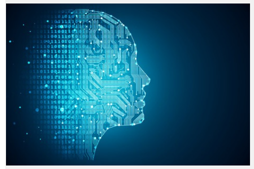


Data Science for Researchers and Scholars

Vasant G. Honavar
Dorothy Foehr Huck and J. Lloyd Huck Chair in Biomedical Data Sciences and Artificial Intelligence
Professor of Data Sciences, Informatics, Computer Science and Engineering, Bioinformatics & Genomics, Public Health Sciences and Neuroscience
Director, Center for Artificial Intelligence Foundations and Scientific Applications
Associate Director, Institute for Computational and Data Sciences
Pennsylvania State University

vhonavar@psu.edu
http://faculty.ist.psu.edu/vhonavar
http://ailab.ist.psu.edu

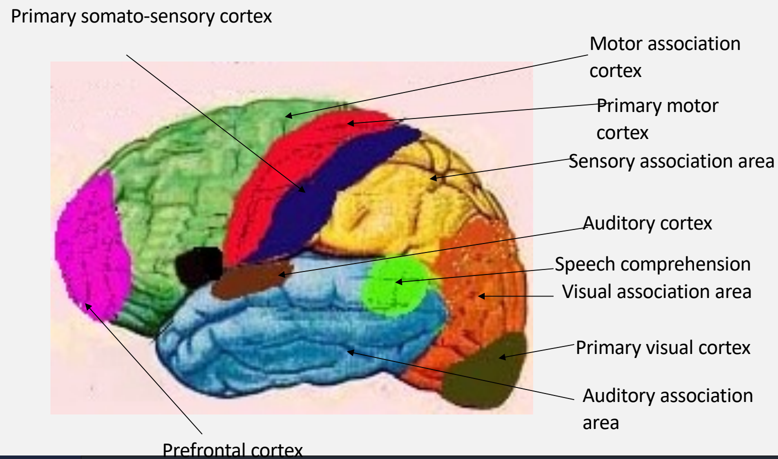
Linear Classifiers: Simple Neural Networks

- Background
- Threshold logic functions
- Connection to logic
- Connection to geometry
- Learning threshold functions – perceptron algorithm
- Perceptron convergence theorem
- Multi-category extensions
- Alternative loss functions and algorithms

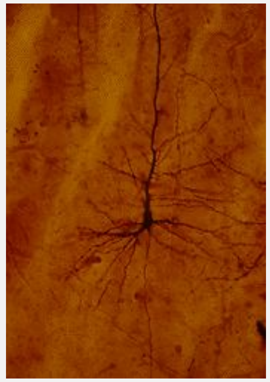
Background – Brains and Computers

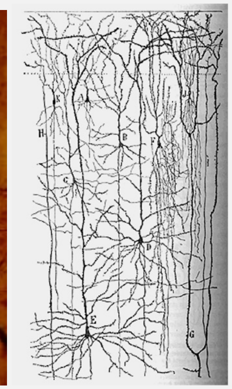
- Brain consists of 10^{11} neurons, each connected to ~10^{4} other neurons
- Each neuron is slow
  - 1 millisecond to respond to a stimulus
- Brain is astonishingly fast at perceptual tasks
- Brain processes and learns from multiple sources of sensory information (visual, tactile, auditory…)
- Brain is massively parallel, shallowly serial, modular and roughly hierarchical with recurrent and lateral connectivity within and between modules
- Turing: Thinking can be modeled by computation
- If thinking can be modeled by – computation
  - how and what are the algorithms that underly thinking or
  - what do brains compute

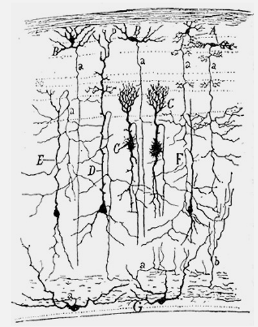
Brain and information processing



Neural Networks

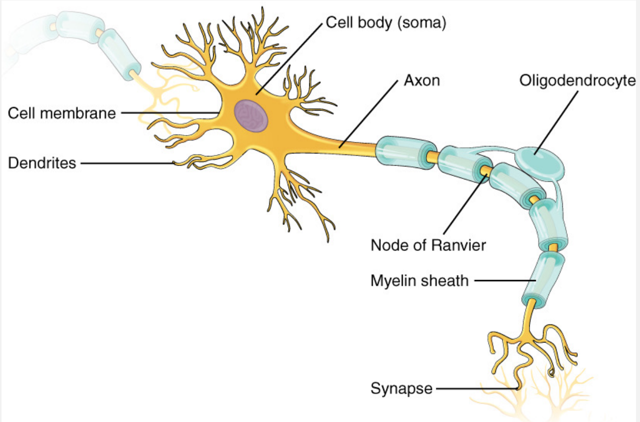




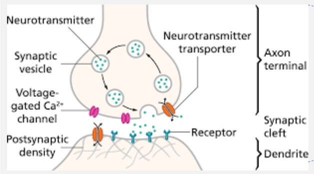


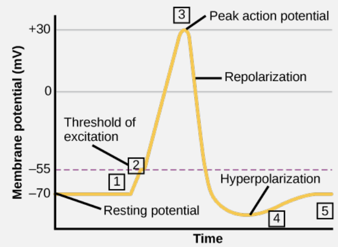
Ramon Cajal, 1900

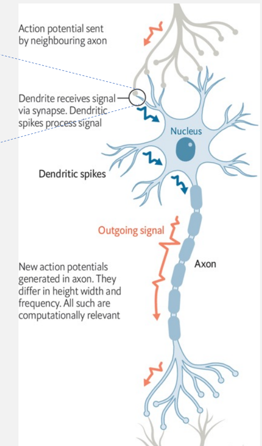
Neurons and Computation



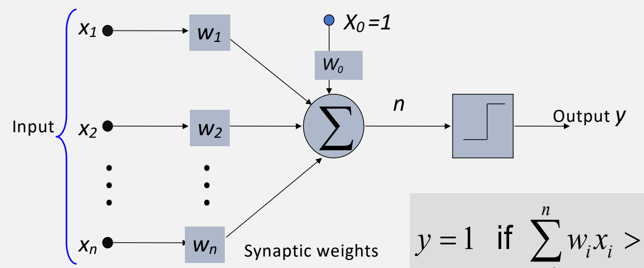
Neural information processing







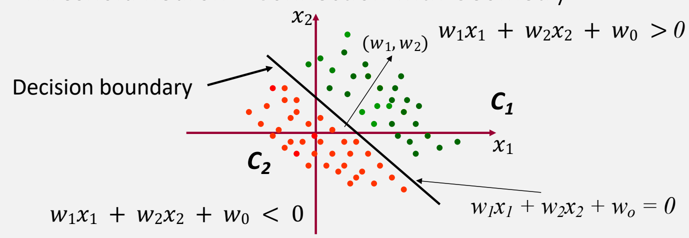
McCulloch-Pitts computational model of a neuron



$$y = 1 \ \text{ if } \sum_{i=0}^{n} w_i x_i > 0$$

$$y = -1 \ \text{ otherwise}$$

Threshold neuron – Connection with Geometry



$\sum_{i=1}^{n} w_i x_i + w_0 = 0$ describes a hyperplane which divides the instance space $\Re^n$ into two half–spaces

$\chi_+ = \left\{ \mathbf{X}_p \in \Re^n \middle| \mathbf{W} \bullet \mathbf{X}_p + w_0 > 0 \right\}$ and $\chi_- = \left\{ \mathbf{X}_p \in \Re^n \middle| \mathbf{W} \bullet \mathbf{X}_p + w_0 < 0 \right\}$

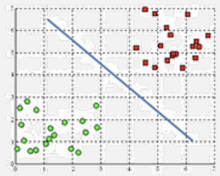
McCulloch-Pitts Neuron or Threshold Neuron

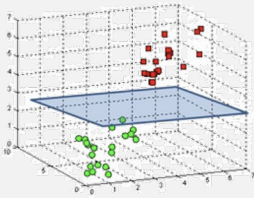
$$
\begin{aligned}
y &= sign\left(\mathrm{W} \bullet \mathrm{X} + w_0\right) \\
&= sign\left(\sum_{i=0}^{n} w_i x_i\right) \\
&= sign\left(\mathrm{W}^T \mathrm{X} + w_0\right)
\end{aligned}
$$

$$
\mathrm{X} = \begin{bmatrix} x_1 \\ x_2 \\ \\ x_n \end{bmatrix} \qquad \mathrm{W} = \begin{bmatrix} w_1 \\ w_2 \\ \\ w_n \end{bmatrix}
$$

$$
\begin{aligned}
sign\left(v\right) &= 1 \text{ if } v > 0 \\
&= 0 \text{ otherwise}
\end{aligned}
$$

McCulloch-Pitts Neuron - Connection to geometry





- A perceptron with 3 weights $[w_0, w_1, w_2]$implements a line in 2-D
- A perceptron with 4 weights $[w_0, w_1, w_2, w_4]$ implements a plane in 3-D
- A perceptron with $n+1$ weights $[w_0, \cdots, w_n]$ implements an $(n-1)$ dimensional hyperplane in n-D
  - Dividing the n-D space into two half spaces

