BLG 506E Computer Vision 2022-2023 Spring Assignment IV

Resul Dagdanov - 511211135

***Abstract*— This assignment focuses on implementing and using a fully-connected 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. Implemented Linear fully-connected layers, Rectified Linear Unit (ReLU) activation function for forward propagation, and applied Stochastic Gradient Descent (SGD), Momentum, RMSProp, and Adam optimizers for backward propagation. Evaluating the results against each implementation, and utilizing numeric gradient testing to verify the code's accuracy is part of the assignment. The code is developed in the fully-connected-networks.py file and the guidelines are in the fully-connected-networks.ipynb which is given a notebook in order to complete the assignment. Additionally, 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 fully-connected neural networks for image classification tasks along with popular optimization methods.**

***Index Terms*— Image Classifier, Deep Learning, Fully-Connected Neural Network, Linear ReLU, SGD, Momentum, RMSProp, Adam, CIFAR10**

I. Introduction

Fully-Connected Neural Networks, also known as Multi-Layer Perceptrons (MLPs), are one of the simplest types of artificial neural networks used in machine learning. They consist of multiple layers of neurons where each neuron is fully connected to all neurons in the previous layer. These networks are capable of learning complex nonlinear relationships between inputs and outputs and have been successfully applied to a wide range of applications including image recognition [1], natural language processing [2], and speech recognition. In this assignment, we will implement and train a fully-connected neural network for image classification on the CIFAR-10 dataset.

Forward and backward propagation are two critical processes for training fully-connected neural networks. In the forward propagation process, the input data is fed into the neural network, and the activations of each neuron are computed through a series of linear and non-linear transformations. These activations are then used to compute the output of the network. In the backward propagation process, the error between the predicted output and the true output is computed and propagated backward through the network using the chain rule of derivatives. This allows for the computation of the gradients of the loss function with respect to the weights of each neuron, which can be used to update the weights during training. The combination of forward and backward propagation forms the basis of backpropagation, the most widely used algorithm for training fully-connected neural networks.

Stochastic Gradient Descent (SGD) [3] is a popular optimization algorithm used in training machine learning models, particularly neural networks. It is a variant of the gradient descent algorithm that updates the model's parameters incrementally by minimizing the loss function over small randomly selected batches of data, rather than the entire dataset. This makes SGD computationally efficient and well-suited for large-scale datasets. In each iteration, SGD calculates the gradients of the loss function with respect to the model's parameters and updates the parameters in the direction that reduces the loss. By iteratively repeating this process, the model gradually improves its performance on the training data. On the other hand, RMSProp (Root Mean Square Propagation) [4] is a similar optimization algorithm used for training neural networks. It adjusts the learning rate of each parameter based on the moving average of the squared gradients for that parameter. RMSProp is useful for dealing with sparse gradients, as it divides the learning rate by a running average of the magnitudes of recent gradients for a given parameter. This can help prevent oscillations in the training process and improve convergence to a local minimum of the loss function. Additionally, the ADAM optimizer [5] combines the benefits of both RMSProp and Momentum algorithms to achieve faster convergence and better performance on a wide range of optimization problems. While SGD+Momentum [6] maintains a single learning rate for all model parameters and utilizes an exponentially weighted average of past gradients to speed up convergence, ADAM adapts the learning rate for each parameter based on the first and second moments of the gradient. ADAM also incorporates bias correction terms to avoid large steps in the early stages of training. In comparison to SGD+Momentum, ADAM often requires fewer hyperparameter tuning efforts and is more robust to noisy gradients, which can lead to better convergence and more accurate models. However, in some cases, SGD+Momentum can still outperform ADAM, especially when the dataset is small or the model is simple.

The CIFAR10 dataset [8] 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

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

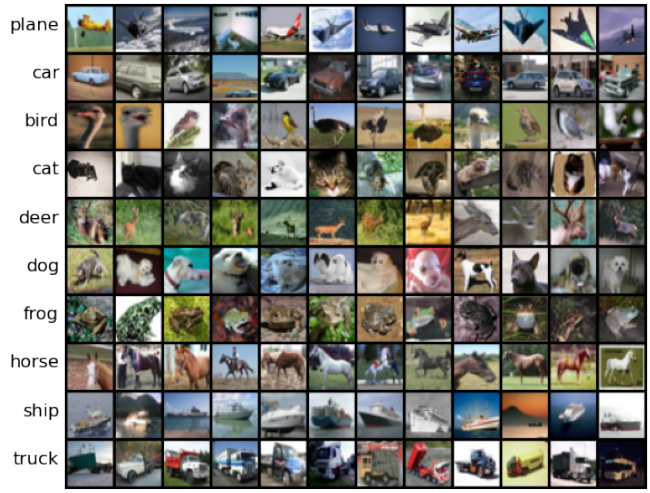

Fig. 1. CIFAR10 dataset example class label images [7].

