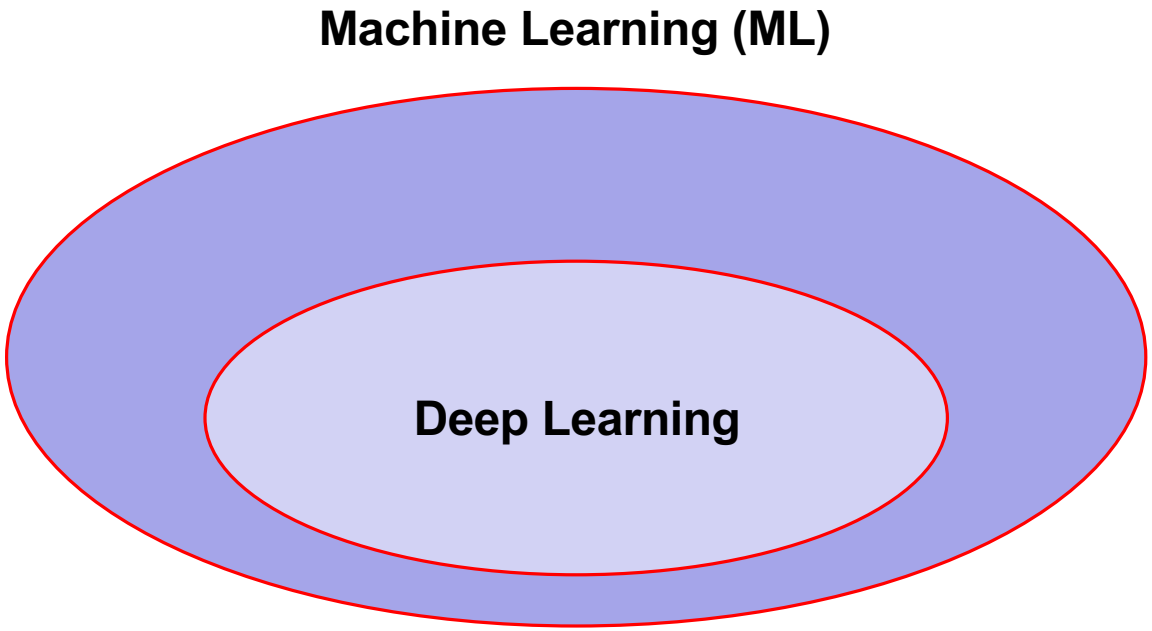
236781: Deep Learning

Linear Models

Or Litany

TECHNION
Israel Institute of Technology

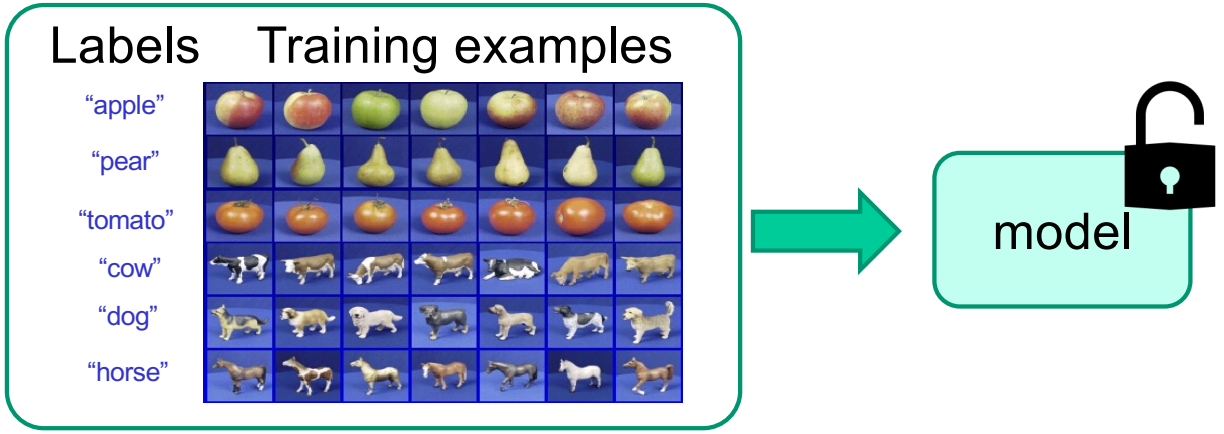
Recap:



Enables computers to reason, process, and generate data through computational models called “neural networks” from large amounts of data.

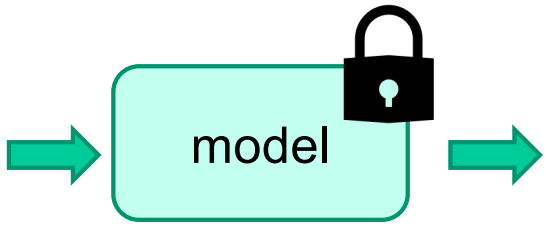
The basic statistical learning framework

Train



Test (generalization)

Test sample



Prediction

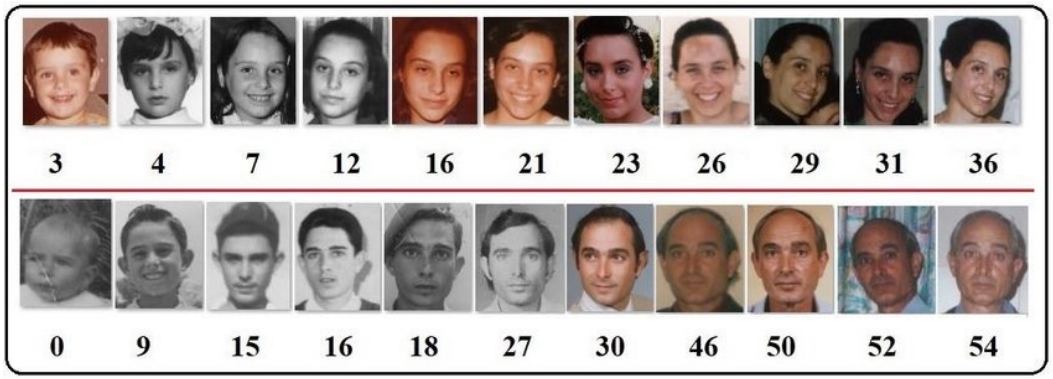
“apple”

Taxonomy of Learning Problems

- **Data modality**
  - 1D signals: audio, text, stock
  - 2D Image: color, xray, depth
  - 3D: point-cloud, mesh, occupancy
  - Spatiotemporal
- **Type of output**
  - Classification
  - Regression
  - Structured prediction
  - Dense prediction
  - Multi-modal prediction
- **Type of supervision**
  - Fully supervised
  - Unsupervised
  - Self-supervised or predictive learning
- **Training regime**
  - Batch offline learning
  - Online/continual learning
  - Active learning
  - Reinforcement learning

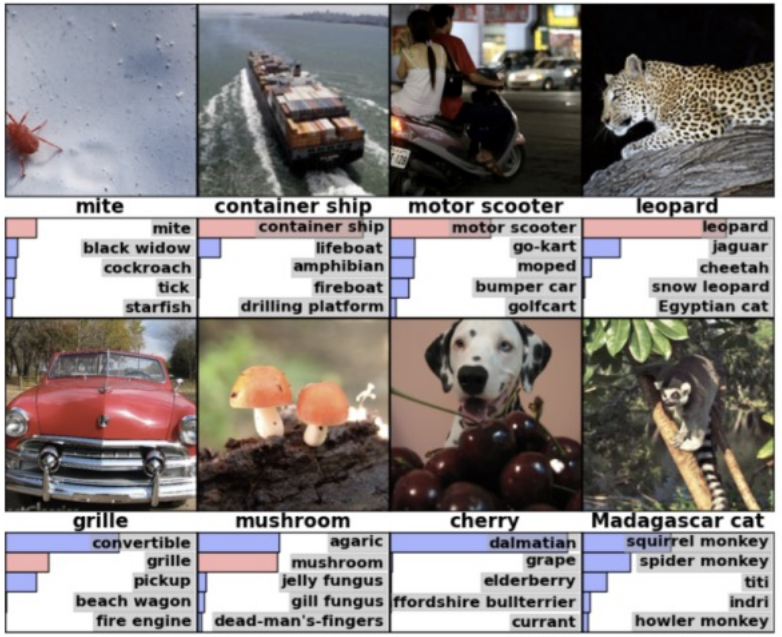
Regression vs Classification

Age estimation



Mousavi et al. (2019)

Image classification



ImageNet Large-Scale Visual Recognition Challenge (ILSVRC)

LINEAR MODELS

Supervised learning: Problem formulation

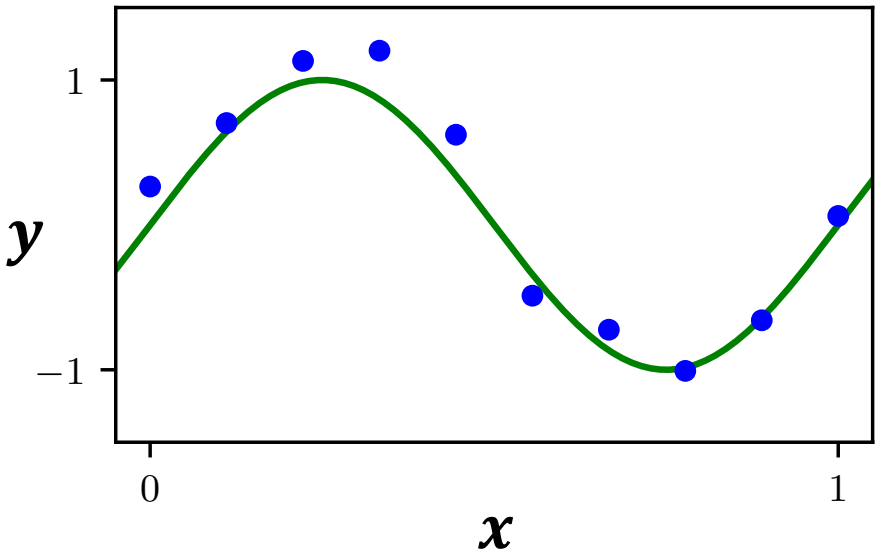
- Given: training data $\{(x_i, y_i), i = 1, \ldots, N\}$
- Find: predictor f
- Goal: make good predictions $\hat{y} = f(x)$ on *test* data

Source: Y. Liang, Bishop

Supervised learning: Problem formulation

- Given: training data $\{(x_i, y_i), i = 1, \ldots, N\}$
- Find: predictor f
- Goal: make good predictions $\hat{y} = f(x)$ on *test* data

Example



$$N=10$$

$$y_i = sin(2\pi x_i) + \epsilon_i$$

(unknown)

Source: Y. Liang, Bishop

Supervised learning: Problem formulation

- Given: training data $\{(x_i, y_i), i = 1, \ldots, N\}$
- Find: predictor $f \in \mathcal{H}$
- Goal: make good predictions $\hat{y} = f(x)$ on *test* data

Hypothesis class

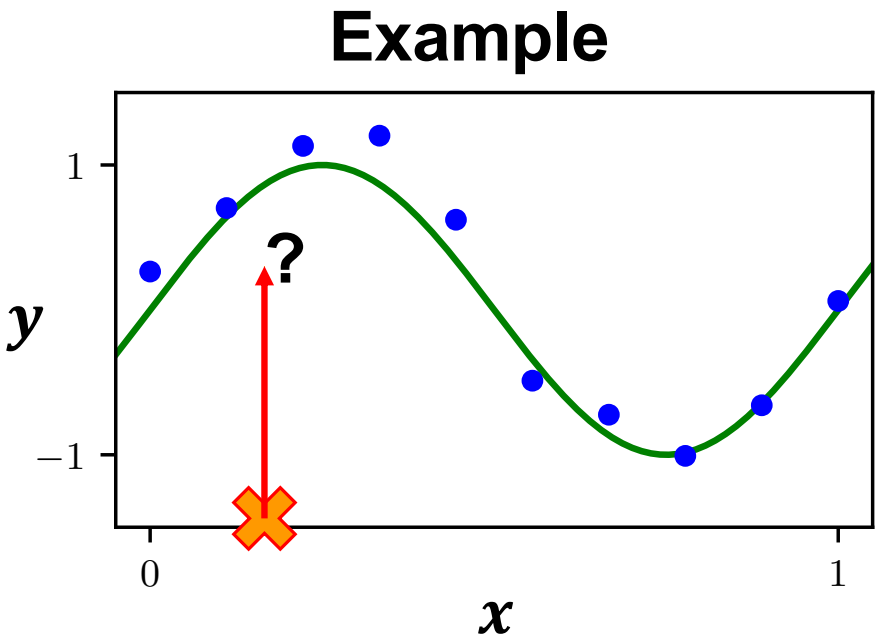
Example

$$f(x, \boldsymbol{w}) = w_0 + w_1 x + w_2 x^2 + \cdots + w_M x^M = \sum_{j=0}^{M} w_j x^j$$

Source: Y. Liang

Supervised learning: Problem formulation

- Given: training data $\{(x_i, y_i), i = 1, \ldots, N\}$
- Find: predictor $f \in \mathcal{H}$
- Goal: make good predictions $\hat{y} = f(x)$ on *test* data

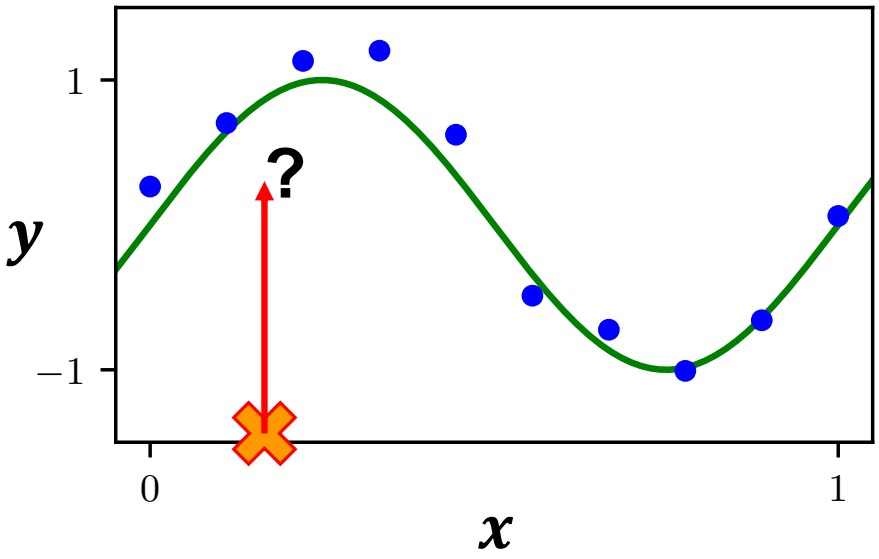


Connection between training and test data?

