Perceptron Algorithm Visualization

One epoch terminate if no change in $w$
Least Mean Square

Let us consider the weight correction in terms of an error function

\[ E^{(i)} = \frac{1}{2} (y^{(i)} - t^{(i)})^2, \]

where \( g(\cdot) \) is a differentiable function. Apply gradient descent rule

\[ w^{(i+1)} = w^{(i)} - \eta \frac{\partial E^{(i)}}{\partial w}, \]

where \( \frac{\partial E^{(i)}}{\partial w} = \left( y^{(i)} - t^{(i)} \right) x^{(i)} \delta^{(i)} \)

gives change in weights

\[ \Delta w = -\eta \delta^{(i)} x^{(i)} = -\eta \frac{\partial E^{(i)}}{\partial w} \]

Delta rule \( \equiv \) \{ Adaline rule, Widrow-Hoff rule, Least Mean Square (LMS) \}
Least Mean Square

Note, if we choose $g(a) = a$ to be the linear activation function $w^T x$, then there exists a closed analytical solution (pseudo-inverse solution).

Let $g(a)$ be a differentiable non-linear activation function, where $a = w^T x$.

$$\frac{\partial E^{(i)}}{\partial w} = \delta^{(i)} x^{(i)}, \text{ where } \delta^{(i)} = g'(a)(y^{(i)} - t^{(i)})$$

gives change in weights

$$\Delta w = -\eta \delta^{(i)} x^{(i)} = -\eta \frac{\partial E^{(i)}}{\partial w}$$
LMS Online/Batch Learning

Online learning:

• Update weight $w^{(i+1)} = w^{(i)} - \eta \frac{\partial E^{(i)}}{\partial w}$ (pattern by pattern).

This type of online learning is also called stochastic gradient descent, it is an approximation of the true gradient.

Batch learning:

• Update weight $w^{(i+1)} = w^{(i)} - \eta \sum_{i=1}^{N} \frac{\partial E^{(i)}}{\partial w}$ by computing derivatives for each pattern separately and then sum over all patterns.
Minimum Squared Error and Pseudoinverse

Recall that we want to minimize the squared error

\[ E(w) = \sum_{i=1}^{N} \frac{1}{2} \left( y^{(i)} - t^{(i)} \right)^2 \]

where \( y^{(i)} = w^T x^{(i)} \)

Let \( X \) be the \( N \times \tilde{d} \) matrix where \( \tilde{d} = d + 1 \) and \( i \)th row denotes training pattern \( x^{(i)T} \), \( w \) is weight vector, \( t \) class label vector.

\[
\begin{pmatrix}
  x_{10} & x_{11} & \cdots & x_{1d} \\
  x_{20} & x_{21} & \cdots & x_{2d} \\
  \vdots & \vdots & \ddots & \vdots \\
  x_{N0} & x_{N1} & \cdots & x_{Nd}
\end{pmatrix}
\begin{pmatrix}
  w_0 \\
  w_1 \\
  \vdots \\
  w_d
\end{pmatrix}
= 
\begin{pmatrix}
  t_0 \\
  t_1 \\
  \vdots \\
  t_N
\end{pmatrix}
\]

\( Xw = t \)
Problem: find weight vector $\mathbf{w}$, that is, solve $\mathbf{X}\mathbf{w} = \mathbf{t}$.

If $\mathbf{X}$ is non-singular solve $\mathbf{w} = \mathbf{X}^{-1}\mathbf{t}$, however, if $\mathbf{X}$ is rectangular (which is usually the case), then there are more equations than unknowns, that is, the equation system is overdetermined.

Let us search for $\mathbf{w}$ that minimizes the error

$$e = \mathbf{X}\mathbf{w} - \mathbf{t}$$

one approach is to minimize the squared length of the error vector $\mathbf{e}$

$$J(\tilde{\mathbf{w}}) = \|\mathbf{X}\mathbf{w} - \mathbf{t}\|^2 = \sum_{i=1}^{N} \left(\mathbf{w}^T \mathbf{x}^{(i)} - t^{(i)}\right)^2$$
MSE and Pseudoinverse (cont.)

