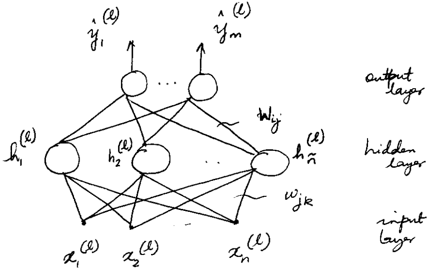
The Generalized Delta Rule aka Back-Propagation

Ravi Kothari, Ph.D.
ravi.kothari@ashoka.edu.in

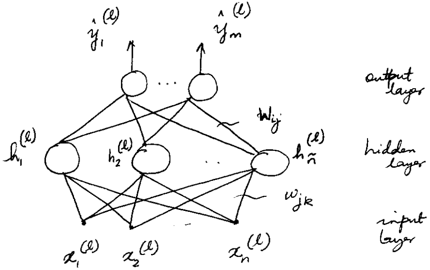
"Everything that is possible demands to exist" – Gottfried Wilhelm Leibniz

Multi-Layered Perceptrons

Multi-Layered Perceptrons

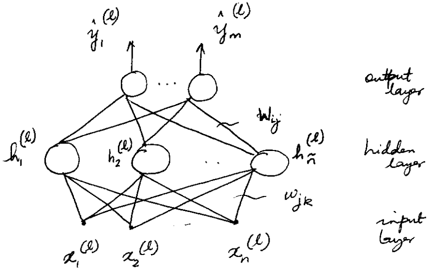


Multi-Layered Perceptrons



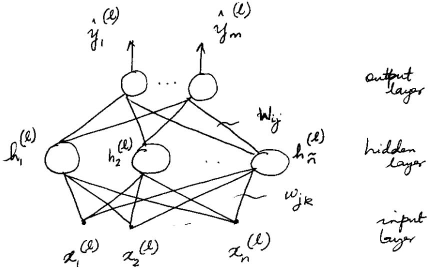
- A bias is present in each neuron (not shown explicitly)

Multi-Layered Perceptrons



- A bias is present in each neuron (not shown explicitly)
- Output is indexed using i, hidden neurons are indexed using j and the inputs are indexed using k

Multi-Layered Perceptrons



- A bias is present in each neuron (not shown explicitly)
- Output is indexed using i, hidden neurons are indexed using j and the inputs are indexed using k
- W_{ij} is the weight from output i and hidden neuron j. w_{jk} is the weight from hidden neuron j and input k

The Generalized Delta Rule aka Back-Propagation

The Generalized Delta Rule aka Back-Propagation

- As before, let the training data be denoted by,

$$\left\{ \left(x^{(l)}, y^{(l)} \right) \right\}_{l=1}^{N} \qquad ; x^{(l)} \in \mathcal{R}^n, \quad y^{(l)} \in \mathcal{R}^m$$

The Network Output

The Network Output

- The output of a hidden neuron is,

$$h_j^{(l)} = f\left(S_j^{(l)}\right) = f\left(\sum_{k=0}^{n} w_{jk} x_k^{(l)}\right)$$

The Network Output

- The output of a hidden neuron is,

$$h_j^{(l)} = f\left(S_j^{(l)}\right) = f\left(\sum_{k=0}^{n} w_{jk} x_k^{(l)}\right)$$

- Similarly, the output can be defined as,

$$\hat{y}_i^{(l)} = f\left(S_i^{(l)}\right) = f\left(\sum_{j=0}^{\tilde{n}} W_{ij} h_j^{(l)}\right)$$

The Cost Function

The Cost Function

- $$\begin{aligned} J^{(l)} &= \sum_{i=1}^{m} \left(y_i^{(l)} - \hat{y}_i^{(l)} \right)^2 \\ &= \sum_{i=1}^{m} {e_i^{(l)}}^2 \end{aligned}$$

The Cost Function

- $$J^{(l)} = \sum_{i=1}^{m} \left(y_i^{(l)} - \hat{y}_i^{(l)} \right)^2 = \sum_{i=1}^{m} e_i^{(l)2}$$

- $$J = \sum_{l=1}^{N} J^{(l)} = \sum_{l=1}^{N} \sum_{i=1}^{m} \left(y_i^{(l)} - \hat{y}_i^{(l)} \right)^2$$

