# Deep Learning

Vijaya Saradhi

**IIT Guwahati**

Tue, 22nd Sept 2020

# Multi-layer Perceptrons

## Error function

- Of the form $\{\mathbf{x}(n), \mathbf{d}(n)\}_{n=1}^{N}$
- Let $y_j(n)$: function signal at the output neuron $j$ in the output layer
- Error at neuron $j$ is: $e_j(n) = d_j(n) - y_j(n)$
- Instantaneous error energy:

$$\mathcal{E}_j(n) = \frac{1}{2} e_j^2(n)$$

- The above equation is error made by one output neuron

# Multi-layer Perceptrons

## Error function

- Total instantaneous error energy is:

$$\begin{aligned}\mathcal{E}(n) &= \sum_{j\in C}\mathcal{E}_j(n)\\ &= \frac{1}{2}\sum_{j\in C} e_j^2(n)\end{aligned}$$

- The above equation is for one training example.
- Error incurred over all the training examples is given by:
- Average error

$$\begin{aligned}\mathcal{E}_{av}(N) &= \frac{1}{N}\sum_{n=1}^{N}\mathcal{E}(n)\\ &= \frac{1}{2N}\sum_{n=1}^{N}\sum_{j\in C} e_j^2(n)\end{aligned}$$

# Multi-layer Perceptrons

## Error function depends on weights

- The error function depends on weights indirectly
- $\mathcal{E}_{av} \rightarrow \mathcal{E}(n)$
- $\mathcal{E}(n) \rightarrow e_j(n)$
- $e_j(n) \rightarrow y_j(n)$
- $y_j(n) \rightarrow v_j(n)$
- $v_j(n) \rightarrow w_{ji}(n)$
- These dependencies are used in computing the gradient of $\mathcal{E}$ or $\mathcal{E}_{av}$ with respect to $w_{ji}$

# Multi-layer Perceptrons

## Online vs Offline Training

- Difference in objective function
- Difference in the weight updates using gradient descent method
- Obtained solution need not be identical

# Back-Propagation Algorithm On-line learning

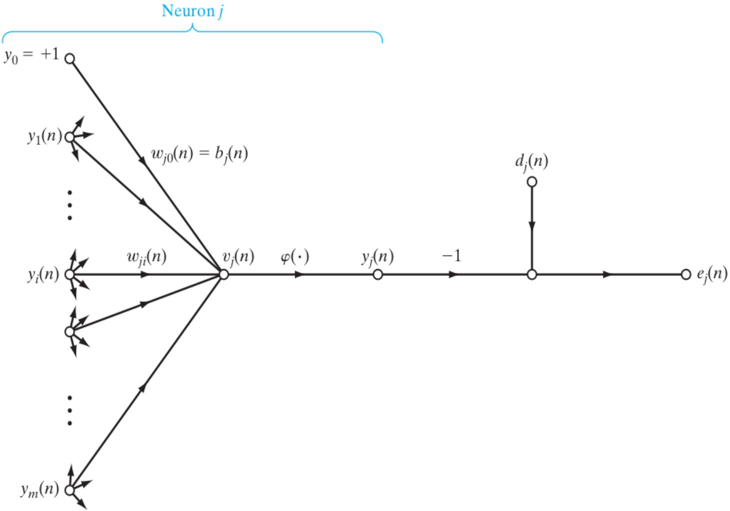


FIGURE 4.3 Signal-flow graph highlighting the details of output neuron $j$.

# Multi-layer Perceptrons

## Error function depends on weights

- The error function depends on weights indirectly
- $\mathcal{E}(n) \rightarrow e_j(n)$
- $e_j(n) \rightarrow y_j(n)$
- $y_j(n) \rightarrow v_j(n)$
- $v_j(n) \rightarrow w_{ji}(n)$
- These dependencies are used in computing the gradient of $\mathcal{E}$ or $\mathcal{E}_{av}$ with respect to $w_{ji}$
- $v_j(n) = \sum_{i=0}^{m} w_{ji}(n) y_i(n)$

# Multi-layer Perceptrons

## Error function depends on weights

- The error function depends on weights indirectly
- $\mathcal{E}(n) \rightarrow e_j(n)$
- $e_j(n) \rightarrow y_j(n)$
- $y_j(n) \rightarrow v_j(n)$
- $v_j(n) \rightarrow w_{ji}(n)$
- These dependencies are used in computing the gradient of $\mathcal{E}$ or $\mathcal{E}_{av}$ with respect to $w_{ji}$
- $y_j(n) = \phi(v_j(n))$

