Deep Learning

Vijaya Saradhi

IIT Guwahati

Fri, 25th Sept 2020

Multi-layer Perceptrons

Summary

- $w_{ji}(n+1) = w_{ji}(n) - \eta \frac{\partial \mathcal{E}(n)}{\partial w_{ji}(n)}$
- $\Delta w_{ji}(n) = -\eta \frac{\partial \mathcal{E}(n)}{\partial w_{ji}(n)}$
- $\Delta w_{ji}(n) = \eta \delta_j(n) y_i(n)$

$$\delta_j(n) = \begin{cases} e_j(n)\phi_j'(v_j(n)) & \text{if } j \text{ is output neuron} \\ \phi_j'(v_j(n)) \sum_k \delta_k(n) w_{kj}(n) & \text{if } j \text{ is a hidden neuron} \end{cases}$$

Multi-layer Perceptrons

Local Gradient - Hidden Neuron

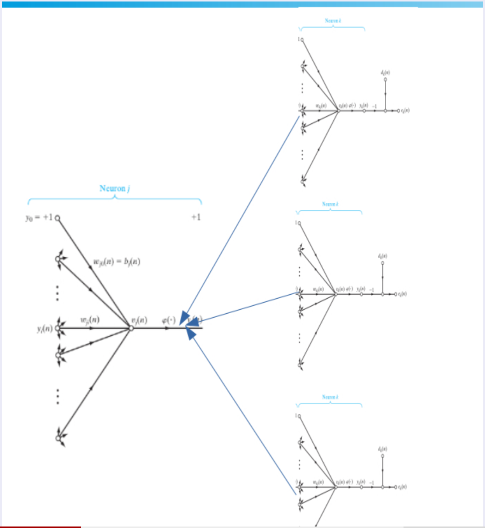
- Output emitted by neuron j: Local gradient is computed using chain rule as:

$$\begin{aligned}(output)\delta_j(n) &= \frac{\partial \mathcal{E}(n)}{\partial e_j(n)} \frac{\partial e_j(n)}{\partial y_j(n)} \frac{\partial y_j(n)}{\partial v_j(n)} \\ (hidden)\delta_j(n) &= \frac{\partial \mathcal{E}(n)}{\partial y_j(n)} \frac{\partial y_j(n)}{\partial v_j(n)} \\ &= \frac{\partial \mathcal{E}(n)}{\partial y_j(n)} \phi_j'(v_j(n))\end{aligned}$$

- To be computed: $\frac{\partial \mathcal{E}(n)}{\partial y_j(n)}$
- In turn depends on error made by all neurons in (right) layer of j^{th} neuron

Multi-layer Perceptrons

Error Back progpagation



Multi-layer Perceptrons

Where is Back propagation of error?

- The error of made by all neurons to which j is connected need to be minimized.
- That is
$$\mathcal{E}(n) = \frac{1}{2}\sum_{k} e_k^2(n)$$
- Draw figure 4.4 by extending the idea.

Multi-layer Perceptrons

Local Gradient

- We need: $\frac{\partial \mathcal{E}(n)}{\partial y_j(n)}$

$$
\begin{aligned}
\frac{\partial \mathcal{E}(n)}{\partial y_j(n)} &= \sum_k e_k(n) \frac{\partial e_k(n)}{\partial y_j(n)} \\
&= \sum_k e_k(n) \frac{\partial e_k(n)}{\partial v_k(n)} \frac{\partial v_k(n)}{\partial y_j(n)}
\end{aligned}
$$

Multi-layer Perceptrons

Local Gradient

- To compute first term we use: $e_k(n) = d_k(n) - \phi_k(v_k(n))$
  $\frac{\partial e_k(n)}{\partial v_k(n)} = \phi'_k(v_k(n))$

$$\begin{aligned}\frac{\partial \mathcal{E}(n)}{\partial y_j(n)} &= \sum_k e_k(n)\frac{\partial e_k(n)}{\partial y_j(n)} \\ &= \sum_k e_k(n)\frac{\partial e_k(n)}{\partial v_k(n)}\frac{\partial v_k(n)}{\partial y_j(n)}\end{aligned}$$

