



# Lecture 2: Basic Artificial Neural Networks

Xuming He  
SIST, ShanghaiTech  
Fall, 2020

# Logistics

- Course project

- ☐ Each team consists of 3~5 members
- ☐ You may make exceptions if you are among top 10% in first 3 quizzes

- Full course schedule on Piazza

- ☐ HW1 out next Monday
- ☐ Tutorial schedule: please vote on Piazza

- TA office hours

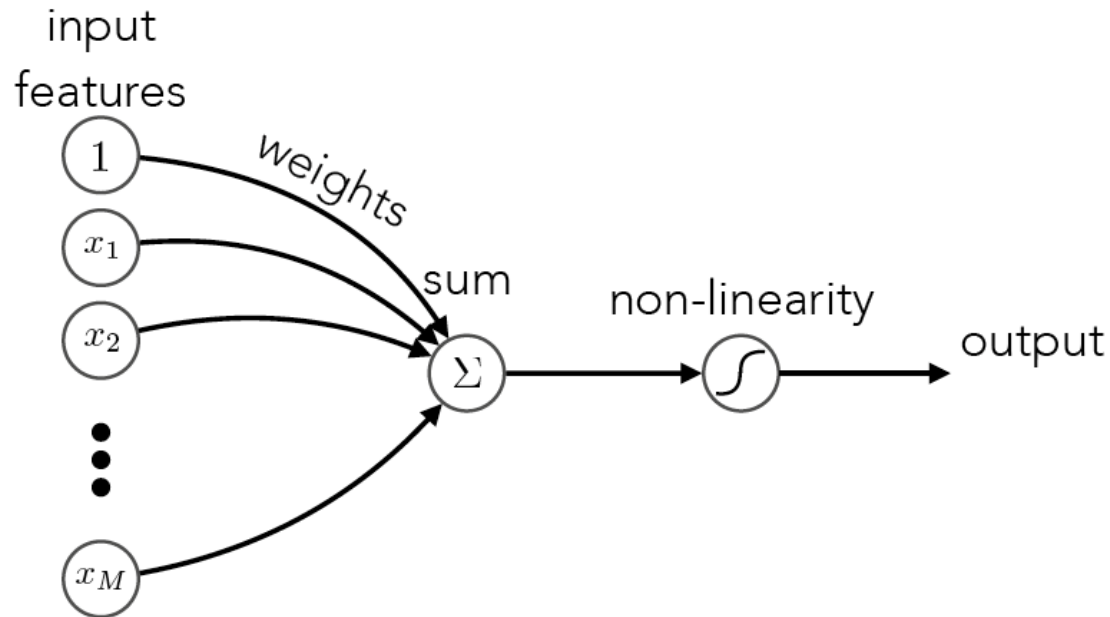
- ☐ See Piazza for detailed schedule and location

# Outline

- Artificial neuron
  - Perceptron algorithm
- Single layer neural networks
  - Network models
  - Example: Logistic Regression
- Multi-layer neural networks
  - Limitations of single layer networks
  - Networks with single hidden layer

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

# Mathematical model of a neuron



**artificial neuron:** *weighted sum and non-linearity*

$$s = \underset{\substack{\text{bias} \\ \uparrow}}{b} + \underset{\substack{\text{weights} \\ \uparrow}}{w_1}x_1 + w_2x_2 + \cdots + w_Mx_M = \mathbf{w}^T \mathbf{x}$$

