Name: AL. Faiq Ahamed

Email: faiq.fq@gmail.com

Project link:

https://github.com/Faiqlatheef/optimize-neural-nets.git

Table of Contents

Table of figures

3.0 Step by step process of the code

3.0.1 Code of all algorithm training

1. **Import the necessary libraries (Matplotlib's pyplot, PyTorch, and torchvision).**
2. **Set the device to use (GPU if available, otherwise CPU).**
3. **Load the CIFAR-10 dataset and define a transform to apply to the data.**
4. **Set up the training and test dataloaders for the dataset.**
5. **Define a convolutional neural network (CNN) class to use for training and testing.**
6. **Set up a list of optimization algorithms to test.**
7. **For each optimization algorithm:**
    i. **Initialize the model and the optimizer.**
    ii. **Train the model for 10 epochs.**
    iii. **Calculate the training and test accuracies for each epoch.**
    iv. **Plot the training and test accuracies.**
8. **Add a legend to the plot and show it**

3.0.2 Code of stochastic gradient descent optimization algorithm training

1. **Necessary imports in order for us to load and divide our data**
2. **Then we create the sets and loaders.**
3. **Explore the data for the training and test sets**
4. **View some of the images from the training and test loaders**
5. **nn are a combination of different layers to come up with the architecture**
6. **Instantiate the model class**
7. **Check if GPU is present on the current machine running the code**
8. **Neural networks pretrained using stochastic gradient descent**
9. **Proceed with the training of the model**
10. **Plot a graph showing our training loss and epoch**
11. **Measure the accuracy of our model**
12. **Test prediction on particular images**
13. **Conveniently save the model**

3.1 Clearly discuss the data used in the software.

CIFAR10 Dataset

We used the CIFAR10 dataset, which consists of 60000 32x32 colour images in 10 classes, with 6000 images per class. There are 50000 training images and 10000 test images.

The dataset is divided into five training batches and one test batch, each with 10000 images. The test batch contains exactly 1000 randomly-selected images from each class. The training batches contain the remaining images in random order, but some training batches may contain more images from one class than another. Between them, the training batches contain exactly 5000 images from each class.

Here are some sample images from the dataset:

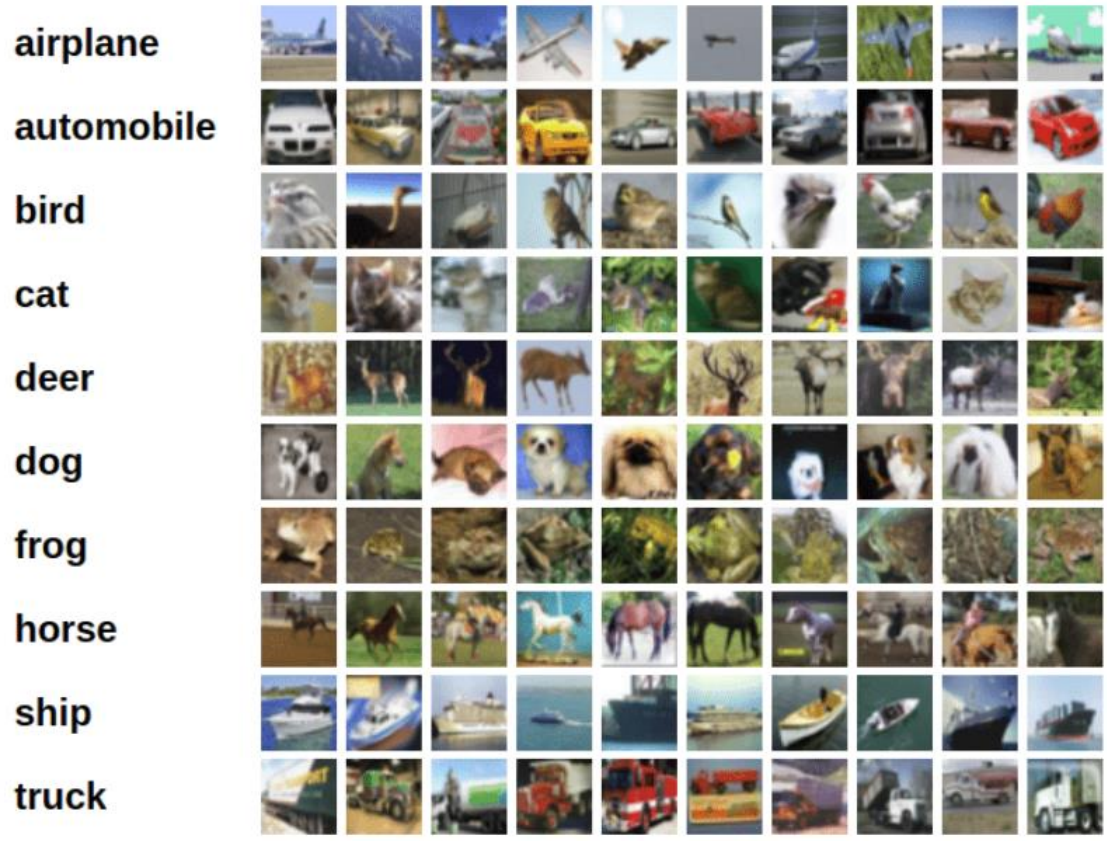

Figure 1: sample images from the dataset

3.2 Clearly discuss the algorithms implemented in the software. including: hyper parameters choice, training parameters, evaluation metrics and etc.

The code provided tests six different optimization algorithms: Stochastic Gradient Descent (SGD), Averaged Stochastic Gradient Descent (ASGD), Adadelta, Adam, Adamax, and RMSprop.

Stochastic Gradient Descent (SGD) is an optimization algorithm that is commonly used to train neural networks. It updates the model's parameters using the gradient of the loss function with respect to the parameters, calculated using a single training example (or a mini-batch of examples). SGD is an efficient algorithm that can handle very large datasets, but it can also be sensitive to the choice of the learning rate and other hyperparameters.

Averaged Stochastic Gradient Descent (ASGD) is a variant of SGD that uses an averaging technique to reduce the variance of the model's parameters. This can help improve the stability of the optimization process and lead to better generalization performance on the test set.

Adadelta is an optimization algorithm that adapts the learning rate for each parameter based on the parameter's past gradient updates. It uses a moving average of the squared gradient updates to scale the learning rate, which can help the optimization process converge more quickly and avoid oscillations.

Adam (Adaptive Moment Estimation) is an optimization algorithm that combines the ideas of SGD and Adadelta by using moving averages of the parameters' gradient updates and squared gradient updates to adapt the learning rate for each parameter. Adam has been shown to be effective on a wide range of deep learning tasks and is generally considered a good default choice for many applications.

Adamax is a variant of Adam that uses the infinity norm instead of the second moment to scale the learning rates. This can help the optimization process converge more quickly and avoid oscillations, especially in deep neural networks.

RMSprop (Root Mean Squared Propagation) is an optimization algorithm that uses a moving average of the squared gradient updates to scale the learning rate for each parameter. RMSprop can help the optimization process converge more quickly and avoid oscillations, especially when the gradients have a non-zero mean and a high variance.

For each optimization algorithm, the code sets some default hyperparameters and training parameters. For example, the SGD optimizer uses a learning rate of 0.001 and momentum of 0.9, while the Adam optimizer uses a learning rate of 0.001 and default values for the beta1 and beta2 parameters (0.9 and 0.999, respectively).

The code also uses the Cross Entropy Loss as the evaluation metric. This loss function is commonly used for classification tasks, as it penalizes incorrect classifications with higher loss values.

3.2.1 Main algorithm used after research

After the training of these algorithms, we decided to use stochastic gradient descent. Neural networks pretrained using stochastic gradient descent, thus we have to choose a loss function, which will assist in optimization.

