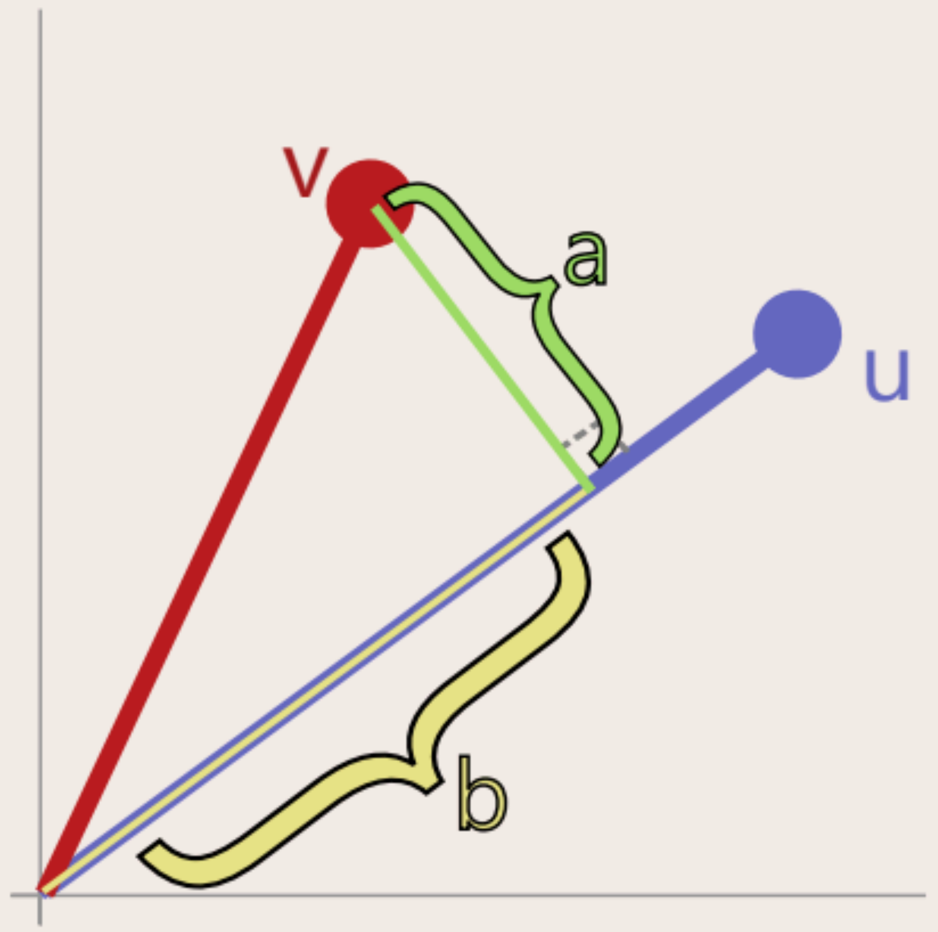
Machine Learning Recap
(linear classification - cont')

Wei Xu

(many slides from Greg Durrett)

Dot Product (math review)

MATH REVIEW | DOT PRODUCTS



Given two vectors $\boldsymbol{u}$ and $\boldsymbol{v}$ their dot product $\boldsymbol{u} \cdot \boldsymbol{v}$ is $\sum_d u_d v_d$. The dot product grows large and positive when $\boldsymbol{u}$ and $\boldsymbol{v}$ point in same direction, grows large and negative when $\boldsymbol{u}$ and $\boldsymbol{v}$ point in opposite directions, and is zero when their are perpendicular. A useful geometric interpretation of dot products is **projection**. Suppose $||\boldsymbol{u}|| = 1$, so that $\boldsymbol{u}$ is a **unit vector**. We can think of any other vector $\boldsymbol{v}$ as consisting of two components: (a) a component in the direction of $\boldsymbol{u}$ and (b) a component that's perpendicular to $\boldsymbol{u}$. This is depicted geometrically to the right: Here, $\boldsymbol{u} = \langle 0.8, 0.6 \rangle$ and $\boldsymbol{v} = \langle 0.37, 0.73 \rangle$. We can think of $\boldsymbol{v}$ as the sum of two vectors, $\boldsymbol{a}$ and $\boldsymbol{b}$, where $\boldsymbol{a}$ is parallel to $\boldsymbol{u}$ and $\boldsymbol{b}$ is perpendicular. The length of $\boldsymbol{b}$ is exactly $\boldsymbol{u} \cdot \boldsymbol{v} = 0.734$, which is why you can think of dot products as projections: the dot product between $\boldsymbol{u}$ and $\boldsymbol{v}$ is the "projection of $\boldsymbol{v}$ onto $\boldsymbol{u}$."

Credit: Hal Daumé III

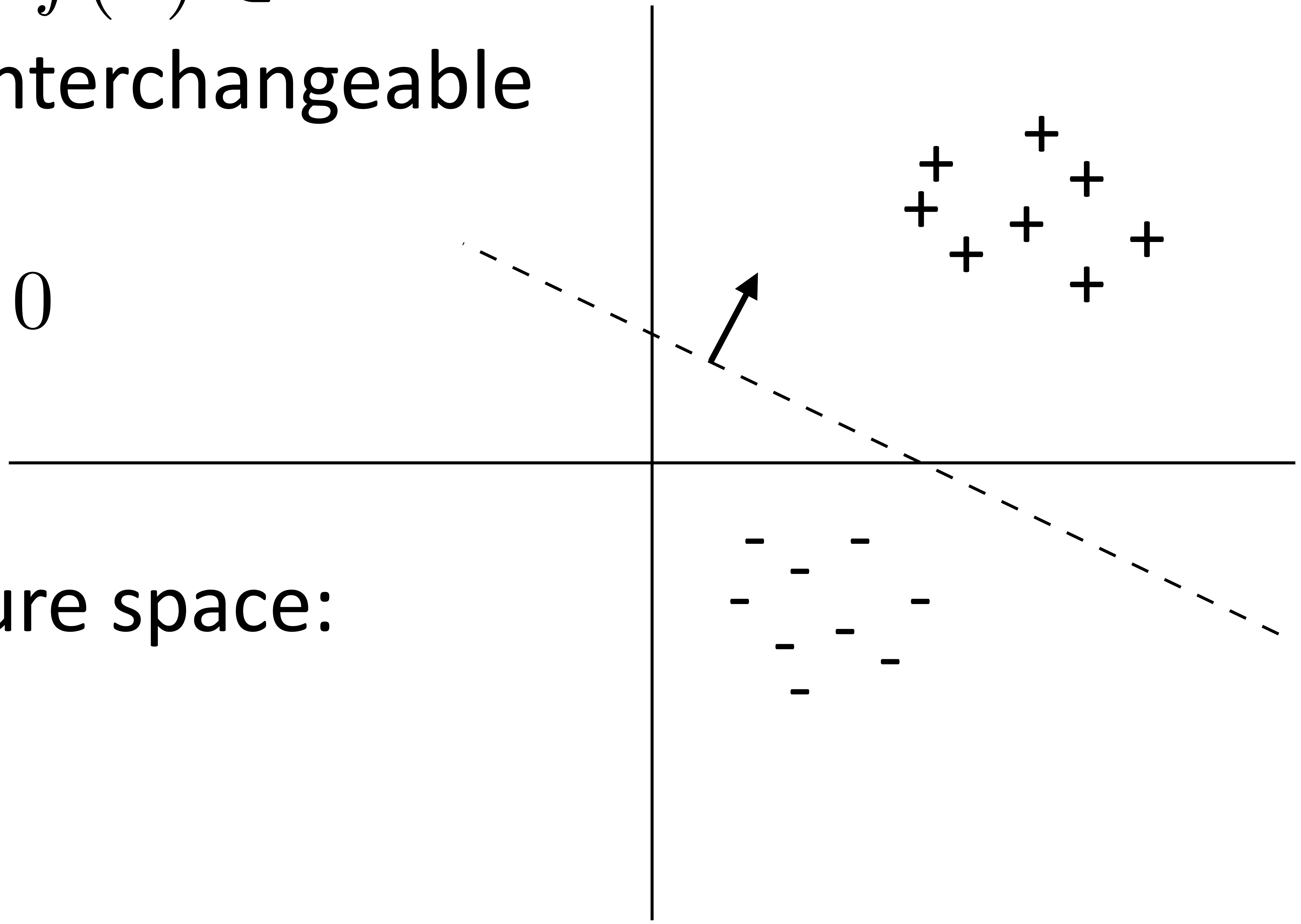
Administrivia

- Programming project 0 is released, due on Jan 17 (Friday).
- Problem Set 1 will be released soon.
- TA Office hours have been announced on Piazza.

Classification

- Datapoint x with label $y \in \{0, 1\}$
- Embed datapoint in a feature space $f(x) \in \mathbb{R}^n$
  but in this lecture $f(x)$ and x are interchangeable
- Linear decision rule: $w^\top f(x) + b > 0$

  $w^\top f(x) > 0$



- Can delete bias if we augment feature space:

$f(x)$ = [0.5, 1.6, 0.3]

↓

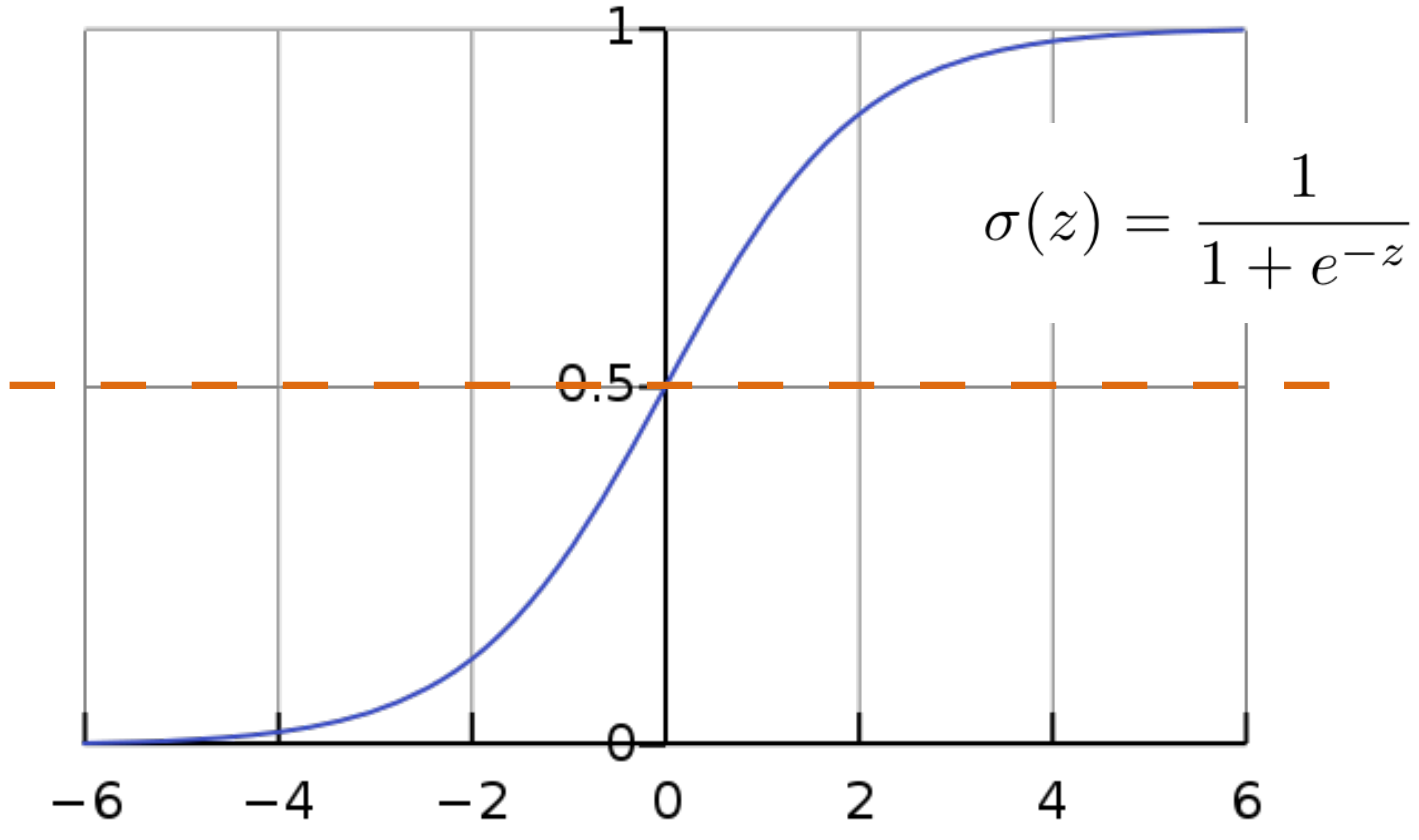
[0.5, 1.6, 0.3, **1**]