input features

sum

$$h = g(s)$$

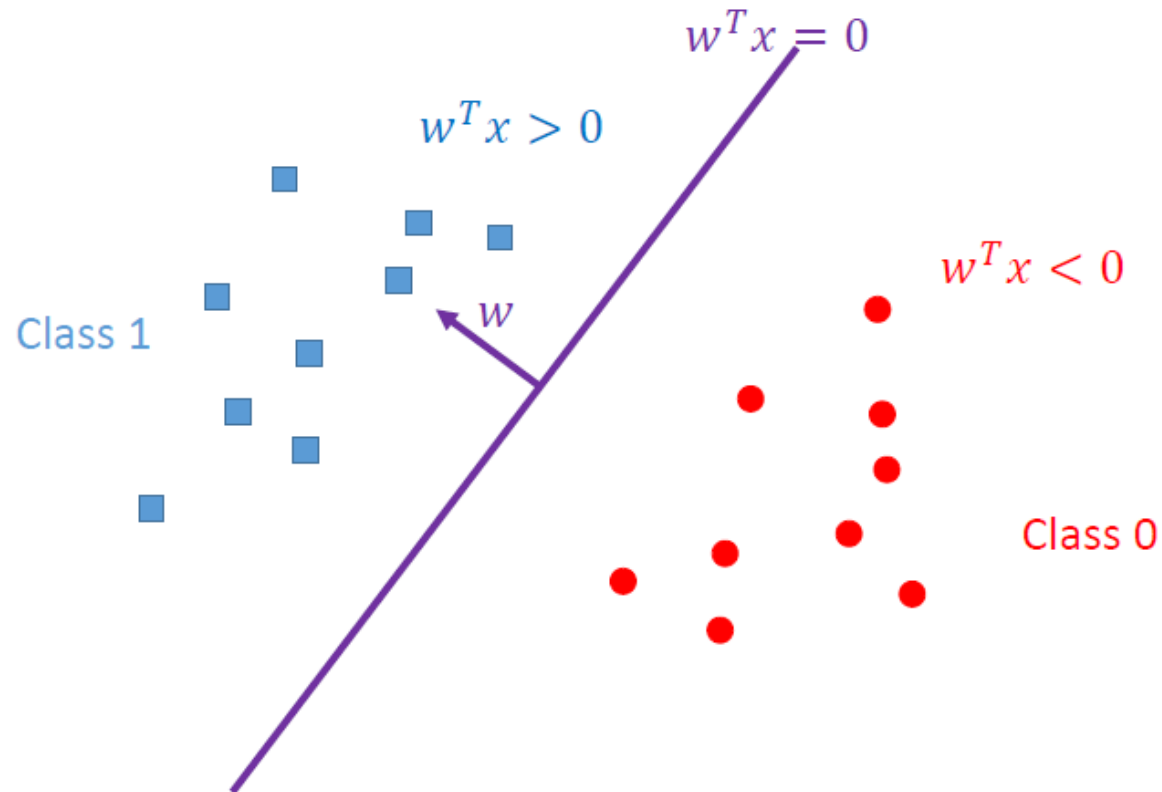
output

non-linearity

sum

# 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

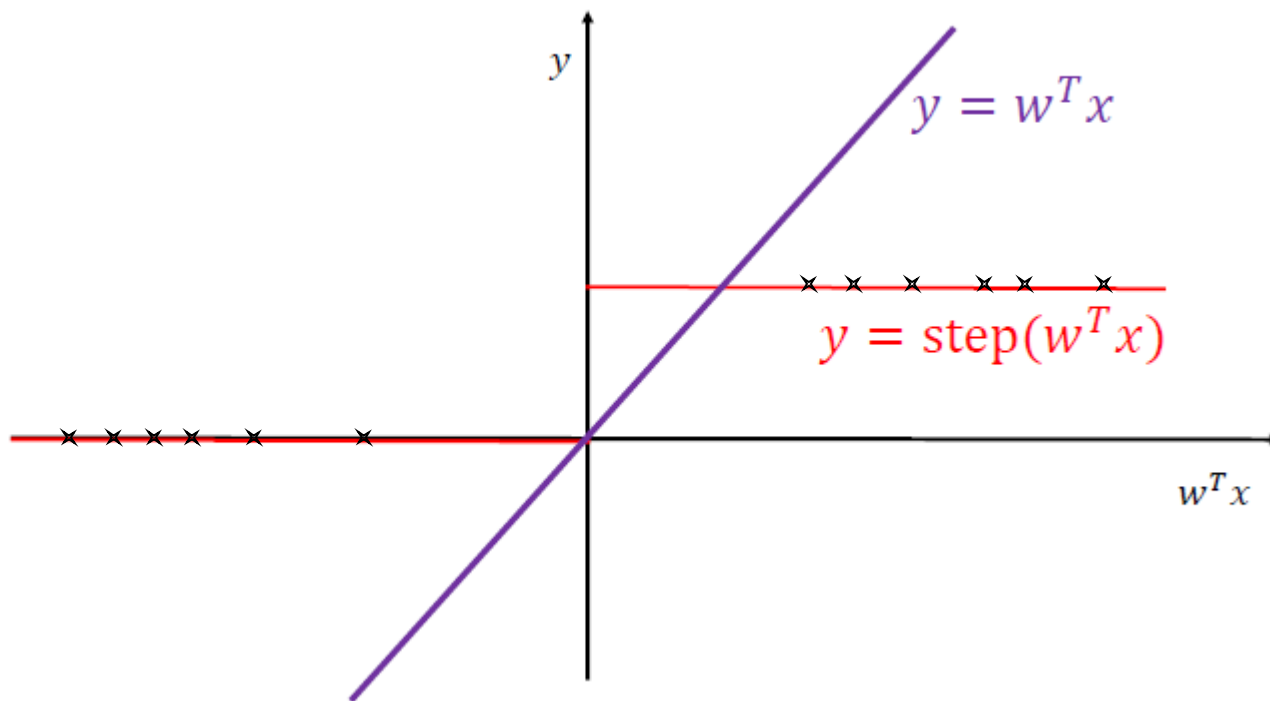