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.

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. Linear Layer

The forward and backward functions of the Linear class are implemented with comments in the code. It is expected to observe errors less than 1×10^{-7} compared to the correct outputs. In the forward function, we observe 3.683×10^{-8} error, and in the backward function implementation output,

we observe 5.221×10^{-10} for gradient for the input x, 3.4983×10^{-10} for gradient for the weights w, and 5.3731×10^{-10} for gradient for the bias b.

B. ReLU Activation

Similar to the Linear Layer implementation, the forward and backward functions of the ReLU Activation class are implemented with comments in the code. It is expected to observe errors less than 1×10^{-7} for forward calculation and 1×10^{-8} for backward computations compared to the correct outputs. In the forward implementation output, we observe an error of 4.5454×10^{-9}, and in the backward implementation output, we observe an error of 2.6317×10^{-10} with the correct gradients given.

C. Sandwich Layers

When the above two implementations of the Linear class and ReLU classes are finished, the combined error in calculations is calculated. It is expected to observe an error with the correct value being less than 1×10^{-8}. After testing Linear-ReLU.forward and Linear-ReLU.backward, we observe errors of 1.4564×10^{-9}, 7.4629×10^{-10}, 8.915×10^{-10} for dx, dw, and db respectively. These results show that our implementations are correct.

D. Loss Layers: SoftMax and SVM

We have implemented SoftMax and SVM loss functions in the previous assignment. Here we test actual loss function implementations given with our neural network model. At given conditions, the SVM loss is 9.0004 and a corresponding test error of SVM loss is 7.973×10^{-9}. At given conditions, the SoftMax loss is 2.3026 and a corresponding test error of SVM loss is 1.0417×10^{-7}. It is expected to observe at least 1×10^{-7}. This shows that out implementations of the neural network are correct!

E. Two-Layer Network

In this task, we have successfully completed the tasks of implementing a two-layer neural network using modular implementations of the necessary layers. We have re-implemented the TwoLayerNet class using these modular implementations and thoroughly tested the forward and backward passes of our two-layer net. We can now use this class as a model for the other networks we will implement in this assignment. Overall, we have made significant progress in our understanding of neural networks and their implementation, setting the foundation for more complex models to come.

1) Solver: We have successfully completed the task of implementing a modular design for training models in this assignment. The logic for training models has been separated into a separate class called Solver. We have read through the API by using the help() function to familiarize ourselves with the methods available in Solver. Then we used a Solver instance to train a TwoLayerNet that achieved at least a %50 accuracy on the validation set. The loss and accuracy results are shown in Figure 2.

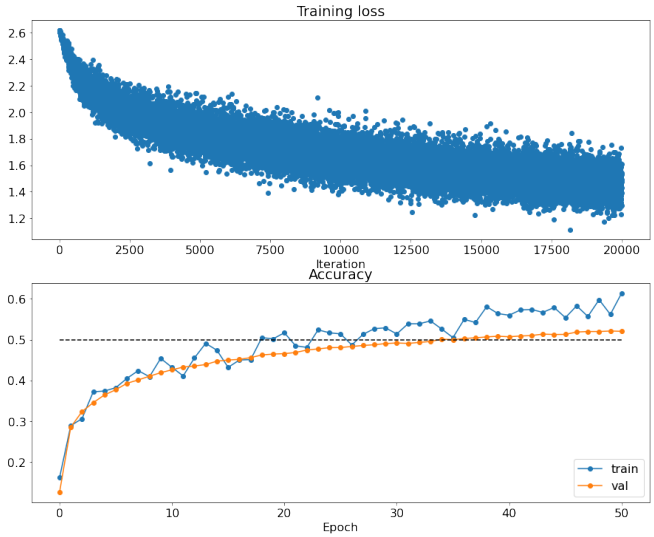

Fig. 2. Application of Solver Class with Two-Layer Network on CIFAR10.