Multi-layer Perceptrons

Local Gradient

- To compute first term we use: $e_k(n) = d_k(n) - \phi_k(v_k(n))$
  $\frac{\partial e_k(n)}{\partial v_k(n)} = \phi_k'(v_k(n))$

$$\begin{aligned}\frac{\partial \mathcal{E}(n)}{\partial y_j(n)} &= \sum_k e_k(n) \frac{\partial e_k(n)}{\partial y_j(n)} \\ &= \sum_k e_k(n) \phi_k'(v_k(n)) \frac{\partial v_k(n)}{\partial y_j(n)}\end{aligned}$$

Multi-layer Perceptrons

Local Gradient

- To compute second term we use: $v_k(n) = \sum_{j=0}^{m} w_{kj}(n) y_j(n)$

$$\frac{\partial v_k(n)}{\partial y_j(n)} = w_{kj}(n)$$

$$\begin{aligned}\frac{\partial \mathcal{E}(n)}{\partial y_j(n)} &= \sum_k e_k(n) \frac{\partial e_k(n)}{\partial y_j(n)} \\ &= \sum_k e_k(n) \phi'_k(v_k(n)) \frac{\partial v_k(n)}{\partial y_j(n)}\end{aligned}$$

Multi-layer Perceptrons

Local Gradient

- To compute second term we use: $v_k(n) = \sum_{j=0}^{m} w_{kj}(n) y_j(n)$

$$\frac{\partial v_k(n)}{\partial y_j(n)} = w_{kj}(n)$$

$$\begin{aligned}
\frac{\partial \mathcal{E}(n)}{\partial y_j(n)} &= \sum_k e_k(n) \frac{\partial e_k(n)}{\partial y_j(n)} \\
&= \sum_k e_k(n) \phi'_k(v_k(n)) w_{kj}(n)
\end{aligned}$$

Multi-layer Perceptrons

Local Gradient

- The complete derivative is:
$\frac{\partial v_k(n)}{\partial y_j(n)} = w_{kj}(n)$

$$\begin{aligned}\frac{\partial \mathcal{E}(n)}{\partial y_j(n)} &= \sum_k e_k(n) \frac{\partial e_k(n)}{\partial y_j(n)} \\ &= \sum_k e_k(n) \phi'_k(v_k(n)) w_{kj}(n) \\ &= \sum_k e_k(n) \phi'_k(v_k(n)) w_{kj}(n) \\ &= \sum_k \delta_k(n) w_{kj}(n)\end{aligned}$$

- For all the k^{th} neurons in the forward layer that connect to j^{th} neuron

Multi-layer Perceptrons

Local Gradient - Hidden Neuron

- Output emitted by neuron j: Local gradient is computed using chain rule as:

$$
\begin{aligned}
(hidden)\delta_j(n) &= \frac{\partial \mathcal{E}(n)}{\partial y_j(n)} \frac{\partial y_j(n)}{\partial v_j(n)} \\
&= \frac{\partial \mathcal{E}(n)}{\partial y_j(n)} \phi_j'(v_j(n)) \\
&= \phi_j'(v_j(n)) \sum_k \delta_k(n) w_{kj}(n)
\end{aligned}
$$

Multi-layer Perceptrons

Summary

- $w_{ji}(n+1) = w_{ji}(n) - \eta \frac{\partial \mathcal{E}(n)}{\partial w_{ji}(n)}$
- $\Delta w_{ji}(n) = -\eta \frac{\partial \mathcal{E}(n)}{\partial w_{ji}(n)}$
- $\Delta w_{ji}(n) = \eta \delta_j(n) y_i(n)$

$$\delta_j(n) = \begin{cases} e_j(n)\phi_j'(v_j(n)) & \text{if } j \text{ is output neuron} \\ \phi_j'(v_j(n)) \sum_k \delta_k(n) w_{kj}(n) & \text{if } j \text{ is a hidden neuron} \end{cases}$$

Multi-layer Perceptrons

Activation Function: Sigmoid

$$\phi_j(v_j(n)) = \frac{1}{1+exp(-av_j(n))} \quad a > 0$$

$$\phi_j'(v_j(n)) = \frac{a\ exp(-a\ v_j(n))}{[1+exp(-av_j(n))]^2} \quad a > 0$$

We know: $y_j(n) = \phi_j(v_j(n))$

$$\phi_j'(v_j(n)) = a\ y_j(n)[1 - y_j(n)]$$

$\phi_j'(v_j(n))$ is expressed in terms of j^{th} neuron's output $y_j(n)$

Multi-layer Perceptrons

Update rule - output layer

- $\phi_j'(v_j(n)) = a\, y_j(n)[1 - y_j(n)]$

Multi-layer Perceptrons

Update rule - output layer

- $\phi_j'(v_j(n)) = a\, y_j(n)[1 - y_j(n)]$
- In the output layer let $y_j(n) = o_j(n)$

