6.867 Final Project: Simple Versus Complex Neural Networks for Computer Vision Tasks

Michele Pratusevich

December 4, 2014

Abstract

Something here. Do I even need this?

1 Introduction

Collaboration between artificial intelligence, neuroscience, and machine learning resulted in the invention of neural networks for use in classification or prediction tasks. With roots from single-layer perceptrons [12] to multi-layer and kernel perceptrons [1], increasing success on non-linear classification tasks (especially related to vision or images) have seen the revival of their popularity among the computer vision community.

A popular success story is the development of a back-propagation convolutional neural network (CNN) for handwritten digit recognition [8]. The original data used was approximately 10,000 handwritten digits, classified into one of 10 categories (one category for each digit). This original dataset was later expanded into the MNIST handwritten digit database with over 60,000 examples [9], [11]. The network used by LeCunn et. al. used the idea of convolution layers - described essentially as features extracted from parts of the image by kernels of varying sizes, generally getting smaller and smaller. The general idea is that these $n \times n$ kernels will 'find' specific features in the image. In between, there are layers that perform averaging or weighting of the features that it receives. The final layer is fully-connected to the receptive fields of the previous layer and outputs a probability estimate for the class the image belongs to, meaning it essentially performs a weighted sum of all the inputs it sees for each parameter, with different weight parameters for each predicted class. Improvements to this network have been made incrementally over time, but the error rates on the test set for this initial network was 3.7 percent.

The interesting insights from LeCun's simplified architecture [8] compared to a previous network that had more connections (also designed by LeCun) [7] were two-fold: (1) that images could be fed directly into the neural network rather than extracting features, then feeding the features into the network, and (2) knowing a-priori the nature of the classification task can give hints about how to simplify the network to contain fewer parameters to optimize over. In general, these ideas of simplification of neural networks can be applied to larger and more complex neural networks [10].

Since LeCun's successful use of CNNs for handwritten digit classification, CNNs became a popular technique for completing computer vision tasks. Krizhevsky et. al. [6] constructed an 8-layer CNN that achieved 17 percent top-5 error rates on the ImageNet database [13], taking advantage of optimizations using GPUs and a few other numerical improvements cited in the paper. Since then, very deep CNNs have been used for nearly every computer vision task with a standard dataset, and countless other non-standardized tasks. Since the original AlexNet was published, fine-tunings of the original network, in addition to changes to the architecture, were performed with the hopes of creating a neural network that can be used as a generalized feature extractor for all kinds of image tasks [4]. Software packages to facilitate this and the use of GPUs for speed-up were created [5].

One big problem with using these extremely deep CNNs for feature extraction, fine-tuning, and general image classification tasks is the time needed to train these networks. With GPU speedups and optimized code, this task is made easier, but it still takes approximately 3 days on a GPU to train on the entirety of the ImageNet database (6 million images). Fine-tuning for a specific task using a pre-trained network takes less time, but depending on the desired result takes a few hours. Another big problem with deep CNNs for vision tasks is the need for extremely large quantities of labeled training data. For a large network with 10000s of parameters, having only 50 labeled training images is not enough to optimize over. AlexNet was only able to train from scratch because the training set for ImageNet is incredibly large, with approximately 10,000 images per class and a total of 1000 classes.

Training neural networks still requires the fine-tuning of hyperparameters dictating the learning rate and the weight decay, but some have offered suggestions for how to improve back-propagation and the choosing of hyperparameters [3]. However, simultaneously, there have been a number of papers questioning the need for the depth and complexity of the recently-used CNNs [2]. LeCun provided suggestions for the simplification of neural networks when working with the original handwritten digit classification network [10]. Another interesting area of research has been the idea of using Gaussian processes to optimize parameters in a neural network for a specific task.

In this project, I will explore the idea of simplified CNNs for the scene recognition vision task, drawing on ideas from recent literature about the effectiveness of using pre-trained networks, simplifying existing networks, and using extracted features to do various tasks. The main concepts I will explore are:

1. The effect of network size on training time
2. The effect of kernel size on the learned receptive field and on training time
3. The process of choosing hyperparameters for learning

The main guiding this exploration is: can you construct a significantly simpler neural network for scene classification that has comparable results to an extremely deep neural network? A general answer to this question is a philosophical question in the field of computer vision, since it touches on the question of whether a highly-specific system that is quick to construct or a highly-general system that is versatile a better outcome. I claim that it depends on the task, but the idea of constructing a simple neural network for complex tasks like scene classification has not been addressed in the literature.

