Lecture 2: Basic Artificial Neural Networks

Xuming He
SIST, ShanghaiTech
Fall, 2019

Logistics

- Course project
  - Each team is up to 3 members
  - You should form the team right after the National holiday week

- Homework
  - Programming: not the same as CS231n
  - Write-up: problem set
  - HW1 out next Tuesday

- Quiz
  - ~20 mins
  - Q1 next Tuesday ~30 mins

Outline

- Review: Supervised learning
  - Linear regression
- Artificial neuron
  - Neuron models
  - Perceptron algorithm
- Single layer neural networks
  - Network models

Acknowledgement: Hugo Larochelle's, Mehryar Mohri@NYU's & Yingyu Liang@Princeton's course notes

Learning problem

- Problem setup
  - Given training data $\{(x_i, y_i): 1 \le i \le n\}$ i.i.d. from distribution D
  - Find $y = f(x) \in \mathcal{H}$ that minimizes $\hat{L}(f) = \frac{1}{n}\sum_{i=1}^{n} l(f, x_i, y_i)$
  - s.t. the expected loss is small

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

Linear regression

- Formulation
  - Given training data $\{(x_i, y_i): 1 \leq i \leq n\}$ i.i.d. from distribution D
  - Find $f_w(x) = w^T x$ that minimizes $\hat{L}(f_w) = \frac{1}{n} \sum_{i=1}^{n} (w^T x_i - y_i)^2$

l_2 loss; also called mean square error

Hypothesis class $\mathcal{H}$

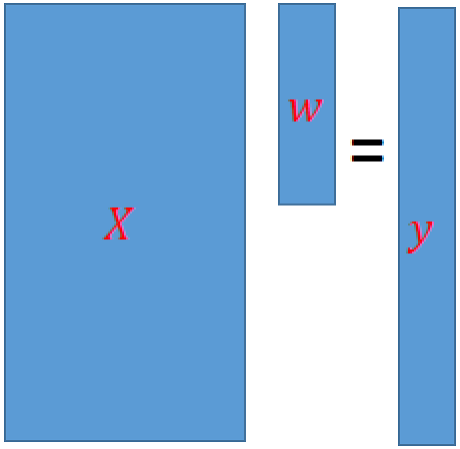
Linear regression

Optimization

- Given training data $\{(x_i, y_i): 1 \le i \le n\}$ i.i.d. from distribution D
- Find $f_w(x) = w^T x$ that minimizes $\hat{L}(f_w) = \frac{1}{n}\sum_{i=1}^{n}(w^T x_i - y_i)^2$

- Let X be a matrix whose i-th row is x_i^T, y be the vector $(y_1, \dots, y_n)^T$

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



Linear regression

- Optimization
  - Set the gradient to 0 to get the minimizer

$$\nabla_w \hat{L}(f_w) = \nabla_w \frac{1}{n} \|Xw - y\|_2^2 = 0$$

$$\nabla_w [(Xw - y)^T (Xw - y)] = 0$$

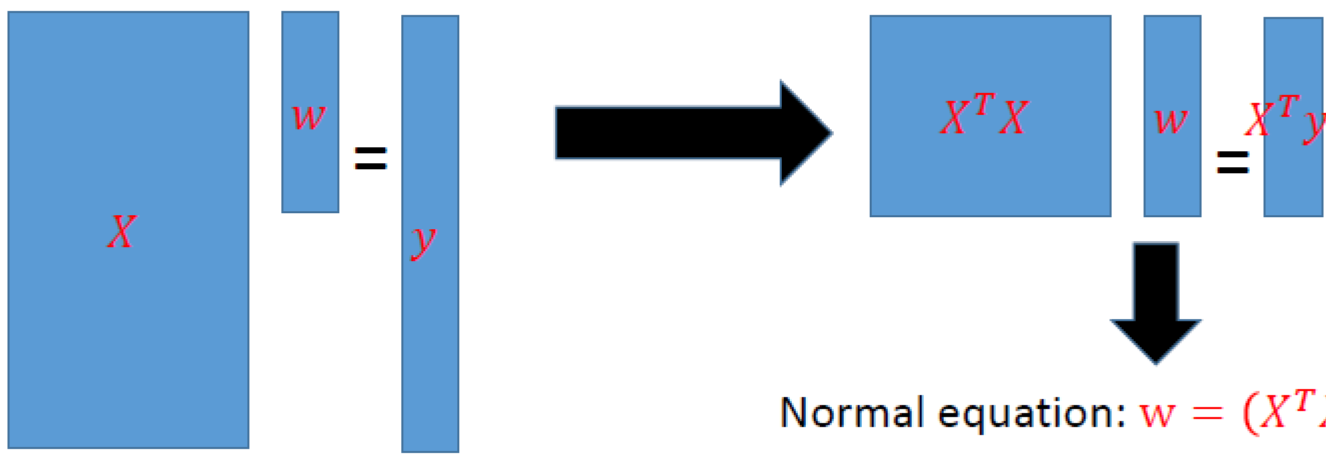
$$\nabla_w [\, w^T X^T X w - 2 w^T X^T y + y^T y] = 0$$

$$2X^T X w - 2X^T y = 0$$

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

Linear regression

- Optimization
  - Algebraic view of the minimizer
    - If X is invertible, just solve $Xw = y$ and get $w = X^{-1}y$
    - But typically X is a tall matrix



Normal equation: $\mathrm{w} = (X^TX)^{-1}X^Ty$

- What if not invertible?

Linear regression

- With bias term

Bias term

- Given training data $\{(x_i, y_i): 1 \leq i \leq n\}$ i.i.d. from distribution D
- Find $f_{w,b}(x) = w^T x + b$ to minimize the loss

- Reduce to the case without bias:
  - Let $w' = [w;\ b], x' = [x; 1]$
  - Then $f_{w,b}(x) = w^T x + b = (w')^T (x')$

Linear regression

- Why l_2 loss?
  - Why not choose another loss
    - l_1 loss, hinge loss, exponential loss, …
  - Empirical: easy to optimize
    - For linear case: $\mathrm{w} = (X^T X)^{-1} X^T y$
  - Theoretical: a way to encode prior knowledge

Questions:

- What kind of prior knowledge?
- Principal way to derive loss?

A probabilistic view

- Maximum likelihood estimation (MLE)
  - Given training data $\{(x_i, y_i): 1 \leq i \leq n\}$ i.i.d. from distribution D
  - Let $\{P_\theta(x, y): \theta \in \Theta\}$ be a family of distributions indexed by θ
  - Would like to pick θ so that $P_\theta(x, y)$ fits the data well

A probabilistic view

- Maximum likelihood estimation (MLE)
  - Given training data $\{(x_i, y_i): 1 \leq i \leq n\}$ i.i.d. from distribution D
  - Let $\{P_\theta(x, y): \theta \in \Theta\}$ be a family of distributions indexed by θ
  - "fitness" of θ to one data point (x_i, y_i)

    $$\text{likelihood}(\theta; x_i, y_i) := P_\theta(x_i, y_i)$$

A probabilistic view

- Maximum likelihood estimation (MLE)
  - Given training data $\{(x_i, y_i): 1 \le i \le n\}$ i.i.d. from distribution D
  - Let $\{P_\theta(x, y): \theta \in \Theta\}$ be a family of distributions indexed by θ
  - "fitness" of θ to i.i.d. data points $\{(x_i, y_i)\}$

$$\text{likelihood}(\theta; \{x_i, y_i\}) := P_\theta(\{x_i, y_i\}) = \prod_i P_\theta(x_i, y_i)$$

A probabilistic view

- Maximum likelihood estimation (MLE)
  - Given training data $\{(x_i, y_i): 1 \leq i \leq n\}$ i.i.d. from distribution D
  - Let $\{P_\theta(x, y): \theta \in \Theta\}$ be a family of distributions indexed by θ
  - MLE: maximize “fitness” of θ to i.i.d. data points $\{(x_i, y_i)\}$

$$\theta_{ML} = \text{argmax}_{\theta \in \Theta} \prod_i P_\theta(x_i, y_i)$$

A probabilistic view

- Maximum likelihood estimation (MLE)
  - Given training data $\{(x_i, y_i): 1 \leq i \leq n\}$ i.i.d. from distribution D
  - Let $\{P_\theta(x, y): \theta \in \Theta\}$ be a family of distributions indexed by θ
  - MLE: maximize "fitness" of θ to i.i.d. data points $\{(x_i, y_i)\}$

$$\theta_{ML} = \mathrm{argmax}_{\theta\in\Theta} \log[\textstyle\prod_i P_\theta(x_i, y_i)]$$

$$\theta_{ML} = \mathrm{argmax}_{\theta\in\Theta} \textstyle\sum_i \log[P_\theta(x_i, y_i)]$$

A probabilistic view

- Maximum likelihood estimation (MLE)
  - Given training data $\{(x_i, y_i): 1 \leq i \leq n\}$ i.i.d. from distribution D
  - Let $\{P_\theta(x, y): \theta \in \Theta\}$ be a family of distributions indexed by θ
  - MLE: negative log-likelihood loss

$$\theta_{ML} = \text{argmax}_{\theta\in\Theta} \sum_i \log(P_\theta(x_i, y_i))$$

$$l(P_\theta, x_i, y_i) = -\log(P_\theta(x_i, y_i))$$
$$\hat{L}(P_\theta) = -\sum_i \log(P_\theta(x_i, y_i))$$

A probabilistic view

- MLE: conditional log-likelihood
  - Given training data $\{(x_i, y_i): 1 \le i \le n\}$ i.i.d. from distribution D
  - Let $\{P_\theta(y|x): \theta \in \Theta\}$ be a family of distributions indexed by θ

Only care about predicting y from x; do not care about p(x)

