Introduction to Artificial Intelligence

Multilayer Perceptron

Andres Mendez-Vazquez

March 11, 2019

Outline

Cinvestav

Outline

Cinvestav

Do you remember?

The Perceptron has the following problem

Given that the perceptron is a linear classifier

It is clear that

It will never be able to classify stuff that is not linearly separable

Do you remember?

The Perceptron has the following problem

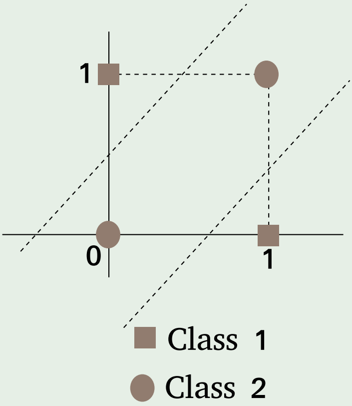
Given that the perceptron is a linear classifier

It is clear that

It will never be able to classify stuff that is not linearly separable

Example: XOR Problem

The Problem



The Perceptron cannot solve it

Because

The perceptron is a linear classifier!!!

Thus

Something needs to be done!!!

Maybe

Add an extra layer!!!

The Perceptron cannot solve it

Because

The perceptron is a linear classifier!!!

Thus

Something needs to be done!!!

Maybe

Add an extra layer!!!

The Perceptron cannot solve it

Because

The perceptron is a linear classifier!!!

Thus

Something needs to be done!!!

Maybe

Add an extra layer!!!

A little bit of history

It was first cited by Vapnik

Vapnik cites (Bryson, A.E.; W.F. Denham; S.E. Dreyfus. Optimal programming problems with inequality constraints. I: Necessary conditions for extremal solutions. AIAA J. 1, 11 (1963) 2544-2550) as the first publication of the backpropagation algorithm in his book "Support Vector Machines."

It was first used by

Arthur E. Bryson and Yu-Chi Ho described it as a multi-stage dynamic system optimization method in 1969.

However

It was not until 1974 and later, when applied in the context of neural networks and through the work of Paul Werbos, David E. Rumelhart, Geoffrey E. Hinton and Ronald J. Williams that it gained recognition.

A little bit of history

It was first cited by Vapnik

Vapnik cites (Bryson, A.E.; W.F. Denham; S.E. Dreyfus. Optimal programming problems with inequality constraints. I: Necessary conditions for extremal solutions. AIAA J. 1, 11 (1963) 2544-2550) as the first publication of the backpropagation algorithm in his book "Support Vector Machines."

It was first used by

Arthur E. Bryson and Yu-Chi Ho described it as a multi-stage dynamic system optimization method in 1969.

However

It was not until 1974 and later, when applied in the context of neural networks and through the work of Paul Werbos, David E. Rumelhart, Geoffrey E. Hinton and Ronald J. Williams that it gained recognition.

A little bit of history

It was first cited by Vapnik

Vapnik cites (Bryson, A.E.; W.F. Denham; S.E. Dreyfus. Optimal programming problems with inequality constraints. I: Necessary conditions for extremal solutions. AIAA J. 1, 11 (1963) 2544-2550) as the first publication of the backpropagation algorithm in his book "Support Vector Machines."

It was first used by

Arthur E. Bryson and Yu-Chi Ho described it as a multi-stage dynamic system optimization method in 1969.

However

It was not until 1974 and later, when applied in the context of neural networks and through the work of Paul Werbos, David E. Rumelhart, Geoffrey E. Hinton and Ronald J. Williams that it gained recognition.

Then

Something Notable

It led to a "renaissance" in the field of artificial neural network research.

Nevertheless

During the 2000s it fell out of favour but has returned again in the 2010s, now able to train much larger networks using huge modern computing power such as GPUs.

Cinvestav

Then

Something Notable

It led to a "renaissance" in the field of artificial neural network research.

Nevertheless

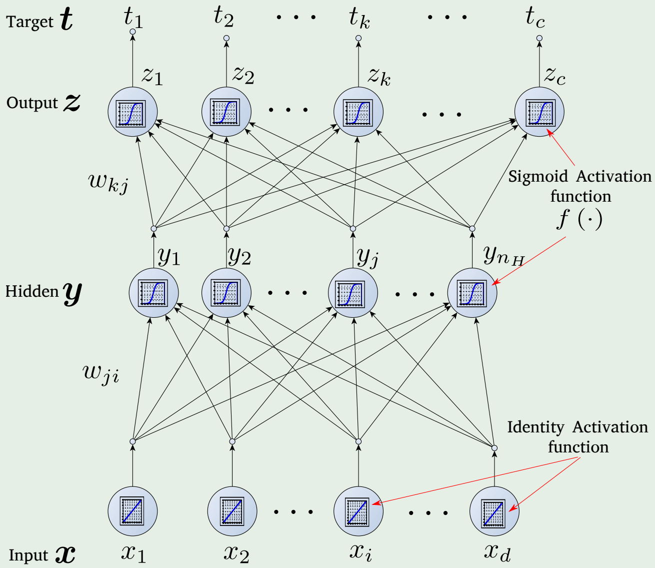
During the 2000s it fell out of favour but has returned again in the 2010s, now able to train much larger networks using huge modern computing power such as GPUs.

Outline

Cinvestav

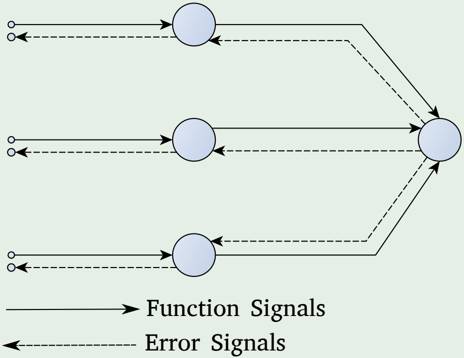
Multi-Layer Perceptron (MLP)

Multi-Layer Architecture.



Information Flow

We have the following information flow



Explanation

Problems with Hidden Layers

1. Increase complexity of Training
2. It is necessary to think about "Long and Narrow" network vs "Short and Fat" network.

Explanation

Problems with Hidden Layers

1. Increase complexity of Training
2. It is necessary to think about "Long and Narrow" network vs "Short and Fat" network.

Intuition for a One Hidden Layer

1. For every input case of region, that region can be delimited by hyperplanes on all sides using hidden units on the first hidden layer.
2. A hidden unit in the second layer than ANDs them together to bound the region.

Explanation

Problems with Hidden Layers

1. Increase complexity of Training
2. It is necessary to think about "Long and Narrow" network vs "Short and Fat" network.

Intuition for a One Hidden Layer

1. For every input case of region, that region can be delimited by hyperplanes on all sides using hidden units on the first hidden layer.
2. A hidden unit in the second layer than ANDs them together to bound the region.

Advantages

It has been proven that an MLP with one hidden layer can learn any nonlinear function of the input.

Explanation

Problems with Hidden Layers

1. Increase complexity of Training
2. It is necessary to think about "Long and Narrow" network vs "Short and Fat" network.

Intuition for a One Hidden Layer

1. For every input case of region, that region can be delimited by hyperplanes on all sides using hidden units on the first hidden layer.
2. A hidden unit in the second layer than ANDs them together to bound the region.

Advantages

It has been proven that an MLP with one hidden layer can learn any nonlinear function of the input.

Explanation

Problems with Hidden Layers

1. Increase complexity of Training
2. It is necessary to think about "Long and Narrow" network vs "Short and Fat" network.

Intuition for a One Hidden Layer

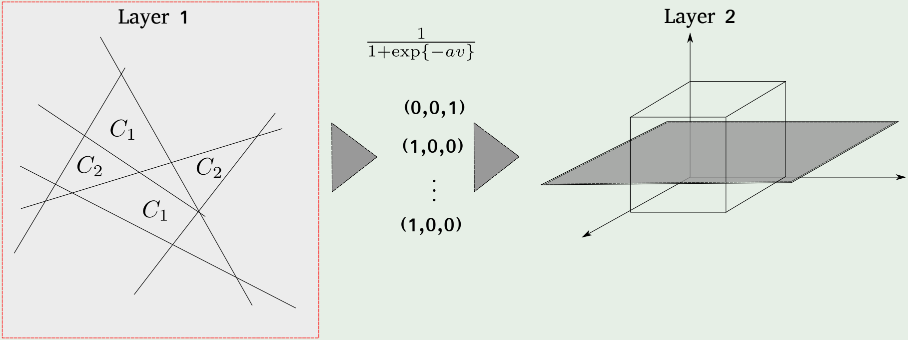
1. For every input case of region, that region can be delimited by hyperplanes on all sides using hidden units on the first hidden layer.
2. A hidden unit in the second layer than ANDs them together to bound the region.

Advantages

It has been proven that an MLP with one hidden layer can learn any nonlinear function of the input.

The Process

We have something like this



Outline

Cinvestav

Remember!!! The Quadratic Learning Error function

Cost Function our well know error at pattern m

$$J(m) = \frac{1}{2} e_k^2(m) \quad (1)$$

Delta Rule or Widrow-Hoff Rule

$$\Delta w_{kj}(m) = -\eta e_k(m) x_j(m) \quad (2)$$

Actually this is know as Gradient Descent

$$w_{kj}(m+1) = w_{kj}(m) + \Delta w_{kj}(m) \quad (3)$$

Remember!!! The Quadratic Learning Error function

Cost Function our well know error at **pattern** m

$$J\left(m\right) = \frac{1}{2} e_k^2\left(m\right) \quad (1)$$

Delta Rule or Widrow-Hoff Rule

$$\Delta w_{kj}\left(m\right) = -\eta e_k\left(m\right) x_j(m) \quad (2)$$

Actually this is know as Gradient Descent

$$w_{kj}\left(m+1\right) = w_{kj}\left(m\right) + \Delta w_{kj}\left(m\right) \quad (3)$$

Remember!!! The Quadratic Learning Error function

Cost Function our well know error at pattern m

$$J(m) = \frac{1}{2} e_k^2(m) \quad (1)$$

Delta Rule or Widrow-Hoff Rule

$$\Delta w_{kj}(m) = -\eta e_k(m) x_j(m) \quad (2)$$

Actually this is know as Gradient Descent

$$w_{kj}(m+1) = w_{kj}(m) + \Delta w_{kj}(m) \quad (3)$$

Cinvestav

Back-propagation

Setup

Let t_k be the k-th target (or desired) output and z_k be the k-th computed output with $k = 1, \dots, d$ and $\boldsymbol{w}$ represents all the weights of the network

Training Error for a single Pattern or Sample!!!

$$J(\boldsymbol{w}) = \frac{1}{2}\sum_{k=1}^{c}(t_k - z_k)^2 = \frac{1}{2}\|\boldsymbol{t} - \boldsymbol{z}\|^2 \quad (4)$$

Cinvestav

Back-propagation

Setup

Let t_k be the k-th target (or desired) output and z_k be the k-th computed output with $k = 1, \ldots, d$ and $\boldsymbol{w}$ represents all the weights of the network

Training Error for a single Pattern or Sample!!!

$$J(\boldsymbol{w}) = \frac{1}{2} \sum_{k=1}^{c} (t_k - z_k)^2 = \frac{1}{2} \|\boldsymbol{t} - \boldsymbol{z}\|^2 \quad (4)$$

Cinvestav

Outline

Cinvestav

Gradient Descent

Gradient Descent

The back-propagation learning rule is based on gradient descent.

Reducing the Error

The weights are initialized with pseudo-random values and are changed in a direction that will reduce the error:

$$\Delta w = -\eta \frac{\partial J}{\partial w} \tag{5}$$

Where

η is the learning rate which indicates the relative size of the change in weights:

$$w(m+1) = w(m) + \Delta w(m) \tag{6}$$

where m is the m-th pattern presented

Gradient Descent

Gradient Descent

The back-propagation learning rule is based on gradient descent.

Reducing the Error

The weights are initialized with pseudo-random values and are changed in a direction that will reduce the error:

$$\Delta \boldsymbol{w} = -\eta \frac{\partial J}{\partial \boldsymbol{w}} \tag{5}$$

Where

η is the learning rate which indicates the relative size of the change in weights:

$$w(m+1) = w(m) + \Delta w(m) \tag{6}$$

where m is the m-th pattern presented

Gradient Descent

Gradient Descent

The back-propagation learning rule is based on gradient descent.

