The Perceptron
Support Vector Machine
Loss function and Regulization

Machine Learning - Lecture 4

Sardar Hamidian

September 20, 2019

The Perceptron (Neuron)

The Perceptron is seen as an **analogy** to a biological neuron.

Biological neurons fire an impulse once the sum of all inputs is over a threshold.

The sigmoid emulates the thresholding behavior → act like a switch.

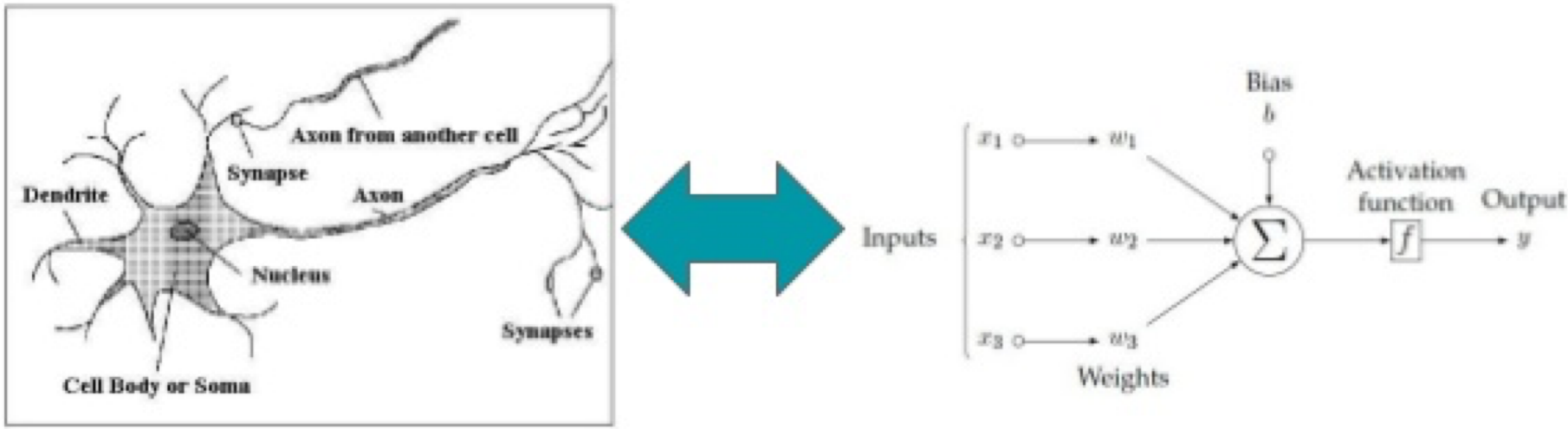
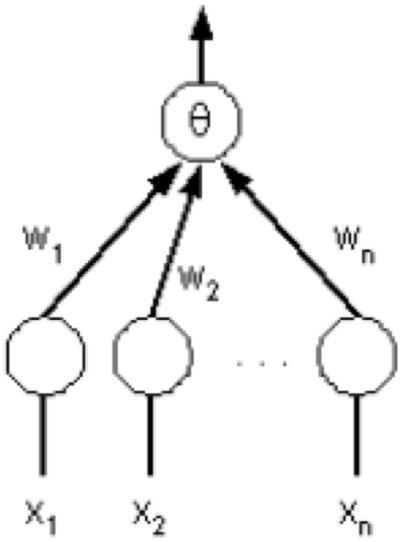


Figure credit: Introduction to AI

Artificial neurons

McCulloch & Pitts (1943) described an artificial neuron

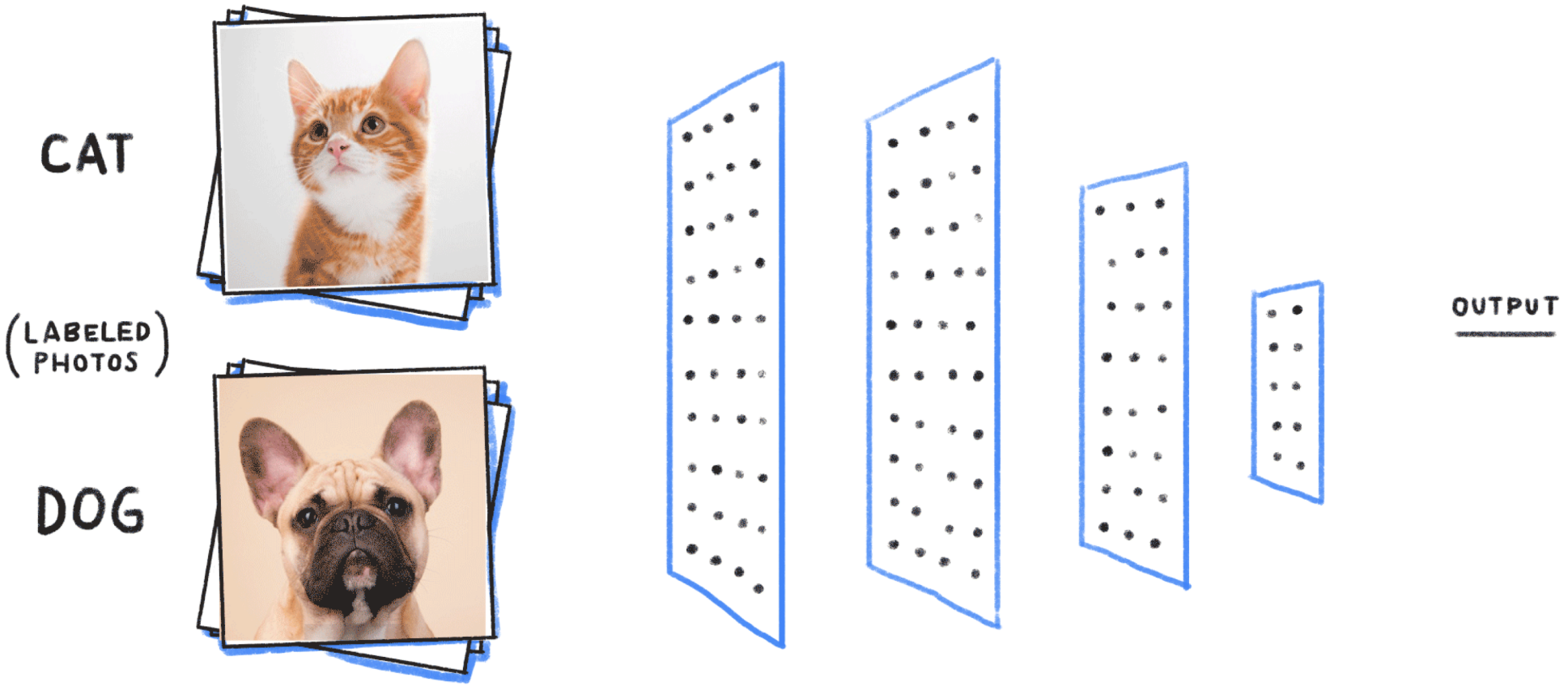
- inputs are either excitatory (+1) or inhibitory (-1)
- each input has a weight associated with it
- the activation function multiplies each input value by its weight
- if the sum of the weighted inputs >= **θ**,
  then the neuron fires (returns 1), else doesn't fire (returns –1)



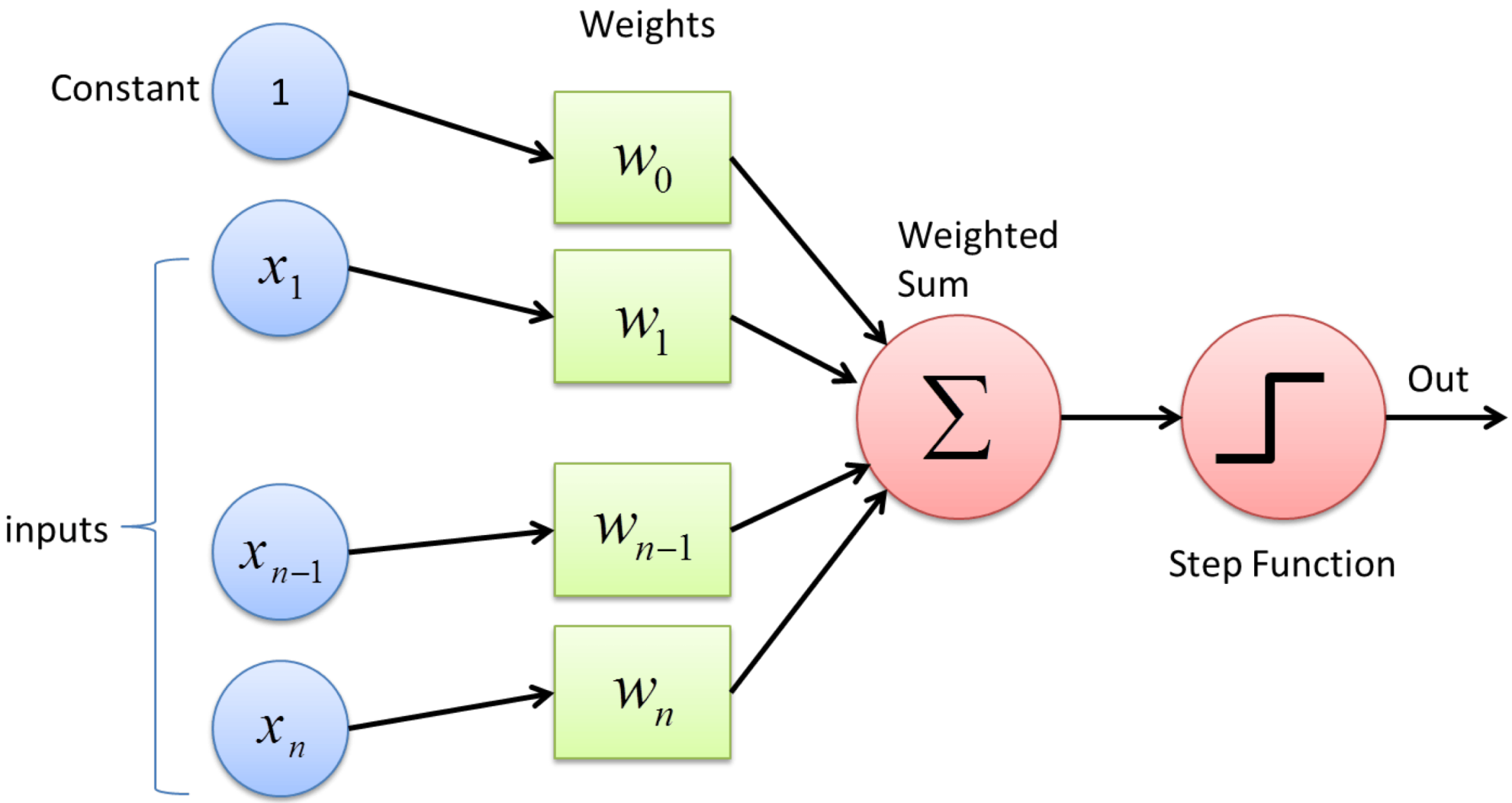
if $\Sigma w_i x_i >= \theta$, output = 1

if $\Sigma w_i x_i < \theta$, output -1

Neural networks



Perceptron


Weights
Constant
1
w_0
x_1
w_1
Weighted
Sum
Σ
Out
inputs
x_{n-1}
w_{n-1}
x_n
w_n
Step Function

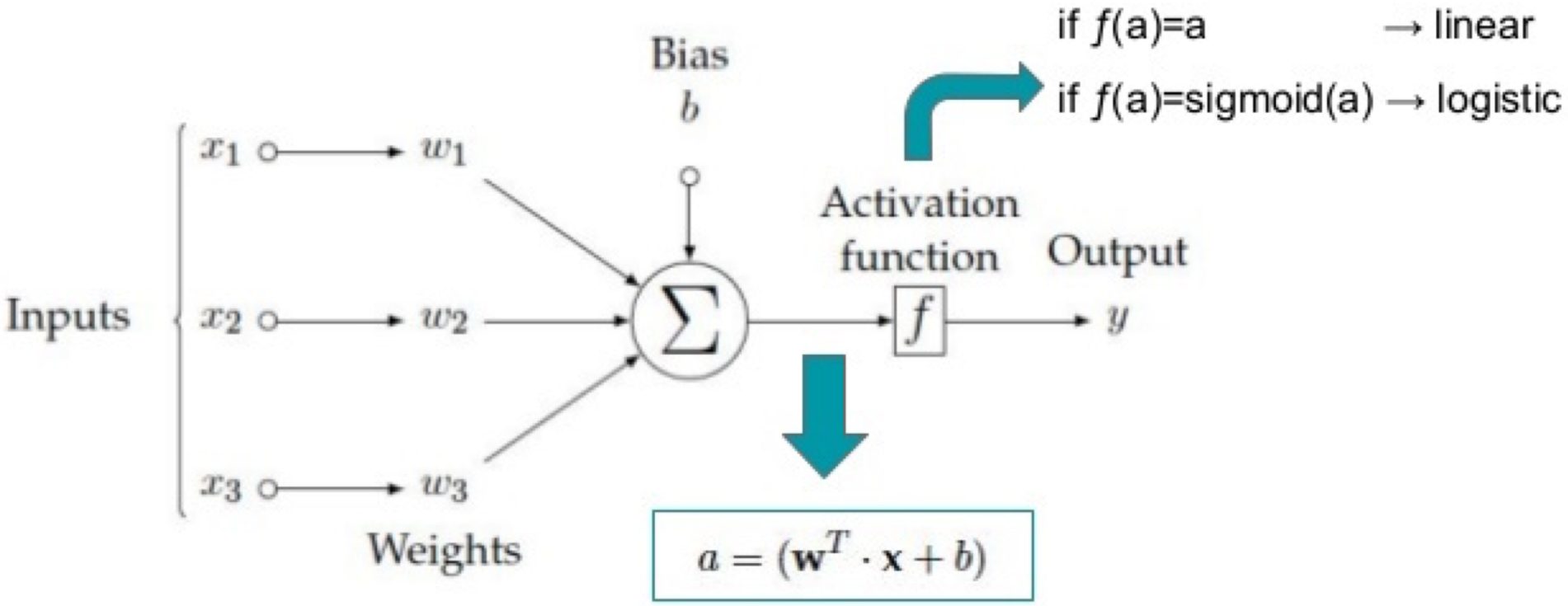
Perceptron is a one layer neural network

Consists of 4 parts :

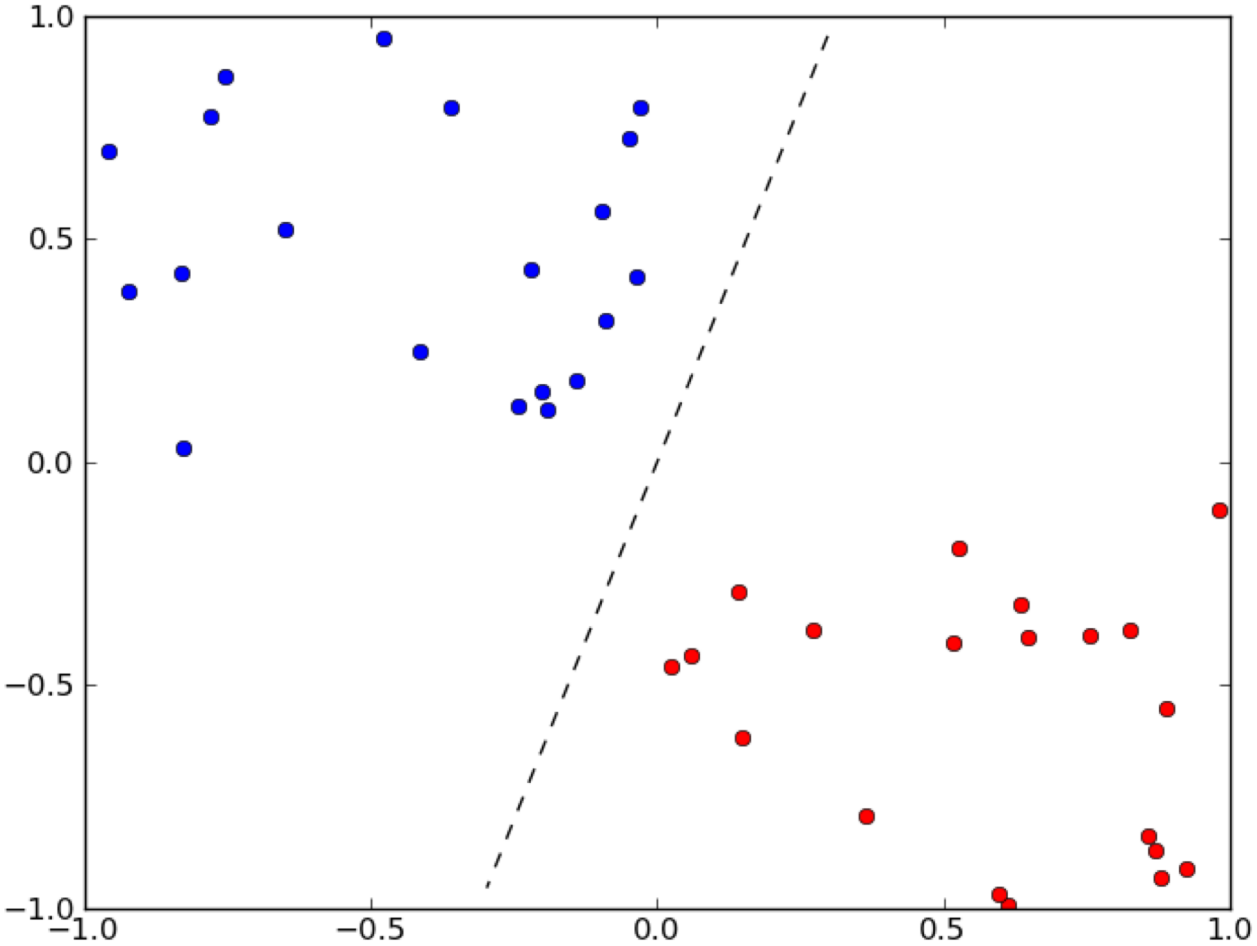
1. Input values or One input layer
2. Weights and Bias
3. Net sum
4. Activation Function

The Perceptron (Neuron)

The Perceptron can represent both linear & logistic regression:

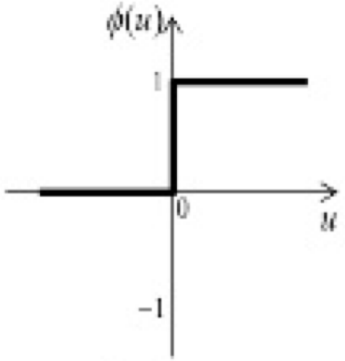


Perceptron as a linear binary classifier



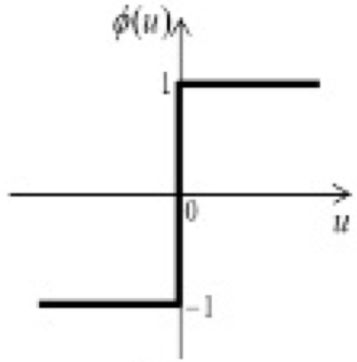
Activation Functions

step function



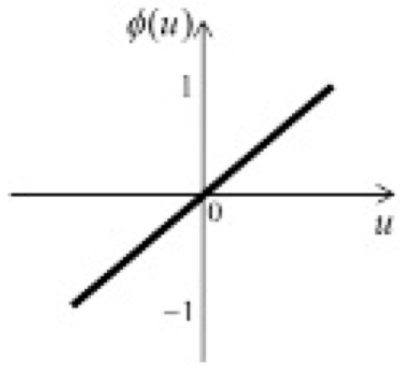
$$\phi_{step}(u) = \begin{cases} 1 & \text{if } u \geq 0 \\ 0 & \text{otherwise} \end{cases}$$

sign function



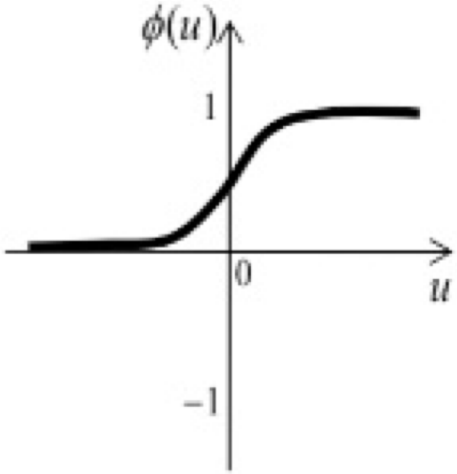
$$\phi_{sign}(u) = \begin{cases} 1 & \text{if } u \geq 0 \\ -1 & \text{otherwise} \end{cases}$$

identity function



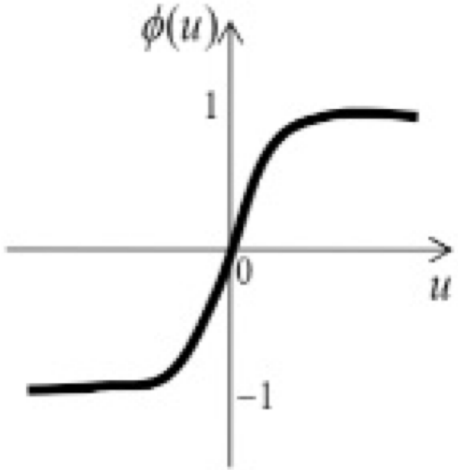
$$\phi_{id}(u) = u$$

sigmoid function



$$\phi_{sig}(u) = \frac{1}{1+e^{-u}}$$

hyper tangent function



$$\phi_h = \frac{e^u - 1}{e^u + 1}$$

Perceptron algorithm

Rosenblatt (1958) devised a learning algorithm for artificial neurons

given a training set (example inputs & corresponding desired outputs)

1. start with some initial weights
2. iterate through the training set, collect incorrect examples
3. if all examples correct, then DONE
4. otherwise, update the weights for each incorrect example

   if $x_1, \ldots, x_n$ should have fired but didn't, `w_i += x_i (0 <= i <= n)`

   if $x_1, \ldots, x_n$ shouldn't have fired but did, `w_i -= x_i (0 <= i <= n)`
5. GO TO 2

artificial neurons that utilize this learning algorithm are known as *perceptrons*

Algorithm **Perceptron**

Initial weight vector: $\mathbf{w}_1 = \mathbf{0} \in \mathbb{R}^d$

For $t = 1, \ldots, T$:

- Receive instance $\mathbf{x}_t \in \mathcal{X} \subseteq \mathbb{R}^d$
- Predict $\widehat{y}_t = \text{sign}(\mathbf{w}_t^\top \mathbf{x}_t)$
- Receive true label $y_t \in \{\pm 1\}$
- Incur loss $\mathbf{1}(\widehat{y}_t \neq y_t)$
- Update: If $\widehat{y}_t \neq y_t$ then

$$\mathbf{w}_{t+1} \leftarrow \mathbf{w}_t + y_t \mathbf{x}_t$$

else

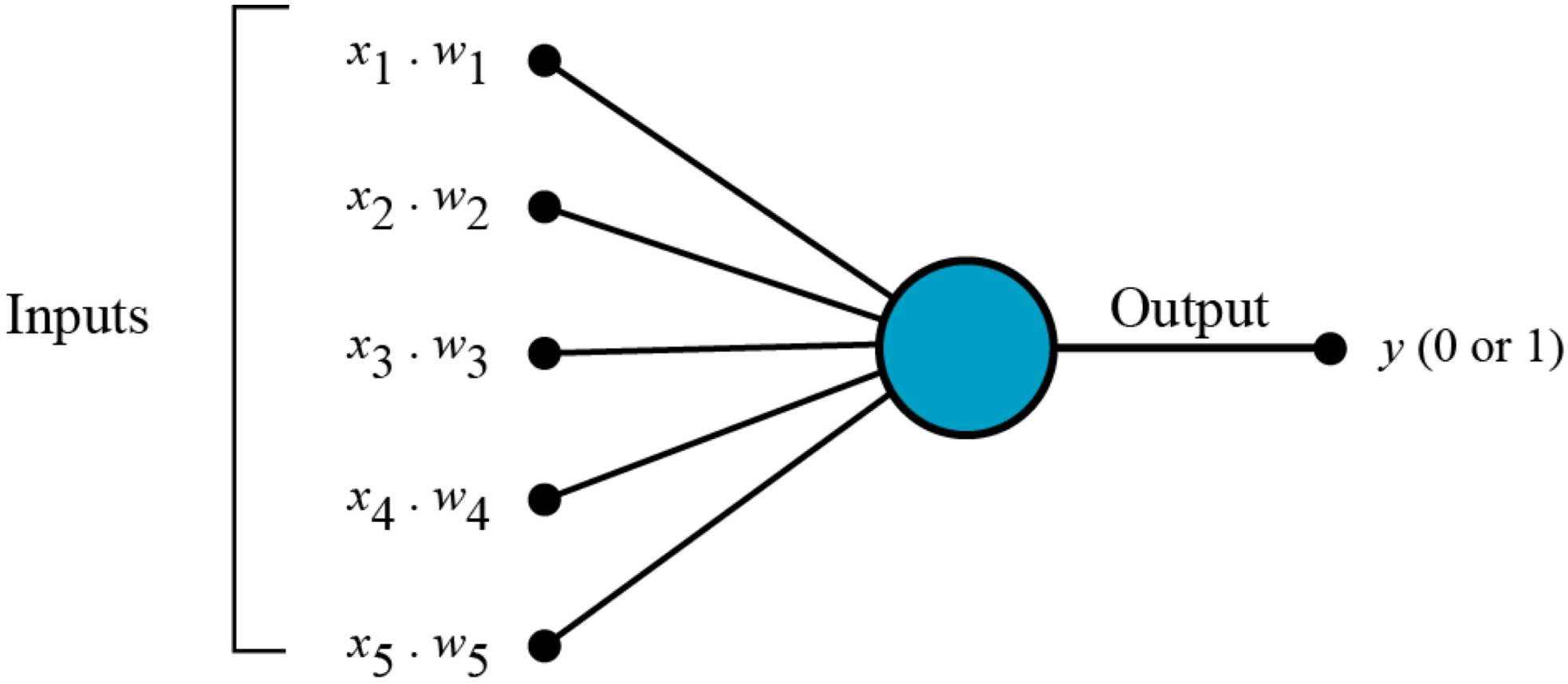
$$\mathbf{w}_{t+1} \leftarrow \mathbf{w}_t$$

Perceptron is a one layer neural network

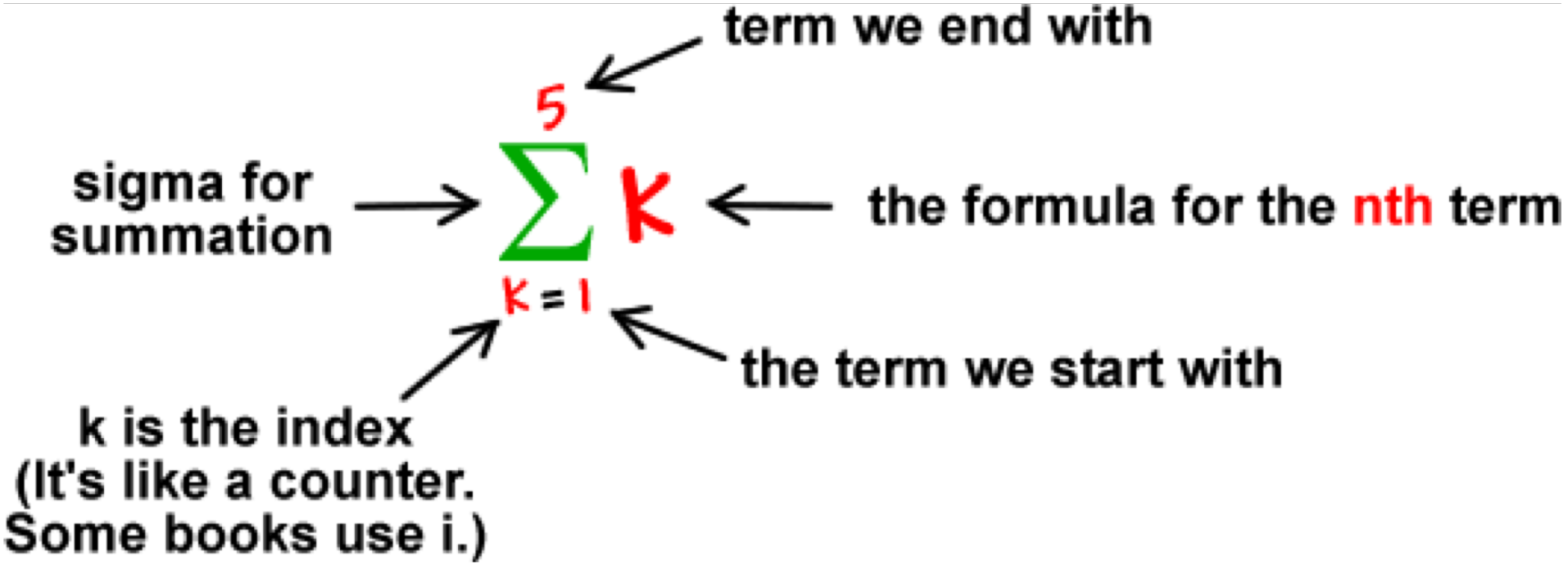
Consists of 4 parts :

1. Input values or One input layer
2. Weights and Bias
3. Net sum
4. Activation Function

a. All the inputs ***x*** are multiplied with their weights ***w***. Let's call it ***k***.



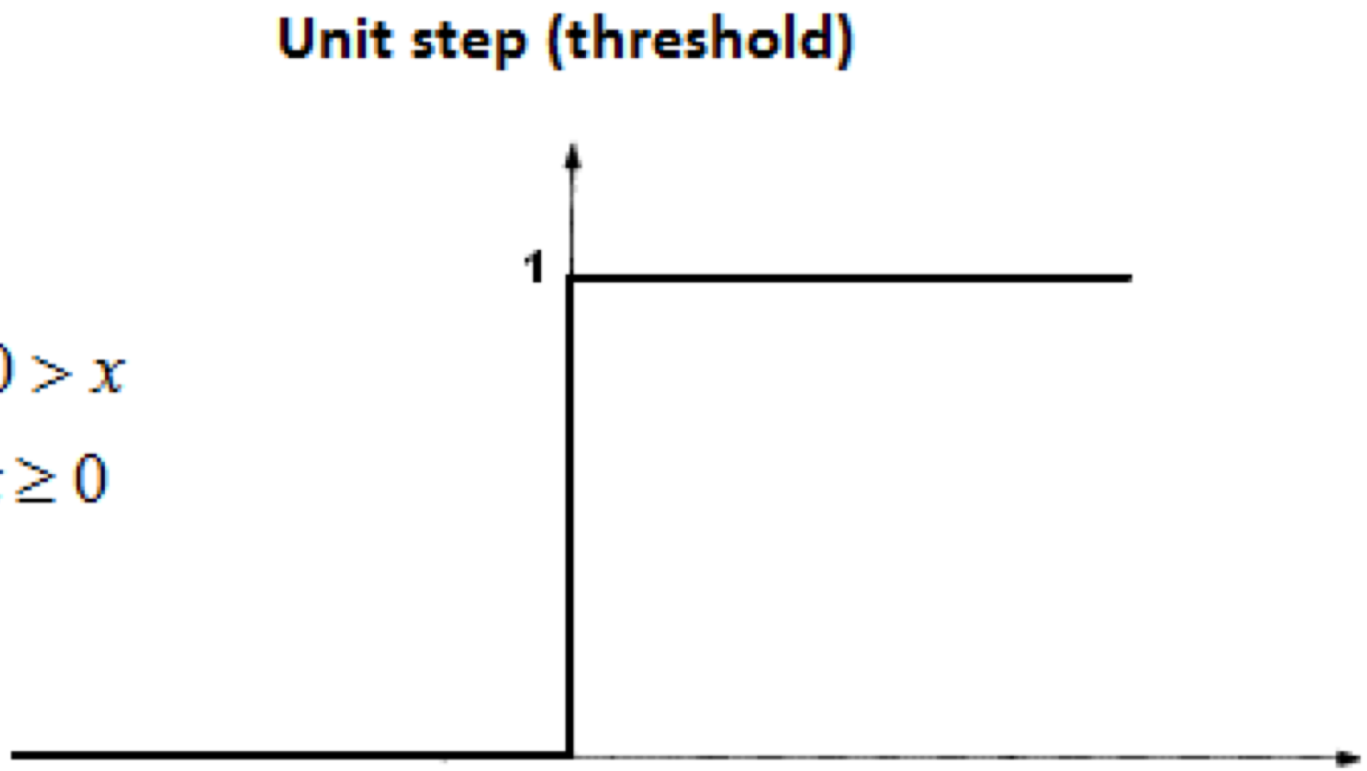
b. ***Add*** all the multiplied values and call them ***Weighted Sum.***



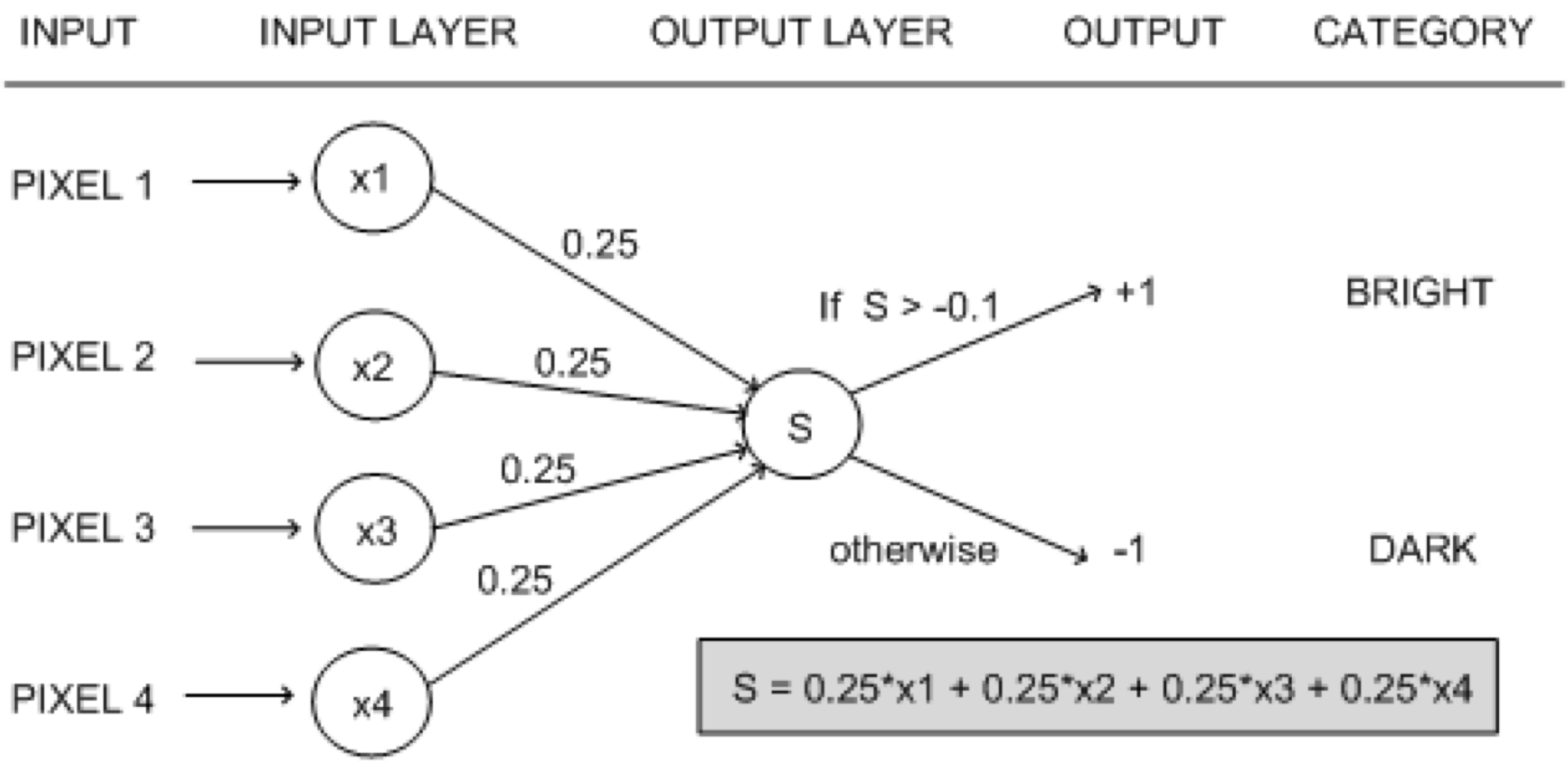
c. ***Apply*** that weighted sum to the correct *Activation Function*.