Perceptron – Connection with Geometry

- Data live in $\Re^n$ or $\Re^{n+1}$ if we add a dummy input of $x_0 = 1$
- Weights that define hyperplanes live in $\Re^{n+1}$
- A particular choice of weights defines a hyperplane given by

$$\sum_{i=1}^{n} w_i \, x_i + w_0 = 0$$

or

$$\sum_{i=0}^{n} w_i \, x_i = 0$$

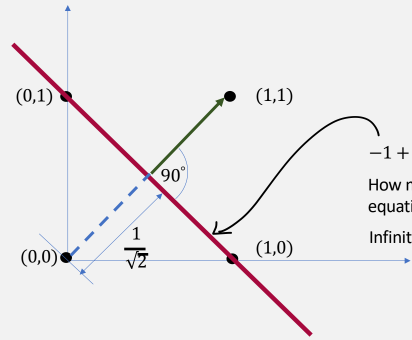
or

$$\mathbf{w} \cdot \mathbf{x} = 0$$

- The orientation of the hyperplane is specified by its normal vector$[w_1 \cdots w_n]^T$
- The distance of the hyperplane from a given data point $\mathbf{x}_p$ is given by

$$\frac{|\mathbf{w}\cdot\mathbf{x}_p|}{\sqrt{w_1^2+\cdots+w_n^2}} = \frac{|w_0+w_1x_{1p}+\cdots+w_nx_{np}|}{\sqrt{w_1^2+\cdots+w_n^2}}$$

Example



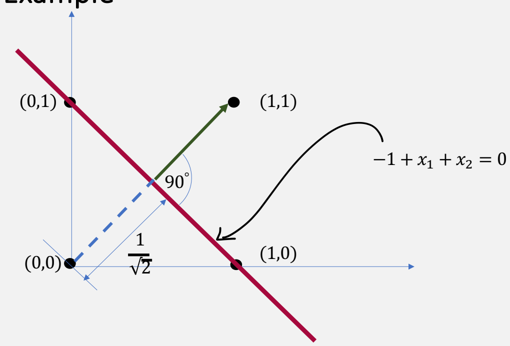
$-1 + x_1 + x_2 = 0$

How many other equivalent equations define the same hyperplane?

Infinite number of them!

-1

Example



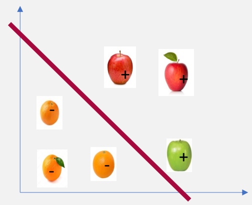
-1

Which side of the hyperplane does the point (2,1) lie?

Check the sign of $w_0 + \sum_{i=1}^{2} w_i x_i = -1 + 2 + 1 = 2$
The sign is positive, so on the "positive side" pointed by the direction of the positive normal

Perceptron as a pattern classifier

- The threshold neuron, that implements the "right" hyperplane, can be used to classify a set of data samples into one of two classes C_1, C_2 *e.g.,* apples and oranges when they are represented as points in a suitable feature space
- If the output of the neuron for input pattern X_p is +1 then X_p is assigned to class C_1
- If the output is -1 then the pattern X_p is assigned to C_2



Threshold neuron – Connection with Logic

- Suppose the input space is $\{0,1\}^n$
- Then threshold neuron computes a Boolean function $f : \{0,1\}^n \rightarrow \{-1,1\}$

<u>Example</u>

Let $w_0 = -1.5; w_1 = w_2 = 1$

- In this case, if we interpret 1 as TRUE and -1 as FALSE, the threshold neuron implements the logical AND function

x_1	x_2	$h(X) = \mathbf{w} \cdot \mathbf{x}$	y
0	0	-1.5	-1
0	1	-0.5	-1
1	0	-0.5	-1
1	1	0.5	1

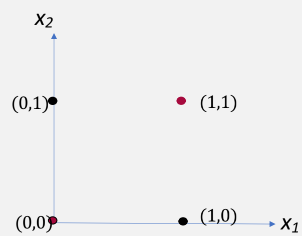
Threshold neuron – Connection with Logic

- A threshold neuron with the appropriate choice of weights can implement Boolean AND, OR, and NOT function
- Theorem: For any arbitrary Boolean function *f*, there exists a network of threshold neurons that can implement *f*.
- Theorem: Any arbitrary finite state automaton can be realized using threshold neurons and *delay* units
- Networks of threshold neurons, given access to unbounded memory, can compute any Turing-computable function
- Corollary: Brains if given access to enough working memory, can compute any computable function

Threshold neuron: Connection with Logic

Theorem: There exist Boolean functions that cannot be implemented by a single threshold neuron.

Example: Exclusive OR



x_1	x_2	y
0	0	-1
0	1	1
1	0	1
1	1	-1

Why?
A hyperplane separating the red points from the black points does not exist!

Threshold neuron – Connection with Logic

- Definition: A function that can be computed by a single threshold neuron is called a threshold function
- Of the 16 2-input Boolean functions, 14 are Boolean threshold functions
- As *n* increases, the number of Boolean threshold functions becomes an increasingly small fraction of the total number of *n*-input Boolean functions

Terminology and Notation

- Synonyms: Threshold function, Linearly separable function, linear discriminant function
- Synonyms: Threshold neuron, McCulloch-Pitts neuron, Perceptron, Threshold Logic Unit (TLU)
- We often include w_0 as one of the components of **w** and incorporate x_0 as the corresponding component of **x** with the understanding that $x_0 = 1$.
- Then $y = 1$ if $\mathbf{w} \cdot \mathbf{x} > 0$ and $y = -1$ otherwise.

Learning Threshold functions

A training example E_k is an ordered pair (X_k, d_k) where

$$\mathrm{X}_k = \left[x_{0k}\ x_{1k}\ \ldots.\ x_{nk}\right]^T$$

is an $(n+1)$ dimensional input sample, $d_k = f(\mathbf{X}_k) \in \{-1, 1\}$ is the desired output of the classifier and f is an unknown target function to be learned.

A training set E is simply a multi-set of examples.

Learning Threshold functions

$$S^{+} = \{ X_k | (X_k, d_k) \in E \text{ and } d_k = 1 \}$$
$$S^{-} = \{ X_k | (X_k, d_k) \in E \text{ and } d_k = -1 \}$$

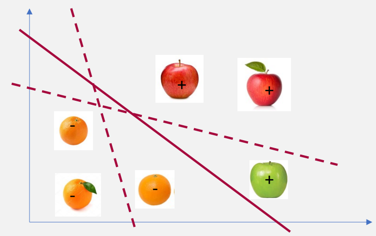
We say that a training set E is linearly separable if and only if

$$\exists W^{*} \text{ such that } \forall X_p \in S^{+}, W^{*} \bullet X_p > 0$$
$$\text{and } \forall X_p \in S^{-}, W^{*} \bullet X_p < 0$$

Learning Task: Given a linearly separable training set *E,* find a solution

$$W^{*} \text{ such that } \forall X_p \in S^{+}, W^{*} \bullet X_p > 0 \text{ and } \forall X_p \in S^{-}, W^{*} \bullet X_p < 0$$

Learning to classify = finding separating hyperplane



Rosenblatt's Perceptron Learning Algorithm

Initialize $\mathrm{W} = [0\ 0 \ldots\ldots 0]^T$ Set learning rate $\eta > 0$

Repeat until a complete pass through E results in no weight updates

For each training example $E_k \in E$

$$\{ \quad y_k \leftarrow sign\,(\mathbf{W} \bullet \mathbf{X}_k)$$

$$\mathrm{W} \leftarrow \mathrm{W} + \eta(d_k - y_k)\mathrm{X}_k \quad \}$$

$$\mathrm{W}^* \leftarrow \mathrm{W}; \qquad \text{Return}(\mathrm{W}^*)$$

Perceptron learning algorithm – Example

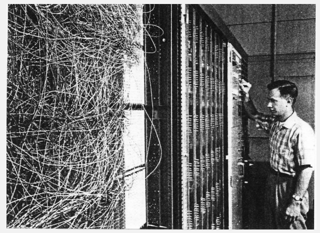
Let $S^+ = \{(1,1,1),(1,1,-1),(1,0,-1)\}$
$S^- = \{(1,-1,-1),(1,-1,1),(1,0,1)\}$

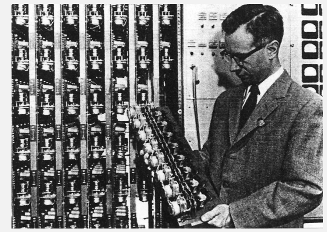
$\mathbf{W} = (0\ 0\ 0)$

$\eta = \frac{1}{2}$

X_k	d_k	W	$W.X_k$	y_k	Update?	Updated W
(1, 1, 1)	1	(0, 0, 0)	0	-1	Yes	(1, 1, 1)
(1, 1, -1)	1	(1, 1, 1)	1	1	No	(1, 1, 1)
(1,0, -1)	1	(1, 1, 1)	0	-1	Yes	(2, 1, 0)
(1, -1, -1)	-1	(2, 1, 0)	1	1	Yes	(1, 2, 1)
(1,-1, 1)	-1	(1, 2, 1)	0	-1	No	(1, 2, 1)
(1,0, 1)	-1	(1, 2, 1)	2	1	Yes	(0, 2, 0)
(1, 1, 1)	1	(0, 2, 0)	2	1	No	(0, 2, 0)

Perceptron circa 1957



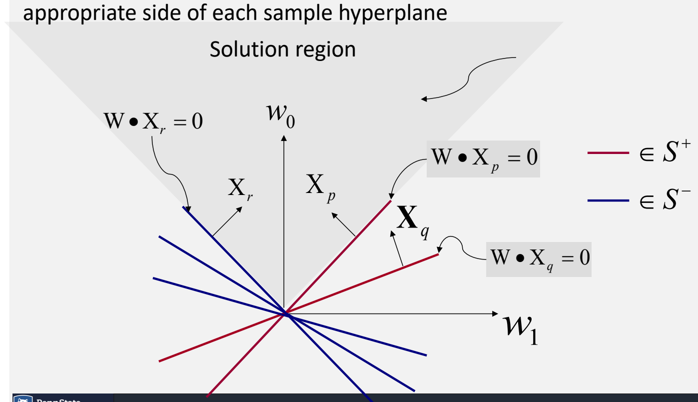


Understanding Weight Updates

- During learning, training data are fixed
- What is being updated are the weights
- Consider the weight space defined by the coordinates of the weight vector
- Points in this space correspond to different choices of the weights
- Just as in the data space, weights defined hyperplanes, in the weight space, training data samples define (fixed) hyperplanes.