## ■ Learning problem: simple approach

- 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

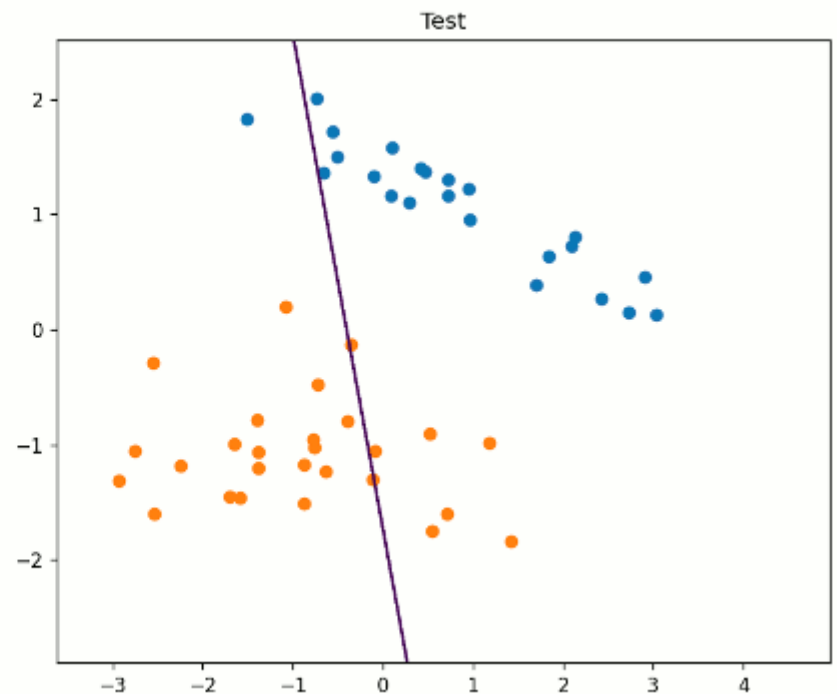
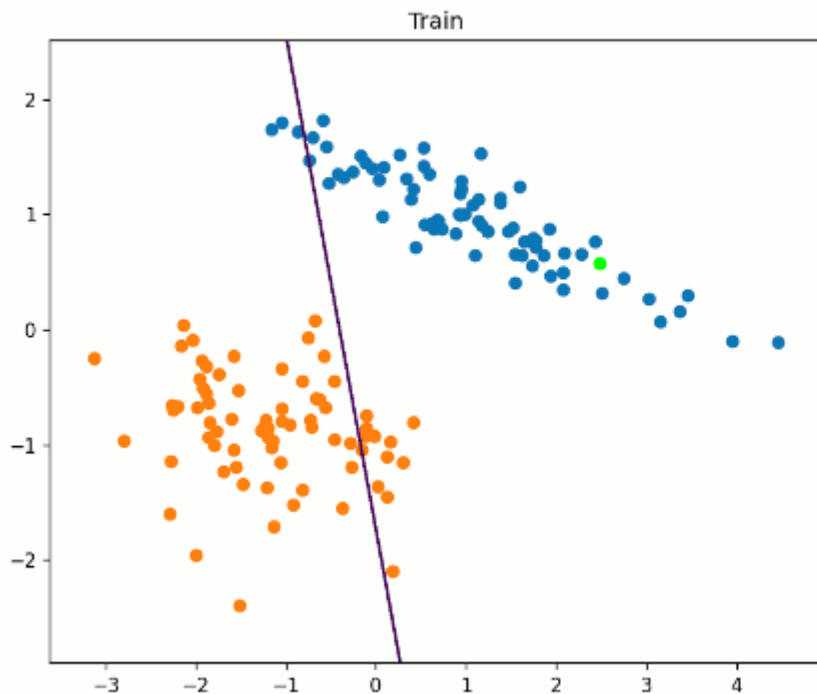




