Cognitive Robotics Lab Report

Sean Parker
34212-Lab-S-Report

April 15, 2020

1 Introduction

The aim of this report is to describe the current problem in computer vision for solving image recognition, which is very important in the area of robotic vision, to describe the current state-of-the-art network architectures and finally to descirbe my approach for solving this problem. This project used the CIFAR-10 dataset which consists of 60000 32x32 colour images, grouped into 10 classes. Although the dataset has low quality images, and for a real-world robot it's visual acuity would be much greater (for example iCub which use 2x Dragonfly2 camera, each with a resolution of 648x488 pixels), solving image recognition on a simple dataset provides a proof of concept for a system that could be upscaled in order to perform image recognition on higher resolution images.

Firstly, the network architecture was chosen before evaluation of the network with a selection of hyperparameters. In order to find a good architecture, I researched the current state of the art (see Section 2) in order to determine the baseline performance. Using a very deep CNN architecture, one can achieve an error rate of only 1.0%[1], however this architecture has over 500-million parameters which is far too large for the available computer power for this project.

As such, I looked for simplier network architectures that still achieved good results on the CIFAR dataset. After some research in modern techniques to improve training networks, I decided to include layers using Batch Normalisation and Dropout layers which have been shown to improve the accuracy of networks[2, 3].

Lastly, in terms of hyperparameter choice, there were some main parameters that had to be refined for the network topology. The main hyperparameters that had to be defined were the learning rate of the optimiser, and weight decay for L2 regularization. Additionally, the network topology is a based on blocks of (CONV + ReLU + Batch Norm)[1]. More details of the network architecture can be found in Section 3.

2 State of the art

Over the past decades, there has been a ever accelerating improvement with neural networks, convolution neural networks and other architectures that have been developed. Over the past 5-7 years, a renewed focus has been on deep neural networks, trying to overcome the problem of vanishing graidents. In this effort, the networks have been able to "adapt" to new experiences much faster and with much greater accuracy. (TODO cite)

Modern networks have millions of parameters and can take days or even weeks to train; with the advances in GPU parallel processing it takes a long time. However, once trained, the models don't require a large amount of processing power, hense, for robotics this can lead to cheaper robots where the designer can innstead make investment into better joints/motors. Also, with the advancement of DNN (Deep neural networks), the networks (and therefore the robots) can operate on the raw sensor data, without little or no pre-processing.

3 Implementation

Figure **??** shows the network architecture diagramatically, it is also described below for completeness. After testing different architectures, I found a good middle ground between network complexity, training time, and performace was the following.

Firstly, we expect the input to be a 32x32 input image with three channels (RGB). The network consits of three blocks, each using the same architecture, but with different hyperparameters. The structure of each block is as follows. Convolution of filters with stride 1, L2 regularization (weight decay

[1]In this case, "+" refers to a combination of ideas, rather than the plus operator in mathematics.

of 5×10^{-4}), ReLU activation, and Batch Normalisation. Next, aonther Convolution layer, ReLU activation and Batch Normalisation. Finally, the block finishes with Max pooling (size of 2x2), ReLU activation and a dropout layer.

We repeat this structure two more times, before finally flattening the output layer with a single dropout layer which is connected to a fully connected layer with 10 units. These units predict the category in which the image should be classified.

Additionally, I also used a learning rate schedule in order to start with a high learning rate at the start of training, and then at specific points during training, the learning rate is reduced in order to perform fine-tuning. The schedule used was as follows:

- Learning rate = 0.002, epoch ≥ 0
- Learning rate = 0.001, epoch ≥ 25
- Learning rate = 0.0005, epoch ≥ 50
- Learning rate = 0.0003, epoch ≥ 100

Finally, the training was performed for 120 epochs, I found this was a good balance between training time and performance. Using the learning rate schedule above, around epoch 60, the accuracy reached 83% at which point the network needed fine tuning in order to reach the final accuracy of 85-87%. Using Google Colab and a NVIDIA P100, it took $\approx$ 20s per epoch, and therefore around 40 minutes to train the model.

4 Experiments

This section will describe the process of finding the optimal hyperparameters for the network in order to get the best performance. My approach to find these was in large part trail and error, documenting each run with the loss and accuracy, both during the training and testing phases. The starting point for the hyperparameter search was that of the papers that introduced some of the techniques used in the report [4].

As can be seen in Figure (TODO), by keeping all else the same, just changing the learning rate has a large effect on the overall accuracy of the network. It is important to note that since we use random initalisation of the

network, some tests may start closer to a local minima than others. However, the overall effect can still be observed by looking at the results. Too low of a learning rate means the network will struggle to learn the classification, and too high learning rate leads to quickly overfitting the training data and getting stuck in a poor local minimum. A promising area of hyperparameter exploration is that of Bayesian hyperparameter optimisation which has been shown to lead to better performance over hand tuning of the learning rates [5, 6].

5 Conclusion

References

[1] Yanping Huang, Youlong Cheng, Ankur Bapna, Orhan Firat, Mia Xu Chen, Dehao Chen, HyoukJoong Lee, Jiquan Ngiam, Quoc V. Le, Yonghui Wu, and Zhifeng Chen. Gpipe: Efficient training of giant neural networks using pipeline parallelism, 2018.

[2] Nitish Srivastava, Geoffrey Hinton, Alex Krizhevsky, Ilya Sutskever, and Ruslan Salakhutdinov. Dropout: A simple way to prevent neural networks from overfitting. *Journal of Machine Learning Research*, 15(56):1929–1958, 2014.

[3] Sergey Ioffe and Christian Szegedy. Batch normalization: Accelerating deep network training by reducing internal covariate shift, 2015.

[4] Karen Simonyan and Andrew Zisserman. Very deep convolutional networks for large-scale image recognition, 2014.

[5] Jasper Snoek, Hugo Larochelle, and Ryan P. Adams. Practical bayesian optimization of machine learning algorithms, 2012.

[6] James Bergstra, Rémi Bardenet, Yoshua Bengio, and Balázs Kégl. Algorithms for hyper-parameter optimization. In *Proceedings of the 24th International Conference on Neural Information Processing Systems*, NIPS'11, page 2546–2554, Red Hook, NY, USA, 2011. Curran Associates Inc.