Lecture 11: Single Layer Perceptrons
**Perceptron: architecture**

- We consider the architecture: feed-forward NN with one layer.
- It is sufficient to study single layer perceptrons with just one neuron:
Single layer perceptrons

- Generalization to single layer perceptrons with more neurons is easy because:
  - The output units are independent among each other
  - Each weight only affects one of the outputs
The (McCulloch-Pitts) perceptron is a single layer NN with a non-linear \( \varphi \), the sign function.

\[
o(x_1, \ldots, x_n) = \begin{cases} 1 & \text{if } w_0 + w_1 x_1 + \cdots + w_n x_n > 0 \\ -1 & \text{otherwise.} \end{cases}
\]

Sometimes we’ll use simpler vector notation:

\[
o(\vec{x}) = \begin{cases} 1 & \text{if } \vec{w} \cdot \vec{x} > 0 \\ -1 & \text{otherwise.} \end{cases}
\]
Perceptron

- Perceptron is based on a nonlinear neuron.
  - $\mathbf{x}(n) = [+1, x_1(n), x_2(n), ..., x_m(n)]^T$
  - $\mathbf{w}(n) = [b(n), w_1(n), w_2(n), ..., x_m(n)]^T$
  - $v(n) = w^T(n)x(n)$ defines a hyperplane

$$v = \sum_{i=1}^{m} w_i x_i + b$$

$$y = \text{sign} \left( b + \sum_{i=1}^{m} w_i x_i \right)$$

![Perceptron Diagram]
The perceptron is used for classification: classify correctly a set of examples into one of the two classes $C_1$ and $C_2$:

*If the output of the perceptron is $+1$, then the input is assigned to class $C_1$*

*If the output of the perceptron is $-1$, then the input is assigned to $C_2$*
How can we train a perceptron for a classification task?

We try to find suitable values for the weights in such a way that the training examples are correctly classified.

Geometrically, we try to find a hyper-plane that separates the examples of the two classes.
The equation below describes a hyperplane in the input space. This hyperplane is used to separate the two classes $C_1$ and $C_2$. The equation is:

$$\sum_{i=1}^{m} w_i x_i + b = 0$$

The decision region for $C_1$ is:

$$w_1 x_1 + w_2 x_2 + b > 0$$

The decision region for $C_2$ is:

$$w_1 x_1 + w_2 x_2 + b \leq 0$$
Example: AND

- Here is a representation of the AND function
- White means \textit{false}, black means \textit{true} for the output
- -1 means \textit{false}, +1 means \textit{true} for the input
- A linear decision surface separates \textit{false} from \textit{true} instances

\begin{align*}
-1 \text{ AND } -1 &= \text{false} \\
-1 \text{ AND } +1 &= \text{false} \\
+1 \text{ AND } -1 &= \text{false} \\
+1 \text{ AND } +1 &= \text{true}
\end{align*}
Example: AND

- Watch a perceptron learn the AND function:
Example: XOR

- Here’s the XOR function:

\[-1 \text{ XOR } -1 = false\]
\[-1 \text{ XOR } +1 = true\]
\[+1 \text{ XOR } -1 = true\]
\[+1 \text{ XOR } +1 = false\]

Perceptrons cannot learn such \textit{linearly inseparable} functions
Example: XOR

- Watch a perceptron try to learn XOR
Perceptron learning

- If the n-th member of the training set, $x(n)$, is correctly classified by the weight vector $w(n)$ computed at the n-th iteration of the algorithm, no correction is made to the weight vector, i.e:
  - $w(n+1) = w(n)$ if $w^T x(n) > 0$ and $x(n)$ belongs to class $C_1$
  - $w(n+1) = w(n)$ if $w^T x(n) \leq 0$ and $x(n)$ belongs to class $C_2$