Forming the gradient

\[
\nabla J = \sum_{i=1}^{N} 2 \left( w^T x^{(i)} - t^{(i)} \right) x^{(i)} = 2X^T(Xw - t)
\]

and setting \( \nabla J \) to zero gives \( X^T Xw = X^T t \). Observe that \( X^T X \) is a \( \tilde{d} \times \tilde{d} \) matrix which often is non-singular. In the non-singular case, one can solve \( w \) uniquely as

\[
w = \left( X^T X \right)^{-1} X^T t = X^\dagger t
\]

The \( \tilde{d} \times N \) matrix \( X^\dagger \equiv \left( X^T X \right)^{-1} X^T \) is called pseudoinverse of \( X \).
Linear Separability

Decision boundaries of single-layer networks are linear (hyperplanar in higher dimensions).

- Very restricted class of decision boundaries
- Examples:

XOR-Problem

Points are not linearly separable
Probability for Linear Separability

• Probability that a random set of points will be linearly separable

• Suppose we have \( N \) points distributed at random in \( d \) dimensions in general position (not collinear)

• Randomly assign each of the points to one of the two classes \( C_1 \) and \( C_2 \) (with eq. probability)

• For \( N \) data points there are \( 2^N \) possible class assignments (dichotomies \( \equiv \) binary partitions)

Question: What fraction \( F(N, d) \) of these dichotomies is linearly separable?
Probability for Linear Separability (cont.)

\[
F(N, d) = \begin{cases} 
1 & \text{when } N \leq d + 1 \\
\frac{1}{2^{N-1}} \sum_{i=0}^{d} \binom{N-1}{i} & \text{when } N \geq d + 1
\end{cases}
\]

If number of points is \( \leq d + 1 \), then any labeling leads to a separable problem.
History of Neural Networks

1943 Model of McCulloch and Pitts
1962/1960 Adaline (Widrow and Hoff), Perceptron (Rosenblatt)
1969 Book: Perceptrons (Minsky and Papert)

Decline of neural network research

1969 Book: Perceptrons (Minsky and Papert)
1962/1960 Adaline (Widrow and Hoff), Perceptron (Rosenblatt)
1943 Model of McCulloch and Pitts

1982 Hopfield Network, (Hopfield), Recurrent Networks, Energy Function
1986/1985 Backpropagation (Rumelhart, Hinton, Williams, Le Cun (actually first proposed by Werbos, 1974))
1992 A Training Algorithm for Optimal Margin Classifiers (Boser, Guyon and Vapnik), first paper on SVM
1995 Support-Vector Networks (Cortes and Vapnik)

Era of Neural Networks

Era of Kernel Methods (SVM, Kernel-PCA, Kernel-Fisher Discriminants, etc.)
Neural Networks are however still and frequently used

Note, this historical overview is far from being complete (see books for detailed historical overview)
Activation functions

Discrimination functions of the form \( y(x) = w^T x + w_0 \) are simple linear functions of the input variables \( x \), where distances are measured by means of the dot product.

Let us consider the non-linear \textit{logistic sigmoid} activation function \( g(\cdot) \) for limiting the output to \((0, 1)\), that is,

\[
y(x) = g(w^T x + w_0),
\]

where

\[
g(a) = \frac{1}{1 + \exp(-a)}
\]

Single-layer network with a logistic sigmoid activation function can also output posterior probabilities (rather than geometric distances).
Activation functions (cont.)

Heaviside step function:

\[ g(a) = \begin{cases} 
0 & \text{if } a < 0 \\ 
1 & \text{if } a \geq 0 
\end{cases} \]

Hyperbolic tangent function:

\[ g(a) = \tanh(a) = \frac{\exp(a) - \exp(-a)}{\exp(a) + \exp(-a)} \]

Note, \( \tanh(a) \in (-1, 1) \)
Multi-Layer vs. Single-Layer Networks

Single-layer networks

• based on a linear combination of the input variables which is transformed by linear/non-linear activation function
• are limited in terms of the range of functions they can represent

Multi-layer networks

• consist of multiple layers and are capable of approximating any continuous functional mapping
• are compared to single-layer networks not so straightforward to train
Multi-Layer Network

Connection in first layer from input unit $i$ to hidden unit $j$ is denoted as $w_{ji}$. Connection from hidden unit $j$ to output unit $k$ is denoted as $v_{kj}$.
Hidden unit $j$ receives input

$$a_j = \sum_{i=1}^{d} w_{ji} x_i + w_{j0} = \sum_{i=0}^{d} w_{ji} x_i$$

and produces output

$$z_j = g(a_j) = g \left( \sum_{i=0}^{d} w_{ji} x_i \right).$$

Output unit $k$ thus receives

$$a_k = \sum_{j=1}^{M} v_{kj} z_j + v_{k0} = \sum_{j=0}^{M} v_{kj} z_j.$$
Multi-Layer Network (cont.)