Source: Y. Liang

Supervised learning: Problem formulation

- Given: training data $\{(x_i, y_i), i = 1, \ldots, N\}$ i.i.d. from distribution D
- Find: predictor $f \in \mathcal{H}$
- Goal: make good predictions $\hat{y} = f(x)$ on *test* data i.i.d. from distribution D



Source: Y. Liang

Supervised learning: Problem formulation

- Given: training data $\{(x_i, y_i), i = 1, \ldots, N\}$ i.i.d. from distribution D
- Find: predictor $f \in \mathcal{H}$
- Goal: make good predictions $\hat{y} = f(x)$ on *test* data i.i.d. from distribution D

What kind of performance measure?

Source: Y. Liang

Supervised learning: Problem formulation

- Given: training data $\{(x_i, y_i), i = 1, \ldots, N\}$ i.i.d. from distribution D
- Find: predictor $f \in \mathcal{H}$
- S.t. the *expected loss* is small:

$$L(f) = \mathbb{E}_{(x,y)\backsim D}[l(f, x, y)]$$

Various loss functions

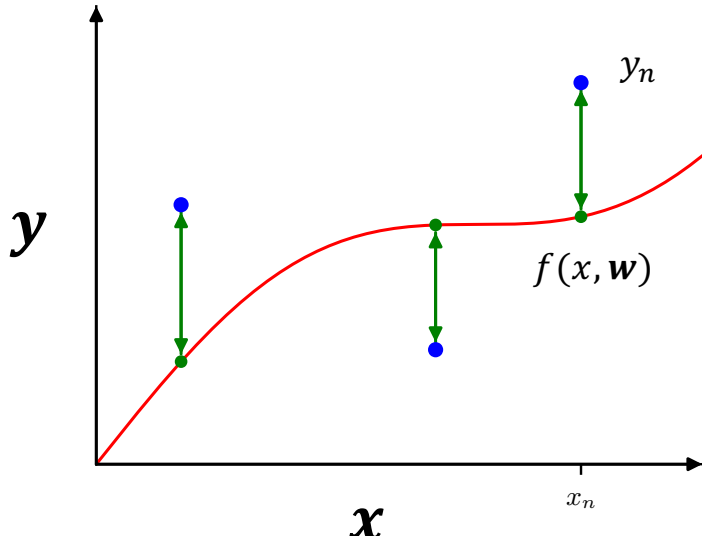
Source: Y. Liang

Supervised learning: Problem formulation

- Given: training data $\{(x_i, y_i), i = 1, \ldots, N\}$ i.i.d. from distribution D
- Find: predictor $f \in \mathcal{H}$
- S.t. the *expected loss* is small:

$$L(f) = \mathbb{E}_{(x,y)\sim D}[l(f, x, y)]$$

- Example: l_2 loss: $l(f, x, y) = [f(x) - y]^2$ and $L(f) = \mathbb{E}[\,[f(x) - y]^2]$



Source: Y. Liang

Supervised learning: Problem formulation

- Given: training data $\{(x_i, y_i), i = 1, \ldots, N\}$ i.i.d. from distribution D
- Find: predictor $f \in \mathcal{H}$
- S.t. the *expected loss* is small:

$$L(f) = \mathbb{E}_{(x,y)\sim D}[l(f, x, y)]$$

Can't optimize this directly

Source: Y. Liang

Supervised learning: Problem formulation

- Given: training data $\{(x_i, y_i), i = 1, \ldots, N\}$ i.i.d. from distribution D
- Find: predictor $f \in \mathcal{H}$ that minimizes

$$\hat{L}(f) = \frac{1}{n} \sum_{i=1}^{N} l(f, x_i, y_i)$$

Empirical loss

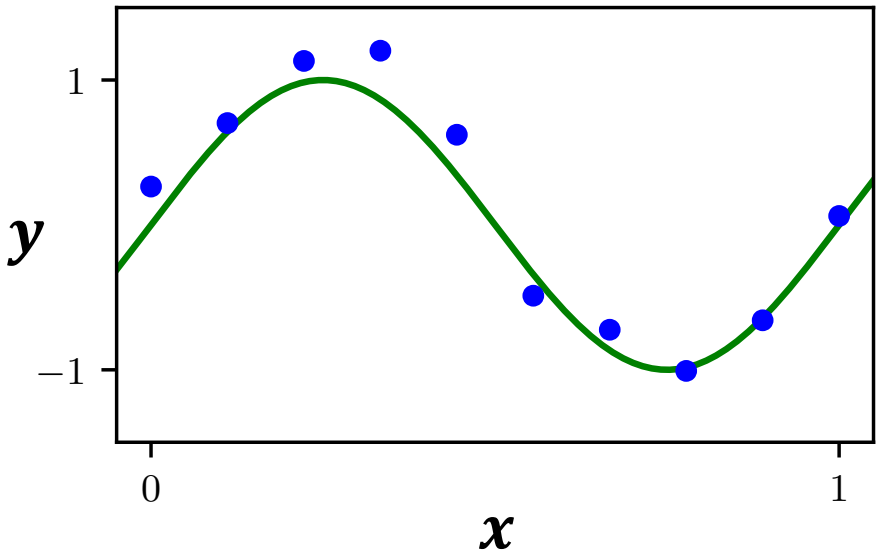
Source: Y. Liang

Learning Recipe

1. **Collect *training data* and labels**
2. **Specify model:** select *hypothesis class* and *loss function*
3. **Train model:** find the function in the hypothesis class that minimizes the *empirical loss* on the training data

Example: Linear regression 1D

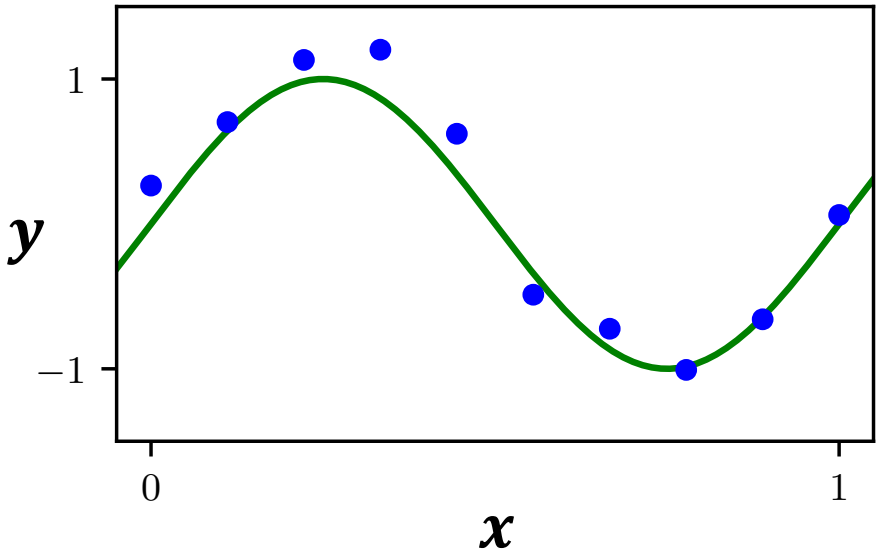
$$f(x, \boldsymbol{w}) = w_0 + w_1 x + w_2 x^2 + \cdots + w_M x^M = \sum_{j=0}^{M} w_j x^j$$



$$\hat{L}(f_w) = \frac{1}{n} \sum_{i=1}^{n} (f(x_i, \boldsymbol{w}) - y_i)^2$$

Linear regression 1D example

$$f(x, \boldsymbol{w}) = w_0 + w_1 x + w_2 x^2 + \cdots + w_M x^M = \sum_{j=0}^{M} w_j x^j$$

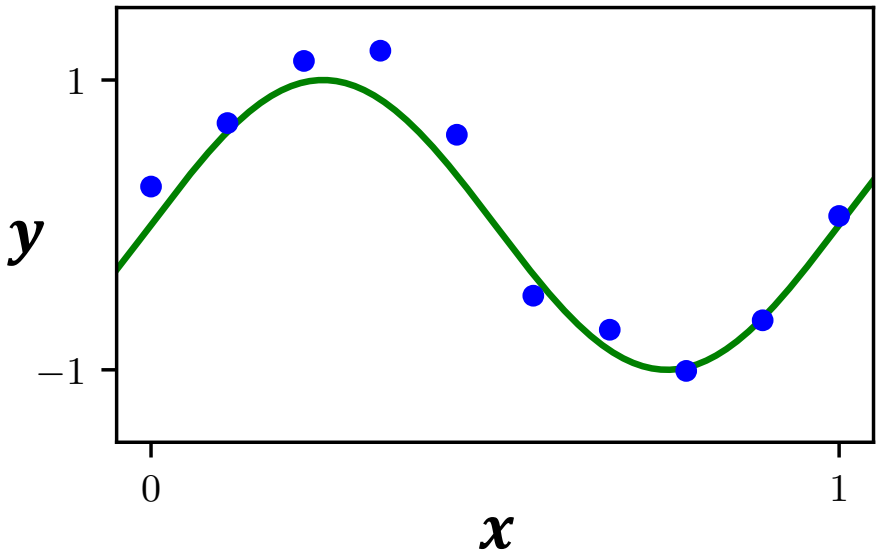


$$\hat{L}(f_w) = \frac{1}{n} \sum_{i=1}^{n} (f(x_i, \boldsymbol{w}) - y_i)^2$$

Closed form solution: *f(x,w*)*

Linear regression 1D example

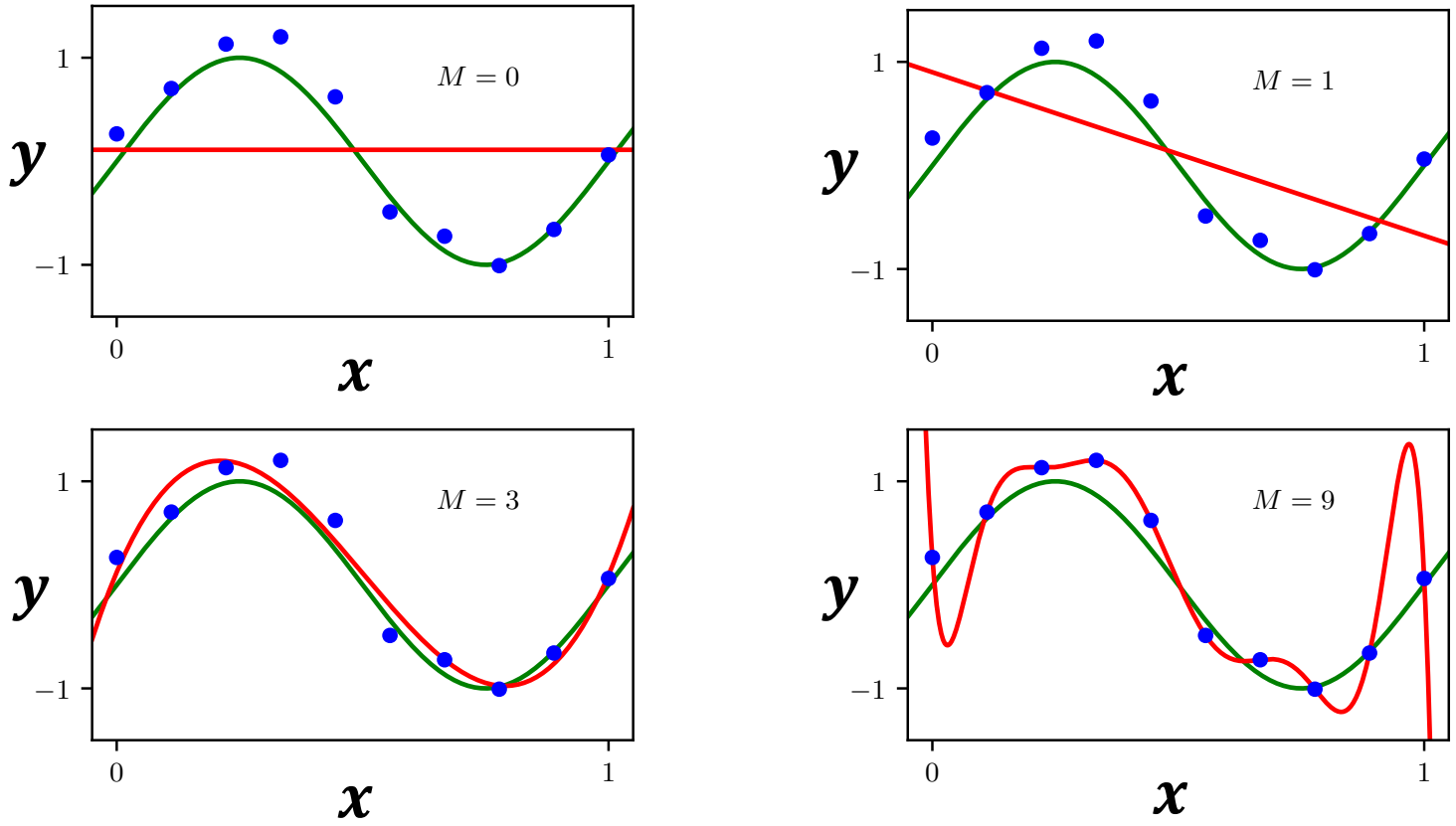
$$f(x, \boldsymbol{w}) = w_0 + w_1 x + w_2 x^2 + \cdots + w_M x^M = \sum_{j=0}^{M} w_j x^j$$



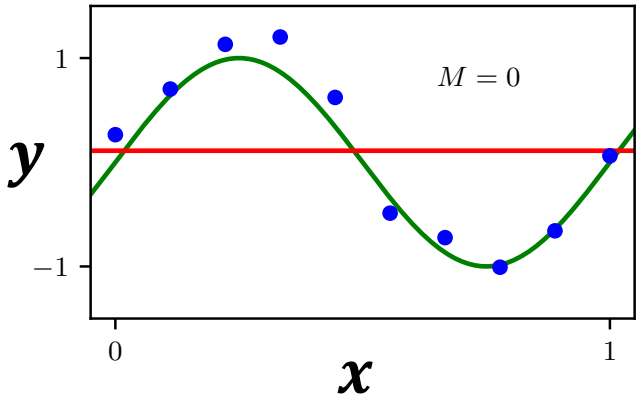
$$\hat{L}(f_w) = \frac{1}{n} \sum_{i=} f(x_i, \boldsymbol{w}) - y_i)^2$$