Multi-layer Perceptrons

Update rule - output layer

- $\phi_j'(v_j(n)) = a\, y_j(n)[1 - y_j(n)]$
- In the output layer let $y_j(n) = o_j(n)$
- $\phi_j'(v_j(n)) = a\, o_j(n)[1 - o_j(n)]$

Multi-layer Perceptrons

Update rule - output layer

- $\phi_j'(v_j(n)) = a\ y_j(n)[1 - y_j(n)]$
- In the output layer let $y_j(n) = o_j(n)$
- $\phi_j'(v_j(n)) = a\ o_j(n)[1 - o_j(n)]$

$$\delta_j(n) = \begin{cases} e_j(n)\phi_j'(v_j(n)) & \text{if } j \text{ is output neuron} \\ \phi_j'(v_j(n))\sum_k \delta_k(n) w_{kj}(n) & \text{if } j \text{ is a hidden neuron} \end{cases}$$

Multi-layer Perceptrons

Update rule - output layer

- $\phi_j'(v_j(n)) = a\, y_j(n)[1 - y_j(n)]$
- In the output layer let $y_j(n) = o_j(n)$
- $\phi_j'(v_j(n)) = a\, o_j(n)[1 - o_j(n)]$

$$\delta_j(n) = \begin{cases} e_j(n) a\, o_j(n)[1 - o_j(n)] & \text{if } j \text{ is output neuron} \\ \phi_j'(v_j(n)) \sum_k \delta_k(n) w_{kj}(n) & \text{if } j \text{ is a hidden neuron} \end{cases}$$

Multi-layer Perceptrons

Update rule - hidden layer

- $\phi_j'(v_j(n)) = a\, y_j(n)[1 - y_j(n)]$

Multi-layer Perceptrons

Update rule - hidden layer

- $\phi_j'(v_j(n)) = a\, y_j(n)[1 - y_j(n)]$

$$\delta_j(n) = \begin{cases} a[d_j(n) - o_j(n)]o_j(n)[1 - o_j(n)] & \text{if } j \text{ is output neuron} \\ \phi_j'(v_j(n)) \sum_k \delta_k(n) w_{kj}(n) & \text{if } j \text{ is a hidden neuron} \end{cases}$$

Multi-layer Perceptrons

Update rule - hidden layer

- $\phi_j'(v_j(n)) = a\, y_j(n)[1 - y_j(n)]$

$$\delta_j(n) = \begin{cases} a[d_j(n) - o_j(n)]o_j(n)[1 - o_j(n)] & \text{if } j \text{ is output neuron} \\ a\, y_j(n)[1 - y_j(n)]\sum_k \delta_k(n) w_{kj}(n) & \text{if } j \text{ is a hidden neuron} \end{cases}$$

Multi-layer Perceptrons

Summary - Sigmoid function

- $w_{ji}(n+1) = w_{ji}(n) - \eta \frac{\partial \mathcal{E}(n)}{\partial w_{ji}(n)}$
- $\Delta w_{ji}(n) = -\eta \frac{\partial \mathcal{E}(n)}{\partial w_{ji}(n)}$
- $\Delta w_{ji}(n) = \eta \delta_j(n) y_i(n)$

$$\delta_j(n) = \begin{cases} e_j(n)\phi_j'(v_j(n)) & \text{if } j \text{ is output neuron} \\ \\ \phi_j'(v_j(n)) \sum_k \delta_k(n) w_{kj}(n) & \text{if } j \text{ is a hidden neuron} \end{cases}$$

