# Classification

Liang Zheng
Australian National University
liang.zheng@anu.edu.au

Reference: Bishop, Christopher M. *Pattern recognition and machine learning*. Springer, 2006.

# Week 11 lecture arrangement

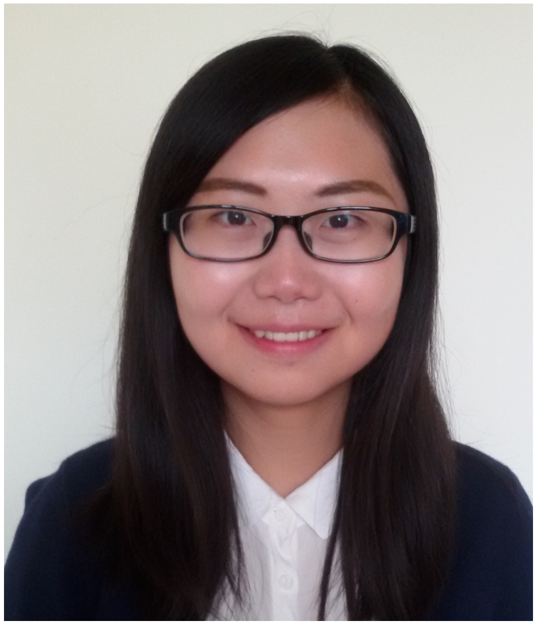
- Tuesday: 4pm-5:30pm, Copland lecture theatre & online (classification 1)
- Wednesday: 1pm-2:30pm, fully online (guest lecture)
- Thursday: 10:30am–12:00pm, fully online (classification 2)
- Friday: 3pm-4:30pm, Kambri cinema & online (review lecture)

**For all these lectures, we use the same Zoom link for the online part:**

https://anu.zoom.us/j/85047332371?pwd=RDNOTnAvaG9VQnBDNzdGcnBlNWpWZz09

Meeting ID: 850 4733 2371
Password: 634639

# Guest speakers

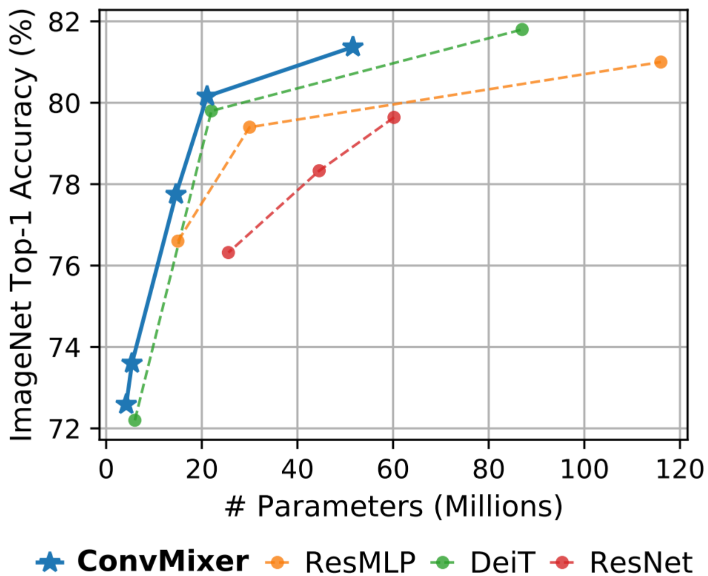


Liyue Shen, Postdoc@Harvard University
Incoming Assistant Professor @University of Michigan

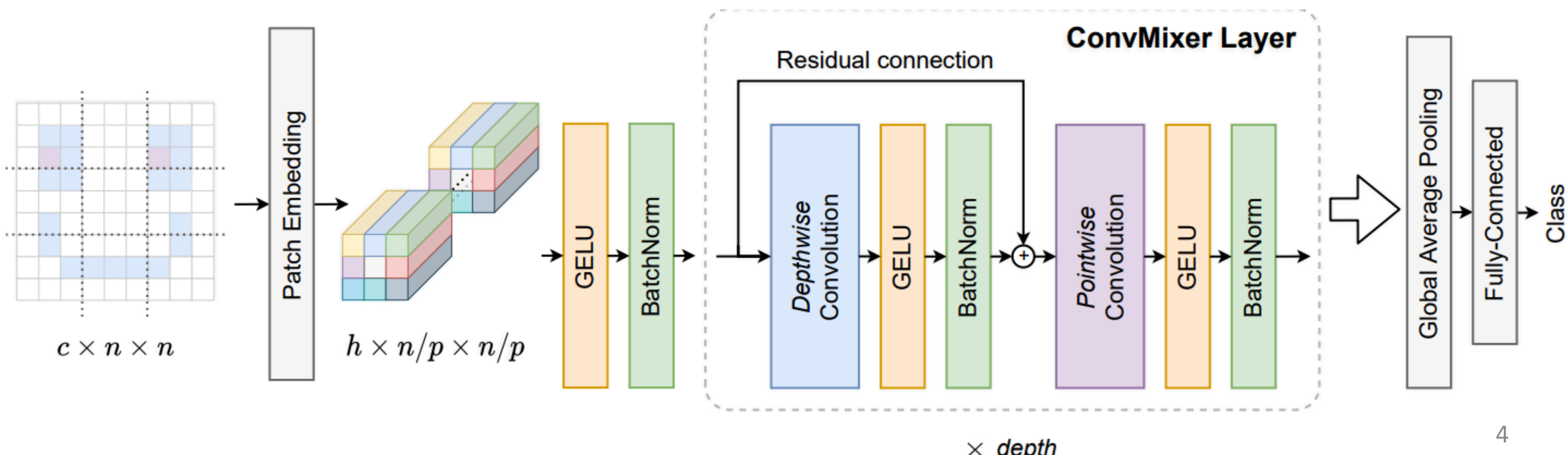
José Lezama, Research Scientist @Google Research

# ~~Convolutions~~ ~~Attention~~ ~~MLPs~~

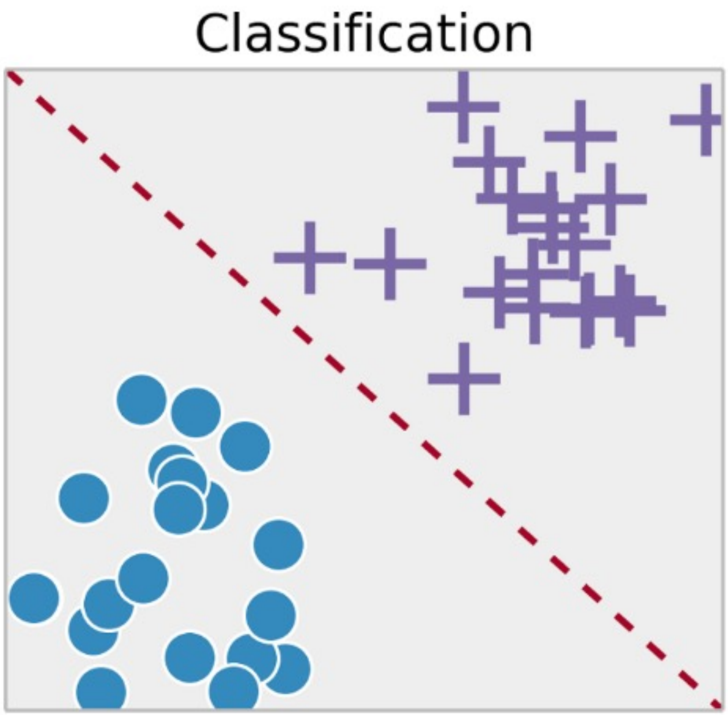
# Patches Are All You Need? 🤷

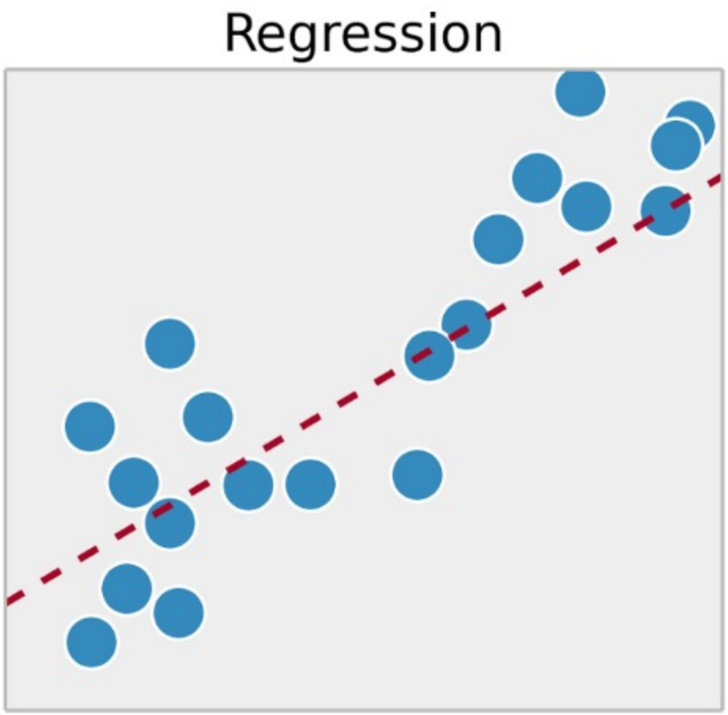



"**A note on paper length**. Expecting more text in this paper? Wondering if it's a workshop paper we hastily submitted to ICLR? No. This paper presents a simple idea, one where we genuinely believe that a short paper presentation is more effective. Do we really need exactly 8 (now 9? 10?) pages to describe every machine learning architecture and algorithm in existence? We proposed an incredibly simple architecture and made a very simple point that we think is worth more discussion: patches work well in convolutional architectures. We think that four pages is more than enough space for this."



# Classification problem




Goal: The goal in classification is to take an input vector $\boldsymbol{x}$ and to assign it to one of $K$ discrete classes $\mathcal{C}_k, k = 1, \ldots, K$.

Goal: The goal in regression is to take an input vector $\boldsymbol{x}$ and predict the value of a real number.

image source

# Regression

**Training data**

$$\mathcal{D} = \{ (\boldsymbol{x}_n, y_n) \mid n = 1, \dots, N \}$$

- Features/Inputs $\boldsymbol{x}_n \in \mathbb{R}^D$
- Response/Output $\boxed{y_n \in \mathbb{R}}$

# Classification

**Training data**

$$\mathcal{D} = \{\, (\boldsymbol{x}_n, y_n) \mid n = 1, \ldots, N \,\}$$

- Features/Inputs $\boldsymbol{x}_n \in \mathbb{R}^D$
- Labels/Output $\boxed{y_n \in \text{a set of discrete numbers}}$

# 4.1.1 Two classes

- Linear classifiers $h: \mathbb{R}^D \rightarrow \{-1, +1\}$

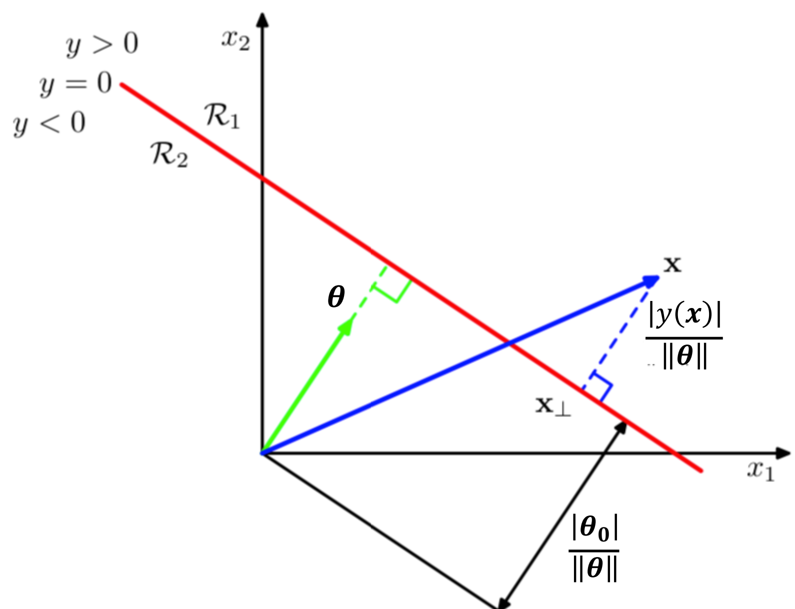
$$h(\boldsymbol{x};\ \boldsymbol{\theta}, \theta_0) = \mathrm{sign}\big(y(\boldsymbol{x})\big) = \mathrm{sign}\big(\boldsymbol{\theta}^{\mathrm{T}}\boldsymbol{x} + \theta_0\big)$$

where $\boldsymbol{\theta}$ is called a weight vector, and $\theta_0$ is a bias or offset. $y(\boldsymbol{x})$ is the discriminant function.

$$\mathrm{sign}(y) = \begin{cases} +1 & \text{if } y \geq 0, \\ -1 & \text{if } y < 0. \end{cases}$$

An input vector $\boldsymbol{x}$ is assigned to class $+1$ if $y \geq 0$ and to class $-1$ otherwise.

