High Capacity Neural Network

Domenic Donato

2016-10-06

1 Introduction

TODO(domenic): Write this section

2 Related Literature

TODO(domenic): Write this section

3 Model

3.1 Forward Computation

The forward computation of a fully connected feedforward neural network with N layers can be described as follows:

$n \in N$
$\{N \in \mathbb{N} | N > 0\}$
$t_n(x)$ is a transform function
$a_n(x)$ is an activation function
x_n is the vector of inputs to layer n
x_N is the initial input given to the network
$x_{n-1} = a_n(t_n(x_n))$ is how the next layers input is computed
$\hat{y} = x_1$ is the output vector

The standard transform function, $t_n(x)$, is:

$W_n \in \mathbb{R}^2$
$b_n \in \mathbb{R}$

$$t_n(x) = f_n(x) \tag{1}$$

$$f_n(x) = xW_n + b_n \tag{2}$$

The transform function proposed in this paper is:

$Y_n \in \mathbb{R}^2$
$c_n \in \mathbb{R}$

$$t_n(x) = f_n(x) \cdot g_n(x) \tag{3}$$

$$g_n(x) = xY_n + c_n \tag{4}$$

Rather than using the standard transform, (1), a quadratic transform, (3), is performed. The quadratic transform is a superset of the standard design, which becomes apparent when $Y_n \in \{0, ..., 0\}$ and $c_n \in \{1, ..., 1\}$. Using a

quadratic transformation enables the neural network to represent any bounded degree polynomial over an infinite domain using a finite number of nodes. The standard design would require an infinite number of nodes to do the same.

3.2 Backward Computation

The total network error given a cost function $c(y, \hat{y})$ is:

$$E = \sum_i c(y_i, \hat{y_i}) \tag{5}$$

Back propagation was used to train the networks. The key thing about back propagation is that its a dynamic program and caches part of the gradient calculation so it can be reused upstream.

C_n is the matrix of cached calculations at level n
$C_1 = c'(y, \hat{y})$ the cache is seeded with the derivative of the cost function

$$C_{n+1} = a'_n(x_n) \cdot (C_n \cdot t'_n(x_n)^\top) \tag{6}$$

The gradient of parameter θ_n is calculated as follows:

$$\nabla\theta_n = \frac{\partial t_n(x)}{\partial \theta_n} C_n \tag{7}$$

Quadratic Transform The interesting part about using a quadratic transform function, (3), is that the derivative computation now involves a product, which means that in addition to the standard chain rule, the product rule also needs to be used. The derivative of the quadratic transform function is

$$t'_n(x) = Y_n(xW_n + b_n) + W_n(xY_n + C_n) \tag{8}$$

and the gradients of the parameters, $\nabla\theta_n$, are

$$\nabla W_n = x_n^\top (g_n(x_n) \cdot C_n) \tag{9}$$

$$\nabla Y_n = x_n^\top (f_n(x_n) \cdot C_n) \tag{10}$$

$$\nabla b_n = g_n(x_n) \cdot C_n \tag{11}$$

$$\nabla c_n = f_n(x_n) \cdot C_n \tag{12}$$

3.3 Parameter Updating

When it comes to training a model, computing the gradients of the parameters is only a prerequisite to the real issue at hand, which is modifying the parameters so that they better represent the data.

Largest Gradients Filter This update strategy was inspired by dropout (citation needed), but rather than randomly dropping nodes during the forward pass, it only updates $X\%$ of parameters based on the size of their gradient during the backwards pass.

Adaptive Gradient Updater This updater analyzes the path of each parameters gradient and uses that to determine how to update the parameter.

4 Experiments

TODO(domenic): Write this section

A squared error cost function was used for the experiments in this paper.

$$c(y, \hat{y}) = \frac{(y - \hat{y})^2}{2} \tag{13}$$

$$c'(y, \hat{y}) = y - \hat{y} \tag{14}$$

5 Conclusion

TODO(domenic): Write this section