Neurons

**Soma** is the central part of the neuron:
- *where the input signals are combined.*

**Dendrites** are cellular extensions:
- *where majority of the input occurs.*

**Axon** is a fine, long projection:
- *carries nerve signals to other neurons.*

**Synapses** are molecular structures between axon terminals and other neurons:
- *where the communication takes place.*
- **Biological Interpretation:**
  - The output of the neuron is a linear combination of inputs from other neurons, rescaled by the synaptic weights.
    - It is often transformed through a monotonic function such as *signum*, or *sigmoid*.
    - weights $w_i$ correspond to the synaptic weights (activating or inhibiting).
    - summation corresponds to combination of signals in the soma.
The Perceptron Algorithm: Two Classes

1. initialize parameters \( w = 0 \)
2. for \( i = 1 \ldots n \)
3. \[ y_i = sgn(w^T \varphi(x_i)) \]
4. if \( y_i \neq t_i \) then
5. \[ w = w + t_i \varphi(x_i) \]

Repeat:
- a) until convergence.
- b) for a number of epochs E.

Theorem [Rosenblatt, 1962]:
If the training dataset is linearly separable, the perceptron learning algorithm is guaranteed to find a solution in a finite number of steps.
- see Theorem 1 (Block, Novikoff) in [Freund & Schapire, 1999].
Motivation: Error function minimization

- Error: total number of missclassified patterns?
  - piecewise constant function of $w$ with discontinuities.
  - cannot use gradient methods (gradient zero almost everywhere).

- The *Perceptron Criterion*:
  - Assume classes $T = \{c_1, c_2\} = \{-1, +1\}$.
  - Want $w^T \varphi(x_n) \geq 0$ for $t_n = +1$, and $w^T \varphi(x_n) < 0$ for $t_n = -1$.
  \[ \Rightarrow \] would like to have $w^T \varphi(x_n) t_n > 0$ for all patterns
  \[ \Rightarrow \] want to minimize $-w^T \varphi(x_n) t_n$ for all missclassified patterns.

\[ \Rightarrow \text{minimize } E_P(w) = - \sum_{n \in M} w^T \varphi(x_n) t_n \]
The Perceptron Algorithm as Stochastic Gradient Descent

• Update parameters \( \mathbf{w} \) sequentially:

\[
\mathbf{w}^{(\tau+1)} = \mathbf{w}^{(\tau)} - \eta \nabla E_P(\mathbf{w}^{(\tau)}, x_n)
\]

\[
\mathbf{w}^{(\tau+1)} = \mathbf{w}^{(\tau)} + \eta \varphi(x_n) t_n
\]

• The magnitude of \( \mathbf{w} \) is inconsequential \( \Rightarrow \eta = 1 \).
The Perceptron Algorithm: K classes

1. **initialize** parameters $w = 0$
2. for $i = 1 \ldots n$
3. $c_j = \arg \max_{t \in T} w^T \varphi(x_i, t)$
4. if $c_j \neq t_i$ then
5. $w = w + \varphi(x_i, t_i) - \varphi(x_i, c_j)$

During testing:

$$t^* = \arg \max_{t \in T} w^T \varphi(x, t)$$

Repeat:

a) until convergence.
b) for a number of epochs E.
Averaged Perceptron

1. **initialize** parameters \( w = 0, \tau = 1, \overline{w} = 0 \)
2. **for** \( i = 1 \ldots n \)
3. \( y_i = \text{sgn}(w^T \varphi(x_i)) \)
4. **if** \( y_i \neq t_i \) **then**
5. \( w = w + t_i \varphi(x_i) \)
6. \( \overline{w} = \overline{w} + w \)
7. \( \tau = \tau + 1 \)
8. **return** \( \overline{w} / \tau \)

During testing: \( t^* = \text{sgn} \overline{w}^T \phi(x) \)

Repeat:
- a) until convergence.
- b) for a number of epochs \( E \).
Averaged Perceptron: K classes

1. **initialize** parameters $w = 0, \tau = 1, \overline{w} = 0$
2. **for** $i = 1 \ldots n$
3. $c_j = \arg\max_{t \in T} w^T \varphi(x_i, t)$
4. **if** $c_j \neq t_i$ **then**
5. $w = w + \varphi(x_i, t_i) - \varphi(x_i, c_j)$
6. $\overline{w} = \overline{w} + w$
7. $\tau = \tau + 1$
8. **return** $\overline{w} / \tau$

During testing: $t^* = \arg\max_{t \in T} \overline{w}^T \varphi(x, t)$

Repeat:
- a) until convergence.
- b) for a number of epochs $E$. 