  - MLE: negative conditional log-likelihood loss

$$\theta_{ML} = \text{argmax}_{\theta\in\Theta} \sum_i \log(P_\theta(y_i|x_i))$$

$$l(P_\theta, x_i, y_i) = -\log(P_\theta(y_i|x_i))$$

$$\hat{L}(P_\theta) = -\sum_i \log(P_\theta(y_i|x_i))$$

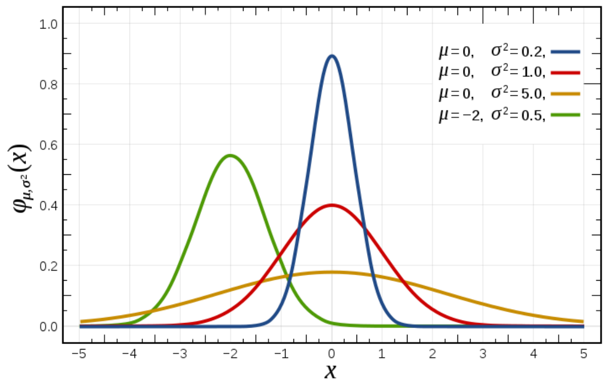
MLE example: Regression

- Regression with l2 Loss

  - Given training data $\{(x_i, y_i): 1 \leq i \leq n\}$ i.i.d. from distribution D
  - Find $f_\theta(x)$ that minimizes $\hat{L}(f_\theta) = \frac{1}{n}\sum_{i=1}^{n}(f_\theta(x_i) - y_i)^2$

l_2 loss: Normal + MLE

  - Define $P_\theta(y|x) = \text{Normal}(y; f_\theta(x), \sigma^2)$
  - $\log(P_\theta(y_i|x_i)) = \frac{-1}{2\sigma^2}(f_\theta(x_i) - y_i)^2 - \log(\sigma) - \frac{1}{2}\log(2\pi)$
  - $\theta_{ML} = \text{argmin}_{\theta\in\Theta}\frac{1}{n}\sum_{i=1}^{n}(f_\theta(x_i) - y_i)^2$

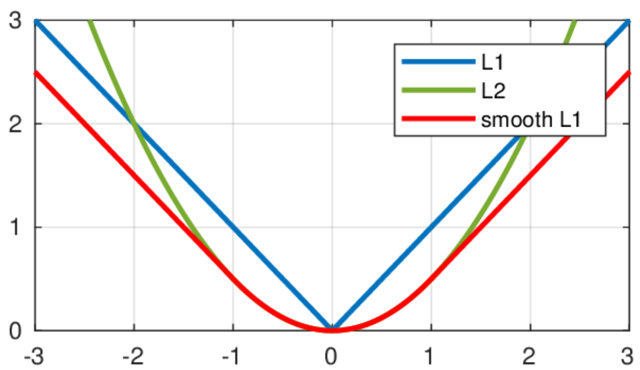


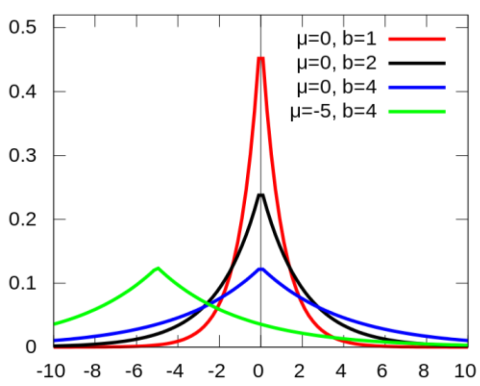
Outliers

- L2 loss = Gaussian distribution
- What if we have outliers?
  - Some data points lie far away from the linear structure

- Robust estimation
  - L1 loss: Laplace distribution

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





Generalization

- Overfitting or under-determined?
  - Ridge regression

$$\hat{L}(f_w) = \frac{1}{n}\sum_{i=1}^{n}\|w^\intercal x_i - y_i\|^2 + \lambda\sum_{j=1}^{d} w_j^2$$

    - The optimal weights $w^* = (X^\intercal X + \lambda I)^{-1}X^\intercal Y$

  - Lasso regression

$$\hat{L}(f_w) = \frac{1}{n}\sum_{i=1}^{n}\|w^\intercal x_i - y_i\|^2 + \lambda\sum_{j=1}^{d} |w_j|$$

    - Convex optimization: proximal gradient descent

  - The hyper-parameter λ controls the model complexity

Outline

- Review: Supervised learning
  - Linear regression
- Artificial neuron
  - Neuron models
  - Perceptron algorithm
- Single layer neural networks
  - Network models
  - Optimization by (sub-)gradient descent

Artificial Neuron

- Biological inspiration
  - Our brain has $\sim 10^{11}$ neurons, each of which communicates (is connected) to $\sim 10^4$ other neurons

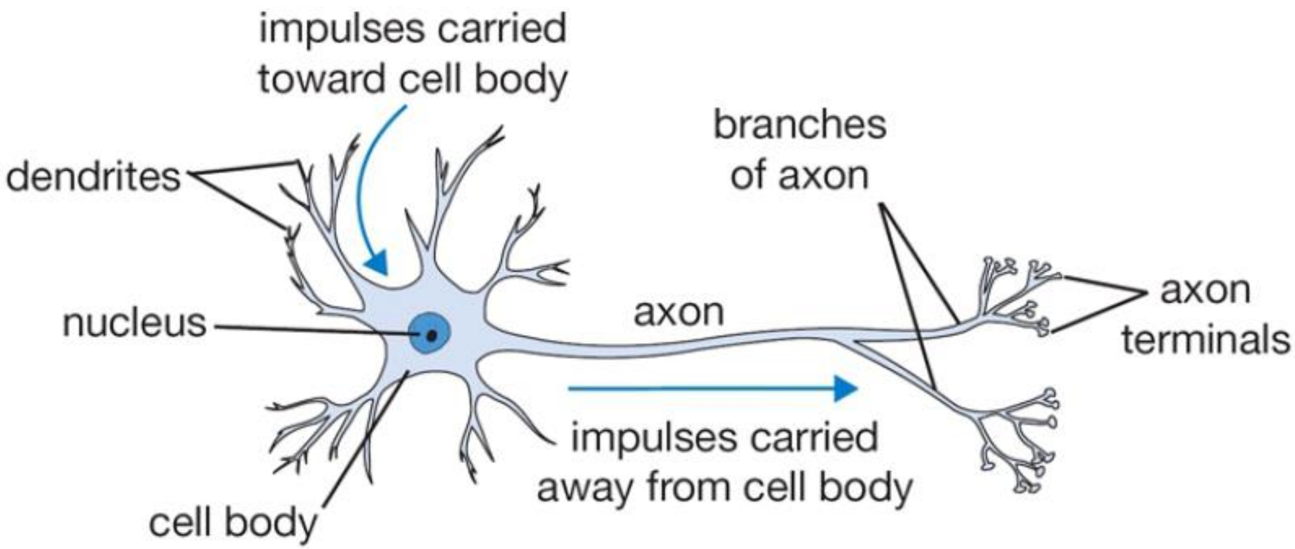
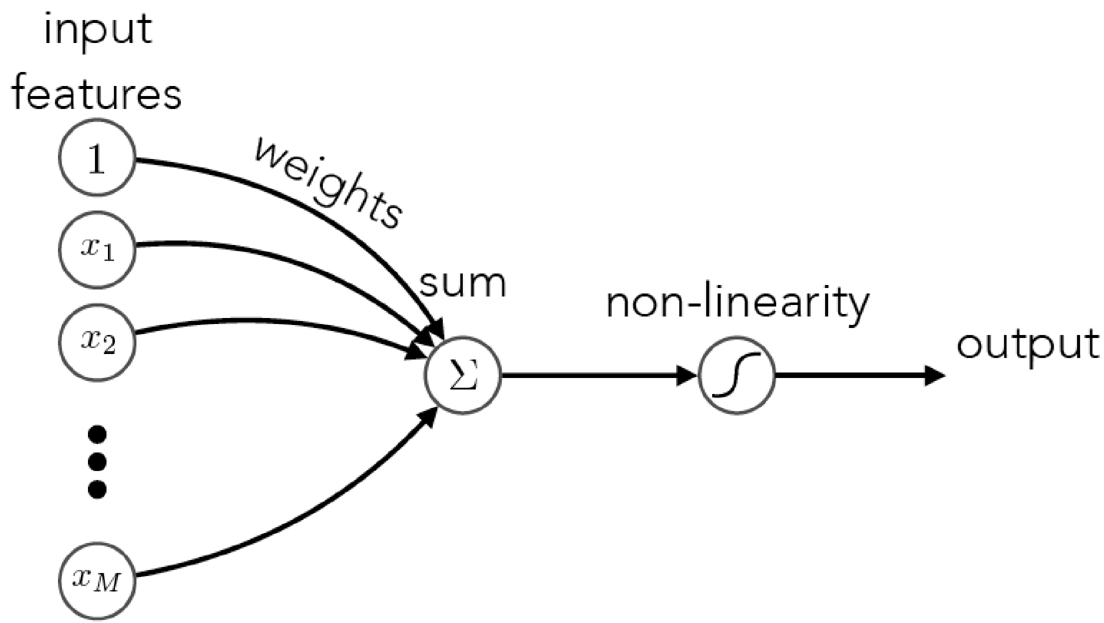


