Foundations of Machine Learning
Neural Networks

ECS Southampto

December 8, 202

repeat until convergence:

$$w \quad w \quad \eta \frac{\partial J}{} \qquad (1)$$

wher
weig

In order to get Gradient Descent working in practi compute $\frac{\partial J}{\partial w}$. For neural networks, there are 2 st computation, (1) the forward pass and (2) the bac

Batch Gradient Descent

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

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

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

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)**

$m \quad 0$

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 co

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

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:

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

Note k k

To compute the weight updates, we compute the cost function with respect to each weight. The deri with respect to the weights at the output layer can be com 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 as ... $\frac{1}{\ldots^{-z_k^{(2)}}}$

$$\frac{\ldots}{\partial z_k^{(2)}}$$

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

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:

[illegible] (10)

Backpropagation: The Backwards Pass

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

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

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

Backpropagation: The Backwards Pass

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

where $\frac{\partial z_k^{(2)}}{\partial a_k^{(1)}} = w_{kj}^{(2)}$ since $z_k^{(2)} = \sum_j w_k^{(2)} \quad {}^{(1)}$

take the derivative with respect to $a_k^{(1)}$

Backpropagation in General

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

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

Backpropagation in General

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

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

-

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

Backpropagation in General

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

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

-

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

-

- For the case of a MSE cost function:

Backpropagation in General

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

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

-

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

- 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

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

-

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

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

Backpropagation in General

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

Backpropagation in General

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

Backpropagation in General

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

$$\frac{\partial J}{\partial w^{l}_{jk}} = a_k^{l-1} \delta_j^{l} \tag{18}$$

Backpropagation in General

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

$$\frac{\partial J}{\partial w^l_{jk}} = a_k^{l-1} \delta_j^l \tag{18}$$

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

Gradient Descent with Momentum

- Allows the network to learn more quickly than standard gradient descent
-
-

bp)

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

where $\Delta w_{bp}(m)$ is the change in weight gi
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

-

- Calibrating the variances to $\frac{1}{\sqrt{n}}$ e
  network initially have approximately the
  distribution and empirically improves th

- 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!
-

- 0^{-3}
- Note, it is difficult to choose a threshold. Look overall plot of the cost function vs. iterations always most informative.

L1 Regularization

In general

Cost function = Loss + Regularization Term

In L1 re

pena

usef

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 weig

For L

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