Understanding Weight Updates

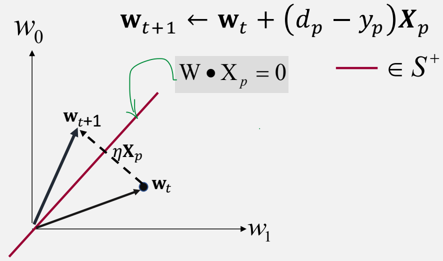
Goal: Find a point in the weight space that lies on the appropriate side of each sample hyperplane



Understanding Weight Updates

- Goal: Find a point in the weight space that lies on the appropriate side of each sample hyperplane
- If the current weight vector is on the wrong side of a pattern hyperplane, the most efficient way to go to the right side is to move along the direction of the normal to the hyperplane

$$\mathbf{w}_{t+1} \leftarrow \mathbf{w}_t + (d_p - y_p)\boldsymbol{X}_p$$



Perceptron Convergence Theorem (Novikoff)

Theorem Let $E = \{(\mathbf{X}_k, d_k)\}$ be a training set where $\mathbf{X}_k \in \{1\} \times \Re^n$ and $d_k \in \{-1, 1\}$

Let $S^+ = \{\mathbf{X}_k | (\mathbf{X}_k, d_k) \in E \;\&\; d_k = 1\}$ and $S^- = \{\mathbf{X}_k | (\mathbf{X}_k, d_k) \in E \;\&\; d_k = -1\}$

The perceptron algorithm is guaranteed to terminate after a bounded number t of weight updates with a weight vector $\mathbf{W}^*$ such that $\forall\, \mathbf{X}_k \in S^+, \mathbf{W}^* \bullet \mathbf{X}_k \geq \delta$ and $\forall\, \mathbf{X}_k \in S^-, \mathbf{W}^* \bullet \mathbf{X}_k \leq -\delta$ for *some* $\delta > 0,$ whenever such $\mathbf{W}^* \in \Re^{n+1}$ and $\delta > 0$ exist -- that is, *E is linearly separable.* The bound on the number *t* of weight updates is given by

$$t \leq \left(\frac{\|\mathbf{W}^*\| L}{\delta} \right)^2 \text{ where } L = \max_{\mathbf{X}_k \in S} \|\mathbf{X}_k\| \text{ and } S = S^+ \cup S^-$$

Notes on the Perceptron Convergence Theorem

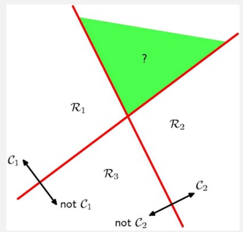
- The bound on the number of weight updates does not depend on the learning rate
- The bound is not useful in determining when to stop the algorithm because it depends on the norm of the unknown weight vector and delta
- The convergence theorem offers no guarantees when the training data set is not linearly separable

- It is easy to prove that the perceptron algorithm is robust with respect to fluctuations in the learning rate

$$0 < \eta_{\text{min}} \le \eta_t \le \eta_{\text{max}} < \infty$$

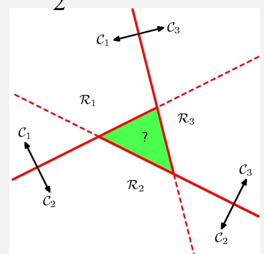
Multi-category classifiers

$K-1$ binary classifiers



One-versus-rest

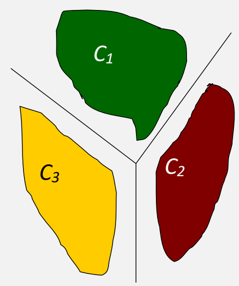
$\frac{K(K-1)}{2}$ binary classifiers



One-versus-one

Problem: Green region has ambiguous class membership

Multi-category classifiers



Define K linear functions of the form:

$$y_k(\mathbf{X}) = \mathbf{W}_k{}^T\mathbf{X} + w_{k0}$$

$$h(\mathbf{X}) = \arg\max_k y_k(\mathbf{X})$$
$$= \arg\max_k \left(\mathbf{W}_k{}^T\mathbf{X} + w_{k0}\right)$$

Decision surface between class C_k and C_j

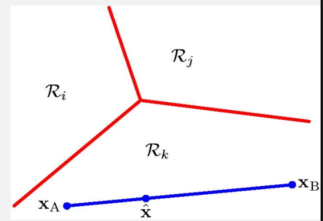
$$\left(\mathbf{W}_k - \mathbf{W}_j\right)^T\mathbf{X} + \left(w_{k0} - w_{j0}\right) = 0$$

Linear separator for K classes

- Decision regions defined by

$$(\mathbf{W}_k - \mathbf{W}_j)^T \mathbf{X} + (w_{k0} - w_{j0}) = 0$$

are singly connected and convex



For any points $\mathbf{X}_A, \mathbf{X}_B \in R_k$,
any $\hat{\mathbf{X}}$ that lies on the line connecting $\mathbf{X}_A$ and $\mathbf{X}_B$
$\hat{\mathbf{X}} = \lambda \mathbf{X}_A + (1-\lambda)\mathbf{X}_B$ where $0 \le \lambda \le 1$
also lies in R_k

Winner-Take-All Networks

$y_{ip} = 1$ iff $\mathbf{W}_i \bullet \mathbf{X}_p > \mathbf{W}_j \bullet \mathbf{X}_p \quad \forall j \neq i$

$y_{ip} = 0$ otherwise   Note: $\mathbf{W}_j$ are augmented weight vectors

$$\mathbf{W}_1 = [1\ \text{-}1\ \text{-}1]^T, \mathbf{W}_2 = [1\ 1\ 1]^T, \mathbf{W}_3 = [2\ 0\ 0]^T$$

			$W_1.X_p$	$W_2.X_p$	$W_3.X_p$	y_1	y_2	y_3
1	-1	-1	3	-1	2	1	0	0
1	-1	+1	1	1	2	0	0	1
1	+1	-1	1	1	2	0	0	1
1	+1	+1	-1	3	2	0	1	0

What does neuron 3 compute?

Linear separability of multiple classes

Let $S_1, S_2, S_3 \ldots S_M$ be multisets of instances

Let $C_1, C_2, C_3 \ldots C_M$ be disjoint classes

$\forall i \;\; S_i \subseteq C_i$

$\forall i \neq j \;\; C_i \bigcap C_j = \varnothing$

We say that the sets $S_1, S_2, S_3 \ldots S_M$ are linearly separable iff $\exists$ weight vectors $\mathrm{W}_1^*, \mathrm{W}_2^*, .. \mathrm{W}_M^*$ such that

$\forall i \;\; \left\{ \forall \mathrm{X}_p \in S_i, \left(\mathrm{W}_i^* \bullet \mathrm{X}_p > \mathrm{W}_j^* \bullet \mathrm{X}_p \right) \forall j \neq i \right\}$

Training WTA Classifiers

$d_{kp} = 1$ iff $\mathbf{X}_p \in C_k$; $d_{kp} = 0$ otherwise

$y_{kp} = 1$ iff $\mathbf{W}_k \bullet \mathbf{X}_p > \mathbf{W}_j \bullet \mathbf{X}_p \quad \forall k \neq j$

Suppose $d_{kp} = 1, y_{jp} = 1$ and $y_{kp} = 0$

$\mathbf{W}_k \leftarrow \mathbf{W}_k + \eta \mathbf{X}_p; \mathbf{W}_j \leftarrow \mathbf{W}_j - \eta \mathbf{X}_p;$

All other weights are left unchanged.

Suppose $d_{kp} = 1, y_{jp} = 0$ and $y_{kp} = 1.$

The weights are unchanged.

Suppose $d_{kp} = 1, \forall j \; y_{jp} = 0$ (there was a tie)

$\mathbf{W}_k \leftarrow \mathbf{W}_k + \eta \mathbf{X}_p$

All other weights are left unchanged.

WTA Convergence Theorem

Given a linearly separable training set, the WTA learning algorithm is guaranteed to converge to a solution within a finite number of weight updates.

Digression: Minimizing functions

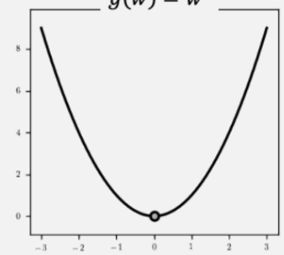
Minima of a function

- In many applications, machine learning included, we are often interested in minimizing a function of many variables.
- For a function $g(\mathbf{w})$ of N variables $w_1 \cdots w_N$, this problem is formally phrased as

$$\underset{\mathbf{w}}{\text{minimize}}\; g(\mathbf{w})$$

- That is, examine the value of $g(\mathbf{w})$ over all possible values of $\mathbf{w}$ in the domain of $g(\mathbf{w})$ and pick one or more where the value of $g(\mathbf{w})$ is minimum (over the range of $g(\mathbf{w})$.

