Abstract

In this report, Alex Net, a convolutional neural network is implemented and evaluated using the CIFAR-10 dataset, all through the deep learning platform PyTorch. The main goal of this study is to look at the performance of Alex Net while testing different activation functions and changing hyper parameter tuning strategies. The study is set up to optimize the learning rate and minimize the loss rate. The tuning strategies that were tweaked were the activation functions, which were ReLU, Leaky ReLU, and sigmoid, and the hyper parameters, which were the learning rate, batch size, and dropout rate.

Introduction

Recently, deep learning models have been widely known to have great success in image classification tasks and training. Out of these models, convolutional neural networks have become a top choice in their field due to their learning strategies and their ability to learn hierarchical features from images. In this report, Alex Net is used as the deep learning platform with PyTorch, and its performance is evaluated using the CIFAR-10 dataset. The main points of this report are the following:

- Implementation of Alex Net with different activation functions
- Experimentation with different hyper parameters, those of which are the learning rate, batch size, and the dropout rate.
- Finding an optimal combination of hyper parameters and the best activation function.

Methodology

Dataset

The CIFAR-10 dataset is built into the PyTorch library, and is used standardly in many image machine learning tasks. The only data preprocessing steps that were taken were random cropping of the photos and horizontal flipping to help with generalization. Some other steps were taken just to help with convergence and accuracy scores during training, like setting the dataset to a mean of .5 with a standard deviation of .5 across all channels.

Model Architecture and Activation Functions

In this project, the Alex Net system is made up of 5 convolutional layers, followed by 3 fully connected layers. The convolutional layers job is to extract spatial features from the images given, while the three final layers perform the final judgement. The model also incorporates dropout layers, to help reduce overfitting, which is super helpful with the model we have here with many layers. The model is also able to work with different activation functions, which helps explore different training behaviors and optimal settings. The two main activation functions that were explored in this project were ReLU, and Leaky ReLU. ReLU was used as the main activation function, as it is most commonly used in these types of models. Leaky ReLU and Sigmoid were also explored in this project, however both did not stand up against ReLU. A learning rate scheduler was also implemented, reducing the learning rate by 0.1 whenever loss plateaued for more than 3 epochs.

Hyperparameter Tuning

- Learning rate: Started at 0.001, reducing to 0.0001 helped the learning rate plateau immensely
- Batch Size: tested batch sizes of 64, 128, and 256, to help balance stability, and a batch size of 64 ended up being the best.
- Dropout Rate: Tested between 0.3-0.8, a higher dropout rate was the optimal regularization strength at 0.8.

Experiments

Training Procedure

Originally, the training procedure was set for 20 epochs, however it was found that there was much more learning done past 20, and it was found that the optimum amount of epochs was 50, even though it had a longer runtime. After every single epoch, the loss accuracy was updated, allowing us to confidently keep track of progress, plateaus, or any other outcomes. A validation set was used to help track general performance. While early stopping was considered, it was not implemented due to how well the scheduler managed learning rate decay.

Evaluation Metrics

Cross entropy loss and accuracy were both the primary metrics for evaluating model performance. Convergence speed and stability were also monitored throughout this process. The final model reached about 90% after 50 epochs when using ReLU, and other models and hyper parameters came close, however not reaching the peak that ReLU did.

Results with hyper parameter tuning

This was the first setup of hyper parameters on the ReLU function.While the loss rate was dropping pretty regularly, it did not converge fast enough, only getting to a cross entropy loss rate of .4166

Loss Rate Table | Hyper parameters: Batch Size = 128, Learning Rate = 0.001, Activation Function = ReLU, Dropout = 0.5

Epoch	Loss Rate
1	1.775
5	0.9906
10	0.7382
15	0.6333

20	0.5765
30	0.5028
40	0.4494
50	0.4166

Here, we tweaked almost all of our hyper parameters, along with the activation function. While Batch size was kept the same, the learning rate was changed by a tenth to 0.0001, and the dropout rate was increased up to 0.6. We also established Leaky ReLU as a function that is definitely viable, bringing the cross entropy loss rate down to 0.1411.