- Otherwise the weight vector of the perceptron is updated:
  - $w(n+1) = w(n) - \eta(n)x(n)$ if $w^T(n)x(n) > 0$ and $x(n)$ belongs to class $C_2$
  - $w(n+1) = w(n) + \eta(n)x(n)$ if $w^T(n)x(n) \leq 0$ and $x(n)$ belongs to class $C_1$

- Learning-rate parameter $\eta(n)$ controls the adjustment applied to the weight vector at iteration $n$. 
Perceptron learning

- Or, simply
  - $w(n + 1) = w(n) + \eta(n)e(n)x(n)$

- where
  - $e(n) = d(n) - y(n)$ [the error]

- The perceptron uses the McCulloch-Pitts formal model of a neuron.
- The learning process is performed for a finite number of iterations and then stops
Learning algorithm

**Epoch:** Presentation of the entire training set to the neural network.
In the case of the AND function an epoch consists of four sets of inputs being presented to the network (i.e. [0,0], [0,1], [1,0], [1,1])

**Error:** The error value is the amount by which the value output by the network differs from the target value. For example, if we required the network to output 0 and it outputs a 1, then Error = -1
In simple cases, divide feature space by drawing a hyperplane across it.

Known as a decision boundary.

Discriminant function: returns different values on opposite sides.

Problems which can be thus classified are linearly separable.

---

Linearly separable

Non-Linearly separable
Example

- Consider the 2-dimensional training set \( C_1 \cup C_2 \),
- \( C_1 = \{(1, 1, 1), (1, 1, -1), (1, 0, -1)\} \) with class label 1
- \( C_2 = \{(1, -1,-1), (1, -1,1), (1, 0,1)\} \) with class label 0
- Train a perceptron on \( C_1 \cup C_2 \)
- First pass is as follows

<table>
<thead>
<tr>
<th>Input</th>
<th>Weight</th>
<th>Desired</th>
<th>Actual</th>
<th>Update?</th>
<th>New weight</th>
</tr>
</thead>
<tbody>
<tr>
<td>(1, 1, 1)</td>
<td>(1, 0, 0)</td>
<td>1</td>
<td>1</td>
<td>No</td>
<td>(1, 0, 0)</td>
</tr>
<tr>
<td>(1, 1, -1)</td>
<td>(1, 0, 0)</td>
<td>1</td>
<td>1</td>
<td>No</td>
<td>(1, 0, 0)</td>
</tr>
<tr>
<td>(1,0, -1)</td>
<td>(1, 0, 0)</td>
<td>1</td>
<td>1</td>
<td>No</td>
<td>(1, 0, 0)</td>
</tr>
<tr>
<td>(1,-1, -1)</td>
<td>(1, 0, 0)</td>
<td>0</td>
<td>1</td>
<td>Yes</td>
<td>(0, 1, 1)</td>
</tr>
<tr>
<td>(1,-1, 1)</td>
<td>(0, 1, 1)</td>
<td>0</td>
<td>0</td>
<td>No</td>
<td>(0, 1, 1)</td>
</tr>
<tr>
<td>(1, 0, 1)</td>
<td>(0, 1, 1)</td>
<td>0</td>
<td>1</td>
<td>Yes</td>
<td>(-1, 1, 0)</td>
</tr>
</tbody>
</table>
C1: \{ (1, 1, 1), (1, 1, -1), (1, 0, -1) \}
C2: \{ (1, -1,-1), (1, -1,1), (1, 0,1) \}

Fill out this table sequentially (Second pass):