Logistic Regression

Logistic Regression

$$P(y=+|x) = \text{logistic}(w^\top x)$$

$$P(y=+|x) = \frac{\exp(\sum_{i=1}^n w_i x_i)}{1+\exp(\sum_{i=1}^n w_i x_i)}$$



- Decision rule: $P(y=+|x) \geq 0.5 \Leftrightarrow w^\top x \geq 0$
- To learn weights: maximize discriminative log likelihood of data P(y|x)

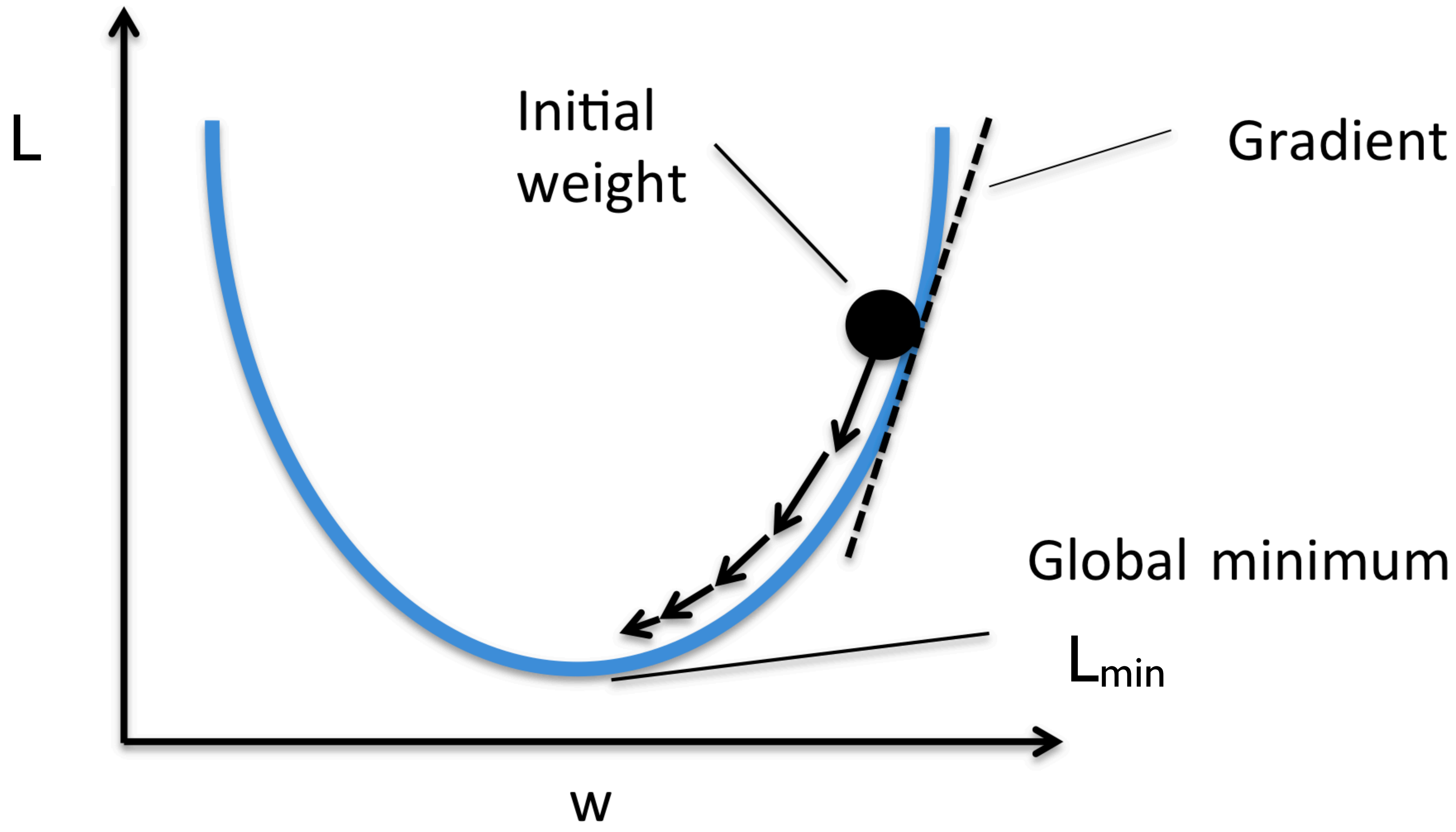
$$\mathcal{L}(x_j, y_j=+) = \log P(y_j=+|x_j)$$

$$= \sum_{i=1}^n w_i x_{ji} - \log\left(1+\exp\left(\sum_{i=1}^n w_i x_{ji}\right)\right)$$

sum over features

Gradient Decent

- Gradient decent (or ascent) is an iterative optimization algorithm for finding the minimum (or maximum) of a function.



Repeat until convergence {

$$w := w - \alpha \frac{\partial \mathcal{L}(w)}{\partial w}$$

}

learning rate (step size)

Logistic Regression

chain rule:

$$\frac{\partial f}{\partial x} = \frac{\partial f}{\partial g}\frac{\partial g}{\partial x} = \frac{\partial f(g)}{\partial g}\frac{\partial g(x)}{\partial x}$$

maximize!

$$\mathcal{L}(x_j, y_j = +) = \log P(y_j = +|x_j) = \boxed{\sum_{i=1}^{n} w_i x_{ji}} - \log\left(1 + \exp\left(\sum_{i=1}^{n} w_i x_{ji}\right)\right)$$

$$\frac{\partial \mathcal{L}(x_j, y_j)}{\partial w_i} = \boxed{x_{ji}} - \frac{\partial}{\partial w_i}\log\left(\boxed{1 + \exp\left(\sum_{i=1}^{n} w_i x_{ji}\right)}\right)$$

$$= x_{ji} - \frac{1}{\boxed{1 + \exp\left(\sum_{i=1}^{n} w_i x_{ji}\right)}}\frac{\partial}{\partial w_i}\left(\boxed{1 + \exp\left(\sum_{i=1}^{n} w_i x_{ji}\right)}\right)$$

deriv. of log

$$\frac{\partial \log \mathrm{x}}{\partial \mathrm{x}} = \frac{1}{\mathrm{x}}$$

$$= x_{ji} - \frac{1}{1 + \exp\left(\sum_{i=1}^{n} w_i x_{ji}\right)} x_{ji} \exp\left(\sum_{i=1}^{n} w_i x_{ji}\right)$$

deriv. of exp

$$\frac{\partial e^{\mathrm{x}}}{\partial \mathrm{x}} = e^{\mathrm{x}}$$

$$= x_{ji} - x_{ji}\frac{\exp\left(\sum_{i=1}^{n} w_i x_{ji}\right)}{1 + \exp\left(\sum_{i=1}^{n} w_i x_{ji}\right)} = x_{ji}(1 - P(y_j = +|x_j))$$

Logistic Regression

- Recall that $y_j = 1$ for positive instances, $y_j = 0$ for negative instances.
- Gradient of w_i on positive example $= x_{ji}(1 - P(y_j = +|x_j))$

  If P(+) is close to 1, make very little update

  Otherwise make w_i look more like x_{ji}, which will increase P(+)

- Gradient of w_i on negative example $= x_{ji}(-P(y_j = +|x_j))$

  If P(+) is close to 0, make very little update

  Otherwise make w_i look less like x_{ji}, which will decrease P(+)

- Can combine these gradients as $\frac{\partial \mathcal{L}(x_j, y_j)}{\partial w} = x_j(y_j - P(y_j = 1|x_j))$

Gradient Decent

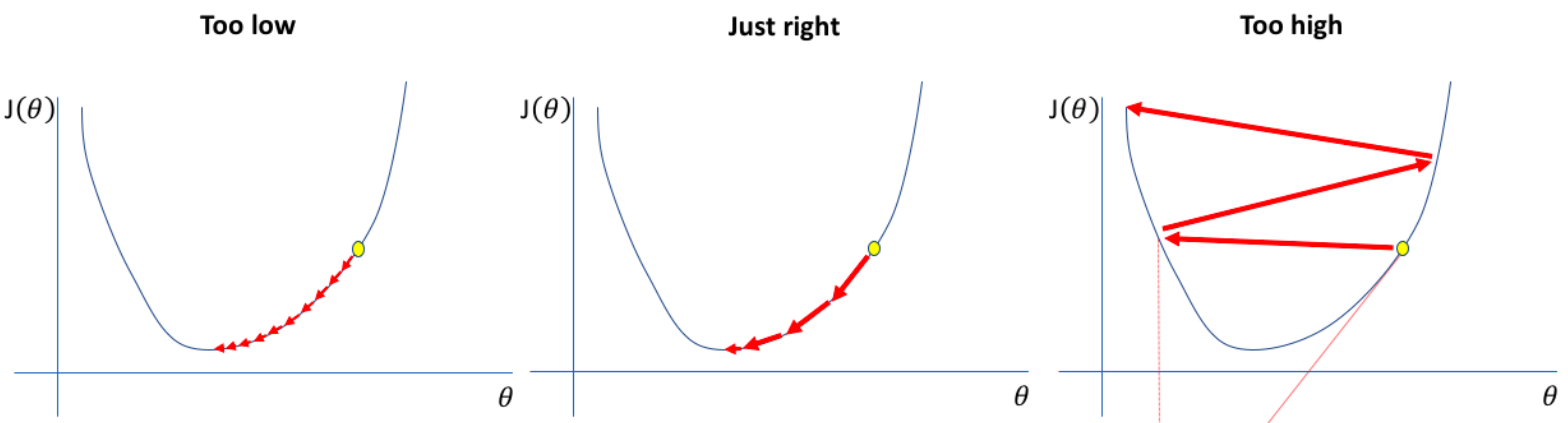
log likelihood of data P(y|x)   data points (*j*)

- Can combine these gradients as $\frac{\partial \mathcal{L}(x_j, y_j)}{\partial w} = x_j(y_j - P(y_j = 1|x_j))$

