Deep Learning Project 1

Rohan Chandra

CMSC 828G

Abstract

This project report describes different architectures and parameters of deep convolutional neural networks trained on the CIFAR-10 dataset. The primary aim of this project is to take an introductory step into the realm of deep learning and to serve as a motivation for future research in the applications of computer vision using deep learning techniques. Architectures have been built and weights have been trained using caffe and testing has been done in matcaffe.

Keywords: convolutional neural networks, deep learning, backpropagation

1. Introduction

Deep learning is a branch of machine learning based on a set of algorithms that attempt to model high level abstractions in data by using a deep graph based algorithm with multiple processing layers, composed of multiple linear and non-linear transformations.Convolutional Neural Networks (CNNs) are an example of such algorithmic architectures that have taken the computer vision community by storm, significantly improving the state of the art in many applications. Inspired by the architecture in [3] which used CNN's on hand-written digit on the MNIST database, this work attempts to replicate and improve upon the work of [2]. [4] and [5] gave an idea on the different types of loss functions and hyper parameters that can be used.

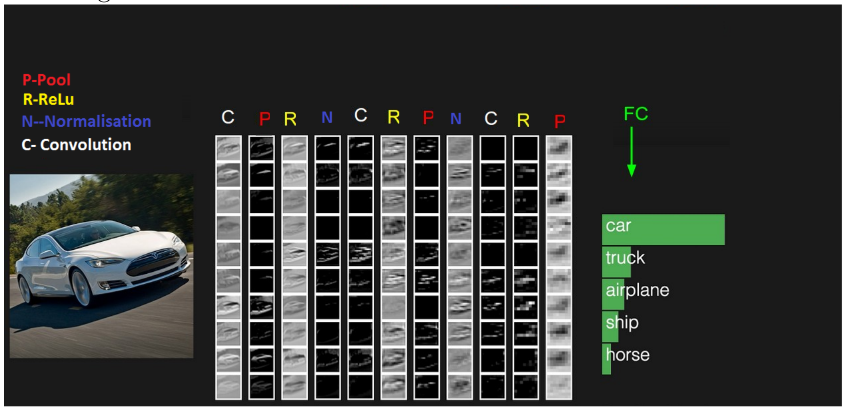
For supervised learning, we use the CIFAR-10 dataset [1], which is a labelled subset of the 80 million tiny images, containing 60,000 images. It consists of 10 classes: airplane, automobile, bird, cat, deer, dog, frog, horse, ship, and truck. Each class contains exactly 6,000 images in which the dominant object in the image is of that class. The classes are designed to be completely mutually exclusive- for example, neither automobile nor truck contains images of pickup trucks.

This project is being undertaken as part of the course requirements for CMSC 828G: Image Understanding taught by Prof. Rama Chellapa. We start by taking a pre-trained model architecture and analysing its performance. We then proceed to test the architecture using different hyper parameters. The default model has also been challenged by introducing new/ deleting existing layers and otherwise modifying the layers of the old model and evaluating the performace.

2. Model Architecture

The CNN consists of 14 layers including the data layer and the softmax loss function layer. The input is a 32x32 pixel image. The reason for this size is that it is desirable that potential distinctive feature points such as stroke end-points or corners can appear in the center of the receptive field of the highest level feature detectors. The input pixels are normalized so that at the background level, the mean input is roughly zero and the variance roughly one which accelerates learning.

Figure 1: The modified architecture of the CNN used in this work



Layer C1 is a convolutional layer with 32 feature maps. Each unit in each feauture map is of size 5x5. Padding of 2 is provided with a stride of 1.

Layer 2 is a maxpool layer of kernel size 3x3 and stride 2. The maxpool layer basically shrinks the feature map size to compress data while retaining the essential information. This ensures that further convolutions done by the later convolutional layers generate useful feature maps. The kernel is passed over all the pixels with a specified strides and the maximum value of the kernel is chosen. Rest are discarded. This method works very efficiently.

Layer 3 is a ReLu layer which is a sort of cleanup layer which normalizes the values which keeps the math from breaking. Positive values remain as is and negative values are set to zero. This is called a rectified linear unit and is one of the most widely used activation functions.

Layer 4 is a normalization layer.

Layer 5 is the second convolutional layer. It again has a kernel size of 5x5 and outputs 32 maps. As before, it has a stride of 1 and padding of 2.

Layer 6 is the ReLu layer. This is different as earlier the first convolutional layer was followed by a max pooling layer.

Layer 7 is the second poolong layer but this time it is an ave- pooling layer.

Layer 8 is normalization.

Layer 9 spits out 64 feature maps using a kernel of 5x5 with padding of 2 and stride 1. This is followed by another ReLu and AVE-pool layer.

Consider a fully connected layer as a matrix multiplication of 1xN and NxM to produce a result of dimension 1xM. For computing this data the ip layer or the fully connected layer reshapes the input data into a one dimensional vector. The weights are in the form of an NxM matrix. The fully connected layer allows the multiplication of the input data with the weight matrix to produce a 1xM matrix. That is the role of the ip or inner product layer.

