Annealed Pruning of Neural Network Connections

Implementing a new machine-learning technique with Keras

Brian Bartoldson
Florida State University
Department of Scientific Computing
bb11t@my.fsu.edu

Abstract

Keras is a Python module for building neural networks that are capable of solving artificial intelligence problems. Neural networks must be trained, and preventing *overfitting* means ensuring that a neural network's accuracy on training data is not significantly greater than the network's accuracy on general data. To prevent overfitting, Keras provides tools such as *Dropout*. I add to Keras the ability to use a new overfitting-prevention tool, *annealed pruning*, which reduces the size of the neural network as the network trains. Tests on the MNIST database show that annealed pruning is competitive with Dropout on both training time and accuracy.

Introduction

Artificial neural networks (ANNs) grew out of early models of biological neural networks and are now the state-of-the-art in a variety of machine learning problems. An ANN computes a function $f_w(x)$ that is parameterized by a weight vector w, and w is optimized via gradient descent. A concern of those who use neural networks is ensuring that the ANN does not overfit to the training data, $x_{training}$. Overfitting happens when the neural network is computing a function $f_w(x)$ that performs worse in the wild (on $x_{general}$) than it does while training (on $x_{training}$). For example, a neural network responsible for driving a car has each turn/acceleration/honk/etc. being determined by $f_w(x)$. Preventing overfitting would ensure that the car's performance in the real world is not worse than the car's performance on test tracks; i.e., the car would only be safe for $x_{general}$ if the car's ANN did not overfit to $x_{training}$.

This work seeks to evaluate annealed pruning, an overfitting prevention technique, by comparing it to Dropout, an overfitting prevention technique that facilitates record-setting performance on machine learning problems [3].

Background

Test Problem: MNIST Handwritten Digit Classification

The Mixed National Institute of Standards and Technology (MNIST) database is a collection of images of handwritten digits. A sample of images from this database is shown in Figure 1. Each image x_i is 256 (28x28) pixels and possesses a corresponding label y_i for the digit that is represented in the image. When an image is fed to the neural network, the neural network computes $f_w(x_i)$, and the loss/error is some function $L(f_w(x_i), y_i)$.

Figure 1: MNIST digit examples [5].

Overfitting is a concern when making a model to classify the MNIST training data, which makes it possible to compare the overfitting-prevention abilities of annealing and Dropout. The best performing algorithm for MNIST is a neural network that uses Dropout/DropConnect and correctly classifies 99.8% of handwritten digits, misclassifying 21/10,000 test images [6].

Neural Networks

A neural network is constructed from a set of neurons, each of which takes input values from the data or other neurons in the network. A neuron's input values are multiplied by weights, summed, added to a bias term, and then passed through a non-linear activation function σ. Using the MNIST example, which has pixel intensities as input values, a neuron behaves as shown in Figure 2.

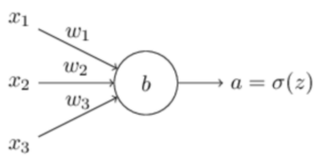

Figure 2: The input z to this neuron is the product of three pixel intensities and three weights plus the bias: $z = \sum_j w_j x_j + b$. The neuron's output a, assuming a logistic activation function σ, is $a = \sigma(z) = (1 + e^{-z})^{-1}$ [5].

A neural network typically has 3+ layers of neurons. As shown in Figure 3, these layers can be broken up into three types: input, hidden, and output. An input layer contains the data, a hidden layer contains neurons of the type shown in Figure 2, and the output layer contains the result of the input data passing through the entire neural network.

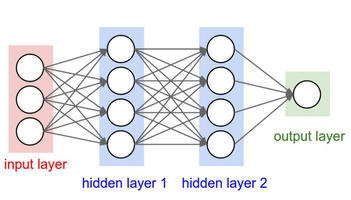

Figure 3: A neural network with four layers [4].

Each connection between neurons has an associated weight, as shown in Figure 2. The neural network is trained by modifying each of these weights via gradient descent. Basically, after the neural network computes $f_w(x_i)$, the loss $L(f_w(x_i), y_i)$ is calculated, and each weight w_j is updated with $\frac{\partial L}{\partial w_j}$ such that the loss on the training example x_i is reduced.

This training procedure leads to a neural network $f_w(x)$ that minimizes $L(f_w(x_{training}), y_{training})$, but $f_w(x)$ does not necessarily minimize $L(f_w(x_{general}), y_{general})$. $f_w(x)$ will only minimize $L(f_w(x_{general}), y_{general})$ if we prevent overfitting to the idiosyncrasies of $x_{training}$ and $y_{training}$.

Dropout

Dropout is a tool used during neural network training that reduces overfitting. Dropout randomly blocks a percentage of connections during training. The set of connections that is blocked changes as training proceeds. Figure 4 illustrates a network with a Dropout rate of 50%, and we can think of the resulting sub-network as computing and training $f_{w_1}(x)$. Since different, random groups of three neurons are shut down, the neural network computes different functions $f_{w_k}(x)$. Once training is finished, all connections are turned on. The output $f_w(x)$ behaves similarly to a vote among the many trained networks $f_{w_k}(x)$, and voting helps prevent weighting idiosyncrasies of any particular x_i too heavily (overfitting).