- The decision boundary is defined by the hyperplane $y(\boldsymbol{x}) = 0$, a $(D-1)$-dimensional hyperplane within the $D$-dimensional input space.

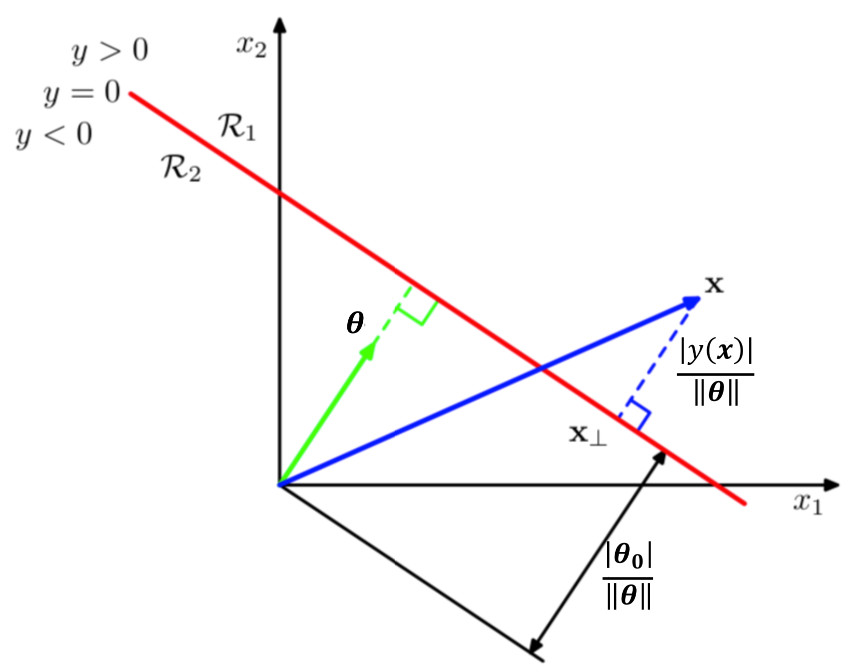


- Two points $\boldsymbol{x}_{\mathrm{A}}$ and $\boldsymbol{x}_{\mathrm{B}}$ both lie on the decision boundary $y(\boldsymbol{x}) = 0$. Then, we have $\boldsymbol{\theta}^{\mathrm{T}}(\boldsymbol{x}_{\mathrm{A}} - \boldsymbol{x}_{\mathrm{B}}) = 0$ and hence the vector $\boldsymbol{\theta}$ is orthogonal to the decision boundary.
- Consider a point $\boldsymbol{x}$ and let $\boldsymbol{x}_{\perp}$ be its orthogonal projection onto the decision boundary. The perpendicular distance between $\boldsymbol{x}$ and the decision boundary is given by,

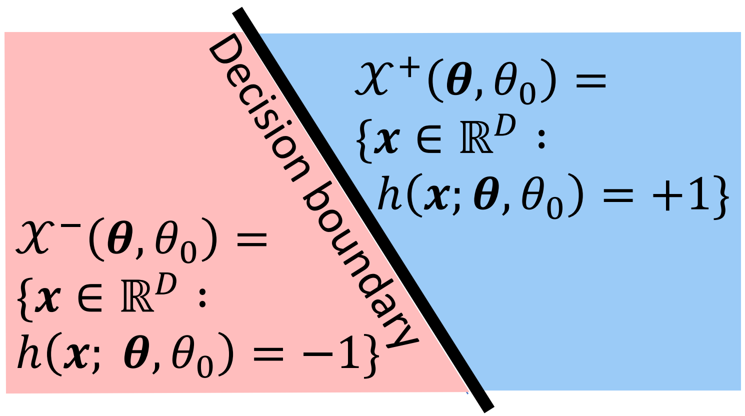
$$r = \frac{|y(\boldsymbol{x})|}{\|\boldsymbol{\theta}\|}$$

- The distance from the origin to the decision surface is given by

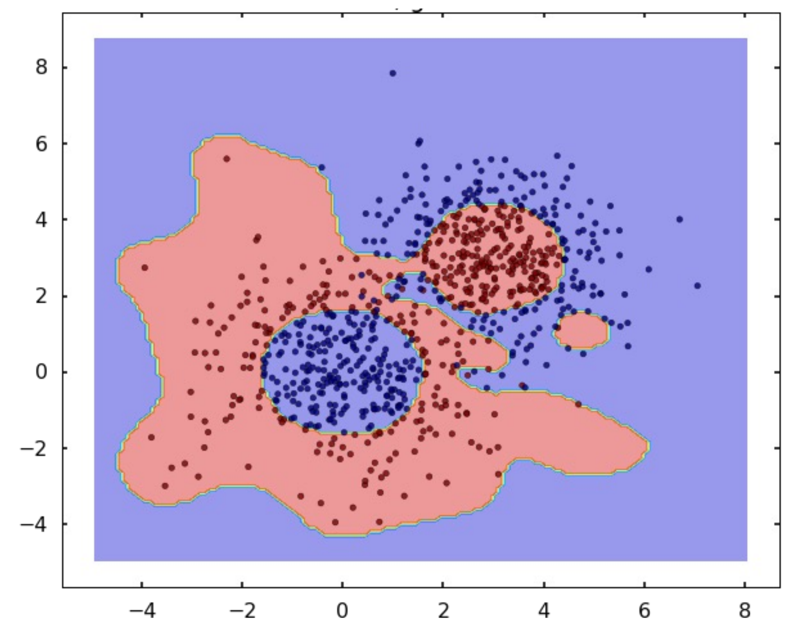
$$\frac{|\boldsymbol{\theta}_0|}{\|\boldsymbol{\theta}\|}$$



# Decision Regions




linear classifier




non-linear classifier

A classifier $h$ partitions the space into decision regions that are separated by decision boundaries. In each region, all the points map to the same label. Many regions could have the same label.

# 4.1.1 Two classes

- Compact representation

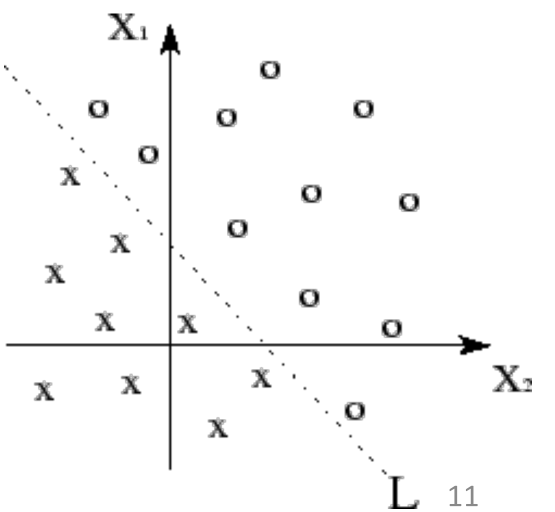
- Similar to linear regression, it is convenient to use a more compact notation in which we introduce an additional dummy 'input' value $x_0 = 1$ and then define $\boldsymbol{\theta} = [\theta_0, \theta_1, \theta_2, \dots, \theta_D]^T$ and $\boldsymbol{x} = [x_0, x_1, x_2, \dots, x_D]^T$ so that

$$y(\boldsymbol{x}) = \boldsymbol{\theta}^{\mathrm{T}}\boldsymbol{x}$$

- In this case, the decision surfaces are $D$-dimensional hyperplanes passing through the origin of the $D+1$-dimensional expanded input space.

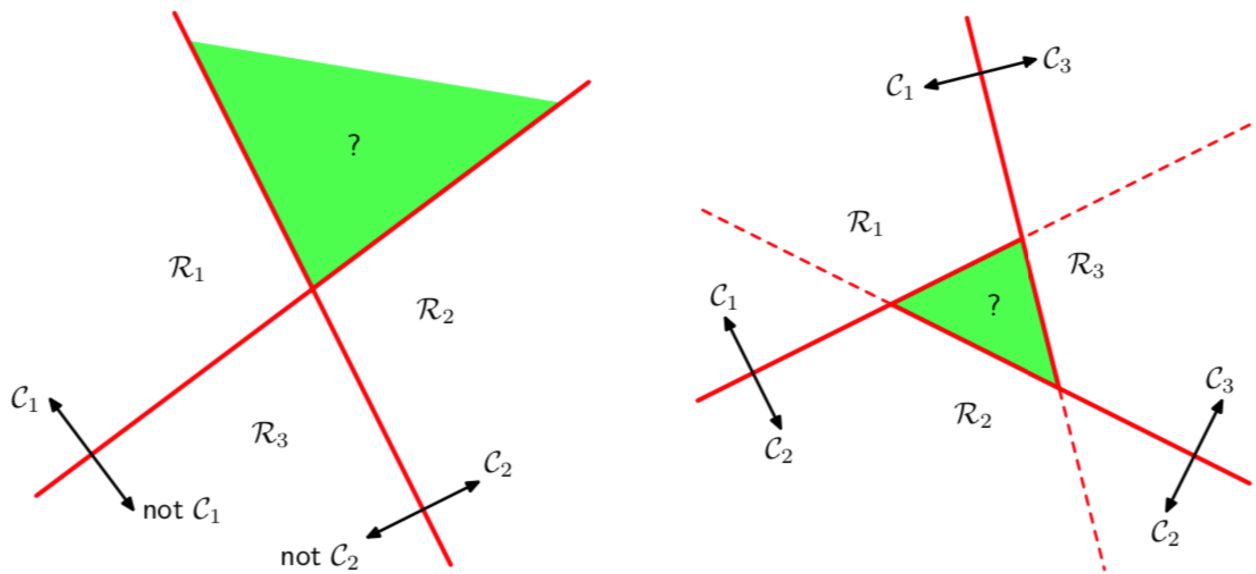
- The training data $\mathcal{D}$ is linearly separable if there exist parameters $\boldsymbol{\theta} = [\theta_0, \theta_1, \theta_2, \dots, \theta_D]^T$ such that for all $(\boldsymbol{x}, y) \in \mathcal{D}$,

$$y(\boldsymbol{\theta}^{\mathrm{T}}x) > 0$$



# 4.1.2 Multiple classes

- Now we consider a linear classifier for $K > 2$ classes.




- *One-versus-the-rest* classifier. Use $K-1$ classifiers. Each classifier solves a two-class problem: separating points in a particular class $\mathcal{C}_k$ from points not in that class.
- *One-versus-one* classifier. Use $K(K-1)/2$ classifiers, one for every possible pair of classes. Each point is classified according to a majority vote amongst the classifiers.
- However, both methods suffer from the problem of ambiguous regions.