Lecture 03
The Perceptron Algorithm: Two Classes

1. initialize parameters $w = 0$
2. for $i = 1 \ldots n$
3. $y_i = \text{sgn}(w^T \phi(x_i))$
4. if $y_i \neq t_i$ then
5. $w = w + t_i \phi(x_i)$

Loop invariant: $w$ is a weighted sum of training vectors:

$$w = \sum_i \alpha_i t_i \phi(x_i) \quad \Rightarrow \quad w^T \phi(x) = \sum_i \alpha_i t_i \phi(x_i)^T \phi(x)$$

Repeat:

a) until convergence.
b) for a number of epochs $E$. 

Lecture 03
Kernel Perceptron: Two Classes

1. define $f(x) = \sum_i \alpha_i t_i \varphi(x_i)^T \varphi(x) = \sum_i \alpha_i t_i K(x_i, x)$
2. initialize dual parameters $\alpha_i = 0$
3. for $i = 1 \ldots n$
4. $y_i = \text{sgn} f(x_i)$
5. if $y_i \neq t_i$ then
6. $\alpha_i = \alpha_i + 1$

During testing: $t = \text{sgn} f(x)$
Kernel Perceptron: Two Classes

1. **define** \( f(\mathbf{x}) = \sum_{i} \alpha_i \varphi(\mathbf{x}_i)^T \varphi(\mathbf{x}) = \sum_{i} \alpha_i K(\mathbf{x}_i, \mathbf{x}) \)

2. **initialize** dual parameters \( \alpha_i = 0 \)

3. **for** \( i = 1 \ldots n \)

4. \( y_i = \text{sgn} \, f(\mathbf{x}_i) \)

5. **if** \( y_i \neq t_i \) **then**

6. \( \alpha_i = \alpha_i + t_i \)

During testing: \( t = \text{sgn} \, f(\mathbf{x}) \)
The Perceptron Algorithm: K classes

1. **initialize** parameters $w = 0$
2. for $i = 1 \ldots n$
3. $c_j = \arg \max_{t \in T} w^T \varphi(x_i, t)$
4. if $c_j \neq t_i$ then
5. \[ w = w + \varphi(x_i, t_i) - \varphi(x_i, c_j) \]

<table>
<thead>
<tr>
<th>Repeat:</th>
</tr>
</thead>
<tbody>
<tr>
<td>a) until convergence.</td>
</tr>
<tr>
<td>b) for a number of epochs E.</td>
</tr>
</tbody>
</table>

Loop invariant: $w$ is a weighted sum of training vectors:

\[
w = \sum_{i,j} \alpha_{ij} (\varphi(x_i, t_i) - \varphi(x_i, c_j))
\]

\[
\Rightarrow w^T \varphi(x, t) = \sum_{i,j} \alpha_{ij} (\varphi(x_i, t_i)^T \varphi(x, t) - \varphi(x_i, c_j)^T \varphi(x, t))
\]
Kernel Perceptron: K classes

1. **define** $f(x, t) = \sum_{i,j} \alpha_{ij} (\varphi(x_i, t_i)^T \varphi(x, t) - \varphi(x_i, c_j)^T \varphi(x, t))$

2. **initialize** dual parameters $\alpha_{ij} = 0$

3. **for** $i = 1 \ldots n$

4. $c_j = \arg \max_{t \in T} f(x_i, t)$

5. **if** $c_j \neq t_i$ **then**

6. $\alpha_{ij} = \alpha_{ij} + 1$

**Repeat:**

a) until convergence.

b) for a number of epochs $E$.