Figure : The basic computational unit of the brain: Neuron

Mathematical model of a neuron



artificial neuron: *weighted sum and non-linearity*

bias   input features

$$s = b + w_1 x_1 + w_2 x_2 + \cdots + w_M x_M = \mathbf{w}^\mathsf{T}\mathbf{x}$$

sum   weights

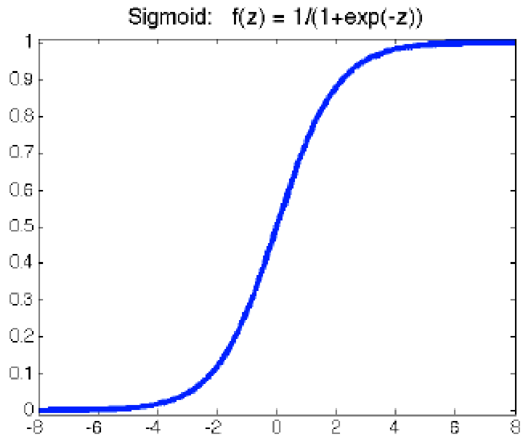
$$h = g(s)$$

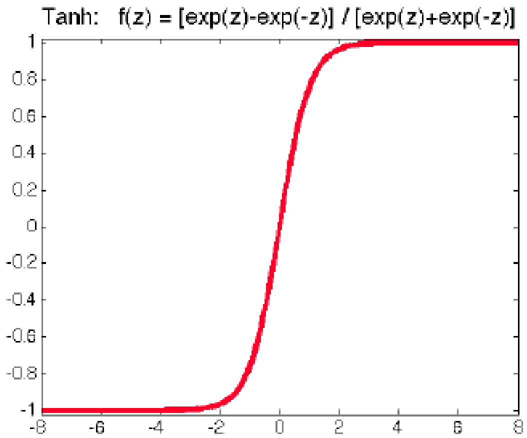
output   non-linearity   sum

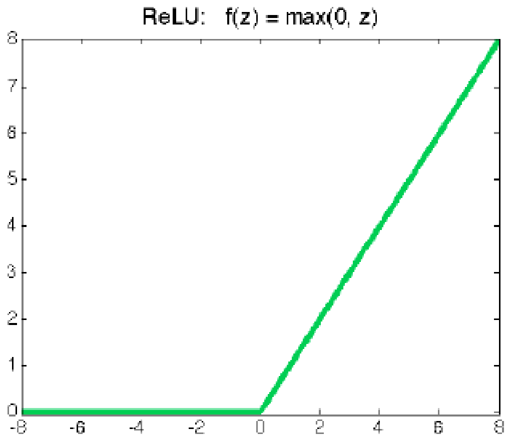
Activation functions

Most commonly used activation functions:

- Sigmoid: $\sigma(z) = \frac{1}{1+\exp(-z)}$
- Tanh: $\tanh(z) = \frac{\exp(z)-\exp(-z)}{\exp(z)+\exp(-z)}$
- ReLU (Rectified Linear Unit): $\mathrm{ReLU}(z) = \max(0, z)$

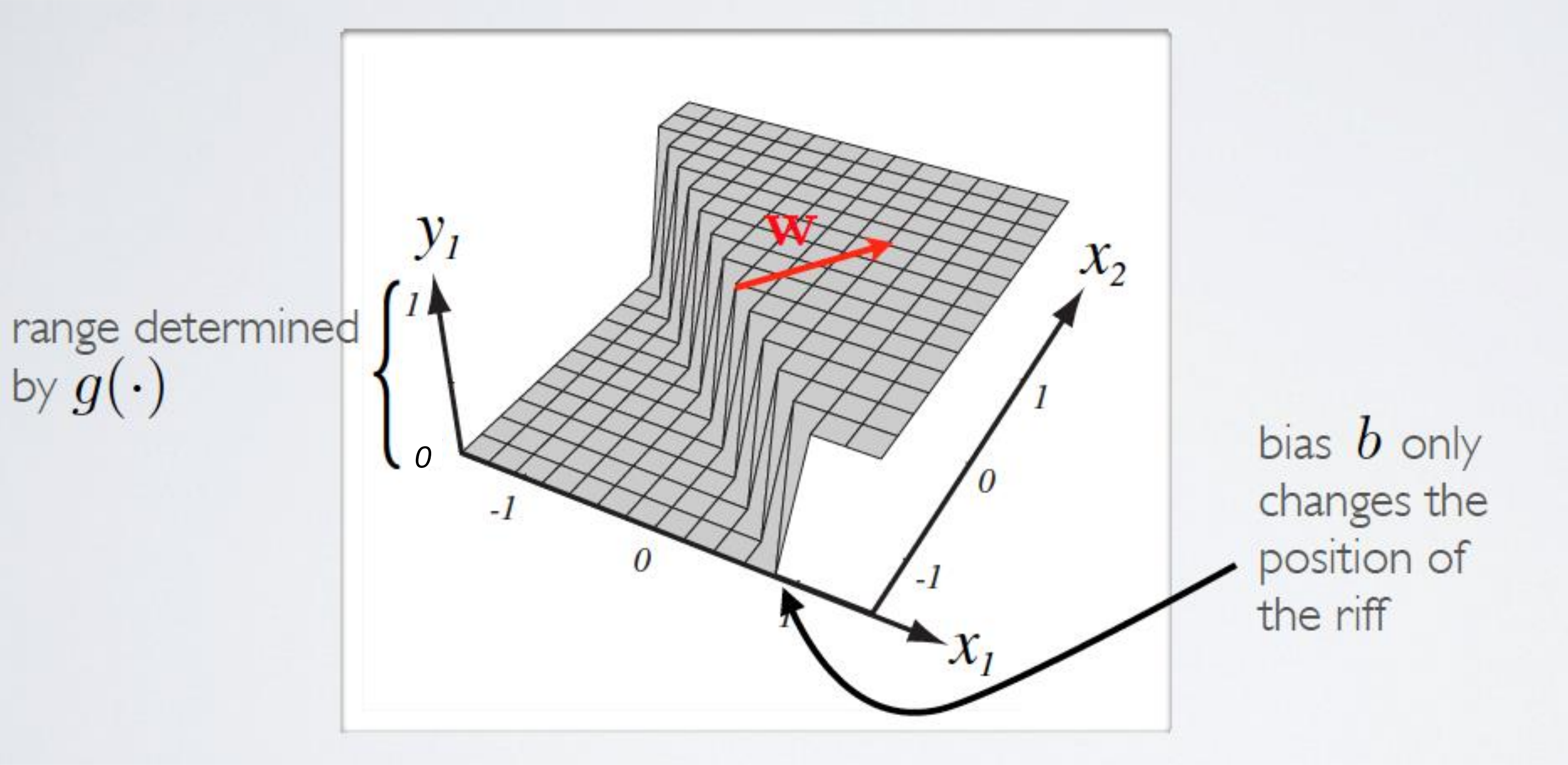






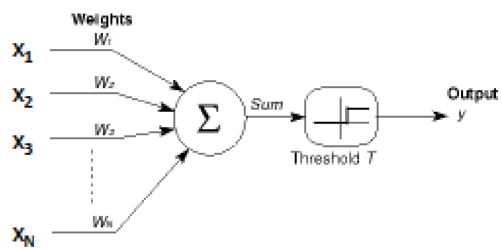
Capacity of single neuron

- Sigmoid activation function



What a single neuron does?

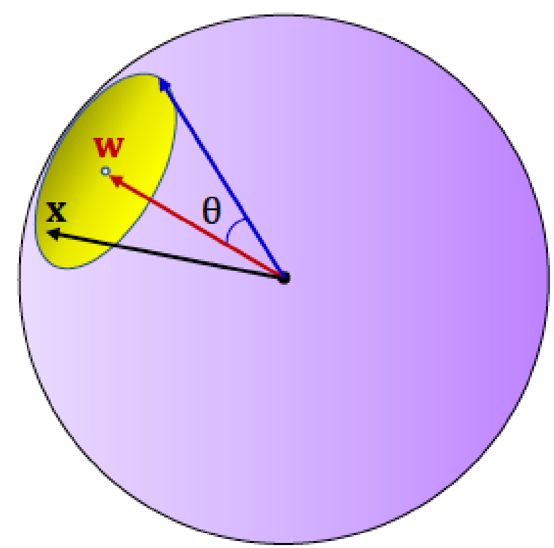
- A neuron (perceptron) fires if its input is within a specific angle of its weight
  - If the input pattern matches the weight pattern closely enough



