due to the lack of hyperparameter tuning.

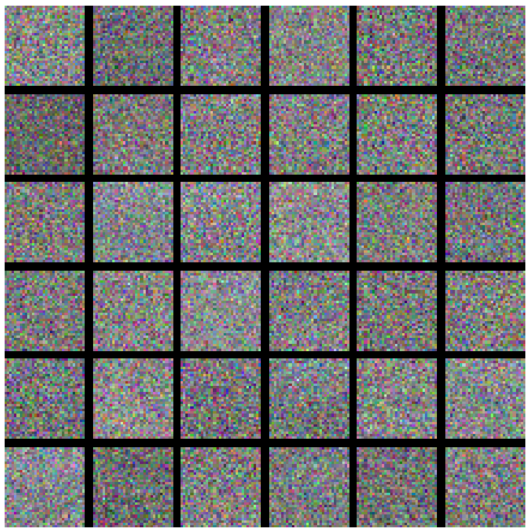

Fig. 5. Illustration of Weights on the Trained Model without Hyperparameter Tuning.

It seems that the model is underfitting, which indicates that its capacity may not be enough to capture the patterns in the data. Increasing the capacity of the model is a good idea to improve its performance. One way to do that is to increase the size of its hidden layer. It will be interesting to see how increasing the size of the hidden layer affects the performance of the model on the validation set. We should expect that the performance will increase as we increase the size of the hidden layer, but there may be a point where the returns start to diminish. Let's try and see how it goes. The effect of changing the hidden layer size on the training performance is shown in Figure 6.

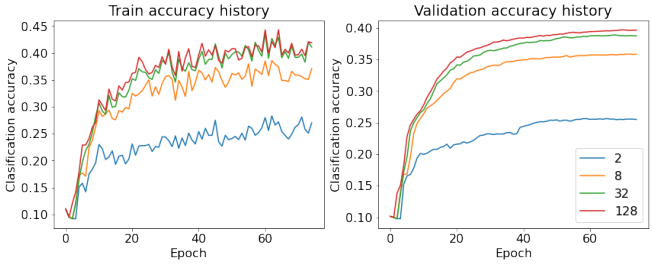

Fig. 6. Model training performance as neural network's hidden layer size changes.

Regularization is another possible factor that may have contributed to the small gap between the training and validation accuracies of our model. A high regularization coefficient can prevent the model from fitting the training data effectively. To investigate this phenomenon, we can train a set of models with varying regularization strengths while keeping the other hyperparameters constant. By doing so, we can observe that setting the regularization strength too high can indeed harm the validation-set performance of the model. Therefore, it is important to find an appropriate balance between regularization strength and model performance to achieve optimal results. The effect of changing the regularization strength on the training performance is shown in Figure 7.

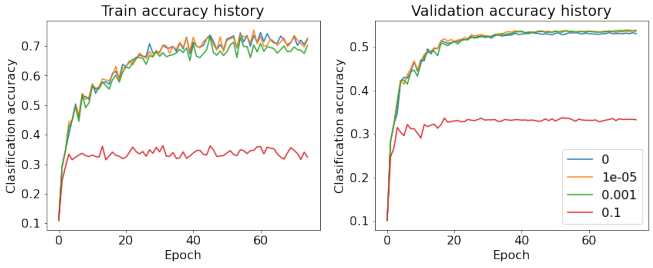

Fig. 7. Model training performance as neural network training as regularization changes.

Understanding the impact of the learning rate on the performance of a neural network model is crucial for effective optimization. As we adjust the learning rate, the model's behavior can change significantly, impacting its ability to converge to the optimal solution. It is essential to note that choosing an appropriate learning rate depends on various factors such as problem complexity, dataset size, and network architecture. Therefore, it is essential to perform a careful analysis of the learning rate's impact on model performance to achieve optimal results. The effect of changing the learning rate on the training performance is shown in Figure 8.

Fig. 8. Model training performance as neural network training as learning rate changes.

Overall, it is beneficial to apply hyperparameter tuning considering model capacity size, regularization strengths, learning rate, and learning rate decay. The grid search method for tuning these parameters is applied and the results are shown below. The best hyperparameters are selected based on validation accuracy. Table III shows the grid space elements for hyperparameters to be tuned.

TABLE III
LISTS OF GRID VALUES FOR TUNING HYPERPARAMETES

Learning Rates	Reg. Strengths	Hidden Sizes	LR Decays
5×10^{-4}	1×10^{-4}	16	1.0
1×10^{-3}	5×10^{-4}	32	0.99
5×10^{-3}	1×10^{-3}	64	0.98
1×10^{-2}	5×10^{-3}	128	0.97
5×10^{-2}	1×10^{-2}	256	0.95
1×10^{0}	1×10^{-5}	512	0.92
		1024	

The performance results of the best model, obtained by applying the grid-search technique to find the best hyperparameters for maximum validation data set accuracy performance on CIFAR10 dataset, are shown in Figure 9.

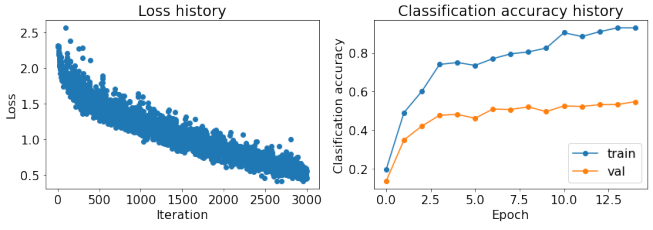

Fig. 9. Best Model Performance with Tuned Hyperparameters

It is expected to obtain at least $\%50$ validation accuracy to get full points. After training the best model on the CIFAR10 dataset, the validation accuracy is reached at maximum **%54.37** and the test accuracy is **%54.00**.

ACKNOWLEDGEMENT

This work is conducted as an assignment of the "BLG 506E Computer Vision" graduate course lecture given by Prof. Hazım Kemal Ekenel.

REFERENCES

[1] F. Perronnin, J. Sánchez, and T. Mensink, "Improving the fisher kernel for large-scale image classification," in *Computer Vision–ECCV 2010: 11th European Conference on Computer Vision, Heraklion, Crete, Greece, September 5-11, 2010, Proceedings, Part IV 11*. Springer, 2010, pp. 143–156.

[2] S. Alshahrani and E. Kapetanios, "Are deep learning approaches suitable for natural language processing?" in *Natural Language Processing and Information Systems: 21st International Conference on Applications of Natural Language to Information Systems, NLDB 2016, Salford, UK, June 22-24, 2016, Proceedings 21*. Springer, 2016, pp. 343–349.

[3] Y. Abouelnaga, O. S. Ali, H. Rady, and M. Moustafa, "Cifar-10: Knn-based ensemble of classifiers," in *2016 International Conference on Computational Science and Computational Intelligence (CSCI)*. IEEE, 2016, pp. 1192–1195.

[4] S. Mannor, D. Peleg, and R. Rubinstein, "The cross entropy method for classification," in *Proceedings of the 22nd international conference on Machine learning*, 2005, pp. 561–568.