- Training set log-likelihood: $\mathcal{L}(w) = \frac{1}{m}\sum_{j=1}^{m} \mathcal{L}(x_j, y_j)$

- Gradient vector: $\frac{\partial \mathcal{L}(w)}{\partial w} = \left(\frac{\partial \mathcal{L}}{\partial w_1}, \frac{\partial \mathcal{L}}{\partial w_2}, \ldots, \frac{\partial \mathcal{L}}{\partial w_n}\right)$

Learning Rate



A small learning rate requires many updates before reaching the minimum point

The optimal learning rate swiftly reaches the minimum point

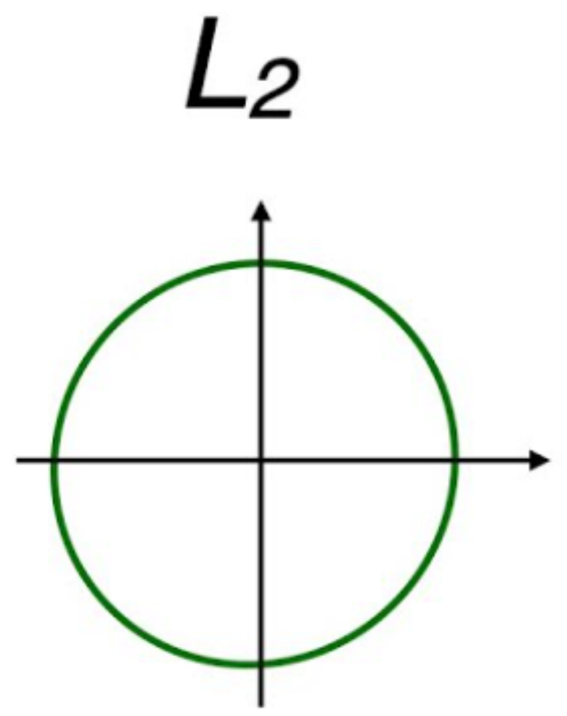
Too large of a learning rate causes drastic updates which lead to divergent behaviors

Credit: Jeremy Jordan

Regularization

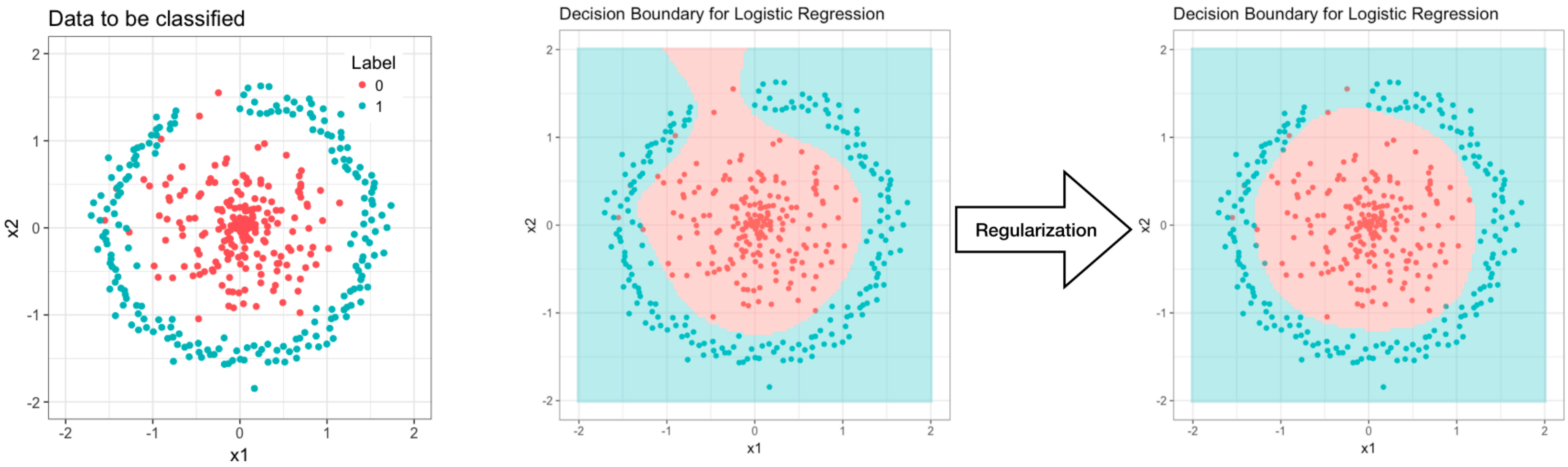
- Regularizing an objective can mean many things, including an L2-norm penalty to the weights:

$$\sum_{j=1}^{m} \mathcal{L}(x_j, y_j) - \lambda \|w\|_2^2$$



- Keeping weights small can prevent overfitting
- For most of the NLP models we build, explicit regularization isn't necessary
  - Early stopping
  - Large numbers of sparse features are hard to overfit in a really bad way
  - For neural networks: dropout and gradient clipping

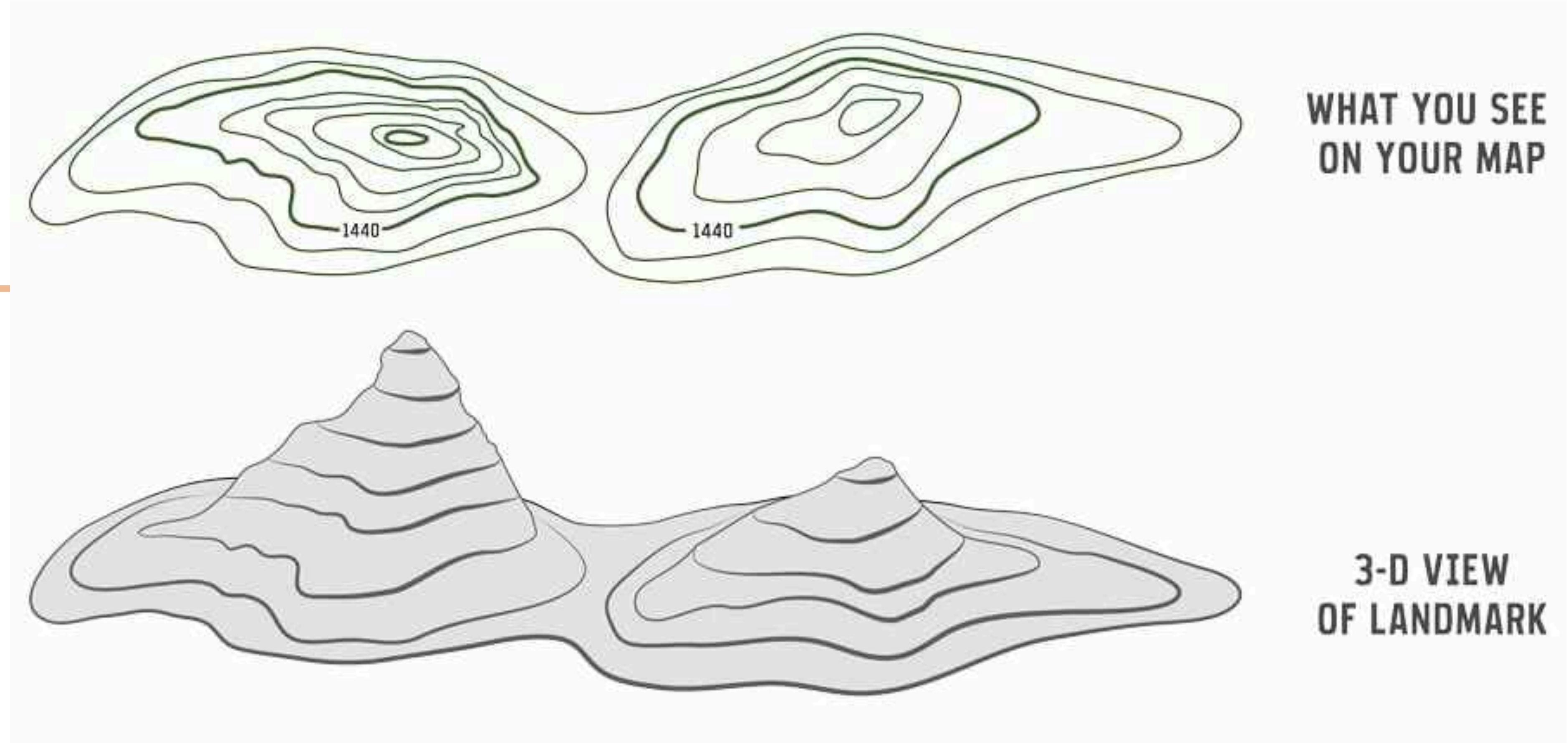
Regularization



$f(x) = [x_1, x_2, x_1^2, x_2^2, x_1x_2, \ldots]$

https://towardsdatascience.com/understanding-regularization-in-machine-learning-5a0369ac73b9

Optimization

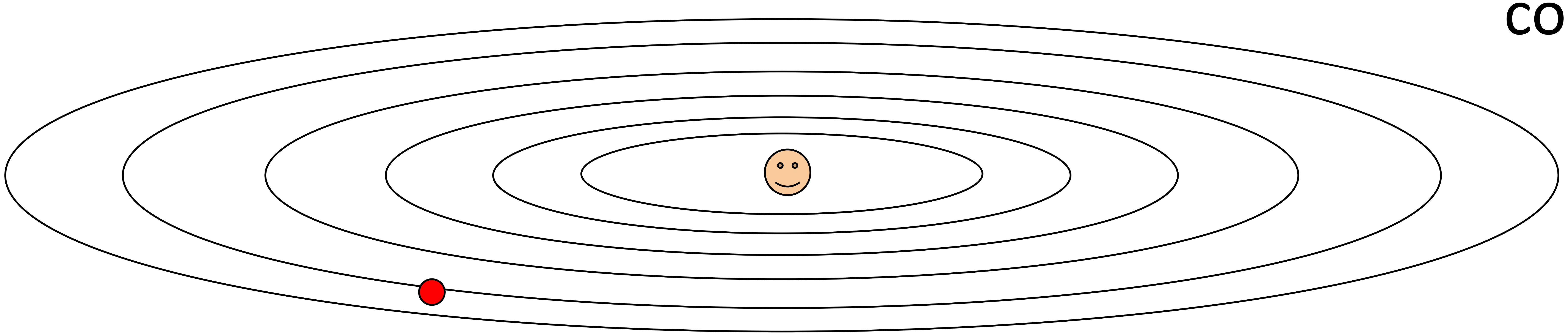


- Gradient descent

$$w := w - \alpha \frac{\partial \mathcal{L}(w)}{\partial w}$$

Q: What if loss changes quickly in one direction and slowly in another direction?