<table>
<thead>
<tr>
<th>Input</th>
<th>Weight</th>
<th>Desired</th>
<th>Actual</th>
<th>Update?</th>
<th>New weight</th>
</tr>
</thead>
<tbody>
<tr>
<td>(1, 1, 1)</td>
<td>(-1, 1, 0)</td>
<td>1</td>
<td>0</td>
<td>Yes</td>
<td>(0, 2, 1)</td>
</tr>
<tr>
<td>(1, 1, -1)</td>
<td>(0, 2, 1)</td>
<td>1</td>
<td>1</td>
<td>No</td>
<td>(0, 2, 1)</td>
</tr>
<tr>
<td>(1,0, -1)</td>
<td>(0, 2, 1)</td>
<td>1</td>
<td>0</td>
<td>Yes</td>
<td>(1, 2, 0)</td>
</tr>
<tr>
<td>(1,-1, -1)</td>
<td>(1, 2, 0)</td>
<td>0</td>
<td>0</td>
<td>No</td>
<td>(1, 2, 0)</td>
</tr>
<tr>
<td>(1,-1, 1)</td>
<td>(1, 2, 0)</td>
<td>0</td>
<td>0</td>
<td>No</td>
<td>(1, 2, 0)</td>
</tr>
<tr>
<td>(1, 0, 1)</td>
<td>(1, 2, 0)</td>
<td>0</td>
<td>1</td>
<td>Yes</td>
<td>(0, 2, -1)</td>
</tr>
</tbody>
</table>
Example

C1: \{(1, 1, 1), (1, 1, -1), (1, 0, -1)\}
C2: \{(1, -1, -1), (1, -1, 1), (1, 0, 1)\}

Fill out this table sequentially (Third pass):

<table>
<thead>
<tr>
<th>Input</th>
<th>Weight</th>
<th>Desired</th>
<th>Actual</th>
<th>Update?</th>
<th>New weight</th>
</tr>
</thead>
<tbody>
<tr>
<td>(1, 1, 1)</td>
<td>(0, 2, -1)</td>
<td>1</td>
<td>1</td>
<td>No</td>
<td>(0, 2, -1)</td>
</tr>
<tr>
<td>(1, 1, -1)</td>
<td>(0, 2, -1)</td>
<td>1</td>
<td>1</td>
<td>No</td>
<td>(0, 2, -1)</td>
</tr>
<tr>
<td>(1, 0, -1)</td>
<td>(0, 2, -1)</td>
<td>1</td>
<td>1</td>
<td>No</td>
<td>(0, 2, -1)</td>
</tr>
<tr>
<td>(1, -1, -1)</td>
<td>(0, 2, -1)</td>
<td>0</td>
<td>0</td>
<td>No</td>
<td>(0, 2, -1)</td>
</tr>
<tr>
<td>(1, -1, 1)</td>
<td>(0, 2, -1)</td>
<td>0</td>
<td>0</td>
<td>No</td>
<td>(0, 2, -1)</td>
</tr>
<tr>
<td>(1, 0, 1)</td>
<td>(0, 2, -1)</td>
<td>0</td>
<td>0</td>
<td>No</td>
<td>(0, 2, -1)</td>
</tr>
</tbody>
</table>

At epoch 3 no weight changes.
⇒ stop execution of algorithm.
Final weight vector: (0, 2, -1).
⇒ decision hyperplane is \(2x_1 - x_2 = 0\).
Example: final results

Decision boundary: $2x_1 - x_2 = 0$
Some Unhappiness About Perceptron Training

- The perceptron learning rule fails to converge if examples are not linearly separable
  - Can only model linearly separable classes, like (those described by) the following Boolean functions:
    - AND, OR, but not XOR
- When a perceptron gives the right answer, no learning takes place.
- Anything below the threshold is interpreted as “no”, even if it is just below the threshold.
  - Might be better to train the neuron based on how far below the threshold it is?
Suppose datasets $C_1$ and $C_2$ are linearly separable. The perceptron learning algorithm converges after $n_0$ iterations, with $n_0 \leq n_{\text{max}}$ on training set $C_1 \cup C_2$.

**Proof:**
- Suppose $x \in C_1 \Rightarrow \text{output} = 1$ and $x \in C_2 \Rightarrow \text{output} = -1$.
- For simplicity assume $w(1) = 0, \eta = 1$.
- Suppose perceptron incorrectly classifies $x(1) \ldots x(n) \in C_1$. Then $w^T(k) x(k) \leq 0$.