# 4.1.2 Multiple classes

- To avoid these difficulties, we consider a single $K$-class classifier comprising $K$ linear functions of the form

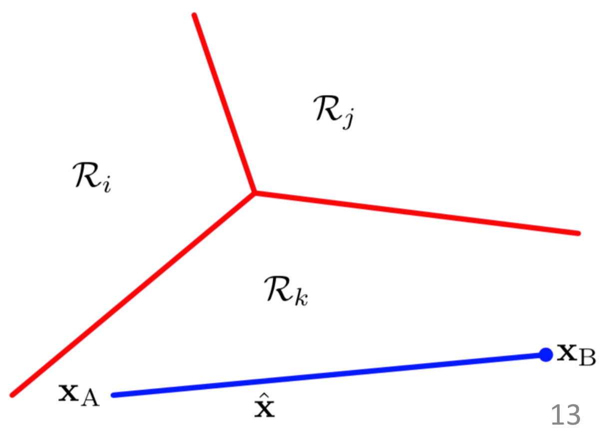
$$y_k(\boldsymbol{x}) = \boldsymbol{\theta}_k^T \boldsymbol{x} + \theta_{k0}$$

- We then assign a point $\boldsymbol{x}$ to class $\mathcal{C}_k$ if $y_k(\boldsymbol{x}) > y_j(\boldsymbol{x})$ for all $j \neq k$.
- The decision boundary between class $\mathcal{C}_k$ and class $\mathcal{C}_j$ is therefore given by $y_k(\mathbf{x}) = y_j(\mathbf{x})$ and hence corresponds to a $(D - 1)$-dimensional hyperplane defined by

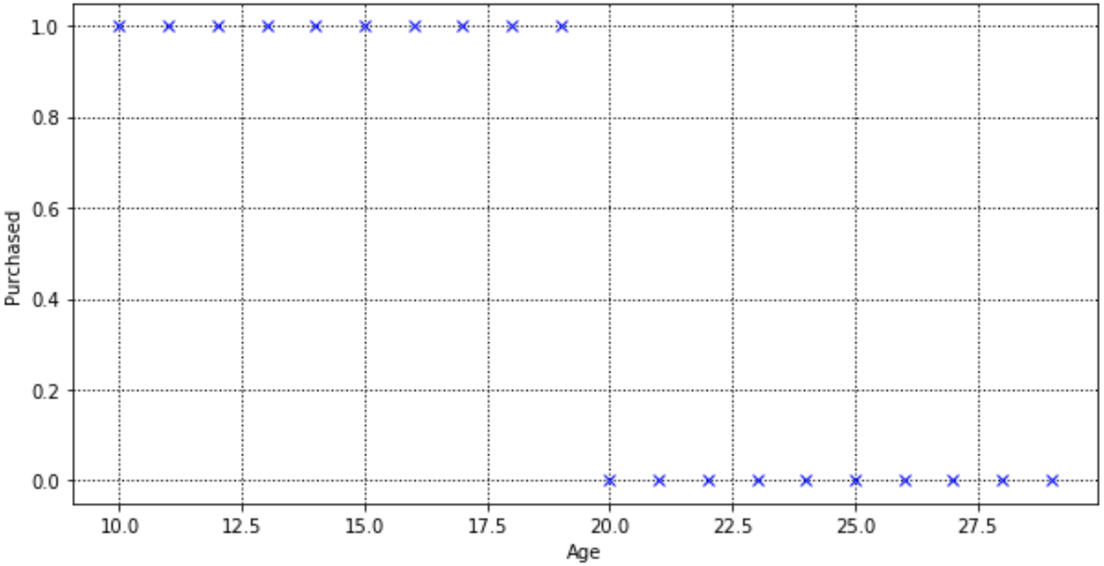
$$(\boldsymbol{\theta}_k - \boldsymbol{\theta}_j)^{\mathrm{T}} \boldsymbol{x} + (\theta_{k0} - \theta_{j0}) = 0$$

- This has the same form as the decision boundary for the two-class case discussed in Section 4.1.1, and so analogous geometrical properties apply.
- The decision regions of this classifier are singly connected and convex

"Singly connected and convex": Two points $\boldsymbol{x}_A$ and $\boldsymbol{x}_B$ both lie inside decision region $\mathcal{R}_k$. Any point $\widehat{\boldsymbol{x}}$ that lies on the line connecting $\boldsymbol{x}_A$ and $\boldsymbol{x}_B$ must also lie in $\mathcal{R}_k$

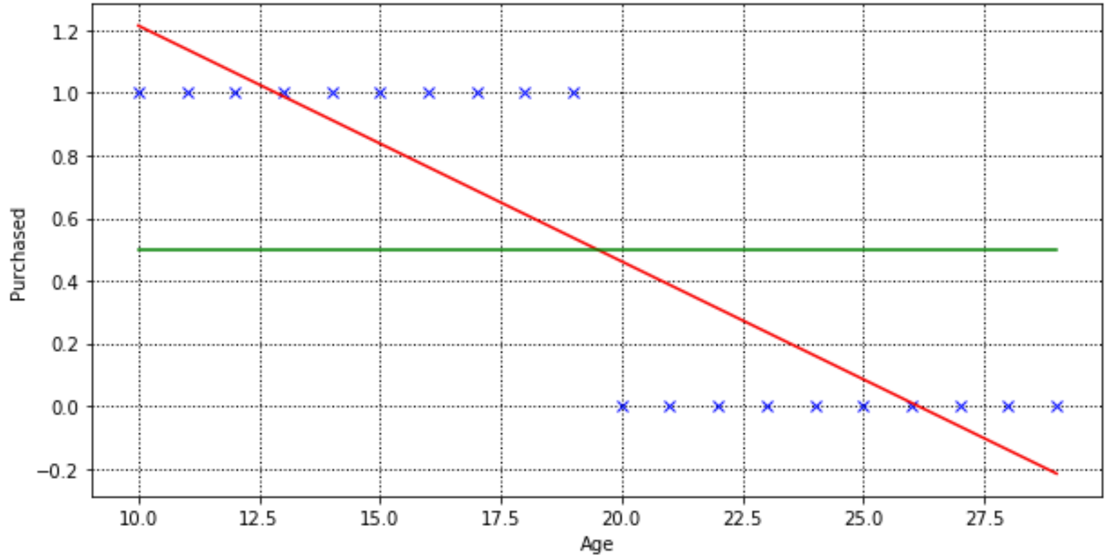


# 4.1.3 Least squares for classification




Our sample training dataset of 20 customers and thei

You can use 0.5 as threshold to tell which class a data point belongs to




https://towardsdatascience.com/why-linear-regression-is-not-suitable-for-binary-classification-c64457be8e28

# 4.1.3 Least squares for classification

- We use the tools from least squares to solve classification.
- Problem setting
- Each class $\mathcal{C}_k$ is described by its own compact linear model

$$y_k(\boldsymbol{x}) = \boldsymbol{\theta}_k^T \boldsymbol{x}$$

where $k = 1, \ldots, K$. We group these together using vector notation

$$\boldsymbol{y}(\boldsymbol{x}) = \boldsymbol{\Theta}^T \boldsymbol{x}$$

- Where $\boldsymbol{\Theta}$ is a matrix whose $k^{th}$ column comprises the $D+1$-dimensional vector $\boldsymbol{\theta}_{\boldsymbol{k}} = (\theta_{k0}, \theta_{k1}, \ldots, \theta_{kD})^{\mathrm{T}}$ and $\boldsymbol{x} = (x_0, x_1, x_2, \ldots, x_D)^{\mathrm{T}}$ is the corresponding augmented input vector with a dummy input $x_0 = 1$.
- A new input $\boldsymbol{x}$ is assigned to the class for which $y_k = \boldsymbol{\theta}_k^T \boldsymbol{x}$ is largest.

# 4.1.3 Least squares for classification

- We determine the parameter matrix $\boldsymbol{\Theta}$ by minimizing the square error function
- Consider a training data set $\{\boldsymbol{x}_n, \boldsymbol{y}_n\}$ where $n = 1, \ldots, N$, and define a matrix $\boldsymbol{Y}$ whose $n^{th}$ row is the vector $\boldsymbol{y}_n^{\mathrm{T}}$ , together with a matrix $\boldsymbol{X}$ whose $n^{th}$ row is $\boldsymbol{x}_n^{\mathrm{T}}$. The square error function is written as

$$L(\boldsymbol{\Theta}) = \frac{1}{2}\mathrm{Tr}\{(\boldsymbol{X}\boldsymbol{\Theta} - \boldsymbol{Y})^{\mathrm{T}}(\boldsymbol{X}\boldsymbol{\Theta} - \boldsymbol{Y})\}$$

- Setting the derivative with respect to $\boldsymbol{\Theta}$ to zero, and rearranging, we obtain the closed-form solution for $\boldsymbol{\Theta}$:

$$\boldsymbol{\Theta} = \left(\boldsymbol{X}^{\mathrm{T}}\boldsymbol{X}\right)^{-1}\boldsymbol{X}^{\mathrm{T}}\boldsymbol{Y} = \boldsymbol{X}^{\dagger}\boldsymbol{Y}$$

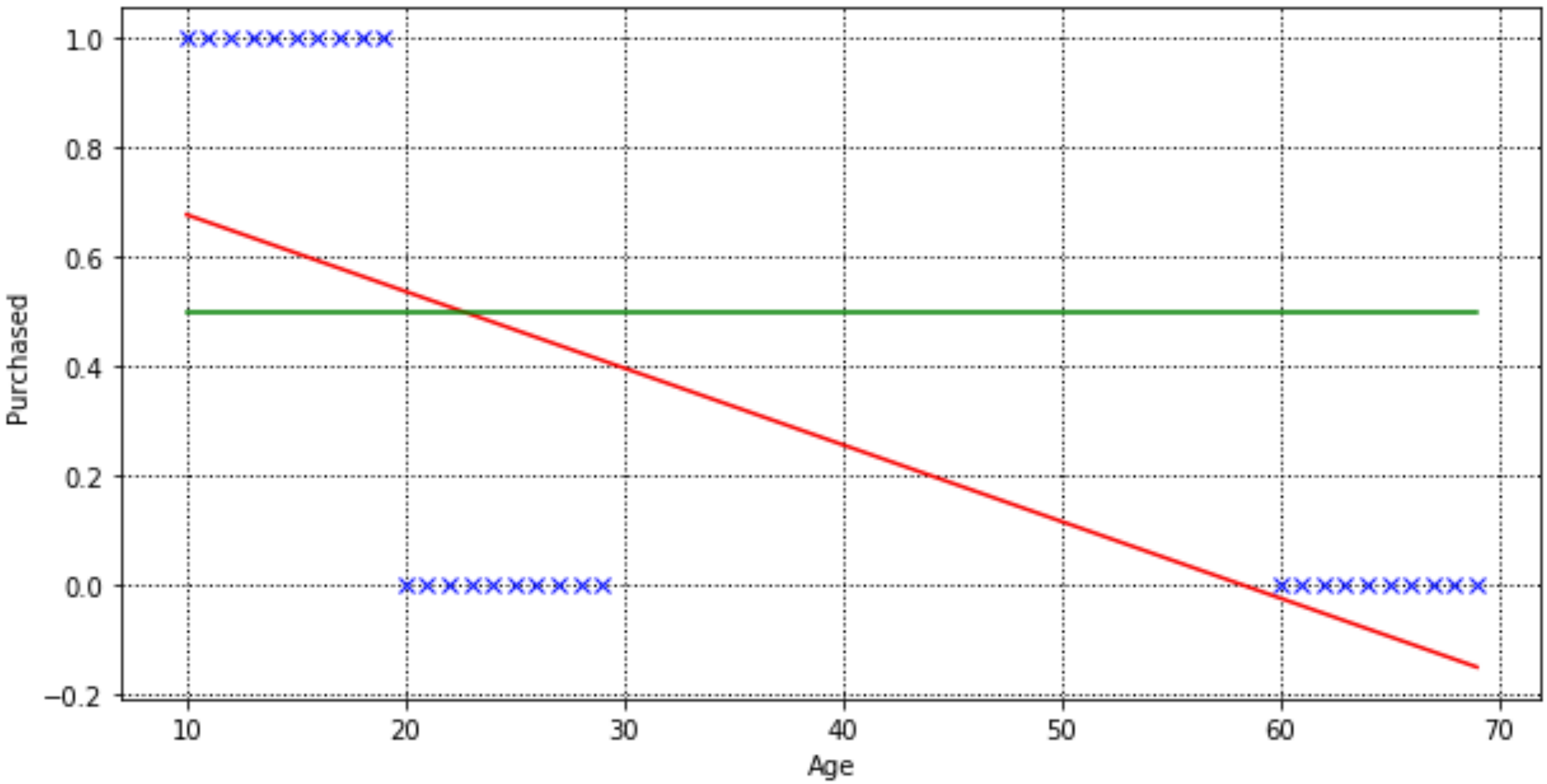
- where $\boldsymbol{X}^{\dagger}$ is the pseudo-inverse of $\boldsymbol{X}$. We obtain

$$\boldsymbol{y}(\boldsymbol{x}) = \boldsymbol{\Theta}^{T}\boldsymbol{x} = \boldsymbol{Y}^{T}(\boldsymbol{X}^{\dagger})^{T}\boldsymbol{x}$$

- $\boldsymbol{y}(\boldsymbol{x})$ is a $K\times 1$-dim vector. The predicted class label corresponds to the largest value in $\boldsymbol{y}(\boldsymbol{x})$.