The best model is saved as "best-two-layer-net.pth". We have obtained a two-layer model validation accuracy of **%52.12** on the CIFAR10 dataset. To achieve this performance, we have changed the default parameters with the values in Table I.

TABLE I
HYPERPARAMETERS FOR BEST VALIDATION ACCURACY WITH TWO-LAYER NEURAL NETWORK.

Hyperparameter Name	Value
Hidden Size	1024
Weight Scale	1×10^{-2}
L2 regularization strength	1×10^{-3}
Epoch Number	50
Validation Accuracy: **%52.12**	

F. Multi-Layer Network

After implementing fully-connected layers, the gradient check applications are conducted for different L2 regularization strength (reg.) values. As given in Table II, the initial loss value is 2.305357 and the gradients have relative errors less than 1×10^{-5} as expected. Similarly, as given in Table III, the initial loss value is 7.293696 and the gradients have relative errors less than 1×10^{-6} as expected. The L2 regularization value in the first experiment is $reg = 0.0$ as shown in Table II caption and the L2 regularization value in the second experiment is $reg = 3.14$ as shown in Table III caption.

TABLE II
PARAMETER RELATIVE ERROR VALUES FOR CHECK WITH REG. = 0.0

Parameter	**Relative Error Value**
$W1$	6.06×10^{-8}
$W2$	1.02×10^{-7}
$W3$	5.89×10^{-8}
$b1$	1.28×10^{-7}
$b2$	2.05×10^{-8}
$b3$	3.41×10^{-9}

TABLE III

PARAMETER RELATIVE ERROR VALUES FOR CHECK WITH REG. = 3.14

Parameter	Relative Error Value
$W1$	9.17×10^{-9}
$W2$	9.38×10^{-9}
$W3$	9.56×10^{-9}
$b1$	1.57×10^{-7}
$b2$	2.91×10^{-8}
$b3$	6.23×10^{-9}

As another sanity check, we made the model overfit on a small dataset of 50 CIFAR10 images. First, we tried a three-layer network with 100 units in each hidden layer. In the following cell, tweak the learning rate and weight initialization scale to overfit and achieve %100 training accuracy within 20 epochs. The learning rate is chosen as $lr = 5e-1$ and the weight initialization scale is selected as $ws = 1e-1$ to overfit with three layers neural network with 100 units in each hidden layer. Figure 3 shows the training loss values on the training dataset over 20 epochs.

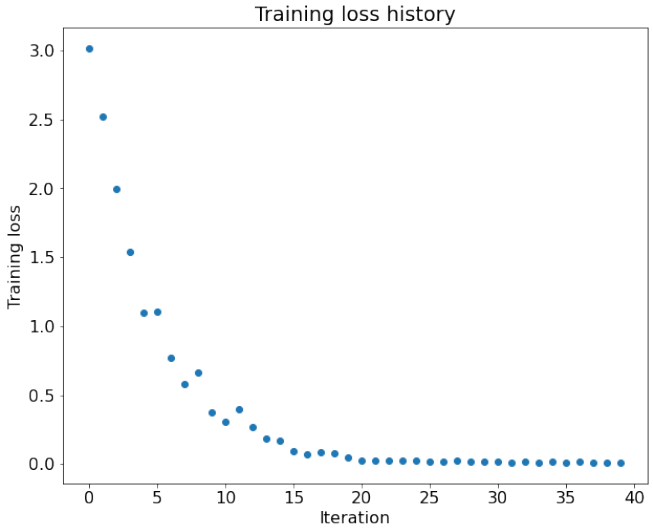

Fig. 3. Overfitting with Three-Layer Neural Network on 50 images of CIFAR10 Dataset for Sanity Checks with LR = 5e-1 and WS = 1e-1.

In the second experiment, we tried a five-layer network with 100 units in each hidden layer. In the following cell, tweak the learning rate and weight initialization scale to overfit and achieve %100 training accuracy within 20 epochs. The learning rate is chosen as $lr = 2e-1$ and the weight initialization scale is selected as $ws = 1e-1$ to overfit with five layers neural network with 100 units in each hidden layer. Figure 4 shows the training loss values on the training dataset over 20 epochs.

The overfitted five-layer neural network model parameters are saved as "best-overfit-five-layer-net.pth".