$$\mathbf{x}^T\mathbf{w} > T$$

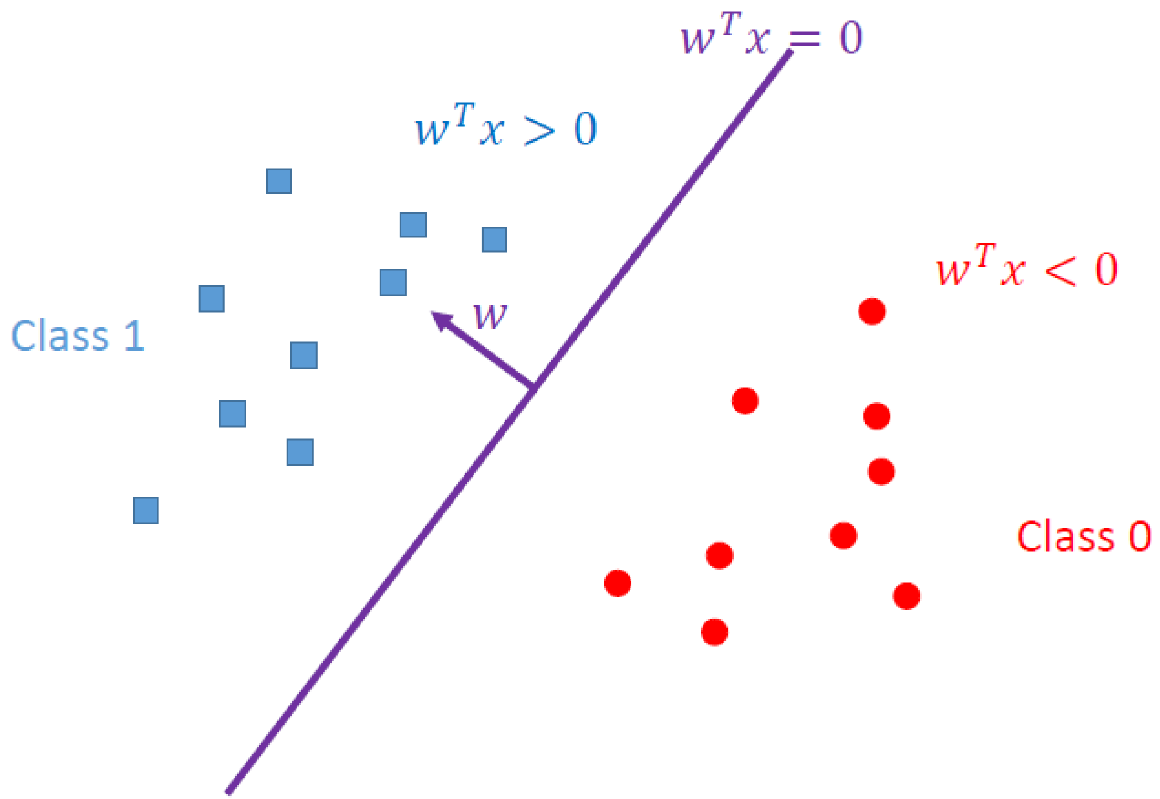
$$\Rightarrow \cos\theta > \frac{T}{|\mathbf{x}||\mathbf{w}|}$$

$$\Rightarrow \theta < \cos^{-1}\left(\frac{T}{|\mathbf{x}||\mathbf{w}|}\right)$$



Single neuron as a linear classifier

- Binary classification



How do we determine the weights?

- Learning problem
  - Given training data $\{(x_i, y_i): 1 \leq i \leq n\}$ i.i.d. from distribution D
  - Hypothesis $f_w(x) = w^T x$
    - $y = 1$ if $w^T x > 0$
    - $y = 0$ if $w^T x < 0$
  - Prediction: $y = \text{step}(f_w(x)) = \text{step}(w^T x)$

Linear model $\mathcal{H}$

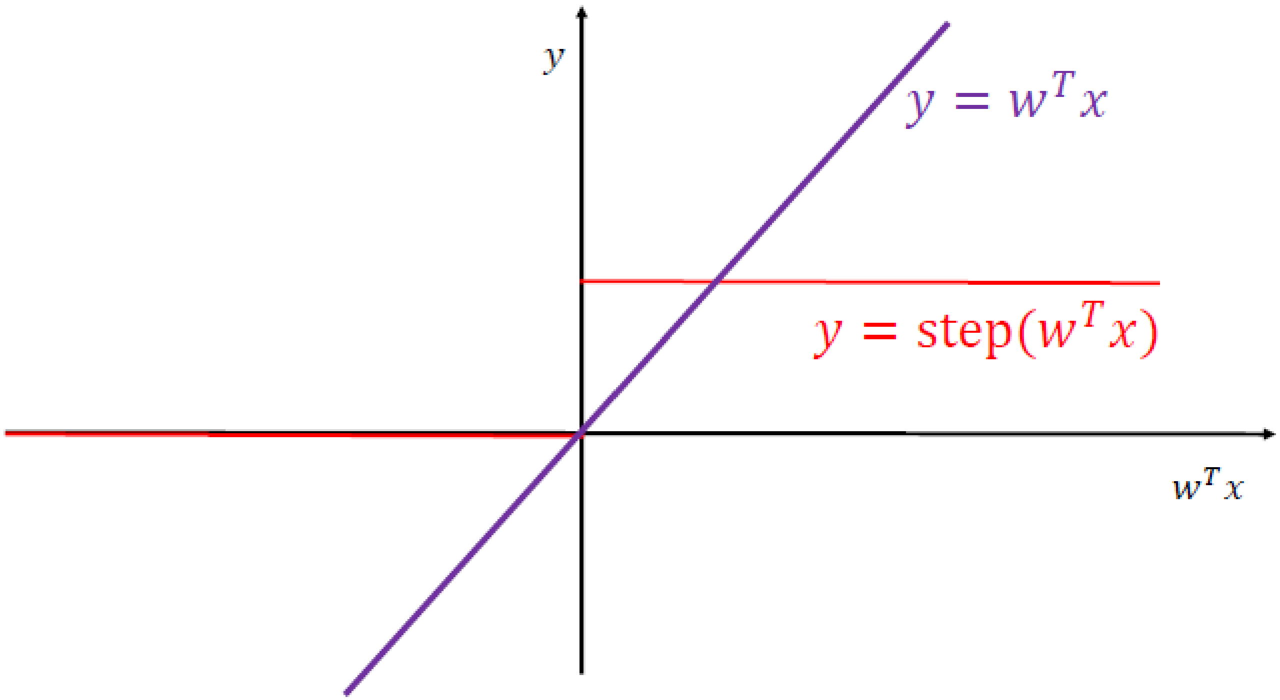
Linear classification

- Learning problem: simple approach
  - Given training data $\{(x_i, y_i): 1 \leq i \leq n\}$ i.i.d. from distribution D
  - Find $f_w(x) = w^T x$ that minimizes $\hat{L}(f_w) = \frac{1}{n}\sum_{i=1}^{n}(w^T x_i - \ y_i)^2$
  - Drawback: Sensitive to "outliers"

Reduce to linear regression; ignore the fact $y \in \{0,1\}$

1D Example

- Compare two predictors



Outline

- Review: Supervised learning
  - Linear regression
- Artificial neuron
  - Neuron models
  - Perceptron algorithm
- Single layer neural networks
  - Network models
  - Optimization by (sub-)gradient descent

Perceptron algorithm

- Learn a single neuron for binary classification

- Task formulation

  - Given training data $\{(x_i, y_i): 1 \leq i \leq n\}$ i.i.d. from distribution D
  - Hypothesis $f_w(x) = w^T x$
    - $y = +1$ if $w^T x > 0$
    - $y = -1$ if $w^T x < 0$
  - Prediction: $y = \text{sign}(f_w(x)) = \text{sign}(w^T x)$

  - Goal: minimize classification error

Perceptron algorithm

- Algorithm outline

• Assume for simplicity: all x_i has length 1

1. Start with the all-zeroes weight vector $\mathbf{w}_1 = \mathbf{0}$, and initialize t to 1.
2. Given example $\mathbf{x}$, predict positive iff $\mathbf{w}_t \cdot \mathbf{x} > 0$.
3. On a mistake, update as follows:
   - Mistake on positive: $\mathbf{w}_{t+1} \leftarrow \mathbf{w}_t + \mathbf{x}$.
   - Mistake on negative: $\mathbf{w}_{t+1} \leftarrow \mathbf{w}_t - \mathbf{x}$.

   $t \leftarrow t + 1$.

Perceptron: figure from the lecture note of Nina Balcan

Perceptron algorithm

- Intuition: correct the current mistake
  - If mistake on a positive example

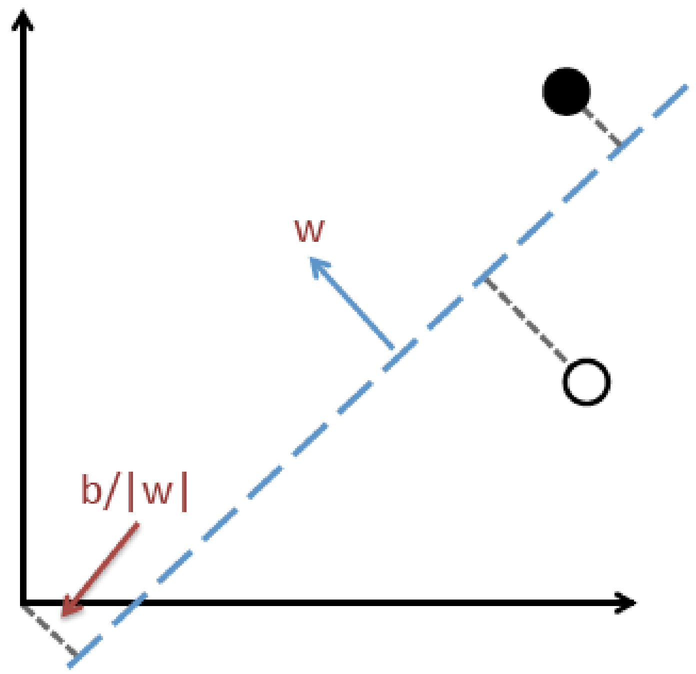
$$w_{t+1}^T x = (w_t + x)^T x = w_t^T x + x^T x = w_t^T x + 1$$

  - If mistake on a negative example

$$w_{t+1}^T x = (w_t - x)^T x = w_t^T x - x^T x = w_t^T x - 1$$

Hyperplane Distance

- Line is a 1D, Plane is 2D
- Hyperplane is many D
  - Includes Line and Plane
- Defined by (w,b)
- Distance: $\dfrac{|w^T x - b|}{\|w\|}$
- Signed Distance: $\dfrac{w^T x - b}{\|w\|}$



Linear Model = un-normalized signed distance!

Perceptron algorithm

- The Perceptron theorem
  - Suppose there exists w^* that correctly classifies $\{(x_i, y_i)\}$
  - W.L.O.G., all x_i and w^* have length 1, so the minimum distance of any example to the decision boundary is

$$\gamma = \min_i |(w^*)^T x_i|$$

  - Then Perceptron makes at most $\left(\frac{1}{\gamma}\right)^2$ mistakes

Perceptron algorithm

- The Perceptron theorem
  - Suppose there exists w^* that correctly classifies $\{(x_i, y_i)\}$
  - W.L.O.G., all x_i and w^* have length 1, so the minimum distance of any example to the decision boundary is

$$\gamma = \min_i |(w^*)^T x_i|$$

Need not be i.i.d. !