The Cost Function

The Cost Function

$$
\begin{aligned}
J &= \sum_{l=1}^{N}\sum_{i=1}^{m}\left(y_i^{(l)} - \hat{y}_i^{(l)}\right)^2 \\
&= \sum_{l=1}^{N}\sum_{i=1}^{m}\left[y_i^{(l)} - f\left(\sum_{j=0}^{\tilde{n}} W_{ij}\, f\left(\sum_{k=0}^{n} w_{jk} x_k^{(l)}\right)\right)\right]^2
\end{aligned}
$$

The Cost Function

$$
\begin{aligned}
J &= \sum_{l=1}^{N} \sum_{i=1}^{m} \left(y_i^{(l)} - \hat{y}_i^{(l)} \right)^2 \\
&= \sum_{l=1}^{N} \sum_{i=1}^{m} \left[y_i^{(l)} - f\left(\sum_{j=0}^{\tilde{n}} W_{ij}\, f\left(\sum_{k=0}^{n} w_{jk} x_k^{(l)} \right) \right) \right]^2
\end{aligned}
$$

So, we can start with random values of w's and adopt the weights by moving opposite to the gradient computed from the equation above, i.e.

$$\Delta w = -\eta \nabla J$$

The Generalized Delta Rule Derived

The Generalized Delta Rule Derived

In batch learning, we aggregate the gradients computed for each pattern, i.e.,

$$\nabla J = \nabla J^{(1)} + \nabla J^{(2)} + \ldots + \nabla J^{(N)}$$

The Generalized Delta Rule Derived

In batch learning, we aggregate the gradients computed for each pattern, i.e.,

$$\nabla J = \nabla J^{(1)} + \nabla J^{(2)} + \ldots + \nabla J^{(N)}$$

In online learning, we make the approximation that as long as η is small, we can,

- Feed pattern I
- Compute $\nabla J^{(I)}$
- Update W's and w's and repeat

Output–Hidden Weights

Output–Hidden Weights

$$
\begin{aligned}
J &= \sum_{l=1}^{N} \sum_{i=1}^{m} {e_i^{(l)}}^2 = \sum_{l=1}^{N} \sum_{i=1}^{m} \left(y_i^{(l)} - \hat{y}_i^{(l)} \right)^2 \\
&= \sum_{l=1}^{N} \sum_{i=1}^{m} \left[y_i^{(l)} - f \left(\sum_{j=0}^{\tilde{n}} W_{ij}\, f \left(\sum_{k=0}^{n} w_{jk} x_k^{(l)} \right) \right) \right]^2
\end{aligned}
$$

Output–Hidden Weights

$$
\begin{aligned}
J &= \sum_{l=1}^{N}\sum_{i=1}^{m} {e_i^{(l)}}^2 = \sum_{l=1}^{N}\sum_{i=1}^{m}\left(y_i^{(l)} - \hat{y}_i^{(l)}\right)^2 \\
&= \sum_{l=1}^{N}\sum_{i=1}^{m}\left[y_i^{(l)} - f\left(\sum_{j=0}^{\tilde{n}} W_{ij}\, f\left(\sum_{k=0}^{n} w_{jk} x_k^{(l)}\right)\right)\right]^2
\end{aligned}
$$

$$
\begin{aligned}
\frac{\partial J^{(l)}}{\partial W_{ij}} &= \frac{\partial J^{(l)}}{\partial e_i^{(l)}} \frac{\partial e_i^{(l)}}{\partial \hat{y}_i^{(l)}} \frac{\partial \hat{y}_i^{(l)}}{\partial S_i^{(l)}} \frac{\partial S_i^{(l)}}{\partial W_{ij}} \\
&= e_i^{(l)} \cdot (-1) \cdot f'(S_i^{(l)}) \cdot h_j^{(l)} \\
&= -\left(y_i^{(l)} - \hat{y}_i^{(l)}\right) \cdot f'(S_i^{(l)}) \cdot h_j^{(l)}
\end{aligned}
$$

