Landon Buell

Quioyan Yu

Network Optimizers

22 June 2020

Optimization Methods for Deep Neural Networks

Optimizer Index

1. Stochastic Gradient Descent
   1. Learning Rate
   2. Momentum
   3. Nesterov Momentum
2. AdaGrad
3. RMS-Prop
4. ADAM
5. AdaDelta
6. Adamax
7. Nadam
8. Ftrl
9. L-BFGS

Notes on Model Structure

All optimizers are applied to linear, sequential, deep neural network models composed of *L* layers. In every case, a model is assumed to be composed of an input layer (layer 0), an output layer (layer L-1), and L-2 hidden layers. These hidden layers can be Densely Connected, Sparely Connected, Convolutional, Pooling, or some variant thereof. Where applicable, each layer, kernel or output is also assumed to be composed of a finite, integer number of units. Each layer transforms a finite-dimensional tensor-like object into another finite-dimensional tensor-like object. Within each layer, there is a finite set of parameters, that contribute to the final output of the neural network.

Optimizer Functionality

Optimizers are algorithms that are used to minimize the error of the output of a network model by iteratively reducing the value of some cost function to approach a local or global minimum. This is often done by concatenating all of the parameters from each layer into a single object, θ. The value of the cost function is then the difference between the networks given output and the expected output, as a result of the parameters in the network. Thus, we deem the cost function **J** to be dependent on the elements in the object **θ**. The optimizer than seeks to adjust the parameters in θ to achieve a locally or global minimum.

1. Stochastic Gradient Descent
   1. Most commonly used back-propagation method.
   2. Numerically Estimate Gradient of **J** using a subset of samples called a *mini-batch*
      1. Use average gradient over samples in mini-batch
         1. Produced a tensor-like object, same shape as θ
         2. Add *negative* gradients element-wise to θ
      2. Adjust parameters in θ to reduce the value outputted from J
      3. Process is repeated many times, each time producing a smaller value for **J**
      4. Allows to find a local minimum condition – never guaranteed to final global min
      5. Can be unreliable, unstable, lead to vanishing/exploding gradients
   3. Learning Rate parameter – very important
      1. Scalar multiple to multiple gradient vector by
         1. Can be used to accelerate/ slow learning
         2. Small values require many iterations, more succinct learning
         3. Larger values require few iterations potentially volatile
      2. Can be a static scalar, called *constant learning rate*
         1. Can decay based on number of iterations
         2. Can be functional depending on the value of elements in gradient
   4. Momentum Parameter
      1. Regular SGD can be slow, even with a proper learning rate.
      2. Momentum can accelerate learning even with messy or unstable gradient values
      3. Uses a “momentum” *p* and “initial velocity” *v* to add to update
      4. Added to gradient times learning rate to update θ
   5. Nesterov’s Momentum
2. AdaGrad
3. RMS-Prop
4. ADAM
5. AdaDelta
6. AdaMax
7. N-ADAM
8. FTRL
9. L-BFGS