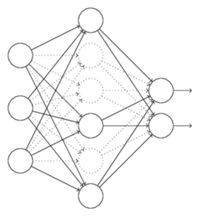

Figure 4: No information passes through faded connections [5].

Annealed Pruning of Connections

Algorithm

While Dropout pauses certain connections, annealed pruning permanently removes connections until only a specified number of parameters (weights/connections) remains. Annealed pruning proceeds as shown in Figure 5, in which $\mathcal{B}$ represents what this poster has called w.

Algorithm 1 Feature Selection with Annealing (FSA)

Input: Training examples $\{(\boldsymbol{x}_i, y_i)\}_{i=1}^N$.
Output: Trained classifier parameter vector $\boldsymbol{\beta}$.

1: Initialize $\boldsymbol{\beta} = 0$.
2: **for** e=1 to N^{iter} **do**
3: Update $\boldsymbol{\beta} \leftarrow \boldsymbol{\beta} - \eta \frac{\partial L(\boldsymbol{\beta})}{\partial \boldsymbol{\beta}}$
4: Keep only the M_e variables with highest $|\beta_j|$ and renumber them $1, ..., M_e$.
5: **end for**

Figure 5: A general algorithm for gradient descent with annealing [1].

"Key ideas in the algorithm design are: a) using an annealing plan to lessen the greediness in reducing the dimensionality from M to k, and b) gradually removing the most irrelevant variables to facilitate computation" [1]. When a neural network's size is restricted, it can't as easily afford to model the aspects of the data that aren't helpful, like idiosyncratic noise. This helps prevent overfitting.

Keras Implementation

Keras is a library that simplifies the construction of neural networks in Python. Keras provides users with the ability to execute custom functions (known as callbacks) while a network trains. I constructed a custom callback that takes layer numbers as arguments and performs annealing on those layers of the neural network that the callback is issued on.

The neural network that classifies MNIST digits with annealing is ANN_A. The starting point for ANN_A was ANN_D, which is mnist_cnn.py from the Keras GitHub repository [2]. ANN_D uses Dropout in its final two layers, and I replaced Dropout with annealing to make ANN_A. The annealing schedule I used for ANN_A removes 0.5% of the connections in a layer every 50 weight updates.

Results

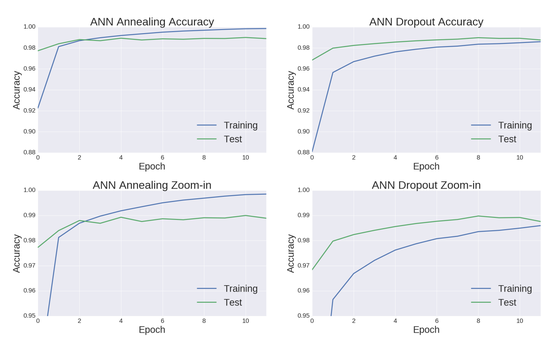

Figure 6: Training data and test data accuracies at each epoch for both ANN_A and ANN_D. Each epoch includes 60,000 training images.

The accuracies of ANN_A and ANN_D are plotted in Figure 6 below. The final accuracies are: $ANN_A = 98.89\%$, and $ANN_D = 98.76\%$. Thus, annealing beats Dropout by 0.13 percentage points, or 13 images.

ANN_A and ANN_D trained for 12 epochs, seeing 60,000 training images each epoch. Both networks took fewer than 7 minutes to train on a K1100M GPU, with ANN_D training faster than ANN_A by about 10 seconds, or 2% of total training time. If the annealing function can be modified such that annealed weights are absent instead of masked, then the reduced number of computations may make ANN_A quicker to train.

Conclusions

Annealed pruning, a new overfitting-prevention technique, removes connections between layers in a neural network to make the function that the neural network computes more parsimonious. Python and Keras were used to compare annealed pruning to Dropout, a separate technique for preventing overfitting that has had empirical success. Preliminary results on the MNIST handwritten digit classification problem showed that annealed pruning led to slightly higher classification accuracy (by 0.13 percentage points, or 13 images) at the cost of a slightly longer training time ($\approx 2\%$ slower).

Forthcoming Research

- Delete connections instead of masking connections to reduce computations.
- ANN_D does not reach 99.25% accuracy, which it should be doing according to the Keras GitHub page (Keras, 2016).
- Push ANN_A and ANN_D closer to state-of-the-art accuracy.
- Implement the prescribed annealing schedule from [1].

References

[1] A. Barbu et al. Feature selection with annealing for computer vision and big data learning.

[2] F. Chollet. mnist_cnn.py. https://github.com/fchollet/keras/blob/master/examples/mnist_cnn.py, 2016.

[3] G. Hinton et al. Improving neural networks by preventing co-adaptation of feature detectors.

[4] A. Karpathy. Cs231n: Convolutional neural networks for visual recognition. http://cs231n.stanford.edu/, 2016.

[5] Michael A. Nielsen. Neural networks and deep learning. 2015.

[6] L. Wan et al. Regularization of neural network using dropconnect.

Acknowledgements

Dr. Erlebacher and Dr. Barbu assisted and inspired my work on this project.