Reducing the Error

The weights are initialized with pseudo-random values and are changed in a direction that will reduce the error:

$$\Delta \boldsymbol{w} = -\eta \frac{\partial J}{\partial \boldsymbol{w}} \tag{5}$$

Where

η is the learning rate which indicates the relative size of the change in weights:

$$w\left(m+1\right) = w\left(m\right) + \Delta w\left(m\right) \tag{6}$$

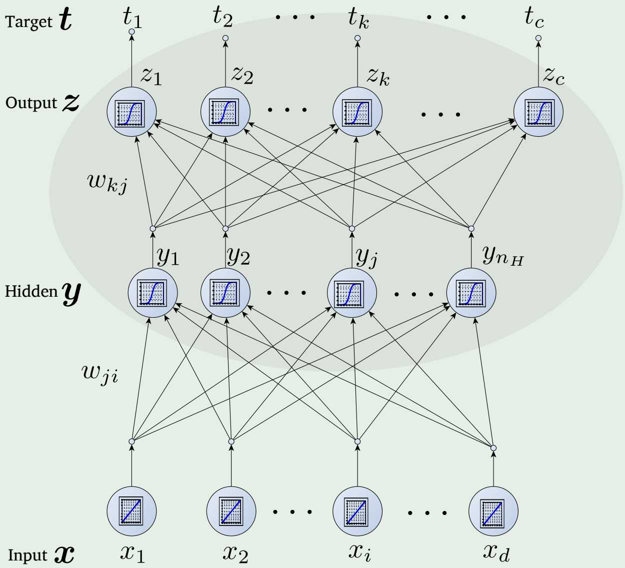
where m is the m-th pattern presented

Outline

Cinvestav

Multilayer Architecture

Multilayer Architecture: hidden–to-output weights



Observation about the activation function

Hidden Output is equal to

$$y_j = f\left(\sum_{i=1}^{d} w_{ji} x_i\right)$$

Output is equal to

$$z_k = f\left(\sum_{j=1}^{y_{n_H}} w_{kj} y_j\right)$$

Observation about the activation function

Hidden Output is equal to

$$y_j = f\left(\sum_{i=1}^{d} w_{ji} x_i\right)$$

Output is equal to

$$z_k = f\left(\sum_{j=1}^{y_{n_H}} w_{kj} y_j\right)$$

Hidden–to-Output Weights

Error on the hidden–to-output weights

$$\frac{\partial J}{\partial w_{kj}} = \frac{\partial J}{\partial net_k} \cdot \frac{\partial net_k}{\partial w_{kj}} = -\delta_k \cdot \frac{\partial net_k}{\partial w_{kj}} \quad (7)$$

net_k

It describes how the overall error changes with the activation of the unit's net:

$$net_k = \sum_{j=1}^{n_H} w_{kj} y_j = w_k^T \cdot y \quad (8)$$

Now

$$\delta_k = -\frac{\partial J}{\partial net_k} = -\frac{\partial J}{\partial z_k} \cdot \frac{\partial z_k}{\partial net_k} = (t_k - z_k) f'(net_k) \quad (9)$$

Hidden–to-Output Weights

Error on the hidden–to-output weights

$$\frac{\partial J}{\partial w_{kj}} = \frac{\partial J}{\partial net_k} \cdot \frac{\partial net_k}{\partial w_{kj}} = -\delta_k \cdot \frac{\partial net_k}{\partial w_{kj}} \tag{7}$$

net_k

It describes how the overall error changes with the activation of the unit's net:

$$net_k = \sum_{j=1}^{y_{n_H}} w_{kj} y_j = \boldsymbol{w}_k^T \cdot \boldsymbol{y} \tag{8}$$

Now

$$\delta_k = -\frac{\partial J}{\partial net_k} = -\frac{\partial J}{\partial z_k} \cdot \frac{\partial z_k}{\partial net_k} = (t_k - z_k)\, f'(net_k) \tag{9}$$

Hidden–to-Output Weights

Error on the hidden–to-output weights

$$\frac{\partial J}{\partial w_{kj}} = \frac{\partial J}{\partial net_k} \cdot \frac{\partial net_k}{\partial w_{kj}} = -\delta_k \cdot \frac{\partial net_k}{\partial w_{kj}} \quad (7)$$

net_k

It describes how the overall error changes with the activation of the unit's net:

$$net_k = \sum_{j=1}^{y_{n_H}} w_{kj} y_j = \boldsymbol{w}_k^T \cdot \boldsymbol{y} \quad (8)$$

Now

$$\delta_k = -\frac{\partial J}{\partial net_k} = -\frac{\partial J}{\partial z_k} \cdot \frac{\partial z_k}{\partial net_k} = (t_k - z_k)\, f'(net_k) \quad (9)$$

Hidden–to-Output Weights

Why?

$$z_k = f\left(net_k\right) \quad (10)$$

Thus

$$\frac{\partial z_k}{\partial net_k} = f'\left(net_k\right) \quad (11)$$

Since $net_k = w_k^T \cdot y$ therefore:

$$\frac{\partial net_k}{\partial w_{kj}} = y_j \quad (12)$$

Hidden–to-Output Weights

Why?

$$z_k = f\left(net_k\right) \quad (10)$$

Thus

$$\frac{\partial z_k}{\partial net_k} = f'\left(net_k\right) \quad (11)$$

Since $net_k = w_k^T \cdot y$ therefore:

$$\frac{\partial net_k}{\partial w_{kj}} = y_j \quad (12)$$

Hidden–to-Output Weights

Why?

$$z_k = f\left(net_k\right) \tag{10}$$

Thus

$$\frac{\partial z_k}{\partial net_k} = f'\left(net_k\right) \tag{11}$$

Since $net_k = \boldsymbol{w}_k^T \cdot \boldsymbol{y}$ therefore:

$$\frac{\partial net_k}{\partial w_{kj}} = y_j \tag{12}$$

Cinvestav

Finally

The weight update (or learning rule) for the hidden-to-output weights is:

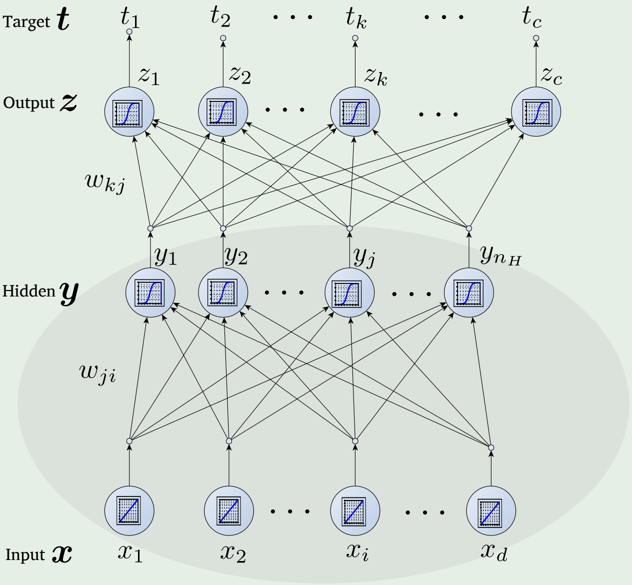
$$\triangle w_{kj} = \eta \delta_k y_j = \eta \left(t_k - z_k \right) f' \left(net_k \right) y_j \tag{13}$$

Outline

Cinvestav

Multi-Layer Architecture

Multi-Layer Architecture: Input–to-Hidden weights



Input–to-Hidden Weights

Error on the Input–to-Hidden weights

$$\frac{\partial J}{\partial w_{ji}} = \frac{\partial J}{\partial y_j} \cdot \frac{\partial y_j}{\partial net_j} \cdot \frac{\partial net_j}{\partial w_{ji}} \tag{14}$$

Thus

$$\begin{aligned}
\frac{\partial J}{\partial y_j} &= \frac{\partial}{\partial y_j}\left[\frac{1}{2}\sum_{k=1}^{c}\left(t_k - z_k\right)^2\right] \\
&= -\sum_{k=1}^{c}\left(t_k - z_k\right)\frac{\partial z_k}{\partial y_j} \\
&= -\sum_{k=1}^{c}\left(t_k - z_k\right)\frac{\partial z_k}{\partial net_k} \cdot \frac{\partial net_k}{\partial y_j} \\
&= -\sum_{k=1}^{c}\left(t_k - z_k\right)\frac{\partial f\left(net_k\right)}{\partial net_k} \cdot w_{kj}
\end{aligned}$$

Input–to-Hidden Weights

Error on the Input–to-Hidden weights

$$\frac{\partial J}{\partial w_{ji}} = \frac{\partial J}{\partial y_j} \cdot \frac{\partial y_j}{\partial net_j} \cdot \frac{\partial net_j}{\partial w_{ji}} \quad (14)$$

Thus

$$\begin{aligned}
\frac{\partial J}{\partial y_j} &= \frac{\partial}{\partial y_j}\left[\frac{1}{2}\sum_{k=1}^{c}(t_k - z_k)^2\right] \\
&= -\sum_{k=1}^{c}(t_k - z_k)\frac{\partial z_k}{\partial y_j} \\
&= -\sum_{k=1}^{c}(t_k - z_k)\frac{\partial z_k}{\partial net_k} \cdot \frac{\partial net_k}{\partial y_j} \\
&= -\sum_{k=1}^{c}(t_k - z_k)\frac{\partial f(net_k)}{\partial net_k} \cdot w_{kj}
\end{aligned}$$

Input–to-Hidden Weights

Finally

$$\frac{\partial J}{\partial y_j} = -\sum_{k=1}^{c} (t_k - z_k) f'(net_k) \cdot w_{kj} \quad (15)$$

Remember

$$\delta_k = -\frac{\partial J}{\partial net_k} = (t_k - z_k) f'(net_k) \quad (16)$$

Cinvestav

What is $\frac{\partial y_j}{\partial net_j}$?

First

$$net_j = \sum_{i=1}^{d} w_{ji} x_i = \boldsymbol{w}_j^T \cdot \boldsymbol{x} \tag{17}$$

Then

$$y_j = f\left(net_j\right)$$

Then

$$\frac{\partial y_j}{\partial net_j} = \frac{\partial f\left(net_j\right)}{\partial net_j} = f'\left(net_j\right)$$

What is $\frac{\partial y_j}{\partial net_j}$?

First

$$net_j = \sum_{i=1}^{d} w_{ji} x_i = \boldsymbol{w}_j^T \cdot \boldsymbol{x} \tag{17}$$

Then

$$y_j = f\left(net_j\right)$$

Then

$$\frac{\partial y_j}{\partial net_j} = \frac{\partial f\left(net_j\right)}{\partial net_j} = f'\left(net_j\right)$$

What is $\frac{\partial y_j}{\partial net_j}$?

First

$$net_j = \sum_{i=1}^{d} w_{ji} x_i = \boldsymbol{w}_j^T \cdot \boldsymbol{x} \tag{17}$$

Then

$$y_j = f\left(net_j\right)$$

Then

$$\frac{\partial y_j}{\partial net_j} = \frac{\partial f\left(net_j\right)}{\partial net_j} = f'\left(net_j\right)$$

Then, we can define δ_j

By defying the sensitivity for a hidden unit:

$$\delta_j = f'(net_j) \sum_{k=1}^{c} w_{kj}\delta_k \tag{18}$$

Which means that:

"The sensitivity at a hidden unit is simply the sum of the individual sensitivities at the output units weighted by the **hidden-to-output weights** w_{kj}; all multiplied by $f'(net_j)$"

Then, we can define δ_j

By defying the sensitivity for a hidden unit:

$$\delta_j = f'\left(net_j\right) \sum_{k=1}^{c} w_{kj}\delta_k \tag{18}$$

Which means that:

"The sensitivity at a hidden unit is simply the sum of the individual sensitivities at the output units weighted by the **hidden-to-output weights** w_{kj}; all multiplied by $f'\left(net_j\right)$"

Cinvestav

Then, we can define δ_j

By defying the sensitivity for a hidden unit:

$$\delta_j = f'\left(net_j\right) \sum_{k=1}^{c} w_{kj}\delta_k \tag{18}$$

Which means that:

“The sensitivity at a hidden unit is simply the sum of the individual sensitivities at the output units weighted by the **hidden-to-output weights** w_{kj}; all multiplied by $f'\left(net_j\right)$”

What about $\frac{\partial net_j}{\partial w_{ji}}$?

We have that

$$\frac{\partial net_j}{\partial w_{ji}} = \frac{\partial \boldsymbol{w}_j^T \cdot \boldsymbol{x}}{\partial w_{ji}} = \frac{\partial \sum_{i=1}^{d} w_{ji} x_i}{\partial w_{ji}} = x_i$$

Finally

The learning rule for the input-to-hidden weights is:

$$\Delta w_{ji} = \eta x_i \delta_j = \eta \left[\sum_{k=1}^{c} w_{kj} \delta_k \right] f'\left(net_j\right) x_i \qquad (19)$$

Basically, the entire training process has the following steps

Initialization

Assuming that no prior information is available, pick the synaptic weights and thresholds

Forward Computation

Compute the induced function signals of the network by proceeding forward through the network, layer by layer.

Backward Computation

Compute the local gradients of the network.

Finally

Adjust the weights!!!

Basically, the entire training process has the following steps

Initialization

Assuming that no prior information is available, pick the synaptic weights and thresholds

Forward Computation

Compute the induced function signals of the network by proceeding forward through the network, layer by layer.

Backward Computation

Compute the local gradients of the network.

Finally

Adjust the weights!!!

Cinvestav

Basically, the entire training process has the following steps

Initialization

Assuming that no prior information is available, pick the synaptic weights and thresholds

Forward Computation

Compute the induced function signals of the network by proceeding forward through the network, layer by layer.

Backward Computation

Compute the local gradients of the network.

Finally

Adjust the weights!!!

Basically, the entire training process has the following steps

Initialization

Assuming that no prior information is available, pick the synaptic weights and thresholds

Forward Computation

Compute the induced function signals of the network by proceeding forward through the network, layer by layer.

Backward Computation

Compute the local gradients of the network.

Finally

Adjust the weights!!!

Outline

Cinvestav

Now, Calculating Total Change

We have for that

The Total Training Error is the sum over the errors of N individual patterns

The Total Training Error

$$J = \sum_{p=1}^{N} J_p = \frac{1}{2} \sum_{p=1}^{N} \sum_{k=1}^{d} \left(t_k^p - z_k^p\right)^2 = \frac{1}{2} \sum_{p=1}^{n} \left\| t^p - z^p \right\|^2 \quad (20)$$

Now, Calculating Total Change

We have for that

The Total Training Error is the sum over the errors of N individual patterns

The Total Training Error

$$J = \sum_{p=1}^{N} J_p = \frac{1}{2} \sum_{p=1}^{N} \sum_{k=1}^{d} \left(t_k^p - z_k^p\right)^2 = \frac{1}{2} \sum_{p=1}^{n} \left\| \boldsymbol{t}^p - \boldsymbol{z}^p \right\|^2 \qquad (20)$$

About the Total Training Error

Remarks

- A weight update may reduce the error on the single pattern being presented but can increase the error on the full training set.
- However, given a large number of such individual updates, the total error of equation (20) decreases.

Cinvestav

About the Total Training Error

Remarks

- A weight update may reduce the error on the single pattern being presented but can increase the error on the full training set.
- However, given a large number of such individual updates, the total error of equation (20) decreases.

Cinvestav

Outline

Cinvestav

Now, we want the training to stop

Therefore

It is necessary to have a way to stop when the change of the weights is enough!!!

Cinvestav

Now, we want the training to stop

Therefore

It is necessary to have a way to stop when the change of the weights is enough!!!