# Perceptron algorithm

- Learn a single neuron for binary classification

Iteration: 1/2; Point: 1/150



<https://towardsdatascience.com/perceptron-explanation-implementation-and-a-visual-example-3c8e76b4e2d1>

# 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  $\mathbf{x}_i$  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

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

# 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

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

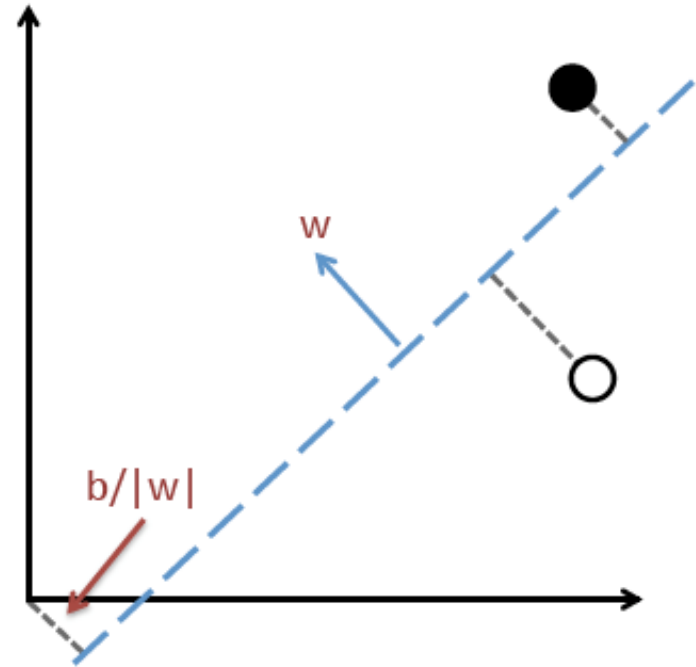
- Then Perceptron makes at most  $\left(\frac{1}{\gamma}\right)^2$  mistakes

# Hyperplane Distance

- Line is a 1D, Plane is 2D
- Hyperplane is many D
  - Includes Line and Plane
- Defined by  $(w, b)$

- Distance: 
$$\frac{|w^T x - b|}{\|w\|}$$

- Signed Distance: 
$$\frac{w^T x - b}{\|w\|}$$



Linear Model = un-normalized signed distance!

# 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 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 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