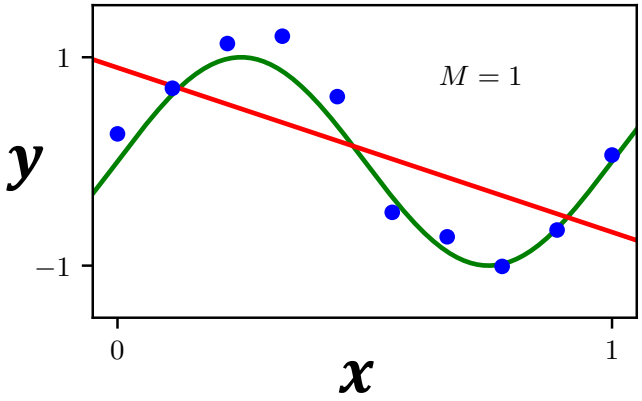
Model complexity

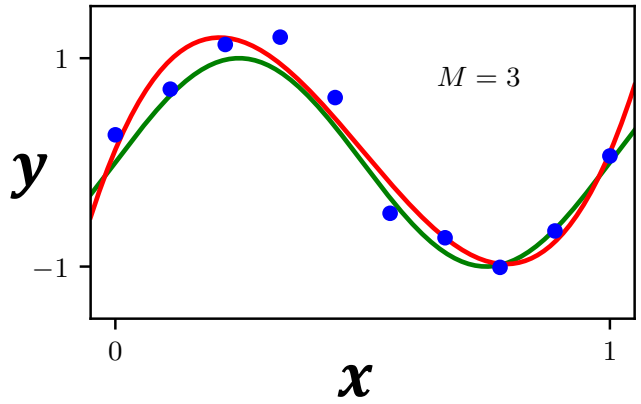
Linear regression 1D example: model complexity

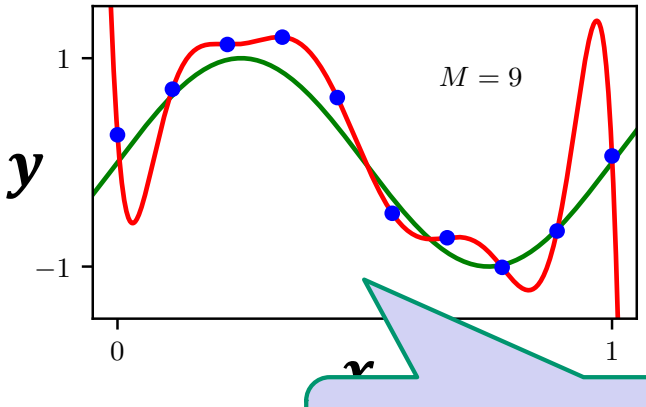


Linear regression 1D example: model complexity







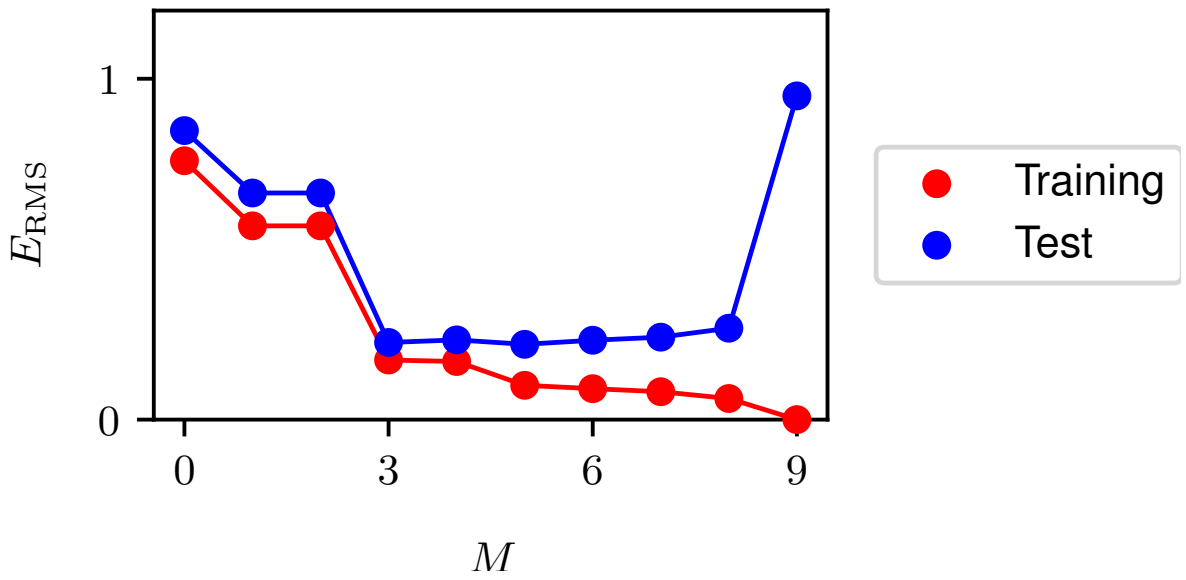


overfitting

Linear regression 1D example: model complexity

$$RMSE = \sqrt{\frac{1}{N}\sum_{i=1}^{N}(f(x_i, \boldsymbol{w}) - y_i)^2}$$

Overfitting:



Linear regression 1D example: model complexity

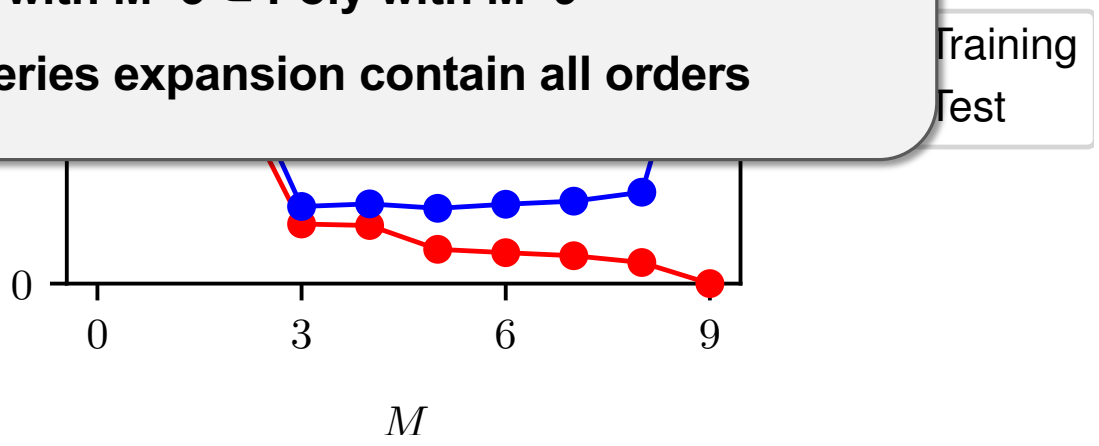
$$RMSE = \sqrt{\frac{1}{N}\sum_{i=1}^{N}(f(x_i, \boldsymbol{w}) - y_i)^2}$$

Paradox?

(1) Poly with M=3 ⊂ Poly with M=9

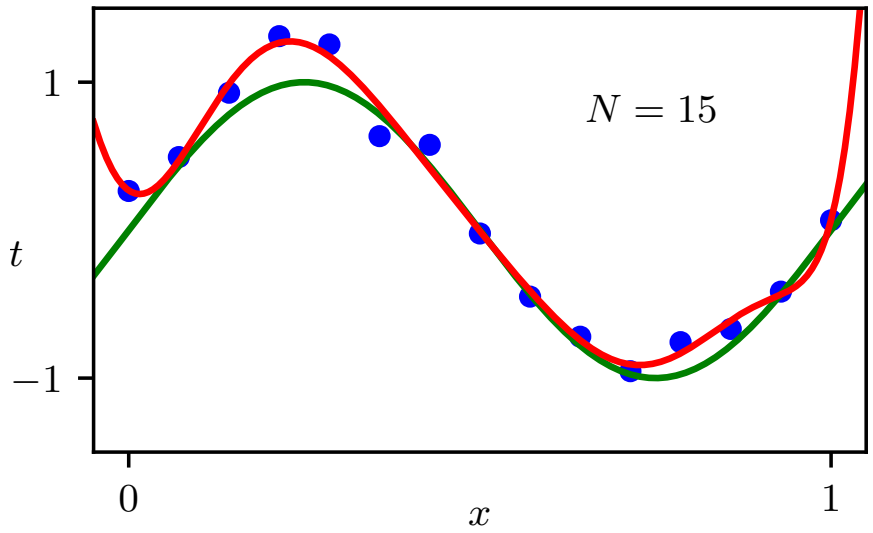
(2) $\sin\theta$ **power series expansion contain all orders**