# 4.1.3 Least squares for classification

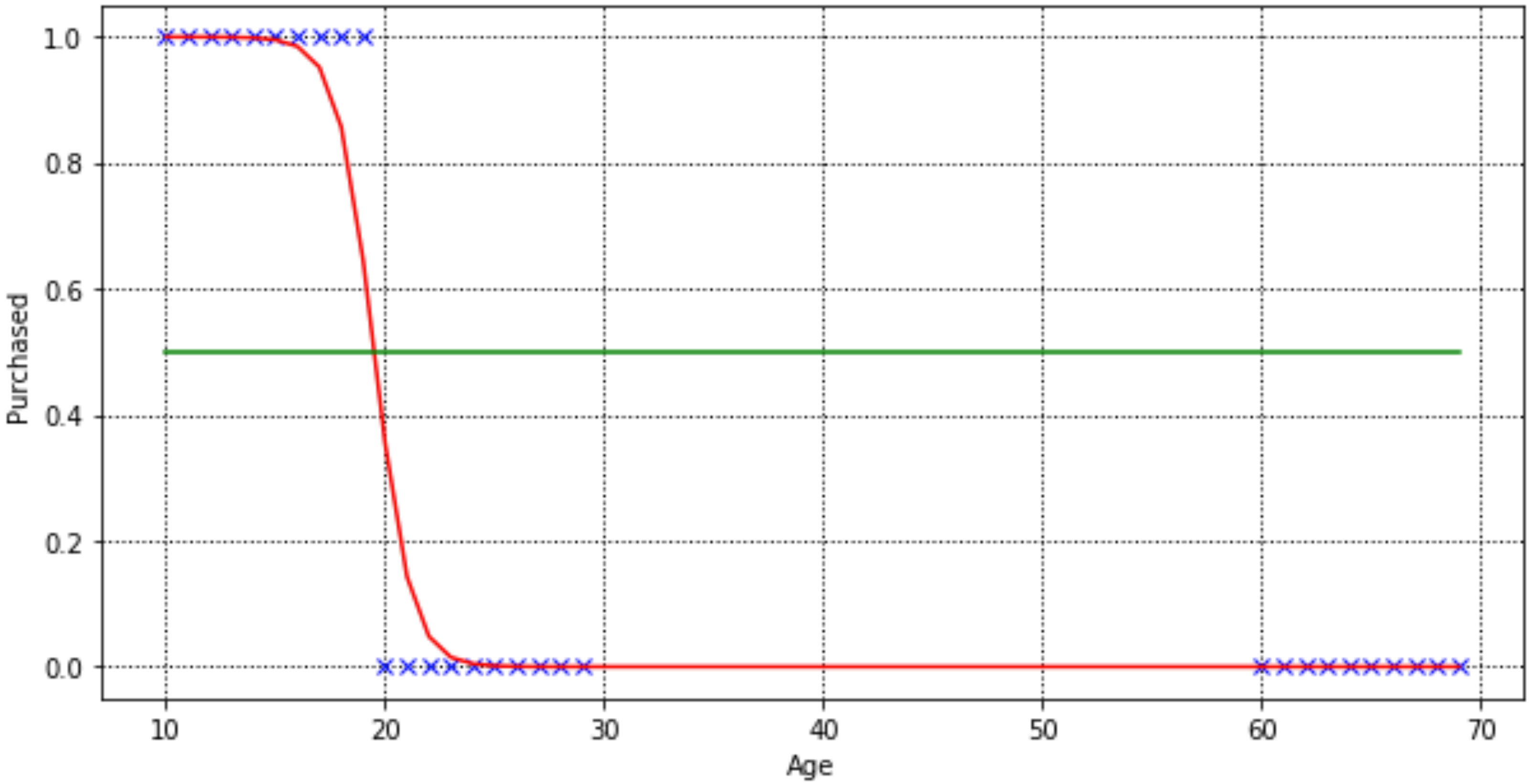
- Problems with Least squares




- If you still use 0.5 as the threshold, people between 20 and 22 will be misclassified.

# 4.1.3 Least squares for classification

- We can use Logistic regression to solve this problem
- Note: logistic regression is a classification method, not a linear regression method.



# 4.1.7 The Perceptron Algorithm

Rosenblatt (1962)

$\mathcal{L}_1(\boldsymbol{\theta}; \boldsymbol{x}, y)$ is the loss for one training sample $(\boldsymbol{x}, y)$

Classifier:

$$h(\boldsymbol{x}; \boldsymbol{\theta}) = \operatorname{sign}\left(\boldsymbol{\theta}^{\mathrm{T}} \boldsymbol{x}\right)$$

Let $\mathcal{L}_1(\boldsymbol{\theta}; \boldsymbol{x}, y) = 1$ ($0$ otherwise) if

- $y \neq h(\boldsymbol{x}; \boldsymbol{\theta})$, or [misclassified]
- $(x, y)$ is on decision boundary [boundary]

Note that $y\left(\boldsymbol{\theta}^{\mathrm{T}} \boldsymbol{x}\right) \leq 0$ if

- $\boldsymbol{\theta}^{\mathrm{T}} \boldsymbol{x}$ and $y$ differ in sign, or [misclassified]
- $\boldsymbol{\theta}^{\mathrm{T}} \boldsymbol{x}$ is zero [boundary]

$$\mathcal{L}_1(\boldsymbol{\theta}; \boldsymbol{x}, y) = [\![ y\left(\boldsymbol{\theta}^{\mathrm{T}} \boldsymbol{x}\right) \leq 0 ]\!] = \operatorname{Loss}\left(y\left(\boldsymbol{\theta}^{\mathrm{T}} \boldsymbol{x}\right)\right)$$

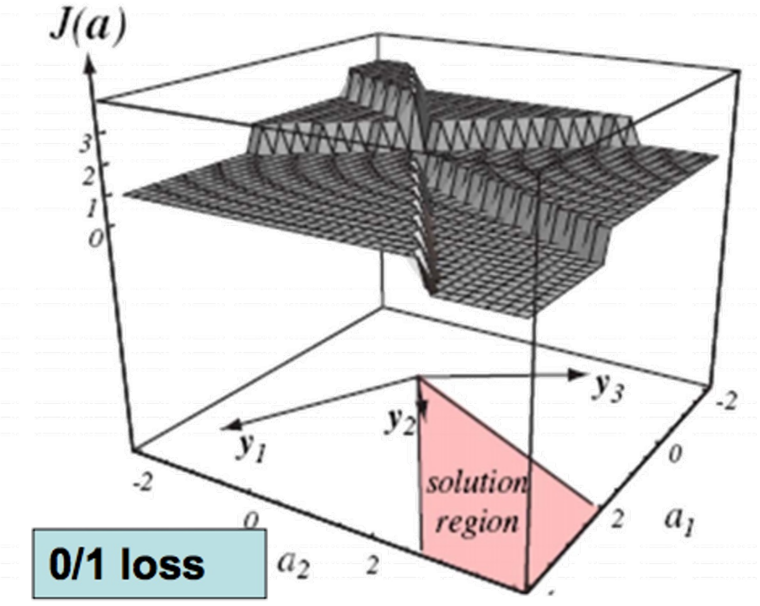
where $\operatorname{Loss}(z) = [\![ z \leq 0 ]\!]$ is the zero-one loss.

# Training Loss

$$\mathrm{Loss}(z) = [\![z \le 0]\!]$$

$$\mathcal{L}_1(\boldsymbol{\theta}; \boldsymbol{x}, y) = \mathrm{Loss}\left(y\left(\boldsymbol{\theta}^{\mathrm{T}}\boldsymbol{x}\right)\right)$$

$$\mathcal{L}_n(\boldsymbol{\theta}; \mathcal{D}) = \frac{1}{n}\sum_{(\boldsymbol{x},y)\in\mathcal{D}} \mathcal{L}_1(\boldsymbol{\theta}; \boldsymbol{x}, y)$$




Gradient is zero almost everywhere!

Gradient descent not possible.

# Perceptron – a Mistake-Driven Algorithm

1. Initialize $\boldsymbol{\theta} = \mathbf{0}$.
2. For each data $(\boldsymbol{x}, y) \in \mathcal{D}$,
   a. Check if $h(\boldsymbol{x}; \boldsymbol{\theta}) = y$.
   b. If not, update $\boldsymbol{\theta}$ to correct the mistake.
3. Repeat Step (2) until no mistakes are found.

# Perceptron Algorithm

Weights may be initialized to $\mathbf{0}$ or to a small random value

1. Initialize $\boldsymbol{\theta} = \mathbf{0}$.
2. For each data $(\boldsymbol{x}, y) \in \mathcal{S}_n$ ,
   a. If $y(\boldsymbol{\theta}^\top \boldsymbol{x}) \leq \mathbf{0}$,
      i. $\boldsymbol{\theta} \longleftarrow \boldsymbol{\theta} + y\boldsymbol{x}$.
3. Repeat Step (2) until no mistakes are found.

Due to the constant feature trick
$x_0 = 1$, update for $\theta_0$ will be
$\theta_0 \longleftarrow \theta_0 + yx_0 = \theta_0 + y$.

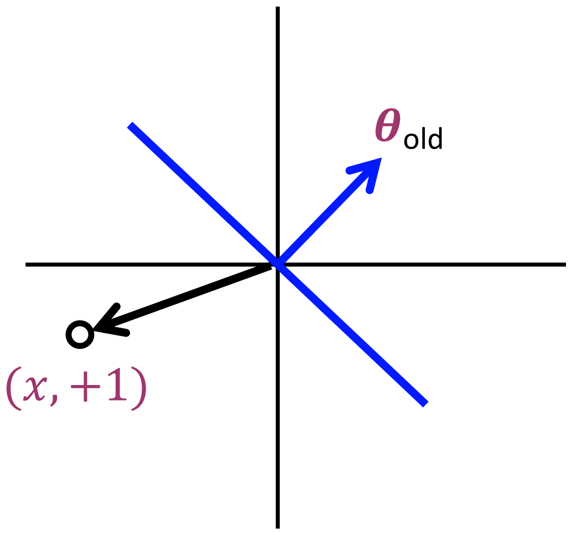
# Intuition behind $\boldsymbol{\theta} \longleftarrow \boldsymbol{\theta} + y\boldsymbol{x}$.

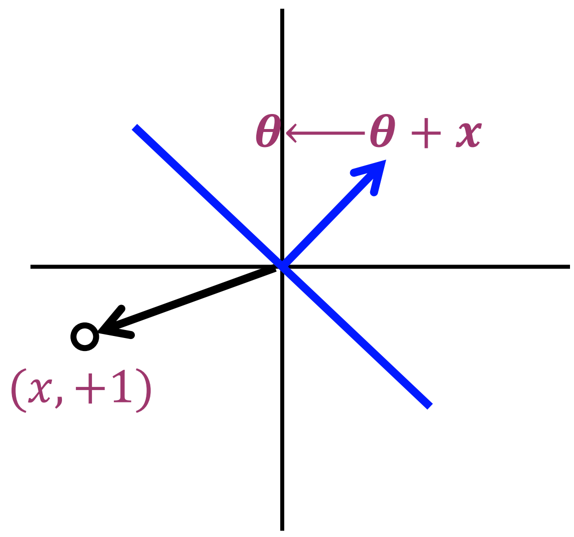
- We are making a mistake on a positive sample
- That is, $y = +1$, but $\boldsymbol{\theta}^\top \boldsymbol{x} \leq 0$
- According to this update, the new vector $\boldsymbol{\theta}_{new} = \boldsymbol{\theta}_{old} + y\boldsymbol{x} = \boldsymbol{\theta}_{old} + \boldsymbol{x}$
- The new prediction will be

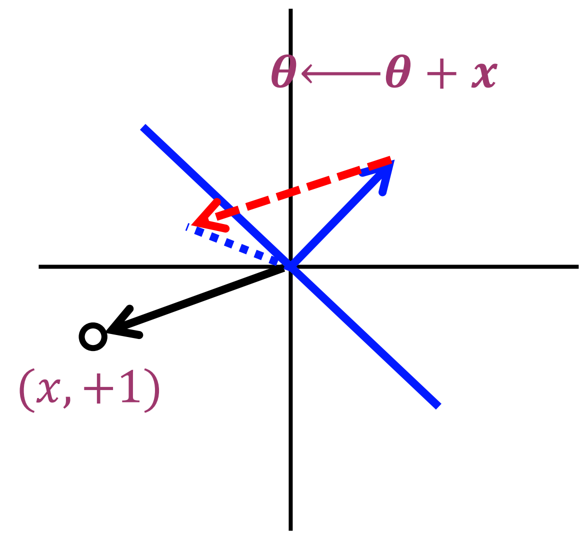
$$\boldsymbol{\theta}_{new}{}^\top \boldsymbol{x} = (\boldsymbol{\theta}_{old} + \boldsymbol{x})^\top \boldsymbol{x} = \boldsymbol{\theta}_{old}{}^\top \boldsymbol{x} + \boldsymbol{x}^\top \boldsymbol{x} > \boldsymbol{\theta}_{old}{}^\top \boldsymbol{x}$$

- For a positive example, the Perceptron update will increase the score assigned to the same input

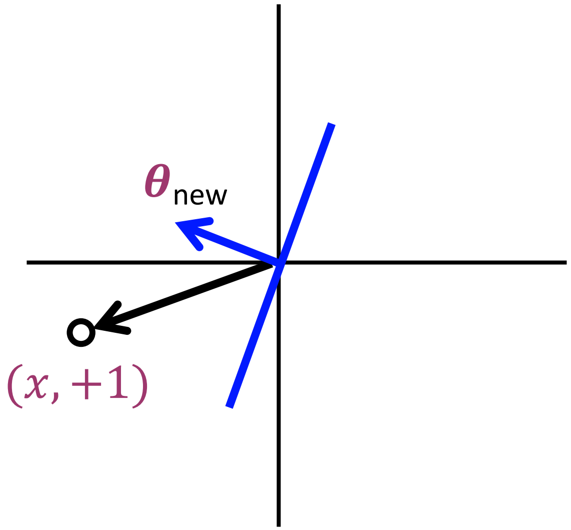
# Intuition behind $\boldsymbol{\theta} \longleftarrow \boldsymbol{\theta} + y\boldsymbol{x}$.