A simple way to stop the training

- The algorithm terminates when the change in the criterion function $J(\boldsymbol{w})$ is smaller than some preset value Θ.

$$\Delta J(\boldsymbol{w}) = |J(\boldsymbol{w}(t+1)) - J(\boldsymbol{w}(t))| \tag{21}$$

- There are other stopping criteria that lead to better performance than this one.

Cinvestav

Now, we want the training to stop

Therefore

It is necessary to have a way to stop when the change of the weights is enough!!!

A simple way to stop the training

- The algorithm terminates when the change in the criterion function $J(\boldsymbol{w})$ is smaller than some preset value Θ.

$$\Delta J(\boldsymbol{w}) = \left| J(\boldsymbol{w}(t+1)) - J(\boldsymbol{w}(t)) \right| \tag{21}$$

- There are other stopping criteria that lead to better performance than this one.

Now, we want the training to stop

Therefore

It is necessary to have a way to stop when the change of the weights is enough!!!

A simple way to stop the training

- The algorithm terminates when the change in the criterion function $J(\boldsymbol{w})$ is smaller than some preset value Θ.

$$\Delta J(\boldsymbol{w}) = \left|J(\boldsymbol{w}(t+1)) - J(\boldsymbol{w}(t))\right| \tag{21}$$

- There are other stopping criteria that lead to better performance than this one.

Other Stopping Criteria

Norm of the Gradient

The back-propagation algorithm is considered to have converged when the Euclidean norm of the gradient vector reaches a sufficiently small gradient threshold.

$$\left\|\nabla_{\boldsymbol{w}} J\left(m\right)\right\| < \Theta \tag{22}$$

Rate of change in the average error per epoch

The back-propagation algorithm is considered to have converged when the absolute rate of change in the average squared error per epoch is sufficiently small.

$$\left|\frac{1}{N}\sum_{p=1}^{N} J_p\right| < \Theta \tag{23}$$

Other Stopping Criteria

Norm of the Gradient

The back-propagation algorithm is considered to have converged when the Euclidean norm of the gradient vector reaches a sufficiently small gradient threshold.

$$\left\| \nabla_{\boldsymbol{w}} J\left(m\right) \right\| < \Theta \tag{22}$$

Rate of change in the average error per epoch

The back-propagation algorithm is considered to have converged when the absolute rate of change in the average squared error per epoch is sufficiently small.

$$\left| \frac{1}{N} \sum_{p=1}^{N} J_p \right| < \Theta \tag{23}$$

About the Stopping Criteria

Observations

1. Before training starts, the error on the training set is high.
    - Through the learning process, the error becomes smaller.
2. The error per pattern depends on the amount of training data and the expressive power (such as the number of weights) in the network.
3. The average error on an independent test set is always higher than on the training set, and it can decrease as well as increase.
4. A validation set is used in order to decide when to stop training.
    - We do not want to over-fit the network and decrease the power of the classifier generalization "we stop training at a minimum of the error on the validation set"

About the Stopping Criteria

Observations

1. Before training starts, the error on the training set is high.
   - Through the learning process, the error becomes smaller.
2. The error per pattern depends on the amount of training data and the expressive power (such as the number of weights) in the network.
3. The average error on an independent test set is always higher than on the training set, and it can decrease as well as increase.
4. A validation set is used in order to decide when to stop training.
   - We do not want to over-fit the network and decrease the power of the classifier generalization "we stop training at a minimum of the error on the validation set"

Cinvestav

About the Stopping Criteria

Observations

1. Before training starts, the error on the training set is high.
   - Through the learning process, the error becomes smaller.
2. The error per pattern depends on the amount of training data and the expressive power (such as the number of weights) in the network.
3. The average error on an independent test set is always higher than on the training set, and it can decrease as well as increase.
4. A validation set is used in order to decide when to stop training.
   - We do not want to over-fit the network and decrease the power of the classifier generalization "we stop training at a minimum of the error on the validation set"

About the Stopping Criteria

Observations

1. Before training starts, the error on the training set is high.
   - Through the learning process, the error becomes smaller.
2. The error per pattern depends on the amount of training data and the expressive power (such as the number of weights) in the network.
3. The average error on an independent test set is always higher than on the training set, and it can decrease as well as increase.
4. A validation set is used in order to decide when to stop training.
   - We do not want to over-fit the network and decrease the power of the classifier generalization "we stop training at a minimum of the error on the validation set"

Cinvestav

About the Stopping Criteria

Observations

1. Before training starts, the error on the training set is high.
   - Through the learning process, the error becomes smaller.
2. The error per pattern depends on the amount of training data and the expressive power (such as the number of weights) in the network.
3. The average error on an independent test set is always higher than on the training set, and it can decrease as well as increase.
4. A validation set is used in order to decide when to stop training.
   - We do not want to over-fit the network and decrease the power of the classifier generalization "we stop training at a minimum of the error on the validation set"

About the Stopping Criteria

Observations

1. Before training starts, the error on the training set is high.
   - Through the learning process, the error becomes smaller.
2. The error per pattern depends on the amount of training data and the expressive power (such as the number of weights) in the network.
3. The average error on an independent test set is always higher than on the training set, and it can decrease as well as increase.
4. A validation set is used in order to decide when to stop training.
   - We do not want to over-fit the network and decrease the power of the classifier generalization "we stop training at a minimum of the error on the validation set"

Some More Terminology

Epoch

As with other types of backpropagation, 'learning' is a supervised process that occurs with each cycle or 'epoch' through a forward activation flow of outputs, and the backwards error propagation of weight adjustments.

In our case

I am using the batch sum of all correcting weights to define that epoch.

Some More Terminology

Epoch

As with other types of backpropagation, 'learning' is a supervised process that occurs with each cycle or 'epoch' through a forward activation flow of outputs, and the backwards error propagation of weight adjustments.

In our case

I am using the batch sum of all correcting weights to define that epoch.

Outline

Cinvestav

Final Basic Batch Algorithm

Perceptron($\boldsymbol{X}$)

Initialize random w, **number of hidden units** n_H, **number of outputs** z, **stopping criterion** Θ, **learning rate** η, **epoch** $m = 0$
do
  $m = m + 1$
    for $s = 1$ **to** N
      $x(m) = X(:, s)$
      for $k = 1$ **to** c
        $\delta_k = (t_k - z_k) f'\left(w_k^T \cdot y\right)$
        for $j = 1$ **to** n_H
          $net_j = w_j^T \cdot x; y_j = f\left(net_j\right)$
          $w_{kj}(m) = w_{kj}(m) + \eta \delta_k y_j(m)$
      for $j = 1$ **to** n_H
        $\delta_j = f'\left(net_j\right) \sum_{k=1}^{c} w_{kj} \delta_k$
        for $i = 1$ **to** d
          $w_{ji}(m) = w_{ji}(m) + \eta \delta_j x_i(m)$
until $\|\nabla_w J(m)\| < \Theta$
return $w(m)$

Final Basic Batch Algorithm

Perceptron$(\boldsymbol{X})$

1. **Initialize random w, number of hidden units n_H, number of outputs z, stopping criterion Θ, learning rateη, epoch** $m = 0$
2. do
3. $m = m + 1$
4. for $s = 1$ to N
5. $x(m) = X(:, s)$
6. for $k = 1$ to c
7. $\delta_k = (t_k - z_k) f'\left(w_k^T \cdot y\right)$
8. for $j = 1$ to n_H
9. $net_j = w_j^T \cdot x; y_j = f\left(net_j\right)$
10. $w_{kj}(m) = w_{kj}(m) + \eta \delta_k y_j(m)$
11. for $j = 1$ to n_H
12. $\delta_j = f'\left(net_j\right) \sum_{k=1}^{c} w_{kj} \delta_k$
13. for $i = 1$ to d
14. $w_{ji}(m) = w_{ji}(m) + \eta \delta_j x_i(m)$
15. until $\|\nabla_w J(m)\| < \Theta$
16. return $w(m)$

Final Basic Batch Algorithm

Perceptron$(\boldsymbol{X})$

1. **Initialize random w, number of hidden units n_H, number of outputs z, stopping criterion Θ, learning rateη, epoch** $m = 0$
2. **do**
3. $m = m + 1$
4. for $s = 1$ to N
5. $x(m) = X(:, s)$
6. for $k = 1$ to c
7. $\delta_k = (t_k - z_k) f'\left(w_k^T \cdot y\right)$
8. for $j = 1$ to n_H
9. $net_j = w_j^T \cdot x; y_j = f\left(net_j\right)$
10. $w_{kj}(m) = w_{kj}(m) + \eta \delta_k y_j(m)$
11. for $j = 1$ to n_H
12. $\delta_j = f'\left(net_j\right) \sum_{k=1}^{c} w_{kj} \delta_k$
13. for $i = 1$ to d
14. $w_{ji}(m) = w_{ji}(m) + \eta \delta_j x_i(m)$
15. until $\|\nabla_w J(m)\| < \Theta$
16. return $w(m)$

Final Basic Batch Algorithm

Perceptron$(\boldsymbol{X})$

1. **Initialize random** w**, number of hidden units** n_H**, number of outputs** z**, stopping criterion** Θ**, learning rate**η**, epoch** $m = 0$
2. **do**
3. $m = m + 1$
4. for $s = 1$ to N
5. $x(m) = X(:, s)$
6. for $k = 1$ to c
7. $\delta_k = (t_k - z_k) f'\left(w_k^T \cdot y\right)$
8. for $j = 1$ to n_H
9. $net_j = w_j^T \cdot x; y_j = f\left(net_j\right)$
10. $w_{kj}(m) = w_{kj}(m) + \eta \delta_k y_j(m)$
11. for $j = 1$ to n_H
12. $\delta_j = f'\left(net_j\right) \sum_{k=1}^{c} w_{kj} \delta_k$
13. for $i = 1$ to d
14. $w_{ji}(m) = w_{ji}(m) + \eta \delta_j x_i(m)$
15. until $\|\nabla_w J(m)\| < \Theta$
16. return $w(m)$

Final Basic Batch Algorithm

Perceptron$(\boldsymbol{X})$

1. **Initialize random** w, **number of hidden units** n_H, **number of outputs** z, **stopping criterion** Θ, **learning rate** η, **epoch** $m = 0$
2. **do**
3. $\quad m = m + 1$
4. $\qquad$ **for** $s = 1$ **to** N
5. $\qquad\quad x(m) = X(:, s)$
6. $\qquad\quad$ for $k = 1$ to c
7. $\qquad\qquad \delta_k = (t_k - z_k) f'\left(w_k^T \cdot y\right)$
8. $\qquad\qquad$ for $j = 1$ to n_H
9. $\qquad\qquad\quad net_j = w_j^T \cdot x; y_j = f\left(net_j\right)$
10. $\qquad\qquad\quad w_{kj}(m) = w_{kj}(m) + \eta \delta_k y_j(m)$
11. $\qquad\quad$ for $j = 1$ to n_H
12. $\qquad\qquad \delta_j = f'\left(net_j\right) \sum_{k=1}^{c} w_{kj} \delta_k$
13. $\qquad\qquad$ for $i = 1$ to d
14. $\qquad\qquad\quad w_{ji}(m) = w_{ji}(m) + \eta \delta_j x_i(m)$
15. until $\|\nabla_w J(m)\| < \Theta$
16. return $w(m)$

Final Basic Batch Algorithm

Perceptron$(\boldsymbol{X})$

1. **Initialize random** w, **number of hidden units** n_H, **number of outputs** z, **stopping criterion** Θ, **learning rate**η, **epoch**

   $m = 0$
2. **do**
3. $\quad m = m + 1$
4. $\quad\quad$ **for** $s = 1$ **to** N
5. $\quad\quad\quad \boldsymbol{x}(m) = \boldsymbol{X}(:, s)$
6. $\quad\quad\quad$ for $k = 1$ to c
7. $\quad\quad\quad\quad \delta_k = (t_k - z_k) f'\left(w_k^T \cdot y\right)$
8. $\quad\quad\quad\quad$ for $j = 1$ to n_H
9. $\quad\quad\quad\quad\quad net_j = w_j^T \cdot x; y_j = f\left(net_j\right)$
10. $\quad\quad\quad\quad\quad w_{kj}(m) = w_{kj}(m) + \eta \delta_k y_j(m)$
11. $\quad\quad\quad$ for $j = 1$ to n_H
12. $\quad\quad\quad\quad \delta_j = f'\left(net_j\right) \sum_{k=1}^{c} w_{kj} \delta_k$
13. $\quad\quad\quad\quad$ for $i = 1$ to d
14. $\quad\quad\quad\quad\quad w_{ji}(m) = w_{ji}(m) + \eta \delta_j x_i(m)$
15. until $\|\nabla_w J(m)\| < \Theta$
16. return $w(m)$

Final Basic Batch Algorithm

Perceptron$(\boldsymbol{X})$

1. **Initialize random** w**, number of hidden units** n_H**, number of outputs** $\boldsymbol{z}$**, stopping criterion** Θ**, learning rate**η**, epoch** $m = 0$
2. **do**
3. $m = m + 1$
4. **for** $s = 1$ **to** N
5. $\boldsymbol{x}(m) = \boldsymbol{X}(:, s)$
6. **for** $k = 1$ **to** c
7. $\delta_k = (t_k - z_k) f'\left(w_k^T \cdot y\right)$
8. for $j = 1$ to n_H
9. $net_j = w_j^T \cdot x; y_j = f\left(net_j\right)$
10. $w_{kj}(m) = w_{kj}(m) + \eta \delta_k y_j(m)$
11. for $j = 1$ to n_H
12. $\delta_j = f'\left(net_j\right) \sum_{k=1}^{c} w_{kj} \delta_k$
13. for $i = 1$ to d
14. $w_{ji}(m) = w_{ji}(m) + \eta \delta_j x_i(m)$
15. until $\|\nabla_w J(m)\| < \Theta$
16. return $w(m)$

Final Basic Batch Algorithm

Perceptron$(\boldsymbol{X})$

1. **Initialize random** w, **number of hidden units** n_H, **number of outputs** $\boldsymbol{z}$, **stopping criterion** Θ, **learning rate**η, **epoch**