2 Dataset, Tools, and Related Work

The task of scene recognition is different from object recognition in that scene recognition relies on clues from the entire image and the general layout rather than specific qualities of objects. The standard dataset for scene recognition is the SUN397 database [15], which has 397 scene categories, with at least 100 images in both categories. The standard train / test splits have 50 images per category in both training and testing. The benchmark classification results on this dataset using hand-crafted features is reported by Xiao, et. al. [15], [14]. To build on the SUN397 database is the PLACES database [16], which benchmarks itself against SUN397, and notably uses CNNs for scene classification, rather than hand-crafted features. Because SUN397 has a manageable amount of data to use for training, I will use that database to do tests, benchmarking against the classification rates reported in the SUN papers. The PLACES database and results will provide a second state-of-the-art benchmark for scene classification as well.

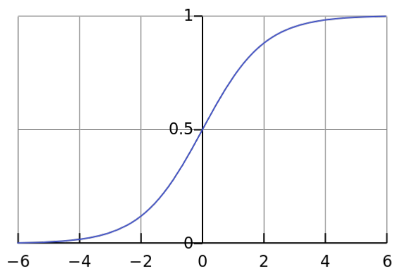
To construct the neural networks and take advantage of GPU capabilities, I will use the Caffe software package [5], optimized for GPU performance and slowly becoming the standard in academic work for building CNNs. There are bindings for C++, Python, and MATLAB, allowing for the use of various interfaces for analysis. The added benefit of using this system is that the PLACES database provides a trained CNN model that can be used for fine-tuning with SUN. Additionally, the Vision Group has a cluster of GPU machines that can effectively run Caffe with some persuasion, so the training times for the networks will be drastically improved over running on a small laptop PC CPU. One of the bottleneck steps in using training data is reading data from disk, so the training and testing sets are converted into LMDB databases optimized for multi-threaded querying. All images are resized to 227 x 227 pixels, in accordance with comparable computer vision experiments. The reason for the small size in images is three-fold: (1) consistent higher-resolution images were only available as standard datasets relatively recently; (2) it has been shown in prior experiments from both neuroscience and computer vision that image-level features do not change significantly when shrunk to this size REFERENCE, and (3) smaller images mean performance increases when reading from disk and computing features over the entire image.

3 Method

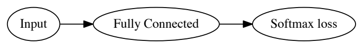
The first step in the neural network analysis is to set up Caffe to run on both my personal computer (for small-scale testing and development) and the Vision Cluster (for running experiments). Both proved to be a challenging task because of the need to link highly optimized numerical libraries, resulting in my writing an addendum for how to install Caffe on OSX 10.9 on the Caffe Github wiki.

To set up a single network in Caffe is complicated without knowing the interface, so as a benchmark to start, I constructed a basic logistic regression classifier as a neural network. In the default neural network, activation is computed using the sigmoid function, shown in Figure 1a. In multiclass logistic regression, the loss function is the softmax function given K linear input functions, represented by

$$P(y = j|x) = \frac{e^{x^T w_k}}{\sum_{k=1}^{K} e^{x^T w_k}} \tag{1}$$



(a) Sigmoid activation function (taken from Wikipedia)



(b) Simplified neural network architecture for logistic regression

The setup for (linear) logistic regression as a neural network is then represented as shown in Figure 1b. A fully-connected layer is equivalent to a weighted sum of the dot product between the inputs, represented in the softmax function as $x^T w_k$. Represented as a neural network with 397 outputs in the softmax loss function, there are 397 neurons in the fully connected layer, meaning approximately 500 parameters. Using stochastic gradient descent to compute the parameters, the optimal learning weight (by grid search and suggestions from Bottou [3]) was found to be approximately 1×10^{-8}. On a GPU for 20000 iterations, this takes approximately one hour and results in a top-1 accuracy prediction of 4 percent and a top-5 accuracy prediction of 13 percent. Because simple logistic regression learns relatively few parameters and is only capable of linear combinations of inputs, the idea is that adding any sort of more complicated network will increase this base classification rate. Methods involving more complicated networks were only considered if they yielded results better than this baseline 13 percent top-5 accuracy with logistic regression.

The standard metric for measuring success against the SUN database is the top-1 classification accuracy rate for correctly predicting with maximum likelihood the correct class. However, because many categories in the SUN397 database are similar (like for example ‘building exterior’ and ‘apartment building outdoor’), I will also report a top-5 accuracy rate.

After exploring this pipeline using Caffe, for each set of experiments I had to do the following steps:

1. Define the network representation
2. Define the solver parameters
3. Define the validation / hyperparameter search procedure
4. Train the network on the GPU using the training set
5. Test the network on the GPU using the test set

For all experiments, the same training and test sets were used.

