Foundations of Machine Learning
Neural Networks

Kate Farrahi

ECS Southampton

December 8, 2020

Gradient Descent

repeat until convergence:

$$w \leftarrow w - \eta \frac{\partial J}{\partial w} \tag{1}$$

where w is a multidimensional vector representing all of the weights in the model and η is the learning rate.

In order to get Gradient Descent working in practice, we need to compute $\frac{\partial J}{\partial w}$. For neural networks, there are 2 stages to this computation, (1) the forward pass and (2) the backwards pass.

Batch Gradient Descent

begin initialize w, Th, η, $m = 0$, $r = 0$

    do $r \leftarrow r + 1$ **(increment epoch)**

        $m \leftarrow 0$; $\Delta w \leftarrow 0$

        do $m \leftarrow m + 1$

            $x^m \leftarrow$ **selected pattern**

            $\Delta w \leftarrow \Delta w - \eta \frac{\partial J}{\partial w}$

        until $m = M$

        $w \leftarrow w + \Delta w$

    until $J(w) < Th$

return w

end

Stochastic Gradient Descent

begin initialize w, Th, η, $m = 0$, $r = 0$

  do $r \leftarrow r + 1$ **(increment epoch)**

    $m \leftarrow 0$

    do $m \leftarrow m + 1$

      $x^m \leftarrow$ **selected pattern (randomly)**

      $w \leftarrow w - \eta \frac{\partial J}{\partial w}$

    until $m = M$

  until $J(w) < Th$

return w

end

Backpropagation: The Forward Pass

We need to compute the multilayer perceptron outputs y_k in order to compute the cost function $J(w)$. For a 3 layer network:

$$a^{(1)} = f(w^{(1)}x + b^{(1)}) \tag{2}$$

$$\hat{y} = a^{(2)} = f(w^{(2)}a^{(1)} + b^{(2)}) \tag{3}$$

Backpropagation: The Backwards Pass

In order to update the weights, we need to compute the gradient of the cost function with respect to each of the weights. Let us consider the quadratic cost function as follows:

$$J(w) = \frac{1}{2} \sum_{k=1}^{M} (\hat{y}_k - y_k)^2 \tag{4}$$

Note, we are considering the case of $\hat{y}_k = a_k^{(2)}$.

To compute the weight updates, we compute the derivative of the cost function with respect to each weight. The derivative of J with respect to the weights at the output layer can be computed as follows:

$$\frac{\partial J}{\partial w_{kj}^{(2)}} = \frac{\partial J}{\partial a_k^{(2)}} \frac{\partial a_k^{(2)}}{\partial z_k^{(2)}} \frac{\partial z_k^{(2)}}{\partial w_{kj}^{(2)}} \tag{5}$$

Backpropagation: The Backwards Pass

Let us assume the quadratic cost function and the sigmoid activation function.

$$\frac{\partial J}{\partial a_k^{(2)}} = (a_k^{(2)} - y_k) \tag{6}$$

If we assume a sigmoid activation function then $a_k^{(2)} = \frac{1}{1+e^{-z_k^{(2)}}}$

$$\frac{\partial a_k^{(2)}}{\partial z_k^{(2)}} = a_k^{(2)}(1 - a_k^{(2)}) \tag{7}$$

$$\frac{\partial z_k^{(2)}}{\partial w_{kj}^{(2)}} = a_k^{(1)} \tag{8}$$

Since $z_k^{(2)} = \sum_j w_{kj}^{(2)} a_k^{(1)}$ therefore,

$$\frac{\partial J}{\partial w_{kj}^{(2)}} = (a_k^{(2)} - y_k) a_k^{(2)} (1 - a_k^{(2)}) a_k^{(1)} \tag{9}$$

To compute the weight updates with respect to the input layer:

$$\frac{\partial J}{\partial w_{ji}^{(1)}} = \frac{\partial J}{\partial a_k^{(1)}} \frac{\partial a_k^{(1)}}{\partial z_k^{(1)}} \frac{\partial z_k^{(1)}}{\partial w_{ji}^{(1)}} \tag{10}$$

Again, this is the derivative of the output of the neuron w.r.t. the sigmoid activation function. Since $a_k^{(1)} = \frac{1}{1+e^{-z_k^{(1)}}}$, therefore

$$\frac{\partial a_k^{(1)}}{\partial z_k^{(1)}} = a_k^{(1)}(1 - a_k^{(1)}) \tag{11}$$

Since $z_k^{(1)} = \sum_i x_i w_{ji}^{(1)}$

$$\frac{\partial z_k^{(1)}}{\partial w_{ji}^{(1)}} = x_i \tag{12}$$

Backpropagation: The Backwards Pass

Again, we can work out these three partial derivatives as follows:

$$\frac{\partial J}{\partial a_k^{(1)}} = \sum_{k=1}^{M} \frac{\partial J_{y_k}}{\partial a_k^{(1)}} = \left(\sum_{k=1}^{M} \frac{\partial J_{y_k}}{\partial a_k^{(2)}} \frac{\partial a_k^{(2)}}{\partial z_k^{(2)}} \frac{\partial z_k^{(2)}}{\partial a_k^{(1)}}\right) \tag{13}$$

where $\frac{\partial z_k^{(2)}}{\partial a_k^{(1)}} = w_{kj}^{(2)}$ since $z_k^{(2)} = \sum_j w_{kj}^{(2)} a_k^{(1)}$, however, this time we take the derivative with respect to $a_k^{(1)}$ instead of $w_{kj}^{(2)}$.