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

# Perceptron algorithm

## ■ What loss function is minimized?

- Hypothesis:  $y = \text{sign}(w^T x)$

- 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

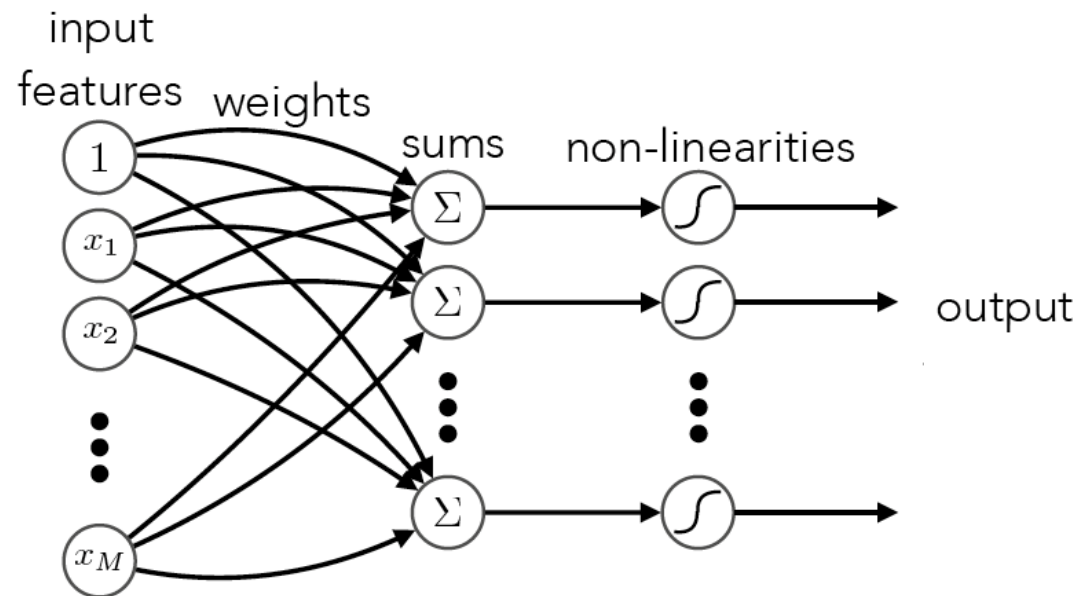
$$w_{t+1} = w_t + y_t x_t = w_t - x$$

# Outline

- Artificial neuron
  - Perceptron algorithm
- Single layer neural networks
  - Network models
  - Example: Logistic Regression
- Multi-layer neural networks
  - Limitations of single layer networks
  - Networks with single hidden layer

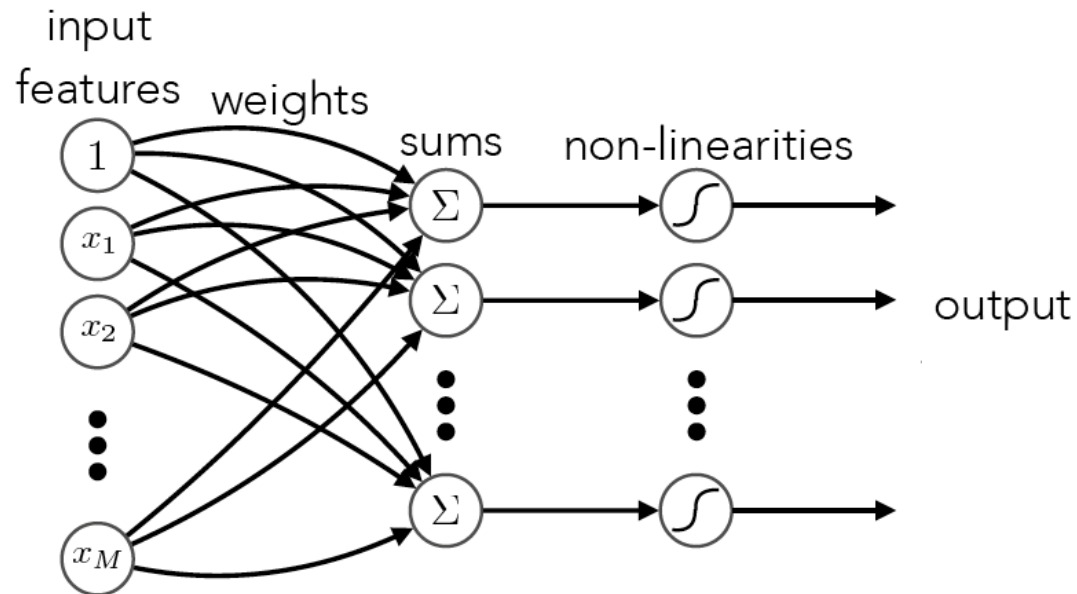
*Acknowledgement: Hugo Larochelle's, Mehryar Mohri@NYU's & Yingyu Liang@Princeton's course notes*