4 Experiments

Because I was limited by computational power and time, for each experiment I chose a few token representations and ran tests on those. The next section will summarize the findings as general conclusions, while here I will report the results and give some observations for specific experiments. A summary of the results of the experiments are given in Table 1, along with a comparison of state-of-the-art results.

Network description	Top-1 Accuracy	Top-5 Accuracy
Logistic regression	4 %	14 %

Table 1: Summary of top-1 and top-5 accuracy results

4.1 Single-layer networks of varying kernel size

The first set of experiments I did were with a single convolutional layer in the network.

4.2 Single-layer networks with max-pooling layers

* max-pooling ‘localizes’ an object? maybe this is not necessary when doing scene recognition

4.3 Single-layer networks with rectifying linear units

4.4 Two-layer networks

4.5 Fine-tuning on PLACES-CNN

5 Conclusions

5.1 Network size and training time

5.2 Kernel size and receptive fields

5.3 Choosing hyperparameters

References

[1] Mark A Aizerman, E Braverman, and L Rozonoer. Theoretical foundations of the potential function method in pattern recognition learning. *Automation and Remote Control*, 25:821–837, 1964.

[2] Lei Jimmy Ba and Rich Caurana. Do deep nets really need to be deep? *arXiv preprint arXiv:1312.6184*, 2013. Eventually in NIPS.

[3] Lon Bottou. Large-scale machine learning with stochastic gradient descent. In *Proceedings of COMPSTAT'2010*, pages 177–186. Springer, 2010.

[4] Jeff Donahue, Yangqing Jia, Oriol Vinyals, Judy Hoffman, Ning Zhang, Eric Tzeng, and Trevor Darrell. Decaf: A deep convolutional activation feature for generic visual recognition. In *Proceedings of the International Conference on Machine Learning*, Beijing, China, 2014.

[5] Yangqing Jia, Evan Shelhamer, Jeff Donahue, Sergey Karayev, Jonathan Long, Ross Girshick, Sergio Guadarrama, and Trevor Darrell. Caffe: Convolutional architecture for fast feature embedding. *arXiv preprint arXiv:1408.5093*, 2014.

[6] Alex Krizhevsky, Ilya Sutskever, and Geoffrey E. Hinton. Imagenet classification with deep convolutional neural networks. In *Advances in neural information processing systems*, pages 1097–1105, 2012.

[7] Yann LeCun, B. Boser, J. S. Denker, D. Henderson, and L. D. Jackel. Backpropagation applied to handwritten zip code recognition y. LeCun. *Neural Computation*, 1(4):541–551, 1989.

[8] Yann LeCun, B Boser, J.S. Denker, D. Henderson, R.E. Howard, W. Hubbard, and L.D. Jackel. Handwritten digit recognition with a back-propagation network. *NIPS*, 1990.

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

[10] Yann LeCun, John S. Denker, Sara A. Solla, Richard E. Howard, and Lawrence D. Jackel. Optimal brain damage. In *NIPS*, volume 2, pages 598–605, 1989.

[11] Li Deng. The MNIST database of handwritten digit images for machine learning research [best of the web]. *IEEE Signal Processing Magazine*, 29(6):141–142, November 2012.

[12] Frank Rosenblatt. The perceptron: a probabilistic model for information storage and organization in the brain. *Psychological review*, 65(6):386, 1958.

[13] Olga Russakovsky, Jia Deng, Hao Su, Jonathan Krause, Sanjeev Satheesh, Sean Ma, Zhiheng Huang, Andrej Karpathy, Aditya Khosla, Michael Bernstein, and others. ImageNet large scale visual recognition challenge (ILSVRC). *arXiv preprint arXiv:1409.0575*, 2014.

[14] Jianxiong Xiao, Krista A. Ehinger, James Hays, Antonio Torralba, and Aude Oliva. SUN database: Exploring a large collection of scene categories. *International Journal of Computer Vision*, August 2014.

[15] Jianxiong Xiao, James Hays, Krista A. Ehinger, Aude Oliva, and Antonio Torralba. Sun database: Large-scale scene recognition from abbey to zoo. In *Computer vision and pattern recognition (CVPR), 2010 IEEE conference on*, pages 3485–3492. IEEE, 2010.

[16] Bolei Zhou, Agata Lapedriza, Jianxiong Xiao, Antonio Torralba, and Aude Oliva. Learning deep features for scene recognition using places database. In *NIPS*, 2014.