A mistake on a positive sample



# Example

Training data

- $(\boldsymbol{x}_1, y_1) = \left((2,\ \ 2)^\top, +1\right)$
- $(\boldsymbol{x}_2, y_2) = \left((2, -1)^\top, -1\right)$

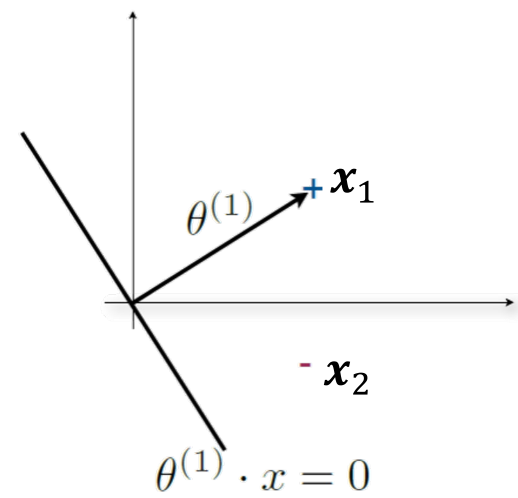
Apply the perceptron algorithm to the data to find a classifier $h(\boldsymbol{x}; \boldsymbol{\theta}), \boldsymbol{\theta} = (\theta_1, \theta_2)$, that separates the data.

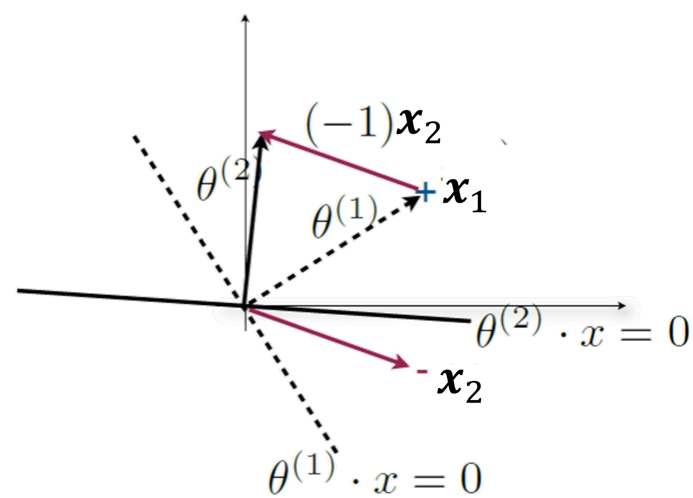
# Example

Training data

- $(\boldsymbol{x}_1, y_1) = ((2,\ \ 2)^\top, +1)$
- $(\boldsymbol{x}_2, y_2) = ((2, -1)^\top, -1)$

- Initialize $\boldsymbol{\theta} = (0,0)^\top$.
- Since $y_1 \boldsymbol{\theta}^\top \boldsymbol{x}_1 = 0$,
  set $\boldsymbol{\theta} = (0,0) + (2,2)^\top = (2,2)^\top$.
- Since $y_2 \boldsymbol{\theta}^\top \boldsymbol{x}_2 = -2$,
  set $\boldsymbol{\theta} = (2,2)^\top - (2,-1)^\top = (0,3)^\top$.
- $y_1 \boldsymbol{\theta}^\top \boldsymbol{x}_1 = 6 > 0$.
- $y_2 \boldsymbol{\theta}^\top \boldsymbol{x}_2 = 3 > 0$.
- No more mistakes, so we are done.



# Perceptron Algorithm

1. Training Set (Linearly Separable)

$$(\boldsymbol{x}_1, y_1), (\boldsymbol{x}_2, y_2), \dots, (\boldsymbol{x}_N, y_N)$$

2. Model (Set of Perceptrons)

$$h(\boldsymbol{x}; \boldsymbol{\theta}) = \operatorname{sign}(\theta_0 x_0 + \theta_1 x_1 + \cdots + \theta_D x_D)$$

3. Training Loss (Fraction of Misclassified/Boundary Points)

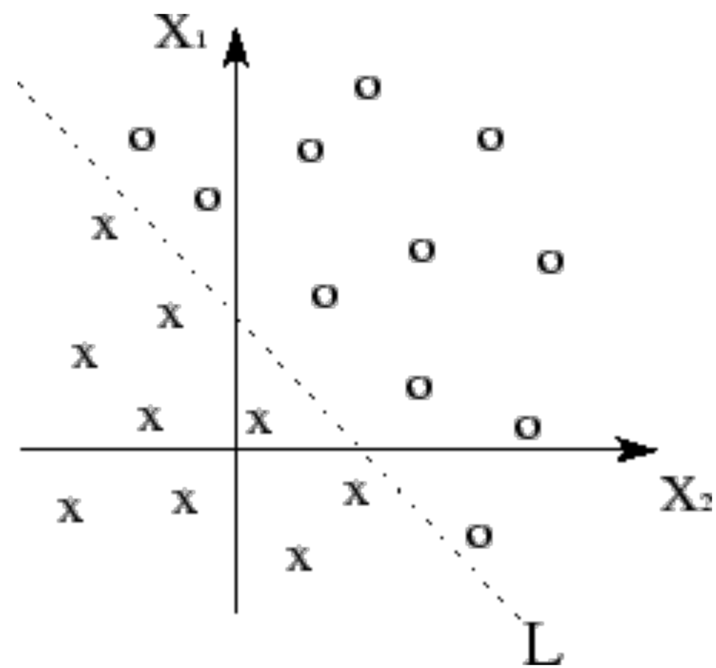
$$\mathcal{L}_n(\boldsymbol{\theta}) = \frac{1}{N} \sum_{(\boldsymbol{x}, y) \in \mathcal{D}} [\![\, y(\boldsymbol{\theta}^\top \boldsymbol{x}) \leq 0 \,]\!]$$

3. Algorithm (Mistake-Driven Algorithm)

# Linearly Separable

The training data $\mathcal{D}$ is
linearly separable
if there exists a
parameters $\boldsymbol{\theta}$ and $\theta_0$ such
that for all $(\boldsymbol{x}, y) \in \mathcal{D}$,

$$y(\boldsymbol{\theta}^\top \boldsymbol{x} + \theta_0) > 0.$$

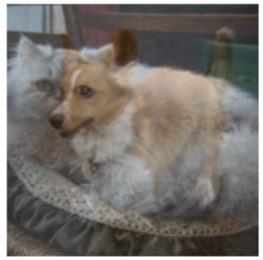



Perceptron algorithm can only be applied to dataset that is linearly separable

# Check your understanding

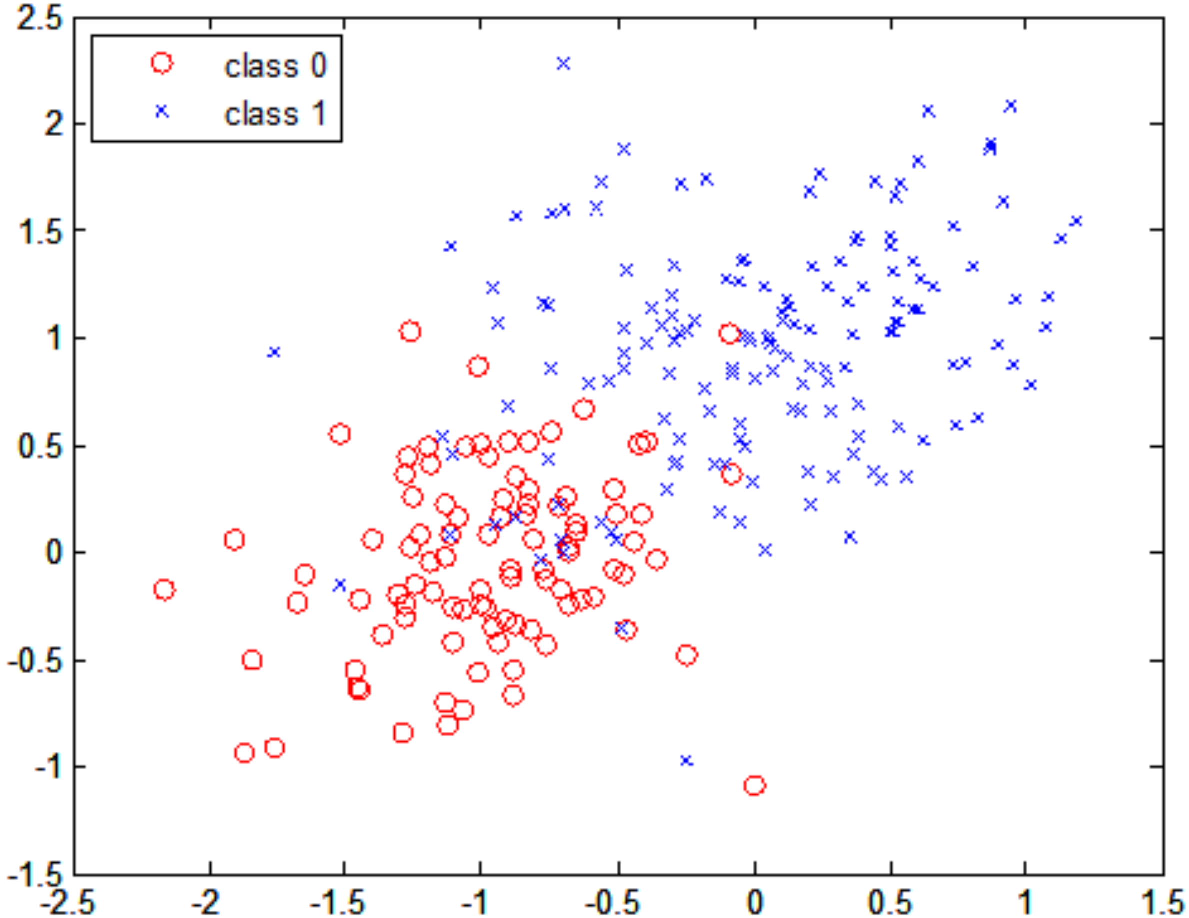
- Classification deals with discrete label space.
- We can do multi-way classification using a single classifier.
- In multi-way classification $\boldsymbol{y}(\boldsymbol{x}) = \boldsymbol{\Theta}^T \boldsymbol{x}$, the classifier weight $\boldsymbol{\theta}_k$ can be shared by different classes.
- We can devise a classifier based on linear regression.
- In Perceptron algorithm, whenever a sample is misclassified, one step of $\boldsymbol{\theta}$ update $\boldsymbol{\theta} \longleftarrow \boldsymbol{\theta} + y\boldsymbol{x}$ will correct it.
- Trained on a linearly separable training set, a perceptron classifier will make no mistake on both training and test sets.