Loss Rate Table | Hyper parameters: Batch Size = 128, Learning Rate = 0.0001, Activation Function = Leaky ReLU, Dropout = 0.6

Epoch	Loss Rate
1	1.8177
5	1.0611
10	0.7448
15	0.5797
20	0.4619
30	0.3080
40	0.2038
50	0.1411

After the previous experiment, we kept all the hyper parameters the same, but changed back to ReLU to see how much the change to hyper parameters affected the cross entropy loss rate, or if it was the change to activation functions. It came to be that Leaky ReLU was the better activation function slightly over ReLU, having a better loss rate by 0.006.

Loss Rate Table | Hyper parameters: Batch Size = 128, Learning Rate = 0.0001, Activation Function = ReLU, Dropout = 0.6

Epoch	Loss Rate
1	1.8271

5	1.0514
10	0.7561
15	0.5910
20	0.4768
30	0.3196
40	0.2136
50	0.1478

With everything else settled except batch size, we experimented with the batch size, increasing it to 256. While there was a consistent loss rate decrease, it happened too slowly for it to be a viable setting.

Loss Rate Table | Hyper parameters: Batch Size = 256, Learning Rate = 0.0001, Activation Function = ReLU, Dropout = 0.6

Epoch	Loss Rate
1	1.9079
5	1.2410
10	0.9430
15	0.7651
20	0.6513
30	0.4750
40	0.3591
50	0.2608

Here, the drop out rate is increased again to 0.8 from 0.6, and the batch size reduced from 128 to 64. Also notably, ReLU has been used in the past couple experiments and from here on out as well, due to its consistency, as Leaky ReLU potentially had better numbers but the results would fluctuate pretty often.

Loss Rate Table | Hyper parameters: Batch Size = 64, Learning Rate = 0.0001, Activation Function = ReLU, Dropout = 0.8

Epoch	Loss Rate
1	1.8059
5	1.9711
10	0.6621
15	0.4952
20	0.3883
30	0.2426
40	0.1656
50	0.1174

Results and Analysis

These experiments and hyper parameter tunings allowed us to find an optimal set up for Alex Net with 5 convolutional layers and 3 connected layers. The optimal set up reduced cross entropy loss rates, as well as consistently gave results that were reliable. The ReLU activation function showed consistent drops in loss rate, as well as being a very reliable option. The Leaky ReLU activation function at some points performed better than the original ReLU function, but this was not a consistent result; ReLU was the clear better option due to its consistency. Not shown in the experiment section was the sigmoid function, which had pretty bad results, consistently ending up around the 2.0 cross entropy loss rate. The hyper parameters were all adjusted accordingly, and it was found that the best batch size was 64, the best learning rate was 0.000, and the best drop out rate was 0.8.

Conclusion

This research and its results show that the Alex Net system for image recognition can be effective depending on the dataset preprocessing, as well as the different activation functions used and how much hyperparameters were tuned and adjusted. With the 5 layer convolutional neural network as well as its 3 connected layers, ReLU was the best choice, beating over Leaky ReLU and sigmoid activation. The hyper parameters that affected the performance the most were the drop out and learning rates, as well as the batch size. Another setting that played a huge role was the learning rate scheduling and the batch normalization, which allowed our loss rate decreases to stay on track. While other hyper parameters could have been used, such as adding more convolutional layers, or weight initialization, the settings tuned in this project were very helpful towards finding how Alex Net performed.

References

- Krizhevsky, A., Sutskever, I., & Hinton, G. E. (2012). ImageNet Classification with Deep Convolutional Neural Networks. In Advances in Neural Information Processing Systems (NeurIPS).
- CIFAR-10 Dataset: https://www.cs.toronto.edu/~kriz/cifar.html
- PyTorch Documentation: https://pytorch.org/docs/stable/index.html
- Professor Tu's lectures and assignments