For Example : Unit Step Activation Function.

$$f(x) = \begin{cases} 0 \text{ if } 0 > x \\ 1 \text{ if } x \geq 0 \end{cases}$$



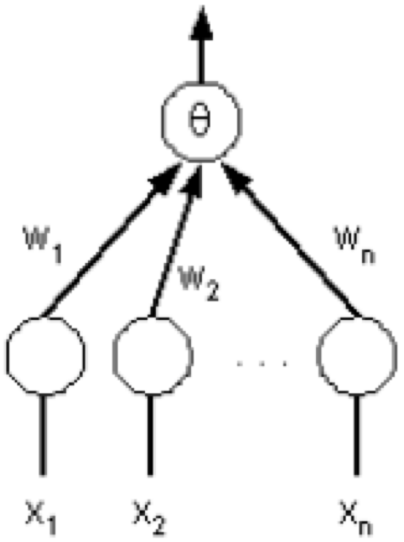
Weights show the strength of the particular node.



Artificial neurons

McCulloch & Pitts (1943) described an artificial neuron

- inputs are either excitatory (+1) or inhibitory (-1)
- each input has a weight associated with it
- the activation function multiplies each input value by its weight
- if the sum of the weighted inputs >= **θ**,
  then the neuron fires (returns 1), else doesn't fire (returns –1)



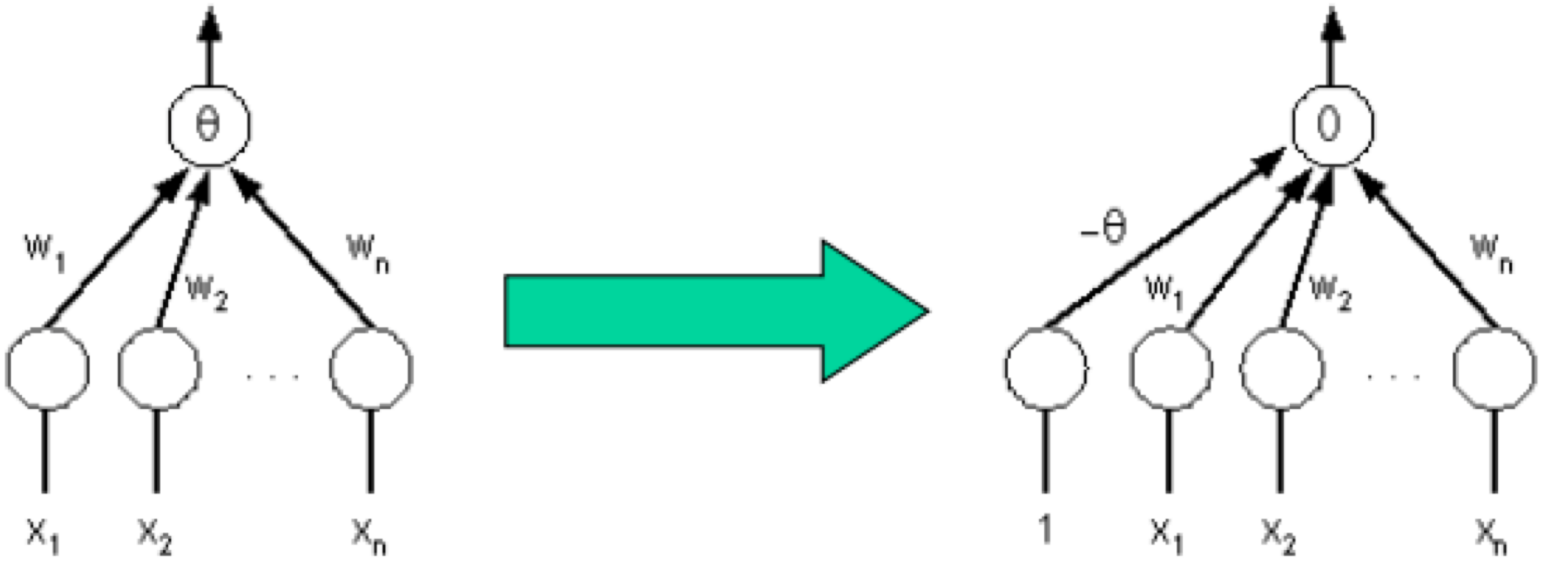
if $\Sigma w_i x_i >= \theta$, output = 1

if $\Sigma w_i x_i < \theta$, output -1

Normalizing thresholds

to make life more uniform, can normalize the threshold to 0

- simply add an additional input $x_0 = 1$, $w_0 = -\theta$



advantage: threshold = 0 for all neurons

$$\Sigma w_i x_i >= \theta \quad \equiv \quad -\theta * 1 + \Sigma w_i x_i >= 0$$

Perceptron algorithm

Rosenblatt (1958) devised a learning algorithm for artificial neurons

given a training set (example inputs & corresponding desired outputs)

1. start with some initial weights
2. iterate through the training set, collect incorrect examples
3. if all examples correct, then DONE
4. otherwise, update the weights for each incorrect example

   if $x_1, \ldots, x_n$ should have fired but didn't, `w_i += x_i (0 <= i <= n)`

   if $x_1, \ldots, x_n$ shouldn't have fired but did, `w_i -= x_i (0 <= i <= n)`
5. GO TO 2

artificial neurons that utilize this learning algorithm are known as *perceptrons*

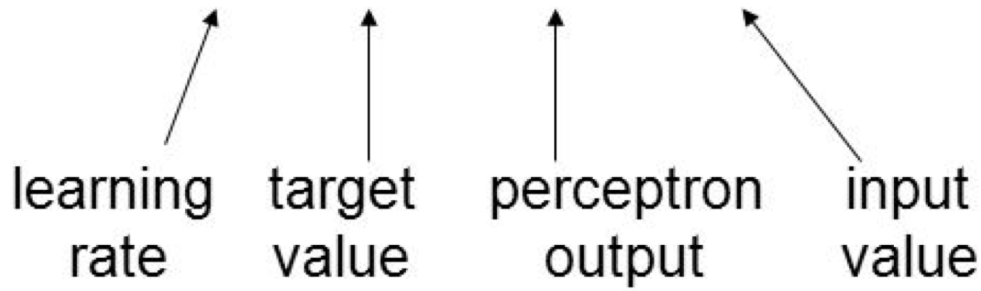
Perceptron Training Rule

- Weights modified for each example
- Update Rule:

$$w_i \leftarrow w_i + \Delta w_i$$

where

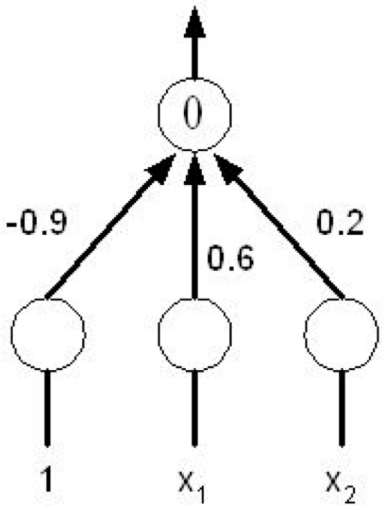
$$\Delta w_i = \eta (t - o) x_i$$



Example: perceptron learning

Suppose we want to train a perceptron to compute AND

training set: $x_1 = 1, x_2 = 1 \rightarrow 1$
$x_1 = 1, x_2 = -1 \rightarrow -1$
$x_1 = -1, x_2 = 1 \rightarrow -1$
$x_1 = -1, x_2 = -1 \rightarrow -1$



randomly, let: $w_0 = -0.9$, $w_1 = 0.6$, $w_2 = 0.2$

using these weights:

$x_1 = 1, x_2 = 1$:	$-0.9*1 + 0.6*1 + 0.2*1 = -0.1 \rightarrow -1$	WRONG
$x_1 = 1, x_2 = -1$:	$-0.9*1 + 0.6*1 + 0.2*-1 = -0.5 \rightarrow -1$	OK
$x_1 = -1, x_2 = 1$:	$-0.9*1 + 0.6*-1 + 0.2*1 = -1.3 \rightarrow -1$	OK
$x_1 = -1, x_2 = -1$:	$-0.9*1 + 0.6*-1 + 0.2*-1 = -1.7 \rightarrow -1$	OK

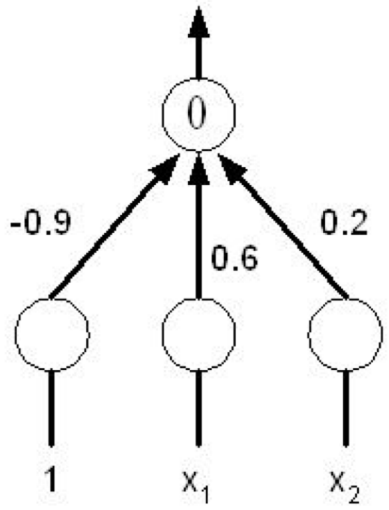
new weights: $w_0 = -0.9 + 1 = 0.1$
$w_1 = 0.6 + 1 = 1.6$
$w_2 = 0.2 + 1 = 1.2$

if $x_1, \ldots, x_n$ should have fired but didn't, `w_i += x_i (0 <= i <= n)`

if $x_1, \ldots, x_n$ shouldn't have fired but did, `w_i -= x_i (0 <= i <= n)`

Suppose we want to train a perceptron to compute AND

training set: $x_1 = 1, x_2 = 1 \rightarrow 1$
$x_1 = 1, x_2 = -1 \rightarrow -1$
$x_1 = -1, x_2 = 1 \rightarrow -1$
$x_1 = -1, x_2 = -1 \rightarrow -1$



randomly, let: $w_0 = -0.9, \quad w_1 = 0.6, \quad w_2 = 0.2$

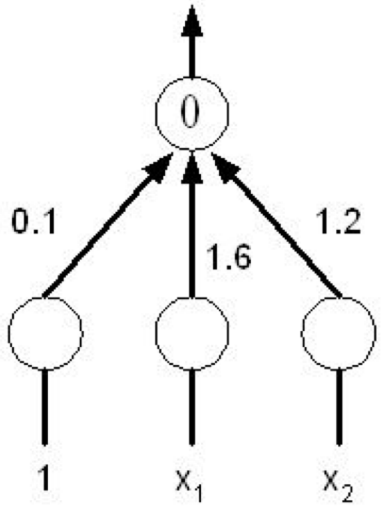
using these weights:

$x_1 = 1, x_2 = 1$:	-0.9*1 + 0.6*1 + 0.2*1 = -0.1 → -1	WRONG
$x_1 = 1, x_2 = -1$:	-0.9*1 + 0.6*1 + 0.2*-1 = -0.5 → -1	OK
$x_1 = -1, x_2 = 1$:	-0.9*1 + 0.6*-1 + 0.2*1 = -1.3 → -1	OK
$x_1 = -1, x_2 = -1$:	-0.9*1 + 0.6*-1 + 0.2*-1 = -1.7 → -1	OK

new weights: $w_0 = -0.9 + 1 = 0.1$
$w_1 = 0.6 + 1 = 1.6$
$w_2 = 0.2 + 1 = 1.2$

if $x_1, \ldots, x_n$ should have fired but didn't, w_i += x_i (0 <= i <= n)

if $x_1, \ldots, x_n$ shouldn't have fired but did, w_i -= x_i (0 <= i <= n)



using these updated weights:

$x_1 = 1, x_2 = 1$: 0.1*1 + 1.6*1 + 1.2*1 = 2.9 → 1 OK

$x_1 = 1, x_2 = -1$: 0.1*1 + 1.6*1 + 1.2*-1 = 0.5 → 1 WRONG

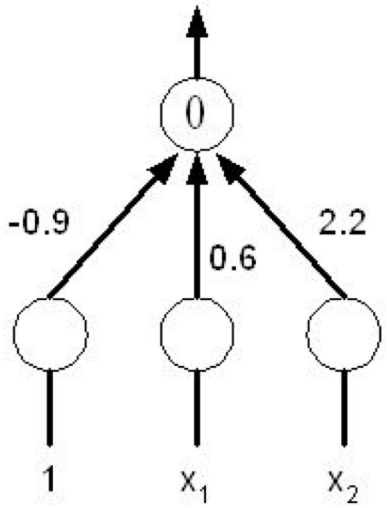
$x_1 = -1, x_2 = 1$: 0.1*1 + 1.6*-1 + 1.2*1 = -0.3 → -1 OK

$x_1 = -1, x_2 = -1$: 0.1*1 + 1.6*-1 + 1.2*-1 = -2.7 → -1 OK

new weights: $w_0 = 0.1 - 1 = -0.9$

$w_1 = 1.6 - 1 = 0.6$

$w_2 = 1.2 + 1 = 2.2$



using these updated weights:

$x_1 = 1, x_2 = 1$: -0.9*1 + 0.6*1 + 2.2*1 = 1.9 → 1 OK

$x_1 = 1, x_2 = -1$: -0.9*1 + 0.6*1 + 2.2*-1 = -2.5 → -1 OK

$x_1 = -1, x_2 = 1$: -0.9*1 + 0.6*-1 + 2.2*1 = 0.7 → 1 WRONG

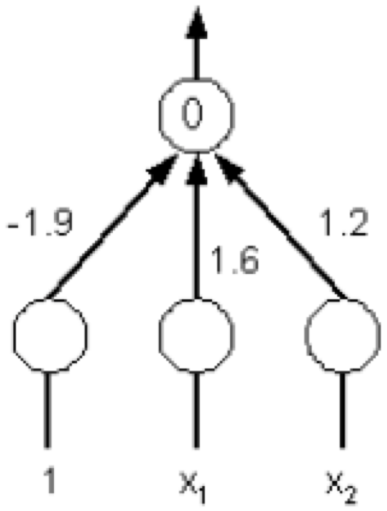
$x_1 = -1, x_2 = -1$: -0.9*1 + 0.6*-1 + 2.2*-1 = -3.7 → -1 OK

new weights: $w_0 = -0.9 - 1 = -1.9$

$w_1 = 0.6 + 1 = 1.6$

$w_2 = 2.2 - 1 = 1.2$

When all training instances classified correctly



using these updated weights:

$x_1 = 1, x_2 = 1$: -1.9*1 + 1.6*1 + 1.2*1 = 0.9 → 1 OK

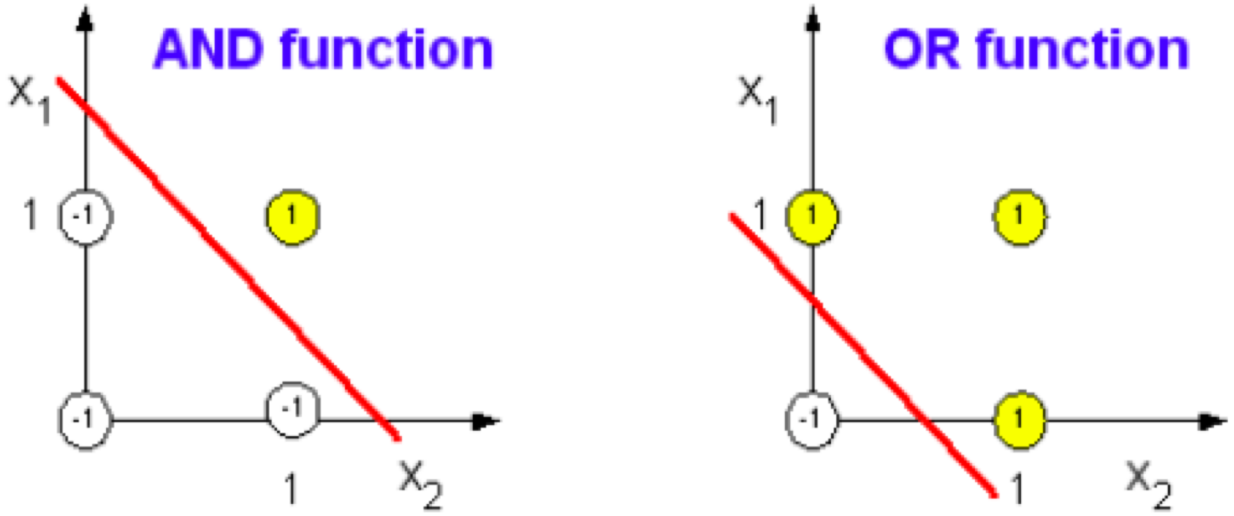
$x_1 = 1, x_2 = -1$: -1.9*1 + 1.6*1 + 1.2*-1 = -1.5 → -1 OK

$x_1 = -1, x_2 = 1$: -1.9*1 + 1.6*-1 + 1.2*1 = -2.3 → -1 OK

$x_1 = -1, x_2 = -1$: -1.9*1 + 1.6*-1 + 1.2*-1 = -4.7 → -1 OK

DONE!