$g(w) = w^2$



- What is the smallest value of $g(w) = w^2$?
- What is the value of w at which this occurs?

Minima of a function

- Obviously, the minimum is smaller than any other value of the function.
- Specifically the smallest value - the global minimum of this function - seemingly occurs close to $w^\star = 0$
- Formally a point $w^\star$ gives the smallest point on the function if

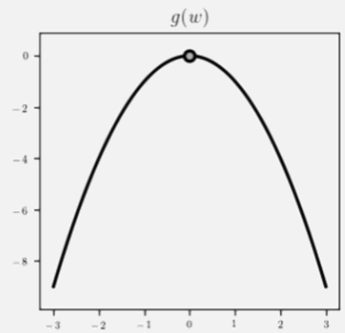
$$g(w^\star) \leq g(w) \quad \text{for all } w.$$

- This is called the ***zero-order definition of a global minimum*** of a function.

Minima of a function

- Suppose we multiply the quadratic function in the previous example by -1.
- The function flips upside down - now its global minima lie at $w^{\star} = \pm\infty$
- Now the point $w^{\star} = 0$ that used to be a global minimum is now global maximum - i.e., where the value of the function is the largest, i.e.,

$$g(w^{\star}) \geq g(w) \quad \text{for all } w.$$



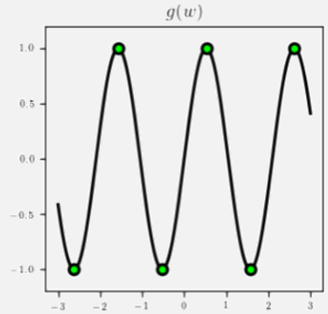
Minima of a function

- These concepts of *minima* and *maxima* of a function are always related to each via multiplication by -1.
- That is, any point that is a minima of a function g is a maxima of the function $-g$, and vice-versa.

$$\underset{\mathbf{w}}{\text{maximize}} \quad g(\mathbf{w}) = -\underset{\mathbf{w}}{\text{minimize}} \quad g(\mathbf{w}).$$

Example

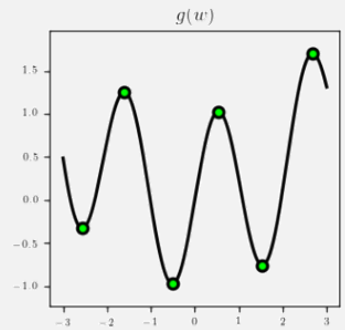
- Let us look at the sinusoid function $g(w) = \sin(2w)$
- Over the range we have plotted the function - that there are three global minima and three global maxima (marked by green dots).



- Technically speaking, this function has an infinite number of global maxima and minima
- Why?

Example: Minima and maxima of the sum of a sinusoid and a quadratic

- Let's look at a weighted sum of the previous two examples, the function $g(w) = \sin(3w) + 0.1w^2$ over a region of its input space.



- We have a global minimum around $w^{\star} = -0.5$ and a global maximum around $w^{\star} = 2.7$
- The point around $w^{\star} = 0.8$ is a local maximum.
- The point around $w^{\star} = 1.5$ is a local minimum
- Can you identify other local minima/maxima?

The zero order condition for optimality

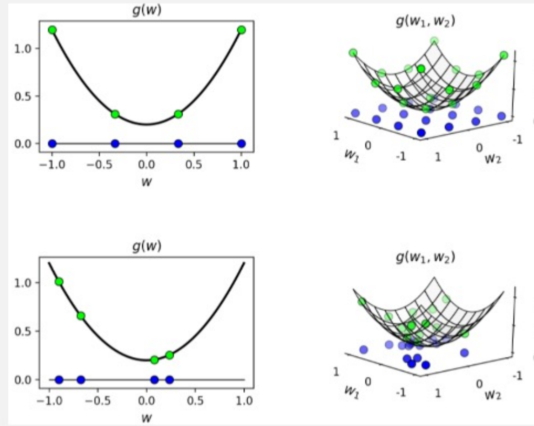
A point $\mathbf{w}^\star$ is

- a global minimum of $g(\mathbf{w})$ if and only if $g(\mathbf{w}^\star) \leq g(\mathbf{w})$ for all $\mathbf{w}$
- a global maximum of $g(\mathbf{w})$ if and only if $g(\mathbf{w}^\star) \geq g(\mathbf{w})$ for all $\mathbf{w}$
- a local minimum of $g(\mathbf{w})$ if and only if $g(\mathbf{w}^\star) \leq g(\mathbf{w})$ for all $\mathbf{w}$ near $\mathbf{w}^\star$
- a local maximum of $g(\mathbf{w})$ if and only if $g(\mathbf{w}^\star) \geq g(\mathbf{w})$ for all $\mathbf{w}$ near $\mathbf{w}^\star$

- Why zero order? Because it depends only on the function $g(\mathbf{w})$ and nothing else.
- Higher order definitions refer to the values of first, second … order derivatives of $g(\mathbf{w})$

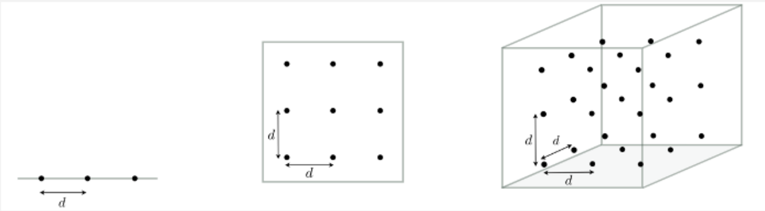
Finding global maxima / minima: A naïve approach

- Evaluate the function at a large number of points
- Among them, choose the input at which the function is the lowest as the approximate global minimum
- How can we choose the points at which to evaluate the function?
  - Uniformly
  - Randomly
- While this naïve approach works for functions of few variables, it fails for functions of more than 2 or 3 variables
- Why? The number of points at which the function needs to be evaluated grows exponentially with the number of variables


$g(w)$
1.0
0.5
0.0
−1.0
−0.5
0.0
0.5
1.0
w
$g(w_1, w_2)$
2
1
0
1
0
-1
1
0
-1
w_1
w_2

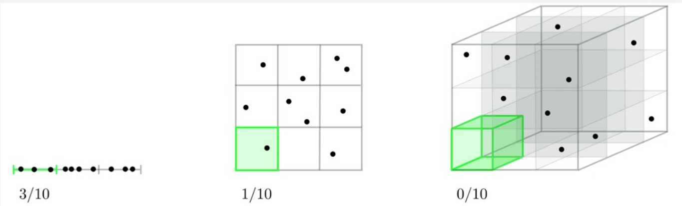
Curse of dimensionality

- While the naïve approach works for functions of few variables, it fails for functions of more than 2 or 3 variables
- Why? If sampled uniformly, the number of points at which the function needs to be evaluated grows exponentially with the number of variables



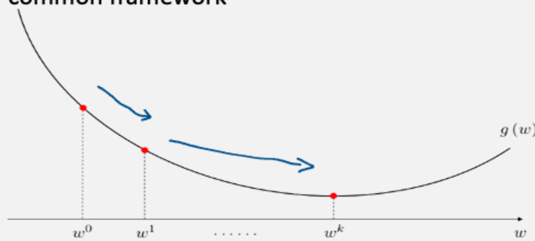
Curse of dimensionality

- The problem does not go away if we sample points randomly
- As the dimensionality n increases, smaller the fraction of samples in a n-dimensional volume

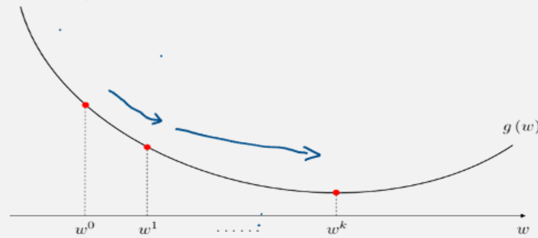


Local optimization methods

- *Local optimization methods* work by evaluating the function at a single point and sequentially updating it until an approximate minimum is reached
- Local optimization methods are by far the most popular optimization methods in machine learning
- While details vary, all local optimization methods fit into a common framework



Local optimization methods



- Starting with an initial point $\mathbf{w}^0$, local optimization methods iteratively update the current point such that:

  $$g\left(\mathbf{w}^0\right) > g\left(\mathbf{w}^1\right) > \cdots > g\left(\mathbf{w}^K\right)$$

- Unlike global optimization methods, local optimization scales gracefully with the number of dimensions

Local optimization methods

- How do we find $\mathbf{d}^{k-1}$?
- A multitude of methods exist – they differ from each other in how the descent direction is found

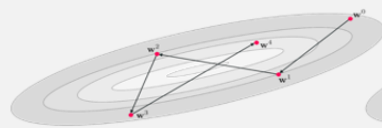
$$\mathbf{w}^k = \mathbf{w}^{k-1} + \mathbf{d}^{k-1}$$

$$\left\|\mathbf{w}^k - \mathbf{w}^{k-1}\right\|_2 = \left\| \left(\mathbf{w}^{k-1} + \mathbf{d}^{k-1}\right) - \mathbf{w}^{k-1}\right\|_2 = \left\|\mathbf{d}^{k-1}\right\|_2.$$