# Data augmentation

| | ResNet-50 | Mixup [48] | Cutout [3] | CutMix |
|---|---|---|---|---|
| Image |  |  |  |  |
| Label | Dog 1.0 | Dog 0.5<br>Cat 0.5 | Dog 1.0 | Dog 0.6<br>Cat 0.4 |
| ImageNet<br>Cls (%) | 76.3<br>(+0.0) | 77.4<br>(+1.1) | 77.1<br>(+0.8) | **78.6**<br>**(+2.3)** |
| ImageNet<br>Loc (%) | 46.3<br>(+0.0) | 45.8<br>(-0.5) | 46.7<br>(+0.4) | **47.3**<br>**(+1.0)** |
| Pascal VOC<br>Det (mAP) | 75.6<br>(+0.0) | 73.9<br>(-1.7) | 75.1<br>(-0.5) | **76.7**<br>**(+1.1)** |

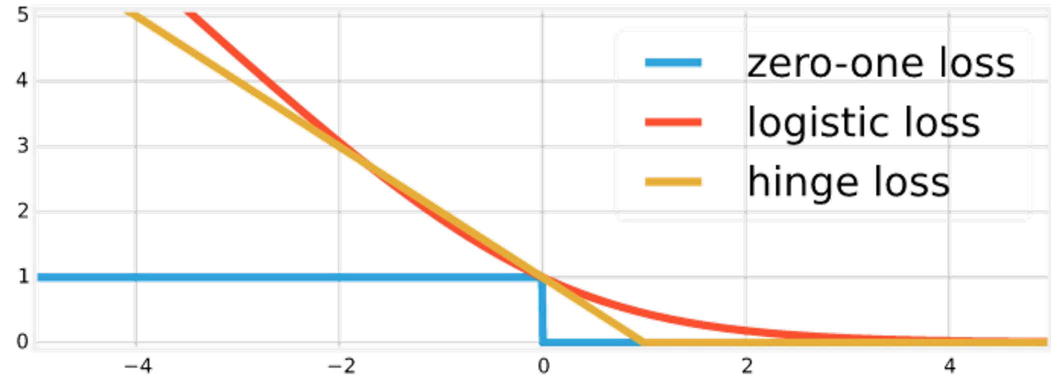
Zhang et al., mixup: Beyond Empirical Risk Minimization. ICLR 2018
Yun et al., CutMix: Regularization Strategy to Train Strong Classifiers with Localizable Features. ICCV 2019.
Zhong et al., Random Erasing Data Augmentation. AAAI 2020.

# Hinge Loss




Perceptron algorithm does not converge for training sets that are not linearly separable.

# Loss Functions




Training Loss

$$\mathcal{L}_n(\boldsymbol{\theta}) = \frac{1}{N}\sum_{(\boldsymbol{x},y)\in\mathcal{D}} \mathrm{Loss}\left(\boxed{y(\boldsymbol{\theta}^\top \boldsymbol{x})}\right) \longrightarrow z$$

Zero-One Loss

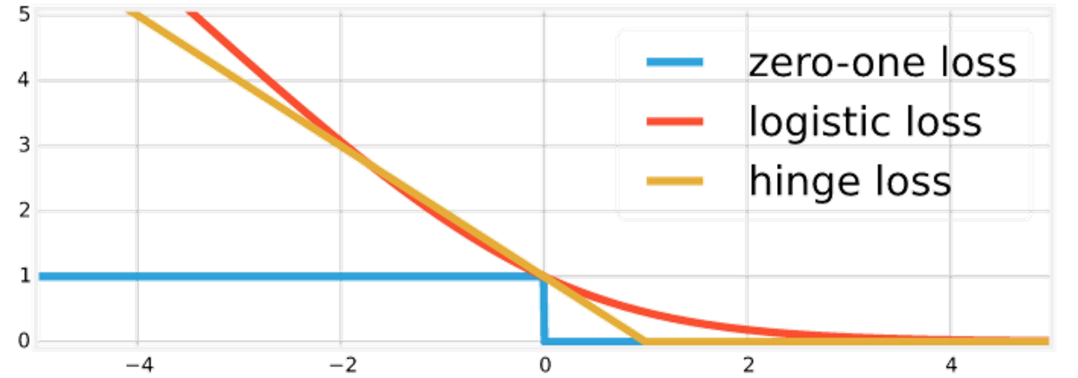
$$\mathrm{Loss}_{01}(z) = [\![\, z \leq 0 \,]\!]$$

Hinge Loss

$$\mathrm{Loss}_{\mathrm{H}}(z) = \max\{1 - z, 0\}$$

**CONVEX!**
Penalize large mistakes more.
Penalize near-mistakes, i.e. $0 \leq z \leq 1$.

# Hinge Loss




Find $\boldsymbol{\theta}$ that minimizes

$$\mathcal{L}_n(\boldsymbol{\theta}) = \frac{1}{N}\sum_{(\boldsymbol{x},y)\in\mathcal{D}} \mathrm{Loss_H}(z)$$
$$= \frac{1}{N}\sum_{(\boldsymbol{x},y)\in\mathcal{D}} \max\{1-z, 0\}$$
$$= \frac{1}{N}\sum_{(\boldsymbol{x},y)\in\mathcal{D}} \max\{1-y(\boldsymbol{\theta}^\top \boldsymbol{x}), 0\}$$

Gradient

$$\nabla_{\mathrm{z}}\mathrm{Loss_H}(z) = \begin{cases} 0 & \text{if } z > 1, \\ -1 & \text{otherwise.} \end{cases}$$

$$\nabla_{\theta}\mathrm{Loss_H}\big(y(\boldsymbol{\theta}^\top \boldsymbol{x})\big) = \begin{cases} 0 & \text{if } y(\boldsymbol{\theta}^\top \boldsymbol{x}) > 1, \\ -y\boldsymbol{x} & \text{otherwise.} \end{cases}$$

# Stochastic Gradient Descent

1. Initialize $\boldsymbol{\theta} = \mathbf{0}$.
2. Select data $(\boldsymbol{x}, y) \in \mathcal{D}$ at random.
   a. If $y(\boldsymbol{\theta}^\top \boldsymbol{x}) \leq 1$, then
      i. $\boldsymbol{\theta} \longleftarrow \boldsymbol{\theta} + \eta_k \, y\boldsymbol{x}.$
3. Repeat Step (2) until convergence.
   (e.g., when improvement in $\mathcal{L}\,(\boldsymbol{\theta})$ is small enough)

Differences from Perceptron Algorithm

- Check $z \leq 1$ rather than $z \leq 0$
- $\eta_k$ rather than $\eta = 1$

# Hinge Loss Algorithm

1. Training Set (Not Necessarily Linearly Separable)

$$(\boldsymbol{x}_1, y_1), (\boldsymbol{x}_2, y_2), \dots, (\boldsymbol{x}_N, y_N)$$

2. Model (Set of Perceptrons)

$$h(\boldsymbol{x}; \boldsymbol{\theta}) = \mathrm{sign}(\theta_0 x_0 + \theta_1 x_1 + \cdots + \theta_D x_D)$$

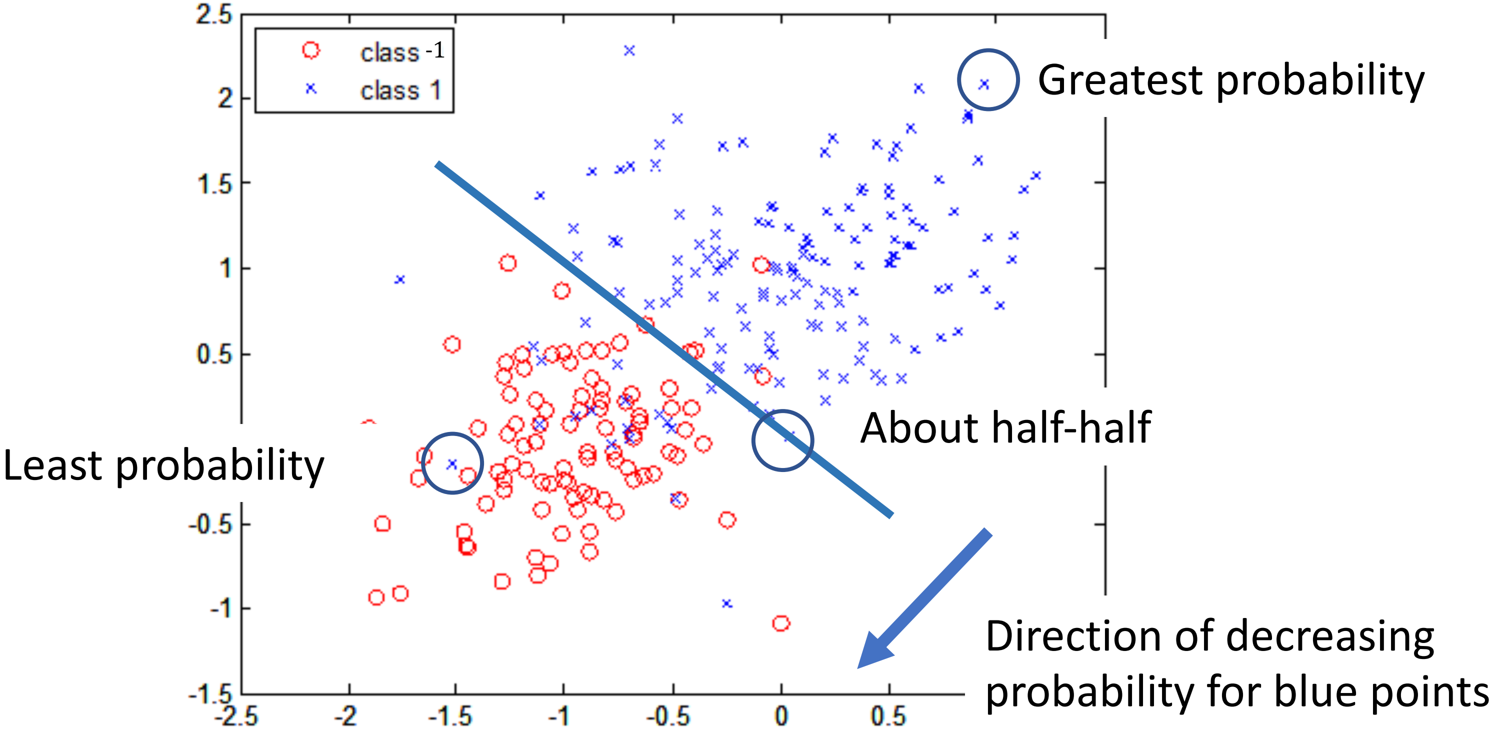
3. Training Loss (Hinge Loss)

$$\mathcal{L}_n(\boldsymbol{\theta}) = \frac{1}{N} \sum_{(\boldsymbol{x},y)\in\mathcal{D}} \max\{1 - y(\boldsymbol{\theta}^\top \boldsymbol{x}), 0\}$$

3. Algorithm (Gradient Descent)

$$\boldsymbol{\theta} \longleftarrow \boldsymbol{\theta} + \frac{\eta_k}{N} \sum_{(\boldsymbol{x},y)\in\mathcal{D}} y\boldsymbol{x}$$

# Logistic Regression



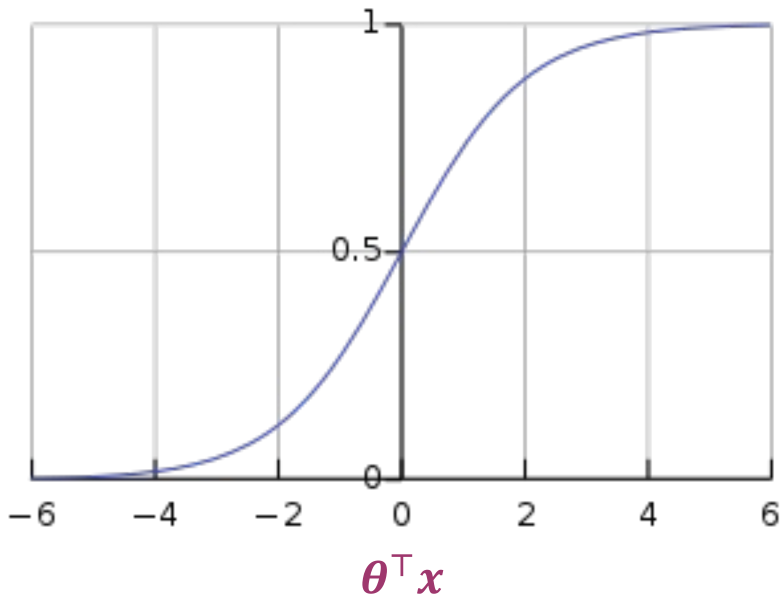
# Probabilistic Model

Model the probability that the label $y$ is $+1$ given the feature is $\boldsymbol{x}$.

$$h: \mathbb{R}^D \to [0, 1]$$

$$h(\boldsymbol{x}; \boldsymbol{\theta}) = \mathbb{P}(\, y = +1 \,|\boldsymbol{x}) = \mathrm{sigmoid}(\boldsymbol{\theta}^\top \boldsymbol{x})$$

For small $\boldsymbol{\theta}^\top \boldsymbol{x}$, the label is very likely to be $-1$.




For large $\boldsymbol{\theta}^\top \boldsymbol{x}$, the label is very likely to be $+1$.