Linearly separable



why does this make sense?

firing depends on $w_0 + w_1x_1 + w_2x_2 >= 0$

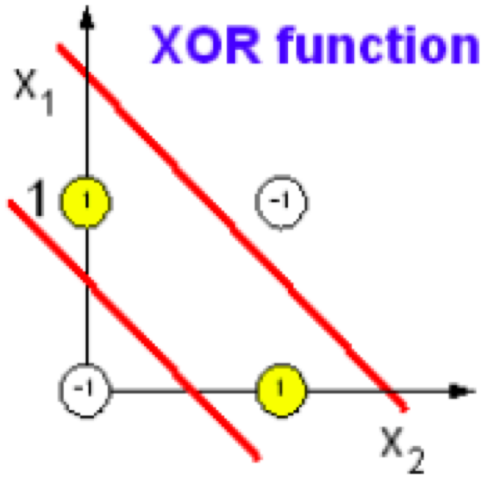
border case is when $w_0 + w_1x_1 + w_2x_2 = 0$

i.e., $x_2 = (-w_1/w_2)\, x_1 + (-w_0/w_2)$ *the equation of a line*

the training algorithm simply shifts the line around (by changing the weight) until the classes are separated

Inadequacy of perceptrons

inadequacy of perceptrons is due to the fact that many simple problems are not linearly separable

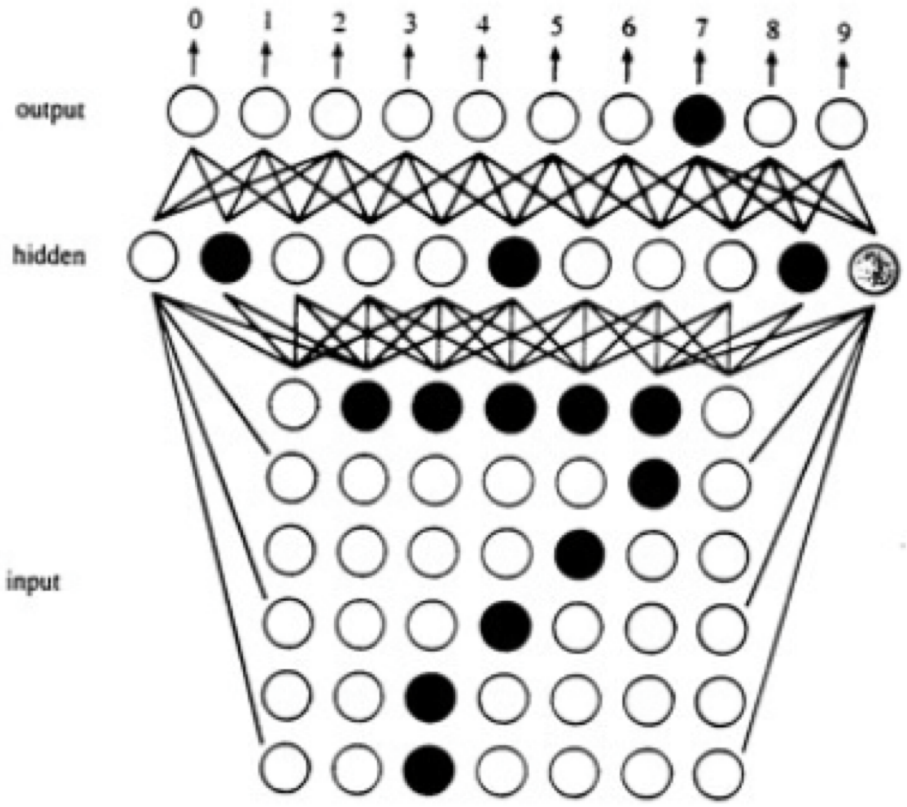


Hidden units

the addition of hidden units allows the network to develop complex feature detectors (i.e., internal representations)

e.g., Optical Character Recognition (OCR)

- perhaps one hidden unit "looks for" a horizontal bar
- another hidden unit "looks for" a diagonal
- the combination of specific hidden units indicates a 7



Hidden units & learning

every classification problem has a perceptron solution if enough hidden layers are used

- i.e., multi-layer networks can compute anything
  (recall: can simulate AND, OR, NOT gates)

expressiveness is not the problem – learning is!

- it is not known how to systematically find solutions
- the Perceptron Learning Algorithm can't adjust weights between levels

Minsky & Papert's results about the "inadequacy" of perceptrons pretty much killed neural net research in the 1970's

rebirth in the 1980's due to several developments

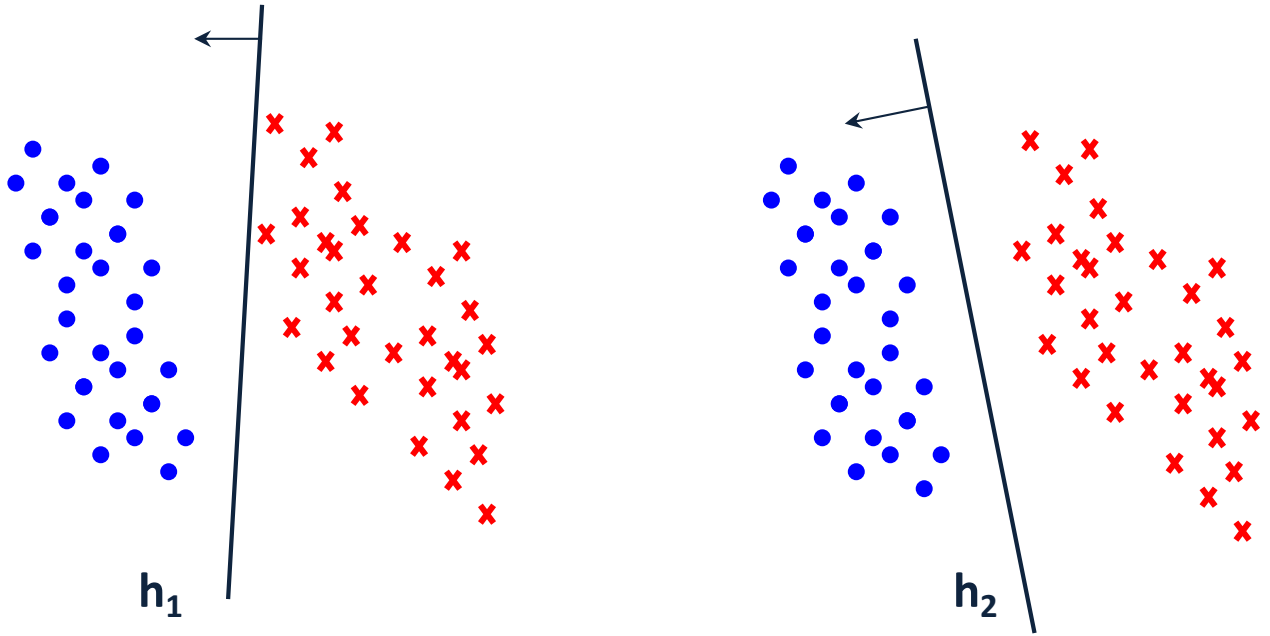
- faster, more parallel computers
- new learning algorithms e.g., backpropagation
- new architectures e.g., Hopfield nets

Support Vector Machines (SVM)

Slides from Dan Roth, UIUC

Linear Classification

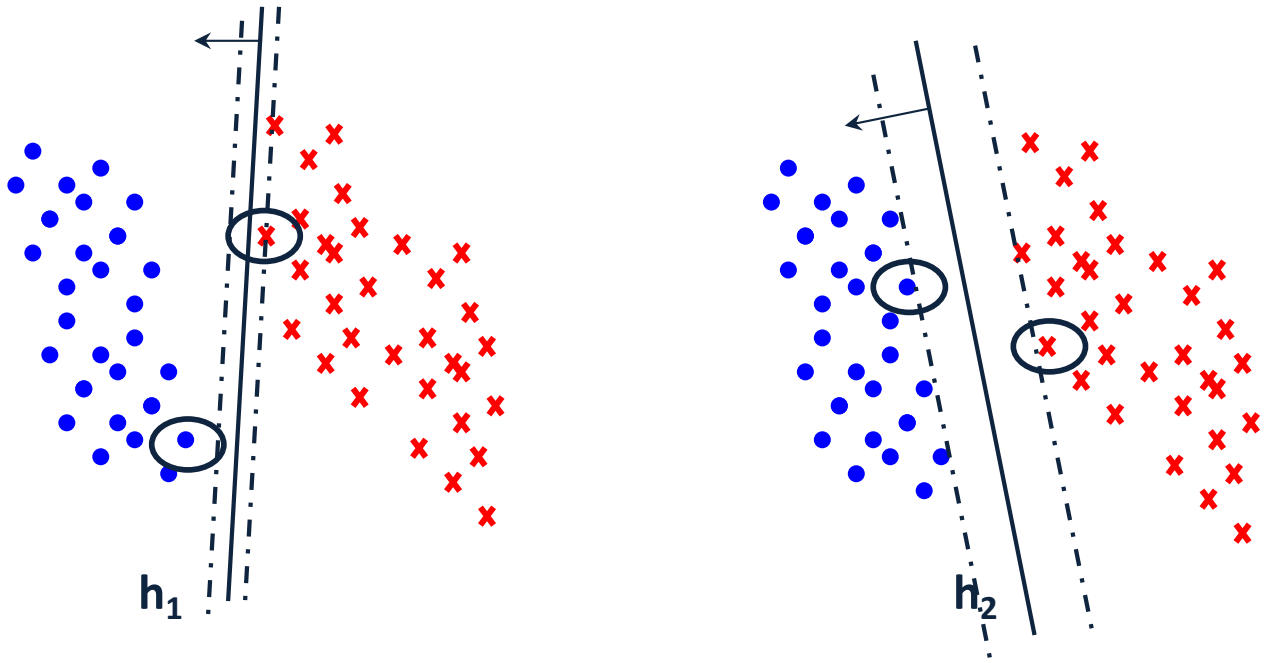
- Let $X = R^2$, $Y = \{+1, -1\}$
- Which of these classifiers would be likely to generalize better?



Slides from Dan Roth

Linear Classification

- Although both classifiers separate the data, the distance with which the separation is achieved is different:



Concept of Margin

- The margin γ_i of a point $x_i \in R^n$ with respect to a linear classifier $h(x) = \text{sign}(w^T \cdot x + b)$ is defined as the distance of x_i from the hyperplane $w^T \cdot x + b = 0$:

$$\gamma_i = |(w^T \cdot x_i + b)/||w|| \ |$$

- The margin of a set of points $\{x_1,\ldots x_m\}$ with respect to a hyperplane w, is defined as the margin of the point closest to the hyperplane:

$$\gamma = \min_i \gamma_i = \min_i |(w^T \cdot x_i + b)/ \ ||w|| \ \ |$$

Smallest (Min) Distance would be the Margin hear.

Slides from Dan Roth

Towards Max Margin Classifiers

- First observation:

  our goal could be to find a separating hyperplane w that maximizes the margin of the set S of examples.

- A second observation that drives an algorithmic approach is that:

  Small ||w|| → Large Margin

- Together, this leads to an algorithm: from among all those w's that agree with the data, find the one with the minimal size ||w||
  - But, if w separates the data, so does w/7….
  - We need to better understand the relations
  - between w and the margin

The distance between a point x and the hyperplane defined by (w; b) is: $|w^T x + b|/||w||$

Maximal Margin

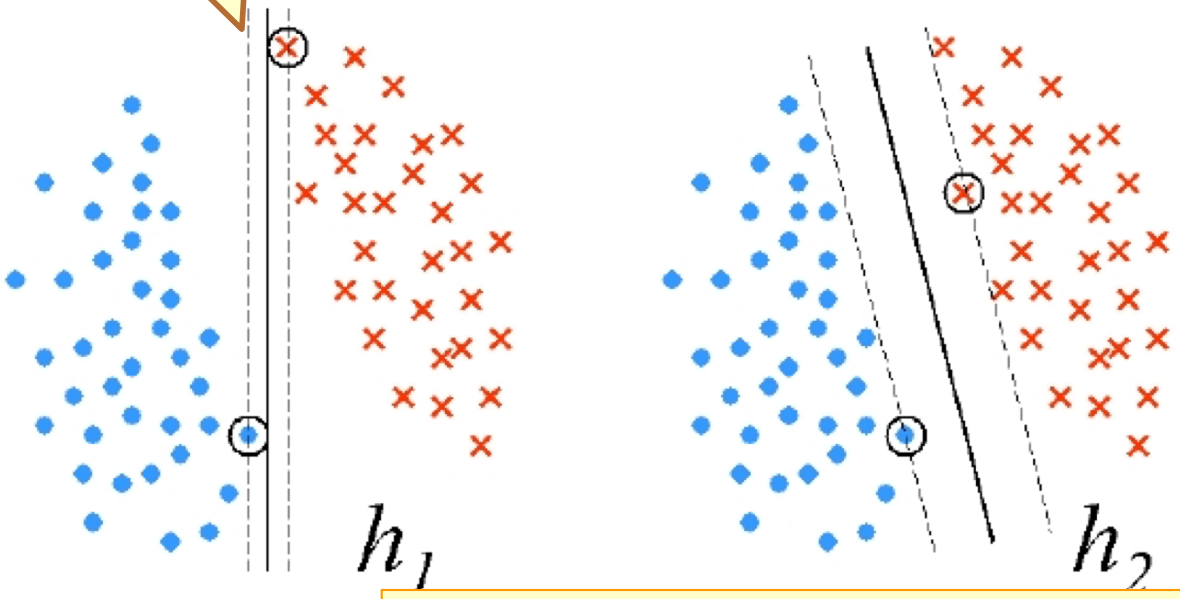
- This discussion motivates the notion of a maximal margin.
- The maximal margin of a data set S is define as:

A hypothesis (w,b) has many names

$\gamma(S) = \max_{||w||=1} \min_{(x,y) \in S} |y\, w^T x|$

1. For a given w: Find the closest point.
2. Then, find the one the gives the maximal margin value across all w's (of size 1).

Note: the selection of the point is in the min and therefore the max does not change if we scale w, so it's okay to only deal with normalized w's.



Interpretation 1: among all w's, choose the one that maximizes the margin.

How does it help us to derive these h's?

$\text{argmax}_{||w||=1} \min_{(x,y) \in S} |y\, w^T x|$

From Margin to ||W||

- We want to choose the hyperplane that achieves the largest margin. That is, given a data set S, find:
  - $w^* = \text{argmax}_{||w||=1} \min_{(x,y) \in S} |y\, w^T x|$
- How to find this w^*?
- Claim: Define w_0 to be the solution of the optimization problem:

  $w_0 = \text{argmin} \{||w||^2 : \forall (x,y) \in S, y\, w^T x \geq 1\}$.

  Then:

  $w_0/||w_0|| = \text{argmax}_{||w||=1} \min_{(x,y) \in S} y\, w^T x$

Interpretation 2: among all w's that separate the data with margin 1, choose the one with minimal size.

That is, the normalization of w_0 corresponds to the largest margin separating hyperplane.

Slides from Dan Roth

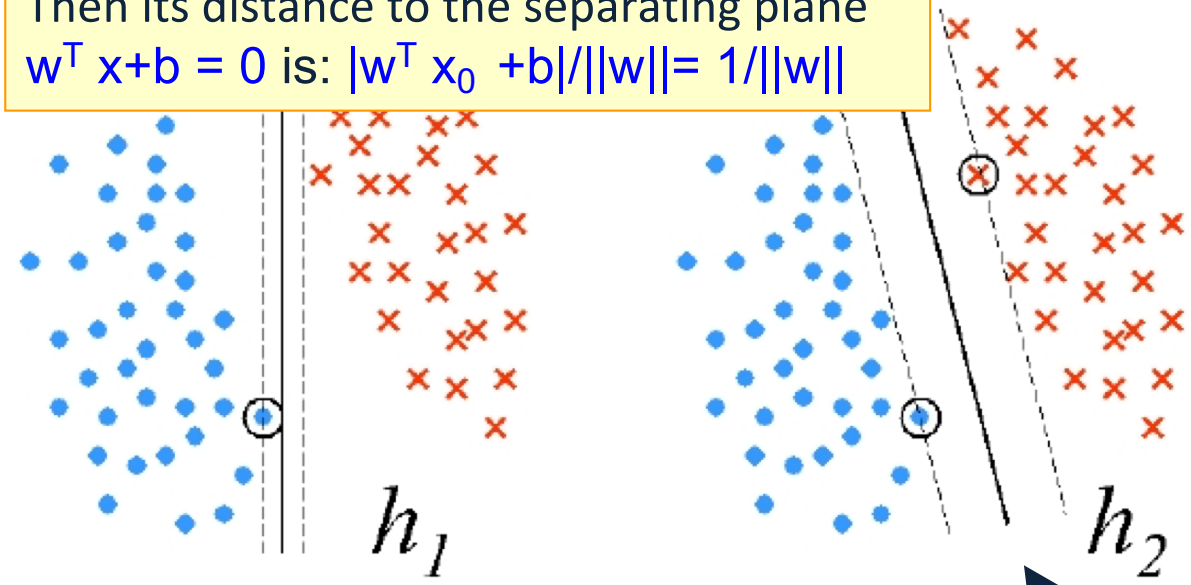
Margin of a Separating Hyperplane

- A separating hyperplane: $w^T x+b = 0$

Assumption: data is linearly separable
Let (x_0, y_0) be a point on $w^T x+b = 1$
Then its distance to the separating plane $w^T x+b = 0$ is: $|w^T x_0 +b|/||w|| = 1/||w||$

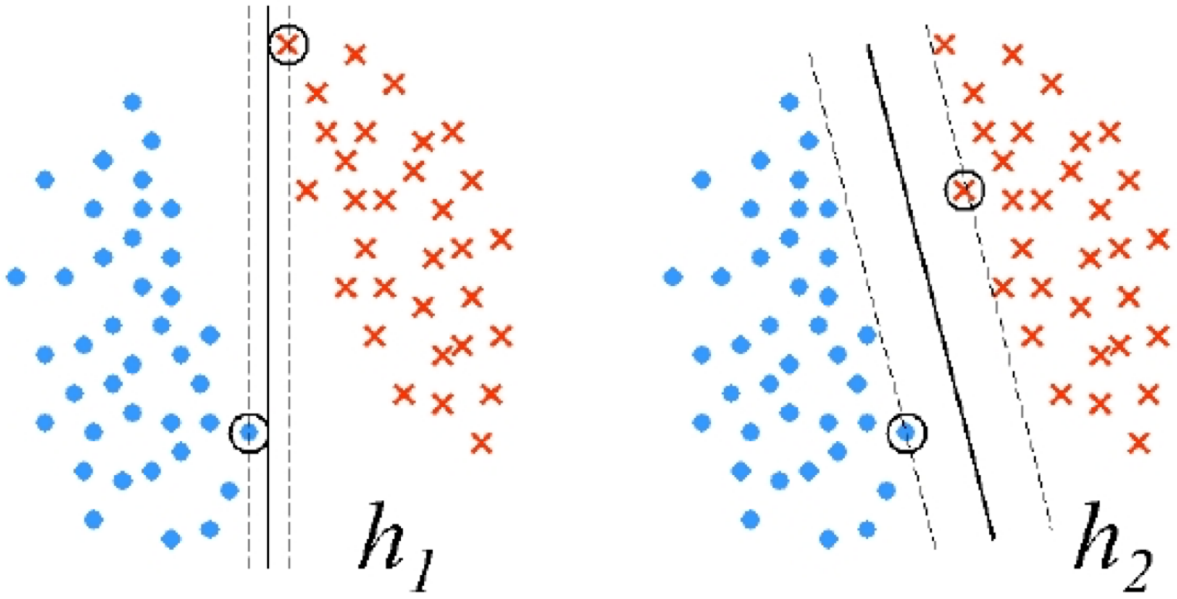
Distance between
$w^T x+b = +1$ and -1 is $2 / ||w||$
What we did:

1. Consider all possible w with different angles
2. Scale w such that the constraints are tight
3. Pick the one with largest margin/minimal size



$w^T x+b = 0$
$w^T x+b = -1$

Maximal Margin



The margin of a linear separator
$w^T x+b = 0$ is $2 / ||w||$

max $2 / ||w||$ = min $||w||$
= min ½ $w^T w$

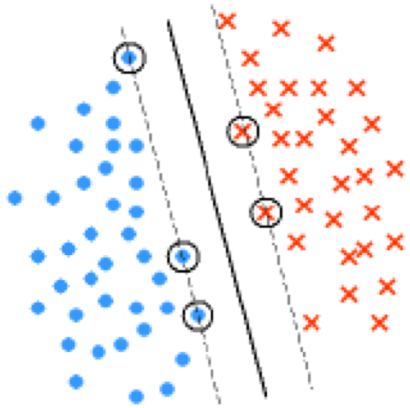
$$\min_{w,b} \quad \frac{1}{2} w^T w$$

$$\text{s.t} \quad y_i(w^T x_i + b) \geq 1, \forall (x_i, y_i) \in S$$

Slides from Dan Roth

Support Vector Machines

- The name "Support Vector Machine" stems from the fact that w* is supported by (i.e. is the linear span of) the examples that are exactly at a distance 1/||w*|| from the separating hyperplane. These vectors are therefore called support vectors.