- $||A||_2$ is the 2nd norm of the vector $A = \begin{bmatrix} a_1 \\ \vdots \\ a_N \end{bmatrix}$
- $||A||_2 = \sqrt[2]{\sum_{i=1}^{N} a_i^2}$
- Hence, the distance moved as a result of update is equal to the length of the vector defining the descent direction

Local optimization methods

- Depending on how our descent direction vectors are obtained, we may or may not have control over their length
  - All we ask is that they point in the right direction, 'down hill'
- Even if they point in the right direction - towards points the function value is lower - that their *length* could be problematic
  - If the length is too large, we may overshoot the minimum
  - If the length is too small, we may take forever to reach the minimum



Local optimization methods

- Most local optimization methods use a *step size parameter* (called a *learning rate* parameter in ML)

  $$\mathbf{w}^k = \mathbf{w}^{k-1} + \alpha \mathbf{d}^{k-1}$$

  - In the simplest case, the step size is fixed
  - In the most general case, it can vary from step to step
- Now,

  $$\left\|\mathbf{w}^k - \mathbf{w}^{k-1}\right\|_2 = \left\| \left(\mathbf{w}^{k-1} + \alpha \mathbf{d}^{k-1}\right) - \mathbf{w}^{k-1}\right\|_2 = \alpha \left\|\mathbf{d}^{k-1}\right\|_2.$$

- We can adjust the step length by choosing α

Calculus review

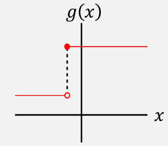
- A function of a real variable $f(x)$ is differentiable at a point a if $\lim_{\epsilon \to 0} \frac{f(a+\epsilon)-f(a)}{\epsilon}$ exists
- The limit is called the derivative of $f(x)$ at $x = a$
- The the derivative of $f(x)$ is denoted by $\frac{df}{dx}$
- If f is differentiable at a, then f must be continuous at a
  - $g(x)$ is not continuous, not differentiable
  - $f(x)$ is continuous and differentiable
  - $h(x)$ is continuous but not differentiable at $x = 0$

$$\frac{d(u+v)}{dx} = \frac{du}{dx} + \frac{dv}{dx}$$

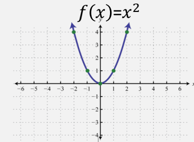
$$\frac{d(uv)}{dx} = u\frac{dv}{dx} + v\frac{du}{dx}$$

$$\frac{d\left(\frac{u}{v}\right)}{dx} = \frac{v\left(\frac{du}{dx}\right) - u\left(\frac{dv}{dx}\right)}{v^2}$$

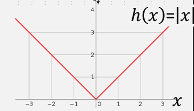
$g(x)$



$f(x)=x^2$



$h(x)=|x|$



Calculus review

$$f(x) = x^2 + 3x$$

$$\frac{d(u+v)}{dx} = \frac{du}{dx} + \frac{dv}{dx}$$

$$\frac{df}{dx} =$$

Examples

$$f(x) = x(x+3)$$

$$\frac{d(uv)}{dx} = u\frac{dv}{dx} + v\frac{du}{dx}$$

$$\frac{df}{dx} =$$

Examples

$$f(x) = \frac{x(x+3)}{x^2}$$

$$\frac{d\left(\frac{u}{v}\right)}{dx} = \frac{v\left(\frac{du}{dx}\right) - u\left(\frac{dv}{dx}\right)}{v^2}$$

$$\frac{df}{dx} =$$

Partial derivatives and chain rule

Let $f(\mathbf{X}) = f(x_{0,} x_1, x_2, \ldots x_n)$

$\dfrac{\partial f}{\partial x_i}$ is obtained by treating all $x_i \mid i \neq j$ as constant.

Chain rule

Let $z = \varphi(u_1 \ldots u_m)$

Let $u_i = f_i(x_{0,} x_1 \ldots x_n)$

Then $\forall k \quad \dfrac{\partial z}{\partial x_k} = \sum_{i=1}^{m} \left(\dfrac{\partial z}{\partial u_i}\right)\left(\dfrac{\partial u_i}{\partial x_k}\right)$

- $z = f(u, v) = u^2 + 2v$
- $u = f_1(x, y) = 2x + y$
- $v = f_2(x, y) = x^2 + y$
- $\frac{\partial z}{\partial x} = \frac{\partial z}{\partial u}\frac{\partial u}{\partial x} + \frac{\partial z}{\partial v}\frac{\partial v}{\partial x}$

$\frac{dz}{dx} = 2u(2) + 2(2x)$

$\frac{dz}{dx} = 4(2x + y) + 4x$

$\frac{dz}{dx} = 4(3x + y)$

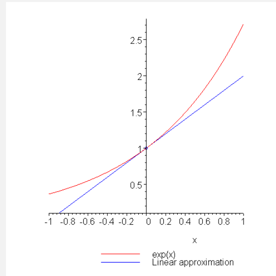
Taylor series approximation

Suppose $f(x)$ is differentiable (i.e., its derivatives $\frac{df}{dx}, \frac{d^2f}{dx^2}, \cdots \frac{d^nf}{dx^n}$ exist) and $f(x)$ is continuous in the neighborhood of x_0. Then the *Taylor Series* expansion of a function $f(x)$ around $x = x_0$ is given by:

$$f(x) = f(x)\mid_{x=x_0} + \frac{df}{dx}\mid_{x=x_0}(x-x_0) + \frac{1}{2}\frac{d^2f}{dx^2}\mid_{x=x_0}(x-x_0)^2 + \ldots + \frac{1}{n!}\frac{d^nf}{dx^n}(x-x_0)$$

- Suppose $f(x) = x^2 + 1$
- $\frac{df}{dx} = 2x$
- Suppose we want to approximate f(x) at $x = 1.01$ given $f(1) = 2$
- $f(1.01) \approx f(1) + 2(1.01 - 1) \approx 2.02$
- $f(x + \Delta x) \approx f(x) + \left.\frac{df}{dx}\right|_x \Delta x$

Locally linear approximation



- Suppose $f(x) = x^2 + 1$
- $\frac{df}{dx} = 2x$
- Suppose we want to approximate f(x) at $x = 1.01$ given $f(1) = 2$
- $f(1.01) \approx f(1) + 2(1.01 - 1) \approx 2.02$
- $f(x + \Delta x) \approx f(x) + \left.\frac{df}{dx}\right|_x \Delta x$

Taylor series approximation of multi-variable functions

The concepts introduced above extend quite naturally to the case of multivariate functions (i.e., functions of several variables). Consider a multivariate function $f(\mathbf{X}) = f(x_0, \ldots, x_n))$. Now we have *partial derivatives* that represent the rate of change of $f(\mathbf{X})$ with respect to each variable x_i. A partial derivative with respect to x_i is computed by taking the derivative of $f(x_0, \ldots x_n)$ by treating $\forall j \neq i$, x_j as though it were a constant.

Taylor Series can be used to approximate a function of several variables in a neighborhood where the function is continuous and differentiable. For example, the Taylor Series expansion for the function $\phi(x_1, x_2)$ around $\mathbf{X_0} = (x_{01}, x_{02})$ is given by:

$$\phi(\mathbf{X_0}) + \frac{\partial \phi}{\partial x_1}|_{\mathbf{X}=\mathbf{X}_0} (x_1 - x_{01}) + \frac{\partial \phi}{\partial x_2}|_{\mathbf{X}=\mathbf{X}_0} (x_2 - x_{02}) + \frac{1}{2}\frac{\partial^2 \phi}{\partial x_1^2}|_{\mathbf{X}=\mathbf{X}_0} (x_1 - x_{01})^2 + \frac{1}{2}\frac{\partial^2 \phi}{\partial x_2^2}|_{\mathbf{X}=\mathbf{X}_0} (x_2 - x_{02})^2 + \ldots$$

Taylor series approximation of multi-variable functions

$$\mathbf{x} = \begin{bmatrix} x_1 \\ \vdots \\ x_N \end{bmatrix}$$

$$f(\mathbf{x_0} + \mathbf{a}) \approx f(\mathbf{x_0}) + \nabla^T \mathbf{x}|_{\mathbf{x}=\mathbf{x_0}}\ \mathbf{a}$$

$$\nabla \mathbf{x} = \begin{bmatrix} \frac{\partial f}{\partial x_1} \\ \vdots \\ \frac{\partial f}{\partial x_N} \end{bmatrix}$$

Taylor Series Approximation of Multivariate Functions

Let $f(\mathbf{X}) = f(x_{0,} x_1, x_2, \ldots\ldots x_n)$ be differentiable and continuous at $\mathbf{X}_0 = (x_{00,} x_{10}, x_{20}, \ldots\ldots x_{n0})$

Then

$$f(\mathbf{X}) \approx f(\mathbf{X}_0) + \sum_{i=0}^{n} \left.\left(\frac{\partial f}{\partial x}\right)\right|_{\mathbf{X}=\mathbf{X}_0} (x_i - x_{i0})$$