and produces the final output

\[ y_k = g(a_k) = g \left( \sum_{j=0}^{M} v_{kj} z_j \right) = g \left( \sum_{j=0}^{M} v_{kj} g \left( \sum_{i=0}^{d} w_{ji} x_i \right) \right) \]

Note that the activation function \( g(\cdot) \) in the first layer can be different from those in the second layer (or other layers).
Multi-Layer Networks Example

Note: sometimes the layers of units are counted (here three layers), rather the layers of adaptive weights. In this course $L$-layer network is referred to a network with $L$ layers of adaptive weights.
LMS Learning Rule for Multi-Layer Networks

• We have seen that the LMS learning rule is based on the gradient descent algorithm.

• The LMS learning rule worked because the error is proportional to the square difference between actual output $y$ and target output $t$ and can be evaluated for each output unit.

• In a multi-layer network we can use LMS learning rule on the hidden-to-output layer weights because target outputs are known.

**Problem:** we cannot compute the target outputs of the input-to-hidden weights because these values are unknown, or, to put it the other way around, how to update the weights in the first layer?
Backpropagation (Hidden-to-Output Layer)

Recall that we want to minimize the error on training patterns between actual output $y_k$ and target output $t_k$:

$$E = \frac{1}{2} \sum_{k=1}^{K} (y_k - t_k)^2.$$

Backpropagation learning rule is based on gradient descent:

$$\Delta w = -\eta \frac{\partial E}{\partial w}, \text{ component form } \Delta w_{st} = -\eta \frac{\partial E}{\partial w_{st}}$$

Apply chain rule for differentiation:

$$\frac{\partial E}{\partial u_{k,j}} = \frac{\partial E}{\partial a_k} \frac{\partial a_k}{\partial v_{k,j}}$$
Gradient descent rule gives:

\[ \Delta v_{kj} = -\eta \frac{\partial E}{\partial v_{kj}} = -\eta (y_k - t_k)g'(a_k)z_j \]

\[ = -\eta \delta_k z_j \]

where

\[ \delta_k = (y_k - t_k)g'(a_k). \]

Observe that this result is identical to that obtained for LMS.
Backpropagation (Input-to-Hidden Layer)

For the input-to-hidden connection we must differentiate with respect to the $w_{ji}$'s which are deeply embedded in

$$E = \frac{1}{2} \sum_{k=1}^{K} \left[ g \left( \sum_{j=0}^{M} v_{kj} g \left( \sum_{i=0}^{d} w_{ji} x_i \right) \right) - t_k \right]^2$$

Apply chain rule:

$$\Delta w_{ji} = -\eta \frac{\partial E}{\partial w_{ji}} = -\eta \frac{\partial E}{\partial z_j} \frac{\partial z_j}{\partial a_j} \frac{\partial a_j}{\partial w_{ji}}$$

$$= -\eta \sum_{k=1}^{K} (y_k - t_k) g'(a_k) v_{kj} g'(a_j) x_i \delta_k$$

$$= -\eta \sum_{k=1}^{K} \delta_k v_{kj} g'(a_j) x_i$$
Backprop. (Input-to-Hidden Layer) (cont.)

\[ \Delta w_{ji} = -\eta \delta_j x_i \]

where

\[ \delta_j = g'(a_j) \sum_{k=1}^{K} v_{kj} \delta_k \]

Observe: that we need to propagate the errors (\( \delta \)'s) backwards to update the weights \( v \) and \( w \)

\[ \Delta v_{kj} = -\eta \delta_k z_j \]
\[ \delta_k = (y_k - t_k) g'(a_k) \]
\[ \Delta w_{ji} = -\eta \delta_j x_i \]
\[ \delta_j = g'(a_j) \sum_{k=1}^{K} v_{kj} \delta_k \]
Error Backpropagation

- Apply input $x$ and forward propagate through the network using $a_j = \sum_{i=0}^{d} w_{ji} x_i$ and $z_j = g(a_j)$ to find the activations of all the hidden and output units.
- Compute the deltas $\delta_k$ for all the output units using $\delta_k = (y_k - t_k) g'(a_k)$.
- Backpropagate the $\delta$'s using $\delta_j = g'(a_j) \sum_{k=1}^{K} v_{kj} \delta_k$ to obtain $\delta_j$ for each hidden unit in the network.