Output–Hidden (Online) Weight Updates

$$\begin{aligned}
W_{ij}^{(l)}(\mathrm{new}) &= W_{ij}^{(l)}(\mathrm{old}) + \Delta W_{ij}^{(l)} \\
&= W_{ij}^{(l)}(\mathrm{old}) - \eta \cdot \frac{\partial J^{(l)}}{\partial W_{ij}} \\
&= W_{ij}^{(l)}(\mathrm{old}) + \eta \cdot e_i^{(l)} \cdot f'(S_i^{(l)}) \cdot h_j^{(l)} \\
&= W_{ij}^{(l)}(\mathrm{old}) + \eta \cdot \left(y_i^{(l)} - \hat{y}_i^{(l)}\right) \cdot f'(S_i^{(l)}) \cdot h_j^{(l)} \\
&= W_{ij}^{(l)}(\mathrm{old}) + \eta \cdot \delta_i^{(l)} \cdot h_j^{(l)}
\end{aligned}$$

where, $\delta_i^{(l)} = \left(y_i^{(l)} - \hat{y}_i^{(l)}\right) \cdot f'(S_i^{(l)})$

Hidden–Input Weights

Hidden–Input Weights

$$
\begin{aligned}
J &= \sum_{l=1}^{N}\sum_{i=1}^{m} {e_i^{(l)}}^2 = \sum_{l=1}^{N}\sum_{i=1}^{m}\left(y_i^{(l)} - \hat{y}_i^{(l)}\right)^2 \\
&= \sum_{l=1}^{N}\sum_{i=1}^{m}\left[y_i^{(l)} - f\left(\sum_{j=0}^{\tilde{n}} W_{ij}\, f\left(\sum_{k=0}^{n} w_{jk} x_k^{(l)}\right)\right)\right]^2
\end{aligned}
$$

Hidden–Input Weights

$$
\begin{aligned}
J &= \sum_{l=1}^{N}\sum_{i=1}^{m} {e_i^{(l)}}^2 = \sum_{l=1}^{N}\sum_{i=1}^{m}\left(y_i^{(l)} - \hat{y}_i^{(l)}\right)^2 \\
&= \sum_{l=1}^{N}\sum_{i=1}^{m}\left[y_i^{(l)} - f\left(\sum_{j=0}^{\tilde{n}} W_{ij}\, f\left(\sum_{k=0}^{n} w_{jk} x_k^{(l)}\right)\right)\right]^2
\end{aligned}
$$

$$
\begin{aligned}
\frac{\partial J^{(l)}}{\partial w_{jk}} &= \sum_{i=1}^{m} \frac{\partial J^{(l)}}{\partial e_i^{(l)}} \frac{\partial e_i^{(l)}}{\partial \hat{y}_i^{(l)}} \frac{\partial \hat{y}_i^{(l)}}{\partial S_i^{(l)}} \frac{\partial S_i^{(l)}}{\partial h_j^{(l)}} \frac{\partial h_j^{(l)}}{\partial S_j^{(l)}} \frac{\partial S_j^{(l)}}{\partial w_{jk}} \\
&= \sum_{i=1}^{m} e_i^{(l)} \cdot (-1) \cdot f'(S_i^{(l)}) \cdot W_{ij} \cdot f'(S_j^{(l)}) \cdot x_k^{(l)} \\
&= -\left[\sum_{i=1}^{m} e_i^{(l)} \cdot f'(S_i^{(l)}) \cdot W_{ij}\right] \cdot f'(S_j^{(l)}) \cdot x_k^{(l)}
\end{aligned}
$$

Hidden–Input (Online) Weight Updates

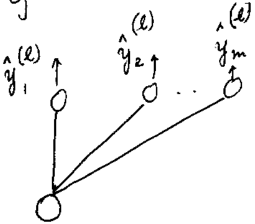
$$
\begin{aligned}
w_{jk}^{(l)}(\text{new}) &= w_{jk}^{(l)}(\text{old}) + \Delta w_{jk}^{(l)} \\
&= w_{jk}^{(l)}(\text{old}) - \eta \cdot \frac{\partial J^{(l)}}{\partial w_{jk}} \\
&= w_{jk}^{(l)}(\text{old}) + \eta \cdot \left[\sum_{i=1}^{m} e_i^{(l)} \cdot f'(S_i^{(l)}) \cdot W_{ij} \right] \cdot f'(S_j^{(l)}) \cdot x_k^{(l)} \\
&= w_{jk}^{(l)}(\text{old}) + \eta \cdot \left[\sum_{i=1}^{m} \delta_i^{(l)} \cdot W_{ij} \right] \cdot f'(S_j^{(l)}) \cdot x_k^{(l)} \\
&= w_{jk}^{(l)}(\text{old}) + \eta \cdot \delta_j^{(l)} \cdot x_k^{(l)}
\end{aligned}
$$

where, $\delta_j^{(l)} = \left[\sum_{i=1}^{m} \delta_i^{(l)} \cdot W_{ij} \right] \cdot f'(S_j^{(l)})$