This representation should ring a bell…

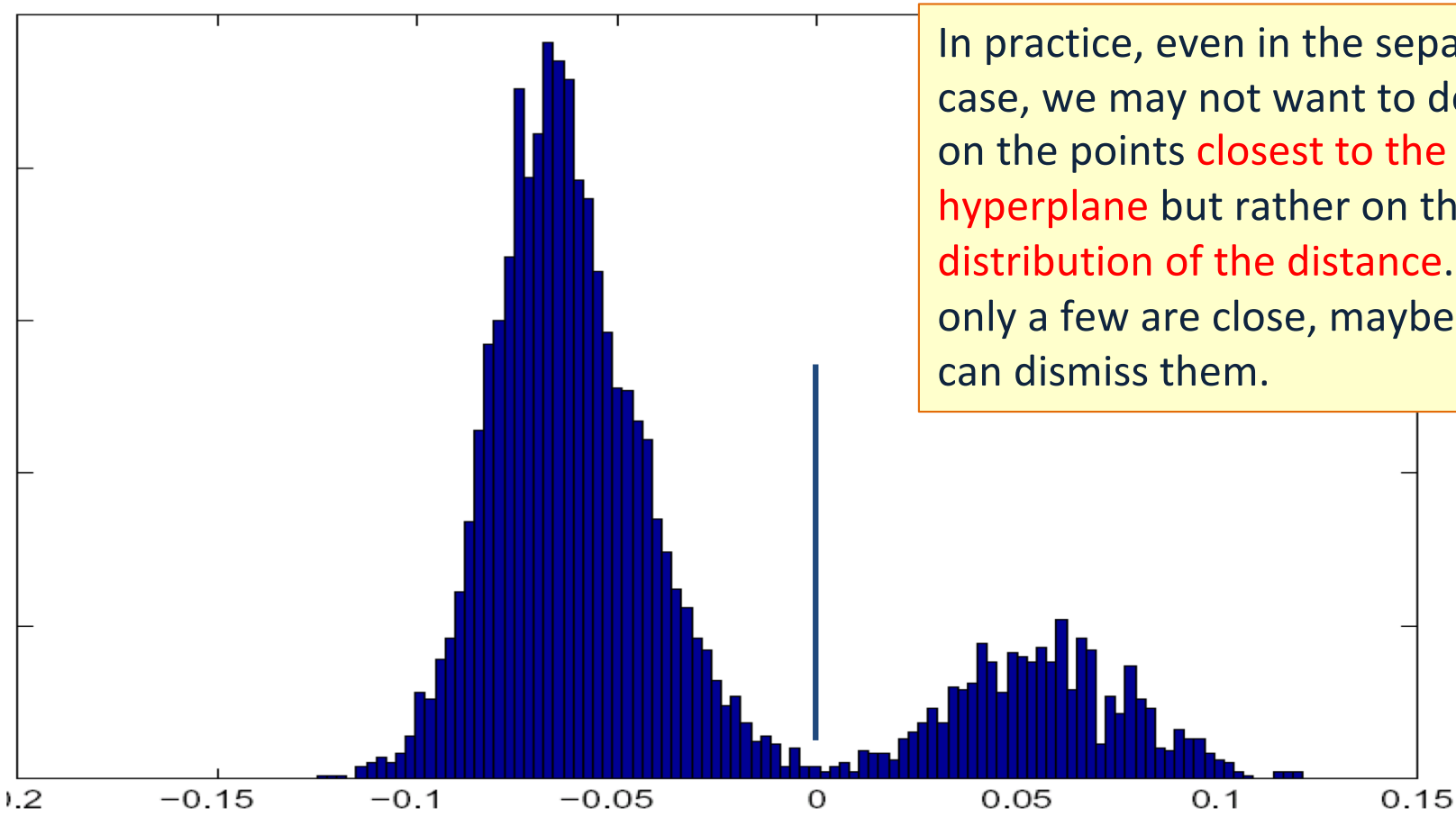
Key Issues

- Computational Issues
  - Training of an SVM used to be is very time consuming – solving quadratic program.
  - Modern methods are based on Stochastic Gradient Descent and Coordinate Descent and are much faster.

- Is it really optimal?
  - Is the objective function we are optimizing the “right” one?

Real Data

17,000 dimensional context sensitive spelling
Histogram of distance of points from the hyperplane



In practice, even in the separable case, we may not want to depend on the points closest to the hyperplane but rather on the distribution of the distance. If only a few are close, maybe we can dismiss them.

Slides from Dan Roth

Soft SVM

- The hard SVM formulation assumes linearly separable data.
- A natural relaxation:
  - maximize the margin while minimizing the # of examples that violate the margin (separability) constraints.
- However, this leads to non-convex problem that is hard to solve.
- Instead, we relax in a different way, that results in optimizing a surrogate loss function that is convex.

Soft SVM

- Notice that the relaxation of the constraint:

$$y_i w^T x_i \geq 1$$

- Can be done by introducing a slack variable ξ_i (per example) and requiring:

$$y_i w^T x_i \geq 1 - \xi_i \; ; \xi_i \geq 0$$

- Now, we want to solve:

$$\min_{w,\xi_i} \quad \frac{1}{2} w^T w + C \sum_i \xi_i$$

$$\text{s.t} \quad y_i w^T x_i \geq 1 - \xi_i \; ; \xi_i \geq 0 \quad \forall i$$

A large value of C means that misclassifications are bad – we focus on a small training error (at the expense of margin). A small C results in more training error, but hopefully better true error.

SVM Objective Function

- The problem we solved is:

$$\text{Min } ½ \, ||w||^2 + c \sum \xi_i$$

- Where $\xi_i > 0$ is called a slack variable, and is defined by:
  - $\xi_i = \max(0, 1 - y_i \, w^t x_i)$
  - Equivalently, we can say that: $y_i \, w^t x_i \geq 1 - \xi_i$; $\xi_i \geq 0$
- And this can be written as:

$$\text{Min } ½ \, ||w||^2 \quad + \quad c \sum \xi_i$$

$$\min_{\mathbf{w},b} \underbrace{\mathbf{w}^T\mathbf{w}}_{l_2-regularizer} + C \sum_{i=1}^{n} \underbrace{\max\left[1 - y_i(\mathbf{w}^T\mathbf{x} + b), 0\right]}_{hinge-loss}$$

SVM Objective Function

- The problem we solved is:

$$\text{Min } \tfrac{1}{2} ||w||^2 + c \sum \xi_i$$

- Where $\xi_i > 0$ is called a slack variable, and is defined by:
  - $\xi_i = \max(0, 1 - y_i w^t x_i)$
  - Equivalently, we can say that: $y_i w^t x_i \geq 1 - \xi_i; \xi_i \geq 0$
- And this can be written as:

$$\underbrace{\text{Min } \tfrac{1}{2} ||w||^2}_{\text{Regularization term}} + \underbrace{c \sum \xi_i}_{\text{Empirical loss}}$$

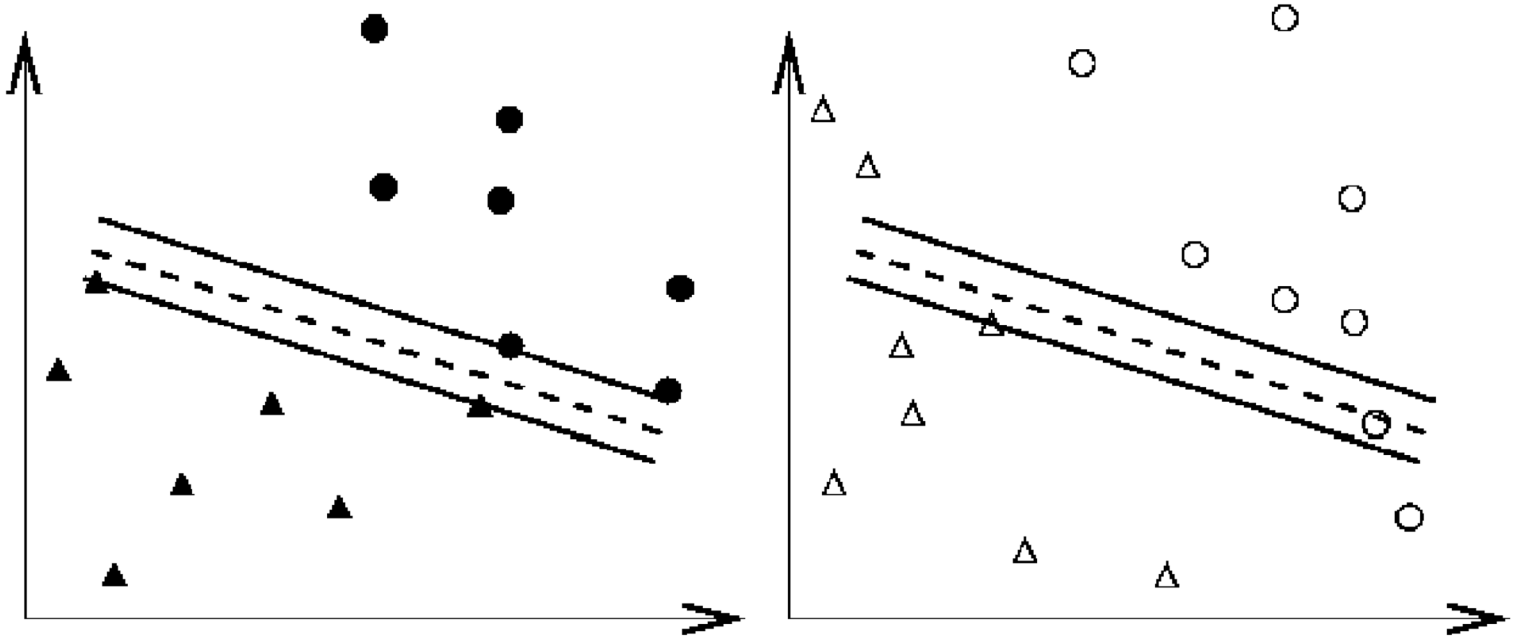
Can be replaced by other **regularization functions**

Can be replaced by other **loss functions**

- General Form of a learning algorithm:
  - Minimize empirical loss, and Regularize (to avoid over fitting)
  - Theoretically motivated improvement over the original algorithm we've seen at the beginning of the semester.

Slides from Dan Roth

Balance between regularization and empirical loss

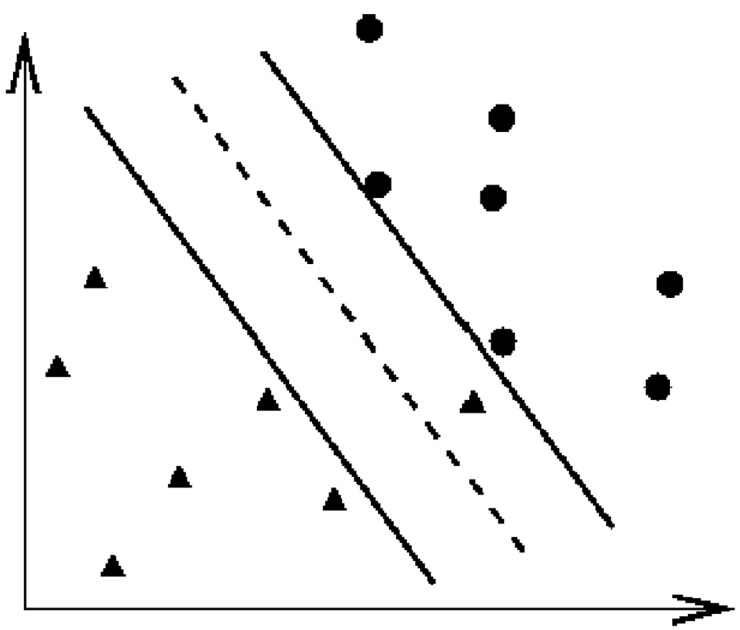


(a) Training data and an over-fitting classifier

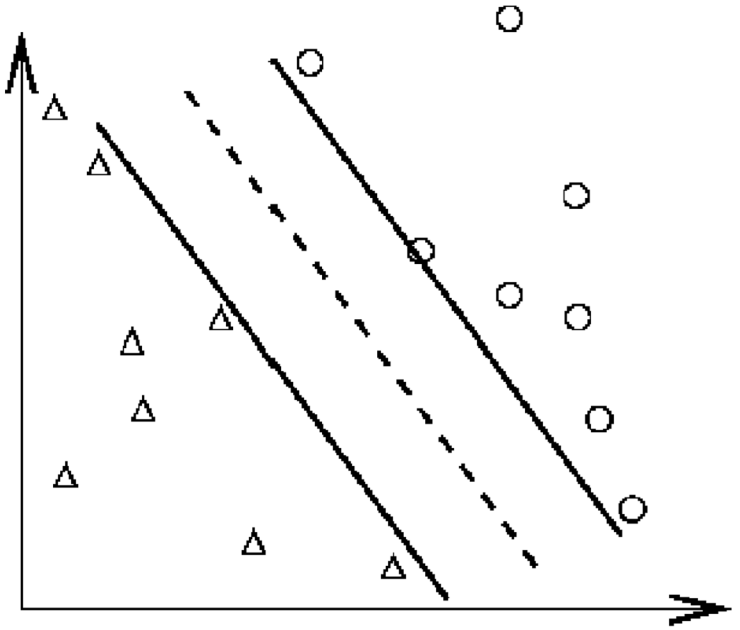
(b) Testing data and an over-fitting classifier

Slides from Dan Roth

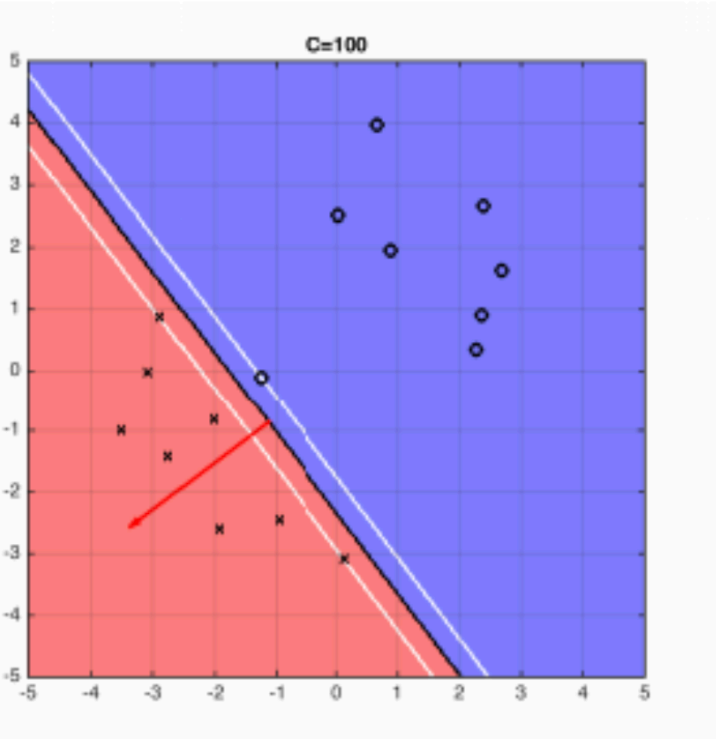
Balance between regularization and empirical loss