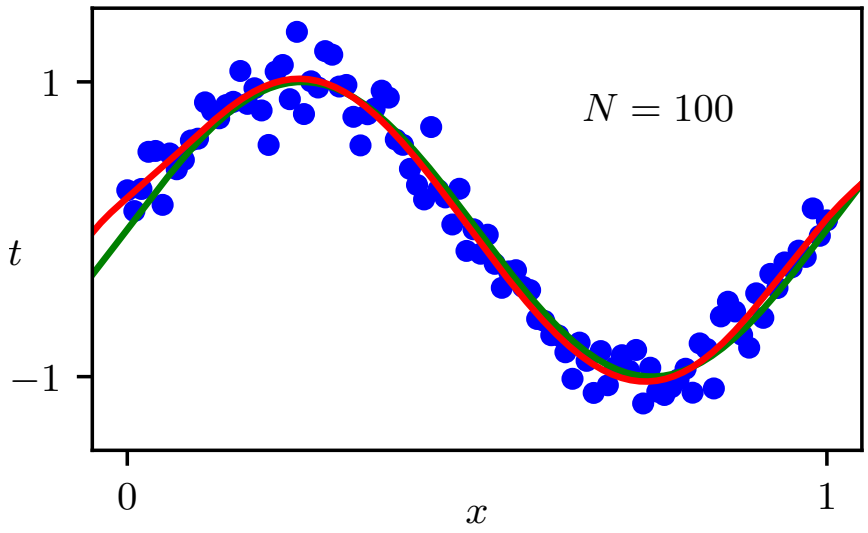
Ov



Linear regression 1D example: model complexity

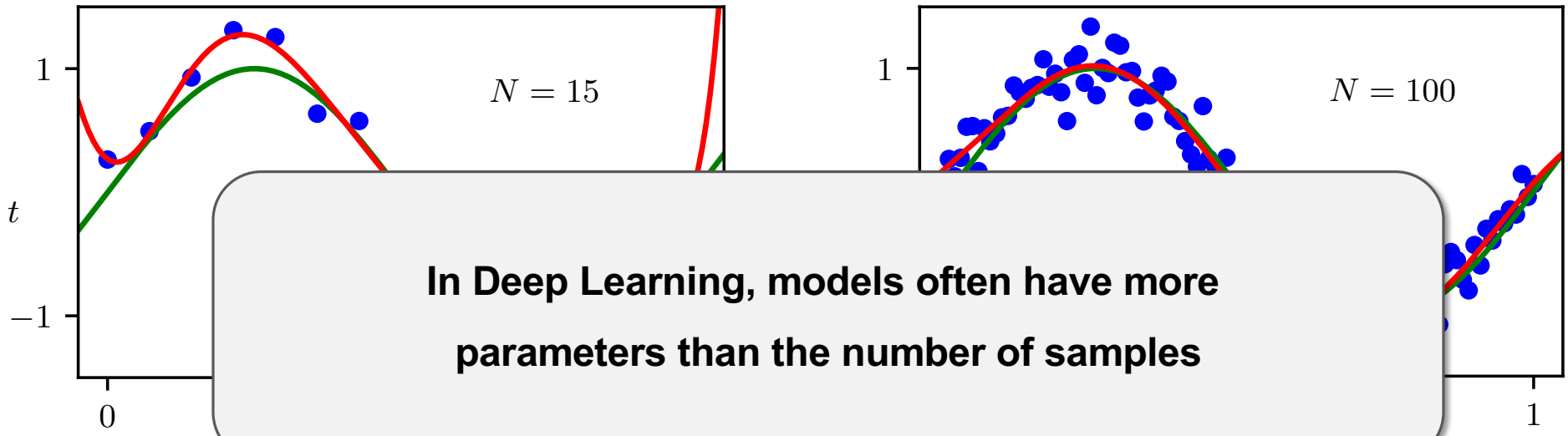
More data / Regularization





Linear regression 1D example: model complexity

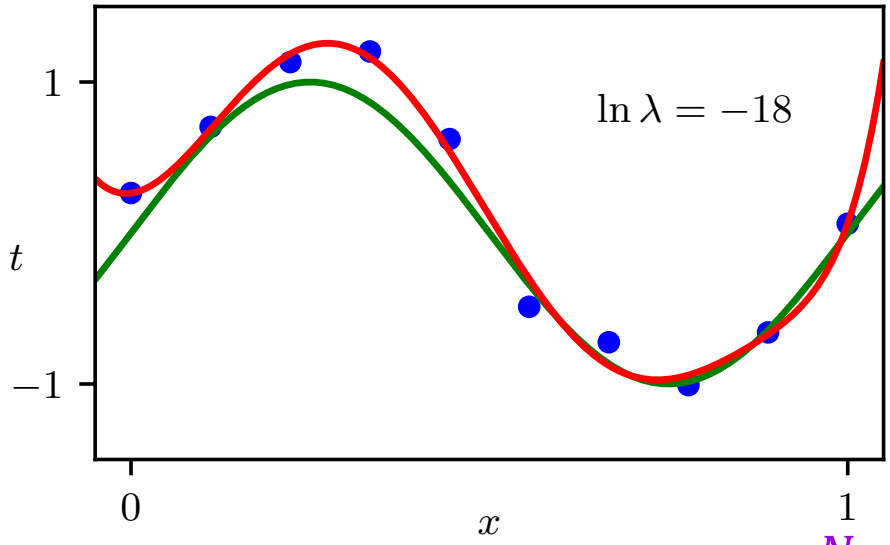
More data / Regularization

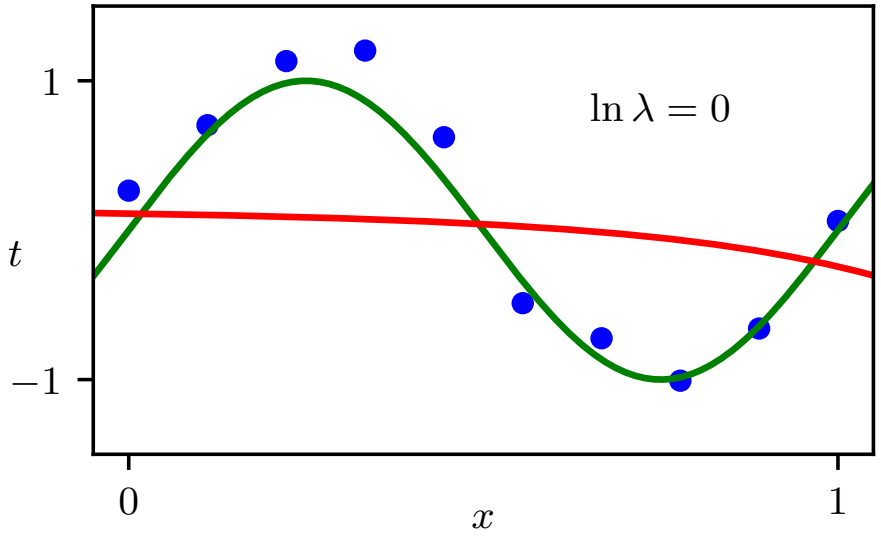


In Deep Learning, models often have more parameters than the number of samples

Linear regression 1D example: model complexity

More data / Regularization

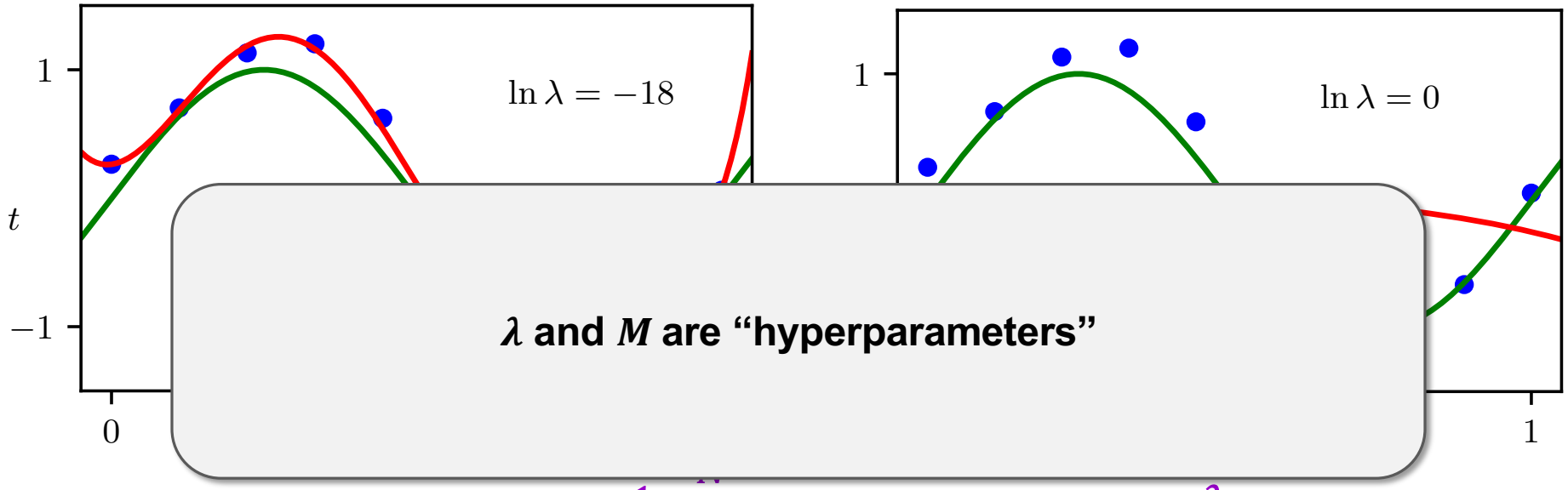




$$\hat{L}(f_w) = \frac{1}{N}\sum_{i=1}^{N}(f(x_i, \boldsymbol{w}) - y_i)^2 + \frac{\lambda}{2}\|\boldsymbol{w}\|^2$$

Linear regression 1D example: model complexity

More data / Regularization



***λ* and *M* are "hyperparameters"**

$$\hat{L}(f_w) = \frac{1}{N}\sum_{i=1}^{N}(f(x_i, \boldsymbol{w}) - y_i)^2 + \frac{\lambda}{2}\|\boldsymbol{w}\|^2$$

Linear regression: Optimization

Find $f_{\boldsymbol{w}}(\boldsymbol{x}) = \boldsymbol{w}^T \boldsymbol{x}$ that minimizes l_2 *loss* or *mean squared error*

$$\hat{L}(f_w) = \frac{1}{n}\sum_{i=1}^{n}(\boldsymbol{w}^T \boldsymbol{x_i} - y_i)^2$$

Example:

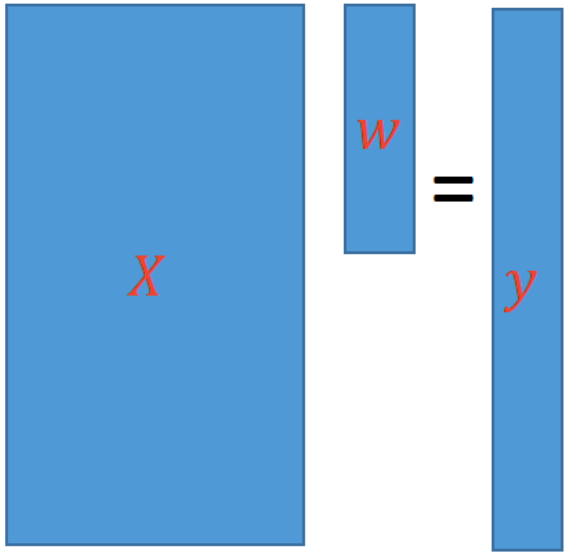
$$\boldsymbol{x_i} \rightarrow \left[x_i, x_i^2, x_i^3, \ldots, x_i^M\right]$$

$$\sum_{j=0}^{M} w_j x_i^j = \boldsymbol{w}^T \boldsymbol{x_i}$$

Linear regression: Optimization

- Let X be a matrix whose ith row is x_i^T, $\boldsymbol{y}$ be the vector $(y_1, \cdots, y_n)^T$

$$\hat{L}(f_w) = \frac{1}{n}\sum_{i=1}^{n}(w^T x_i - y_i)^2 = \frac{1}{n}\|X\boldsymbol{w} - \boldsymbol{y}\|_2^2$$



Source: Y. Liang

Linear regression: Optimization

- Let X be a matrix whose ith row is x_i^T, $\boldsymbol{y}$ be the vector $(y_1, \cdots, y_n)^T$

$$\hat{L}(f_w) = \frac{1}{n}\sum_{i=1}^{n}(w^T x_i - y_i)^2 = \frac{1}{n}\|X\boldsymbol{w} - \boldsymbol{y}\|_2^2$$

- Find the *gradient* w.r.t. w:

$\nabla_w \|Xw - y\|_2^2$

Source: Y. Liang

Linear regression: Optimization

- Let X be a matrix whose ith row is x_i^T, Y be the vector $(y_1, \cdots, y_n)^T$

$$\hat{L}(f_w) = \frac{1}{n}\sum_{i=1}^{n}(w^T x_i - y_i)^2 = \frac{1}{n}\|Xw - y\|_2^2$$

- Find the *gradient* w.r.t. w:

$$\begin{aligned}\nabla_w \|Xw - y\|_2^2 &= \nabla_w [(Xw - y)^T (Xw - y)] \\ &= \nabla_w [w^T X^T Xw - 2w^T X^T y + y^T y] \\ &= 2X^T Xw - 2X^T y\end{aligned}$$

- Set gradient to zero to get the minimizer:

$$X^T Xw = X^T y$$
$$w = (X^T X)^{-1} X^T y$$

Source: Y. Liang

Linear regression: Optimization

- Let X be a matrix whose ith row is x_i^T, Y be the vector $(y_1, \cdots, y_n)^T$

$$\hat{L}(f_w) = \frac{1}{n}\sum_{i=1}^{n}(w^T x_i - y_i)^2 = \frac{1}{n}\|Xw - y\|_2^2$$

- Find the *gradient* w.r.t. w:

$$\begin{aligned}\nabla_w \|Xw - y\|_2^2 &= \nabla_w [(Xw - y)^T (Xw - y)] \\ &= \nabla_w [w^T X^T Xw - 2w^T X^T y + y^T y] \\ &= 2X^T Xw - 2X^T y\end{aligned}$$

- Set gradient to zero to get the minimizer:

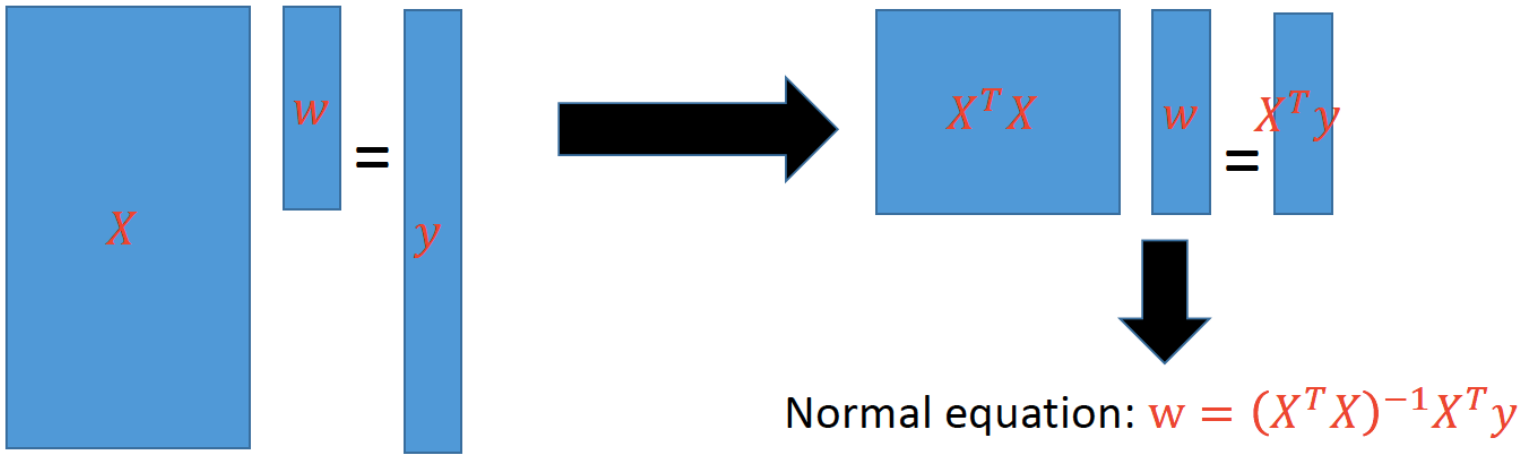
$$X^T Xw = X^T y$$
$$w = (X^T X)^{-1} X^T y$$

Pseudo inverse

Source: Y. Liang

Linear regression: Optimization

- Linear algebra view
  - If X is invertible, simply solve $Xw = y$ and get $w = X^{-1}y$
  - But typically X is a "tall" matrix so you need to find the *least squares solution* to an over-constrained system



Source: Y. Liang

Linear regression as maximum likelihood estimation

- Given: i.i.d. training data $\{(x_i, y_i), i = 1, \ldots, n\}$
- Let $P_w(y|x)$ be a density function parameterized by w
- Maximum (conditional) likelihood estimate:

$$w_{ML} = \text{argmax}_w \prod_i P_w(y_i|x_i)$$

$$= \text{argmin}_w - \sum_i \log P_w(y_i|x_i)$$

Linear regression as maximum likelihood estimation

$$w_{ML} = \text{argmin}_w - \sum_i \log P_w(y_i|x_i)$$

- Assume $P_w(y|x) = \text{Normal}(y; f_w(x), \sigma^2)$

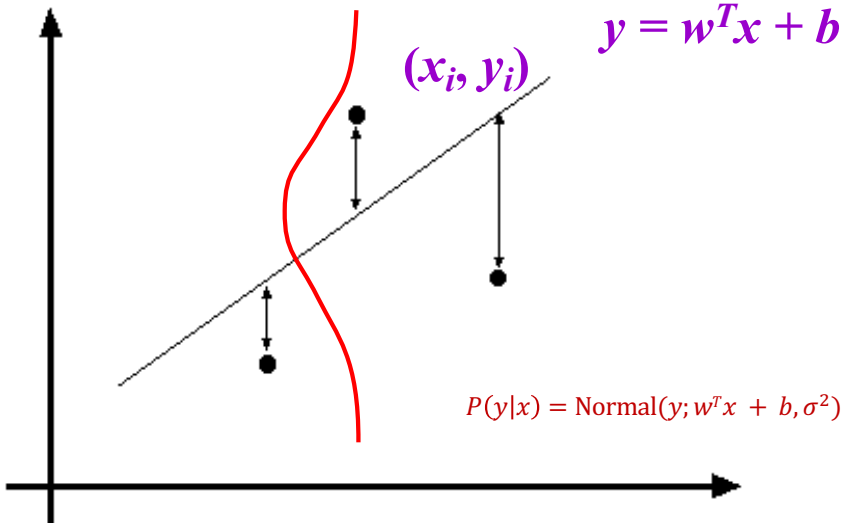
$$\log P_w(y|x) = \log\left[\frac{1}{\sqrt{2\pi\sigma^2}}\exp\left[-\frac{\left(y - f_w(x)\right)^2}{2\sigma^2}\right]\right]$$

$$= -\frac{1}{2\sigma^2}\left(y - f_w(x)\right)^2 - \log\sigma - \frac{1}{2}\log(2\pi)$$

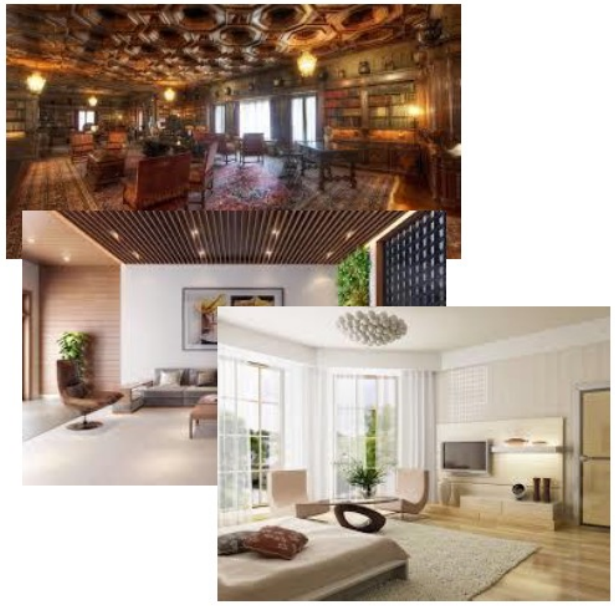
$$w_{ML} = \text{argmin}_w \sum_i \left(y_i - f_w(x_i)\right)^2$$