G. Update Rules

So far we have used vanilla stochastic gradient descent (SGD) as our update rule. More sophisticated update rules can make it easier to train deep networks. We have implemented a few of the most commonly used update rules and compare them to vanilla SGD.

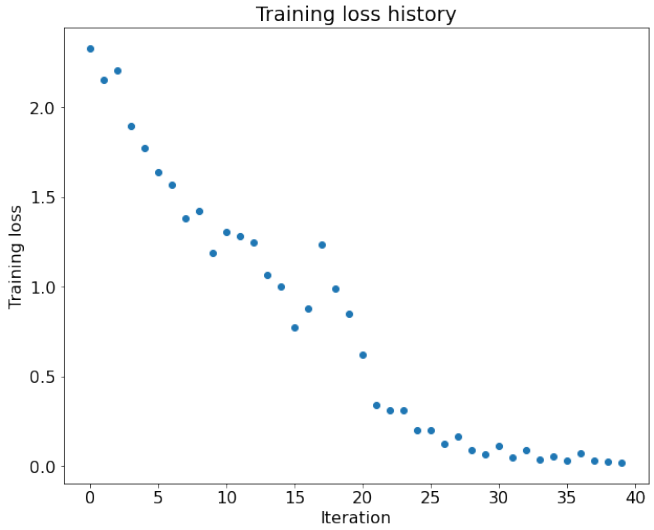

Fig. 4. Overfitting with Five-Layer Neural Network on 50 images of CIFAR10 Dataset for Sanity Checks with LR = 2e-1 and WS = 1e-1.

H. Dropout

Dropout [9] is a technique for regularizing neural networks by randomly setting some output activations to zero during the forward pass. In this exercise, we have implemented a dropout layer and modified your fully-connected network to optionally use dropout.

TABLE IV

RUNNING TESTS WITH DROPOUT OF PROBABILITY = 0.25

Running Tests	Value
mean of input	9.9968
mean of train-time output	10.0115
mean of test-time output	9.9968
fraction of train-time output set to zero	0.7495
fraction of test-time output set to zero	0.0

TABLE V

RUNNING TESTS WITH DROPOUT OF PROBABILITY = 0.4

Running Tests	Value
mean of input	9.9968
mean of train-time output	10.0198
mean of test-time output	9.9968
fraction of train-time output set to zero	0.599
fraction of test-time output set to zero	0.0

TABLE VI

RUNNING TESTS WITH DROPOUT OF PROBABILITY = 0.7

Running Tests	Value
mean of input	9.9968
mean of train-time output	9.9851
mean of test-time output	9.9968
fraction of train-time output set to zero	0.3008
fraction of test-time output set to zero	0.0

1) SGD + Momentum: We have implemented SGD + Momentum technique and have checked the implementation with the difference between the correct parameters. As expected, the next w parameter error is 1.6802×10^{-9} and velocity parameter error is 2.9254×10^{-9} which are less than 1×10^{-7}.