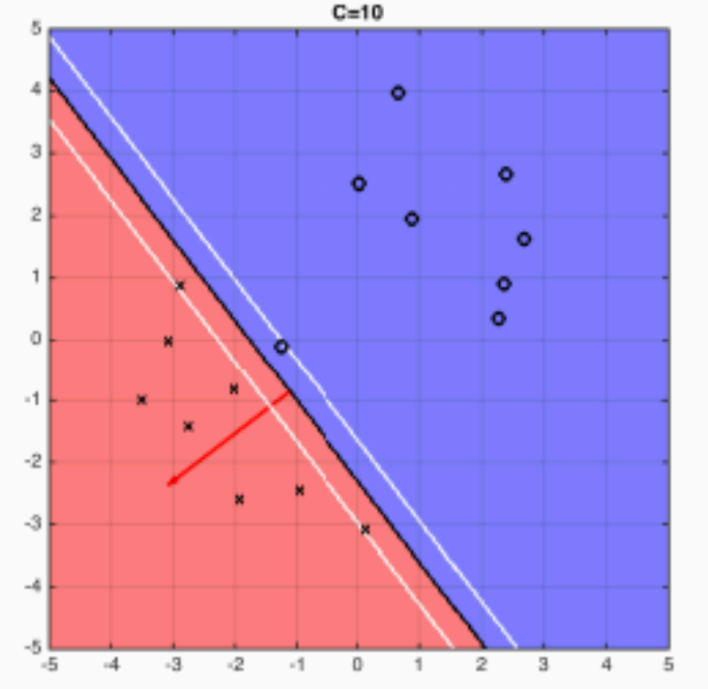


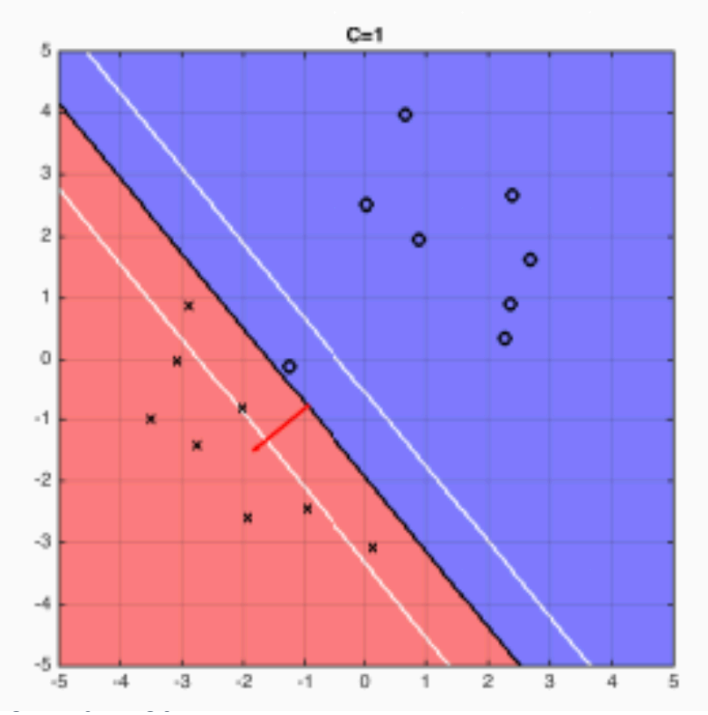
(c) Training data and a better classifier

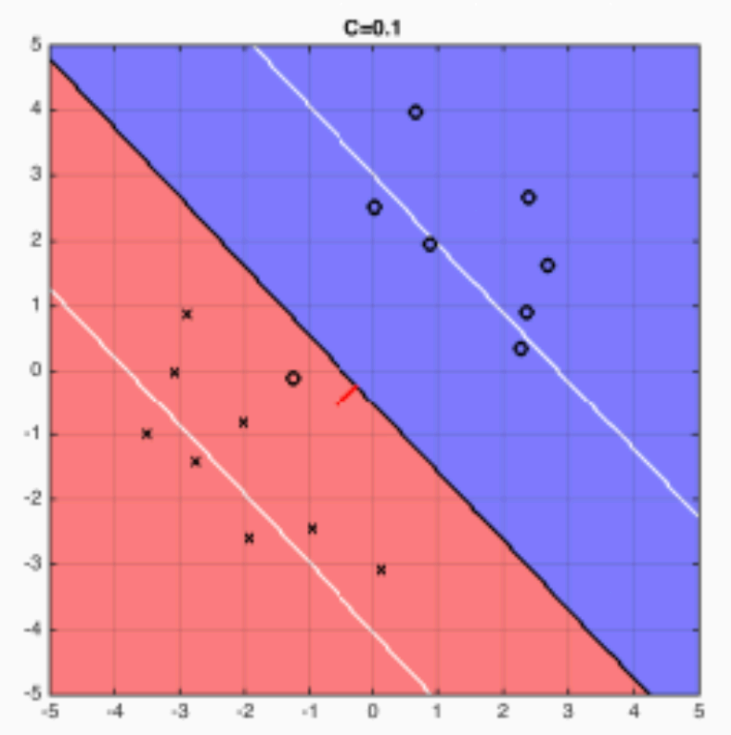


(d) Testing data and a better classifier



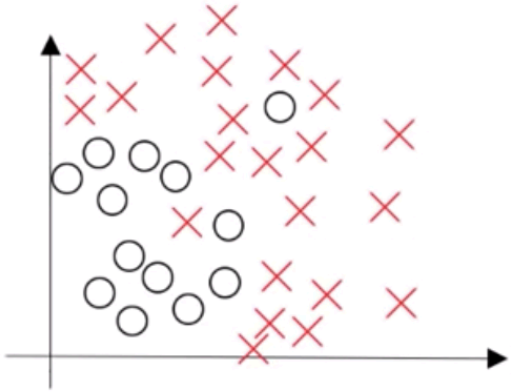




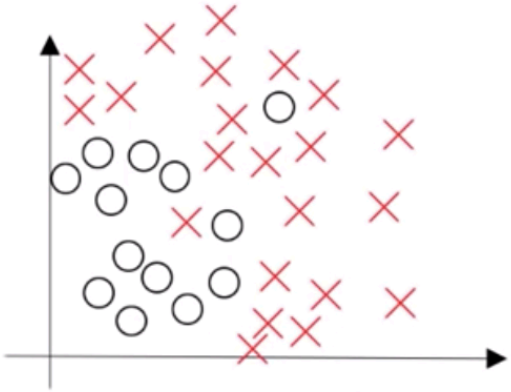


Slides f

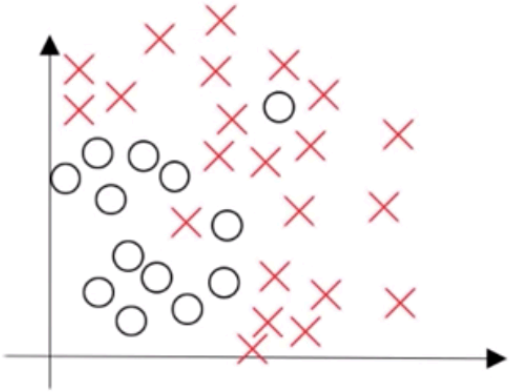
Bias and Variance



high bias



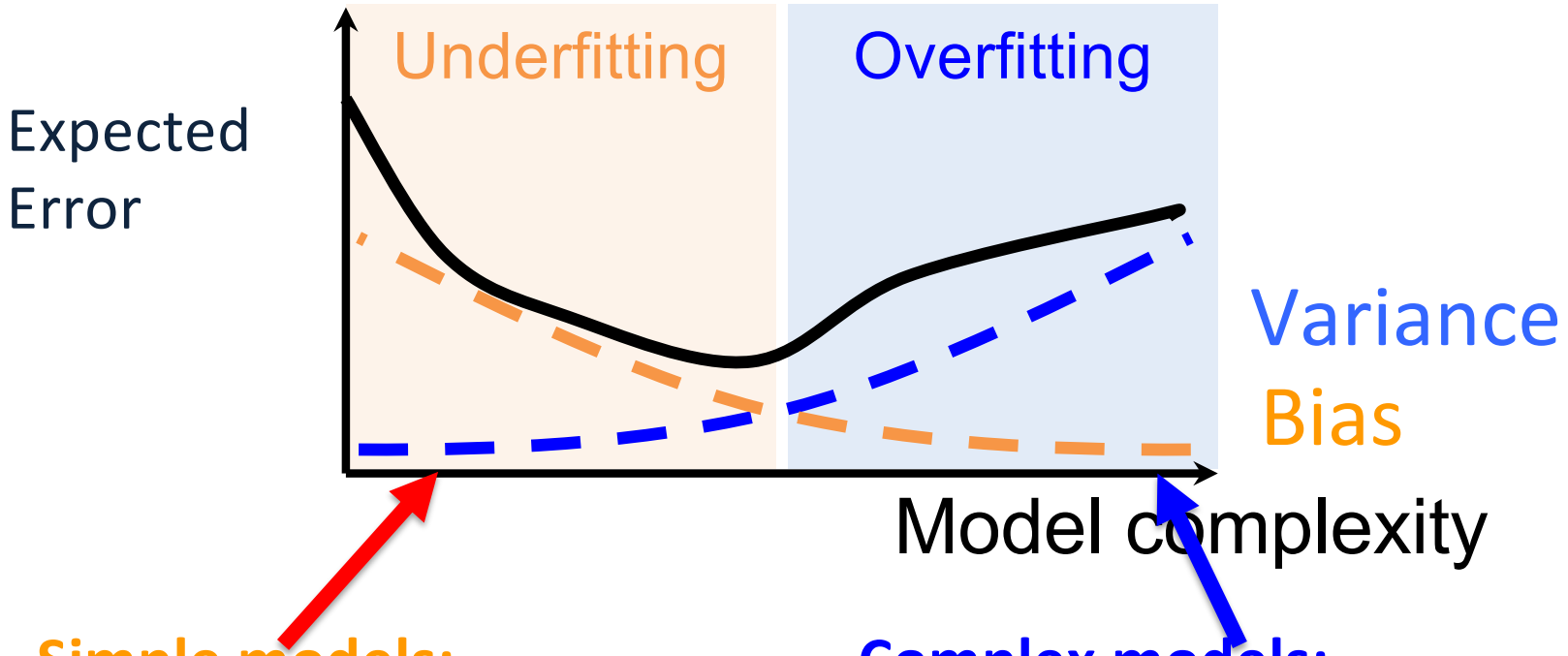
"just right"



high variance

Slides from Dan Roth

Underfitting and Overfitting



- **Simple models:**
  High bias and low variance

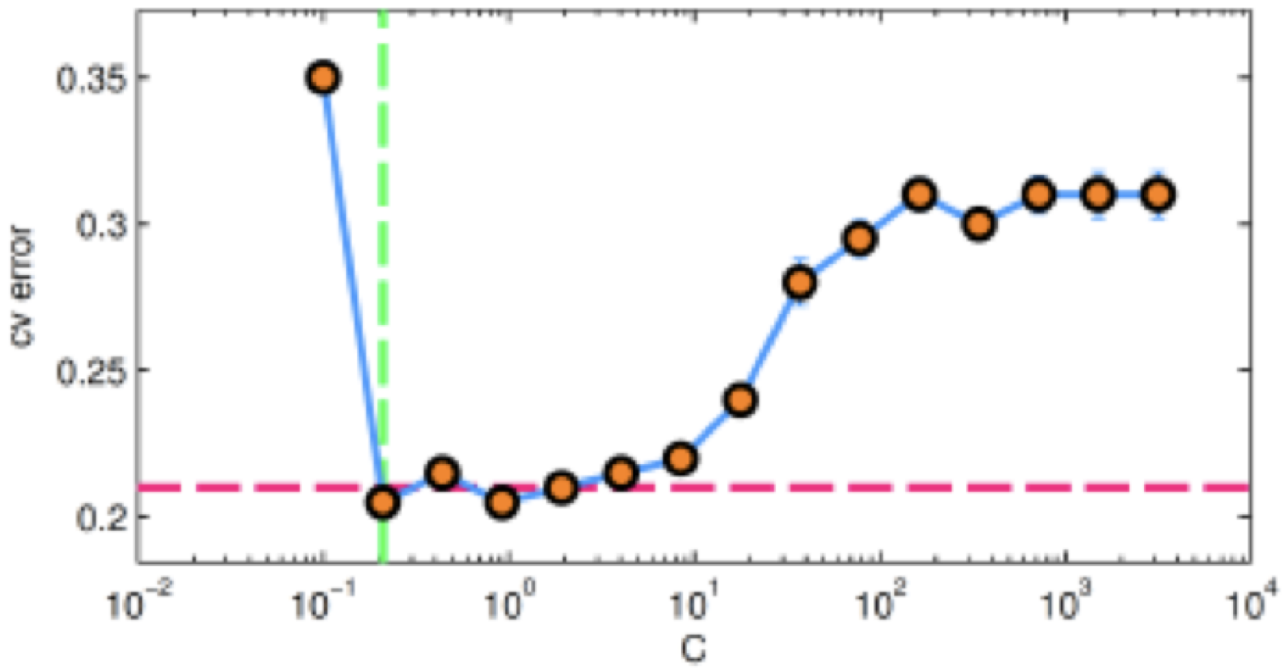
- **Smaller C**

Complex models:
High variance and low bias

Larger C

- We can use cross-validation to choose the best C

- The larger the value of C, the slower the training is:

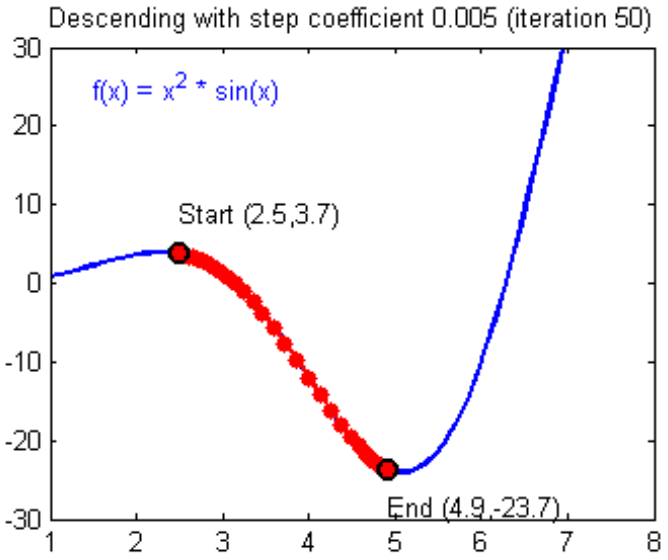


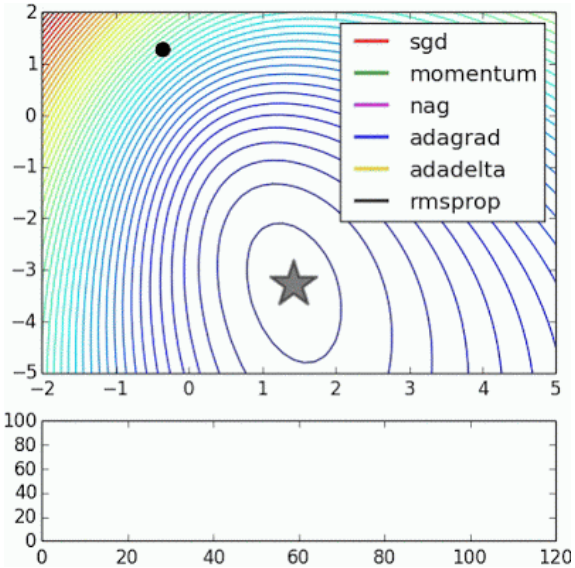
Slides from Dan Roth

Optimization: How to Solve

- 1. Earlier methods used Quadratic Programming. Very slow.
- 2. The soft SVM problem is an unconstrained optimization problems. It is possible to use the gradient descent algorithm.
- Many options within this category:
  - Iterative scaling; non-linear conjugate gradient; quasi-Newton methods; truncated Newton methods; trust-region newton method.
  - All methods are iterative methods, that generate a sequence w_k that converges to the optimal solution of the optimization problem above.
  - Currently: Limited memory BFGS is very popular
- 3. 3rd generation algorithms are based on Stochastic Gradient Decent
  - The runtime does not depend on n=#(examples); advantage when n is very large.
  - Stopping criteria is a problem: method tends to be too aggressive at the beginning and reaches a moderate accuracy quite fast, but it's convergence becomes slow if we are interested in more accurate solutions.
- 4. Dual Coordinated Descent (& Stochastic Version)