Minimizing functions using Gradient descent

- Suppose we want to minimize $f(z)$ with respect to z
- Suppose we start at $z = z_0$
- Suppose we want to move to $z = z_1$ such that $f(z_1) < f(z_0)$
- Change in z, $\Delta z = z_1 - z_0$
- Change in f, $\Delta f = f(z_1) - f(z_0)$
- Gradient of f at z_0 = $\left.\frac{df}{dz}\right|_{z_0} \approx \frac{\Delta f}{\Delta z}$

$$\Delta f = f(z_1) - f(z_0) = \frac{\partial f}{\partial z}|_{z=z_0}\Delta z$$

- We want $\Delta f < 0$ (f decreases as we move from $z_0$to z_1)
- We should choose. $\Delta z = -\eta\frac{\partial f}{\partial z}|_{z=z_0}$ where $\eta > 0$
- $\Delta f = -\eta\left(\left.\frac{df}{dz}\right|_{z_0}\right)^2$ is never positive (as desired)
- Hence, we must update z in the direction of the negative gradient of f

Minimizing / Maximizing Multivariate Functions

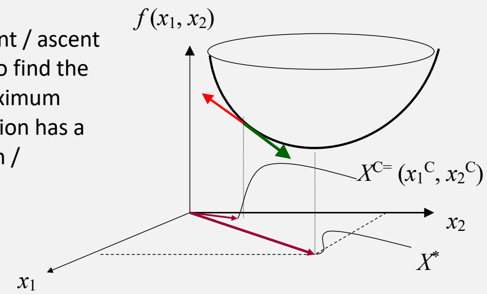
To find $\mathbf{X}^*$ that minimizes $f(\mathbf{X})$, we change current guess $\mathbf{X}^C$ in the direction of the negative gradient of $f(\mathbf{X})$ evaluated at $\mathbf{X}^C$

$$\mathbf{X}^C \leftarrow \mathbf{X}^C - \eta \left(\frac{\partial f}{\partial x_0}, \frac{\partial f}{\partial x_1} \ldots\ldots\ldots\ldots \frac{\partial f}{\partial x_n} \right)\Bigg|_{\mathbf{X}=\mathbf{X}^C} \quad \text{(why?)}$$

for small (ideally infinitesimally small)

Minimizing / Maximizing Functions

Gradient descent / ascent is guaranteed to find the minimum / maximum when the function has a single minimum / maximum



- When does gradient descent stop?
- Technically (when α is chosen well) the algorithm will *halt near stationary points of a function, typically minima or saddle points*.
- How do we know this? By the very form of the gradient descent step itself.
- Say the step

$$\mathbf{w}^{k} = \mathbf{w}^{k-1} - \alpha \nabla g\left(\mathbf{w}^{k-1}\right)$$

does not move from the prior point $\mathbf{w}^{k-1}$ significantly.

- Then this can mean only one thing: *that the direction we are traveling leads us to a point where* $-\nabla g\left(\mathbf{w}^{k}\right) \approx \mathbf{0}_{N\times 1}$
- This is - by definition - a minimum, or saddle point) of the function.

Back to linear classifiers

Perceptron objective function

- We did not so far explicitly specify an objective function or loss function for the perceptron
- Can we write down a loss function for the perceptron?

$$E_{0/1}(\mathbf{w}) = \sum_p \max\{-d_p h_{\mathbf{w}}(\mathbf{x}_p), 0\}$$

where

d_p is the label $(+1$ or $-1)$ for sample $\mathbf{x}_p$

$h_{\mathbf{w}}(\mathbf{x}_p) = +1$ if $\mathbf{w} \cdot \mathbf{x}_p > 0$ and $h_{\mathbf{w}}(\mathbf{x}_p) = -1$ if $\mathbf{w} \cdot \mathbf{x}_p < 0$

$-d_p h_{\mathbf{w}}(\mathbf{x}_p) = +1$ if and only if d_p and $h_{\mathbf{w}}(\mathbf{x}_p)$ agree

in which case $\max\{-d_p h_{\mathbf{w}}(\mathbf{x}_p), 0\} = \max\{1,0\} = 1$

$-d_p h_{\mathbf{w}}(\mathbf{x}_p) = -1$ if and only if d_p and $h_{\mathbf{w}}(\mathbf{x}_p)$ disagree

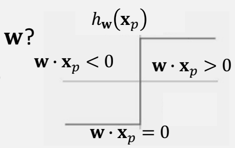
in which case $\max\{-d_p h_{\mathbf{w}}(\mathbf{x}_p), 0\} = \max\{-1,0\} = 0$

The max operation ensures that contribution of a sample $\mathbf{x}_p$ to $E_{0/1}(\mathbf{w})$ is 1 if $h_{\mathbf{w}}(\mathbf{x}_p)$ misclassifies $\mathbf{x}_p$ and it is 0 otherwise.

Perceptron objective function

$$E_{0/1}(\mathbf{w}) = \sum_p \max\{-d_p h_{\mathbf{w}}(\mathbf{x}_p), 0\}$$

The perceptron loss function simply counts the number of misclassified samples.



- Is $E_{0/1}(\mathbf{w})$ differentiable with respect to $\mathbf{w}$?
- No, because $h_{\mathbf{w}}(\mathbf{x}_p)$ is not differentiable with respect to $\mathbf{w}$
- We cannot use gradient descent!
- Nevertheless, there is a non gradient based algorithm, namely, Rosenblatt's perceptron algorithm which is guaranteed to converge to a separating hyperplane but only when the classes are separable.

Perceptron Algorithm

- Perceptron algorithm is guaranteed to converge to a separating hyperplane whenever the training data are linearly separable.
- What if the training data are not linearly separable?
  - All bets are off.
  - The algorithm runs for ever, cycling indefinitely trying to correct errors that cannot be corrected (proof omitted)
- Can we come up with an algorithm that converges when the data are separable, and achieves a reasonable compromise solution when the data are not separable?
  - Yes, as we shall see next

Can we define an alternative differentiable loss function?

$g_{\mathbf{w}}(\mathbf{x}_p) = \mathbf{w} \cdot \mathbf{x}_p$

$h_{\mathbf{w}}(\mathbf{x}_p) = +1$ if $\mathbf{w} \cdot \mathbf{x}_p > 0$ and $h_{\mathbf{w}}(\mathbf{x}_p) = -1$ if $\mathbf{w} \cdot \mathbf{x}_p < 0$

Let $E_p(\mathbf{w}) = \max\{-d_p g_{\mathbf{w}}(\mathbf{x}_p), 0\}$

$E_{soft}(\mathbf{w}) = \sum_p E_p(\mathbf{w})$

where d_p is the label ($+1$ or -1) for sample $\mathbf{x}_p$

$-d_p g_{\mathbf{w}}(\mathbf{x}_p) =$

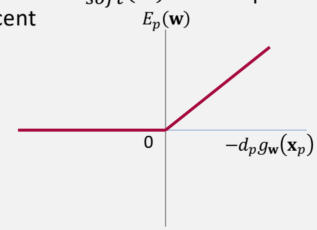
- $+g_{\mathbf{w}}(\mathbf{x}_p)$ if d_p and $g_{\mathbf{w}}(\mathbf{x}_p)$ are of different signs
- $-g_{\mathbf{w}}(\mathbf{x}_p)$ if d_p and $g_{\mathbf{w}}(\mathbf{x}_p)$ are of same sign

The max operation ensures that contribution of a sample $\mathbf{x}_p$ to $E_{soft}(\mathbf{w})$ is

- $g_{\mathbf{w}}(\mathbf{x}_p)$ whenever $h_{\mathbf{w}}(\mathbf{x}_p)$ misclassifies $\mathbf{x}_p$ and
- 0 otherwise.

Can we define an alternative loss function?

- We can show that $E_{soft}(\mathbf{w}) = \sum_p \max\{-d_p g_{\mathbf{w}}(\mathbf{x}_p), 0\}$ is convex, continuous, and has first order derivatives with respect to $\mathbf{w}$ except where $g_{\mathbf{w}}(\mathbf{x}_p)$=0.
- So we can minimize $E_{soft}(\mathbf{w})$ with respect to $\mathbf{w}$ using (sub) gradient descent



Vector and matrix calculus

Scalar analog

$f(x)$	$\frac{df}{dx}$
ax	a
x^2	$2x$
ax^2	$2ax$
e^{ax}	ae^{ax}

Vector or Matrix counterpart

$f(\mathbf{w})$	$\frac{df}{d\mathbf{w}}$
$\mathbf{W}^T\mathbf{A}$	$\mathbf{A}$
$\mathbf{W}^T a$	a
$\mathbf{W}^T\mathbf{W}$	$2\mathbf{W}$
$\mathbf{w}^T\mathbf{B}\mathbf{w}$	$2\mathbf{B}\mathbf{w}$
$\mathbf{a}\cdot\mathbf{w}$	$\mathbf{a}$
$e^{\mathbf{a}\cdot\mathbf{w}}$	$\mathbf{a}e^{\mathbf{a}\cdot\mathbf{w}}$

a scalar constant
x scalar variable
$\mathbf{w}$ vector variable
$\mathbf{A}$ constant matrix
$\mathbf{B}$ a constant square matrix
$\mathbf{W}$ a square matrix variable
$\mathbf{a}$ a constant vector

Reference: http://www.cs.cmu.edu/~mgormley/courses/10601/slides/10601-matrix-calculus.pdf

An alternative loss function $E_{soft}(\mathbf{w})$

- We can show that $E_{soft}(\mathbf{w}) = \sum_p \max\{-d_p g_{\mathbf{w}}(\mathbf{x}_p), 0\}$ is convex, and differentiable with respect to $\mathbf{w}$ except at loss = 0.
- So we can minimize $E_{soft}(\mathbf{w})$ with respect to $\mathbf{w}$ using (sub)gradient descent