  $\Rightarrow$ Error correction rule:
  \[
  \begin{align*}
  w(2) &= w(1) + x(1) \\
  w(3) &= w(2) + x(2) \\
  \vdots & \quad \vdots \\
  w(n+1) &= w(n) + x(n).
  \end{align*}
  \]

  $\Rightarrow w(n+1) = x(1)+ \ldots + x(n)$.

  $w(n+1) = w(n) + x(n)$. 
Perceptron: Convergence Theorem

• Let $w_0$ be such that $w_0^T x(n) > 0 \quad \forall x(n) \in C_1$. $w_0$ exists because $C_1$ and $C_2$ are linearly separable.

• Let $\alpha = \min w_0^T x(n) \mid x(n) \in C_1$.

• Then, $w_0^T w(n+1) = w_0^T x(1) + \ldots + w_0^T x(n) \geq n\alpha$

• Cauchy-Schwarz inequality:
  \[ ||w_0||^2 \cdot ||w(n+1)||^2 \geq [w_0^T w(n+1)]^2 \]
  \[ ||w(n+1)||^2 \geq \frac{n^2 \alpha^2}{||w_0||^2} \quad \text{(A)} \]
• Now we consider another route:

\[ \mathbf{w}(k+1) = \mathbf{w}(k) + \mathbf{x}(k) \]

\[ || \mathbf{w}(k+1) ||^2 = || \mathbf{w}(k) ||^2 + || \mathbf{x}(k) ||^2 + 2 \mathbf{w}^T(k)\mathbf{x}(k) \]

\( \Rightarrow \) Euclidean norm

\[ || \mathbf{w}(k+1) ||^2 \leq || \mathbf{w}(k) ||^2 + || \mathbf{x}(k) ||^2 \quad \text{k}=1,..,n \]

\[ || \mathbf{w}(2) ||^2 \leq || \mathbf{w}(1) ||^2 + || \mathbf{x}(1) ||^2 \]

\[ || \mathbf{w}(3) ||^2 \leq || \mathbf{w}(2) ||^2 + || \mathbf{x}(2) ||^2 \]

\[ \vdots \]

\[ \Rightarrow || \mathbf{w}(n+1) ||^2 \leq \sum_{k=1}^{n} || \mathbf{x}(k) ||^2 \]
Perceptron: Convergence Theorem

- Let \( \beta = \max ||x(n)||^2 \quad x(n) \in C_1 \)
- \( ||w(n+1)||^2 \leq n \beta \) (B)
- For sufficiently large values of \( n \):
  (B) becomes in conflict with (A).
  Then, \( n \) cannot be greater than \( n_{\text{max}} \) such that (A) and (B) are both satisfied with the equality sign.

\[
\frac{n_{\text{max}} \alpha^2}{||w_0||^2} = n_{\text{max}} \beta \quad \Rightarrow \quad n_{\text{max}} = \frac{||w_0||^2}{\alpha^2} \beta
\]

- Perceptron convergence algorithm

\[
n_{\text{max}} = \frac{\beta ||w_0||^2}{\alpha^2} \quad \text{iterations.}
\]
Fixed-increment convergence theorem

• Let the subsets of training vectors C1 and C2 be linearly separable. Let the inputs presented to perceptron originate from these two subsets. The perceptron converges after some \( n_0 \) iterations, in the sense that

\[
w(n_0) = w(n_0+1) = w(n_0+2) = \ldots
\]

• is a solution vector for \( n_0 \leq n_{\text{max}} \).
Learning-Rate Annealing Schedules

- When learning rate is large, trajectory may follow zigzagging path
- When it is small, procedure may be slow
- Simplest learning rate parameter
  \[ \eta(n) = \eta_0 \]
- Stochastic approximation
  - time-varying
  - when \( c \) is large, danger of parameter blowup for small \( n \).
  \[ \eta(n) = \frac{c}{n} \quad (c \text{ is constant}) \]
Learning-Rate Annealing Schedules