Multi-layer Perceptrons

Summary - Sigmoid function

- $w_{ji}(n+1) = w_{ji}(n) - \eta \frac{\partial \mathcal{E}(n)}{\partial w_{ji}(n)}$
- $\Delta w_{ji}(n) = -\eta \frac{\partial \mathcal{E}(n)}{\partial w_{ji}(n)}$
- $\Delta w_{ji}(n) = \eta \delta_j(n) y_i(n)$

$$\delta_j(n) = \begin{cases} a[d_j(n) - o_j(n)] o_j(n)[1 - o_j(n)] & \text{if } j \text{ is output neuron} \\ a\, y_j(n)[1 - y_j(n)] \sum_k \delta_k(n) w_{kj}(n) & \text{if } j \text{ is a hidden neuron} \end{cases}$$

Multi-layer Perceptrons

Stopping Criteria

- Euclidean norm of the gradient vector reaches a sufficiently small gradient threshold
- The absolute rate of change in the average squared error per epoch is sufficiently small.
- η is not further optimized as explained in the Cauchy's gradient descent method

Multi-layer Perceptrons

Complete Algorithm

Initialize Pick weights and threshold from uniform distribution whose mean is zero and variance is some condition

Training Examples For each sample, perform forward and backward computations

Forward computation Compute $v_j^\ell(n)$

$$v_j^\ell(n) = \sum_i w_{ji}^\ell y_i^{\ell-1}(n)$$

Multi-layer Perceptrons

Complete Algorithm

Backward Computations Computing δ's

$$\delta_j^{\ell}(n) = \begin{cases} e_j^L(n)\phi_j'^L(v_j^L(n)) & \text{if } j \text{ is in output la} \\ \phi_j'(v_j^{\ell}(n))\sum_k \delta_k^{\ell+1}(n) w_{kj}^{\ell+1}(n) & \text{if } j \text{ is in hidden la} \end{cases}$$

Complete Algorithm

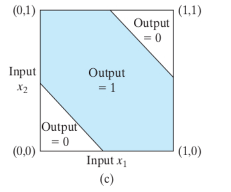
Weight Updation Rule Apply gradient descent rule

$$w_{ji}^{\ell}(n+1) = w_{ji}^{\ell}(n) + \eta\delta_j^{\ell}(n) y_i^{(\ell-1)}(n)$$

Iterate Till stopping criteria is met

Multi-layer Perceptrons

XOR problem



Multi-layer Perceptrons

XOR Architecture

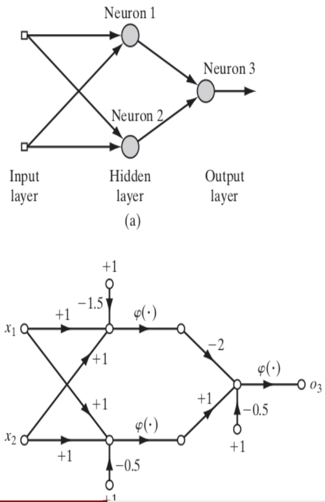


FIGURE 4.8 (a) Architectural graph of network for solving the XOR problem. (b) Signal-flow graph of the network.

Multi-layer Perceptrons

Let training data be $\{(0, 0, C_2), (0, 1, C_1), (1, 0, C_1), (1, 1, C_2)\}$
Presenting first input (0, 0) to the network is as follows

1^{st} layer: 1N $\quad \phi(0 \times 1 + 0 \times 1 - 1.5) = \phi(-1.5) = 0$
1^{st} layer: 2N $\quad \phi(0 \times 1 + 0 \times 1 - 0.5) = \phi(-0.5) = 0$
2^{st} layer: 1N $\quad \phi(0 \times -2 + 0 \times 1 - 0.5) = \phi(-0.5) = 0 \; (0, 0) \in C_2$

Multi-layer Perceptrons

Let training data be $\{(0,0,),(0,1),(1,0),(1,1)\}$
Presenting first input (0, 1) to the network is as follows

1^{st} layer: 1N $\phi(0 \times 1 + 1 \times 1 - 1.5) = \phi(-0.5) = 0$
1^{st} layer: 2N $\phi(0 \times 1 + 1 \times 1 - 0.5) = \phi(0.5) = 1$
2^{st} layer: 1N $\phi(0 \times -2 + 1 \times 1 - 0.5) = \phi(0.5) = 0 \ (0,1) \in C_1$

Multi-layer Perceptrons

Neuron's Learning

FIGURE 4.9 (a) Decision boundary constructed by hidden neuron 1 of the network in Fig. 4.8. (b) Decision boundary constructed by hidden neuron 2 of the network. (c) Decision boundaries constructed by the complete network.