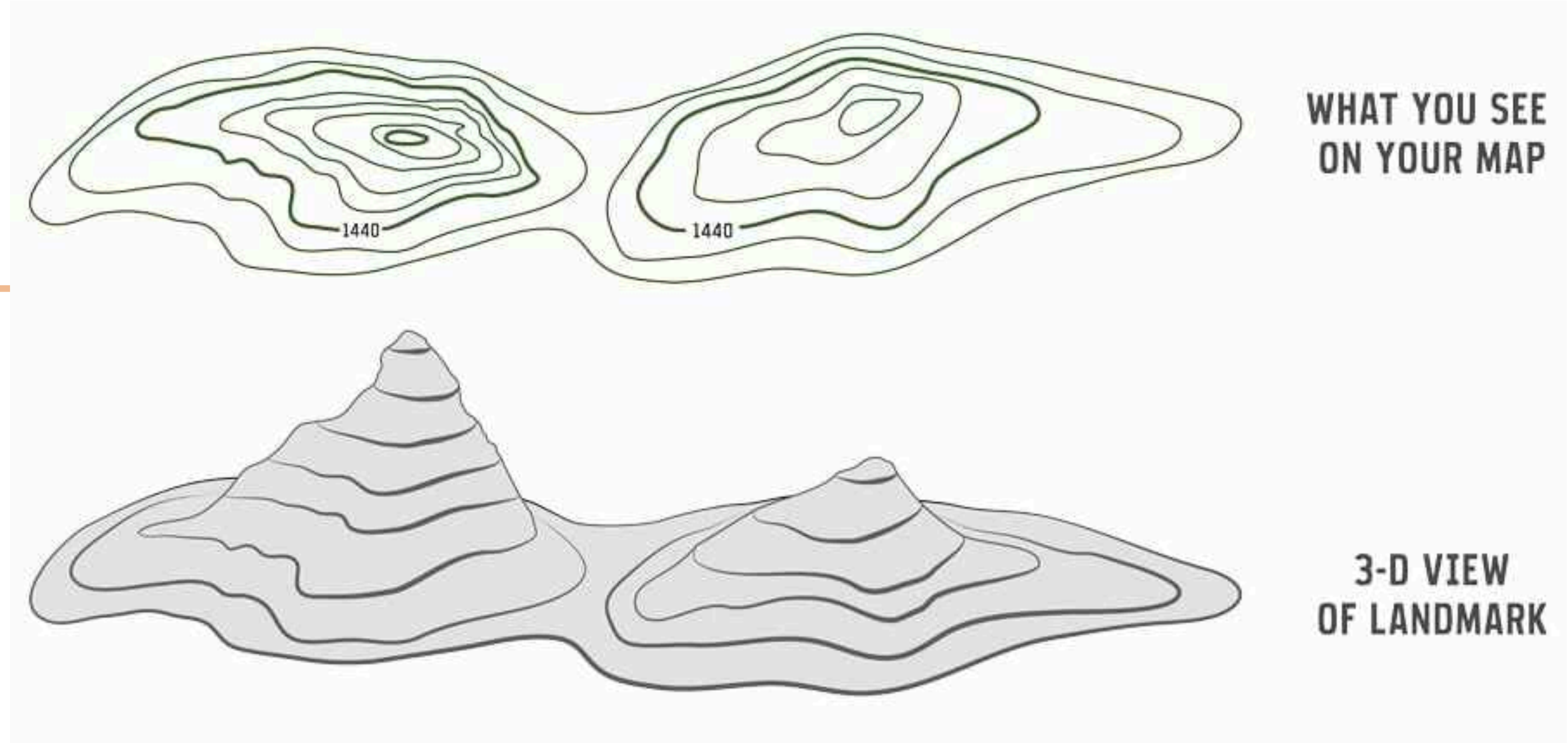
contour plot



Credit: Stanford CS231n

Feature Scaling

Optimization

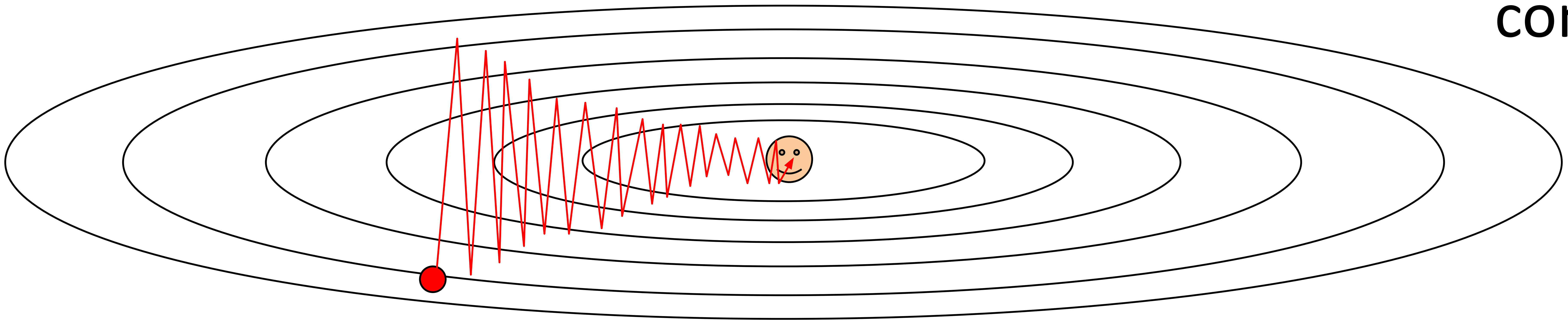


- Gradient descent

$$w := w - \alpha \frac{\partial \mathcal{L}(w)}{\partial w}$$

Q: What if loss changes quickly in one direction and slowly in another direction?

contour plot



Solution: feature scaling!

Credit: Stanford CS231n

Optimization

- Gradient descent
  - Very simple to code up
  - “First-order” technique: only relies on having gradient

$$w \leftarrow w - \alpha g, \quad g = \frac{\partial}{\partial w}\mathcal{L}$$

- Newton’s method
  - Second-order technique
  - Optimizes quadratic instantly

$$w \leftarrow w - \left(\frac{\partial^2}{\partial w^2}\mathcal{L}\right)^{-1} g$$

Inverse Hessian: *n* x *n* mat, expensive!

- Quasi-Newton methods: L-BFGS, etc. approximate inverse Hessian

Logistic Regression: Summary

- Model

$$P(y = +|x) = \frac{\exp(\sum_{i=1}^{n} w_i x_i)}{1 + \exp(\sum_{i=1}^{n} w_i x_i)}$$

- Inference

$\mathrm{argmax}_y P(y|x)$ fundamentally same as Naive Bayes

$$P(y = 1|x) \geq 0.5 \Leftrightarrow w^\top x \geq 0$$

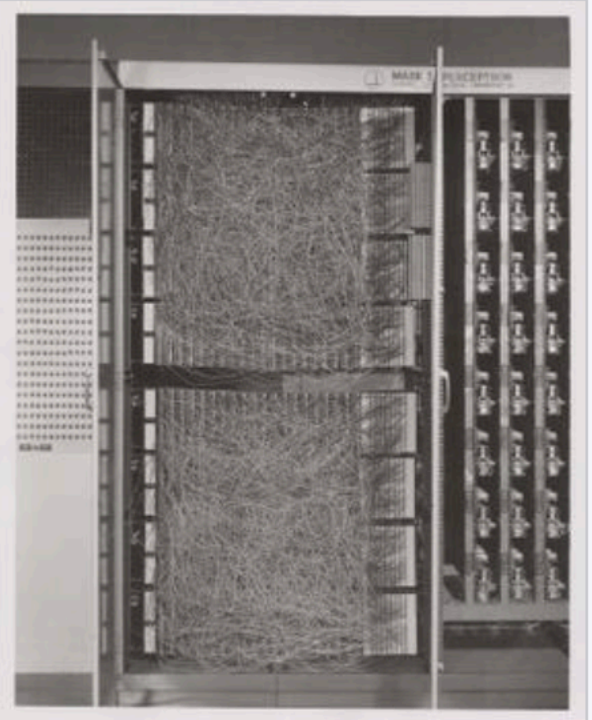
- Learning: gradient ascent on the (regularized) discriminative log-likelihood

Perceptron/SVM

Perceptron

History [edit]

V·T·E



Mark I Perceptron machine, the first implementation of the perceptron algorithm. It was connected to a camera with 20×20 cadmium sulfide photocells to make a 400-pixel image. The main visible feature is a patch panel that set different combinations of input features. To the right, arrays of potentiometers that implemented the adaptive weights.[2]:213

See also: History of artificial intelligence § Perceptrons and the attack on connectionism, and AI winter § The abandonment of connectionism in 1969

The perceptron algorithm was invented in 1958 at the Cornell Aeronautical Laboratory by Frank Rosenblatt,[3] funded by the United States Office of Naval Research.[4]

The perceptron was intended to be a machine, rather than a program, and while its first implementation was in software for the IBM 704, it was subsequently implemented in custom-built hardware as the "Mark 1 perceptron". This machine was designed for image recognition: it had an array of 400 photocells, randomly connected to the "neurons". Weights were encoded in potentiometers, and weight updates during learning were performed by electric motors.[2]:193

In a 1958 press conference organized by the US Navy, Rosenblatt made statements about the perceptron that caused a heated controversy among the fledgling AI community; based on Rosenblatt's statements, *The New York Times* reported the perceptron to be "the embryo of an electronic computer that [the Navy] expects will be able to walk, talk, see, write, reproduce itself and be conscious of its existence."[4]

Although the perceptron initially seemed promising, it was quickly proved that perceptrons could not be trained to recognise many classes of patterns. This caused the field of neural network research to stagnate for many years, before it was recognised that a feedforward neural network with two or more layers (also called a multilayer perceptron) had greater processing power than perceptrons with one layer (also called a single layer perceptron).

Single layer perceptrons are only capable of learning linearly separable patterns. For a classification task with some step activation function a single node will have a single line dividing the data points forming the patterns. More nodes can create more dividing lines, but those lines must somehow be combined to form more complex classifications. A second layer of perceptrons, or even linear nodes, are sufficient to solve a lot of otherwise non-separable problems.

In 1969 a famous book entitled *Perceptrons* by Marvin Minsky and Seymour Papert showed that it was impossible for these classes of network to learn an XOR function. It is often believed (incorrectly) that they also conjectured that a similar result would hold for a multi-layer perceptron network. However, this is not true, as both Minsky and Papert already knew that multi-layer perceptrons were capable of producing an XOR function. (See the page on *Perceptrons (book)* for more information.) Nevertheless, the often-miscited Minsky/Papert text caused a significant decline in interest and funding of neural network research. It took ten more years until neural network research experienced a resurgence in the 1980s. This text was reprinted in 1987 as "Perceptrons - Expanded Edition" where some errors in the original text are shown and corrected.