Back-Propagation of δ's

Back-Propagation of δ's

So, $\delta_j^{(l)}$ is computed as



$\delta_i^{(l)}$ is back-propagated through W_{ij} and summed to give $\delta_j^{(l)}$

The Overall Algorithm

The Overall Algorithm

- Initialize all the weights to small random values

The Overall Algorithm

- Initialize all the weights to small random values
- while J is large

The Overall Algorithm

- Initialize all the weights to small random values
- while J is large
  - for $l = 1, 2, \ldots, N$
    - Explicit loops for all i, j, k not shown but present
    - Do a forward pass i.e. compute,

$$h_j^{(l)} = f\left(S_j^{(l)}\right) = f\left(\sum_{k=0}^{n} w_{jk} x_k^{(l)}\right) \text{ and } \hat{y}_i^{(l)} = f\left(S_i^{(l)}\right) = f\left(\sum_{j=0}^{\tilde{n}} W_{ij} h_j^{(l)}\right)$$

    - Do a backward pass i.e. compute,

$$\delta_i^{(l)} = \left(y_i^{(l)} - \hat{y}_i^{(l)}\right) f'\left(S_i^{(l)}\right) \text{ and } \delta_j^{(l)} = \left(\sum_{i=1}^{m} \delta_i^{(l)} W_{ij}\right) f'(S_j^{(l)})$$

and update the weights,

$$\begin{aligned} W_{ij}^{(l)}(\text{new}) &= W_{ij}^{(l)}(\text{old}) + \eta \cdot \delta_i^{(l)} \cdot h_j^{(l)} \\ w_{jk}^{(l)}(\text{new}) &= w_{jk}^{(l)}(\text{old}) + \eta \cdot \delta_j^{(l)} \cdot x_k^{(l)} \end{aligned}$$

The Overall Algorithm

- Initialize all the weights to small random values
- while J is large
  - for $l = 1, 2, \ldots, N$
    - Explicit loops for all i, j, k not shown but present
    - Do a forward pass i.e. compute,

$$h_j^{(l)} = f\left(S_j^{(l)}\right) = f\left(\sum_{k=0}^{n} w_{jk} x_k^{(l)}\right) \text{ and } \hat{y}_i^{(l)} = f\left(S_i^{(l)}\right) = f\left(\sum_{j=0}^{\tilde{n}} W_{ij} h_j^{(l)}\right)$$

    - Do a backward pass i.e. compute,

$$\delta_i^{(l)} = \left(y_i^{(l)} - \hat{y}_i^{(l)}\right) f'\left(S_i^{(l)}\right) \text{ and } \delta_j^{(l)} = \left(\sum_{i=1}^{m} \delta_i^{(l)} W_{ij}\right) f'(S_j^{(l)})$$

and update the weights,

$$\begin{aligned} W_{ij}^{(l)}(\text{new}) &= W_{ij}^{(l)}(\text{old}) + \eta \cdot \delta_i^{(l)} \cdot h_j^{(l)} \\ w_{jk}^{(l)}(\text{new}) &= w_{jk}^{(l)}(\text{old}) + \eta \cdot \delta_j^{(l)} \cdot x_k^{(l)} \end{aligned}$$

  - end for
- end while

Some Notes

Some Notes

- Small random weights ensure that you are in likely in the steeper portion of the sigmoidal activation function and hence see the largest weights updates

Some Notes

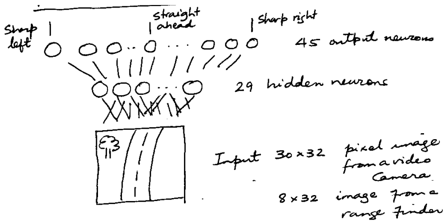
- Small random weights ensure that you are in likely in the steeper portion of the sigmoidal activation function and hence see the largest weights updates
- There are multiple variations tat help speed up learning e.g. use of momentum, conjugate gradient, Levenburg-Marquardt etc.