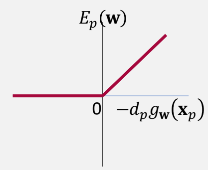
$$\nabla_{\mathbf{w}} E_{soft} = \nabla_{\mathbf{w}} \sum_{p: d_p = h_{\mathbf{w}}(\mathbf{x}_p)} \max\{-d_p g_{\mathbf{w}}(\mathbf{x}_p), 0\} + \nabla_{\mathbf{w}} \sum_{p: d_p \neq h_{\mathbf{w}}(\mathbf{x}_p)} \max\{-d_p g_{\mathbf{w}}(\mathbf{x}_p), 0\}$$

$$= 0 + \nabla_{\mathbf{w}} \sum_{p: d_p \neq h_{\mathbf{w}}(\mathbf{x}_p)} -d_p g_{\mathbf{w}}(\mathbf{x}_p)$$

$$= \sum_{p: d_p \neq h_{\mathbf{w}}(\mathbf{x}_p)} -d_p \nabla_{\mathbf{w}} g_{\mathbf{w}}(\mathbf{x}_p)$$

$$= \sum_{p: d_p \neq h_{\mathbf{w}}(\mathbf{x}_p)} -d_p \nabla_{\mathbf{w}} (\mathbf{w} \cdot \mathbf{x}_p)$$

$$= \sum_{p: d_p \neq h_{\mathbf{w}}(\mathbf{x}_p)} -d_p \mathbf{x}_p$$



Minimizing $E_{soft}(\mathbf{w})$ using (sub) gradient descent

$y_p = h_{\mathbf{w}}(\mathbf{x}_p)$

$\nabla_{\mathbf{w}} E_{soft} = \sum_{p:d_p \neq h_{\mathbf{w}}(\mathbf{x}_p)} -d_p \, \mathbf{x}_p$

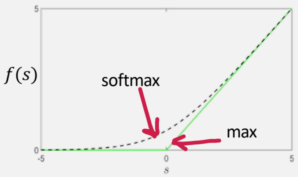
$\mathbf{w} \leftarrow \mathbf{w} - \eta \, \nabla_{\mathbf{w}} E_{soft}$

$\mathbf{w} \leftarrow \mathbf{w} + \eta \sum_{p:d_p \neq h_{\mathbf{w}}(\mathbf{x}_p)} d_p \; \mathbf{x}_p$

- We add a fraction of $\mathbf{x}_p$ if the desired label is $+1$ and the predicted label is -1
- We subtract a fraction of $\mathbf{x}_p$ if the desired label is -1 and the predicted label is $+1$
- The key difference from the perceptron algorithm is that because we perform gradient descent, we minimize the loss (error) over the training data even if the classes are not linearly separable!

Remarks on the E_{soft} loss function

- $E_{soft}(\mathbf{w}) = \sum_p \max\{-d_p g_{\mathbf{w}}(\mathbf{x}_p), 0\}$ has a trivial minimum at $\mathbf{w}$ =0 that we must take steps in our code to avoid
- We can minimize E_{soft} using only first order (sub)gradient descent (higher order derivatives do not exist)
- Can we approximate E_{soft} by a smooth loss function, say E_{smooth} so we can use a broader range of optimization methods, including higher order methods?
  - Yes
  - By replacing max by softmax



Approximating max by softmax

Suppose $\max\{a,b\} = a$

Recall that $\log e^x = x$

$\max\{a,b\} = b + (a-b) = \log e^b + \log e^{a-b}$

Let $\text{softmax}\,\{a,b\} = \log\left(e^a + e^b\right)$

Note that $\log e^b + \log\left(1 + e^{a-b}\right) = \log\left(e^b\left(1 + e^{a-b}\right)\right)$

$$= \log\left(e^a + e^b\right) = \text{softmax}\,\{a,b\}$$

$\text{softmax}\,\{a,b\} - \max\{a,b\}$

$$= \log e^b + \log\left(1 + e^{a-b}\right) - \log e^b - \log e^{a-b}$$

$$= \log\left(1 + e^{a-b}\right) - \log e^{a-b}$$

$$= \log \frac{\left(1+e^{a-b}\right)}{e^{a-b}} = 1 + \frac{1}{e^{a-b}} \approx 1 \text{ especially when } e^{a-b} \gg 1$$

E_{soft} to E_{smooth} via softmax

- $E_{soft}(\mathbf{w}) = \sum_p \max\{-d_p g_{\mathbf{w}}(\mathbf{x}_p), 0\}$

Approximating max by softmax, we have:

- $E_{smooth}(\mathbf{w}) = \sum_p \log \left(e^0 + e^{-d_p \mathbf{w} \cdot \mathbf{x}_p}\right)$
  $= \sum_p \log \left(1 + e^{-d_p \mathbf{w} \cdot \mathbf{x}_p}\right)$

- E_{smooth}
  - Is convex and infinitely differentiable, hence we can use higher order optimization methods
  - Does not have a trivial minimum at $\mathbf{w} = 0$

E_{soft} to E_{smooth} via softmax

- $E_{smooth}(\mathbf{w}) = \sum_p \log \left(1 + e^{-d_p \mathbf{w}\cdot\mathbf{x}_p}\right)$
- Empirically, we find that only the first several iterations improve E_{smooth} before the magnitude of the weights starts to become very large
- Solution: regularization – limit the magnitude of weights from increasing without bounds
- $E^R_{smooth}(w_0, \boldsymbol{\omega}) = \sum_p \log \left(1 + e^{-d_p \boldsymbol{\omega}\cdot\mathbf{x}_p - d_p w_0}\right) + \lambda\|\boldsymbol{\omega}\|^2$ where $\mathbf{w} = [w_0\, w_1 \cdots w_N]^T,\ \boldsymbol{\omega} = [w_1 \cdots w_N]^T$

λ is set to a small value, e.g., 0.0001 and prevents the weights from increasing without bounds

Alternatively, λ can be optimized using cross-validation

We will study regularization in greater detail later

E_{soft} to E_{smooth} via softmax

$E^R_{smooth}(w_0, \boldsymbol{\omega}) = \sum_p \log \left(1 + e^{-d_p \boldsymbol{\omega}\cdot\mathbf{x}_p - d_p w_0}\right) + \lambda\|\boldsymbol{\omega}\|^2$ where if

$$\mathbf{w} = [w_0\ w_1 \cdots w_N]^T, \qquad \boldsymbol{\omega} = [w_1 \cdots w_N]^T$$

Gradient based update:

$$\nabla_{\boldsymbol{\omega}} E^R_{smooth}(w_0, \boldsymbol{\omega}) = \nabla_{\boldsymbol{\omega}}\left(\sum_p \log \left(1 + e^{-d_p \boldsymbol{\omega}\cdot\mathbf{x}_p - d_p w_0}\right) + \lambda\|\boldsymbol{\omega}\|^2\right)$$

$$= \sum_p \frac{1}{\left(1 + e^{-d_p \boldsymbol{\omega}\cdot\mathbf{x}_p - d_p w_0}\right)} \nabla_{\boldsymbol{\omega}}\left(1 + e^{-d_p \boldsymbol{\omega}\cdot\mathbf{x}_p - d_p w_0}\right) + \nabla_{\boldsymbol{\omega}}(\lambda\|\boldsymbol{\omega}\|^2)$$

$$= -\sum_p \frac{e^{-d_p \boldsymbol{\omega}\cdot\mathbf{x}_p - d_p w_0}}{\left(1 + e^{-d_p \boldsymbol{\omega}\cdot\mathbf{x}_p - d_p w_0}\right)} d_p \mathbf{x}_p + 2\lambda\boldsymbol{\omega}$$

Weight update

$$\boldsymbol{\omega} \leftarrow \boldsymbol{\omega} - \eta\, \nabla_{\boldsymbol{\omega}} E^R_{smooth}(w_0, \boldsymbol{\omega})$$

E_{soft} to E_{smooth} via softmax

$E^{R}_{smooth}(w_0, \boldsymbol{\omega}) = \sum_p \log \left(1 + e^{-d_p \boldsymbol{\omega}\cdot\mathbf{x}_p - d_p w_0}\right) + \lambda\|\boldsymbol{\omega}\|^2$ where $\mathbf{w} = [w_0\, w_1 \cdots w_N]^T,\ \boldsymbol{\omega} = [w_1 \cdots w_N]^T$

Gradient based update:

$$\nabla_{w_0} E^{R}_{smooth}(w_0, \boldsymbol{\omega}) = \nabla_{w_0}\left(\sum_p \log \left(1 + e^{-d_p \boldsymbol{\omega}\cdot\mathbf{x}_p - d_p w_0}\right) + \lambda\|\boldsymbol{\omega}\|^2\right)$$

$$= \sum_p \frac{1}{\left(1 + e^{-d_p \boldsymbol{\omega}\cdot\mathbf{x}_p - d_p w_0}\right)} \nabla_{w_0}\left(1 + e^{-d_p \boldsymbol{\omega}\cdot\mathbf{x}_p - d_p w_0}\right) + 0$$

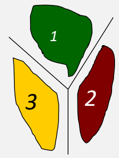
$$= -\sum_p \frac{e^{-d_p \boldsymbol{\omega}\cdot\mathbf{x}_p - d_p w_0}}{\left(1 + e^{-d_p \boldsymbol{\omega}\cdot\mathbf{x}_p - d_p w_0}\right)} d_p$$