The kernel perceptron algorithm was already introduced in 1964 by Aizerman et al.[5] Margin bounds guarantees were given for the Perceptron algorithm in the general non-separable case first by Freund and Schapire (1998),[1] and more recently by Mohri and Rostamizadeh (2013) who extend previous results and give new L1 bounds.[6]

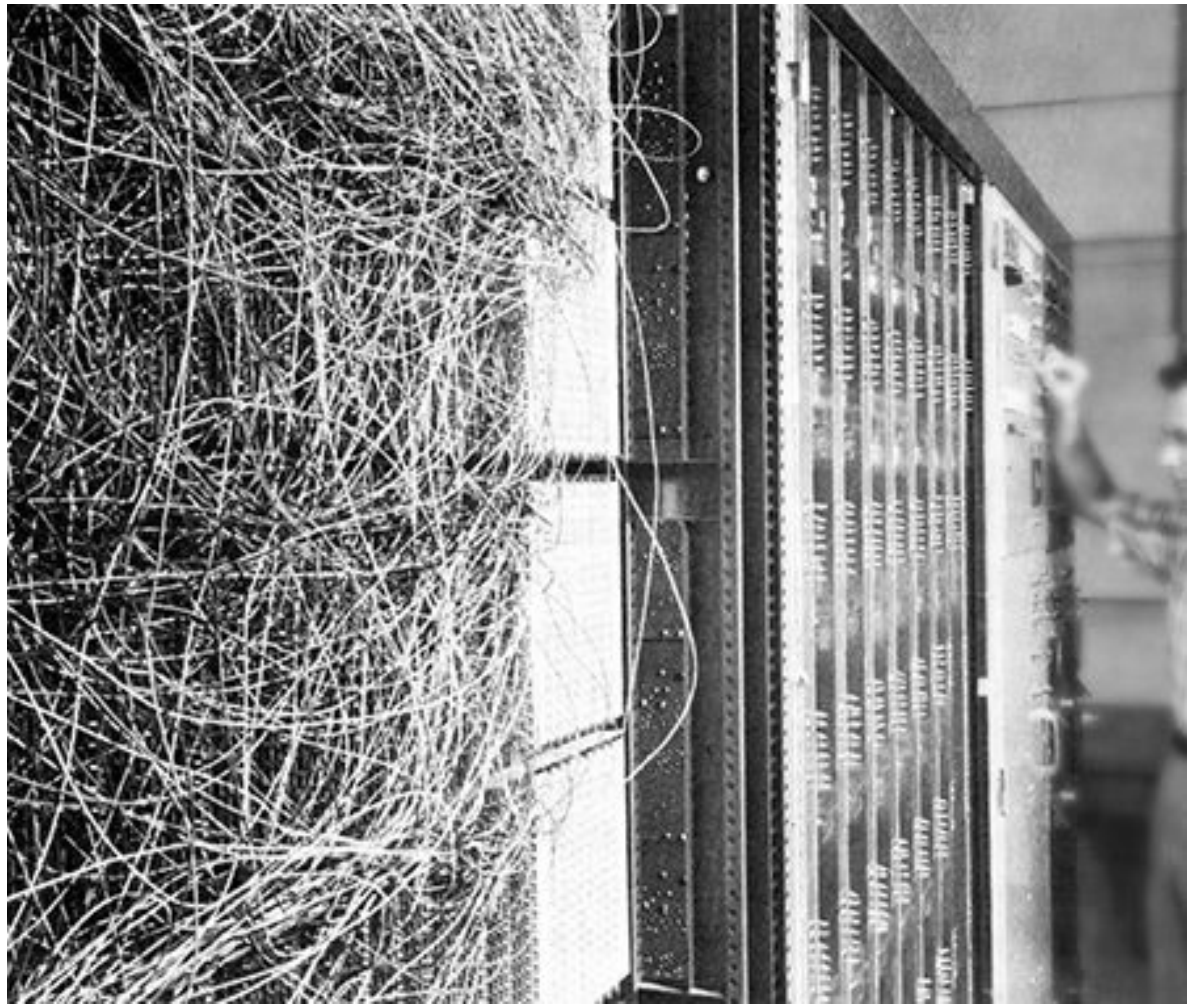
The perceptron is a simplified model of a biological neuron. While the complexity of biological neuron models is often required to fully understand neural behavior, research suggests a perceptron-like linear model can produce some behavior seen in real neurons.[7]

Frank Rosenblatt (1928-1971)

PhD 1956 from Cornell

A Bit of History

- The **Mark I Perceptron** machine was the first implementation of the perceptron algorithm.
- Perceptron (Frank Rosenblatt, 1957)
- Artificial Neuron (McCulloch & Pitts, 1943)



McCulloch Pitts Neuron
(assuming no inhibitory inputs)

$$y = 1 \quad if \sum_{i=0}^{n} x_i \geq 0$$
$$= 0 \quad if \sum_{i=0}^{n} x_i < 0$$

Perceptron

$$y = 1 \quad if \sum_{i=0}^{n} w_i * x_i \geq 0$$
$$= 0 \quad if \sum_{i=0}^{n} w_i * x_i < 0$$

The IBM Automatic Sequence Controlled Calculator, called Mark I by Harvard University's staff. It was designed for image recognition: it had an array of 400 photocells, randomly connected to the "neurons". Weights were encoded in potentiometers, and weight updates during learning were performed by electric motors. The first program was run on Mark I in 1944.

https://www.youtube.com/watch?v=SaFQAoYV1Nw

https://www.youtube.com/watch?time_continue=71&v=cNxadbrN_aI&feature=emb_logo

Perceptron - artificial neuron

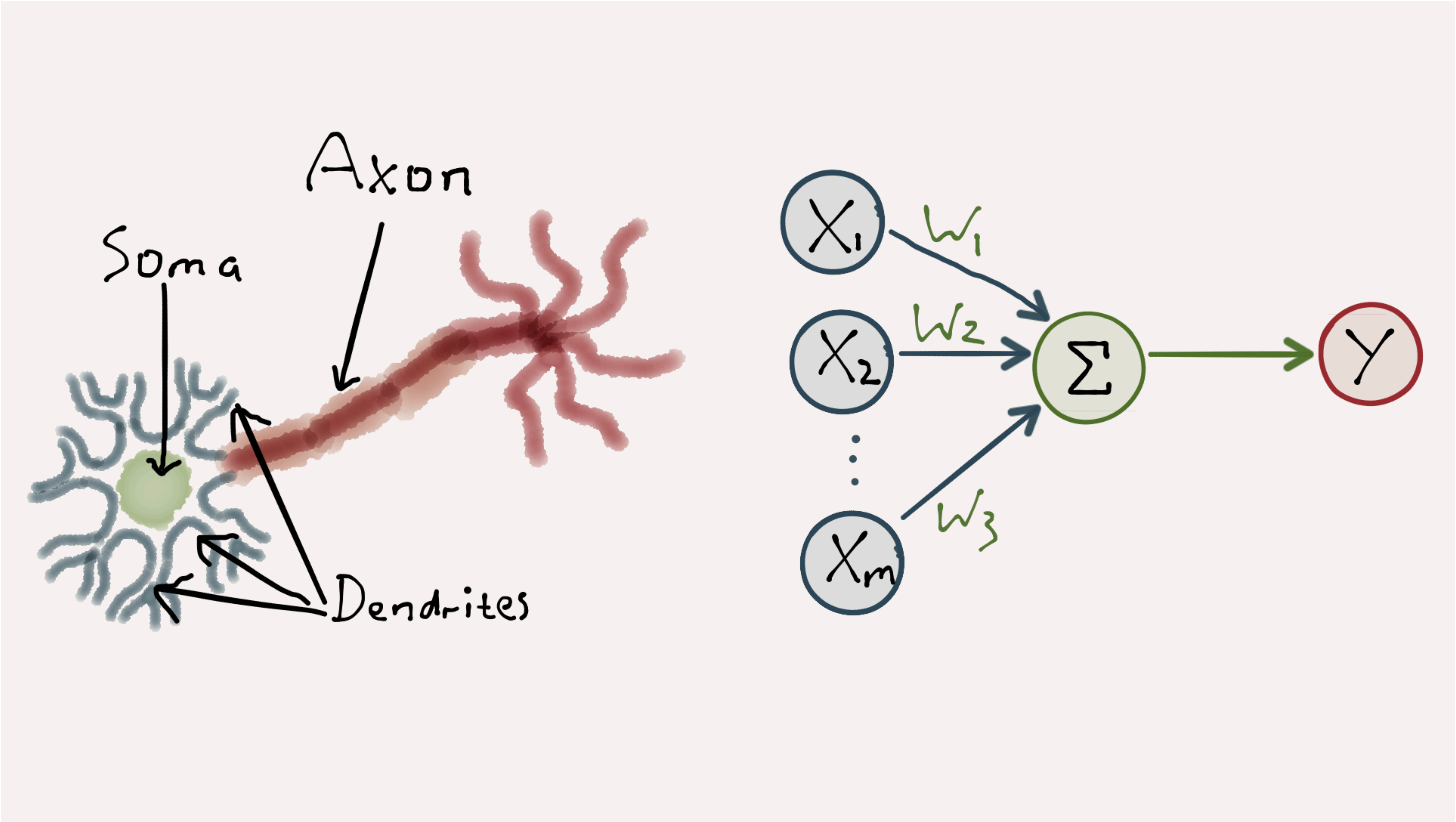


Figure from https://jontysinai.github.io/jekyll/update/2017/11/11/the-perceptron.html

Perceptron

- Simple error-driven learning approach similar to logistic regression

- Decision rule: $w^\top x > 0$
  - If incorrect: if positive, $w \leftarrow w + x$

    if negative, $w \leftarrow w - x$

Logistic Regression

$$w \leftarrow w + x(1 - P(y = 1|x))$$

$$w \leftarrow w - xP(y = 1|x)$$

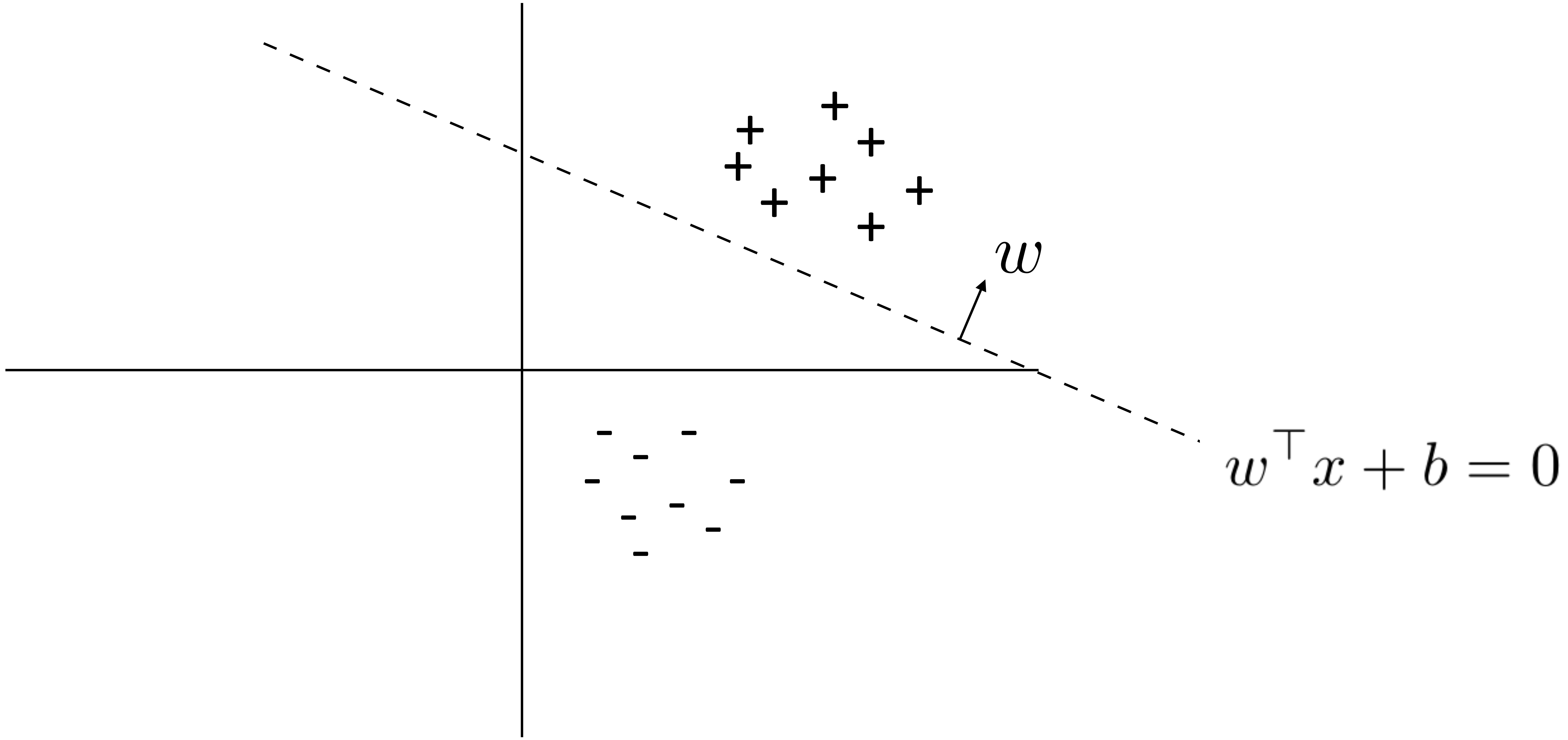
- Algorithm is very similar to logistic regression