Some Notes

- Small random weights ensure that you are in likely in the steeper portion of the sigmoidal activation function and hence see the largest weights updates
- There are multiple variations tat help speed up learning e.g. use of momentum, conjugate gradient, Levenburg-Marquardt etc.
- A large number of regularizers are possible e.g. the second derivative of the output relative to the weights to help minimize sudden changes in the output i.e. promote smooth variations

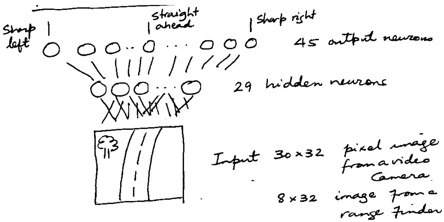
An Application - Autonomous Navigation

An Application - Autonomous Navigation



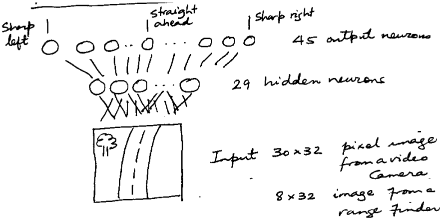
- The output neurons progressively code for sharper turns as one moves away from the center

An Application - Autonomous Navigation



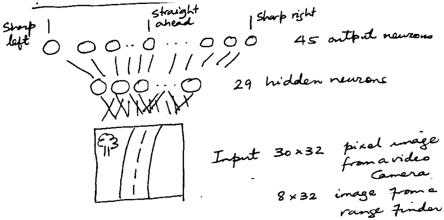
- The output neurons progressively code for sharper turns as one moves away from the center
- Training comprised of 1200 simulated road images. After 40 epocs, the car could autonomously navigate at 5 KM/Hr on a road through a wooded area

An Application - Autonomous Navigation



- The output neurons progressively code for sharper turns as one moves away from the center
- Training comprised of 1200 simulated road images. After 40 epocs, the car could autonomously navigate at 5 KM/Hr on a road through a wooded area
- One could use additional inputs and have additional outputs e.g. amount of acceleration/braking etc.

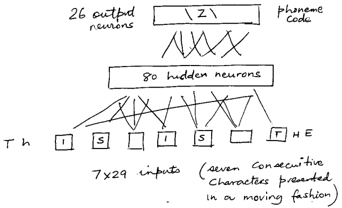
An Application - Autonomous Navigation



- The output neurons progressively code for sharper turns as one moves away from the center
- Training comprised of 1200 simulated road images. After 40 epocs, the car could autonomously navigate at 5 KM/Hr on a road through a wooded area
- One could use additional inputs and have additional outputs e.g. amount of acceleration/braking etc.
- D. A. Pomerleau, "Efficient Training of Artificial Neural Networks for Autonomous Navigation," *Neural Computation*, Vol. 3, No. 1, pp. 88–97, 1991

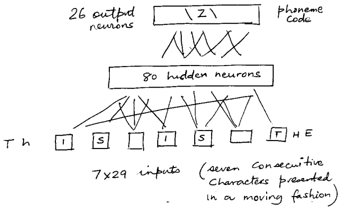
An Application - NETTalk

An Application - NETTalk



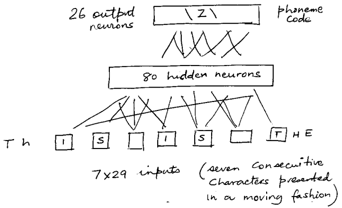
- Desired output was a phoneme code for the letter in the center of the window

An Application - NETTalk



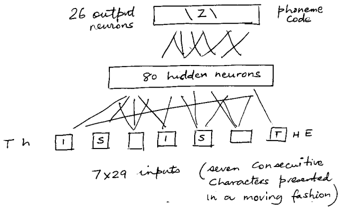
- Desired output was a phoneme code for the letter in the center of the window
- Network trained using 1024 words, Obtained intelligible speech after 10 training epochs (95% accuracy after 50 epochs

An Application - NETTalk



- Desired output was a phoneme code for the letter in the center of the window
- Network trained using 1024 words, Obtained intelligible speech after 10 training epochs (95% accuracy after 50 epochs
- Some hidden neurons were found to represent meaningful properties such as the distinction between vowels and consonants