   $m = 0$
2. **do**
3. $m = m + 1$
4. **for** $s = 1$ **to** N
5. $\boldsymbol{x}(m) = \boldsymbol{X}(:, s)$
6. **for** $k = 1$ **to** c
7. $\delta_k = (t_k - z_k) f'\left(\boldsymbol{w}_k^T \cdot \boldsymbol{y}\right)$
8. **for** $j = 1$ **to** n_H
9. $net_j = \boldsymbol{w}_j^T \cdot \boldsymbol{x}; y_j = f\left(net_j\right)$
10. $w_{kj}(m) = w_{kj}(m) + \eta \delta_k y_j(m)$
11. **for** $j = 1$ **to** n_H
12. $\delta_j = f'\left(net_j\right) \sum_{k=1}^{c} w_{kj} \delta_k$
13. **for** $i = 1$ **to** d
14. $w_{ji}(m) = w_{ji}(m) + \eta \delta_j x_i(m)$
15. **until** $\|\nabla_w J(m)\| < \Theta$
16. **return** $w(m)$

Final Basic Batch Algorithm

Perceptron$(\boldsymbol{X})$

1. **Initialize random** w, **number of hidden units** n_H, **number of outputs** z, **stopping criterion** Θ, **learning rate** η, **epoch**

   $m = 0$

2. **do**
3. $\quad m = m + 1$
4. $\quad\quad$ **for** $s = 1$ **to** N
5. $\quad\quad\quad \boldsymbol{x}(m) = \boldsymbol{X}(:, s)$
6. $\quad\quad\quad$ **for** $k = 1$ **to** c
7. $\quad\quad\quad\quad \delta_k = (t_k - z_k) f'\left(\boldsymbol{w}_k^T \cdot \boldsymbol{y}\right)$
8. $\quad\quad\quad\quad$ **for** $j = 1$ **to** n_H
9. $\quad\quad\quad\quad\quad net_j = w_j^T \cdot x; y_j = f\left(net_j\right)$
10. $\quad\quad\quad\quad\quad w_{kj}(m) = w_{kj}(m) + \eta \delta_k y_j(m)$
11. $\quad\quad\quad$ **for** $j = 1$ **to** n_H
12. $\quad\quad\quad\quad \delta_j = f'\left(net_j\right) \sum_{k=1}^{c} w_{kj} \delta_k$
13. $\quad\quad\quad\quad$ **for** $i = 1$ **to** d
14. $\quad\quad\quad\quad\quad w_{ji}(m) = w_{ji}(m) + \eta \delta_j x_i(m)$
15. **until** $\|\nabla_w J(m)\| < \Theta$
16. **return** $w(m)$

Final Basic Batch Algorithm

Perceptron$(\boldsymbol{X})$

1. **Initialize random** w, **number of hidden units** n_H, **number of outputs** $\boldsymbol{z}$, **stopping criterion** Θ, **learning rate**η, **epoch**

   $m = 0$
2. **do**
3. $\quad m = m + 1$
4. $\quad\quad$ **for** $s = 1$ **to** N
5. $\quad\quad\quad \boldsymbol{x}(m) = \boldsymbol{X}(:, s)$
6. $\quad\quad\quad$ **for** $k = 1$ **to** c
7. $\quad\quad\quad\quad \delta_k = (t_k - z_k) f'\left(\boldsymbol{w}_k^T \cdot \boldsymbol{y}\right)$
8. $\quad\quad\quad\quad$ **for** $j = 1$ **to** n_H
9. $\quad\quad\quad\quad\quad net_j = \boldsymbol{w}_j^T \cdot \boldsymbol{x}; y_j = f\left(net_j\right)$
10. $\quad\quad\quad\quad\quad w_{kj}(m) = w_{kj}(m) + \eta \delta_k y_j(m)$
11. $\quad\quad\quad$ **for** $j = 1$ **to** n_H
12. $\quad\quad\quad\quad \delta_j = f'\left(net_j\right) \sum_{k=1}^{c} w_{kj} \delta_k$
13. $\quad\quad\quad\quad$ **for** $i = 1$ **to** d
14. $\quad\quad\quad\quad\quad w_{ji}(m) = w_{ji}(m) + \eta \delta_j x_i(m)$
15. **until** $\|\nabla_w J(m)\| < \Theta$
16. **return** $w(m)$

Final Basic Batch Algorithm

Perceptron$(\boldsymbol{X})$

1. **Initialize random** w, **number of hidden units** n_H, **number of outputs** z, **stopping criterion** Θ, **learning rate** η, **epoch**

   $m = 0$

2. **do**
3. $\quad m = m + 1$
4. $\quad\quad$ **for** $s = 1$ **to** N
5. $\quad\quad\quad \boldsymbol{x}(m) = \boldsymbol{X}(:, s)$
6. $\quad\quad\quad$ **for** $k = 1$ **to** c
7. $\quad\quad\quad\quad \delta_k = (t_k - z_k) f'\left(\boldsymbol{w}_k^T \cdot \boldsymbol{y}\right)$
8. $\quad\quad\quad\quad$ **for** $j = 1$ **to** n_H
9. $\quad\quad\quad\quad\quad net_j = \boldsymbol{w}_j^T \cdot \boldsymbol{x}; y_j = f\left(net_j\right)$
10. $\quad\quad\quad\quad\quad w_{kj}(m) = w_{kj}(m) + \eta \delta_k y_j(m)$
11. $\quad\quad\quad$ **for** $j = 1$ **to** n_H
12. $\quad\quad\quad\quad \delta_j = f'\left(net_j\right) \sum_{k=1}^{c} w_{kj} \delta_k$
13. $\quad\quad\quad\quad$ **for** $i = 1$ **to** d
14. $\quad\quad\quad\quad\quad w_{ji}(m) = w_{ji}(m) + \eta \delta_j x_i(m)$
15. **until** $\|\nabla_w J(m)\| < \Theta$
16. **return** $w(m)$

Final Basic Batch Algorithm

Perceptron$(\boldsymbol{X})$

1. **Initialize random** w, **number of hidden units** n_H, **number of outputs** z, **stopping criterion** Θ, **learning rate** η, **epoch**

   $m = 0$
2. **do**
3. $\quad m = m + 1$
4. $\quad\quad$ **for** $s = 1$ **to** N
5. $\quad\quad\quad \boldsymbol{x}(m) = \boldsymbol{X}(:, s)$
6. $\quad\quad\quad$ **for** $k = 1$ **to** c
7. $\quad\quad\quad\quad \delta_k = (t_k - z_k) f'\left(\boldsymbol{w}_k^T \cdot \boldsymbol{y}\right)$
8. $\quad\quad\quad\quad$ **for** $j = 1$ **to** n_H
9. $\quad\quad\quad\quad\quad net_j = \boldsymbol{w}_j^T \cdot \boldsymbol{x}; y_j = f\left(net_j\right)$
10. $\quad\quad\quad\quad\quad w_{kj}(m) = w_{kj}(m) + \eta \delta_k y_j(m)$
11. $\quad\quad\quad$ **for** $j = 1$ **to** n_H
12. $\quad\quad\quad\quad \delta_j = f'\left(net_j\right) \sum_{k=1}^{c} w_{kj} \delta_k$
13. $\quad\quad\quad\quad$ **for** $i = 1$ **to** d
14. $\quad\quad\quad\quad\quad w_{ji}(m) = w_{ji}(m) + \eta \delta_j x_i(m)$
15. **until** $\|\nabla_w J(m)\| < \Theta$
16. **return** $w(m)$

Final Basic Batch Algorithm

Perceptron$(\boldsymbol{X})$

1. **Initialize random** w, **number of hidden units** n_H, **number of outputs** z, **stopping criterion** Θ, **learning rate** η, **epoch** $m = 0$
2. **do**
3. $\quad m = m + 1$
4. $\quad\quad$ **for** $s = 1$ **to** N
5. $\quad\quad\quad \boldsymbol{x}(m) = \boldsymbol{X}(:, s)$
6. $\quad\quad\quad$ **for** $k = 1$ **to** c
7. $\quad\quad\quad\quad \delta_k = (t_k - z_k) f'\left(\boldsymbol{w}_k^T \cdot \boldsymbol{y}\right)$
8. $\quad\quad\quad\quad$ **for** $j = 1$ **to** n_H
9. $\quad\quad\quad\quad\quad net_j = \boldsymbol{w}_j^T \cdot \boldsymbol{x}; y_j = f\left(net_j\right)$
10. $\quad\quad\quad\quad\quad w_{kj}(m) = w_{kj}(m) + \eta \delta_k y_j(m)$
11. $\quad\quad\quad$ **for** $j = 1$ **to** n_H
12. $\quad\quad\quad\quad \delta_j = f'\left(net_j\right) \sum_{k=1}^{c} w_{kj} \delta_k$
13. $\quad\quad\quad\quad$ **for** $i = 1$ **to** d
14. $\quad\quad\quad\quad\quad w_{ji}(m) = w_{ji}(m) + \eta \delta_j x_i(m)$
15. **until** $\|\nabla_w J(m)\| < \Theta$
16. **return** $w(m)$

Final Basic Batch Algorithm

Perceptron$(\boldsymbol{X})$

1. **Initialize random** w, **number of hidden units** n_H, **number of outputs** z, **stopping criterion** Θ, **learning rate** η, **epoch**
   $m = 0$
2. **do**
3. $\quad m = m + 1$
4. $\quad\quad$ **for** $s = 1$ **to** N
5. $\quad\quad\quad \boldsymbol{x}(m) = \boldsymbol{X}(:, s)$
6. $\quad\quad\quad$ **for** $k = 1$ **to** c
7. $\quad\quad\quad\quad \delta_k = (t_k - z_k)\, f'\left(\boldsymbol{w}_k^T \cdot \boldsymbol{y}\right)$
8. $\quad\quad\quad\quad$ **for** $j = 1$ **to** n_H
9. $\quad\quad\quad\quad\quad net_j = \boldsymbol{w}_j^T \cdot \boldsymbol{x}; y_j = f\left(net_j\right)$
10. $\quad\quad\quad\quad\quad w_{kj}(m) = w_{kj}(m) + \eta \delta_k y_j(m)$
11. $\quad\quad\quad$ **for** $j = 1$ **to** n_H
12. $\quad\quad\quad\quad \delta_j = f'\left(net_j\right) \sum_{k=1}^{c} w_{kj} \delta_k$
13. $\quad\quad\quad\quad$ **for** $i = 1$ **to** d
14. $\quad\quad\quad\quad\quad w_{ji}(m) = w_{ji}(m) + \eta \delta_j x_i(m)$
15. **until** $\|\nabla_w J(m)\| < \Theta$
16. **return** $w(m)$

Final Basic Batch Algorithm

Perceptron$(\boldsymbol{X})$

1. **Initialize random** w, **number of hidden units** n_H, **number of outputs** $\boldsymbol{z}$, **stopping criterion** Θ, **learning rate**η, **epoch**

   $m = 0$

2. **do**
3. $\quad m = m + 1$
4. $\quad\quad$ **for** $s = 1$ **to** N
5. $\quad\quad\quad \boldsymbol{x}(m) = \boldsymbol{X}(:, s)$
6. $\quad\quad\quad$ **for** $k = 1$ **to** c
7. $\quad\quad\quad\quad \delta_k = (t_k - z_k) f'\left(\boldsymbol{w}_k^T \cdot \boldsymbol{y}\right)$
8. $\quad\quad\quad\quad$ **for** $j = 1$ **to** n_H
9. $\quad\quad\quad\quad\quad net_j = \boldsymbol{w}_j^T \cdot \boldsymbol{x}; y_j = f\left(net_j\right)$
10. $\quad\quad\quad\quad\quad w_{kj}(m) = w_{kj}(m) + \eta \delta_k y_j(m)$
11. $\quad\quad\quad$ **for** $j = 1$ **to** n_H
12. $\quad\quad\quad\quad \delta_j = f'\left(net_j\right) \sum_{k=1}^{c} w_{kj} \delta_k$
13. $\quad\quad\quad\quad$ **for** $i = 1$ **to** d
14. $\quad\quad\quad\quad\quad w_{ji}(m) = w_{ji}(m) + \eta \delta_j x_i(m)$
15. **until** $\|\nabla_w J(m)\| < \Theta$
16. **return** $w(m)$

Final Basic Batch Algorithm

Perceptron$(\boldsymbol{X})$

Initialize random w**, number of hidden units** n_H**, number of outputs** $\boldsymbol{z}$**, stopping criterion** Θ**, learning rate** η**, epoch**

$m = 0$

do

$m = m + 1$

for $s = 1$ **to** N

$\boldsymbol{x}(m) = \boldsymbol{X}(:, s)$

for $k = 1$ **to** c

$\delta_k = (t_k - z_k) f'\left(\boldsymbol{w}_k^T \cdot \boldsymbol{y}\right)$

for $j = 1$ **to** n_H

$net_j = \boldsymbol{w}_j^T \cdot \boldsymbol{x}; y_j = f\left(net_j\right)$

$w_{kj}(m) = w_{kj}(m) + \eta \delta_k y_j(m)$

for $j = 1$ **to** n_H

$\delta_j = f'\left(net_j\right) \sum_{k=1}^{c} w_{kj} \delta_k$

for $i = 1$ **to** d

$w_{ji}(m) = w_{ji}(m) + \eta \delta_j x_i(m)$

until $\|\nabla_{\boldsymbol{w}} J(m)\| < \Theta$

return $w(m)$

Final Basic Batch Algorithm

Perceptron$(\boldsymbol{X})$

1. **Initialize random** w**, number of hidden units** n_H**, number of outputs** $\boldsymbol{z}$**, stopping criterion** Θ**, learning rate**η**, epoch**
   $m = 0$
2. **do**
3. $\quad m = m + 1$
4. $\quad\quad$ **for** $s = 1$ **to** N
5. $\quad\quad\quad \boldsymbol{x}(m) = \boldsymbol{X}(:, s)$
6. $\quad\quad\quad$ **for** $k = 1$ **to** c
7. $\quad\quad\quad\quad \delta_k = (t_k - z_k) f'\left(\boldsymbol{w}_k^T \cdot \boldsymbol{y}\right)$
8. $\quad\quad\quad\quad$ **for** $j = 1$ **to** n_H
9. $\quad\quad\quad\quad\quad net_j = \boldsymbol{w}_j^T \cdot \boldsymbol{x}; y_j = f\left(net_j\right)$
10. $\quad\quad\quad\quad\quad w_{kj}(m) = w_{kj}(m) + \eta \delta_k y_j(m)$
11. $\quad\quad\quad$ **for** $j = 1$ **to** n_H
12. $\quad\quad\quad\quad \delta_j = f'\left(net_j\right) \sum_{k=1}^{c} w_{kj} \delta_k$
13. $\quad\quad\quad\quad$ **for** $i = 1$ **to** d
14. $\quad\quad\quad\quad\quad w_{ji}(m) = w_{ji}(m) + \eta \delta_j x_i(m)$
15. **until** $\|\nabla_{\boldsymbol{w}} J(m)\| < \Theta$
16. **return** $\boldsymbol{w}(m)$