# Sigmoid function

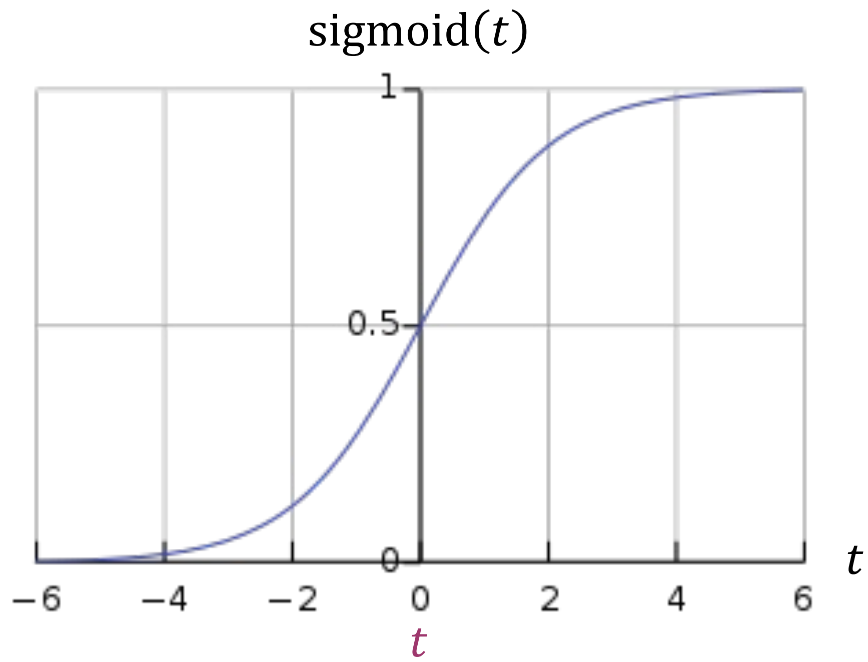
$$\mathrm{sigmoid}(t) = \frac{1}{1 + e^{-t}}$$

$$\mathrm{sigmoid}: \mathbb{R} \rightarrow [0, 1]$$

Sometimes also known as the logistic function.

Super useful formula

$$\mathrm{sigmoid}(-t) = \frac{1}{1+e^{t}} = \frac{e^{-t}}{e^{-t}+1} = 1 - \frac{1}{e^{-t}+1} = 1 - \mathrm{sigmoid}(t)$$



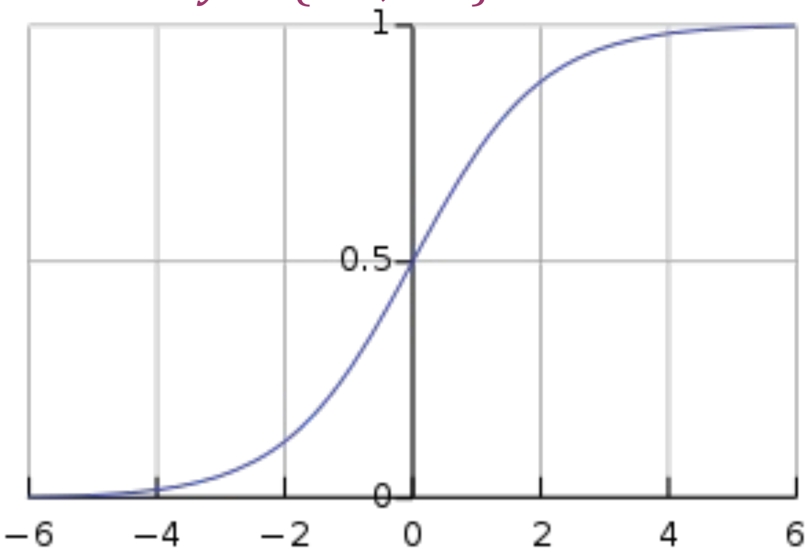
# Label Probabilities

$\mathbb{P}(y = +1 \mid \boldsymbol{x}) = \text{sigmoid}(\boldsymbol{\theta}^\top \boldsymbol{x}) = \text{sigmoid}\big(\mathrm{y}(\boldsymbol{\theta}^\top \boldsymbol{x})\big)$

To get the other label probability,

$$\begin{aligned}\mathbb{P}(\, y = -1 \mid \boldsymbol{x}) \quad &= 1 - \mathbb{P}(\, y = +1 \mid \boldsymbol{x}) \\ &= 1 - \text{sigmoid}(\boldsymbol{\theta}^\top \boldsymbol{x}) \\ &= \text{sigmoid}(-\boldsymbol{\theta}^\top \boldsymbol{x}) = \text{sigmoid}\big(\mathrm{y}(\boldsymbol{\theta}^\top \boldsymbol{x})\big)\end{aligned}$$

Thus, $\mathbb{P}(y|\boldsymbol{x}) = \text{sigmoid}\big(\mathrm{y}(\boldsymbol{\theta}^\top \boldsymbol{x})\big)$ for both $y \in \{+1, -1\}$.



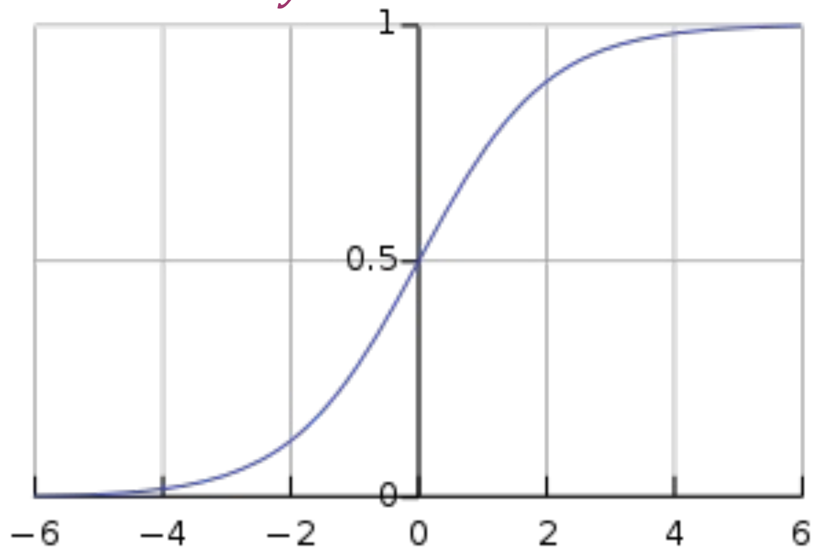
# Label Predictions

Find out which label is more probable.

$$\mathbb{P}(\, y = +1 \mid \boldsymbol{x} \,) \geq \mathbb{P}(\, y = -1 \mid \boldsymbol{x} \,) \iff h(\boldsymbol{x}; \boldsymbol{\theta}) \geq \frac{1}{2}$$

If $h(\boldsymbol{x}; \boldsymbol{\theta}) \geq \frac{1}{2}$, then we predict the label of $\boldsymbol{x}$ is $y = +1$.

If $h(\boldsymbol{x}; \boldsymbol{\theta}) < \frac{1}{2}$, then we predict the label of $\boldsymbol{x}$ is $y = -1$.

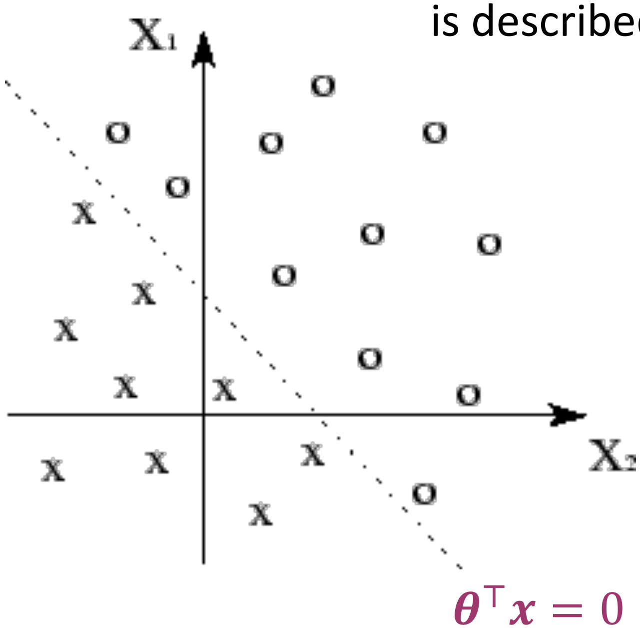


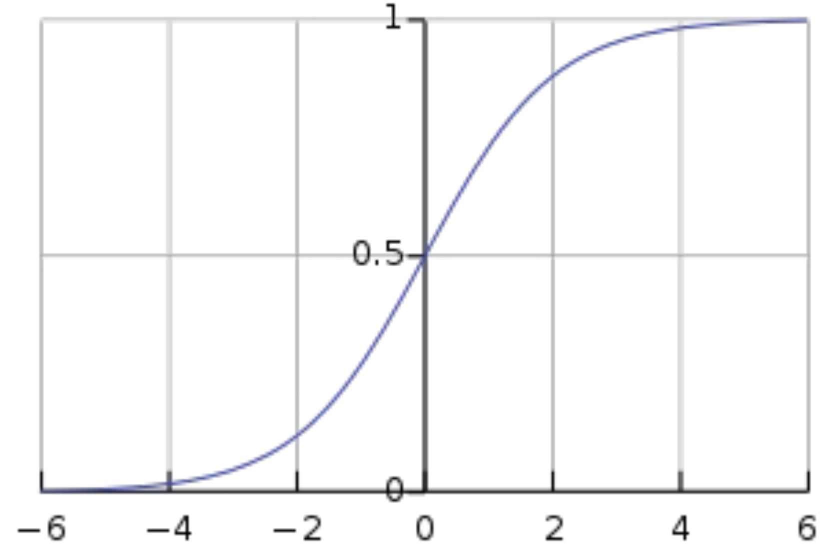
# Decision Boundary

$$h(\boldsymbol{x};\boldsymbol{\theta}) \geq \frac{1}{2} \iff \text{sigmoid}(\boldsymbol{\theta}^\top \boldsymbol{x}) \geq \frac{1}{2} \iff \boldsymbol{\theta}^\top \boldsymbol{x} \geq 0$$

$$h(\boldsymbol{x};\boldsymbol{\theta}) < \frac{1}{2} \iff \text{sigmoid}(\boldsymbol{\theta}^\top \boldsymbol{x}) < \frac{1}{2} \iff \boldsymbol{\theta}^\top \boldsymbol{x} < 0$$

The decision boundary is described by $\boldsymbol{\theta}^\top \boldsymbol{x} = 0$.



# Likelihood

Probability of label $y$ given feature $\boldsymbol{x}$

$$\mathbb{P}(y|\boldsymbol{x}) = \text{sigmoid}\big(y(\boldsymbol{\theta}^\top \boldsymbol{x})\big).$$

$L(\boldsymbol{\theta}; \mathcal{D})$ is called the *likelihood* of the dataset $\mathcal{D}$.

Probability of labels $y_1, \dots, y_N$ given features $\boldsymbol{x}_1, \dots, \boldsymbol{x}_N$

$$\begin{aligned} L(\boldsymbol{\theta}; \mathcal{D}) &= \mathbb{P}(y_1, \dots, y_N | \boldsymbol{x}_1, \dots, \boldsymbol{x}_N) \\ &= \mathbb{P}(y_1|\boldsymbol{x}_1) \times \cdots \times \mathbb{P}(y_N|\boldsymbol{x}_N) \\ &= \prod_{(\boldsymbol{x},y)\in\mathcal{D}} \mathbb{P}(y|\boldsymbol{x}) \end{aligned}$$

Maximizing $L(\boldsymbol{\theta}; \mathcal{D})$ is the same as maximizing $\log L(\boldsymbol{\theta}; \mathcal{D})$,
which is the same as minimizing $-\frac{1}{N}\log L(\boldsymbol{\theta}; \mathcal{D})$.