Backpropagation in General

- We define the error of the neuron j in layer l by

$$\delta_j^l = \frac{\partial J}{\partial z_j^l} \tag{14}$$

Backpropagation in General

- We define the error of the neuron j in layer l by

$$\delta_j^l = \frac{\partial J}{\partial z_j^l} \tag{14}$$

- Then

$$\delta^L = \Delta_a J \odot \sigma'(z^L) \tag{15}$$

Backpropagation in General

- We define the error of the neuron j in layer l by

$$\delta_j^l = \frac{\partial J}{\partial z_j^l} \tag{14}$$

- Then

$$\delta^L = \Delta_a J \odot \sigma'(z^L) \tag{15}$$

- For the case of a MSE cost function:

Backpropagation in General

- We define the error of the neuron j in layer l by

$$\delta_j^l = \frac{\partial J}{\partial z_j^l} \tag{14}$$

- Then

$$\delta^L = \Delta_a J \odot \sigma'(z^L) \tag{15}$$

- For the case of a MSE cost function:
- $\delta^L = (a^L - y) \odot \sigma'(z^L)$

Backpropagation in General

- We define the error of the neuron j in layer l by

$$\delta_j^l = \frac{\partial J}{\partial z_j^l} \tag{14}$$

- Then

$$\delta^L = \Delta_a J \odot \sigma'(z^L) \tag{15}$$

- For the case of a MSE cost function:
- $\delta^L = (a^L - y) \odot \sigma'(z^L)$

Backpropagation in General

$$\delta^L = \Delta_a J \odot \sigma'(z^L) \tag{16}$$

Backpropagation in General

$$\delta^L = \nabla_a J \odot \sigma'(z^L) \tag{16}$$

$$\delta^l = ((w^{l+1})^T \delta^{l+1}) \odot \sigma'(z^l) \tag{17}$$

Backpropagation in General

$$\delta^L = \nabla_a J \odot \sigma'(z^L) \tag{16}$$

$$\delta^l = ((w^{l+1})^T \delta^{l+1}) \odot \sigma'(z^l) \tag{17}$$

$$\frac{\partial J}{\partial w^l_{jk}} = a^{l-1}_k \delta^l_j \tag{18}$$

Backpropagation in General

$$\delta^L = \nabla_a J \odot \sigma'(z^L) \tag{16}$$

$$\delta^l = ((w^{l+1})^T \delta^{l+1}) \odot \sigma'(z^l) \tag{17}$$

$$\frac{\partial J}{\partial w^l_{jk}} = a^{l-1}_k \delta^l_j \tag{18}$$

$$\frac{\partial J}{\partial b^l_j} = \delta^l_j \tag{19}$$

Gradient Descent with Momentum

- Allows the network to learn more quickly than standard gradient descent
- Learning with momentum reduces the variation in overall gradient directions to speed learning
- Learning with momentum is given by
  $w(m+1) = w(m) + (1-\alpha)\Delta w_{bp}(m) + \alpha \Delta w(m-1)$

  $\Delta w(m) = w(m) - w(m-1)$

  where $\Delta w_{bp}(m)$ is the change in weight given by the backpropagation algorithm.

Practical Considerations – Initializing Weights

- All of the weights should be randomly initialized to a small random number close to zero but not identically zero.
- If they're all set to zero, they will all undergo the exact same parameter updates during backprop

- There will be no source of asymmetry if the weights are all initialized to be the same
- Calibrating the variances to $\frac{1}{\sqrt{n}}$ ensures that all neurons in the network initially have approximately the same output distribution and empirically improves the rate of convergence.

- It is common to initialize all of the biases to zero or a small number such as 0.01.

Practical Considerations – Learning Rates

- Plot the cost function J as a function of iterations (epochs)
- J should decrease after every iteration on your training data!
- If J is increasing then something is wrong, it is likely that the learning rate is too high

- Standard test for convergence, $\Delta J < Th$ where $Th = 10^{-3}$
- Note, it is difficult to choose a threshold. Looking at the overall plot of the cost function vs. iterations on data is always most informative.

L1 Regularization

In general

Cost function = Loss + Regularization Term

In L1 regularization, the absolute value of the weights are penalized which can push the weights to reduce to zero. This is useful if we are trying to compress the model.

For L1:

$$\gamma \sum_{i=1}^{n} ||w_i|| \tag{20}$$

L2 Regularization

L2 Regularization is also known as *weight decay* as it forces the weights to decay towards zero.

For L2:

$$\gamma \sum_{i=1}^{n} ||w_i||^2 \tag{21}$$