Final Basic Batch Algorithm

Perceptron$(\boldsymbol{X})$

1. **Initialize random** w, **number of hidden units** n_H, **number of outputs** $\boldsymbol{z}$, **stopping criterion** Θ, **learning rate** η, **epoch**

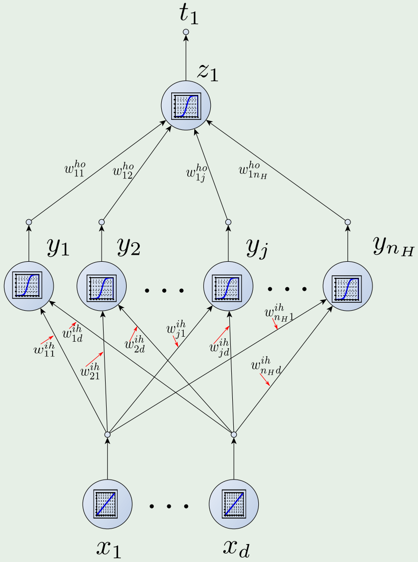
   $m = 0$
2. **do**
3. $\quad m = m + 1$
4. $\qquad$ **for** $s = 1$ **to** N
5. $\qquad\quad \boldsymbol{x}(m) = \boldsymbol{X}(:, s)$
6. $\qquad\quad$ **for** $k = 1$ **to** c
7. $\qquad\qquad \delta_k = (t_k - z_k)\, f'\left(\boldsymbol{w}_k^T \cdot \boldsymbol{y}\right)$
8. $\qquad\qquad$ **for** $j = 1$ **to** n_H
9. $\qquad\qquad\quad net_j = \boldsymbol{w}_j^T \cdot \boldsymbol{x}; y_j = f\left(net_j\right)$
10. $\qquad\qquad\quad w_{kj}(m) = w_{kj}(m) + \eta \delta_k y_j(m)$
11. $\qquad\quad$ **for** $j = 1$ **to** n_H
12. $\qquad\qquad \delta_j = f'\left(net_j\right) \sum_{k=1}^{c} w_{kj} \delta_k$
13. $\qquad\qquad$ **for** $i = 1$ **to** d
14. $\qquad\qquad\quad w_{ji}(m) = w_{ji}(m) + \eta \delta_j x_i(m)$
15. **until** $\|\nabla_{\boldsymbol{w}} J(m)\| < \Theta$
16. **return** $\boldsymbol{w}(m)$

Outline

Cinvestav

Example of Architecture to be used

Given the following Architecture and assuming N samples



Outline

Cinvestav

Generating the output z_k

Given the input

$$\boldsymbol{X} = \begin{bmatrix} \boldsymbol{x}_1 & \boldsymbol{x}_2 & \cdots & \boldsymbol{x}_N \end{bmatrix} \tag{24}$$

Where

x_i is a vector of features

$$x_i = \begin{pmatrix} x_{1i} \\ x_{2i} \\ \vdots \\ x_{di} \end{pmatrix} \tag{25}$$

Generating the output z_k

Given the input

$$\boldsymbol{X} = \begin{bmatrix} \boldsymbol{x}_1 & \boldsymbol{x}_2 & \cdots & \boldsymbol{x}_N \end{bmatrix} \tag{24}$$

Where

$\boldsymbol{x}_i$ is a vector of features

$$\boldsymbol{x}_i = \begin{pmatrix} x_{1i} \\ x_{2i} \\ \vdots \\ x_{di} \end{pmatrix} \tag{25}$$

Cinvestav

Therefore

We must have the following matrix for the input to hidden inputs

$$\boldsymbol{W}_{IH} = \begin{pmatrix} w_{11} & w_{12} & \cdots & w_{1d} \\ w_{21} & w_{22} & \cdots & w_{2d} \\ \vdots & \vdots & \ddots & \vdots \\ w_{n_H 1} & w_{n_H 2} & \cdots & w_{n_H d} \end{pmatrix} = \begin{pmatrix} \boldsymbol{w}_1^T \\ \boldsymbol{w}_2^T \\ \vdots \\ \boldsymbol{w}_{n_H}^T \end{pmatrix} \qquad (26)$$

Given that $\boldsymbol{w}_j = \begin{pmatrix} w_{j1} \\ w_{j2} \\ \vdots \\ w_{jd} \end{pmatrix}$

Thus

We can create the net_j for all the inputs by simply

$$net_j = \boldsymbol{W}_{IH}\boldsymbol{X} = \begin{pmatrix} \boldsymbol{w}_1^T\boldsymbol{x}_1 & \boldsymbol{w}_1^T\boldsymbol{x}_2 & \cdots & \boldsymbol{w}_1^T\boldsymbol{x}_N \\ \boldsymbol{w}_2^T\boldsymbol{x}_1 & \boldsymbol{w}_2^T\boldsymbol{x}_2 & \cdots & \boldsymbol{w}_2^T\boldsymbol{x}_N \\ \vdots & \vdots & \ddots & \vdots \\ \boldsymbol{w}_{n_H}^T\boldsymbol{x}_1 & \boldsymbol{w}_{n_H}^T\boldsymbol{x}_2 & \cdots & \boldsymbol{w}_{n_H}^T\boldsymbol{x}_N \end{pmatrix} \qquad (27)$$

Therefore

We must have the following matrix for the input to hidden inputs

$$
\boldsymbol{W}_{IH} = \begin{pmatrix} w_{11} & w_{12} & \cdots & w_{1d} \\ w_{21} & w_{22} & \cdots & w_{2d} \\ \vdots & \vdots & \ddots & \vdots \\ w_{n_H 1} & w_{n_H 2} & \cdots & w_{n_H d} \end{pmatrix} = \begin{pmatrix} \boldsymbol{w}_1^T \\ \boldsymbol{w}_2^T \\ \vdots \\ \boldsymbol{w}_{n_H}^T \end{pmatrix} \tag{26}
$$

Given that $\boldsymbol{w}_j = \begin{pmatrix} w_{j1} \\ w_{j2} \\ \vdots \\ w_{jd} \end{pmatrix}$

Thus

We can create the $\boldsymbol{net}_j$ for all the inputs by simply

$$
\boldsymbol{net}_j = \boldsymbol{W}_{IH}\boldsymbol{X} = \begin{pmatrix} \boldsymbol{w}_1^T\boldsymbol{x}_1 & \boldsymbol{w}_1^T\boldsymbol{x}_2 & \cdots & \boldsymbol{w}_1^T\boldsymbol{x}_N \\ \boldsymbol{w}_2^T\boldsymbol{x}_1 & \boldsymbol{w}_2^T\boldsymbol{x}_2 & \cdots & \boldsymbol{w}_2^T\boldsymbol{x}_N \\ \vdots & \vdots & \ddots & \vdots \\ \boldsymbol{w}_{n_H}^T\boldsymbol{x}_1 & \boldsymbol{w}_{n_H}^T\boldsymbol{x}_2 & \cdots & \boldsymbol{w}_{n_H}^T\boldsymbol{x}_N \end{pmatrix} \tag{27}
$$

Now, we need to generate the $\boldsymbol{y}_k$

We apply the activation function element by element in $\boldsymbol{net}_j$

$$\boldsymbol{y}_1 = \begin{pmatrix} f\left(\boldsymbol{w}_1^T \boldsymbol{x}_1\right) & f\left(\boldsymbol{w}_1^T \boldsymbol{x}_2\right) & \cdots & f\left(\boldsymbol{w}_1^T \boldsymbol{x}_N\right) \\ f\left(\boldsymbol{w}_2^T \boldsymbol{x}_1\right) & f\left(\boldsymbol{w}_2^T \boldsymbol{x}_2\right) & \cdots & f\left(\boldsymbol{w}_2^T \boldsymbol{x}_N\right) \\ \vdots & \vdots & \ddots & \vdots \\ f\left(\boldsymbol{w}_{n_H}^T \boldsymbol{x}_1\right) & f\left(\boldsymbol{w}_{n_H}^T \boldsymbol{x}_2\right) & \cdots & f\left(\boldsymbol{w}_{n_H}^T \boldsymbol{x}_N\right) \end{pmatrix} \quad (28)$$

Now, we need to generate the $\boldsymbol{y}_k$

We apply the activation function element by element in $\boldsymbol{net}_j$

$$\boldsymbol{y}_1 = \begin{pmatrix} f\left(\boldsymbol{w}_1^T\boldsymbol{x}_1\right) & f\left(\boldsymbol{w}_1^T\boldsymbol{x}_2\right) & \cdots & f\left(\boldsymbol{w}_1^T\boldsymbol{x}_N\right) \\ f\left(\boldsymbol{w}_2^T\boldsymbol{x}_1\right) & f\left(\boldsymbol{w}_2^T\boldsymbol{x}_2\right) & \cdots & f\left(\boldsymbol{w}_2^T\boldsymbol{x}_N\right) \\ \vdots & \vdots & \ddots & \vdots \\ f\left(\boldsymbol{w}_{n_H}^T\boldsymbol{x}_1\right) & f\left(\boldsymbol{w}_{n_H}^T\boldsymbol{x}_2\right) & \cdots & f\left(\boldsymbol{w}_{n_H}^T\boldsymbol{x}_N\right) \end{pmatrix} \quad (28)$$

IMPORTANT about overflows!!!

- Be careful about the numeric stability of the activation function.
- I the case of python, we can use the ones provided by scipy.special

Cinvestav

Now, we need to generate the $\boldsymbol{y}_k$

We apply the activation function element by element in $\boldsymbol{net}_j$

$$\boldsymbol{y}_1 = \begin{pmatrix} f\left(\boldsymbol{w}_1^T\boldsymbol{x}_1\right) & f\left(\boldsymbol{w}_1^T\boldsymbol{x}_2\right) & \cdots & f\left(\boldsymbol{w}_1^T\boldsymbol{x}_N\right) \\ f\left(\boldsymbol{w}_2^T\boldsymbol{x}_1\right) & f\left(\boldsymbol{w}_2^T\boldsymbol{x}_2\right) & \cdots & f\left(\boldsymbol{w}_2^T\boldsymbol{x}_N\right) \\ \vdots & \vdots & \ddots & \vdots \\ f\left(\boldsymbol{w}_{n_H}^T\boldsymbol{x}_1\right) & f\left(\boldsymbol{w}_{n_H}^T\boldsymbol{x}_2\right) & \cdots & f\left(\boldsymbol{w}_{n_H}^T\boldsymbol{x}_N\right) \end{pmatrix} \tag{28}$$

IMPORTANT about overflows!!!

- Be careful about the numeric stability of the activation function.
- I the case of python, we can use the ones provided by scipy.special

However, We can create a Sigmoid function

It is possible to use the following pseudo-code

Sigmoid(x)

```
if x < -BIGREAL
    return 0
else if x > BIGREAL
    return 1
else
    return 1.0 / (1.0 + exp{-αx})  ◁ 1.0 refers to the floating point (Rationals
                                    ◁ trying to represent Reals)
```

However, We can create a Sigmoid function

It is possible to use the following pseudo-code

Sigmoid(x)

if $x < -BIGREAL$

  return 0

else if $x > BIGREAL$

  return 1

else

  return $\frac{1.0}{1.0+\exp\{-\alpha x\}}$ ◁ 1.0 refers to the floating point (Rationals ◁ trying to represent Reals)

However, We can create a Sigmoid function

It is possible to use the following pseudo-code

Sigmoid(x)

```
if x < -BIGREAL
    return 0
else if x > BIGREAL
    return 1
else
    return 1.0 / (1.0 + exp{-αx})   ◁ 1.0 refers to the floating point (Rationals
                                     ◁ trying to represent Reals)
```

Outline

Cinvestav

For this, we get $\boldsymbol{net}_k$

For this, we obtain the $\boldsymbol{W}_{HO}$

$$\boldsymbol{W}_{HO} = \begin{pmatrix} w^o_{11} & w^o_{12} & \cdots & w^o_{1n_H} \end{pmatrix} = \begin{pmatrix} \boldsymbol{w}_o^T \end{pmatrix} \tag{29}$$

Thus

$$\boldsymbol{net}_k = \begin{pmatrix} w^o_{11} & w^o_{12} & \cdots & w^o_{1n_H} \end{pmatrix} \begin{pmatrix} f\left(\boldsymbol{w}_1^T \boldsymbol{x}_1\right) & f\left(\boldsymbol{w}_1^T \boldsymbol{x}_2\right) & \cdots & f\left(\boldsymbol{w}_1^T \boldsymbol{x}_N\right) \\ f\left(\boldsymbol{w}_2^T \boldsymbol{x}_1\right) & f\left(\boldsymbol{w}_2^T \boldsymbol{x}_2\right) & \cdots & f\left(\boldsymbol{w}_2^T \boldsymbol{x}_N\right) \\ \vdots & \vdots & \ddots & \vdots \\ \underbrace{f\left(\boldsymbol{w}_{n_H}^T \boldsymbol{x}_1\right)}_{\boldsymbol{y}_{k1}} & \underbrace{f\left(\boldsymbol{w}_{n_H}^T \boldsymbol{x}_2\right)}_{\boldsymbol{y}_{k2}} & \cdots & \underbrace{f\left(\boldsymbol{w}_{n_H}^T \boldsymbol{x}_N\right)}_{\boldsymbol{y}_{kN}} \end{pmatrix} \tag{30}$$