- Search then converge schedule
  - in early stages, learning rate parameter is approximately equal to $\eta_0$
  - for a number of iteration $n$ large compared to search time constant $\tau$,
  - learning rate parameter approximates as $c/n$

$$\eta(n) = \frac{\eta_0}{1 + (n / \tau)}$$

($\eta_0$, $\tau$ is constant)
Summary

1. Initialization
   - set \( w(0) = 0 \)

2. Activation
   - at time step \( n \), activate perceptron by applying continuous valued input vector \( x(n) \) and desired response \( d(n) \)

3. Computation of actual response
   \[
   y(n) = \text{sgn}[w^T(n)x(n)]
   \]

4. Adaptation of Weight Vector
   \[
   w(n+1) = w(n) + \eta[d(n) - y(n)]x(n)
   \]
   \[
   d(n) = \begin{cases} 
   +1 & \text{if } x(n) \text{ belongs to class } C_1 \\
   -1 & \text{if } x(n) \text{ belongs to class } C_2 
   \end{cases}
   \]

5. Continuation
   - increment time step \( n \) and go back to step 2
Perceptron Learning Example

\[ \theta = w_0 \]

\[ a = \sum_{i=0}^{n} w_i x_i \]

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

Thus, \( y = \text{sign}(a) = 0 \) or 1
Perceptron Learning Example

\[ w = [0.25 \ -0.1 \ 0.5] \]
\[ x_2 = 0.2 \ x_1 - 0.5 \]

\[
\begin{align*}
(x,t) &\equiv ([2,1],1) \\
0 &\equiv \text{sgn}(0.25 + 0.1 - 0.5) \\
&\equiv \text{sgn}(0.45 - 0.6 + 0.3) \\
&\equiv \text{sgn}(0.25 - 0.7 + 0.1) \\
&= -1
\end{align*}
\]

\[
\Delta w = [0.2 = 0.4 = 0.2]
\]
\[ \Delta w = [0.2 \ 0.2 \ 0.2] \]
Adaline: Adaptive Linear Element

- The output $y$ is a linear combination of $x$.

$$y = \sum_{j=0}^{m} x_j(n)w_j(n)$$
Adaline: Adaptive Linear Element

- Adaline: uses a linear neuron model and the Least-Mean-Square (LMS) learning algorithm
  The idea: try to minimize the square error, which is a function of the weights

\[ E(w(n)) = \frac{1}{2} e^2(n) \]

\[ e(n) = d(n) - \sum_{j=0}^{m} x_j(n)w_j(n) \]

- We can find the minimum of the error function \( E \) by means of the Steepest descent method
Steepest Descent Method

- start with an arbitrary point
- find a direction in which $E$ is decreasing most rapidly

\[- (\text{gradient of } E(w)) = - \left[ \frac{\partial E}{\partial w_1}, \ldots, \frac{\partial E}{\partial w_m} \right] \]

- make a small step in that direction

\[w(n + 1) = w(n) - \eta(\text{gradient of } E(n))\]
## Comparison of Perceptron and Adaline

<table>
<thead>
<tr>
<th></th>
<th>Perceptron</th>
<th>Adaline</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Architecture</strong></td>
<td>Single-layer</td>
<td>Single-layer</td>
</tr>
<tr>
<td><strong>Neuron model</strong></td>
<td>Non-linear</td>
<td>Linear</td>
</tr>
<tr>
<td><strong>Learning algorithm</strong></td>
<td>Minimize number of misclassified examples</td>
<td>Minimize total squared error</td>
</tr>
<tr>
<td><strong>Application</strong></td>
<td>Linear classification</td>
<td>Linear classification, regression</td>
</tr>
</tbody>
</table>
Reading

• S Haykin, Neural Networks: A Comprehensive Foundation, 2007 (Chapter 4).