Linear regression as maximum likelihood estimation

- Interpretation of l_2 loss: *negative log likelihood* assuming y is normally distributed with mean $f_w(x) = w^T x + b$



Application: Image classification

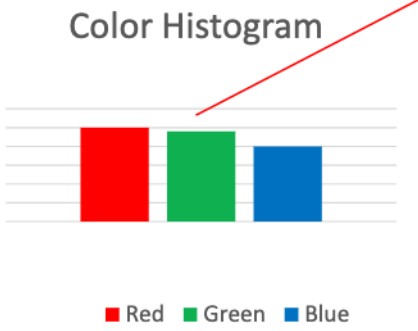


Indoor

outdoor

Example: image classification

Extract features



Feature vector: x_i

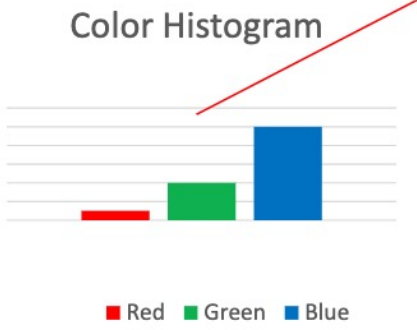
Indoor

0

Label: y_i

Example: image classification

Extract features



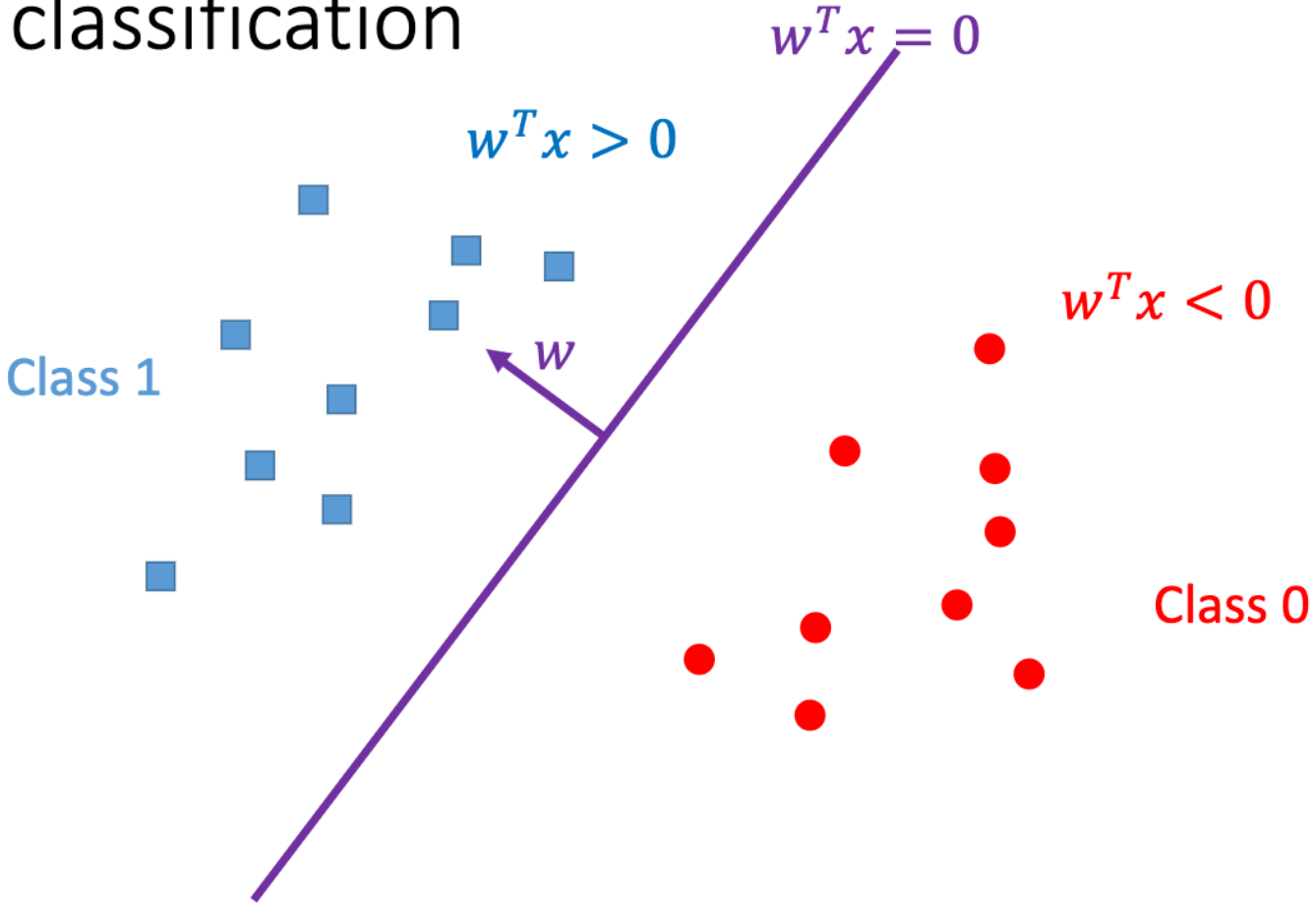
Feature vector: x_j

outdoor

1

Label: y_j

Linear classification



Training linear classifiers

- Given: i.i.d. training data $\{(x_i, y_i), i = 1, \dots, n\}$, $y_i \in \{0,1\}$
- Hypothesis class: $f_w(x) = w^T x$
- Loss: how about minimizing the number of mistakes on the training data?

$$\hat{L}(f_w) = \frac{1}{n}\sum_{i=1}^{n} \mathbb{I}[\text{step}(w^T x_i) \neq y_i]$$

- Difficult to optimize directly, so people resort to *surrogate loss functions*

Source: Y. Liang

Linear regression

- Find $f_w(x) = w^T x$ that minimizes l_2 *loss* or *mean squared error*

$$\hat{L}(f_w) = \frac{1}{n}\sum_{i=1}^{n}(w^T x_i - y_i)^2$$

- Ignores the fact that $y \in \{0,1\}$ but is easy to optimize

Source: Y. Liang

Problem with linear regression

- In practice, very sensitive to outliers

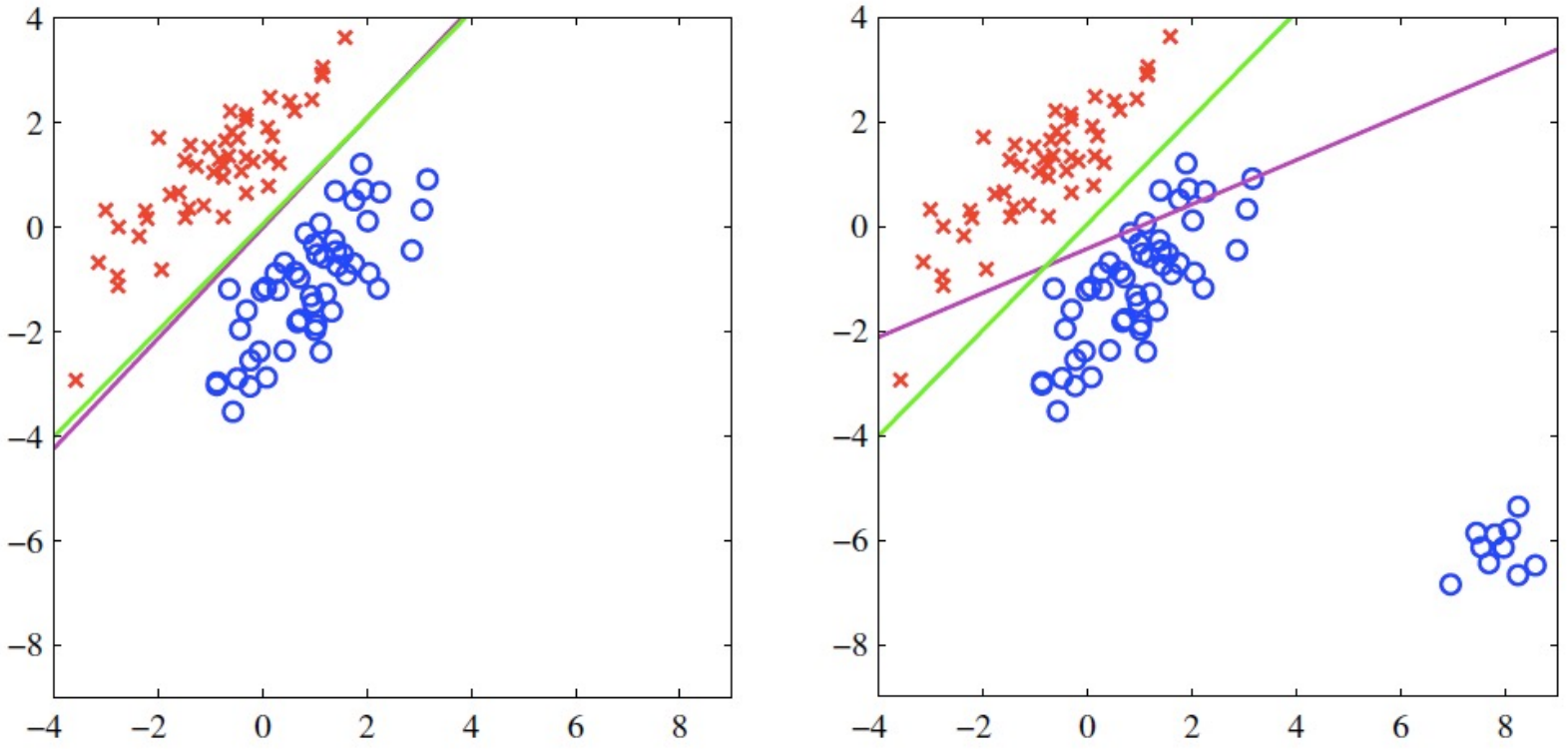
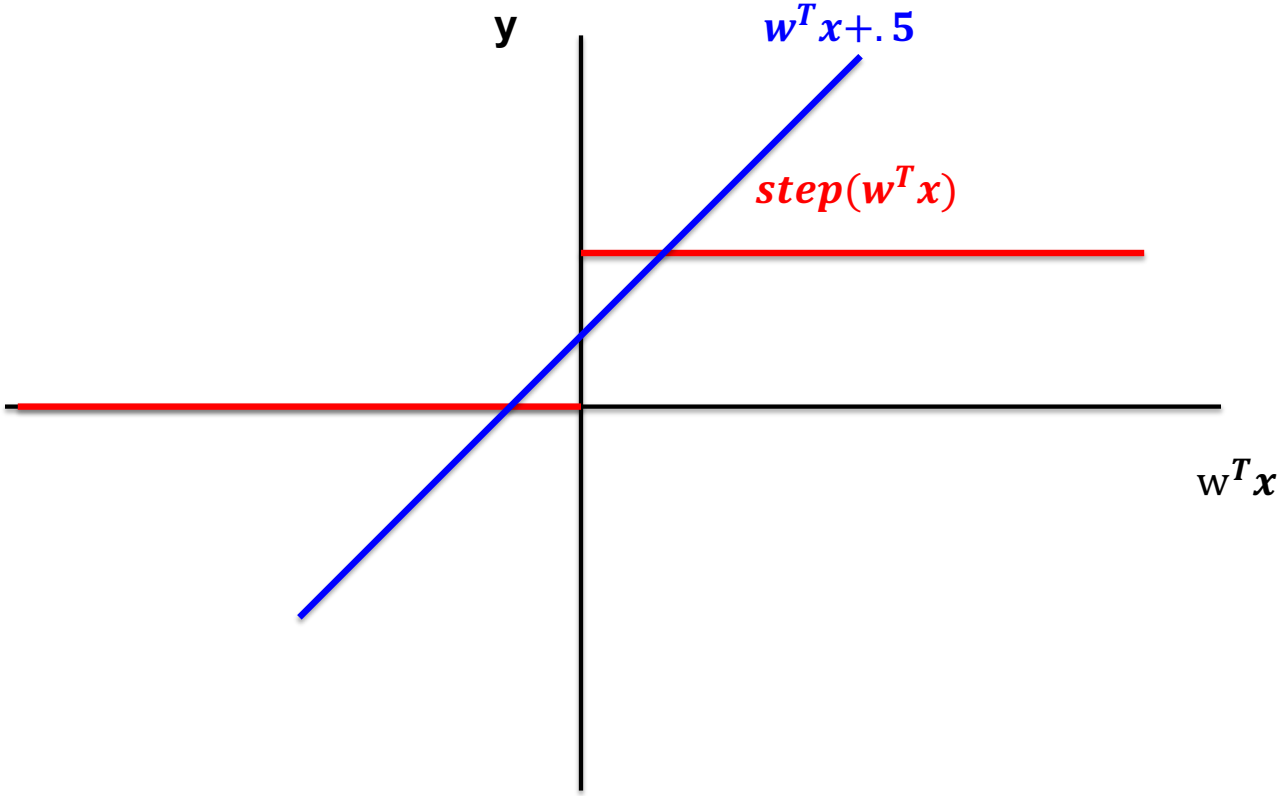