In matrix notation

$$\boldsymbol{net}_k = \begin{pmatrix} \boldsymbol{w}_o^T \boldsymbol{y}_{k1} & \boldsymbol{w}_o^T \boldsymbol{y}_{k2} & \cdots & \boldsymbol{w}_o^T \boldsymbol{y}_{kN} \end{pmatrix} \tag{31}$$

For this, we get $\boldsymbol{net}_k$

For this, we obtain the $\boldsymbol{W}_{HO}$

$$\boldsymbol{W}_{HO} = \begin{pmatrix} w_{11}^{o} & w_{12}^{o} & \cdots & w_{1n_H}^{o} \end{pmatrix} = \left(\boldsymbol{w}_o^T\right) \tag{29}$$

Thus

$$\boldsymbol{net}_k = \begin{pmatrix} w_{11}^{o} & w_{12}^{o} & \cdots & w_{1n_H}^{o} \end{pmatrix} \left(\begin{matrix} f\left(\boldsymbol{w}_1^T \boldsymbol{x}_1\right) & f\left(\boldsymbol{w}_1^T \boldsymbol{x}_2\right) & \cdots & f\left(\boldsymbol{w}_1^T \boldsymbol{x}_N\right) \\ f\left(\boldsymbol{w}_2^T \boldsymbol{x}_1\right) & f\left(\boldsymbol{w}_2^T \boldsymbol{x}_2\right) & \cdots & f\left(\boldsymbol{w}_2^T \boldsymbol{x}_N\right) \\ \vdots & \vdots & \ddots & \vdots \\ \underbrace{f\left(\boldsymbol{w}_{n_H}^T \boldsymbol{x}_1\right)}_{\boldsymbol{y}_{k1}} & \underbrace{f\left(\boldsymbol{w}_{n_H}^T \boldsymbol{x}_2\right)}_{\boldsymbol{y}_{k2}} & \cdots & \underbrace{f\left(\boldsymbol{w}_{n_H}^T \boldsymbol{x}_N\right)}_{\boldsymbol{y}_{kN}} \end{matrix} \right) \tag{30}$$

In matrix notation

$$\boldsymbol{net}_k = \begin{pmatrix} \boldsymbol{w}_o^T \boldsymbol{y}_{k1} & \boldsymbol{w}_o^T \boldsymbol{y}_{k2} & \cdots & \boldsymbol{w}_o^T \boldsymbol{y}_{kN} \end{pmatrix} \tag{31}$$

For this, we get $\boldsymbol{net}_k$

For this, we obtain the $\boldsymbol{W}_{HO}$

$$\boldsymbol{W}_{HO} = \begin{pmatrix} w_{11}^{o} & w_{12}^{o} & \cdots & w_{1n_H}^{o} \end{pmatrix} = \begin{pmatrix} \boldsymbol{w}_o^T \end{pmatrix} \quad (29)$$

Thus

$$\boldsymbol{net}_k = \begin{pmatrix} w_{11}^{o} & w_{12}^{o} & \cdots & w_{1n_H}^{o} \end{pmatrix} \begin{pmatrix} \underbrace{\begin{matrix} f\left(\boldsymbol{w}_1^T\boldsymbol{x}_1\right) \\ f\left(\boldsymbol{w}_2^T\boldsymbol{x}_1\right) \\ \vdots \\ f\left(\boldsymbol{w}_{n_H}^T\boldsymbol{x}_1\right) \end{matrix}}_{\boldsymbol{y}_{k1}} & \underbrace{\begin{matrix} f\left(\boldsymbol{w}_1^T\boldsymbol{x}_2\right) \\ f\left(\boldsymbol{w}_2^T\boldsymbol{x}_2\right) \\ \vdots \\ f\left(\boldsymbol{w}_{n_H}^T\boldsymbol{x}_2\right) \end{matrix}}_{\boldsymbol{y}_{k2}} & \begin{matrix} \cdots \\ \cdots \\ \ddots \\ \cdots \end{matrix} & \underbrace{\begin{matrix} f\left(\boldsymbol{w}_1^T\boldsymbol{x}_N\right) \\ f\left(\boldsymbol{w}_2^T\boldsymbol{x}_N\right) \\ \vdots \\ f\left(\boldsymbol{w}_{n_H}^T\boldsymbol{x}_N\right) \end{matrix}}_{\boldsymbol{y}_{kN}} \end{pmatrix} \quad (30)$$

In matrix notation

$$\boldsymbol{net}_k = \begin{pmatrix} \boldsymbol{w}_o^T\boldsymbol{y}_{k1} & \boldsymbol{w}_o^T\boldsymbol{y}_{k2} & \cdots & \boldsymbol{w}_o^T\boldsymbol{y}_{kN} \end{pmatrix} \quad (31)$$

Outline

Cinvestav

Now, we have

Thus, we have $\boldsymbol{z}_k$ (In our case $k = 1$, but it could be a range of values)

$$\boldsymbol{z}_k = \begin{pmatrix} f\left(\boldsymbol{w}_o^T \boldsymbol{y}_{k1}\right) & f\left(\boldsymbol{w}_o^T \boldsymbol{y}_{k2}\right) & \cdots & f\left(\boldsymbol{w}_o^T \boldsymbol{y}_{kN}\right) \end{pmatrix} \quad (32)$$

Thus, we generate a vector of differences

$$\boldsymbol{d} = \boldsymbol{t} - \boldsymbol{z}_k = \begin{pmatrix} t_1 - f\left(\boldsymbol{w}_o^T \boldsymbol{y}_{k1}\right) & t_2 - f\left(\boldsymbol{w}_o^T \boldsymbol{y}_{k2}\right) & \cdots & t_N - f\left(\boldsymbol{w}_o^T \boldsymbol{y}_{kN}\right) \end{pmatrix} \quad (33)$$

where $\boldsymbol{t} = \begin{pmatrix} t_1 & t_2 & \cdots & t_N \end{pmatrix}$ is a row vector of desired outputs for each sample.

Cinvestav

Now, we have

Thus, we have $\boldsymbol{z}_k$ (In our case $k=1$, but it could be a range of values)

$$\boldsymbol{z}_k = \begin{pmatrix} f\left(\boldsymbol{w}_o^T \boldsymbol{y}_{k1}\right) & f\left(\boldsymbol{w}_o^T \boldsymbol{y}_{k2}\right) & \cdots & f\left(\boldsymbol{w}_o^T \boldsymbol{y}_{kN}\right) \end{pmatrix} \tag{32}$$

Thus, we generate a vector of differences

$$\boldsymbol{d} = \boldsymbol{t} - \boldsymbol{z}_k = \begin{pmatrix} t_1 - f\left(\boldsymbol{w}_o^T \boldsymbol{y}_{k1}\right) & t_2 - f\left(\boldsymbol{w}_o^T \boldsymbol{y}_{k2}\right) & \cdots & t_N - f\left(\boldsymbol{w}_o^T \boldsymbol{y}_{kN}\right) \end{pmatrix} \tag{33}$$

where $\boldsymbol{t} = \begin{pmatrix} t_1 & t_2 & \cdots & t_N \end{pmatrix}$ is a row vector of desired outputs for each sample.

Cinvestav

Now, we multiply element wise

We have the following vector of derivatives of net

$$\boldsymbol{D}_f = \begin{pmatrix} \eta f'\left(\boldsymbol{w}_o^T \boldsymbol{y}_{k1}\right) & \eta f'\left(\boldsymbol{w}_o^T \boldsymbol{y}_{k2}\right) & \cdots & \eta f'\left(\boldsymbol{w}_o^T \boldsymbol{y}_{kN}\right) \end{pmatrix} \quad (34)$$

where η is the step rate.

Finally, by element wise multiplication (Hadamard Product)

$$\boldsymbol{d} = \big(\eta\left[t_1 - f\left(\boldsymbol{w}_o^T \boldsymbol{y}_{k1}\right)\right] f'\left(\boldsymbol{w}_o^T \boldsymbol{y}_{k1}\right) \quad \eta\left[t_2 - f\left(\boldsymbol{w}_o^T \boldsymbol{y}_{k2}\right)\right] f'\left(\boldsymbol{w}_o^T \boldsymbol{y}_{k2}\right) \quad \cdots \quad \eta\left[t_N - f\left(\boldsymbol{w}_o^T \boldsymbol{y}_{kN}\right)\right] f'\left(\boldsymbol{w}_o^T \boldsymbol{y}_{kN}\right) \big)$$

Now, we multiply element wise

We have the following vector of derivatives of net

$$\boldsymbol{D}_f = \begin{pmatrix} \eta f'\left(\boldsymbol{w}_o^T \boldsymbol{y}_{k1}\right) & \eta f'\left(\boldsymbol{w}_o^T \boldsymbol{y}_{k2}\right) & \cdots & \eta f'\left(\boldsymbol{w}_o^T \boldsymbol{y}_{kN}\right) \end{pmatrix} \quad (34)$$

where η is the step rate.

Finally, by element wise multiplication (Hadamard Product)

$$\boldsymbol{d} = \Big(\eta\left[t_1 - f\left(\boldsymbol{w}_o^T \boldsymbol{y}_{k1}\right)\right] f'\left(\boldsymbol{w}_o^T \boldsymbol{y}_{k1}\right) \quad \eta\left[t_2 - f\left(\boldsymbol{w}_o^T \boldsymbol{y}_{k2}\right)\right] f'\left(\boldsymbol{w}_o^T \boldsymbol{y}_{k2}\right) \quad \cdots \\ \eta\left[t_N - f\left(\boldsymbol{w}_o^T \boldsymbol{y}_{kN}\right)\right] f'\left(\boldsymbol{w}_o^T \boldsymbol{y}_{kN}\right)\Big)$$

Cinvestav

Tile $\boldsymbol{d}$

Tile downward

$$\boldsymbol{d}_{tile} = n_H \text{ rows } \left\{ \begin{pmatrix} \boldsymbol{d} \\ \boldsymbol{d} \\ \vdots \\ \boldsymbol{d} \end{pmatrix} \right. \tag{35}$$

Finally, we multiply element wise against $\boldsymbol{y}_1$ (Hadamard Product)

$$\Delta w_{1j}^{temp} = \boldsymbol{y}_1 \circ \boldsymbol{d}_{tile} \tag{36}$$

Tile $\boldsymbol{d}$

Tile downward

$$\boldsymbol{d}_{tile} = n_H \text{ rows } \left\{ \begin{pmatrix} \boldsymbol{d} \\ \boldsymbol{d} \\ \vdots \\ \boldsymbol{d} \end{pmatrix} \right. \tag{35}$$

Finally, we multiply element wise against $\boldsymbol{y}_1$ (Hadamard Product)

$$\Delta \boldsymbol{w}_{1j}^{temp} = \boldsymbol{y}_1 \circ \boldsymbol{d}_{tile} \tag{36}$$

We obtain the total $\Delta \boldsymbol{w}_{1j}$

We sum along the rows of $\Delta \boldsymbol{w}_{1j}^{temp}$

$$\Delta \boldsymbol{w}_{1j} = \begin{pmatrix} \eta \left[t_1 - f\left(\boldsymbol{w}_o^T \boldsymbol{y}_{k1} \right) \right] f'\left(\boldsymbol{w}_o^T \boldsymbol{y}_{k1} \right) y_{11} + \eta \left[t_1 - f\left(\boldsymbol{w}_o^T \boldsymbol{y}_{kN} \right) \right] f'\left(\boldsymbol{w}_o^T \boldsymbol{y}_{kN} \right) y_{1N} \\ \vdots \\ \eta \left[t_1 - f\left(\boldsymbol{w}_o^T \boldsymbol{y}_{k1} \right) \right] f'\left(\boldsymbol{w}_o^T \boldsymbol{y}_{k1} \right) y_{n_H 1} + \eta \left[t_1 - f\left(\boldsymbol{w}_o^T \boldsymbol{y}_{kN} \right) \right] f'\left(\boldsymbol{w}_o^T \boldsymbol{y}_{kN} \right) y_{n_H N} \end{pmatrix} \tag{37}$$

where $y_{hm} = f\left(\boldsymbol{w}_h^T \boldsymbol{x}_m \right)$ with $h = 1, 2, ..., n_H$ and $m = 1, 2, ..., N$.