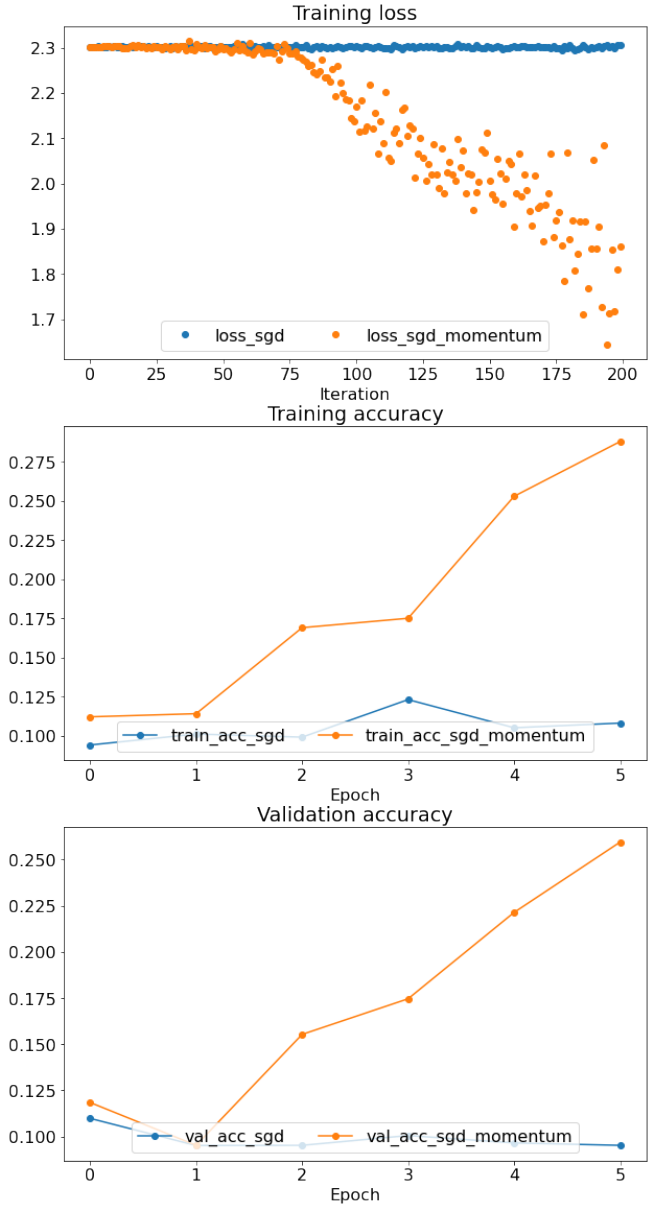

Fig. 5. Comparison of SGD and SGD + Momentum on CIFAR10 Dataset with Six-Layer Neural Network.

After validating that error in parameter calculations is in the valid region, the training with the six-layer neural network on the CIFAR10 dataset images is conducted. The objective is to compare SGD and SGD + Momentum methods in terms of training loss and train-validation accuracies as shown in Figure 5.

2) RMSProp and ADAM: Additionally, RMSProp and ADAM optimizer performances are also illustrated in Figure 6. The training with the six-layer neural network on the CIFAR10 dataset images is conducted. The objective is to compare SGD and SGD + Momentum methods in terms of training loss and train-validation accuracies as shown in Figure 6. The results are as expected, ADAM and RMSProp methods in updating the parameters of the neural network are faster in terms of training time.

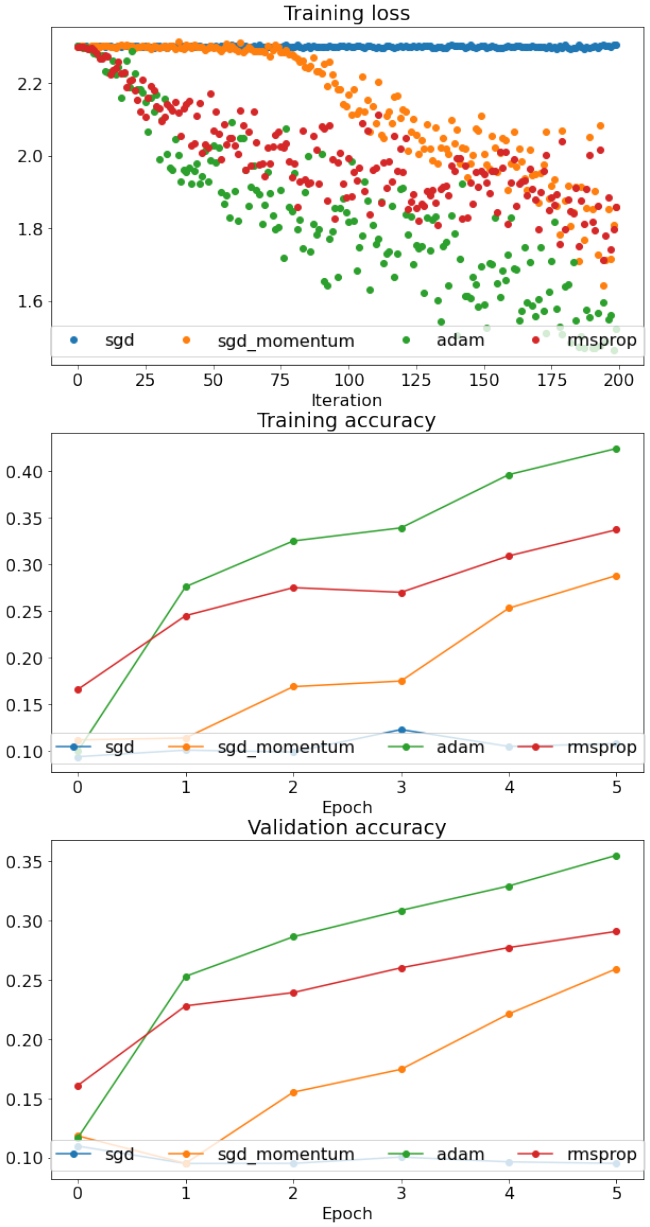

Fig. 6. Comparison of RMSProp and ADAM on CIFAR10 Dataset with Six-Layer Neural Network.