Figure from *Pattern Recognition and Machine Learning*, Bishop

Compare optimization criteria

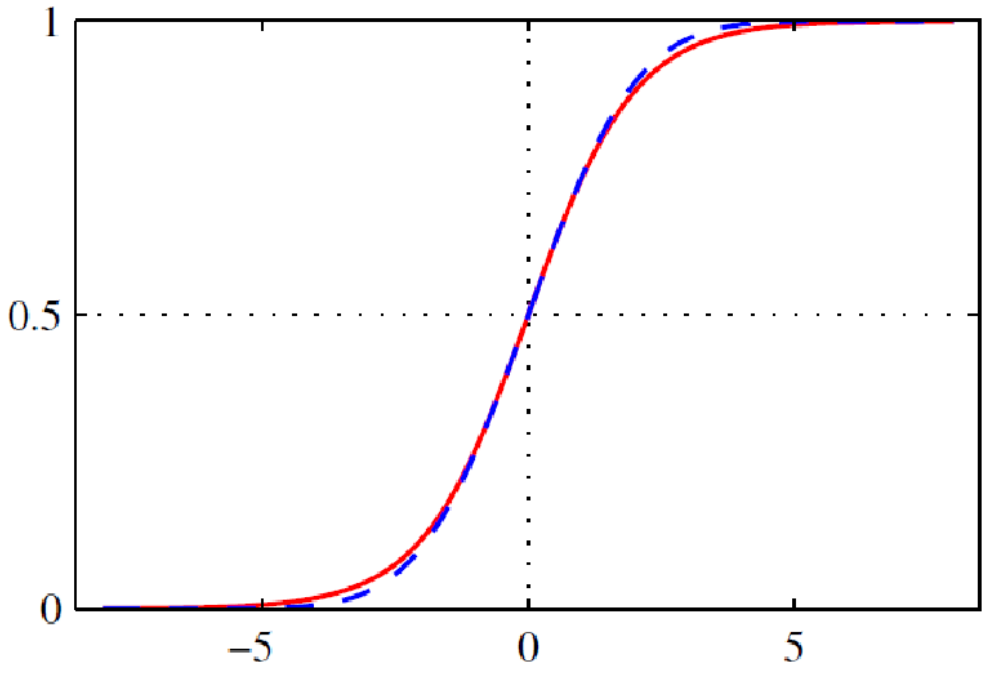


Introducing: logistic sigmoid

- Between the two

  - Prediction bounded in $[0,1]$
  - Smooth

  - Sigmoid: $\sigma(a) = \frac{1}{1+\exp(-a)}$

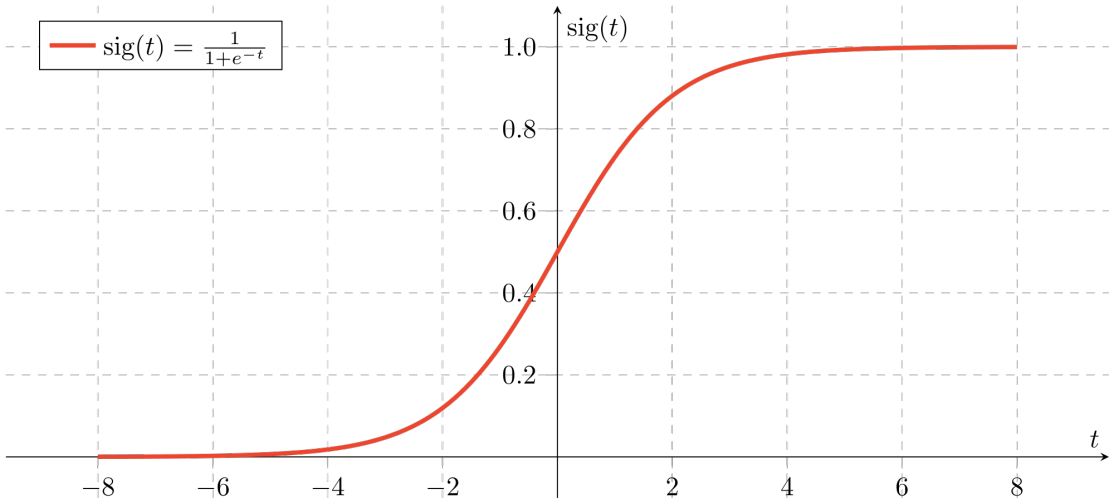


Logistic regression

- Squash the output of the linear function:

$$Sigmoid(w^T x) = \sigma(w^T x) = \frac{1}{1 + \exp(-w^T x)}$$

- Find w that minimizes $\hat{L}(f_w) = \frac{1}{n}\sum_{i=1}^{n}(\sigma(w^T x_i) - y_i)^2$



Logistic regression

- Probabilistic interpretation:

$$P_w(y=1|x) = \frac{p(x|y=1)p(y=1)}{p(x|y=1)p(y=1) + p(x|y=0)p(y=0)}$$
$$= \frac{1}{1+exp(-a)} = \sigma(a)$$

Where,

$$\boldsymbol{a = ln\frac{p(x|y=1)p(y=1)}{p(x|y=0)p(y=0)}}$$

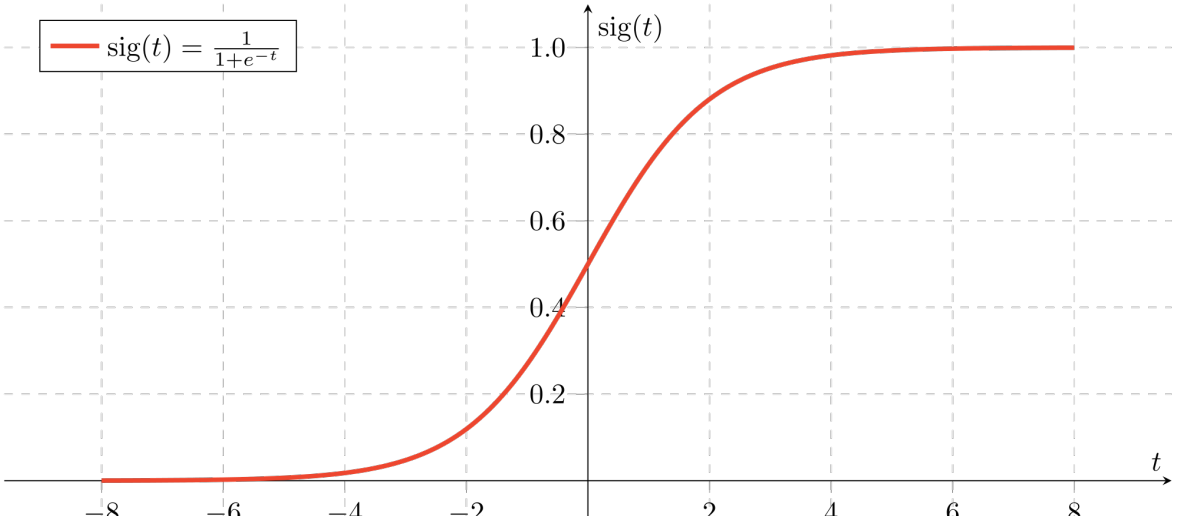
Log odds

Logistic regression

- Probabilistic interpretation:

$$P_w(y = 1|x) = \sigma(w^T x) = \frac{1}{1 + \exp(-w^T x)}$$

$$P_w(y = 0|x) = 1 - \sigma(w^T x)$$



Logistic loss

- Given: $\{(x_i, y_i), i = 1, \dots, n\}$, $y_i \in \{0,1\}$
- Maximum likelihood estimate: find w that minimizes

$$\hat{L}(w) = -\frac{1}{n}\sum_{i=1}^{n} \log P_w(y_i|x_i)$$

$$\hat{L}(w) = -\frac{1}{n}\sum_{y_i=1} \log \sigma(w^T x_i) - \frac{1}{n}\sum_{y_i=0} \log[1 - \sigma(w^T x_i)]$$

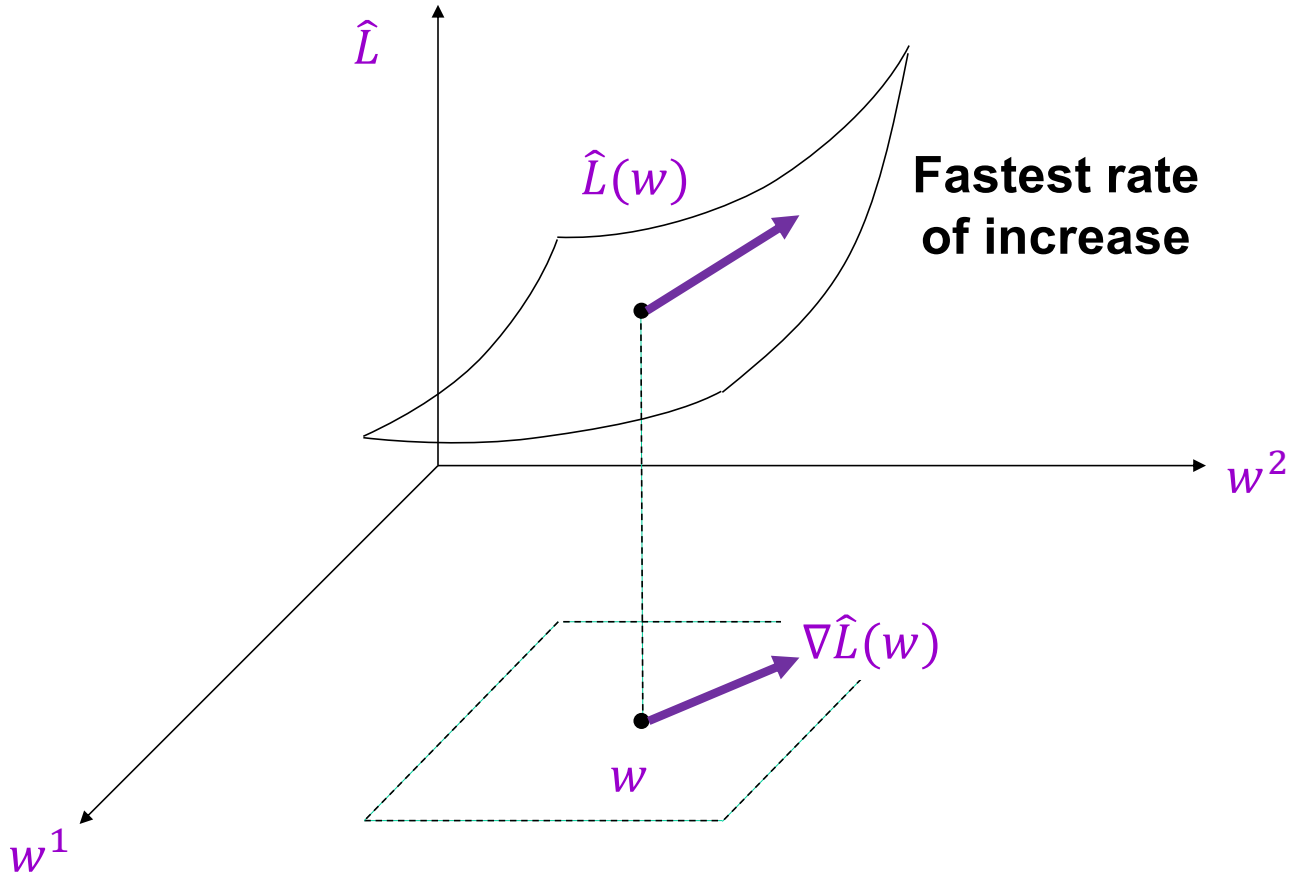
Logistic regression

No closed form solution – need to use Gradient Decent

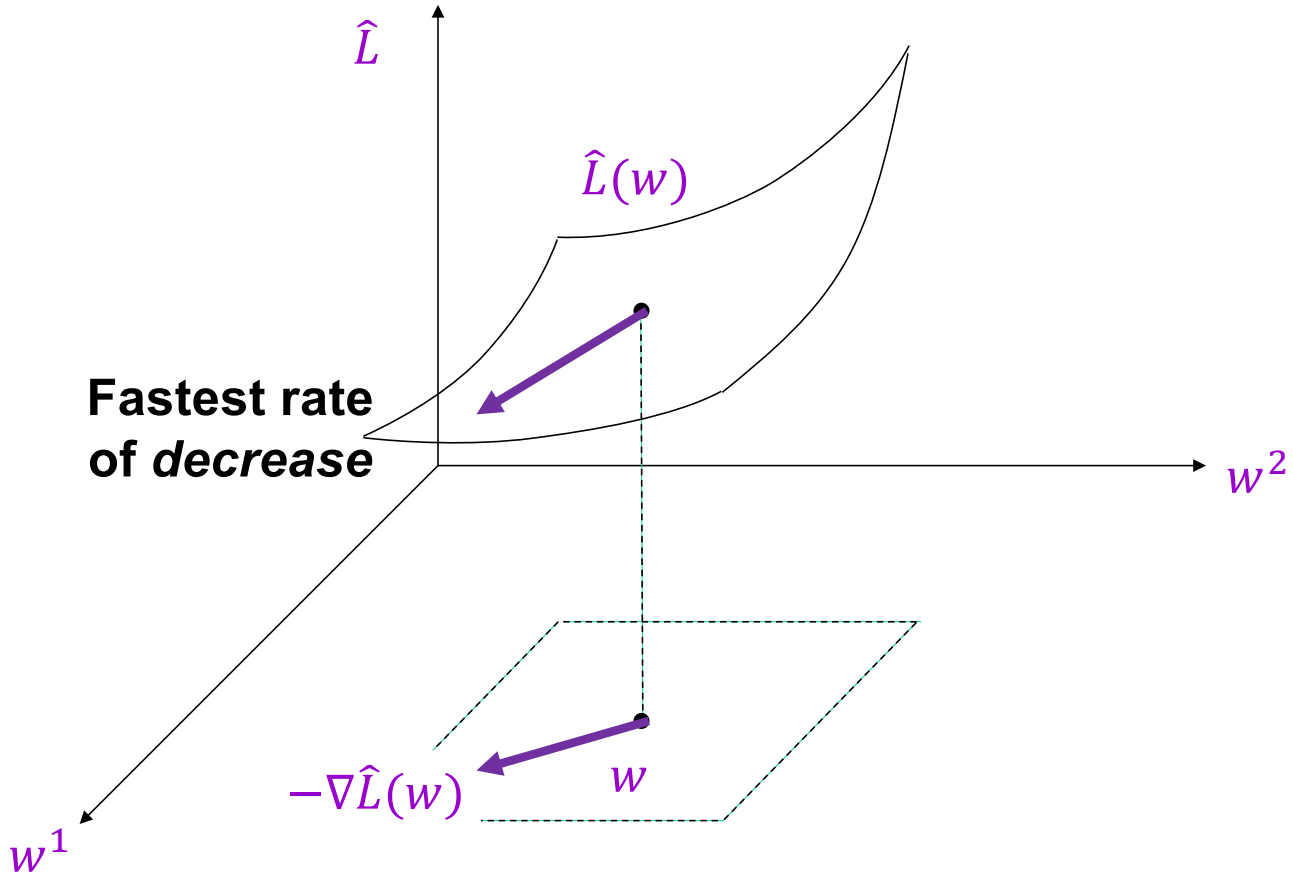
Gradient descent

- Goal: find w to minimize loss $\hat{L}(w)$
- Start with some initial estimate of w
- Repeat until convergence:
  - Find $\nabla\hat{L}(w)$, the *gradient* of the loss w.r.t. w
  - Take a small step in the *opposite* direction: $w \leftarrow w - \eta\,\nabla\hat{L}(w)$