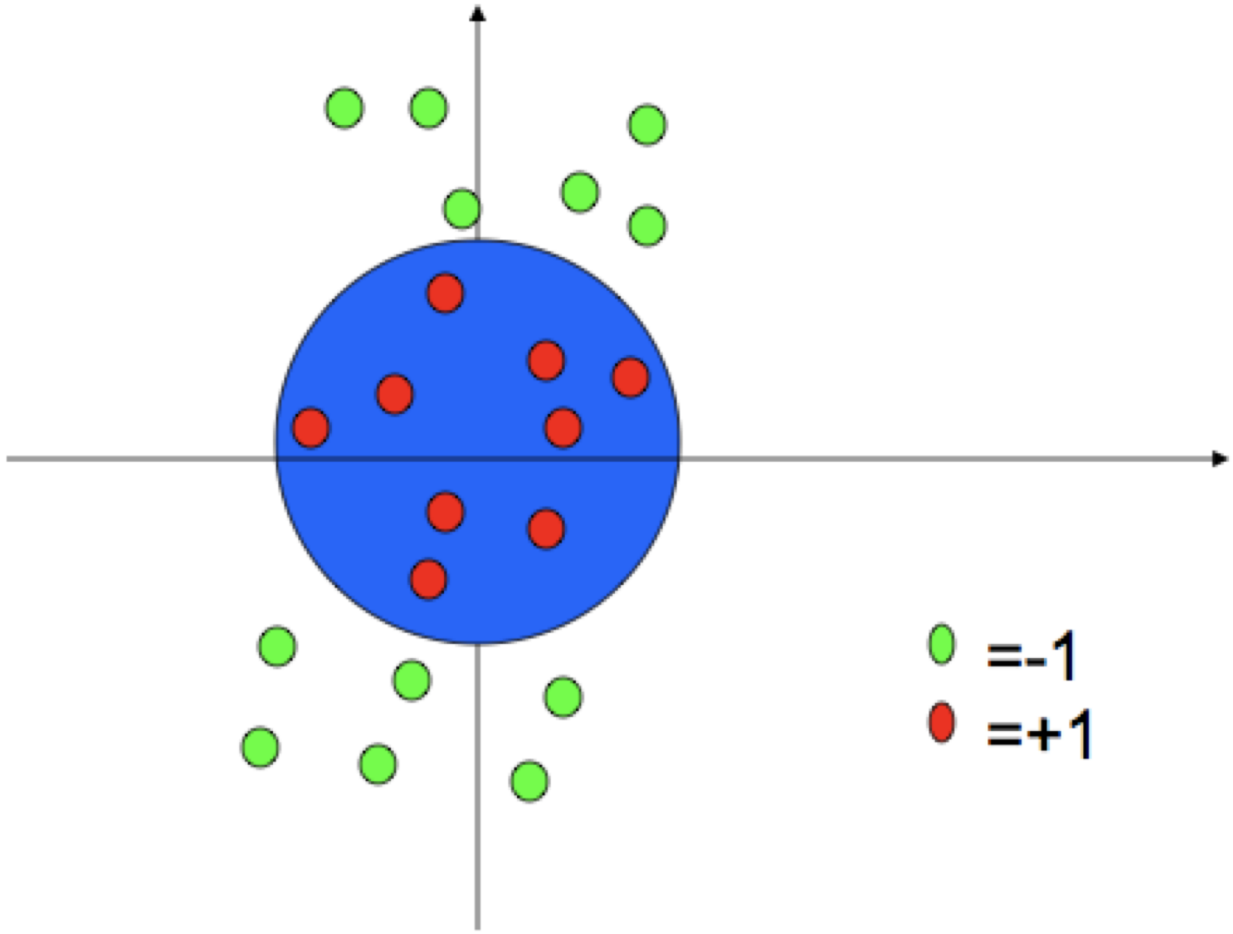
Demo



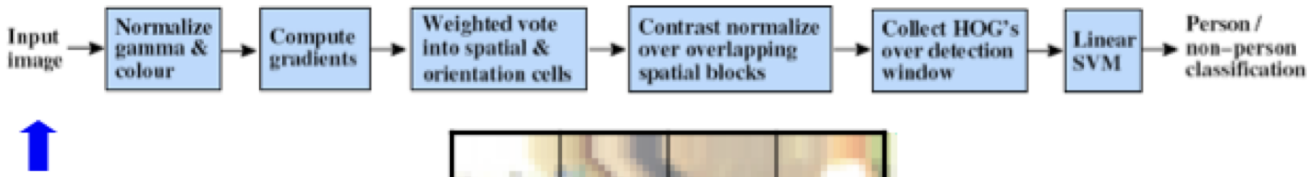


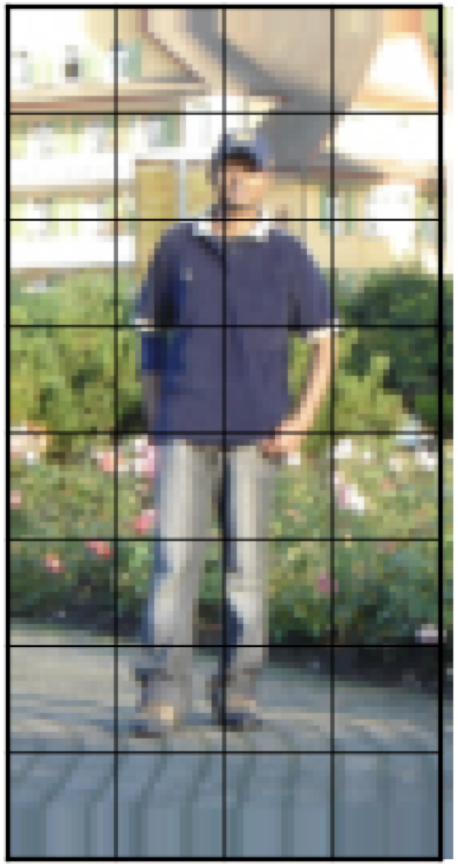
Slides from Dan Roth

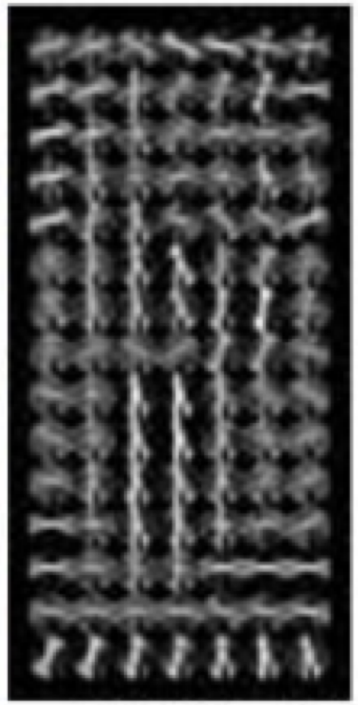
Problems with linear SVM



What if the decision function is not a linear?

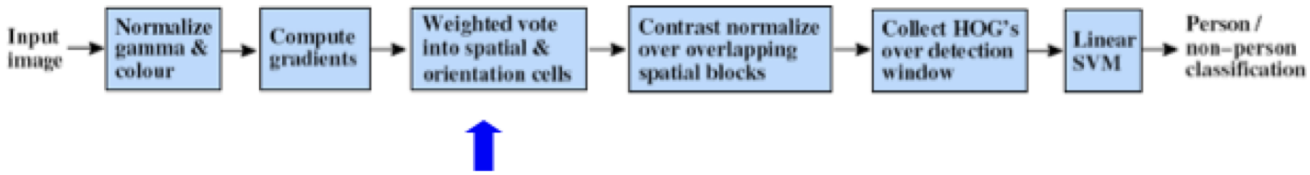




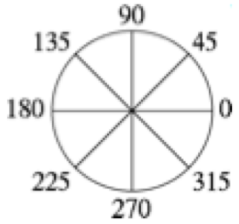


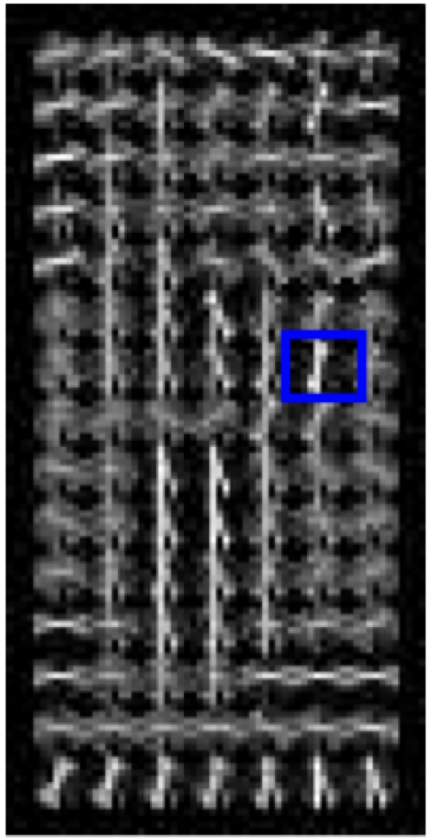
Slides by Pete Barnum

Navneet Dalal and Bill Triggs, Histograms of Oriented Gradients for Human Detection, CVPR05



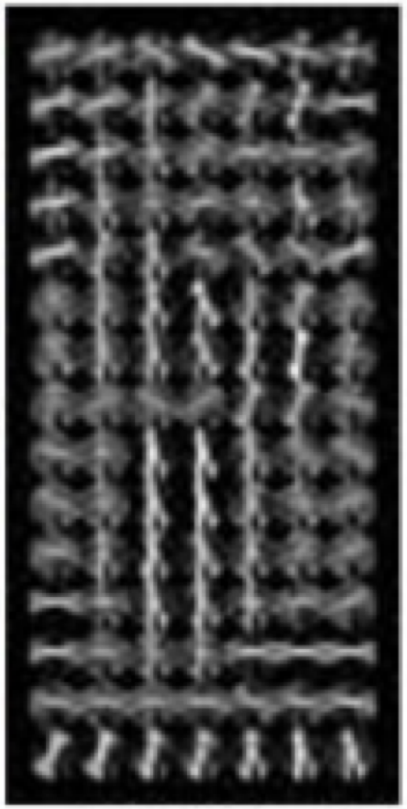
- Histogram of gradient orientations
  - -Orientation





Slides by Pete Barnum

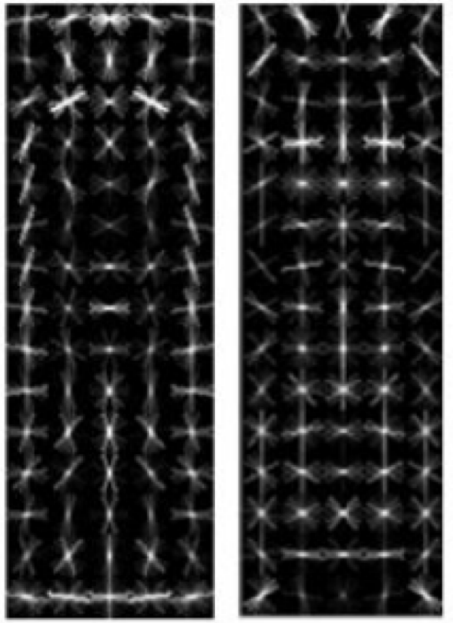
Navneet Dalal and Bill Triggs, Histograms of Oriented Gradients for Human Detection, CVPR05



$$0.16 = w^T x - b$$

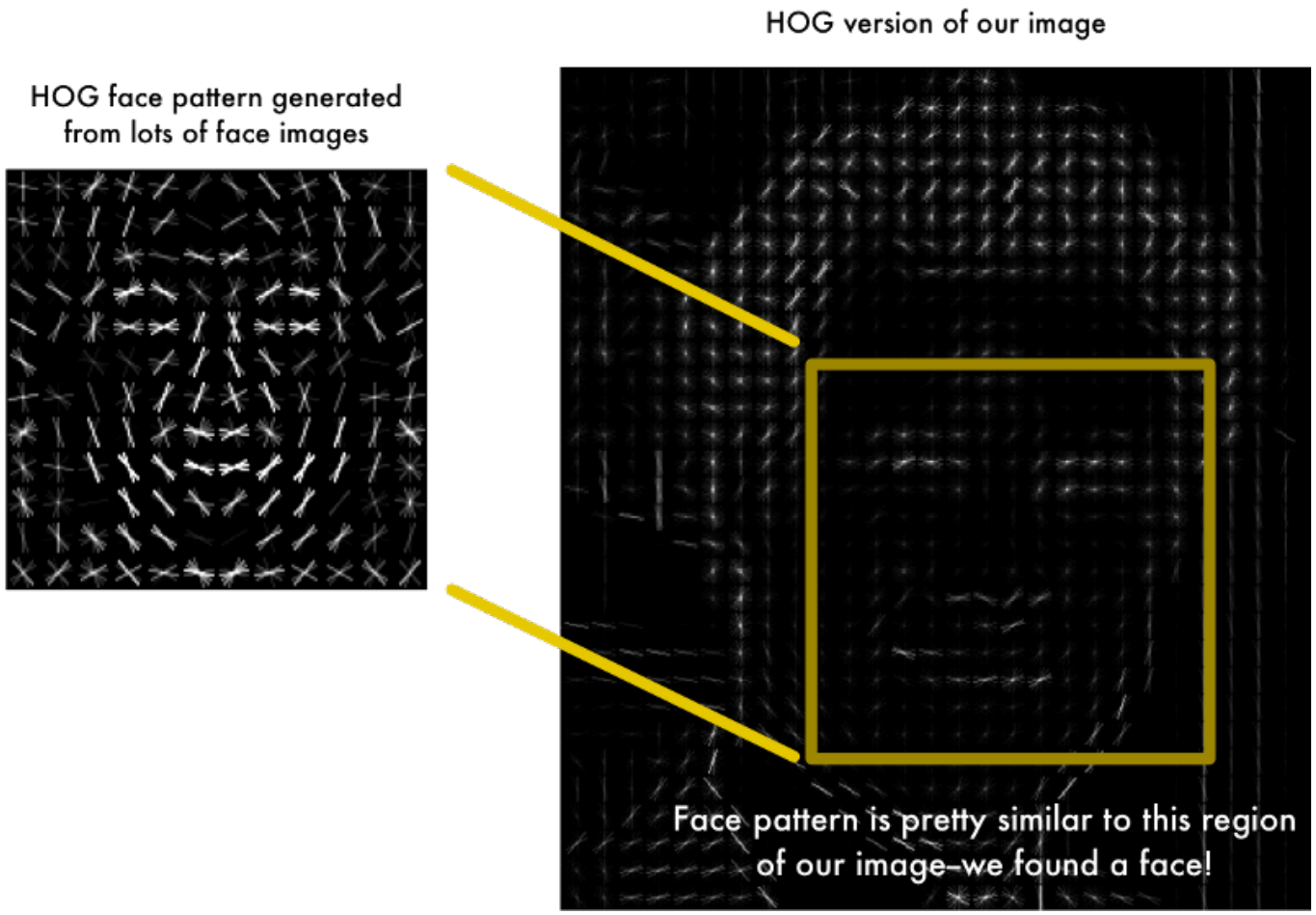
$$sign(0.16) = 1$$

$$=> \quad \text{pedestrian}$$

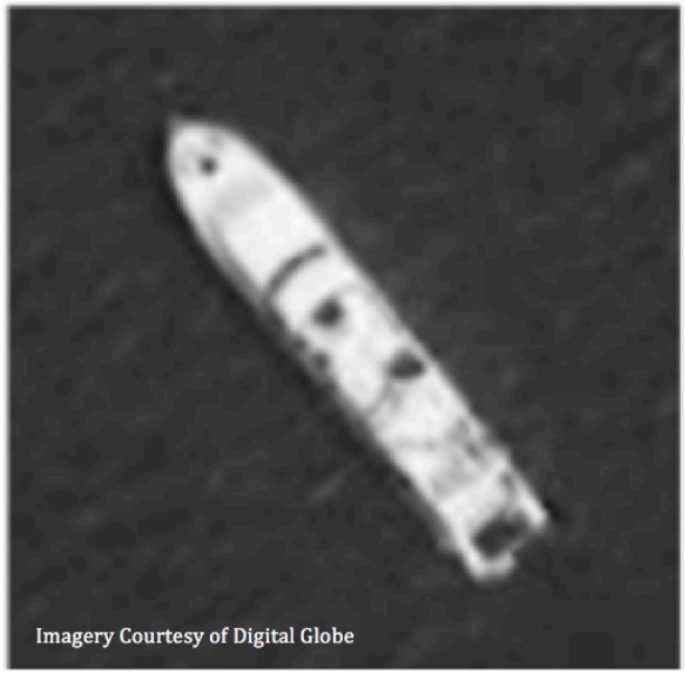


Slides by Pete Barnum

Navneet Dalal and Bill Triggs, Histograms of Oriented Gradients for Human Detection, CVPR05



Slides from Dan Roth



Histogram of Oriented Gradients (HOG) Boat Heading Classification

Slides from Dan Roth

What Do We Optimize?

- Logistic Regression

$$\min_w \frac{1}{2} w^T w + C \sum_{i=1}^{l} \log(1 + e^{-y_i(w^T x_i)})$$

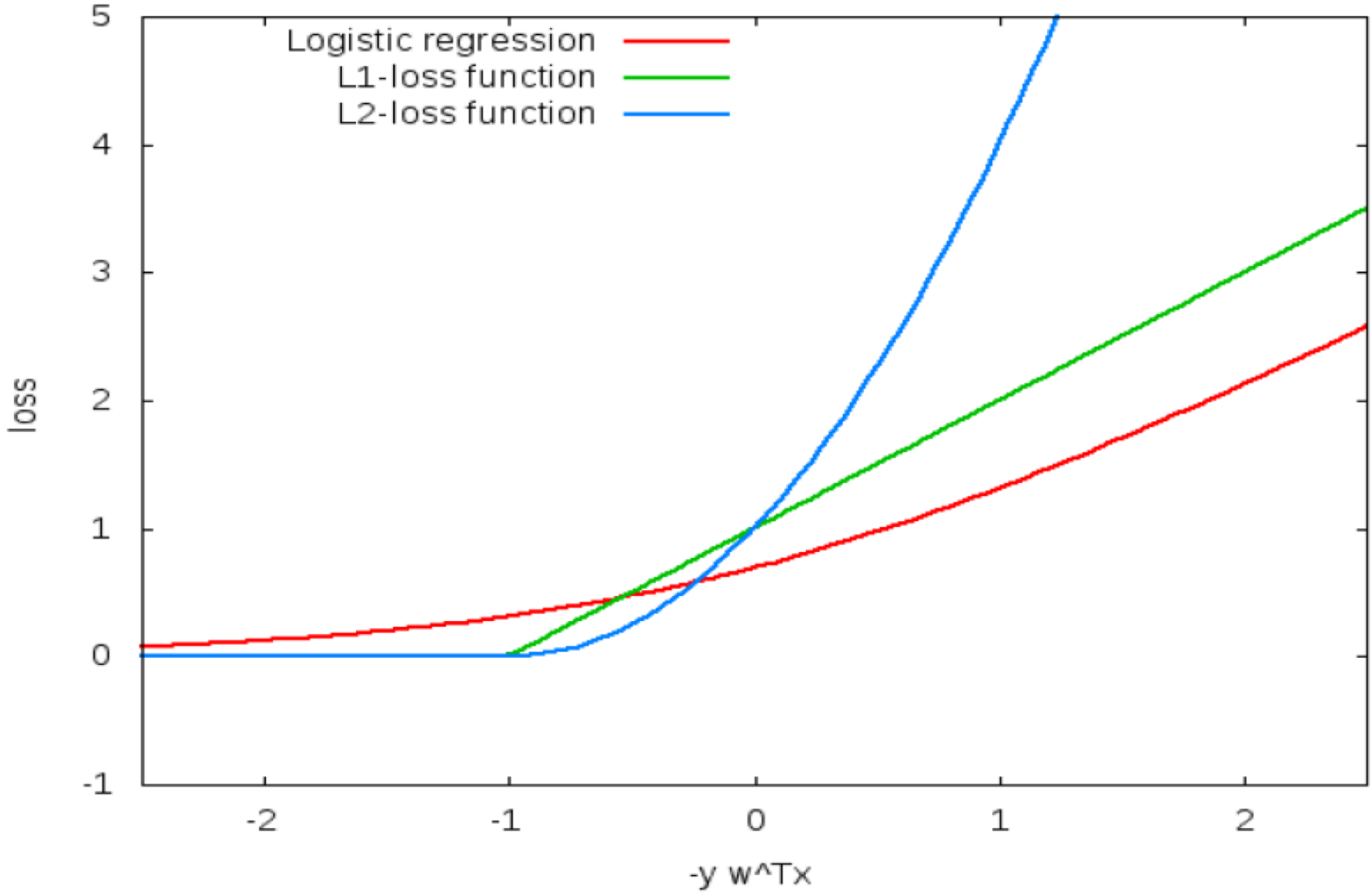
- L1-loss SVM

$$\min_w \frac{1}{2} w^T w + C \sum_{i=1}^{l} max(0, 1 - y_i w^T x_i)$$

- L2-loss SVM

$$\min_w \frac{1}{2} w^T w + C \sum_{i=1}^{l} max(0, 1 - y_i w^T x_i)^2$$

What Do We Optimize(2)?