As shown in Figure 6, with these methods, training loss is much lower and the training-validation accuracies are much higher with default parameters.

3) Fully-Connected Neural Network with Dropout: To get a sense of the way that dropout can regularize a neural network, we have trained three different two-layer networks:

1) Hidden size 256, dropout = 0
2) Hidden size 512, dropout = 0

3) Hidden size 512, dropout = 0.5

We have then visualized the training and validation accuracies of these three networks in Figure 7. In this figure, we see that the network with dropout has lower training accuracies than the networks without dropout, but that it achieves higher validation accuracies.

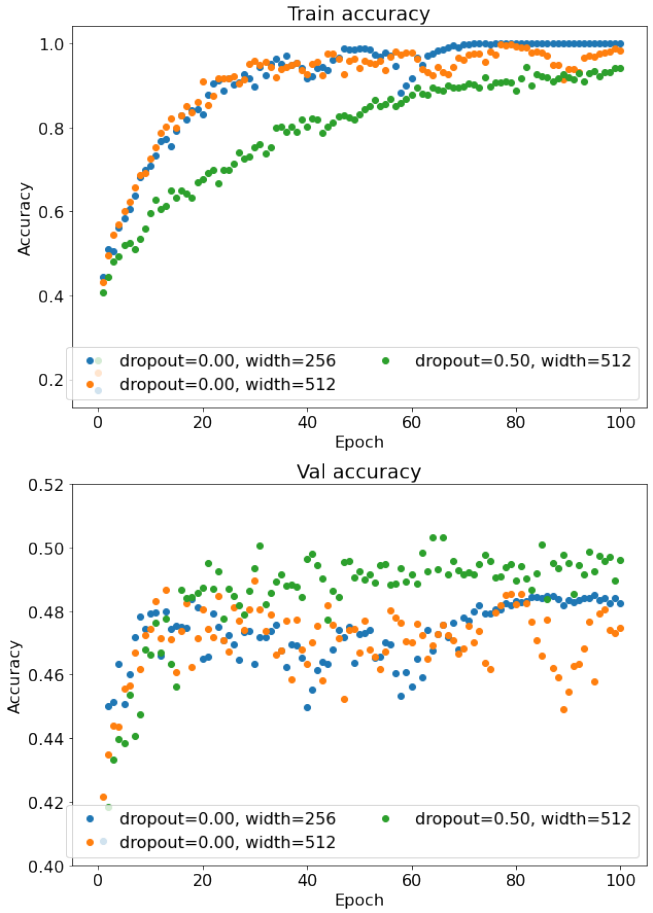

Fig. 7. Training and Validation Accuracy with Dropout in Two-Layer Neural Network Training on CIFAR10 Dataset.

Additionally in Figure 7, we see that a network with a width of 512 and a dropout of 0.5 achieves higher validation accuracies than a network with a width of 256 and no dropout. This demonstrates that reducing the model size is not generally an effective regularization strategy it's often better to use a larger model with explicit regularization.

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] S. Ruder, "An overview of gradient descent optimization algorithms," *arXiv preprint arXiv:1609.04747*, 2016.

[4] S. De, A. Mukherjee, and E. Ullah, "Convergence guarantees for rmsprop and adam in non-convex optimization and an empirical comparison to nesterov acceleration," *arXiv preprint arXiv:1807.06766*, 2018.

[5] D. P. Kingma and J. Ba, "Adam: A method for stochastic optimization," *arXiv preprint arXiv:1412.6980*, 2014.

[6] C. Liu and M. Belkin, "Accelerating sgd with momentum for over-parameterized learning," *arXiv preprint arXiv:1810.13395*, 2018.

[7] 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.

[8] A. Krizhevsky, G. Hinton *et al.*, "Learning multiple layers of features from tiny images," 2009.

[9] G. E. Hinton, N. Srivastava, A. Krizhevsky, I. Sutskever, and R. R. Salakhutdinov, "Improving neural networks by preventing co-adaptation of feature detectors," *arXiv preprint arXiv:1207.0580*, 2012.