Finally, we update the first weights

We have then

$$\boldsymbol{W}_{HO}\left(t+1\right)=\boldsymbol{W}_{HO}\left(t\right)+\Delta\boldsymbol{w}_{1j}^{T}\left(t\right) \tag{38}$$

Cinvestav

Outline

Cinvestav

First

We multiply element wise the $\boldsymbol{W}_{HO}$ and $\Delta \boldsymbol{w}_{1j}$

$$\boldsymbol{T} = \Delta \boldsymbol{w}_{1j}^{T} \circ \boldsymbol{W}_{HO}^{T} \tag{39}$$

Now, we obtain the element wise derivative of net_j

$$\boldsymbol{Dnet}_j = \begin{pmatrix} f'\left(\boldsymbol{w}_1^T \boldsymbol{x}_1\right) & f'\left(\boldsymbol{w}_1^T \boldsymbol{x}_2\right) & \cdots & f'\left(\boldsymbol{w}_1^T \boldsymbol{x}_N\right) \\ f'\left(\boldsymbol{w}_2^T \boldsymbol{x}_1\right) & f'\left(\boldsymbol{w}_2^T \boldsymbol{x}_2\right) & \cdots & f'\left(\boldsymbol{w}_2^T \boldsymbol{x}_N\right) \\ \vdots & \vdots & \ddots & \vdots \\ f'\left(\boldsymbol{w}_{n_H}^T \boldsymbol{x}_1\right) & f'\left(\boldsymbol{w}_{n_H}^T \boldsymbol{x}_2\right) & \cdots & f'\left(\boldsymbol{w}_{n_H}^T \boldsymbol{x}_N\right) \end{pmatrix} \tag{40}$$

First

We multiply element wise the $\boldsymbol{W}_{HO}$ and $\Delta \boldsymbol{w}_{1j}$

$$\boldsymbol{T} = \Delta \boldsymbol{w}_{1j}^T \circ \boldsymbol{W}_{HO}^T \quad (39)$$

Now, we obtain the element wise derivative of $\boldsymbol{net}_j$

$$\boldsymbol{Dnet}_j = \begin{pmatrix} f'\left(\boldsymbol{w}_1^T \boldsymbol{x}_1\right) & f'\left(\boldsymbol{w}_1^T \boldsymbol{x}_2\right) & \cdots & f'\left(\boldsymbol{w}_1^T \boldsymbol{x}_N\right) \\ f'\left(\boldsymbol{w}_2^T \boldsymbol{x}_1\right) & f'\left(\boldsymbol{w}_2^T \boldsymbol{x}_2\right) & \cdots & f'\left(\boldsymbol{w}_2^T \boldsymbol{x}_N\right) \\ \vdots & \vdots & \ddots & \vdots \\ f'\left(\boldsymbol{w}_{n_H}^T \boldsymbol{x}_1\right) & f'\left(\boldsymbol{w}_{n_H}^T \boldsymbol{x}_2\right) & \cdots & f'\left(\boldsymbol{w}_{n_H}^T \boldsymbol{x}_N\right) \end{pmatrix} \quad (40)$$

Thus

We tile to the right T

$$\boldsymbol{T}_{tile} = \underbrace{\begin{pmatrix} \boldsymbol{T} & \boldsymbol{T} & \cdots & \boldsymbol{T} \end{pmatrix}}_{N \text{ Columns}} \tag{41}$$

Now, we multiply element wise together with η

$$\boldsymbol{P}_t = \eta \left(\boldsymbol{Dnet}_j \circ \boldsymbol{T}_{tile} \right) \tag{42}$$

where η is constant multiplied against the result the Hadamar Product (Result a $n_H \times N$ matrix)

Cinvestav

Thus

We tile to the right T

$$\boldsymbol{T}_{tile} = \underbrace{\begin{pmatrix} \boldsymbol{T} & \boldsymbol{T} & \cdots & \boldsymbol{T} \end{pmatrix}}_{N \text{ Columns}} \tag{41}$$

Now, we multiply element wise together with η

$$\boldsymbol{P}_t = \eta \left(\boldsymbol{Dnet}_j \circ \boldsymbol{T}_{tile} \right) \tag{42}$$

where η is constant multiplied against the result the Hadamar Product (Result a $n_H \times N$ matrix)

Finally

We get use the transpose of $\boldsymbol{X}$ which is a $N \times d$ matrix

$$\boldsymbol{X}^T = \begin{pmatrix} \boldsymbol{x}_1^T \\ \boldsymbol{x}_2^T \\ \vdots \\ \boldsymbol{x}_N^T \end{pmatrix} \quad (43)$$

Finally, we get a $n_H \times d$ matrix

$$\Delta w_{ij} = \boldsymbol{P}_i \boldsymbol{X}^T \quad (44)$$

Thus, given $\boldsymbol{W}_{IH}$

$$\boldsymbol{W}_{IH}(t+1) = \boldsymbol{W}_{HO}(t) + \Delta w_{ij}^T(t) \quad (45)$$

Cinvestav

Finally

We get use the transpose of $\boldsymbol{X}$ which is a $N \times d$ matrix

$$\boldsymbol{X}^T = \begin{pmatrix} \boldsymbol{x}_1^T \\ \boldsymbol{x}_2^T \\ \vdots \\ \boldsymbol{x}_N^T \end{pmatrix} \tag{43}$$

Finally, we get a $n_H \times d$ matrix

$$\Delta \boldsymbol{w}_{ij} = \boldsymbol{P}_t \boldsymbol{X}^T \tag{44}$$

Thus, given $\boldsymbol{W}_{IH}$

$$\boldsymbol{W}_{IH}(t+1) = \boldsymbol{W}_{HO}(t) + \Delta w_{ij}^T(t) \tag{45}$$

Cinvestav

Finally

We get use the transpose of $\boldsymbol{X}$ which is a $N \times d$ matrix

$$\boldsymbol{X}^T = \begin{pmatrix} \boldsymbol{x}_1^T \\ \boldsymbol{x}_2^T \\ \vdots \\ \boldsymbol{x}_N^T \end{pmatrix} \quad (43)$$

Finally, we get a $n_H \times d$ matrix

$$\Delta \boldsymbol{w}_{ij} = \boldsymbol{P}_t \boldsymbol{X}^T \quad (44)$$

Thus, given $\boldsymbol{W}_{IH}$

$$\boldsymbol{W}_{IH}(t+1) = \boldsymbol{W}_{HO}(t) + \Delta \boldsymbol{w}_{ij}^T(t) \quad (45)$$

Outline

Cinvestav

We have different activation functions

The two most important

1. Sigmoid function.
2. Hyperbolic tangent function

We have different activation functions

The two most important

1. Sigmoid function.
2. Hyperbolic tangent function

Logistic Function

This non-linear function has the following definition for a neuron j

$$f_j\left(v_j\left(n\right)\right) = \frac{1}{1 + \exp\left\{-a v_j\left(n\right)\right\}} \; a > 0 \text{ and } -\infty < v_j\left(n\right) < \infty \tag{46}$$

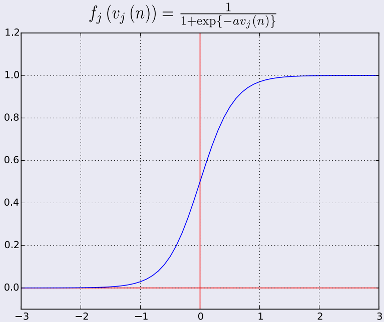
Example

Logistic Function

This non-linear function has the following definition for a neuron j

$$f_j\left(v_j\left(n\right)\right) = \frac{1}{1+\exp\left\{-av_j\left(n\right)\right\}} \; a > 0 \text{ and } -\infty < v_j\left(n\right) < \infty \quad (46)$$

Example



The differential of the sigmoid function

Now if we differentiate, we have

$$f_j'\left(v_j\left(n\right)\right) = \left[\frac{1}{1+\exp\left\{-av_j\left(n\right)\right\}}\right]\left[1-\frac{1}{1+\exp\left\{-av_j\left(n\right)\right\}}\right]$$

$$= \frac{\exp\left\{-av_j\left(n\right)\right\}}{\left(1+\exp\left\{-av_j\left(n\right)\right\}\right)^2}$$

The differential of the sigmoid function

Now if we differentiate, we have

$$
\begin{aligned}
f_j'\left(v_j\left(n\right)\right) &= \left[\frac{1}{1+\exp\left\{-av_j\left(n\right)\right\}}\right]\left[1-\frac{1}{1+\exp\left\{-av_j\left(n\right)\right\}}\right]\\
&= \frac{\exp\left\{-av_j\left(n\right)\right\}}{\left(1+\exp\left\{-av_j\left(n\right)\right\}\right)^2}
\end{aligned}
$$

Cinvestav

The outputs finish as

For the output neurons

$$
\begin{aligned}
\delta_k &= (t_k - z_k)\, f'(net_k) \\
&= (t_k - f_k(v_k(n)))\, f_k(v_k(n))\,(1 - f_k(v_k(n)))
\end{aligned}
$$

Cinvestav

The outputs finish as

For the output neurons

$$\begin{aligned}\delta_k &= (t_k - z_k)\, f'(net_k) \\ &= (t_k - f_k(v_k(n)))\, f_k(v_k(n))\,(1 - f_k(v_k(n)))\end{aligned}$$

For the hidden neurons

$$\delta_j = f_j(v_j(n))\,(1 - f_j(v_j(n))) \sum_{k=1}^{c} w_{kj}\delta_k$$

The outputs finish as

For the output neurons

$$\begin{aligned}\delta_k &= (t_k - z_k)\, f'(net_k)\\ &= (t_k - f_k(v_k(n)))\, f_k(v_k(n))\,(1 - f_k(v_k(n)))\end{aligned}$$

For the hidden neurons

$$\delta_j = f_j(v_j(n))\,(1 - f_j(v_j(n)))\sum_{k=1}^{c} w_{kj}\delta_k$$

Hyperbolic tangent function

Another commonly used form of sigmoidal non linearity is the hyperbolic tangent function

$$f_j\left(v_j\left(n\right)\right) = a\tanh\left(bv_j\left(n\right)\right) \quad (47)$$

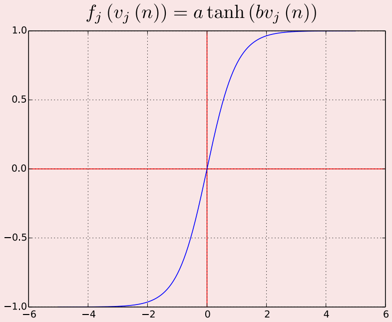
Example

Hyperbolic tangent function

Another commonly used form of sigmoidal non linearity is the hyperbolic tangent function

$$f_j\left(v_j\left(n\right)\right) = a \tanh\left(b v_j\left(n\right)\right) \quad (47)$$

Example



The differential of the hyperbolic tangent

We have

$$
\begin{aligned}
f_j\left(v_j\left(n\right)\right) =& ab\,\mathsf{sech}^2\left(bv_j\left(n\right)\right)\\
=& ab\left(1-\tanh^2\left(bv_j\left(n\right)\right)\right)
\end{aligned}
$$

The differential of the hyperbolic tangent

We have

$$
\begin{aligned}
f_j\left(v_j\left(n\right)\right) =& ab\,\mathrm{sech}^2\left(bv_j\left(n\right)\right)\\
=& ab\left(1-\tanh^2\left(bv_j\left(n\right)\right)\right)
\end{aligned}
$$

BTW

I leave to you to figure out the outputs.

The differential of the hyperbolic tangent

We have

$$
\begin{aligned}
f_j\left(v_j\left(n\right)\right) =& ab\textsf{sech}^2\left(bv_j\left(n\right)\right)\\
=& ab\left(1-\tanh^2\left(bv_j\left(n\right)\right)\right)
\end{aligned}
$$

BTW

I leave to you to figure out the outputs.

Outline

Maximizing information content

Two ways of achieving this, LeCun 1993

- The use of an example that results in the largest training error.
- The use of an example that is radically different from all those previously used.

Maximizing information content

Two ways of achieving this, LeCun 1993

- The use of an example that results in the largest training error.
- The use of an example that is radically different from all those previously used.

For this

Randomized the samples presented to the multilayer perceptron when not doing batch training.

- Use them to train the neural network

Maximizing information content

Two ways of achieving this, LeCun 1993

- The use of an example that results in the largest training error.
- The use of an example that is radically different from all those previously used.

For this

Randomized the samples presented to the multilayer perceptron when not doing batch training.

Or use an emphasizing scheme

By using the error identify the difficult vs. easy patterns:

- Use them to train the neural network

Maximizing information content

Two ways of achieving this, LeCun 1993

- The use of an example that results in the largest training error.
- The use of an example that is radically different from all those previously used.

For this

Randomized the samples presented to the multilayer perceptron when not doing batch training.

Or use an emphasizing scheme

By using the error identify the difficult vs. easy patterns:

- Use them to train the neural network

Maximizing information content

Two ways of achieving this, LeCun 1993

- The use of an example that results in the largest training error.
- The use of an example that is radically different from all those previously used.

For this

Randomized the samples presented to the multilayer perceptron when not doing batch training.

Or use an emphasizing scheme

By using the error identify the difficult vs. easy patterns:

- Use them to train the neural network

Cinvestav

However!!!

Be careful about emphasizing scheme

- The distribution of examples within an epoch presented to the network is distorted.
- The presence of an outlier or a mislabeled example can have a catastrophic consequence on the performance of the algorithm.

However!!!

Be careful about emphasizing scheme

- The distribution of examples within an epoch presented to the network is distorted.
- The presence of an outlier or a mislabeled example can have a catastrophic consequence on the performance of the algorithm.

Definition of Outlier

An outlier is an observation that lies outside the overall pattern of a distribution (Moore and McCabe 1999).

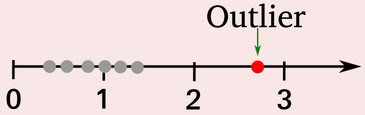
However!!!

Be careful about emphasizing scheme

- The distribution of examples within an epoch presented to the network is distorted.
- The presence of an outlier or a mislabeled example can have a catastrophic consequence on the performance of the algorithm.

Definition of Outlier

An outlier is an observation that lies outside the overall pattern of a distribution (Moore and McCabe 1999).



Outline

Cinvestav

Activation Function

We say that

An activation function $f(v)$ is antisymmetric if $f(-v) = -f(v)$

It seems to be

That the multilayer perceptron learns faster using an antisymmetric function.

Example: The hyperbolic tangent

Activation Function

We say that

An activation function $f\left(v\right)$ is antisymmetric if $f\left(-v\right)=-f\left(v\right)$

It seems to be

That the multilayer perceptron learns faster using an antisymmetric function.

Example: The hyperbolic tangent

Activation Function

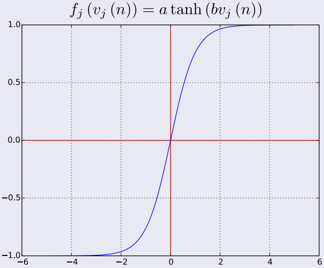
We say that

An activation function $f(v)$ is antisymmetric if $f(-v) = -f(v)$

It seems to be

That the multilayer perceptron learns faster using an antisymmetric function.

Example: The hyperbolic tangent



Outline

Cinvestav

Target Values

Important

- It is important that the target values be chosen within the range of the sigmoid activation function.

Specifically

- The desired response for neuron in the output layer of the multilayer perceptron should be offset by some amount ϵ

Target Values

Important

- It is important that the target values be chosen within the range of the sigmoid activation function.