- Perceptron guaranteed to eventually separate the data if the data are separable

http://ciml.info/dl/v0_99/ciml-v0_99-ch04.pdf

Perceptron

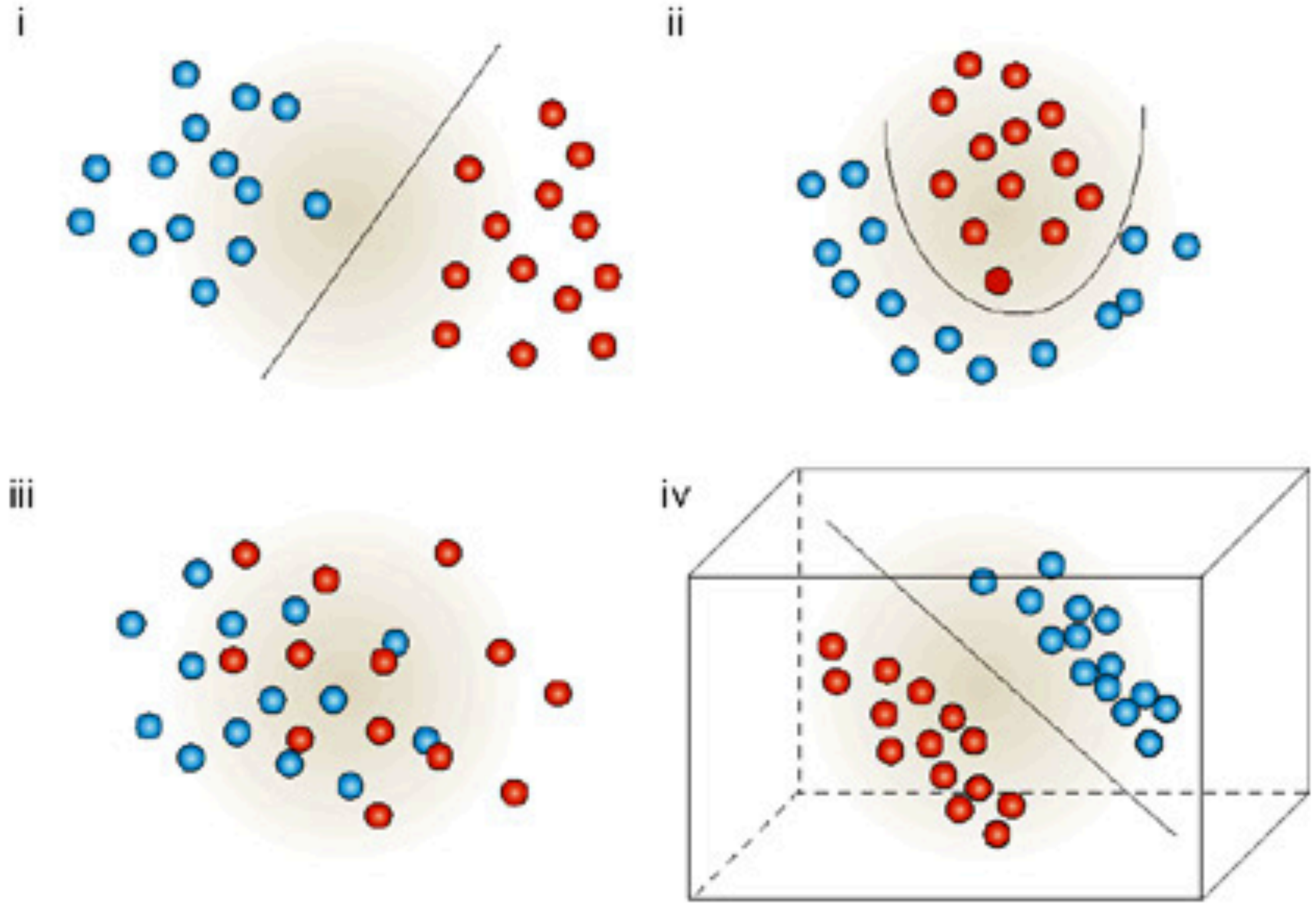
- Separating hyperplane



Two vectors have a zero dot product if and only if they are perpendicular

Linear Separability

- In general, two groups are linearly separable in n-dimensional space, if they can be separated by an (n-1)-dimensional hyperplane.



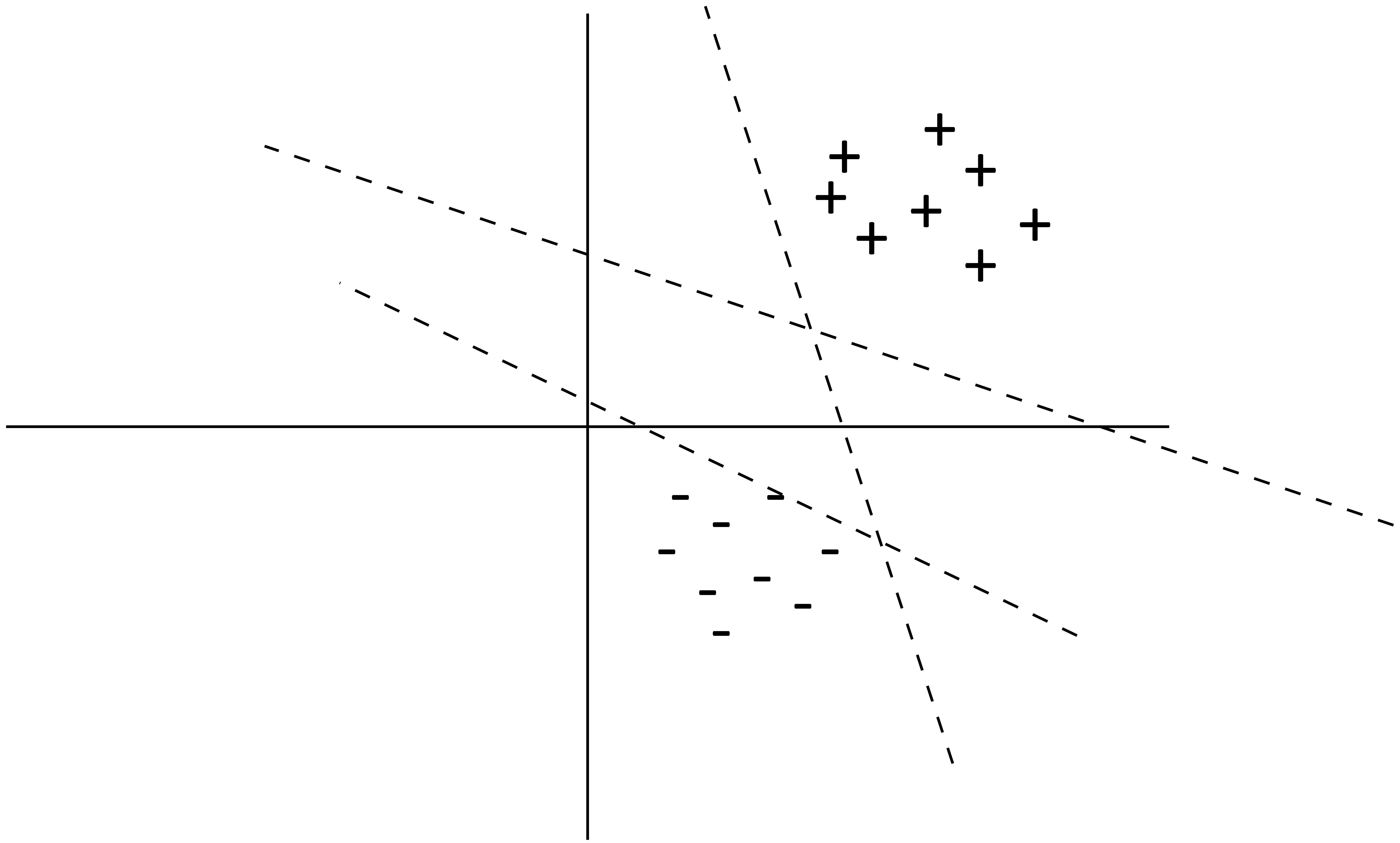
What does “converge” mean?

- It means that it can make an entire pass through the training data without making any more updates.
- In other words, Perceptron has correctly classified every training example.
- Geometrically, this means that it was found some hyperplane that correctly segregates the data into positive and negative examples