# Multi-layer Perceptrons

## Error function depends on weights

- The error function depends on weights indirectly
- $\mathcal{E}(n) \rightarrow e_j(n)$
- $e_j(n) \rightarrow y_j(n)$
- $y_j(n) \rightarrow v_j(n)$
- $v_j(n) \rightarrow w_{ji}(n)$
- These dependencies are used in computing the gradient of $\mathcal{E}$ or $\mathcal{E}_{av}$ with respect to $w_{ji}$
- $e_j(n) = d_j(n) - y_j(n)$

# Multi-layer Perceptrons

## Error function depends on weights

- The error function depends on weights indirectly
- $\mathcal{E}(n) \rightarrow e_j(n)$
- $e_j(n) \rightarrow y_j(n)$
- $y_j(n) \rightarrow v_j(n)$
- $v_j(n) \rightarrow w_{ji}(n)$
- These dependencies are used in computing the gradient of $\mathcal{E}$ or $\mathcal{E}_{av}$ with respect to $w_{ji}$
- $\mathcal{E}(n) = \frac{1}{2}\sum_{j \in C} e_j^2(n)$

# Multi-layer Perceptrons

## Error function derivative

- The error function depends on weights indirectly
- $\mathcal{E}(n) \rightarrow e_j(n)$
- $e_j(n) \rightarrow y_j(n)$
- $y_j(n) \rightarrow v_j(n)$
- $v_j(n) \rightarrow w_{ji}(n)$
- These dependencies are used in computing the gradient of $\mathcal{E}$ or $\mathcal{E}_{av}$ with respect to $w_{ji}$
- Chain rule

$$\frac{\partial \mathcal{E}(n)}{\partial w_{ji}(n)} = \frac{\partial \mathcal{E}(n)}{\partial e_j(n)} \frac{\partial e_j(n)}{\partial y_j(n)} \frac{\partial y_j(n)}{\partial v_j(n)} \frac{\partial v_j(n)}{\partial w_{ji}(n)}$$

# Multi-layer Perceptrons

## Error function derivative

- $\mathcal{E}(n) = \frac{1}{2}\sum_{j\in C} e_j^2(n)$
- $\frac{\partial \mathcal{E}(n)}{\partial e_j(n)} = e_j(n)$
- Chain rule

$$\frac{\partial \mathcal{E}(n)}{\partial w_{ji}(n)} = e_j(n)\frac{\partial e_j(n)}{\partial y_j(n)}\frac{\partial y_j(n)}{\partial v_j(n)}\frac{\partial v_j(n)}{\partial w_{ji}(n)}$$

# Multi-layer Perceptrons

## Error function derivative

- $e_j(n) = d_j(n) - y_j(n)$
- $\frac{\partial e_j(n)}{\partial y_j(n)} = -1$
- Chain rule

$$\frac{\partial \mathcal{E}(n)}{\partial w_{ji}(n)} = e_j(n) \times -1 \times \frac{\partial y_j(n)}{\partial v_j(n)} \frac{\partial v_j(n)}{\partial w_{ji}(n)}$$

# Multi-layer Perceptrons

## Error function derivative

- $y_j(n) = \phi(v_j(n))$
- $\frac{\partial y_j(n)}{\partial v_j(n)} = \phi_j'(v_j(n))$
- Choose an activation function that is differentiable
- Chain rule

$$\frac{\partial \mathcal{E}(n)}{\partial w_{ji}(n)} = e_j(n) \times -1 \times \phi_j'(v_j(n)) \times \frac{\partial v_j(n)}{\partial w_{ji}(n)}$$

# Multi-layer Perceptrons

## Error function derivative

- $v_j(n) = \sum_{i=0}^{m} w_{ji}(n) y_i(n)$
- $\frac{\partial v_j(n)}{\partial w_{ji}(n)} = y_i(n)$
- Chain rule

$$\frac{\partial \mathcal{E}(n)}{\partial w_{ji}(n)} = e_j(n) \times -1 \times \phi_j'(v_j(n)) \times y_i(n)$$

# Multi-layer Perceptrons

## Error function derivative

- We know the following values in the resulting expression:
- Chain rule

$$\frac{\partial \mathcal{E}(n)}{\partial w_{ji}(n)} = e_j(n) \times -1 \times \phi'(v_j(n)) \times y_i(n)$$

- Two cases depending on which layer neuron $j$ is located.