Time and space complexity:

d input units, $M$ hidden units and $K$ output units results in $M(d+1)$ weights in first layer and $K(M+1)$ weights in second layer. Space and time complexity is $O(M(K+d))$. If $e$ training epochs are performed, then time complexity is $O(e M(K+d))$. 

- p. 146
Backprop. (Output-to-Hidden Layer) Vis.

\[ x_0 = 1 \]
\[ z_0 = 1 \]
\[ y_1 \]
\[ y_2 \]
\[ \delta_1 \]
\[ v_{13}^{\text{new}} = v_{13} - \eta \delta_1 z_3 \]
Backprop. (Hidden-to-Input Layer) Vis.

\[ z_0 = 1 \]

\[ x_0 = 1 \]

\[ y_1 \]

\[ y_2 \]

\[ z_0 = 1 \]

\[ z_1 \]

\[ z_2 \]

\[ z_3 \]

\[ w_{12}^{\text{new}} = w_{12} - \eta \left[ g'(a_1)(v_{11}\delta_1 + v_{21}\delta_2) \right] x_2 \]
Property of Activation Functions

• In the Backpropagation algorithm the derivative of $g(a)$ is required to evaluate the $\delta$’s.

• Activation functions

$$g_1(a) = \frac{1}{1 + \exp(-\beta a)} \quad \text{and} \quad g_2(a) = \tanh(\beta a)$$

obey the property

$$g_1'(a) = \beta g_1(a)(1 - g_1(a))$$
$$g_2'(a) = \beta(1 - (g_2(a))^2)$$
Online Backpropagation Algorithm

input : \((x_1, t_1), \ldots, (x_N, t_N) \in \mathbb{R}^d \times \{C_1, C_2, \ldots, C_K\}, \eta \in \mathbb{R}_+, \text{max.epoch} \in \mathbb{N}, \epsilon \in \mathbb{R}_+ \)

output: \(w, v\)

begin
  Randomly initialize \(w, v\)
  epoch ← 0
  repeat
    for \(n ← 1 \text{ to } N\) do
      \(x ← \text{select pattern } x_n\)
      \(v_{kj} ← v_{kj} - \eta \delta_k z_j\)
      \(w_{ji} ← w_{ji} - \eta \delta_j x_i\)
    epoch ← epoch + 1
  until (\(\text{epoch = max.epoch}\) or (\(\|\nabla E\| < \epsilon\))

return \(w, v\)

end
Batch Backpropagation Algorithm

**Input:** 
\((x_1, t_1), \ldots, (x_N, t_N) \in \mathbb{R}^d \times \{C_1, C_2, \ldots, C_K\}, \eta \in \mathbb{R}_+, \text{max. epoch} \in \mathbb{N}, \epsilon \in \mathbb{R}_+

**Output:** \(w, v\)

begin

Randomly initialize \(w, v\)

\[\text{epoch} \leftarrow 0, \Delta w_{ji} \leftarrow 0, \Delta v_{kj} \leftarrow 0\]

repeat

\[\text{for } n \leftarrow 1 \text{ to } N \text{ do}\]

\[x \leftarrow \text{select pattern } x_n\]

\[\Delta v_{kj} \leftarrow \Delta v_{kj} - \eta \delta_k z_j, \Delta w_{ji} \leftarrow \Delta w_{ji} - \eta \delta_j x_i\]

\[v_{kj} \leftarrow v_{kj} + \Delta v_{kj}\]

\[w_{ji} \leftarrow w_{ji} + \Delta w_{ji}\]

\[\text{epoch} \leftarrow \text{epoch} + 1\]

until \((\text{epoch} = \text{max. epoch}) \text{ or } (\|\nabla E\| < \epsilon)\)

return \(w, v\)

end
Possible decision boundaries which can be generated by networks having various numbers of layers and using Heaviside activation function.
Multi-Layer NN for XOR Separability Problem

\[
g(a) = \begin{cases} 
-1 & \text{if } a < 0 \\
+1 & \text{if } a \geq 0
\end{cases}
\]

\[
\begin{array}{c|cc|c}
& x_1 & x_2 & x_1 \text{ XOR } x_2 \\
\hline
-1 & -1 & & -1 \\
-1 & +1 & & +1 \\
+1 & -1 & & +1 \\
+1 & +1 & & -1 \\
\end{array}
\]
Multi-Layer NN for XOR Sep. Problem (cont.)