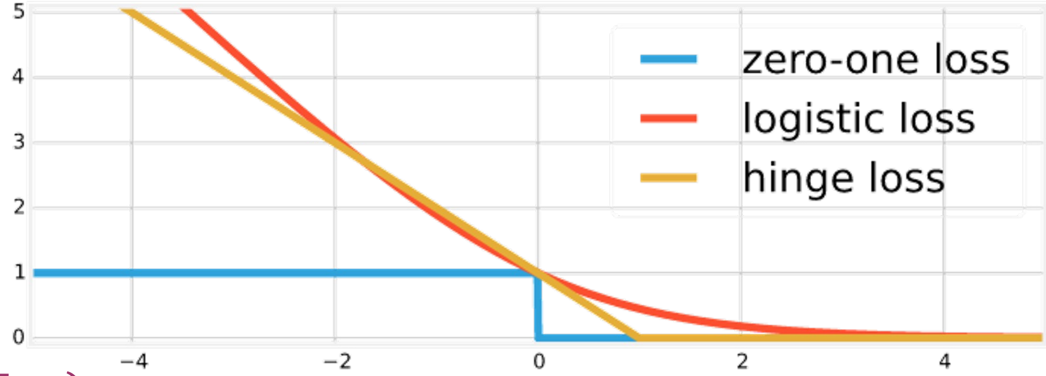
# Logistic Loss

Minimize the training loss

$$\begin{aligned}
\mathcal{L}(\boldsymbol{\theta}) &= -\frac{1}{N}\log L(\boldsymbol{\theta};\mathcal{D}) \\
&= -\frac{1}{N}\log \prod_{(\boldsymbol{x},y)\in\mathcal{D}} \mathbb{P}(y|\boldsymbol{x}) \\
&= -\frac{1}{N}\sum_{(\boldsymbol{x},y)\in\mathcal{D}} \log \mathbb{P}(y|\boldsymbol{x}) \\
&= -\frac{1}{N}\sum_{(\boldsymbol{x},y)\in\mathcal{D}} \log \frac{1}{1+e^{-y(\boldsymbol{\theta}^\top \boldsymbol{x})}} \\
&= \frac{1}{N}\sum_{(\boldsymbol{x},y)\in\mathcal{D}} \log\left(1+e^{-y(\boldsymbol{\theta}^\top \boldsymbol{x})}\right) \\
&= \frac{1}{N}\sum_{(\boldsymbol{x},y)\in\mathcal{D}} \mathrm{Loss}\left(y(\boldsymbol{\theta}^\top \boldsymbol{x})\right)
\end{aligned}$$

$\mathrm{Loss}(z) = \log(1+e^{-z})$ is the *logistic loss*.

# Loss Functions




Training Loss

$$\mathcal{L}(\boldsymbol{\theta}) = \frac{1}{N} \sum_{(\boldsymbol{x},y)\in\mathcal{D}} \mathrm{Loss}\left(y(\boldsymbol{\theta}^\top \boldsymbol{x})\right)$$

$$\boxed{z = y(\boldsymbol{\theta}^\top \boldsymbol{x})}$$

Zero-One Loss

$$\mathrm{Loss}_{01}(z) = [\![\, z \leq 0 \,]\!]$$

Perceptron Algorithm

Hinge Loss

$$\mathrm{Loss}_{\mathrm{H}}(z) = \max\{1 - z, 0\}$$

Hinge Loss Algorithm

Logistic Loss

$$\mathrm{Loss}_{\mathrm{L}}(z) = \log(1 + e^{-z})$$

Logistic Regression

Some folks use $\log_2$ instead of $\log_e$ so that $\mathrm{Loss}_{\mathrm{L}}(0) = 1$.

# Gradient

$$\boxed{h(\boldsymbol{x}; \boldsymbol{\theta}) = \mathbb{P}(\, y = +1 \,| \boldsymbol{x}) = \mathrm{sigmoid}(\boldsymbol{\theta}^\top \boldsymbol{x})}$$

$$\boxed{\mathrm{sigmoid}(-t) = 1 - \mathrm{sigmoid}(t)}$$

$$\mathrm{Loss_L}(z) = \log(1 + e^{-z})$$

$$\nabla_z \mathrm{Loss_L}(z) = \frac{-e^{-z}}{1+e^{-z}} = \frac{-1}{e^z+1} = -\mathrm{sigmoid}(-z) = \mathrm{sigmoid}(z) - 1$$

By chain rule, the gradient for $1$ training sample $\nabla_\theta \mathcal{L}_1(\boldsymbol{\theta}; \boldsymbol{x}, y\,)$ is

$$\nabla_\theta \mathrm{Loss_L}\big(y(\boldsymbol{\theta}^\top \boldsymbol{x})\big) = y\boldsymbol{x}(\mathrm{sigmoid}(y\boldsymbol{\theta}^\top \boldsymbol{x}) - 1)$$

$$= \begin{cases} \boldsymbol{x}(h(\boldsymbol{x}; \boldsymbol{\theta}) - 1) & \text{if } y = +1, \\ \boldsymbol{x}(h(\boldsymbol{x}; \boldsymbol{\theta}) - 0) & \text{if } y = -1. \end{cases}$$

$$= \boldsymbol{x}\,(h(\boldsymbol{x}; \boldsymbol{\theta}) - [\![y = 1]\!])$$

# Training Gradient

$$\nabla_{\boldsymbol{\theta}} \mathcal{L}(\boldsymbol{\theta}; \mathcal{D}) = \frac{1}{N} \sum_{(\boldsymbol{x},y)\in\mathcal{D}} \nabla_{\boldsymbol{\theta}} \mathcal{L}_1(\boldsymbol{\theta}; \boldsymbol{x}, y)$$
$$= \frac{1}{N} \sum_{(\boldsymbol{x},y)\in\mathcal{D}} \boldsymbol{x}(h(\boldsymbol{x}; \boldsymbol{\theta}) - [\![y = 1]\!])$$

Compare this with the training gradient for least squares

$$\nabla_{\boldsymbol{\theta}} \mathcal{L}(\boldsymbol{\theta}; \mathcal{D}) = \frac{1}{N} \sum_{(\boldsymbol{x},y)\in\mathcal{D}} \boldsymbol{x}(f(\boldsymbol{x}; \boldsymbol{\theta}) - y)$$

In regression, we have:

$$L(\boldsymbol{\theta}, \boldsymbol{X}, \boldsymbol{y}) = \frac{1}{N} \sum_{n=1}^{N} \left(y_n - f(\boldsymbol{x}_n, \boldsymbol{\theta})\right)^2$$
$$f(\boldsymbol{x}_n, \boldsymbol{\theta}) = \boldsymbol{\theta}^{\boldsymbol{T}} \boldsymbol{x}_n$$

# Gradient Descent

1. Initialize $\boldsymbol{\theta}$ randomly.
2. Update $\boldsymbol{\theta} \longleftarrow \boldsymbol{\theta} - \frac{\eta_k}{N} \sum_{(\boldsymbol{x},y)\in\mathcal{D}} \boldsymbol{x}(h(\boldsymbol{x};\boldsymbol{\theta}) - [\![y = 1]\!])$
3. Repeat (2) until convergence.
   (e.g., when improvement in $\mathcal{L}(\boldsymbol{\theta};\mathcal{D})$ is small enough)

# Logistic Regression

1. Training Set (Not Necessarily Linearly Separable)

$$(\boldsymbol{x}_1, y_1), (\boldsymbol{x}_2, y_2), \dots, (\boldsymbol{x}_N, y_N)$$

2. Model (Set of Sigmoid Neurons)

$$h(\boldsymbol{x}; \boldsymbol{\theta}) = \mathrm{sigmoid}(\theta_0 x_0 + \theta_1 x_1 + \cdots + \theta_D x_D)$$

3. Training Loss (Logistic Loss)

$$\mathcal{L}(\boldsymbol{\theta}) = \frac{1}{N} \sum_{(\boldsymbol{x}, y) \in \mathcal{D}} \log\left(1 + e^{-y(\boldsymbol{\theta}^\top \boldsymbol{x})}\right)$$

3. Algorithm (Gradient Descent)

$$\boldsymbol{\theta} \longleftarrow \boldsymbol{\theta} - \frac{\eta_k}{N} \sum_{(\boldsymbol{x}, y) \in \mathcal{D}} \boldsymbol{x}(h(\boldsymbol{x}; \boldsymbol{\theta}) - [\![y = 1]\!])$$

# Regularized Logistic Regression

- When your data has a high-dimensional feature, or your training set is small, you might have the over-fitting problem.

$$\mathcal{L}(\boldsymbol{\theta}) = \frac{1}{N} \sum_{(\boldsymbol{x},y)\in\mathcal{D}} \log\left(1 + e^{-y(\boldsymbol{\theta}^\top \boldsymbol{x})}\right) + \frac{\lambda}{2N} \sum_{j=1}^{D} \theta_j^2$$

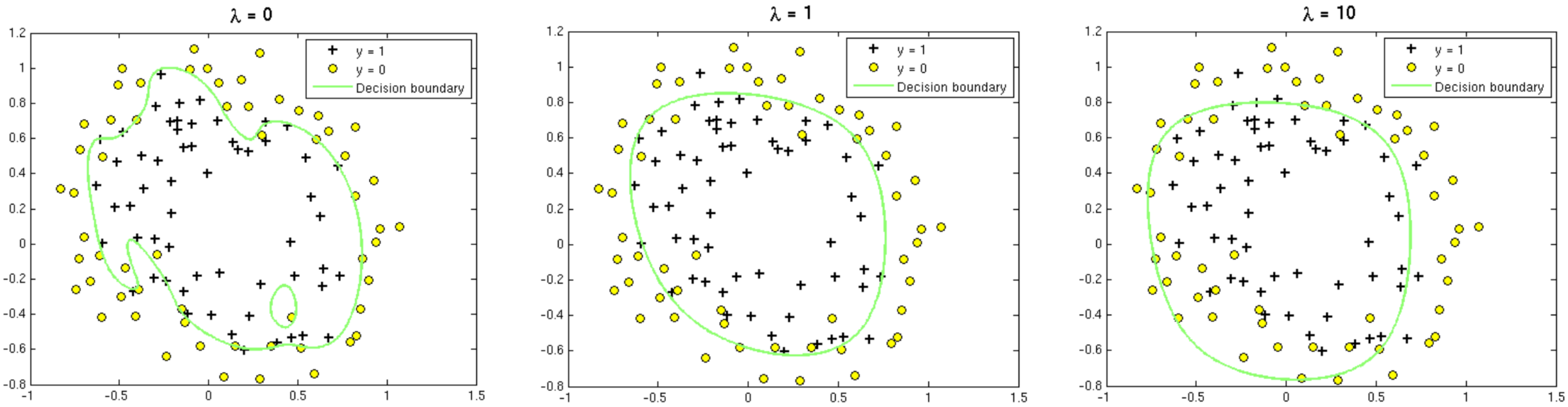
- We are doing regularization on $\theta_1, \theta_2, \dots, \theta_D$
- When using gradient descent, we have

$$\theta_0 \longleftarrow \theta_0 - \frac{\eta_k}{N} \sum_{(\boldsymbol{x},y)\in\mathcal{D}} x^{(0)}\left(h(\boldsymbol{x};\boldsymbol{\theta}) - [\![y=1]\!]\right)$$

$$\theta_j \longleftarrow \theta_j - \frac{\eta_k}{N} \left[\sum_{(\boldsymbol{x},y)\in\mathcal{D}} x^{(j)}\left(h(\boldsymbol{x};\boldsymbol{\theta}) - [\![y=1]\!]\right) + \lambda\,\theta_j\right]$$

$$j = 1,2,\dots,D$$

# Regularized Logistic Regression

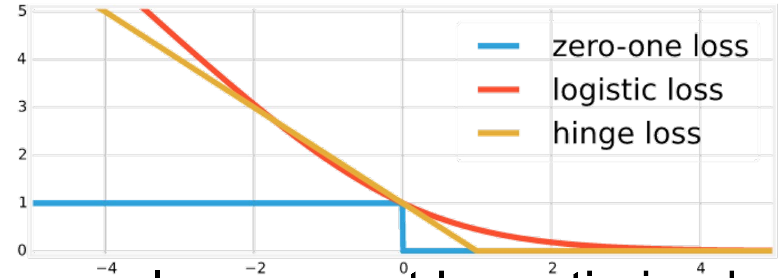



overfitting relatively good underfitting

# Lots of research in classification architecture / loss function design

- Softmax loss (softmax cross-entropy loss)
- AM-Softmax F. Wang, J. Cheng, W. Liu, and H. Liu. Additive margin softmax for face verification. IEEE Signal Processing Letters, 2018.
- NormFace F. Wang, X. Xiang, J. Cheng, and A. L. Yuille. Normface: L2 hypersphere embedding for face verification. ACM MM, 2017
- Triplet loss with hard mining F. Schroff, D. Kalenichenko, and J. Philbin. Facenet: A unified embedding for face recognition and clustering. CVPR 2015
- Lifted structure loss H. Oh Song, Y. Xiang, S. Jegelka, and S. Savarese. Deep metric learning via lifted structured feature embedding. CVPR 2016
- etc
- Circle Loss Sun, Yifan, et al. “Circle loss: A unified perspective of pair similarity optimization.” CVPR 2020.

# Check your understanding




- Among the three loss functions, only the zero-one loss cannot be optimized by gradient descent.
- The logistic loss and hinge loss are generally superior to the zero-one loss.
- It is possible to sum the logistic loss and hinge loss as the final loss function, while keeping the system being able to be optimized by gradient descent.
- It is possible to sum the zero-one loss and hinge loss as the final loss function, while keeping the system optimized by gradient descent.
- "Easy" samples give very small loss values in these three loss functions.
- Comparing with the least square loss, the zero-one loss is more robust to "easy" samples.