The softmax loss layer computes the multinomial logistic loss of the soft3max of its inputs. Its conceptually identical to a softmax layer followed by a multinomial logistic loss layer, but provides a more numerically stable gradient. The Euclidean loss layer computes the sum of squares of differences of its two inputs. A non-complete connection scheme keeps the number of connections within reasonable bounds. More importantly it forces a break of symmetry in the network. Different feature maps are forced to extract different (hopefully complementary)

3. Experiments

We performed an analysis on the learning rate alpha on both the training set and the validation set, the number of iterations and the momentum factor. The learning rate for the training set is denoted by LR and the learning rate for the validation set is denoted by `LR_1`. Through these experiments, we gained valuable insights as to how convolutional neural nets converge to a local optimum depending on how the learning rate is chosen.

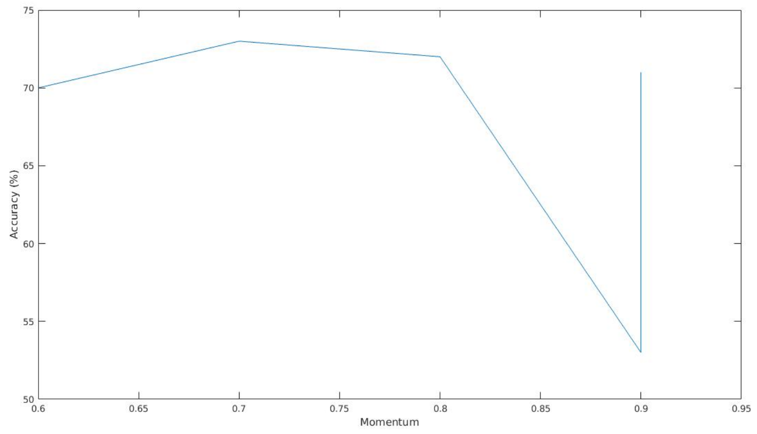
Table 1 shows the various settings of the convolutional neural network.

Table 1: Settings for the CNN used in this work

LR/iter (training)	`LR_1`/iter (validation)	Momentum	Accuracy (%)
0.01/4000	0.0001/4000	0.9	53
0.001/4000	0.0001/5000	0.9	68
0.001/4000	0.001/5000	0.9	70
0.001/3000	0.001/3000	0.9	70
0.001/5000	0.001/5000	0.6	70
0.001/4000	0.001/4000	0.9	71
0.001/5000	0.001/5000	0.8	72
0.001/5000	0.001/5000	0.7	73

As can be shown, we kept the minimum and maximum iterations to be 3000 and 5000 respectively. Different variations have been tested with the number of iterations for training data and validation test set data being different. Setting both LR and `LR_1` to 0.01 broke the network so at best we had to set at most only the LR to 0.01. However, the best results were achieved through setting both to 0.001. If the objective has the form of a long shallow ravine leading to the optimum and steep walls on the sides, standard SGD will tend to oscillate across the narrow ravine since the negative gradient will point down one of the steep sides rather than along the ravine towards the

Figure 2: Variation of momentum



optimum. The objectives of deep architectures have this form near local optima and thus standard SGD can lead to very slow convergence particularly after the initial steep gains. Momentum is one method for pushing the objective more quickly along the shallow ravine.The current network performed best at a momentum of 0.7 as opposed to the pretrained default 0.9. Figure 2 shows how the accuracy varies with momentum.

4. Discussion

It is evident that the learning rate, number of iterations and the momentum factor all have significant roles in making the network function efficiently. Further more, there is limit to the change in the hyper parameters after which the convolutional neural network breaks. For example, setting the momentum greater than 0.9 sets the accuracy to 0.1 and loss function to NaN thus breaking the network. On the other hand, lowering the accuracy to 0.7 (or 0.8) boosts the accuracy by approximately 5%. The network chosen works best at a learning rate of of 0.001 for both the training and validation set for 5000 iterations (8 epochs). After achieving this steady state, further improvements on the accuracy was achieved on lowering the momentum down to 0.7.

By starting out with a pre-trained model with 4000 iterations and a learning rate of 0.001 for the training set and 0.0001 for the validation set, we achieve an accuracy of 68%. We have exerimented with this model extensively and have tested cases where the model degrades in performance to 53% and slowly made our way upwards to an improved model with an accuracy of 73%.

By playing around with the hyperparameters, we have learned that finetuning the CNN has a huge impact on the accuracy of the net. This project has served as a confidance booster and has motivated me to undertake large and more complex datasets. The next steps we would like to make in this work is to changethe architecture of the neural network itself. Just by tweaking the parameters leads to a cap in the design space. Much more flexibility can be achieved if one starts to play around with the layers through matcaffe.

5. References

[1] Alex Krizhevsky. Learning multiple layers of features from tiny images. 2009.

[2] Alex Krizhevsky and G Hinton. Convolutional deep belief networks on cifar-10. *Unpublished manuscript*, 40, 2010.

[3] Yann LeCun, Léon Bottou, Yoshua Bengio, and Patrick Haffner. Gradient-based learning applied to document recognition. *Proceedings of the IEEE*, 86(11):2278–2324, 1998.

[4] Florian Schroff, Dmitry Kalenichenko, and James Philbin. Facenet: A unified embedding for face recognition and clustering. *CoRR*, abs/1503.03832, 2015.

[5] Yaniv Taigman, Ming Yang, Marc'Aurelio Ranzato, and Lior Wolf. Deepface: Closing the gap to human-level performance in face verification. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pages 1701–1708, 2014.