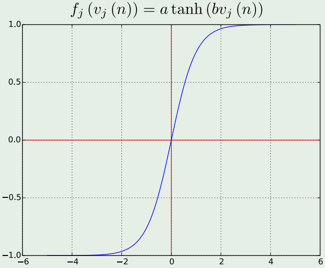
Specifically

- The desired response for neuron in the output layer of the multilayer perceptron should be offset by some amount ϵ

Cinvestav

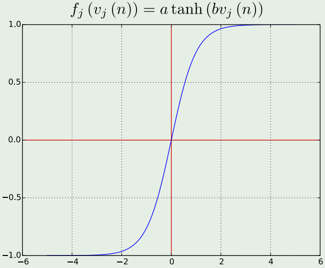
For example

Given the a limiting value



For example

Given the a limiting value

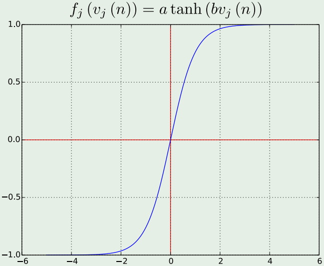


We have then

- If we have a limiting value $+a$, we set $t = a - \epsilon$.
- If we have a limiting value $-a$, we set $t = -a + \epsilon$.

For example

Given the a limiting value



We have then

- If we have a limiting value $+a$, we set $t = a - \epsilon$.
- If we have a limiting value $-a$, we set $t = -a + \epsilon$.

Cinvestav

Outline

Cinvestav

Normalizing the inputs

Something Important (LeCun, 1993)

Each input variable should be preprocessed so that:

- The mean value, averaged over the entire training set, is close to zero.
- Or it is smalll compared to its standard deviation.

Normalizing the inputs

Something Important (LeCun, 1993)

Each input variable should be preprocessed so that:

- The mean value, averaged over the entire training set, is close to zero.
- Or it is smalll compared to its standard deviation.

Example

Normalizing the inputs

Something Important (LeCun, 1993)

Each input variable should be preprocessed so that:

- The mean value, averaged over the entire training set, is close to zero.
- Or it is smalll compared to its standard deviation.

Example

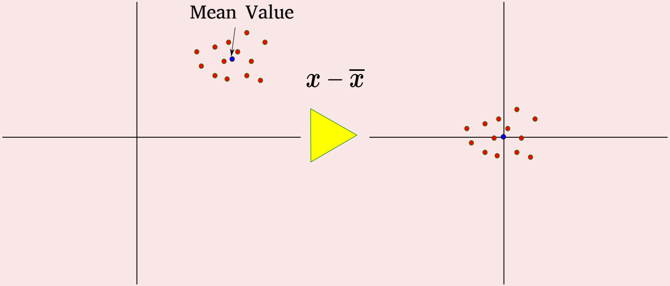
Normalizing the inputs

Something Important (LeCun, 1993)

Each input variable should be preprocessed so that:

- The mean value, averaged over the entire training set, is close to zero.
- Or it is smalll compared to its standard deviation.

Example



The normalization must include two other measures

Uncorrelated

We can use the principal component analysis

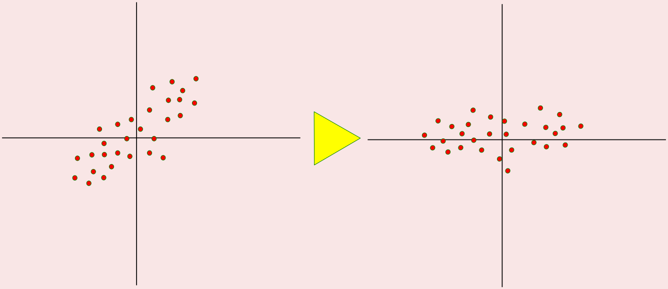
Example

The normalization must include two other measures

Uncorrelated

We can use the principal component analysis

Example



In addition

Quite interesting

- The decorrelated input variables should be scaled so that their covariances are approximately equal.

Why?

- This makes that different synaptic weights in network to learn at approximately the same speed.

In addition

Quite interesting

- The decorrelated input variables should be scaled so that their covariances are approximately equal.

Why?

- This makes that different synaptic weights in network to learn at approximately the same speed.

There are other heuristics

As

- Initialization
- Learning form hints
- Learning rates
- etc

There are other heuristics

As

- Initialization
- Learning form hints
- Learning rates
- etc

There are other heuristics

As

- Initialization
- Learning form hints
- Learning rates
- etc

There are other heuristics

As

- Initialization
- Learning form hints
- Learning rates
- etc

In addition

In section 4.15, Simon Haykin

We have the following techniques:

- Network growing
  - You start with a small network and add neurons and layers to accomplish the learning task.
- Network pruning
  - Start with a large network, then prune weights that are not necessary in an orderly fashion.

Cinvestav

In addition

In section 4.15, Simon Haykin

We have the following techniques:

- Network growing
  - You start with a small network and add neurons and layers to accomplish the learning task.
- Network pruning
  - Start with a large network, then prune weights that are not necessary in an orderly fashion.

Cinvestav

In addition

In section 4.15, Simon Haykin

We have the following techniques:

- Network growing
  - You start with a small network and add neurons and layers to accomplish the learning task.
- Network pruning
  - Start with a large network, then prune weights that are not necessary in an orderly fashion.

Outline

Cinvestav

Virtues and limitations of Back-Propagation Layer

Something Notable

- The back-propagation algorithm has emerged as the most popular algorithm for the training of multilayer perceptrons.

Virtues and limitations of Back-Propagation Layer

Something Notable

- The back-propagation algorithm has emerged as the most popular algorithm for the training of multilayer perceptrons.

It has two distinct properties

- It is simple to compute locally.
- It performs stochastic gradient descent in weight space when doing pattern-by-pattern training

Virtues and limitations of Back-Propagation Layer

Something Notable

- The back-propagation algorithm has emerged as the most popular algorithm for the training of multilayer perceptrons.

It has two distinct properties

- It is simple to compute locally.
- It performs stochastic gradient descent in weight space when doing pattern-by-pattern training

Connectionism

Back-propagation

- It is an example of a connectionist paradigm that relies on local computations to discover the processing capabilities of neural networks.

This constraint

It is known as the locality constraint

Connectionism

Back-propagation

- It is an example of a connectionist paradigm that relies on local computations to discover the processing capabilities of neural networks.

This constraint

It is known as the locality constraint

Why this is advocated in Artificial Neural Networks

First

Artificial neural networks that perform local computations are often held up as metaphors for biological neural networks.

Second

The use of local computations permits a graceful degradation in performance due to hardware errors, and therefore provides the basis for a fault-tolerant network design.

Third

Local computations favor the use of parallel architectures as an efficient method for the implementation of artificial neural networks.

Cinvestav

Why this is advocated in Artificial Neural Networks

First

Artificial neural networks that perform local computations are often held up as metaphors for biological neural networks.

Second

The use of local computations permits a graceful degradation in performance due to hardware errors, and therefore provides the basis for a fault-tolerant network design.

Third

Local computations favor the use of parallel architectures as an efficient method for the implementation of artificial neural networks.

Cinvestav

Why this is advocated in Artificial Neural Networks

First

Artificial neural networks that perform local computations are often held up as metaphors for biological neural networks.

Second

The use of local computations permits a graceful degradation in performance due to hardware errors, and therefore provides the basis for a fault-tolerant network design.

Third

Local computations favor the use of parallel architectures as an efficient method for the implementation of artificial neural networks.

However, all this has been seriously questioned on the following grounds(Shepherd, 1990b; Crick, 1989; Stork, 1989)

First

- The reciprocal synaptic connections between the neurons of a multilayer perceptron may assume weights that are excitatory or inhibitory.
- In the real nervous system, neurons usually appear to be the one or the other.

However, all this has been seriously questioned on the following grounds(Shepherd, 1990b; Crick, 1989; Stork, 1989)

First

- The reciprocal synaptic connections between the neurons of a multilayer perceptron may assume weights that are excitatory or inhibitory.
- In the real nervous system, neurons usually appear to be the one or the other.

Second

In a multilayer perceptron, hormonal and other types of global communications are ignored.

However, all this has been seriously questioned on the following grounds(Shepherd, 1990b; Crick, 1989; Stork, 1989)

First

- The reciprocal synaptic connections between the neurons of a multilayer perceptron may assume weights that are excitatory or inhibitory.
- In the real nervous system, neurons usually appear to be the one or the other.

Second

In a multilayer perceptron, hormonal and other types of global communications are ignored.

However, all this has been seriously questioned on the following grounds(Shepherd, 1990b; Crick, 1989; Stork, 1989)

Third

- In back-propagation learning, a synaptic weight is modified by a presynaptic activity and an error (learning) signal independent of postsynaptic activity.
- There is evidence from neurobiology to suggest otherwise.

- It appears highly unlikely that such an operation actually takes place in the brain.

However, all this has been seriously questioned on the following grounds(Shepherd, 1990b; Crick, 1989; Stork, 1989)

Third

- In back-propagation learning, a synaptic weight is modified by a presynaptic activity and an error (learning) signal independent of postsynaptic activity.
- There is evidence from neurobiology to suggest otherwise.

Fourth

- In a neurobiological sense, the implementation of back-propagation learning requires the rapid transmission of information backward along an axon.
- It appears highly unlikely that such an operation actually takes place in the brain.

However, all this has been seriously questioned on the following grounds(Shepherd, 1990b; Crick, 1989; Stork, 1989)

Third

- In back-propagation learning, a synaptic weight is modified by a presynaptic activity and an error (learning) signal independent of postsynaptic activity.
- There is evidence from neurobiology to suggest otherwise.

Fourth

- In a neurobiological sense, the implementation of back-propagation learning requires the rapid transmission of information backward along an axon.
- It appears highly unlikely that such an operation actually takes place in the brain.

However, all this has been seriously questioned on the following grounds(Shepherd, 1990b; Crick, 1989; Stork, 1989)

Third

- In back-propagation learning, a synaptic weight is modified by a presynaptic activity and an error (learning) signal independent of postsynaptic activity.
- There is evidence from neurobiology to suggest otherwise.

Fourth

- In a neurobiological sense, the implementation of back-propagation learning requires the rapid transmission of information backward along an axon.
- It appears highly unlikely that such an operation actually takes place in the brain.

However, all this has been seriously questioned on the following grounds(Shepherd, 1990b; Crick, 1989; Stork, 1989)

Fifth

- Back-propagation learning implies the existence of a "teacher," which in the con text of the brain would presumably be another set of neurons with novel properties.
- The existence of such neurons is biologically implausible.

However, all this has been seriously questioned on the following grounds(Shepherd, 1990b; Crick, 1989; Stork, 1989)

Fifth

- Back-propagation learning implies the existence of a "teacher," which in the con text of the brain would presumably be another set of neurons with novel properties.
- The existence of such neurons is biologically implausible.

Computational Efficiency

Something Notable

The computational complexity of an algorithm is usually measured in terms of the number of multiplications, additions, and storage involved in its implementation.

- This is the electrical engineering approach!!!

Taking in account the total number of synapses, W including biases

We have $\triangle w_{kj} = \eta \delta_k y_j = \eta (t_k - z_k) f'(net_k) y_j$ (Backward Pass)

We have that for this step

1. We need to calculate net_k linear in the number of weights.
2. We need to calculate $y_j = f(net_j)$ which is linear in the number of weights.

Computational Efficiency

Something Notable

The computational complexity of an algorithm is usually measured in terms of the number of multiplications, additions, and storage involved in its implementation.

- This is the electrical engineering approach!!!

Taking in account the total number of synapses, W including biases

We have $\triangle w_{kj} = \eta \delta_k y_j = \eta \left(t_k - z_k \right) f' \left(net_k \right) y_j$ (Backward Pass)

We have that for this step

1. We need to calculate net_k linear in the number of weights.
2. We need to calculate $y_j = f\left(net_j \right)$ which is linear in the number of weights.

Computational Efficiency

Something Notable

The computational complexity of an algorithm is usually measured in terms of the number of multiplications, additions, and storage involved in its implementation.

- This is the electrical engineering approach!!!

Taking in account the total number of synapses, W including biases

We have $\triangle w_{kj} = \eta \delta_k y_j = \eta \left(t_k - z_k \right) f' \left(net_k \right) y_j$ (Backward Pass)

We have that for this step

1. We need to calculate net_k linear in the number of weights.
2. We need to calculate $y_j = f \left(net_j \right)$ which is linear in the number of weights.

Computational Efficiency

Now the Forward Pass

$$\Delta w_{ji} = \eta x_i \delta_j = \eta f'\left(net_j\right)\left[\sum_{k=1}^{c} w_{kj}\delta_k\right] x_i$$

We have that for this step

$\left[\sum_{k=1}^{c} w_{kj}\delta_k\right]$ takes, because of the previous calculations of δ_k's, linear on the number of weights

Clearly all this takes to have memory

In addition the calculation of the derivatives of the activation functions, but assuming a constant time.

Computational Efficiency

Now the Forward Pass

$$\Delta w_{ji} = \eta x_i \delta_j = \eta f'\left(net_j\right)\left[\sum_{k=1}^{c} w_{kj}\delta_k\right] x_i$$

We have that for this step

$\left[\sum_{k=1}^{c} w_{kj}\delta_k\right]$ takes, because of the previous calculations of δ_k's, linear on the number of weights

Clearly all this takes to have memory

In addition the calculation of the derivatives of the activation functions, but assuming a constant time.

Computational Efficiency

Now the Forward Pass

$$\Delta w_{ji} = \eta x_i \delta_j = \eta f'\left(net_j\right)\left[\sum_{k=1}^{c} w_{kj}\delta_k\right] x_i$$

We have that for this step

$\left[\sum_{k=1}^{c} w_{kj}\delta_k\right]$ takes, because of the previous calculations of δ_k's, linear on the number of weights

Clearly all this takes to have memory

In addition the calculation of the derivatives of the activation functions, but assuming a constant time.

We have that

The Complexity of the multi-layer perceptron is

$$O(W) \text{ Complexity}$$

Exercises

We have from NN by Haykin

4.2, 4.3, 4.6, 4.8, 4.16, 4.17, 3.7