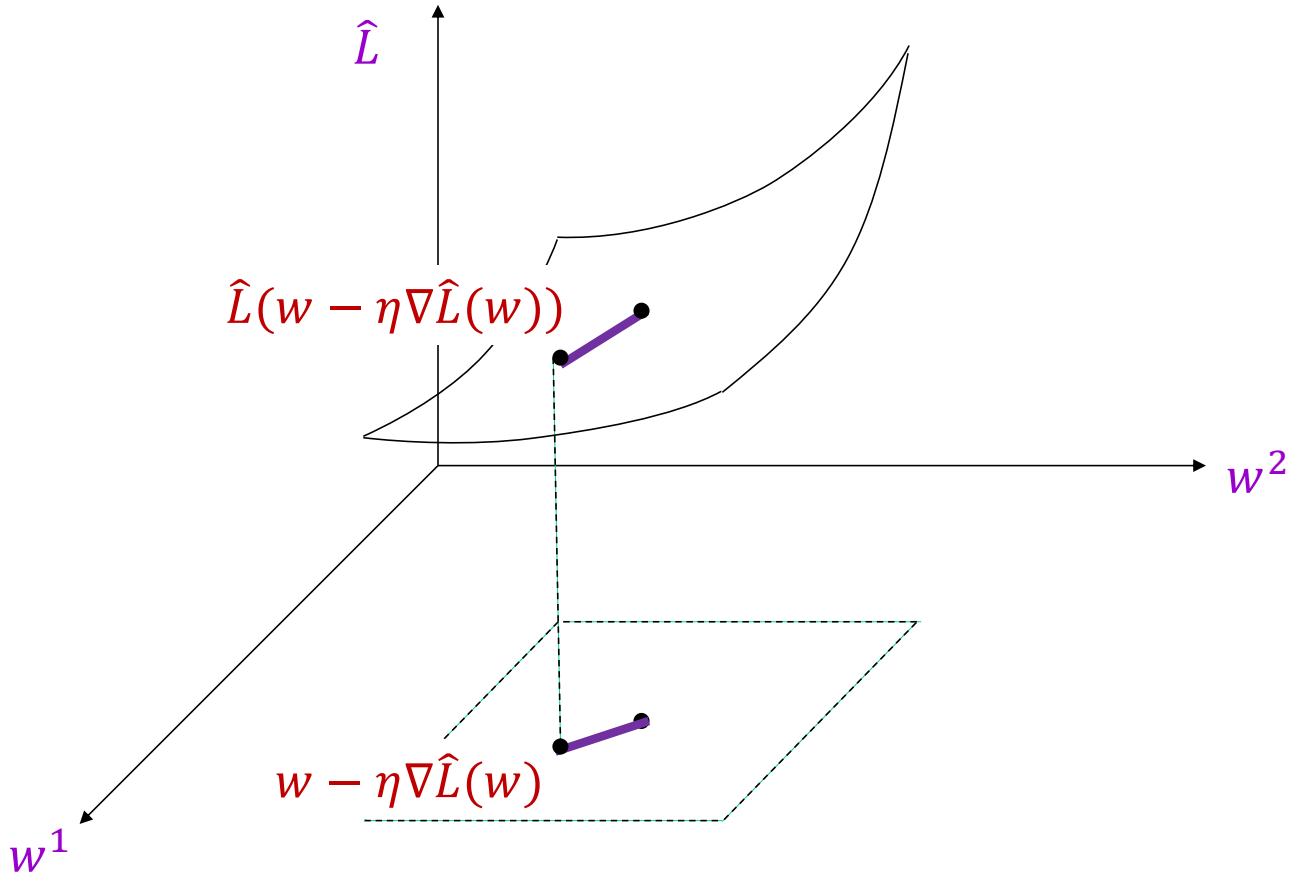
The gradient vector



The gradient vector

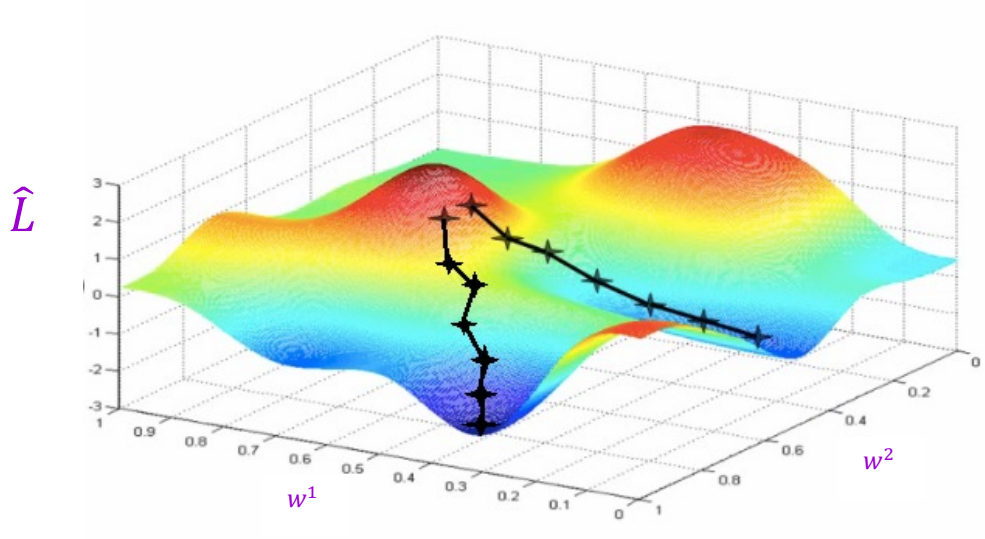


Gradient descent



Gradient descent

- Goal: find w to minimize loss $\hat{L}(w)$
- Start with some initial estimate of w
- Repeat until convergence:
  - Find $\nabla\hat{L}(w)$, the *gradient* of the loss w.r.t. w
  - Take a small step in the *opposite* direction: $w \leftarrow w - \eta\,\nabla\hat{L}(w)$



Gradient descent

- Goal: find w to minimize loss $\hat{L}(w)$
- Start with some initial estimate of w
- Repeat until convergence:
  - Find $\nabla\hat{L}(w)$, the *gradient* of the loss w.r.t. w
  - Take a small step in the *opposite* direction: $w \leftarrow w - \eta\,\nabla\hat{L}(w)$
  - η is the step size or *learning rate*

Full batch gradient descent

- Since $\hat{L}(w) = \frac{1}{n}\sum_{i=1}^{n} l(w, x_i, y_i)$, we have

$$\nabla\hat{L}(w) = \frac{1}{n}\sum_{i=1}^{n} \nabla l(w, x_i, y_i)$$

- For a single parameter update, need to cycle through the entire training set!

Stochastic gradient descent (SGD)

- At each iteration, take a *single data point* (x_i, y_i) and perform a parameter update using $\nabla l(w, x_i, y_i)$, the gradient of the loss for that point:

$$w \leftarrow w - \eta \, \nabla l(w, x_i, y_i)$$

- This is called an *online* or *stochastic* update
- In practice, *mini-batch SGD* is typically used:
  - Group data into mini-batches of size b
    - Compute gradient of the loss for the mini-batch $(x_1, y_1), \ldots, (x_b, y_b)$:

$$\nabla \hat{L} = \frac{1}{b} \sum_{i=1}^{b} \nabla l(w, x_i, y_i)$$

    - Update parameters: $w \leftarrow w - \eta \nabla \hat{L}$

SGD for logistic regression

$$\hat{L}(w) = -\frac{1}{n}\sum_{y_i=1} \log \sigma(w^T x_i) - \frac{1}{n}\sum_{y_i=0} \log[1 - \sigma(w^T x_i)]$$

$$= -\frac{1}{n}\sum_{i=0}^{n} y_i \log\sigma(w^T x_i) + (1 - y_i)\log[1 - \sigma(w^T x_i)]$$

- The gradient:

$$\nabla_w \hat{L}(w) = \frac{1}{n}\sum_{i=0}^{n} (\sigma(w^T x_i) - y_i)x_i$$

Exercise

SGD for logistic regression

$$\hat{L}(w) = -\frac{1}{n}\sum_{y_i=1} \log \sigma(w^T x_i) - \frac{1}{n}\sum_{y_i=0} \log[1 - \sigma(w^T x_i)]$$

$$= -\frac{1}{n}\sum_{i=0}^{n} y_i \log\sigma(w^T x_i) + (1 - y_i)\log[1 - \sigma(w^T x_i)]$$

- The gradient:

$$\nabla_w \hat{L}(w) = \frac{1}{n}\sum_{i=0}^{n} (\sigma(w^T x_i) - y_i)x_i$$

Focus on ambiguous points, proportional. to feature

Problem with linear regression

- Robustness to outliers:

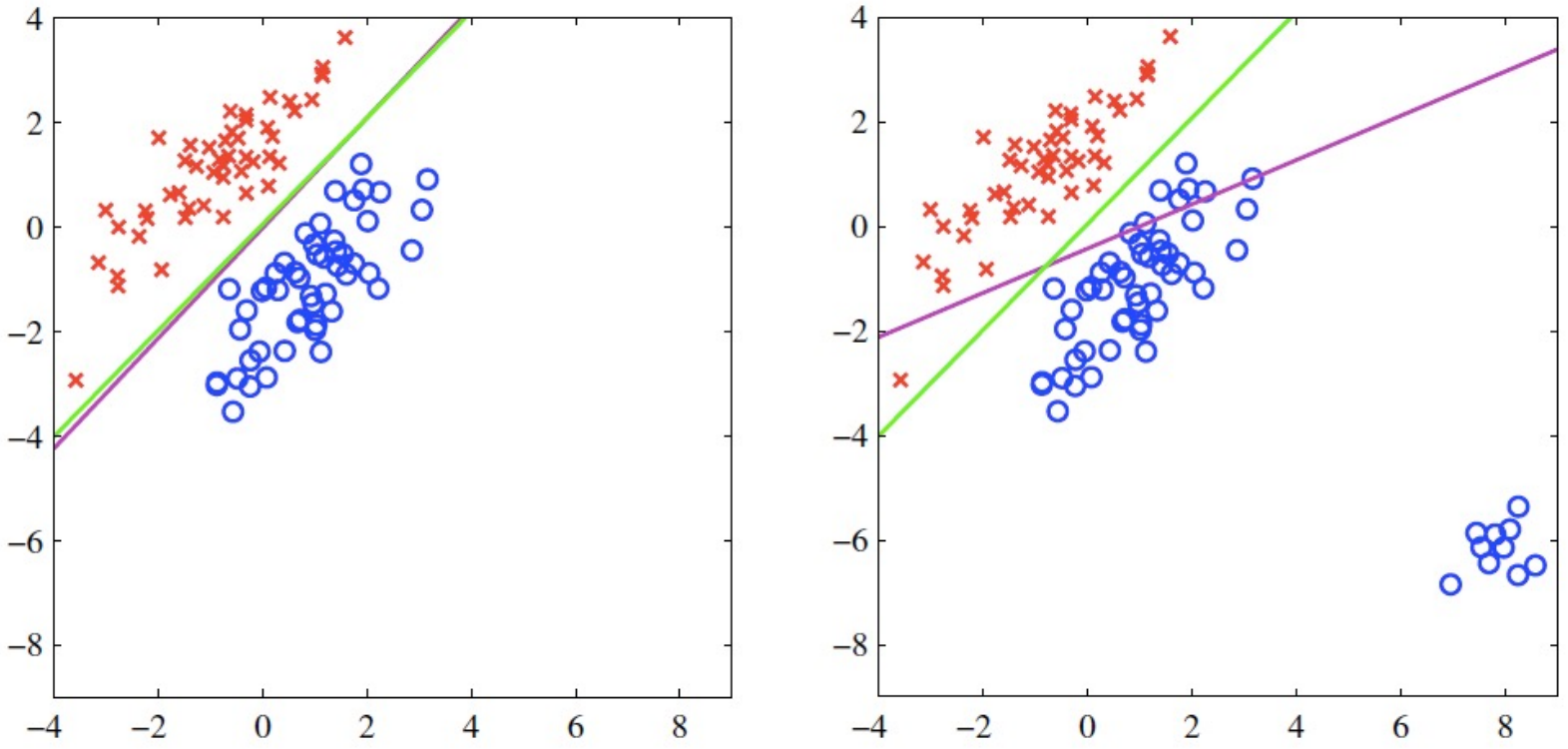


Figure from *Pattern Recognition and Machine Learning*, Bishop

Multiple classes

Multiple classes

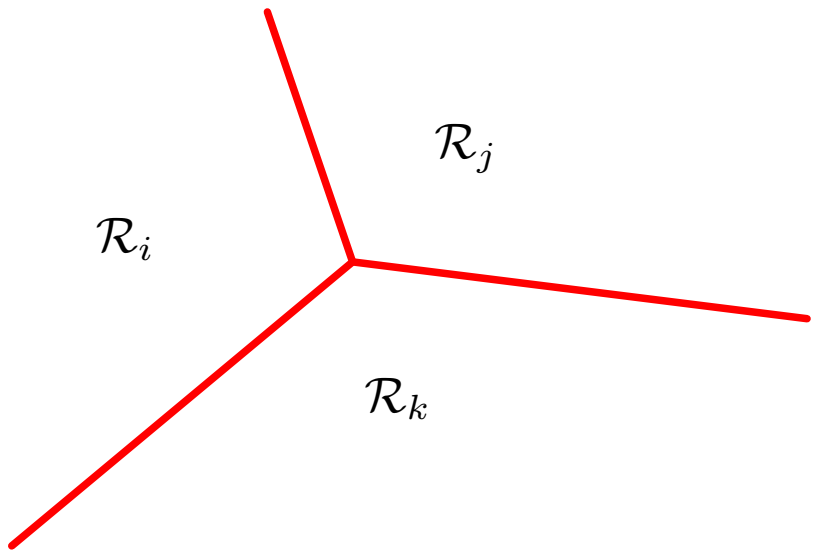
Previously: $y = w^T x \in \mathbb{R}$

Now: $y = Wx \in \mathbb{R}^K$

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

Assign point to class k if:

$$\boldsymbol{y_k(x) > y_j(x)}$$



Multiple classes

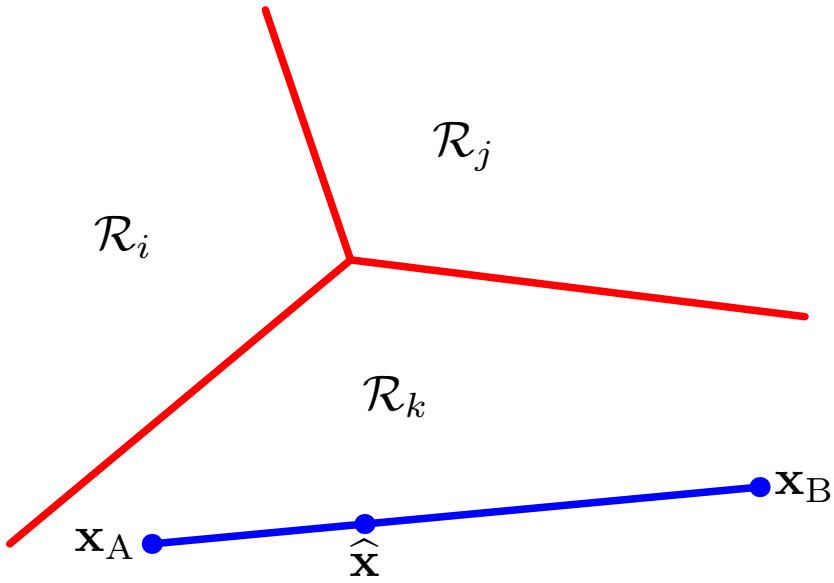
Previously: $y = w^T x \in \mathbb{R}$

Now: $y = Wx \in \mathbb{R}^K$

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

Assign point to class k if:

$$\boldsymbol{y_k(x) > y_j(x)}$$



Regions are convex

Multiple classes

Image

Array of 32x32x3 numbers
(3072 numbers total)

$$f(x,W) = Wx + b$$

3072x1 (x)