During testing:

$t^* = \arg \max_{t \in T} f(x, t)$
Kernel Perceptron: K classes

- Discriminant function:

\[
f(x, t) = \sum_{i,j} \alpha_{i,j} (\varphi(x_i, t_i)^T \varphi(x, t) - \varphi(x_i, c_j)^T \varphi(x, t))
\]

\[
= \sum_{i,j} \alpha_{ij} (K(x_i, t_i, x, t) - K(x_i, c_j, x, t))
\]

where:

\[
K(x_i, t_i, x, t) = \varphi^T (x_i, t_i) \varphi(x, t)
\]

\[
K(x_i, c_j, x, t) = \varphi^T (x_i, c_j) \varphi(x, t)
\]
The Perceptron vs. Boolean Functions

\[ \varphi(x) = [1, x_1, x_2]^T \]
\[ w = [w_0, w_1, w_2]^T \]

\[ \Rightarrow w^T \varphi(x) = [w_1, w_2]^T [x_1, x_2] + w_0 \]
Perceptron with Quadratic Kernel

• Discriminant function:

\[ f(x) = \sum \alpha_t \varphi(x)^T \varphi(x) = \sum \alpha_t K(x, x) \]

• Quadratic kernel:

\[ K(x, y) = (x^T y)^2 = (x_1 y_1 + x_2 y_2)^2 \]

⇒ corresponding feature space \( \varphi(x) = ? \)

conjunctions of two atomic features
Perceptron with Quadratic Kernel

Linear kernel $K(x, y) = x^T y$

Quadratic kernel $K(x, y) = (x^T y)^2$
Quadratic Kernels

- Circles, hyperbolas, and ellipses as separating surfaces:

\[ K(x, y) = (1 + x^T y)^2 = \varphi(x)^T \varphi(y) \]

\[ \varphi(x) = [1, \sqrt{2}x_1, \sqrt{2}x_2, x_1^2, \sqrt{2}x_1x_2, x_2^2]^T \]
Quadratic Kernels

\[ K(x, y) = (x^T y)^2 = \varphi(x)^T \varphi(y) \]
Explicit Features vs. Kernels

- Explicitly enumerating features can be prohibitive:
  - 1,000 basic features for $\mathbf{x}^T\mathbf{y}$ $\Rightarrow$ 500,500 quadratic features for $(\mathbf{x}^T\mathbf{y})^2$
  - Much worse for higher order features.

- Solution:
  - Do not compute the feature vectors, compute kernels instead (i.e. compute dot products between implicit feature vectors).
    - $(\mathbf{x}^T\mathbf{y})^2$ takes 1001 multiplications.
    - $\varphi(\mathbf{x})^T \varphi(\mathbf{y})$ in feature space takes 500,500 multiplications
Kernel Functions

• Definition:
A function \( k : X \times X \rightarrow \mathbb{R} \) is a kernel function if there exists a feature mapping \( \varphi : X \rightarrow \mathbb{R}^n \) such that:
\[
k(x,y) = \varphi(x)^T \varphi(y)
\]

• Theorem:
\( k : X \times X \rightarrow \mathbb{R} \) is a valid kernel \( \iff \) the Gram matrix \( K \) whose elements are given by \( k(x_n,x_m) \) is positive semidefinite for all possible choices of the set \( \{x_n\} \).
Kernel Examples

• Linear kernel: \( K(x, y) = x^T y \)

• Quadratic kernel: \( K(x, y) = (c + x^T y)^2 \)
  – contains constant, linear terms and terms of order two (c > 0).

• Polynomial kernel: \( K(x, y) = (c + x^T y)^M \)
  – contains all terms up to degree \( M \) (c > 0).

• Gaussian kernel: \( K(x, y) = \exp(-\|x - y\|^2 / 2\sigma^2) \)
  – corresponding feature space has infinite dimensionality.
Kernels over Discrete Structures

- **Subsequence Kernels** [Lodhi et al., JMLR 2002]:
  - $\Sigma$ is a finite alphabet (set of symbols).
  - $x, y \in \Sigma^*$ are two sequences of symbols with lengths $|x|$ and $|y|$.
  - $k(x, y)$ is defined as the number of common substrings of length $n$.
  - $k(x, y)$ can be computed in $O(n|x||y|)$ time complexity.

- **Tree Kernels** [Collins and Duffy, NIPS 2001]:
  - $T_1$ and $T_2$ are two trees with $N_1$ and $N_2$ nodes respectively.
  - $k(T_1, T_2)$ is defined as the number of common subtrees.
  - $k(T_1, T_2)$ can be computed in $O(N_1N_2)$ time complexity.
  - in practice, time is linear in the size of the trees.
Reading Assignment

• Chapter 6:
  – Section 6.1 on dual representations for linear regression models.
  – Section 6.2 on techniques for constructing new kernels.