  - Then Perceptron makes at most $\left(\frac{1}{\gamma}\right)^2$ mistakes

Do not depend on n, the length of the data sequence!

Perceptron algorithm

- The Perceptron theorem: proof
  - First look at the quantity $w_t^T w^*$
  - Claim 1: $w_{t+1}^T w^* \geq w_t^T w^* + \gamma$
  - Proof: If mistake on a positive example x

$$w_{t+1}^T w^* = (w_t + x)^T w^* = w_t^T w^* + x^T w^* \geq w_t^T w^* + \gamma$$

  - If mistake on a negative example

$$w_{t+1}^T w^* = (w_t - x)^T w^* = w_t^T w^* - x^T w^* \geq w_t^T w^* + \gamma$$

Perceptron algorithm

- The Perceptron theorem: proof

  - Next look at the quantity $||w_t||$

  Negative since we made a mistake on x

  - Claim 2: $||w_{t+1}||^2 \leq ||w_t||^2 + 1$
  - Proof: If mistake on a positive example x

$$||w_{t+1}||^2 = ||w_t + x||^2 = ||w_t||^2 + ||x||^2 + 2w_t^T x$$

Perceptron algorithm

- The Perceptron theorem: proof intuition
  - Claim 1: $w_{t+1}^T w^* \geq w_t^T w^* + \gamma$
  - Claim 2: $||w_{t+1}||^2 \leq ||w_t||^2 + 1$

The correlation gets larger. Could be:

1. w_{t+1} gets closer to w^*
2. w_{t+1} gets much longer

Rules out the bad case "2. w_{t+1} gets much longer"

Perceptron algorithm

- The Perceptron theorem: proof

  - Claim 1: $w_{t+1}^T w^* \geq w_t^T w^* + \gamma$
  - Claim 2: $||w_{t+1}||^2 \leq ||w_t||^2 + 1$

  After M mistakes:

  - $w_{M+1}^T w^* \geq \gamma M$
  - $||w_{M+1}|| \leq \sqrt{M}$
  - $w_{M+1}^T w^* \leq ||w_{M+1}||$

  So $\gamma M \leq \sqrt{M}$, and thus $M \leq \left(\frac{1}{\gamma}\right)^2$

Perceptron Learning problem

- What loss function is minimized?
  - Given training data $\{(x_i, y_i): 1 \leq i \leq n\}$ i.i.d. from distribution D
  - Find $y = f(x) \in \mathcal{H}$ that minimizes $\hat{L}(f) = \frac{1}{n}\sum_{i=1}^{n} l(f, x_i, y_i)$
  - s.t. the expected loss is small

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

Learning as iterative optimization

- Gradient descent
  - choose initial $w^{(0)}$, repeat

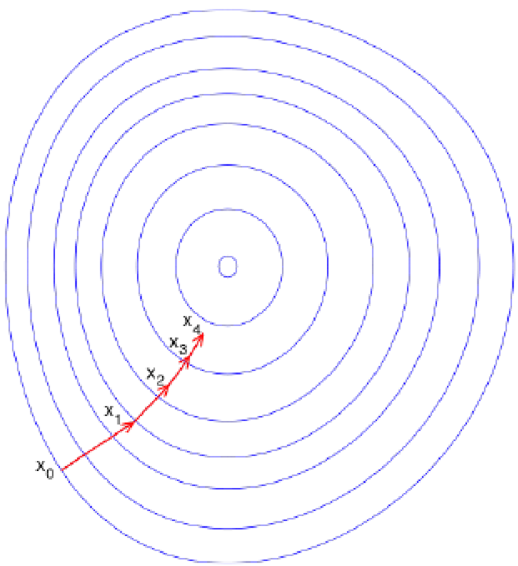
$$w^{(t+1)} = w^{(t)} - \eta_t \cdot \nabla \mathrm{L}\left(w^{(t)}\right)$$

    until stop
  - η_t is the learning rate, and

$$\nabla \mathrm{L}\left(w^{(t)}\right) = \frac{1}{n}\sum_i \nabla_w \mathrm{L_i}(w^{(t)}; y_i, x_i)$$

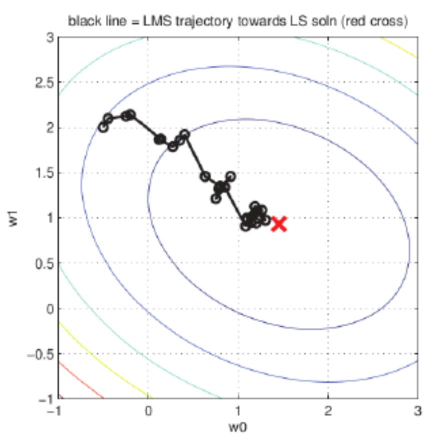
  - How to stop? $\|w^{(t+1)} - w^{(t)}\| \leq \epsilon$ or $\|\nabla \mathrm{L}(w^{(t)})\| \leq \epsilon$

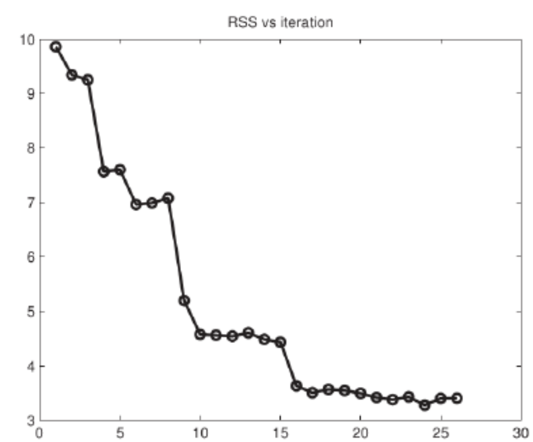
Two dimensional example:



Learning as iterative optimization

- Stochastic gradient descent (SGD)
  - Suppose data points arrive one by one
  - $\hat{L}(\mathrm{w}) = \frac{1}{n}\sum_{t=1}^{n} l(\mathrm{w}, x_t, y_t)$, but we only know $l(\mathrm{w}, x_t, y_t)$ at time t
  - Idea: simply do what you can based on local information
    - Initialize w_0
    - $\mathrm{w}_{t+1} = \mathrm{w}_t - \eta_t \nabla l(\mathrm{w}_t, x_t, y_t)$





Perceptron algorithm

- What loss function is minimized?
  - Hypothesis: $y = \mathrm{sign}(w^T x)$
  - Define hinge loss

$$l(w, x_t, y_t) = -y_t w^T x_t \, \mathbb{I}[\text{mistake on } x_t]$$

$$\hat{L}(w) = -\sum_t y_t w^T x_t \, \mathbb{I}[\text{mistake on } x_t]$$

$$w_{t+1} = w_t - \eta_t \nabla l(w_t, x_t, y_t) = w_t + \eta_t y_t x_t \, \mathbb{I}[\text{mistake on } x_t]$$

Perceptron algorithm

- What loss function is minimized?
  - Hypothesis: $y = \text{sign}(w^T x)$

$$w_{t+1} = w_t - \eta_t \nabla l(w_t, x_t, y_t) = w_t + \eta_t y_t x_t \, \mathbb{I}[\text{mistake on } x_t]$$

  - Set $\eta_t = 1$. If mistake on a positive example

$$w_{t+1} = w_t + y_t x_t = w_t + x$$

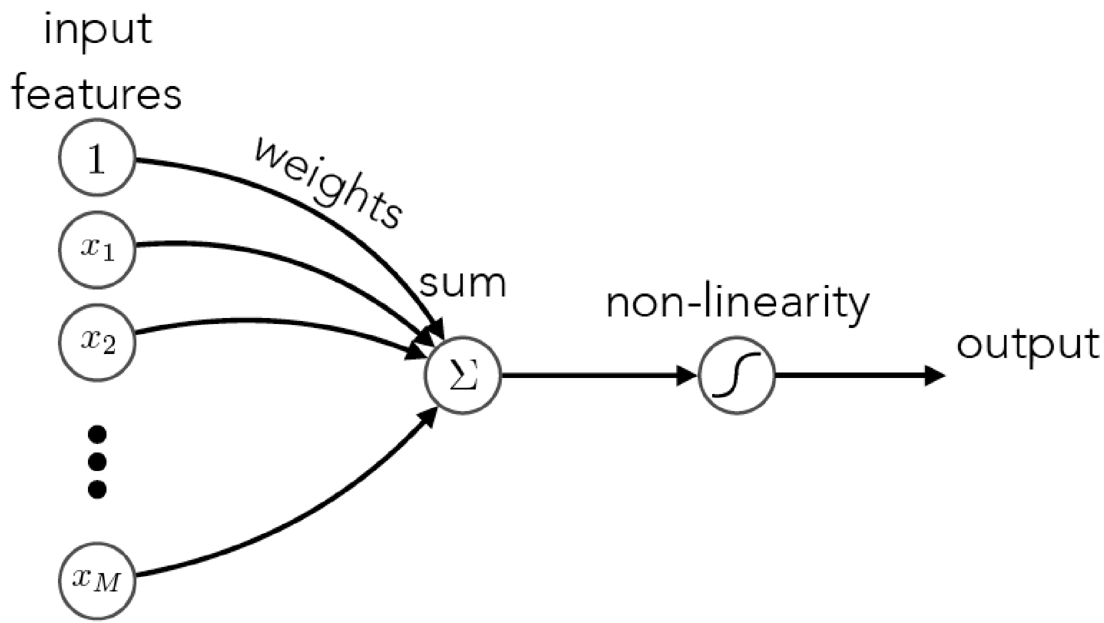
  - If mistake on a negative example

$$w_{t+1} = w_t + y_t x_t = w_t - x$$

Outline

- Review: Supervised learning
  - Linear regression
- Artificial neuron
  - Neuron models
  - Perceptron algorithm
- Single layer neural networks
  - Network models
  - Optimization by (sub-)gradient descent