http://ciml.info/dl/v0_99/ciml-v0_99-ch04.pdf

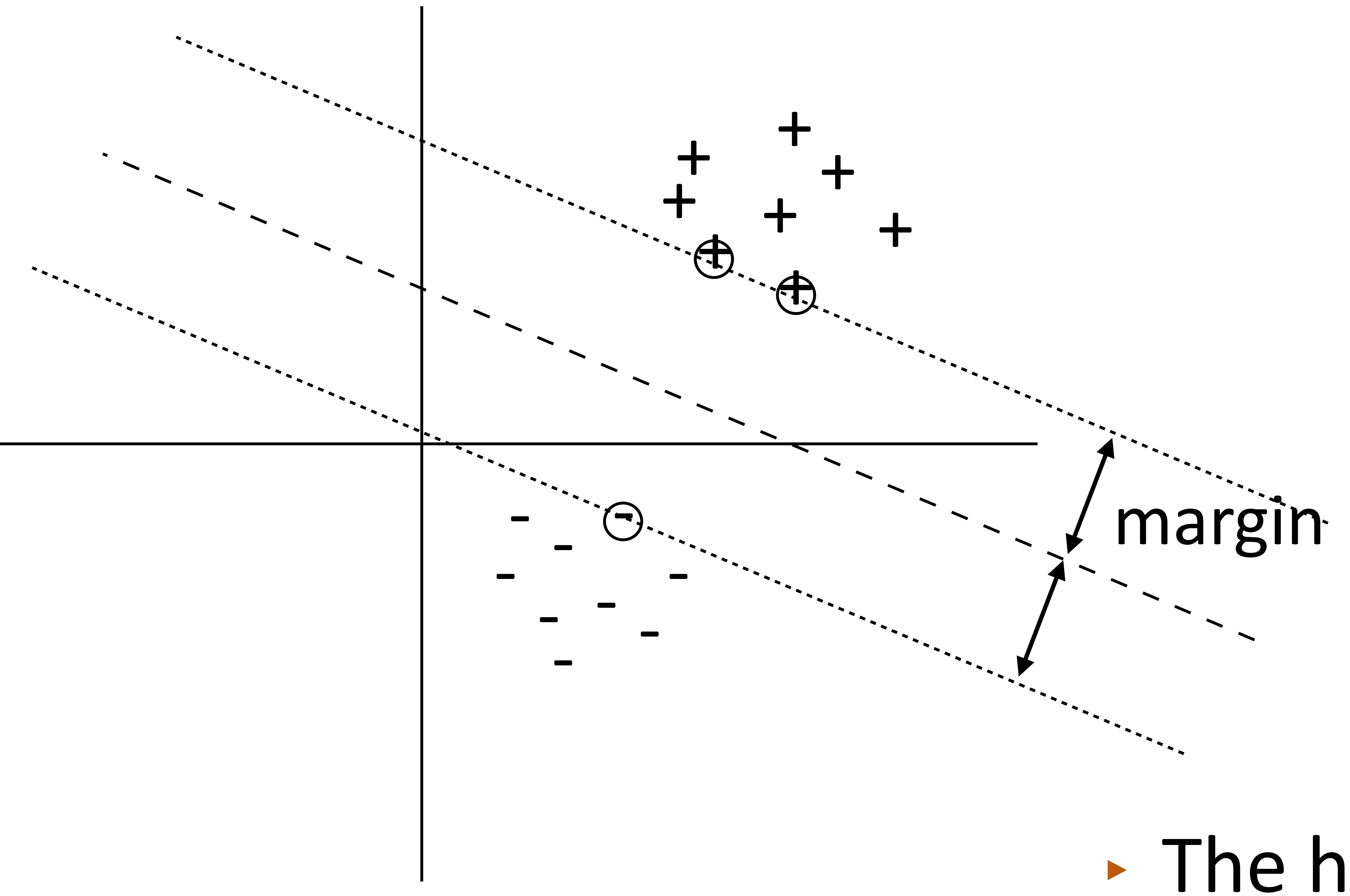
Support Vector Machines

- Many separating hyperplanes — is there a best one?



Support Vector Machines

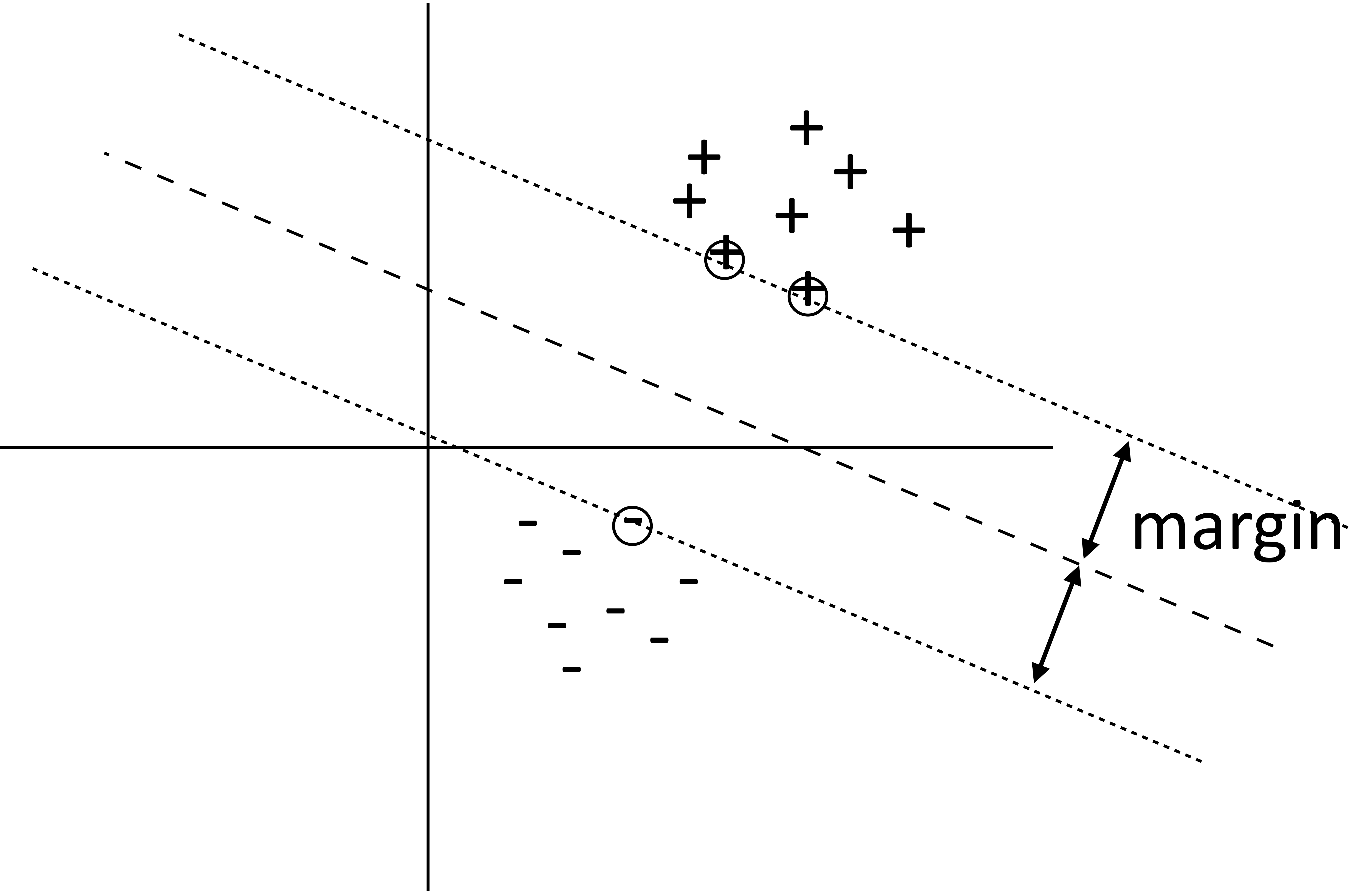
- Many separating hyperplanes — is there a best one?

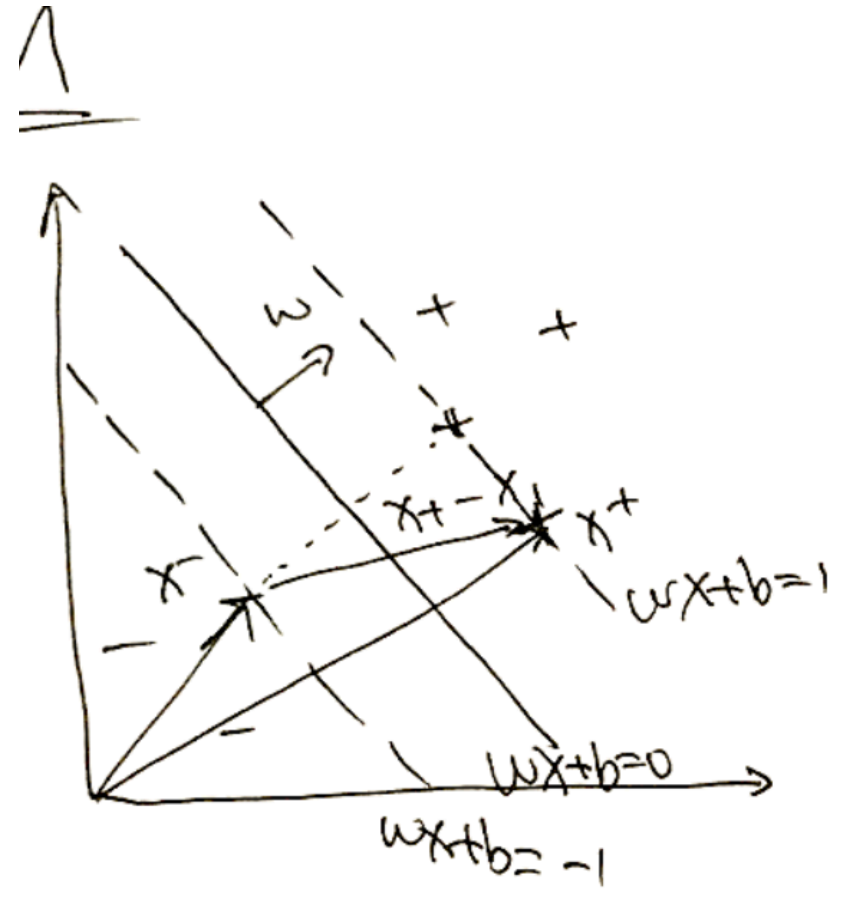


- The hyperplane lies exactly halfway between the nearest positive and negative example.

Support Vector Machines

- Many separating hyperplanes — is there a best one?





hyperplane will lie exactly halfway between the nearest positive point and nearest negative point.

Margin WIDTH $= (x_+ - x_-) \cdot \frac{w}{\|w\|} = \frac{2}{\|w\|}$

$wx_+ + b = 1$
$wx_- + b = -1$

$\text{Max} \frac{2}{\|w\|} \sim \text{Max} \frac{1}{\|w\|} \sim \min \|w\| \sim \min \frac{1}{2}\|w\|$

Support Vector Machines

- Constraint formulation: find *w* via following quadratic program:

Minimize $\|w\|_2^2$

s.t. $\forall j \quad w^\top x_j \geq 1 \text{ if } y_j = 1$

$$w^\top x_j \leq -1 \text{ if } y_j = 0$$

minimizing norm with fixed margin <=> maximizing margin

As a single constraint:

$$\forall j \quad (2y_j - 1)(w^\top x_j) \geq 1$$

- Generally no solution (data is generally non-separable) — need slack!

http://www.cs.toronto.edu/~mbrubake/teaching/C11/Handouts/SupportVectorMachines.pdf

N-Slack SVMs

Minimize $$\lambda \|w\|_2^2 + \sum_{j=1}^{m} \xi_j$$

s.t. $\forall j \quad (2y_j - 1)(w^\top x_j) \geq 1 - \xi_j \qquad \forall j \quad \xi_j \geq 0$