Commonly Used Binary Classification Loss Functions

Different Machine Learning algorithms use different loss functions; Table 4.1 shows just a few:

Loss $\ell(h_{\mathbf{w}}(\mathbf{x}_i, y_i))$	Usage	Comments
Hinge-Loss $\max\left[1 - h_{\mathbf{w}}(\mathbf{x}_i)y_i, 0\right]^p$	• Standard SVM($p = 1$) • (Differentiable) Squared Hingeless SVM ($p = 2$)	When used for Standard SVM, the loss function denotes the size of the margin between linear separator and its closest points in either class. Only differentiable everywhere with $p = 2$.
Log-Loss $\log(1 + e^{-h_{\mathbf{w}}(\mathbf{x}_i)y_i})$	Logistic Regression	One of the most popular loss functions in Machine Learning, since its outputs are well-calibrated probabilities.

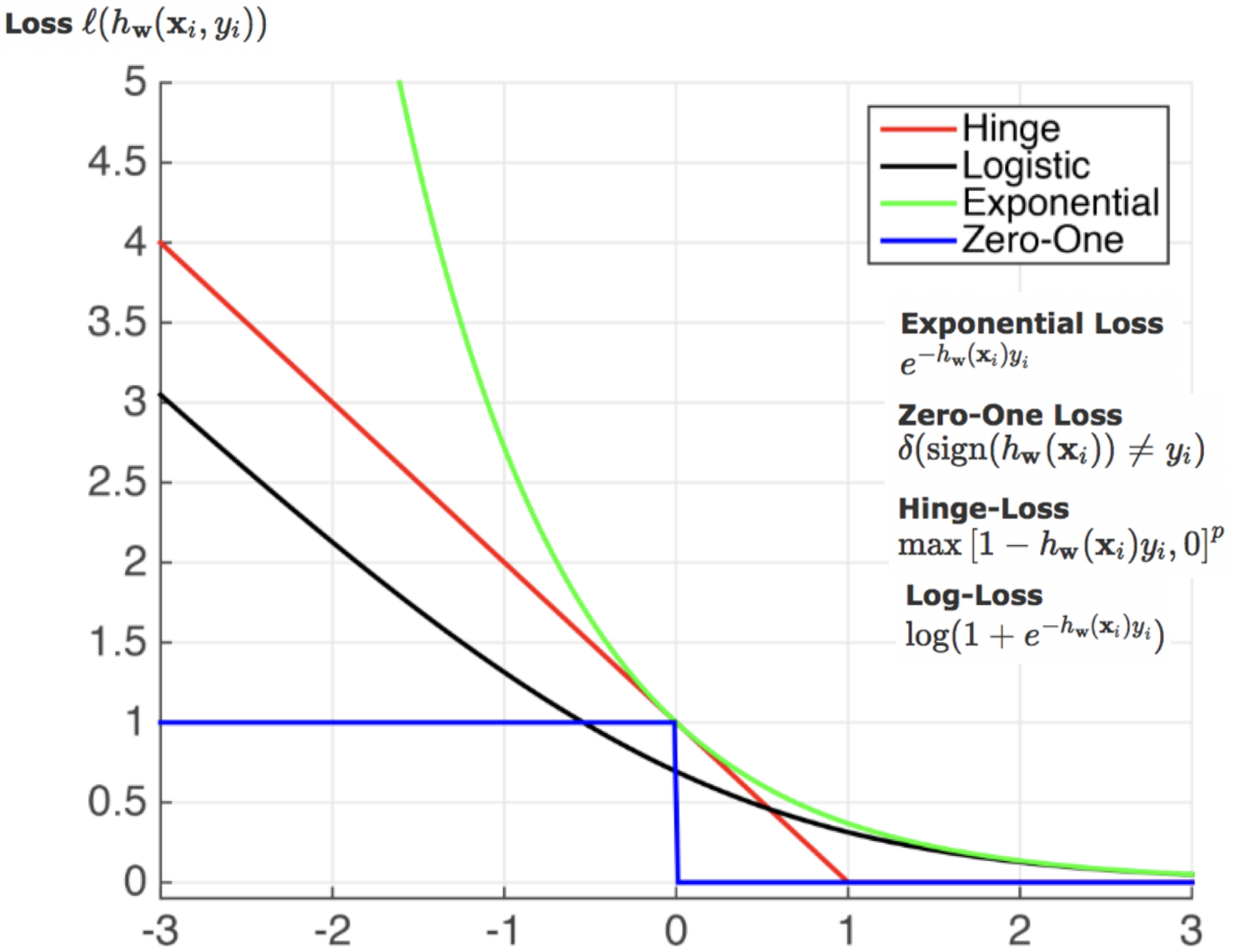
Slides from Dan Roth

Loss Functions With Classification $y \in \{-1, +1\}$

Commonly Used Binary Classification Loss Functions

Exponential Loss $e^{-h_{\mathbf{w}}(\mathbf{x}_i)y_i}$	AdaBoost	This function is very aggressive. The loss of a mis-prediction increases *exponentially* with the value of $-h_{\mathbf{w}}(\mathbf{x}_i)y_i$. This can lead to nice convergence results, for example in the case of Adaboost, but it can also cause problems with noisy data.
Zero-One Loss $\delta(\text{sign}(h_{\mathbf{w}}(\mathbf{x}_i)) \neq y_i)$	Actual Classification Loss	Non-continuous and thus impractical to optimize.

Slides from Dan Roth



Slides from Dan Roth

Commonly Used Regression Loss Functions

Regression algorithms (where a prediction can lie anywhere on the real-number line) also have their own host of loss functions:

Loss $\ell(h_{\mathbf{w}}(\mathbf{x}_i, y_i))$	Comments
Squared Loss $(h(\mathbf{x}_i) - y_i)^2$	◦ Most popular regression loss function ◦ Estimates Mean Label ◦ ADVANTAGE: Differentiable everywhere ◦ DISADVANTAGE: Somewhat sensitive to outliers/noise ◦ Also known as Ordinary Least Squares (OLS)
Absolute Loss $\lvert h(\mathbf{x}_i) - y_i \rvert$	◦ Also a very popular loss function ◦ Estimates Median Label ◦ ADVANTAGE: Less sensitive to noise ◦ DISADVANTAGE: Not differentiable at 0

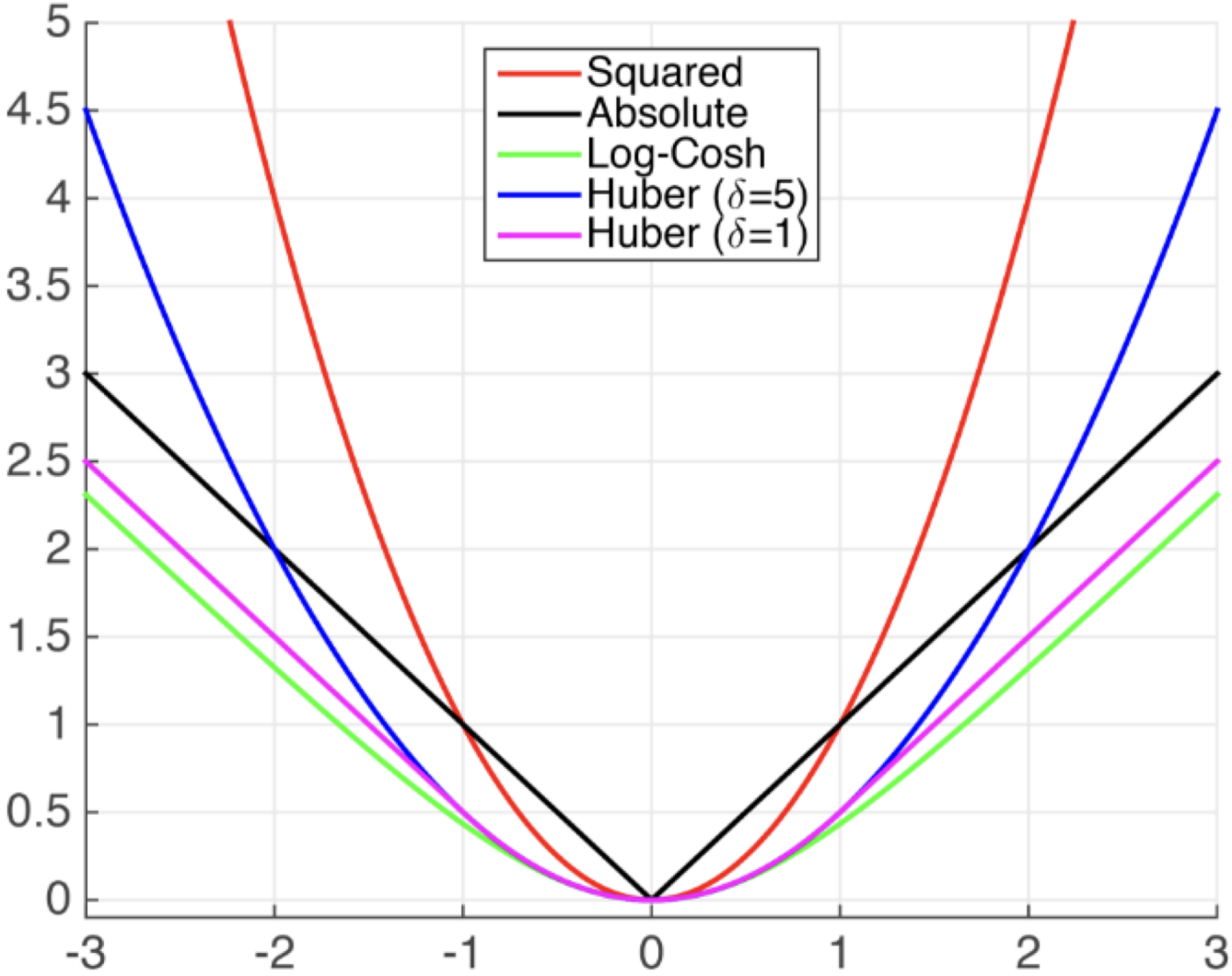
Slides from Dan Roth

Loss Functions With Regression, i.e. $y \in \mathbb{R}$

Huber Loss ◦ $\frac{1}{2}(h(\mathbf{x}_i) - y_i)^2$ if $\lvert h(\mathbf{x}_i) - y_i \rvert < \delta$, ◦ otherwise $\delta(\lvert h(\mathbf{x}_i) - y_i \rvert - \frac{\delta}{2})$	◦ Also known as Smooth Absolute Loss ◦ ADVANTAGE: "Best of Both Worlds" of Squared and Absolute Loss ◦ Once-differentiable ◦ Takes on behavior of Squared-Loss when loss is small, and Absolute Loss when loss is large.
Log-Cosh Loss $log(cosh(h(\mathbf{x}_i) - y_i))$, $cosh(x) = \frac{e^x + e^{-x}}{2}$	ADVANTAGE: Similar to Huber Loss, but twice differentiable everywhere

Loss Functions With Regression, i.e. $y \in \mathbb{R}$

Slides from Dan Roth



Slides from Dan Roth

That was error functions for the data term. What about the model term?

$$\min_w \frac{1}{2} w^T w + C \sum_{i=1}^{l} max(0, 1 - y_i w^T x_i)$$

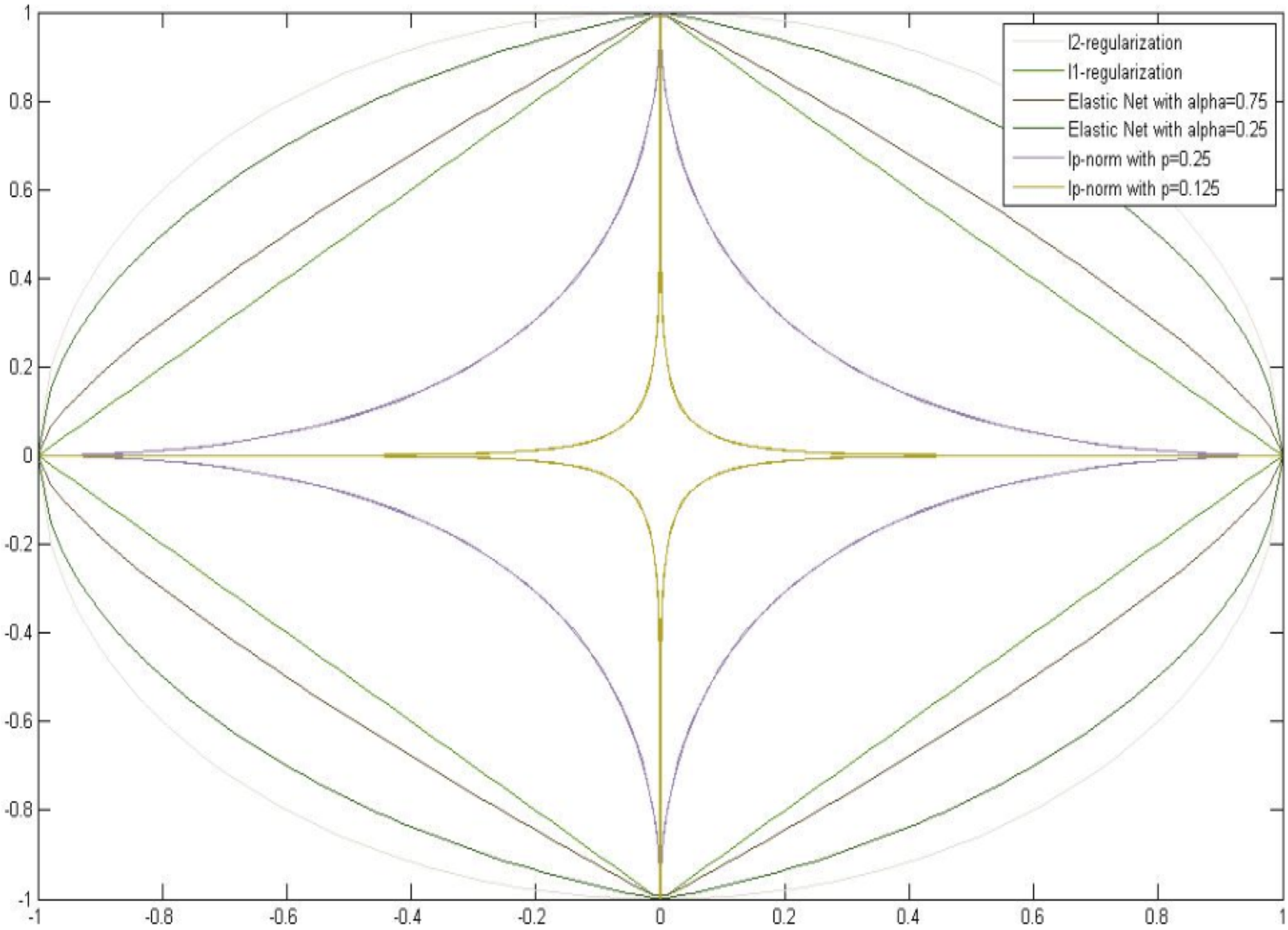
When we look at regularizers it helps to change the formulation of the optimization problem to obtain a better geometric intuition:

$$\min_{\mathbf{w},b} \sum_{i=1}^{n} \ell(h_{\mathbf{w}}(\mathbf{x}), y_i) + \lambda r(\mathbf{w}) \Leftrightarrow \min_{\mathbf{w},b} \sum_{i=1}^{n} \ell(h_{\mathbf{w}}(\mathbf{x}), y_i) \text{ subject to: } r(\mathbf{w}) \leq B$$

Regularizer $r(\mathbf{w})$	**Properties**
l_2-Regularization $r(\mathbf{w}) = \mathbf{w}^\top \mathbf{w} = \|\mathbf{w}\|_2^2$	◦ ADVANTAGE: Strictly Convex ◦ ADVANTAGE: Differentiable ◦ DISADVANTAGE: Uses weights on all features, i.e. relies on all features to some degree (ideally we would like to avoid this) - these are known as Dense Solutions.
l_1-Regularization $r(\mathbf{w}) = \|\mathbf{w}\|_1$	◦ Convex (but not strictly) ◦ DISADVANTAGE: Not differentiable at 0 (the point which minimization is intended to bring us to ◦ Effect: Sparse (i.e. not Dense) Solutions

Elastic Net $\alpha\|\mathbf{w}\|_1 + (1-\alpha)\|\mathbf{w}\|_2^2$ $\alpha \in [0, 1)$	◦ ADVANTAGE: Strictly convex (i.e. unique solution) ◦ DISADVANTAGE: Non-differentiable
l_p-Norm $\|\mathbf{w}\|_p = (\sum_{i=1}^{d} v_i^p)^{1/p}$	◦ (often $0 < p \leq 1$) ◦ DISADVANTAGE: Non-convex ◦ ADVANTAGE: Very sparse solutions ◦ Initialization dependent ◦ DISADVANTAGE: Not differentiable

Slides from Dan Roth



Slides from Dan Roth

L-inf norm?

Loss and Regularizer	Comments
Ordinary Least Squares $\min_{\mathbf{w}} \frac{1}{n} \sum_{i=1}^{n} (\mathbf{w}^\top x_i - y_i)^2$	◦ Squared Loss ◦ No Regularization ◦ Closed form solution: ◦ $\mathbf{w} = (\mathbf{X}\mathbf{X}^\top)^{-1}\mathbf{X}\mathbf{y}^\top$ ◦ $\mathbf{X} = [\mathbf{x}_1, \ldots, \mathbf{x}_n]$ ◦ $\mathbf{y} = [y_1, \ldots, y_n]$
Ridge Regression $\min_{\mathbf{w}} \frac{1}{n} \sum_{i=1}^{n} (\mathbf{w}^\top x_i - y_i)^2 + \lambda \|w\|_2^2$	◦ Squared Loss ◦ l_2-Regularization ◦ $\mathbf{w} = (\mathbf{X}\mathbf{X}^\top + \lambda \mathbb{I})^{-1}\mathbf{X}\mathbf{y}^\top$
Lasso $\min_{\mathbf{w}} \frac{1}{n} \sum_{i=1}^{n} (\mathbf{w}^\top \mathbf{x}_i - y_i)^2 + \lambda \|\mathbf{w}\|_1$	◦ + sparsity inducing (good for feature selection) ◦ + Convex ◦ - Not strictly convex (no unique solution) ◦ - Not differentiable (at 0) ◦ Solve with (sub)-gradient descent or SVEN