10x1 (f(x,W))

10x3072 (W)

10x1 (b)

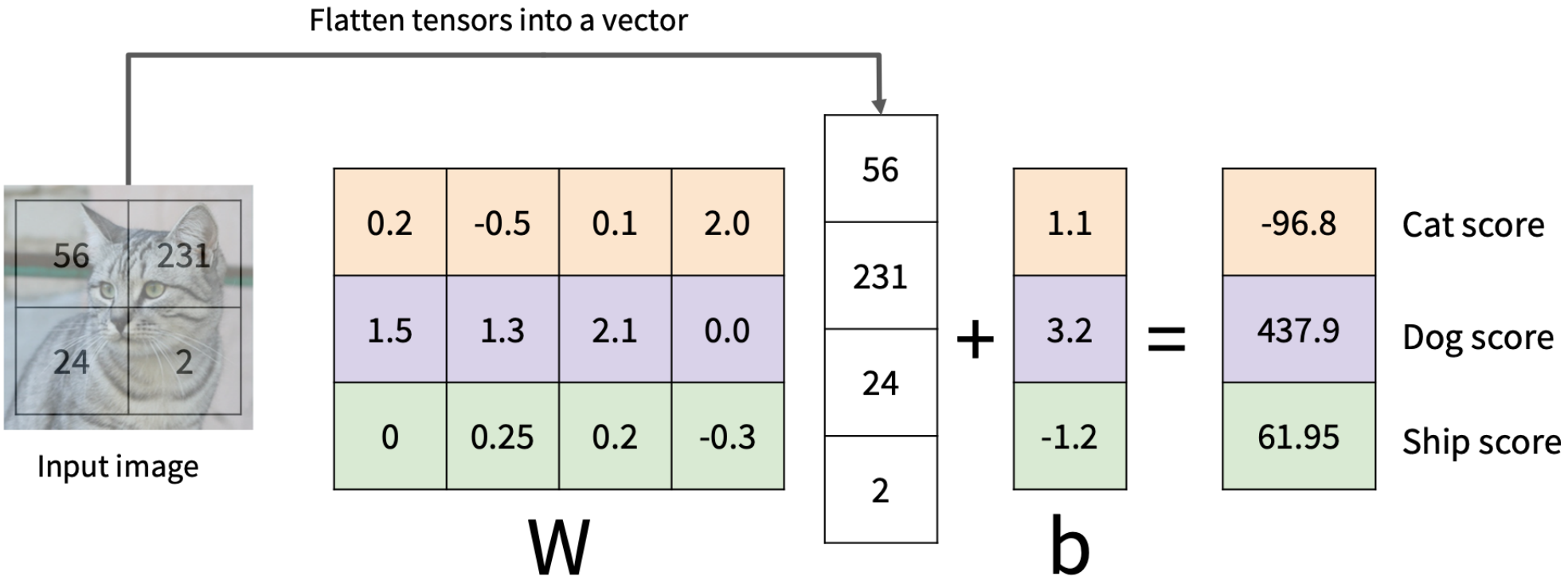
f(x,W)

10 numbers giving class scores

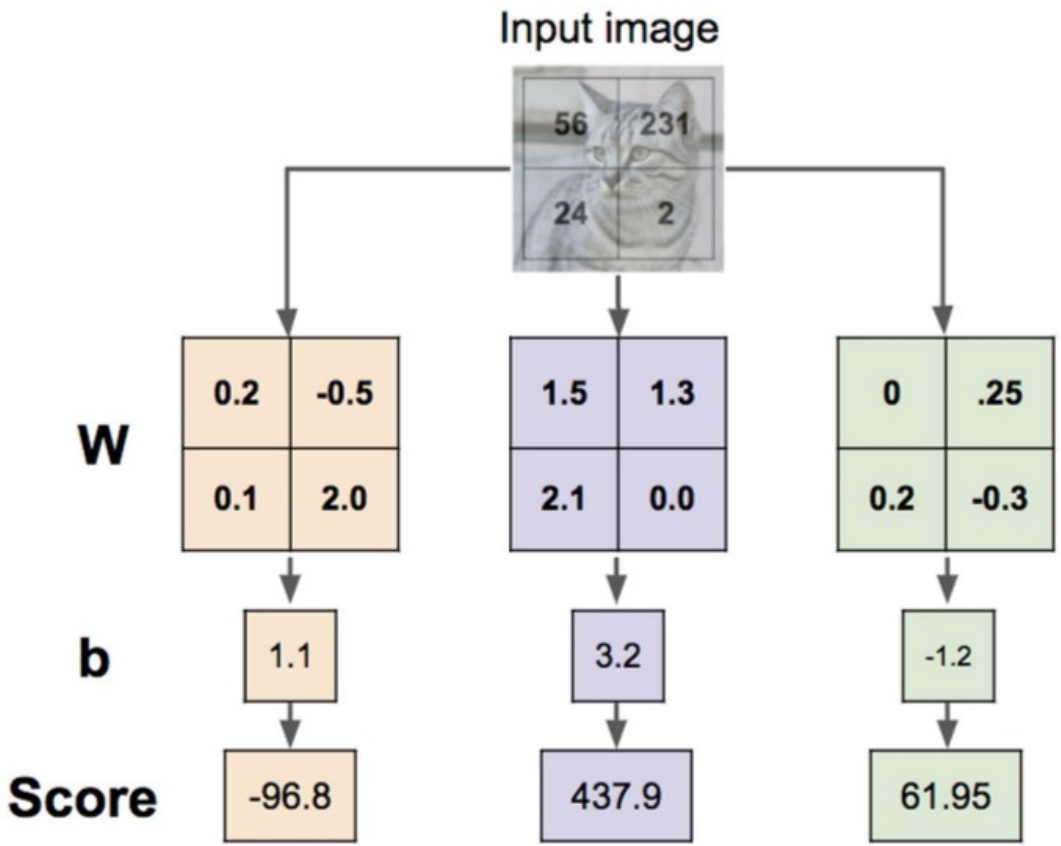
W

parameters or weights

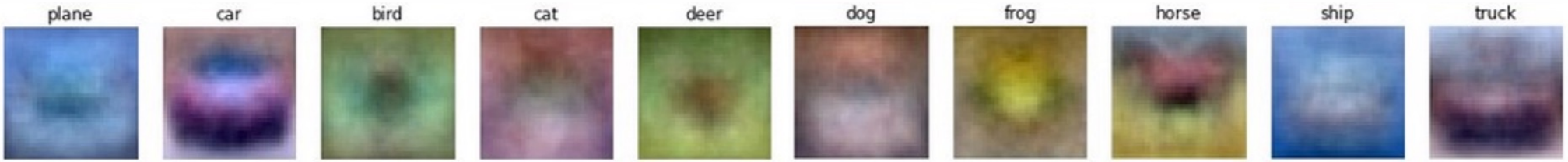
Algebraic Viewpoint: 3 classes (cat/dog/ship)



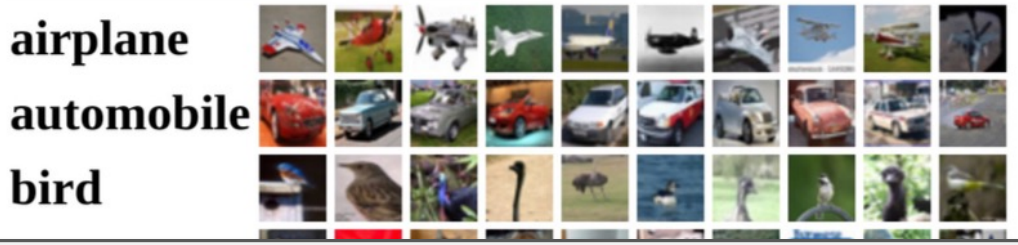
Interpretation



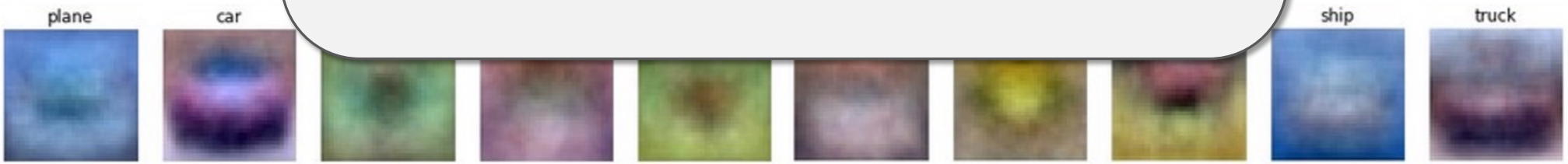
Interpretation



Interpretation



But we've seen the limitations of linear classifiers trained with the L2 loss



Multinomial logistic regression (cross-entropy)

If there are K > 2 classes:

$$p(C_k|x) = \frac{p(x|C_k)p(C_k)}{\sum_j p(x|C_j)p(C_j)} = \frac{exp(a_k)}{\sum_j exp(a_j)}$$

Where,

$$a_k = ln(p(x|C_k)p(C_k)$$

Multinomial logistic regression (cross-entropy)

If there are K > 2 classes:

$$p(C_k|x) = \frac{p(x|C_k)p(C_k)}{\sum_j p(x|C_j)p(C_j)} = \frac{exp(a_k)}{\sum_j exp(a_j)}$$

Where,

$$a_k = ln(p(x|C_k)p(C_k)$$

Probabilistic interpretation: nonnegative, sum up to 1.

Softmax

Multinomial logistic regression (cross-entropy)

If there are K > 2 classes:

$$p(C_k|x) = \frac{p(x|C_k)p(C_k)}{\sum_j p(x|C_j)p(C_j)} = \frac{exp(a_k)}{\sum_j exp(a_j)}$$

Where,

$$a_k = ln(p(x|C_k)p(C_k)$$

Prob m up to 1.

Smoothed max:

If $a_k \gg a_j$ then $p(C_k) \cong 1, p(C_j) \cong 0$

Softmax

Softmax classifier

Want to interpret raw classifier scores as probabilities

$$s = f(x_i; W)$$

$$P(Y = k|X = x_i) = \frac{e^{s_k}}{\sum_j e^{s_j}}$$

Softmax Function

Probabilities must be >= 0

Probabilities must sum to 1

	Unnormalized log-probabilities / logits	exp	unnormalized probabilities	normalize	probabilities
cat	3.2	→	24.5	→	0.13
car	5.1		164.0		0.87
frog	-1.7		0.18		0.00

Softmax classifier

Want to interpret raw classifier scores as probabilities

$$s = f(x_i; W)$$

$$P(Y = k|X = x_i) = \frac{e^{s_k}}{\sum_j e^{s_j}}$$

Softmax Function

Probabilities must be >= 0

Probabilities must sum to 1

	Unnormalized log-probabilities / logits	exp	unnormalized probabilities	normalize	probabilities
cat	3.2		24.5		0.13
car	5.1		164.0		0.87
frog	-1.7		0.18		0.00

Maximize

or minimize its –log()

Softmax classifier

Want to interpret raw classifier scores as probabilities

$$s = f(x_i; W)$$

$$P(Y = k|X = x_i) = \frac{e^{s_k}}{\sum_j e^{s_j}}$$

Softmax Function

Probabilities must be >= 0

Probabilities must sum to 1

	Unnormalized log-probabilities / logits	exp	unnormalized probabilities	normalize	probabilities
cat	3.2		24.5		0.13
car	5.1		164.0		0.87
frog	-1.7		0.18		0.

Maximize

or minimize its –log()

Also interpreted as the KL divergence from the label: [1,0,0]

Softmax classifier

Want to interpret raw classifier scores as probabilities

$$s = f(x_i; W)$$

$$P(Y = k|X = x_i) = \frac{e^{s_k}}{\sum_j e^{s_j}}$$

Softmax Function

Probabilities must be >= 0

Probabilities must sum to 1

	Unnormalized log-probabilities / logits		exp		normalize		
cat	3.2	→	24.5	→	0.13	→	**Maximize**
car	5.1	→	164.0	→	0.87	→	**or minimize its –log()**
frog	-1.7	→					

What happens when we maximize the k element of the softmax?

Softmax classifier

Want to interpret raw classifier scores as probabilities

$$s = f(x_i; W)$$

$$P(Y = k|X = x_i) = \frac{e^{s_k}}{\sum_j e^{s_j}}$$

Softmax Function

Probabilities must be >= 0

Probabilities must sum to 1

	Unnormalized log-probabilities / logits	exp	Probabilities must be >= 0	normalize	Probabilities must sum to 1
cat	3.2	→	24.5	→	0.13
car	5.1		164.0		0.87
frog	-1.7				

→ **Maximize**

or minimize its –log()

The initial training loss ~ log(K)

Recap

- Supervised training formulation
- Linear models
  - Expected loss / Empirical loss
  - I.I.D
  - Performance measure (loss functions)
  - Regularization
  - Hyperparameters
  - Pseduo-inverse
  - Probabilistic view
- Classification with logistic regression
  - Linear classifiers
  - Sigmoid --> logistic loss
  - SGD
  - **Multi-class classification**
  - Interpretation
  - Cross entropy / softmax