Output Layer If neuron $j$ is output layer then we know the desired response. We also know the output computed by $j^{th}$ neuron. Therefore we have $e_j(n)$
Output neurons will not affect network any further as network will anyway output the computation.

# Multi-layer Perceptrons

## Error function derivative

- We know the following values in the resulting expression:
- Chain rule

$$\frac{\partial \mathcal{E}(n)}{\partial w_{ji}(n)} = e_j(n) \times -1 \times \phi'(v_j(n)) \times y_i(n)$$

- At neuron $j$ we do not know $e_j(n)$

  Hidden Layer If neuron $j$ is a neuron in the hidden layer, we do not know the desired resonse at that neuron

  Hidden layer neurons they share responsibility in the output given by the network

- If we compute $e_j(n)$ at output layer and hidden layers we know all quantities in the gradient term and there update the weights

# Multi-layer Perceptrons

## Local Gradient - Output Neuron

- $w_{ji}(n+1) = w_{ji} - \eta \frac{\partial \mathcal{E}(n)}{\partial w_{ji}(n)}$
- $\Delta w_{ji}(n) = -\eta \frac{\partial \mathcal{E}(n)}{\partial w_{ji}(n)}$
- $\Delta w_{ji}(n) = \eta \delta_j(n) y_i(n)$
- Where $\delta_j(n)$ is the local gradient at $v_j$ (before the activation function)
- Local gradient is computed using chain rule as:

$$\begin{aligned}
\delta_j(n) &= \frac{\partial \mathcal{E}(n)}{\partial v_j(n)} \\
&= \frac{\partial \mathcal{E}(n)}{\partial e_j(n)} \frac{\partial e_j(n)}{\partial y_j(n)} \frac{\partial y_j(n)}{\partial v_j(n)} \\
&= e_j(n) \times -1 \times \phi_j'(v_j(n))
\end{aligned}$$

# Multi-layer Perceptrons

**$j$ is output layer**

Compute the error as: $e_j(n) = d_j(n) - y_j(n)$

# Multi-layer Perceptrons

## Local Gradient - Hidden Neuron

- There is no specified desired respons for that neuron
- Error term has to be obtained from the error signals of all the neurons to which that hidden neuron is directly connected
- $\delta_j(n)$ is the local gradient at $v_j$ (before the activation function)
- Output emitted by neuron $j$: Local gradient is computed using chain rule as:

$$(output)\delta_j(n) = \frac{\partial \mathcal{E}(n)}{\partial e_j(n)} \frac{\partial e_j(n)}{\partial y_j(n)} \frac{\partial y_j(n)}{\partial v_j(n)}$$

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

# Multi-layer Perceptrons

## Local Gradient

- To compute $\frac{\partial \mathcal{E}(n)}{\partial y_j(n)}$ where

$$\mathcal{E}(n) = \frac{1}{2}\sum_{k \in C} e_k^2(n) \text{neuron k is in output layer}$$

- When $k$ is a hidden neuron, the error is computed by summing all the hidden neuron errors. That is

$$\mathcal{E}(n) = \frac{1}{2}\sum_{k} e_k^2(n)$$

# 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: $e_k(n) = d_k(n) - \phi_k(v_k(n))$
  $\frac{\partial e_k(n)}{\partial v_k(n)} = \phi_k'(v_k(n))$

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

# 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

- Gradient descent update rule in online learning of MLP is:

$$\begin{pmatrix} \textit{Weight} \\ \textit{Correction} \\ \Delta w_{ji}(n) \end{pmatrix} = \begin{pmatrix} \text{Learning} \\ \text{rate parameter} \\ \eta \end{pmatrix} \times \begin{pmatrix} \text{local} \\ \text{gradient} \\ \delta_j(n) \end{pmatrix} \times \begin{pmatrix} \text{input signal} \\ \text{of neuron j} \\ y_j(n) \end{pmatrix}$$

# Gradient Descent Update Rule in Online MLP Learning

$$\begin{pmatrix} \textit{Weight} \\ \textit{correction} \\ \Delta w_{ji}(n) \end{pmatrix} = \begin{pmatrix} \textit{learning-} \\ \textit{rate parameter} \\ \eta \end{pmatrix} \times \begin{pmatrix} \textit{local} \\ \textit{gradient} \\ \delta_j(n) \end{pmatrix} \times \begin{pmatrix} \textit{input signal} \\ \textit{of neuron } j, \\ y_i(n) \end{pmatrix} \quad (4.27)$$