Mathematical model of a neuron



artificial neuron: *weighted sum and non-linearity*

bias

input features

$$s = b + w_1 x_1 + w_2 x_2 + \cdots + w_M x_M = \mathbf{w}^{\mathsf{T}}\mathbf{x}$$

sum

weights

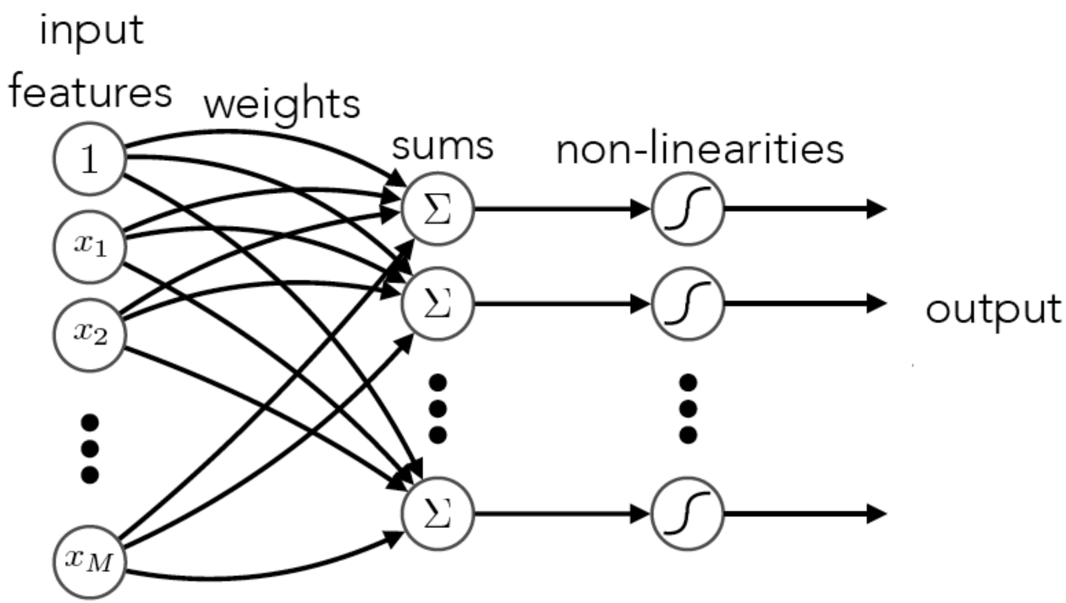
$$h = g(s)$$

output

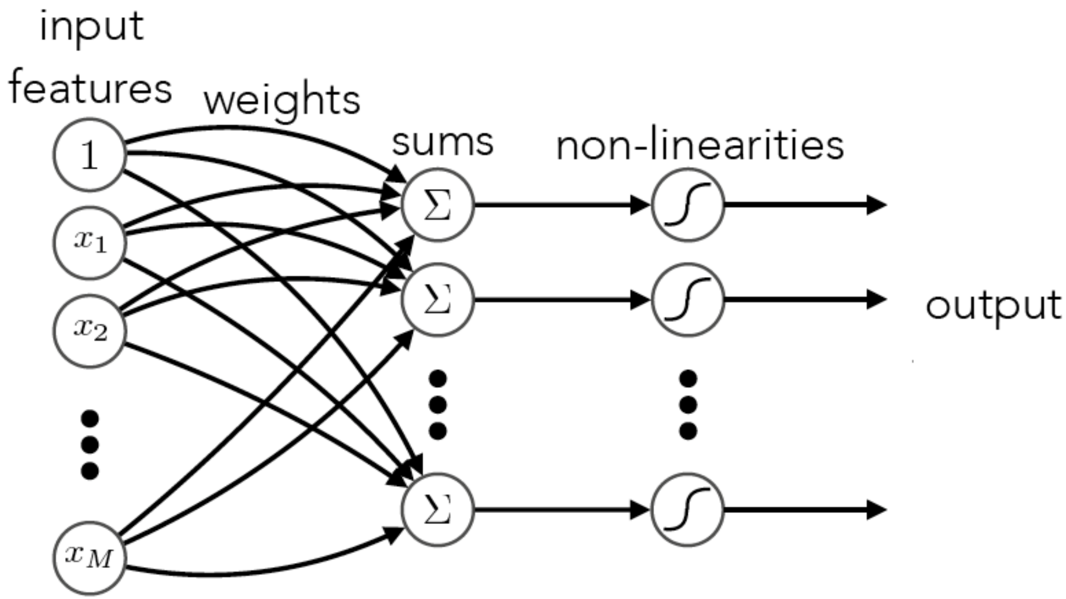
non-linearity

sum

Single layer neural network



Single layer neural network

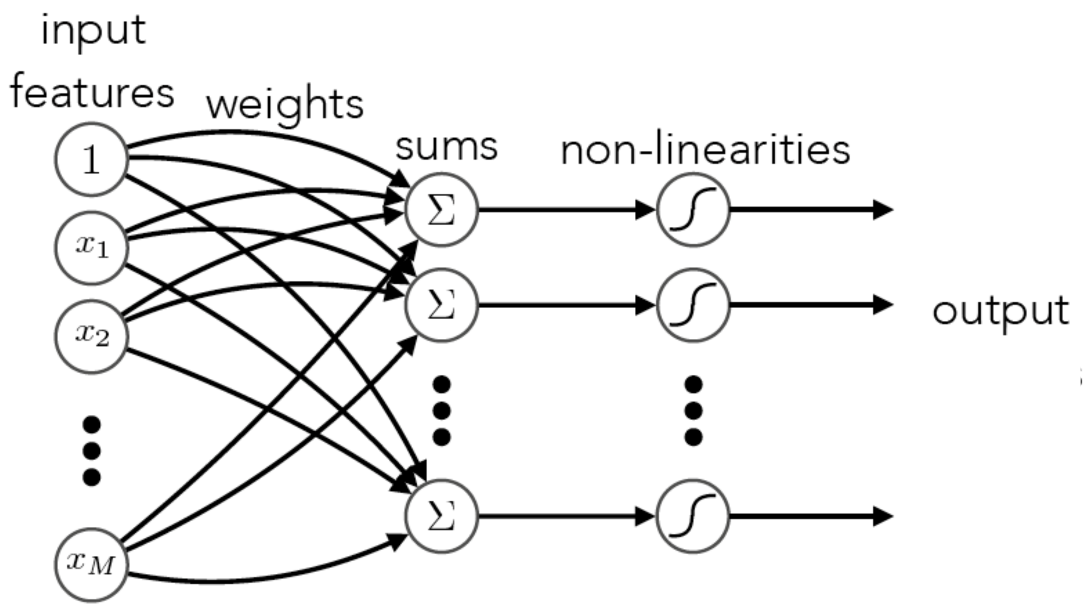


layer: *parallelized weighted sum and non-linearity*

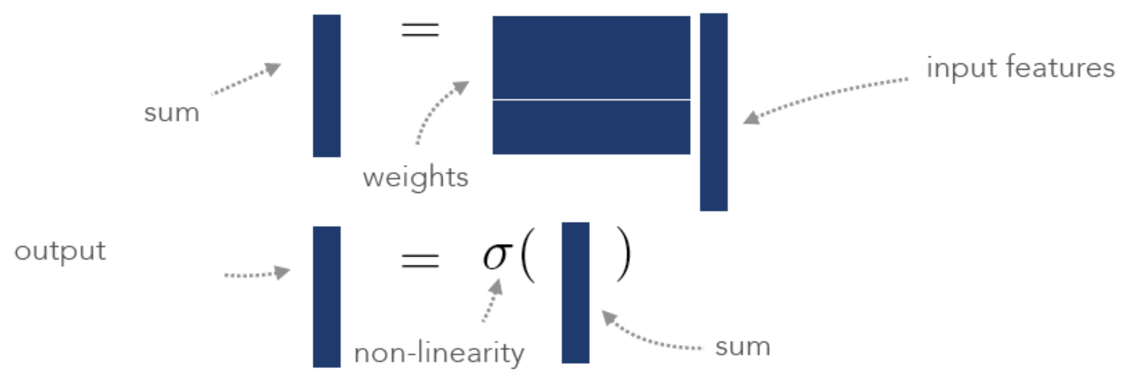
one sum per weight *vector* $s_j = \mathbf{w}_j^\intercal \mathbf{x}$ ⟶ $\mathbf{s} = \mathbf{W}^\intercal \mathbf{x}$ vector of sums from weight *matrix*