Typically, a neural network model is trained using the stochastic gradient descent optimization algorithm and weights are updated using the backpropagation of error algorithm.

The “gradient” in gradient descent refers to an error gradient. The model with a given set of weights is used to make predictions and the error for those predictions is calculated.

The gradient descent algorithm seeks to change the weights so that the next evaluation reduces the error, meaning the optimization algorithm is navigating down the gradient (or slope) of error.

3.3 Clearly discuss the plots. Including: x axis, y axis, legends.

The plots generated by the provided code show the training and test accuracies for each epoch for each optimization algorithm. The x-axis of the plot represents the epoch number, and the y-axis represents the accuracy.

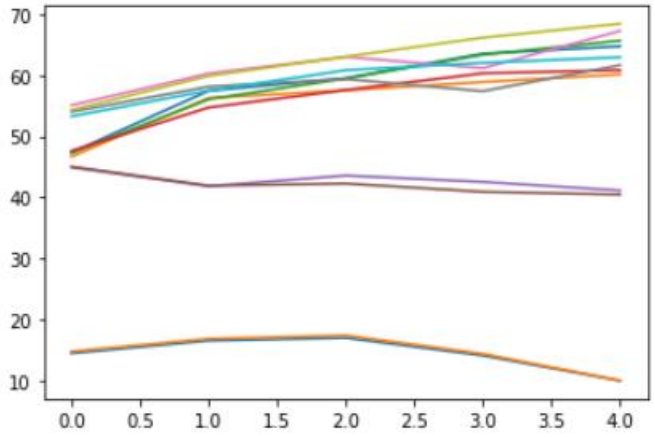

Figure 2: the result plots generated from the software built by the candidate

The code generates one plot for each optimization algorithm, and each plot includes two lines: one for the training accuracy and one for the test accuracy. The training accuracy is the accuracy of the model on the training dataset, while the test accuracy is the accuracy of the model on the test dataset.

The legends of the plots show the names of the optimization algorithms and whether the line represents the training or test accuracy. The legend is located in the upper right corner of the plot and is generated automatically by the **plt.legend** function.

3.3.1 Main algorithm plot used after research

Let me show you a sample of how the curve should look like:

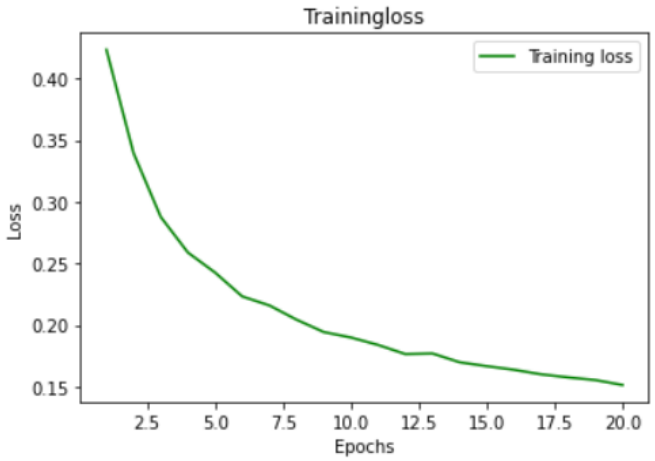

Figure 3: stochastic gradient descent optimization algorithm

Notice how the curve approaches zero. The lower the loss, the better the model can predict. Normally, the lowest point on the curve is where the model can predict well. If the curve starts increasing back, it means that our model is overfitting. Usually, in those cases where the curve starts increasing back, we adjust the number of epochs so that our model trains just for the number of epochs where the curse is lowest. That way, we can get some good performance(accuracy) for our model.

For my case, we trained just for 20 epochs. From the look of the curve, it shows that there is a possibility of the curve going lower (loss reducing), thus increasing the model performance.