Weight update

$$w_0 \leftarrow w_0 - \eta\, \nabla_{\boldsymbol{\omega}}\, E^{R}_{smooth}(w_0, \boldsymbol{\omega})$$

Multi-class extension

Predicted class label for $\mathbf{x}_p$ is given by $y_p = argmax_c \ \mathbf{w}_c \cdot \mathbf{x}_p$



Predicted class label for $\mathbf{x}_p$ is d_p

Suppose we define E_p, the error on sample $\mathbf{x}_p$

$$E_p(\mathbf{w}_1, \cdots \mathbf{w}_C) = \max_{c=1,\cdots,C;\& \ c \neq d_p} \left\{0, \ \mathbf{x}_p \cdot \left(\mathbf{w}_c - \mathbf{w}_{d_p}\right)\right\}$$

Decision surface between class k and j is given by $(\mathbf{w}_k - \mathbf{w}_j) \cdot \mathbf{x} = 0$

Error on the training set

$$E = \sum_p E_p(\mathbf{w}_1, \cdots \mathbf{w}_C)$$

Multi-class softmax

Suppose we define E_p, the error on sample $\mathbf{x}_p$

$$E_p(\mathbf{w}_1, \cdots \mathbf{w}_C) = \max_c \mathbf{x}_p \cdot \mathbf{w}_c - \mathbf{w}_{d_p} \cdot \mathbf{x}_p$$

$$\approx \log\left(\sum_{c=1}^{C} e^{\mathbf{x}_p \cdot \mathbf{w}_c}\right) - \mathbf{w}_{d_p} \cdot \mathbf{x}_p$$

$$E = \sum_p E_p(\mathbf{w}_1, \cdots \mathbf{w}_C)$$

$$\nabla_{\mathbf{w}_c} E = \nabla_{\mathbf{w}_c} \left(\log\left(\sum_{j=1}^{C} e^{\mathbf{x}_p \cdot \mathbf{w}_j}\right) - \mathbf{w}_{d_p} \cdot \mathbf{x}_p\right)$$

$$= \left(\frac{1}{\sum_{j=1}^{C} e^{\mathbf{x}_p \cdot \mathbf{w}_j}}\right) \nabla_{\mathbf{w}_c} \left(\log\left(\sum_{j \neq c} e^{\mathbf{x}_p \cdot \mathbf{w}_j}\right) + \log \quad e^{\mathbf{x}_p \cdot \mathbf{w}_c}\right) - 0$$

$$= \left(\frac{1}{\sum_{j=1}^{C} e^{\mathbf{x}_p \cdot \mathbf{w}_j}}\right) (0 + e^{\mathbf{x}_p \cdot \mathbf{w}_c}) \nabla_{\mathbf{w}_c} (\mathbf{x}_p \cdot \mathbf{w}_c)$$

$$= \left(\frac{e^{\mathbf{x}_p \cdot \mathbf{w}_c}}{\sum_{j=1}^{C} e^{\mathbf{x}_p \cdot \mathbf{w}_j}}\right) \mathbf{x}_p$$

$$\mathbf{w}_c \leftarrow \mathbf{w}_c - \eta \left(\frac{e^{\mathbf{x}_p \cdot \mathbf{w}_c}}{\sum_{j=1}^{C} e^{\mathbf{x}_p \cdot \mathbf{w}_j}}\right) \mathbf{x}_p$$

Multi-class softmax

Suppose we define E_p, the error on sample $\mathbf{x}_p$

$$E_p(\mathbf{w}_1, \cdots \mathbf{w}_C) \approx \log\left(\sum_{c=1}^{C} e^{\mathbf{x}_p \cdot \mathbf{w}_c}\right) - \mathbf{w}_{d_p} \cdot \mathbf{x}_p$$

$$E = \sum_p E_p(\mathbf{w}_1, \cdots \mathbf{w}_C)$$

$$\nabla_{\mathbf{w}_{d_p}} E = \nabla_{\mathbf{w}_{d_p}} \left(\log\left(\sum_{j=1}^{C} e^{\mathbf{x}_p \cdot \mathbf{w}_j}\right) - \mathbf{w}_{d_p} \cdot \mathbf{x}_p\right)$$

$$= \left(\frac{1}{\sum_{j=1}^{C} e^{\mathbf{x}_p \cdot \mathbf{w}_j}}\right) \nabla_{\mathbf{w}_{d_p}} \left(e^{\mathbf{x}_p \cdot \mathbf{w}_{d_p}}\right) - \nabla_{\mathbf{w}_{d_p}} \left(\mathbf{w}_{d_p} \cdot \mathbf{x}_p\right)$$

$$= \left(\frac{1}{\sum_{j=1}^{C} e^{\mathbf{x}_p \cdot \mathbf{w}_j}}\right) \left(e^{\mathbf{x}_p \cdot \mathbf{w}_{d_p}}\right) \nabla_{\mathbf{w}_{d_p}} \left(\mathbf{x}_p \cdot \mathbf{w}_{d_p}\right) - \mathbf{x}_p$$

$$= \left(\frac{e^{\mathbf{x}_p \cdot \mathbf{w}_c}}{\sum_{j=1}^{C} e^{\mathbf{x}_p \cdot \mathbf{w}_j}}\right) \mathbf{x}_p - \mathbf{x}_p = -\mathbf{x}_p \left(1 - \frac{e^{\mathbf{x}_p \cdot \mathbf{w}_c}}{\sum_{j=1}^{C} e^{\mathbf{x}_p \cdot \mathbf{w}_j}}\right)$$

$$\mathbf{w}_{d_p} \leftarrow \mathbf{w}_{d_p} + \eta \left(1 - \frac{e^{\mathbf{x}_p \cdot \mathbf{w}_c}}{\sum_{j=1}^{C} e^{\mathbf{x}_p \cdot \mathbf{w}_j}}\right) \mathbf{x}_p$$

Multi-class extension

- The softmax based loss function for multi-class perceptron needs to be regularized for the same reason its 2-class counterpart needs to be regularized

Capabilities and limitations of a perceptron

Capabilities

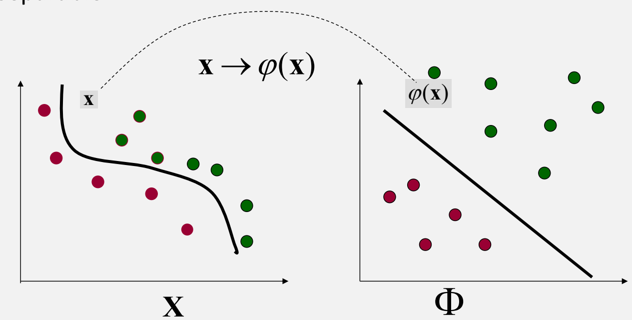
- Perceptron can represent threshold functions
- Perceptron can learn linear decision boundaries

Limitations

- What if the data are not linearly separable?
  - More complex networks?
  - Non-linear transformations into a feature space where the data become separable?

Extending Linear Classifiers

Map data into a feature space where they are linearly separable

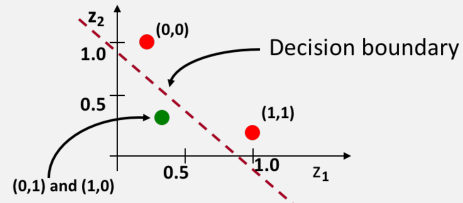


Exclusive OR revisited

In the feature (hidden) space:

$$\varphi_1(x_1, x_2) = e^{-\|\mathbf{X}-\mathbf{W}_1\|^2} = z_1 \qquad \mathbf{W}_1 = [1,1]^T$$

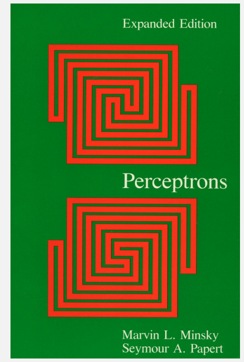
$$\varphi_2(x_1, x_2) = e^{-\|\mathbf{X}-\mathbf{W}_2\|^2} = z_2 \qquad \mathbf{W}_2 = [0,0]^T$$



When mapped into the feature space < z_1 , z_2 >, C1 and C2 become *linearly separable.* So a linear classifier with $\varphi_1(x)$ and $\varphi_2(x)$ as inputs can be used to solve the XOR problem.

"Perceptrons" (1969)

"The perceptron […] has many features that attract attention: its linearity, its intriguing learning theorem; its clear paradigmatic simplicity as a kind of parallel computation. *There is no reason to suppose that any of these virtues carry over to the many-layered version*. Nevertheless, *we consider it to be an important research problem to elucidate (or reject) our intuitive judgement that the extension is sterile*."
[pp. 231 – 232]



Postscript

- Minsky and Papert's book had a chilling effect on machine learning research in the US for the next 25 years
  - A few die-hards continued to work on machine learning
  - Artificial Intelligence research shifted to knowledge-based systems
  - Some success with human-engineered knowledge bases
  - Knowledge engineering bottleneck encountered (1980's)
  - Renewed interest in machine learning (mid-late 1980's)
  - Practical approaches to training multi-layer neural networks (late 1980s)
  - Data and computing revolution (1990s – 2000s)
  - Machine learning takes over Artificial Intelligence (2010 – present)