$$\mathbf{h} = \sigma(\mathbf{s})$$

Single layer neural network

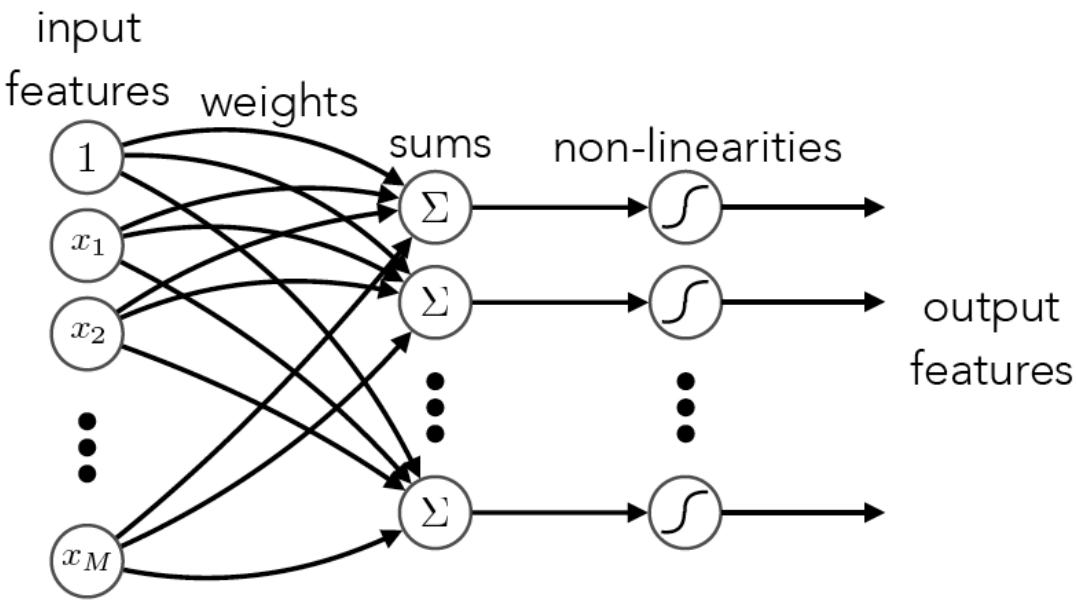


layer: *parallelized weighted sum and non-linearity*



What is the output?

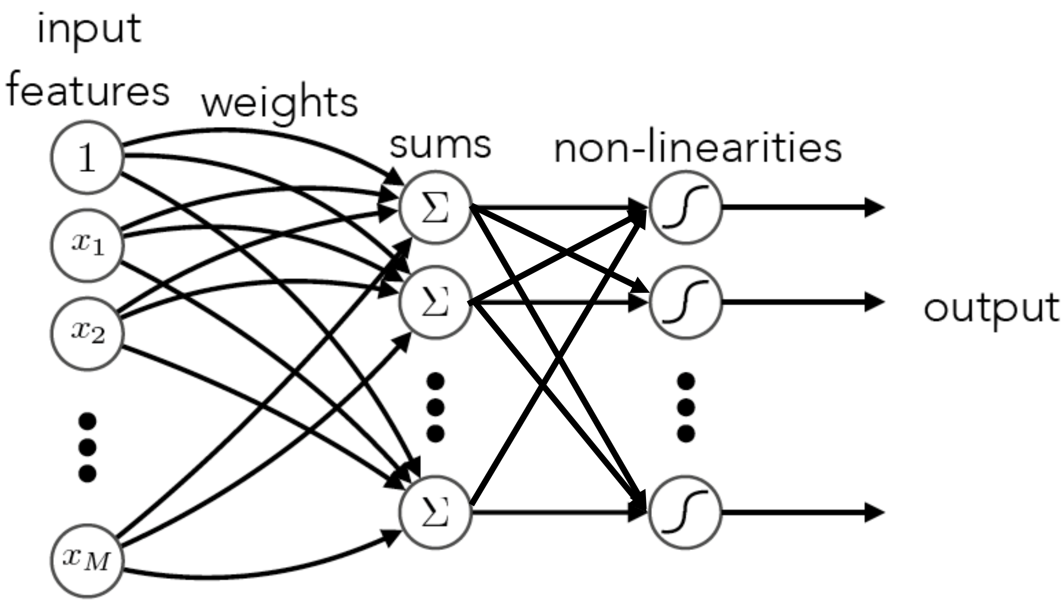
- Element-wise nonlinear functions
  - Independent feature/attribute detectors



$$\mathbf{h} = [h_j] \qquad h_j = \sigma(s_j) = \sigma(\mathbf{w}_j^\intercal \mathbf{x})$$

What is the output?

- Nonlinear functions with vector input
  - Competition between neurons

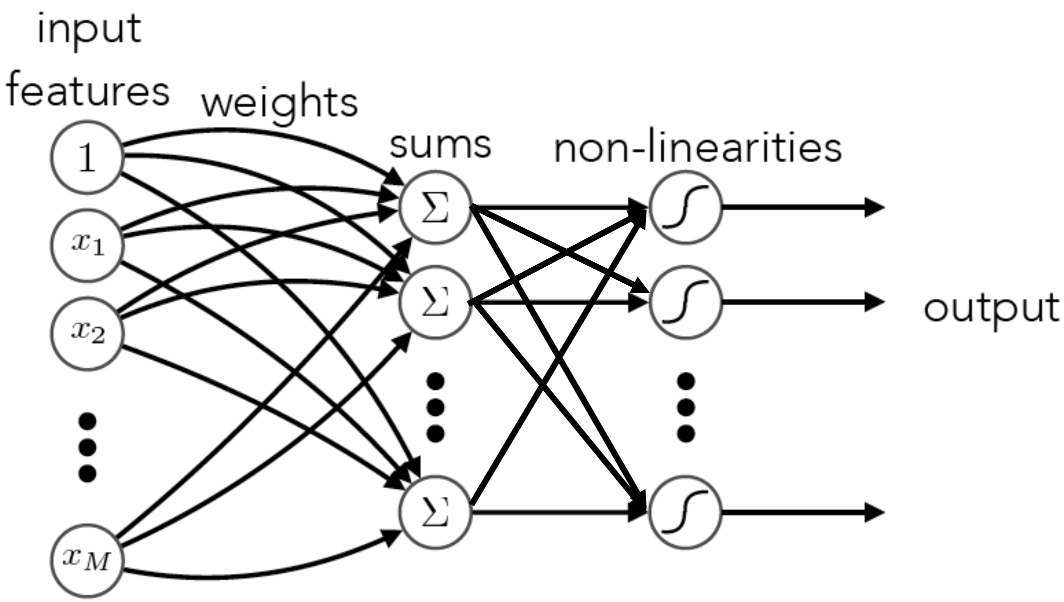


$$\mathbf{h} = [h_j]$$

$$h_j = g(\mathbf{s}) = g(\mathbf{w}_1^\intercal \mathbf{x}, \cdots, \mathbf{w}_m^\intercal \mathbf{x})$$

What is the output?

- Nonlinear functions with vector input
  - Example: Winner-Take-All (WTA)

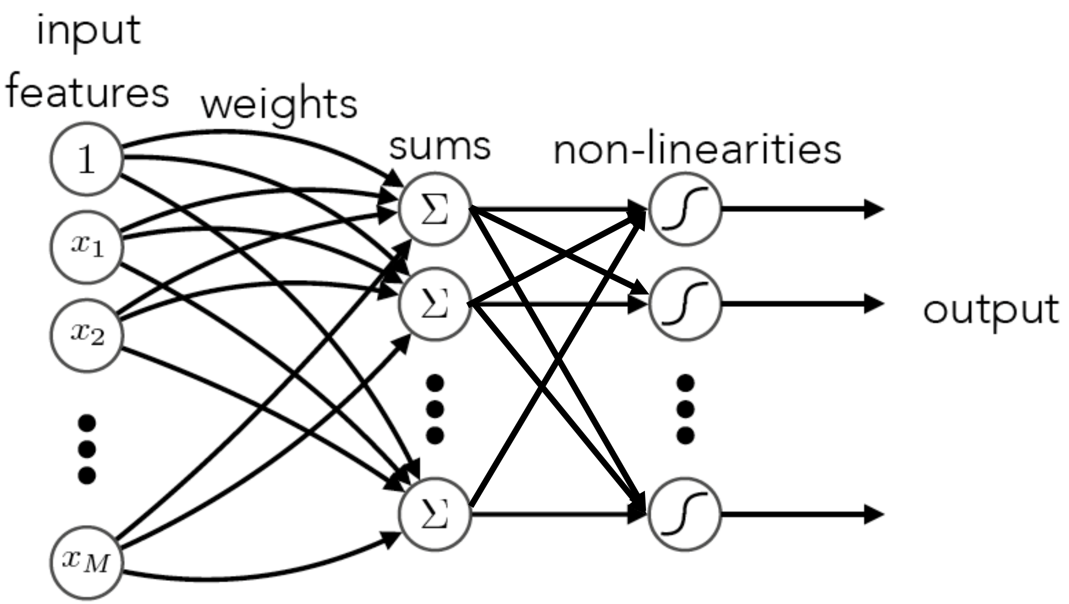


$$\mathbf{h} = [h_j]$$

$$h_j = g(\mathbf{s}) = \begin{cases} 1 & \text{if } j = \arg\max_i \mathbf{w}_i^\intercal \mathbf{x} \\ 0 & \text{if otherwise} \end{cases}$$