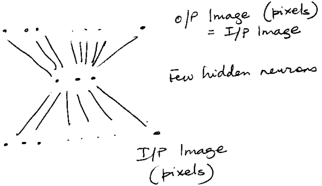
An Application - NETTalk



- Desired output was a phoneme code for the letter in the center of the window
- Network trained using 1024 words, Obtained intelligible speech after 10 training epochs (95% accuracy after 50 epochs
- Some hidden neurons were found to represent meaningful properties such as the distinction between vowels and consonants
- T. J. Sejnowski, and C. R. Rosenburg, "NETTalk: A Parallel Network that Learns to Read Aloud," *The Johns Hopkins University Electrical Engineering and Computer Science Technical Report JHU/EECS-86/01*, 1986

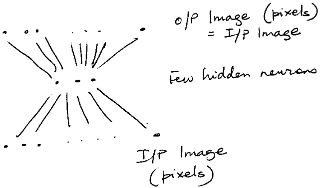
An Application - Image Compression

An Application - Image Compression



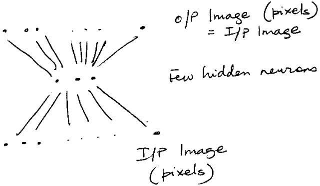
- The input and the desired output are the same i.e. the output is the uncompressed image. The output of the hidden neurons is the compressed image (if the number of neurons in the hidden layer are less than the number of neurons in the input layer)

An Application - Image Compression



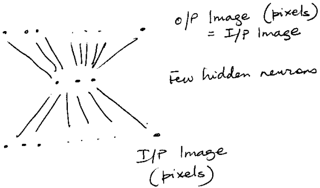
- The input and the desired output are the same i.e. the output is the uncompressed image. The output of the hidden neurons is the compressed image (if the number of neurons in the hidden layer are less than the number of neurons in the input layer)

An Application - Image Compression



- The input and the desired output are the same i.e. the output is the uncompressed image. The output of the hidden neurons is the compressed image (if the number of neurons in the hidden layer are less than the number of neurons in the input layer)

An Application - Image Compression



- The input and the desired output are the same i.e. the output is the uncompressed image. The output of the hidden neurons is the compressed image (if the number of neurons in the hidden layer are less than the number of neurons in the input layer)
- Lossy compression since one may not recover the exact output as the input (training error will not likely go to 0 specially for large amounts of compression)

Some Enhancements - Momentum

Some Enhancements - Momentum

- Factor past weight changes into the present weight change

Some Enhancements - Momentum

- Factor past weight changes into the present weight change
- $$\begin{aligned}\Delta w_{ij}(t+1) &= \eta \delta_i^{(l)} h_j^{(l)} + \alpha \Delta w_{ij}(t) \\ \Delta w_{jk}(t+1) &= \eta \delta_i^{(j)} x_k^{(l)} + \alpha \Delta w_{jk}(t)\end{aligned}$$

  where, $0 \leq \alpha \leq 1$ is the momentum coefficient

Some Enhancements - Momentum

- Factor past weight changes into the present weight change
-

$$\begin{aligned}
\Delta w_{ij}(t+1) &= \eta \delta_i^{(l)} h_j^{(l)} + \alpha \Delta w_{ij}(t) \\
\Delta w_{jk}(t+1) &= \eta \delta_i^{(j)} x_k^{(l)} + \alpha \Delta w_{jk}(t)
\end{aligned}$$

where, $0 \leq \alpha \leq 1$ is the momentum coefficient

- Effect when moving through a plateau,

$$\begin{aligned}
\Delta w(0) &= \frac{\partial J}{\partial w} = -\eta a \\
\Delta w(1) &= \frac{\partial J}{\partial w} + \alpha\left(\Delta w(0)\right) = -\eta a - \eta \alpha a = -\eta a(1+\alpha) \\
\Delta w(2) &= \frac{\partial J}{\partial w} + \alpha\left(-\eta a - \eta \alpha a\right) = -\eta a - \eta \alpha a = -\eta a(1+\alpha+\alpha^2) \\
\Delta w(t) &= -\eta a(1+\alpha+\alpha^2+\ldots+\alpha^t) = \frac{-\eta}{1-\alpha} a
\end{aligned}$$