- The ξ_j are a "fudge factor" to make all constraints satisfied

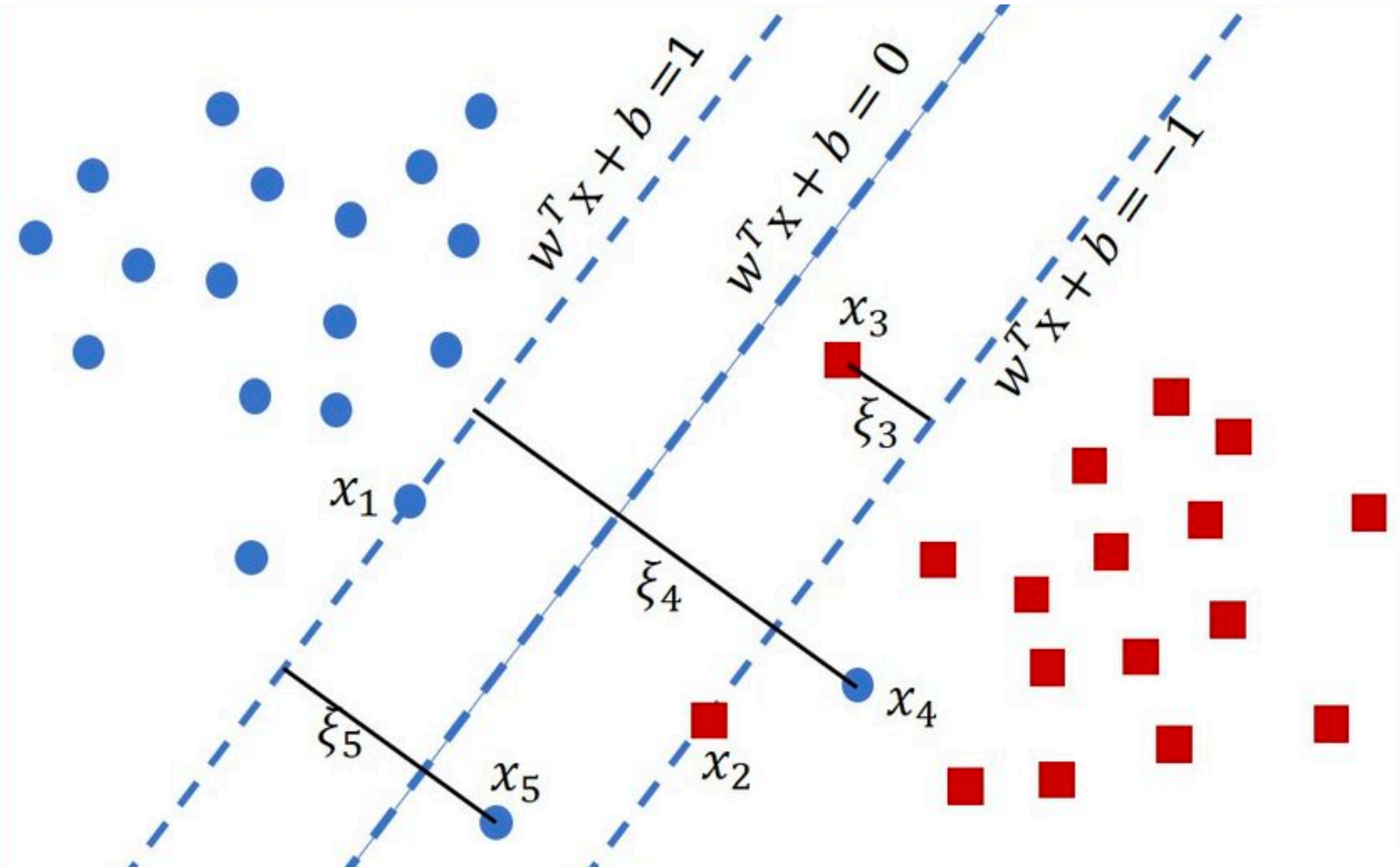


Image credit: Lang Van Tran

http://www.cs.toronto.edu/~mbrubake/teaching/C11/Handouts/SupportVectorMachines.pdf

N-Slack SVMs

Minimize $\lambda\|w\|_2^2 + \sum_{j=1}^{m} \xi_j$

s.t. $\forall j \quad (2y_j - 1)(w^\top x_j) \geq 1 - \xi_j \qquad \forall j \quad \xi_j \geq 0$

- The ξ_j are a "fudge factor" to make all constraints satisfied
- Take the gradient of the objective (flip for maximizing):

$$\frac{\partial}{\partial w_i}\xi_j = 0 \text{ if } \xi_j = 0 \qquad \frac{\partial}{\partial w_i}\xi_j = (2y_j - 1)x_{ji} \text{ if } \xi_j > 0$$

$$= x_{ji} \text{ if } y_j = 1, \; -x_{ji} \text{ if } y_j = 0$$

- Looks like the perceptron! But updates more frequently

http://www.cs.toronto.edu/~mbrubake/teaching/C11/Handouts/SupportVectorMachines.pdf

LR, Perceptron, SVM

- Logistic regression: $P(y=1|x) = \frac{\exp\left(\sum_{i=1}^{n} w_i x_i\right)}{\left(1+\exp\left(\sum_{i=1}^{n} w_i x_i\right)\right)}$

  Decision rule: $P(y=1|x) \geq 0.5 \Leftrightarrow w^\top x \geq 0$

  Gradient (unregularized): $x(y - P(y=1|x))$

- Logistic regression, perceptron, and SVM are closely related

- All gradient updates: “make it look more like the right thing and less like the wrong thing”

LR, Perceptron, SVM

- Gradients on Positive Examples

Logistic regression

$$x(1 - \text{logistic}(w^\top x))$$

Perceptron

$$x \text{ if } w^\top x < 0, \;\; \text{else } 0$$

SVM (ignoring regularizer)

$$x \text{ if } w^\top x < 1, \;\; \text{else } 0$$

*these gradients are for maximizing things, which is why they are flipped

http://ciml.info/dl/v0_99/ciml-v0_99-ch07.pdf

LR, Perceptron, SVM

- Loss on Positive Examples

Logistic Loss

$$log(1 + e^{-z})$$

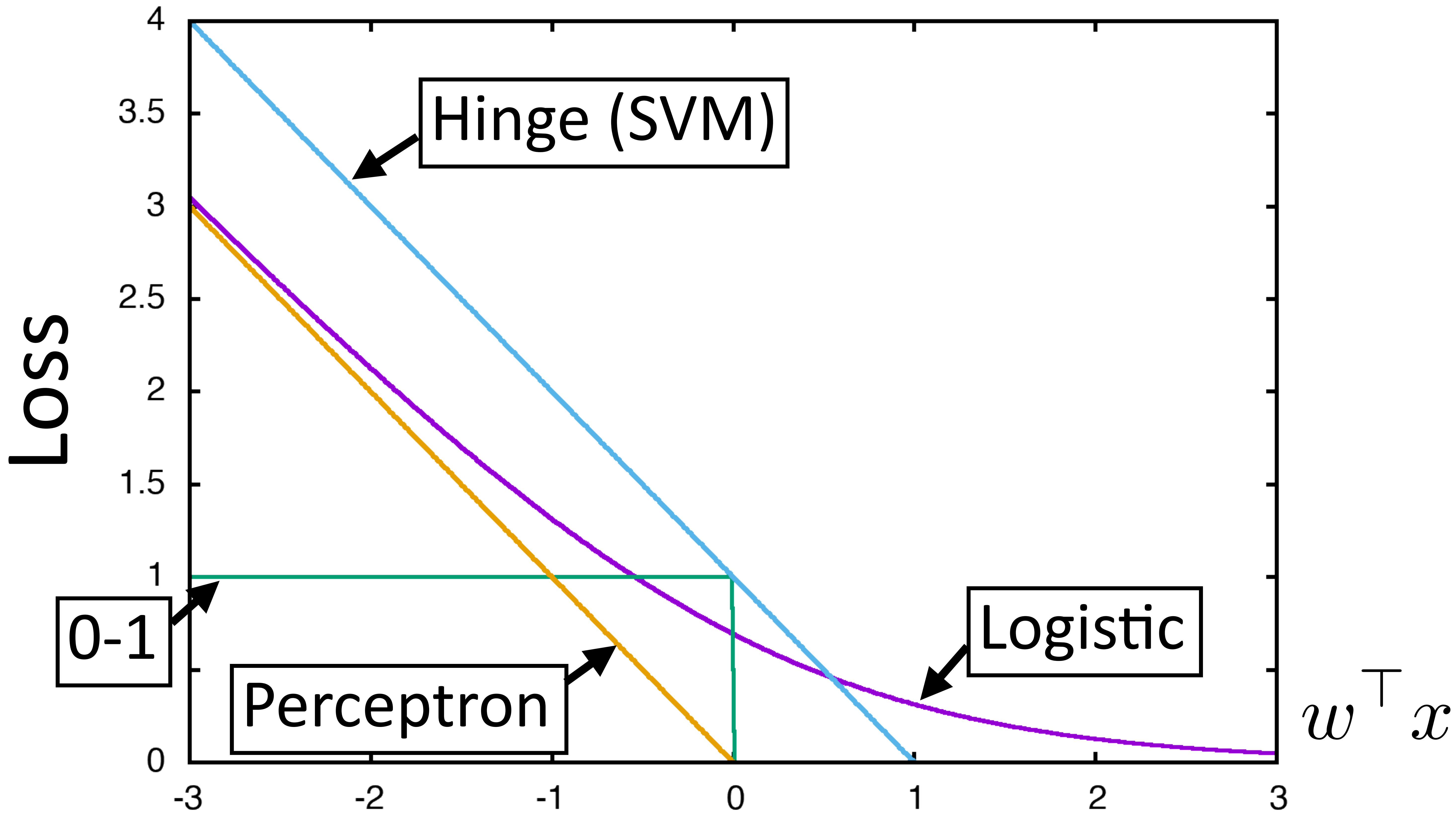
Perceptron Loss

$$\max(0, -z)$$

Hinge Loss

$$\max(0, 1 - z)$$

$$z = w^\top x$$



http://ciml.info/dl/v0_99/ciml-v0_99-ch07.pdf

Optimization — more later …

- Range of techniques from simple gradient descent (works pretty well) to more complex methods (can work better), e.g., Newton's method, Quasi-Newton methods (LBFGS), Adagrad, Adadelta, etc.

- Most methods boil down to: take a gradient and a step size, apply the gradient update times step size, incorporate estimated curvature information to make the update more effective

Sentiment Analysis

this movie was great! would watch again | +

the movie was gross and overwrought, but I liked it | +

this movie was not really very enjoyable | —

- Bag-of-words doesn't seem sufficient (discourse structure, negation)
- There are some ways around this: extract bigram feature for "*not* X" for all X following the *not*

Bo Pang, Lillian Lee, Shivakumar Vaithyanathan (2002)

Sentiment Analysis

	Features	# of features	frequency or presence?	NB	ME	SVM
(1)	unigrams	16165	freq.	**78.7**	N/A	72.8
(2)	unigrams	"	pres.	81.0	80.4	**82.9**
(3)	unigrams+bigrams	32330	pres.	80.6	80.8	**82.7**
(4)	bigrams	16165	pres.	77.3	**77.4**	77.1
(5)	unigrams+POS	16695	pres.	81.5	80.4	**81.9**
(6)	adjectives	2633	pres.	77.0	**77.7**	75.1
(7)	top 2633 unigrams	2633	pres.	80.3	81.0	**81.4**
(8)	unigrams+position	22430	pres.	81.0	80.1	**81.6**

- Simple feature sets can do pretty well!

Bo Pang, Lillian Lee, Shivakumar Vaithyanathan (2002)

Sentiment Analysis

Method	RT-s	MPQA
MNB-uni	77.9	85.3
MNB-bi	**79.0**	**86.3**
SVM-uni	76.2	86.1
SVM-bi	77.7	**86.7**
NBSVM-uni	**78.1**	85.3
NBSVM-bi	**79.4**	**86.3**
RAE	76.8	85.7
RAE-pretrain	77.7	**86.4**
Voting-w/Rev.	63.1	81.7
Rule	62.9	81.8
BoF-noDic.	75.7	81.8
BoF-w/Rev.	76.4	84.1
Tree-CRF	77.3	86.1
BoWSVM	–	–
Kim (2014) CNNs	**81.5**	**89.5**

Naive Bayes is doing well!

Ng and Jordan (2002) — NB can be better for small data

Recursive Auto-encoder. Before neural nets had taken off — results weren't that great

Wang and Manning (2012)

Summary

- Logistic regression, SVM, and perceptron are closely related
- SVM and perceptron inference require taking maxes, logistic regression has a similar update but is "softer" due to its probabilistic nature
- All gradient updates: "make it look more like the right thing and less like the wrong thing"

QA Time