A probabilistic perspective

- Change the output nonlinearity



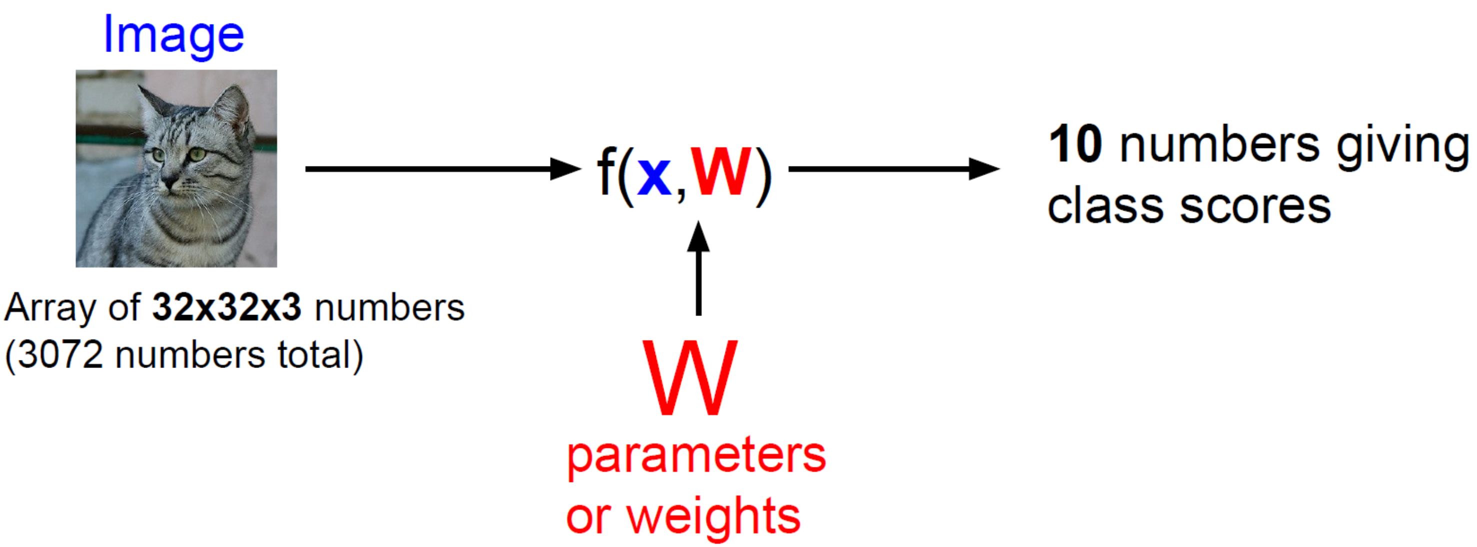
  - From WTA to Softmax function

scores = unnormalized log probabilities of the classes.

$$\boxed{P(Y=k|X=x_i)=\frac{e^{s_k}}{\sum_j e^{s_j}}} \quad \text{where} \quad \boxed{s=f(x_i;W)}$$

Example: Multiclass classification

- CIFAR10 as an example



- The output/prediction: WTA

Multiclass linear classifiers

- Extending linear classifier in binary case

$$\underbrace{\boxed{f(x,W)}}_{10\times1} = \underbrace{\boxed{W}}_{10\times3072}\overbrace{\boxed{x}}^{3072\times1} + \underbrace{\boxed{b}}_{10\times1}$$

Image

Array of **32x32x3** numbers
(3072 numbers total)

$\longrightarrow f(x, W) \longrightarrow$

10 numbers giving
class scores

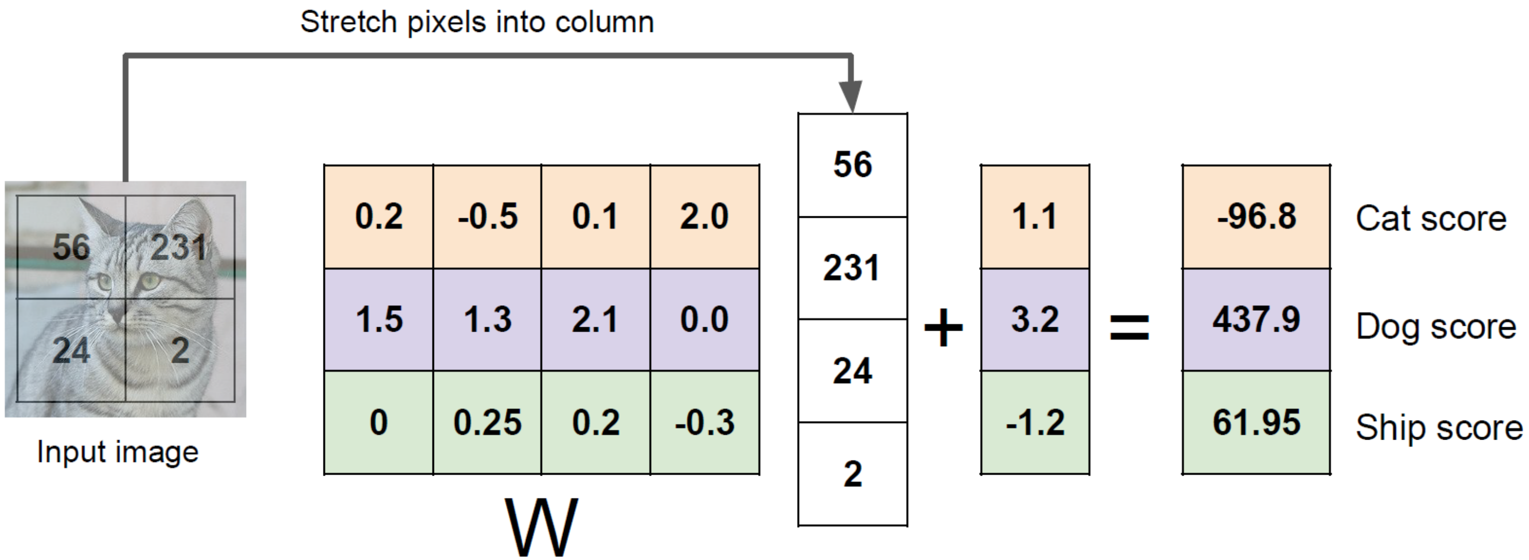
$\mathbf{s}$

W

parameters
or weights

Multiclass linear classifiers

- Example with an image with 4 pixels, and 3 classes (cat/dog/ship)



- The WTA prediction: one-hot encoding of its predicted label

$$y = 1 \Leftrightarrow y = \begin{bmatrix} 1 \\ 0 \\ 0 \end{bmatrix} \qquad y = 2 \Leftrightarrow y = \begin{bmatrix} 0 \\ 1 \\ 0 \end{bmatrix} \qquad y = 3 \Leftrightarrow y = \begin{bmatrix} 0 \\ 0 \\ 1 \end{bmatrix}$$

Probabilistic outputs

scores = unnormalized log probabilities of the classes.

$$P(Y = k | X = x_i) = \frac{e^{s_k}}{\sum_j e^{s_j}}$$ where $$s = f(x_i; W)$$

unnormalized probabilities

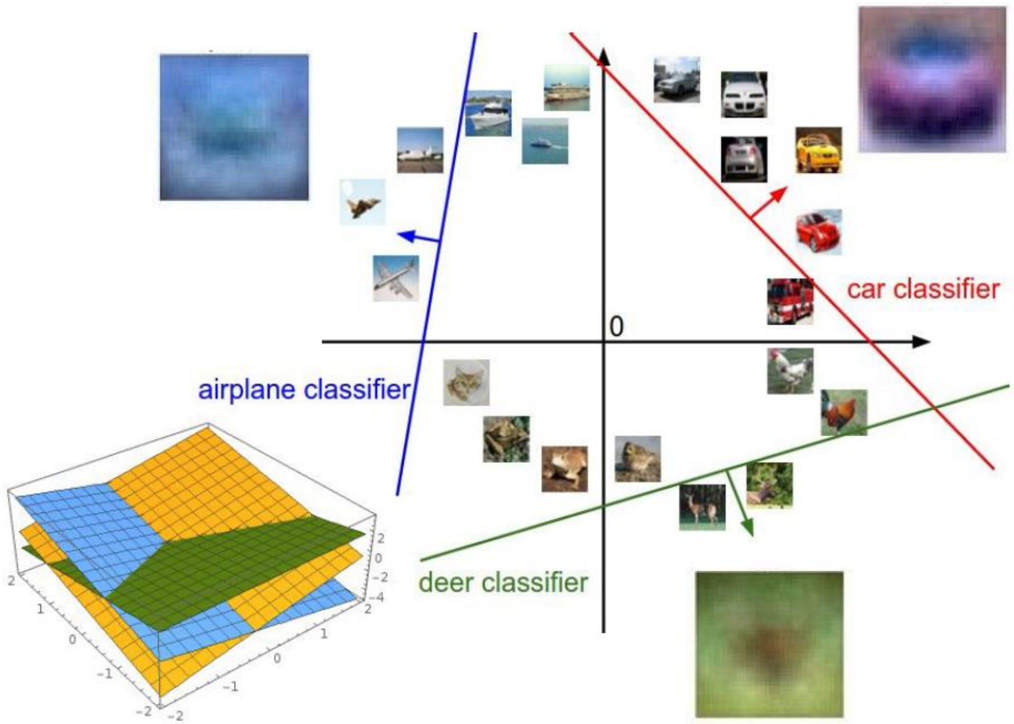
	unnormalized log probabilities	exp →	unnormalized probabilities	normalize →	probabilities
cat	**3.2**		**24.5**		**0.13**
car	5.1		164.0		0.87
frog	-1.7		0.18		0.00

unnormalized log probabilities

probabilities

Interpreting network weights

- What are those weights?



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

Array of **32x32x3** numbers
(3072 numbers total)

How to learn a multiclass classifier?

- Define a loss function and do minimization
  - Given training data $\{(x_i, y_i): 1 \leq i \leq n\}$ i.i.d. from distribution D
  - Find $y = f(x) \in \mathcal{H}$ that minimizes $\hat{L}(f) = \frac{1}{n}\sum_{i=1}^{n} l(f, x_i, y_i)$
  - s.t. the expected loss is small

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

Empirical loss

Learning a multiclass linear classifier

- Design a loss function for multiclass classifiers
  - Perceptron?
    - Yes, see homework
  - Hinge loss
    - The SVM and max-margin
  - Probabilistic formulation
    - Log loss and logistic regression
- Generalization issue
  - Avoid overfitting by regularization
- To be covered next time

Summary

- Supervised learning
  - Linear models
- Artificial neurons
- Single-layer network
  - Multi-class predictions

- Next time …
  - Learning single-layer network
  - Multi-layer neural networks