Some Enhancements - Momentum

- Factor past weight changes into the present weight change
- $$\begin{aligned}\Delta w_{ij}(t+1) &= \eta\delta_i^{(l)} h_j^{(l)} + \alpha\Delta w_{ij}(t) \\ \Delta w_{jk}(t+1) &= \eta\delta_i^{(j)} x_k^{(l)} + \alpha\Delta w_{jk}(t)\end{aligned}$$

  where, $0 \leq \alpha \leq 1$ is the momentum coefficient
- Effect when moving through a plateau,

$$\begin{aligned}\Delta w(0) &= \frac{\partial J}{\partial w} = -\eta a \\ \Delta w(1) &= \frac{\partial J}{\partial w} + \alpha\left(\Delta w(0)\right) = -\eta a - \eta\alpha a = -\eta a(1+\alpha) \\ \Delta w(2) &= \frac{\partial J}{\partial w} + \alpha\left(-\eta a - \eta\alpha a\right) = -\eta a - \eta\alpha a = -\eta a(1+\alpha+\alpha^2) \\ \Delta w(t) &= -\eta a(1+\alpha+\alpha^2+\ldots+\alpha^t) = \frac{-\eta}{1-\alpha} a\end{aligned}$$

- So, effective learning rate is increased. What happens through a rough

Some Enhancements - Higher Order Information

Some Enhancements - Higher Order Information

- $$J(w) = J(w_0) + (w - w_0)\nabla J(w_0) + (w - w_0)^T H(w - w_0)$$

  where, $H_{ij} = \frac{\partial^2 J}{\partial w_i \partial w_j}$

Some Enhancements - Higher Order Information

- $$J(w) = J(w_0) + (w - w_0)\nabla J(w_0) + (w - w_0)^T H(w - w_0)$$

  where, $H_{ij} = \frac{\partial^2 J}{\partial w_i \partial w_j}$
- The Hessian is expensive to compute. So, there are quasi-Newton methods like Conjugate Gradient, Levenberg-Marquardt etc. that can be used

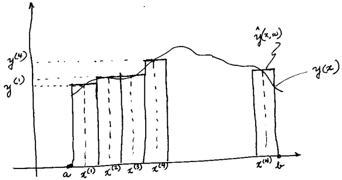
Multi-Layered FFNN's are Universal Approximators

Multi-Layered FFNN's are Universal Approximators

- Take a simple case in $1-D$. We are given $\left\{\left(x^{(l)}, y^{(l)}\right)\right\}_{l=1}^{N}$. Let us assume that $x^{(l)}$ is uniformaly distributed in (a, b)

Multi-Layered FFNN's are Universal Approximators

- Take a simple case in $1 - D$. We are given $\left\{\left(x^{(l)}, y^{(l)}\right)\right\}_{l=1}^{N}$. Let us assume that $x^{(l)}$ is uniformaly distributed in (a, b)



Multi-Layered FFNN's are Universal Approximators

Multi-Layered FFNN's are Universal Approximators

- $x^{(l+1)} - x^{(l)} = \Delta x = \frac{b-a}{N}$, i.e. (a, b) is equally divided into N equal intervals $\left(x^{(l)} - \frac{\Delta x}{2}, x^{(l)} + \frac{\Delta x}{2}\right)$. $x^{(1)} - \frac{\Delta x}{2} = a$ and $x^{(N)} + \frac{\Delta x}{2} = b$

Multi-Layered FFNN's are Universal Approximators

- $x^{(l+1)} - x^{(l)} = \Delta x = \frac{b-a}{N}$, i.e. (a, b) is equally divided into N equal intervals $\left(x^{(l)} - \frac{\Delta x}{2}, x^{(l)} + \frac{\Delta x}{2}\right)$. $x^{(1)} - \frac{\Delta x}{2} = a$ and $x^{(N)} + \frac{\Delta x}{2} = b$
- $$\hat{y}(x, w) = \sum_{l=1}^{N} \left[u\left(x - x^{(l)} + \frac{\Delta x}{2}\right) - u\left(x - x^{(l)} - \frac{\Delta x}{2}\right) \right]$$

Multi-Layered FFNN's are Universal Approximators

- $x^{(l+1)} - x^{(l)} = \Delta x = \frac{b-a}{N}$, i.e. (a, b) is equally divided into N equal intervals $\left(x^{(l)} - \frac{\Delta x}{2}, x^{(l)} + \frac{\Delta x}{2}\right)$. $x^{(1)} - \frac{\Delta x}{2} = a$ and $x^{(N)} + \frac{\Delta x}{2} = b$
- $$\hat{y}(x, w) = \sum_{l=1}^{N} \left[u\left(x - x^{(l)} + \frac{\Delta x}{2}\right) - u\left(x - x^{(l)} - \frac{\Delta x}{2}\right) \right]$$