# Single layer neural network





# Single layer neural network

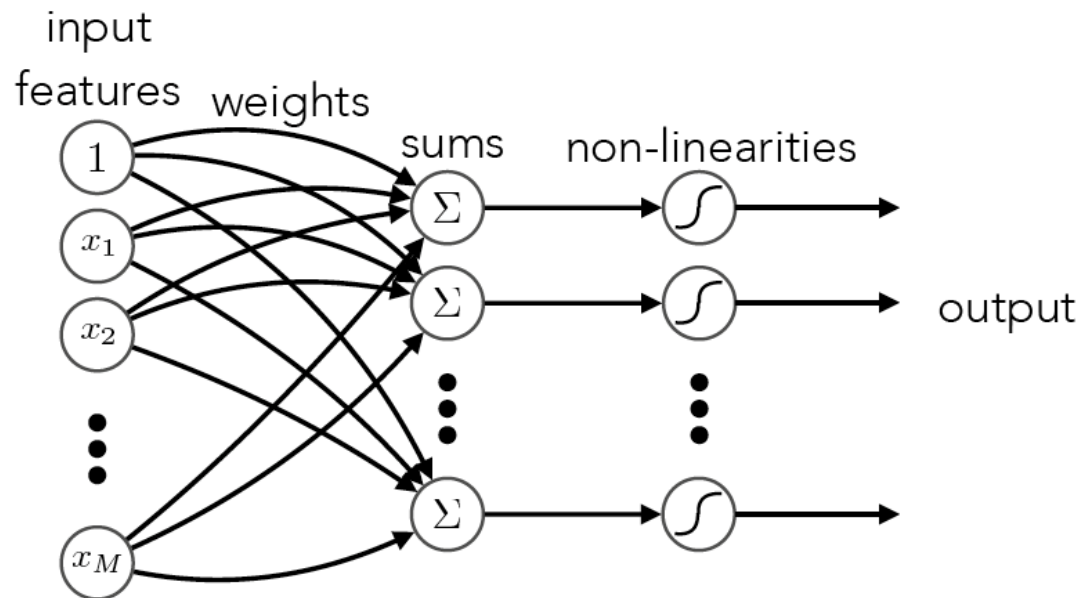


**layer:** *parallelized weighted sum and non-linearity*

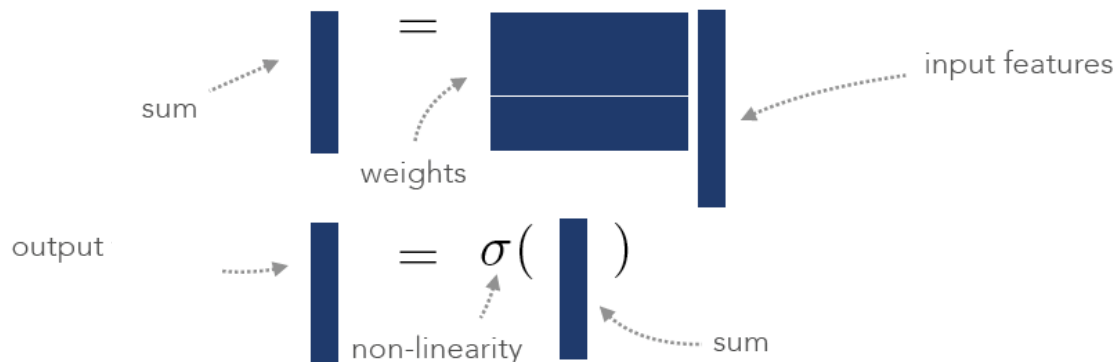
$$\begin{array}{c} \text{one sum} \\ \text{per weight vector} \end{array} s_j = \mathbf{w}_j^T \mathbf{x} \longrightarrow \mathbf{s} = \mathbf{W}^T \mathbf{x} \begin{array}{c} \text{vector of sums} \\ \text{from weight matrix} \end{array}$$

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

# Single layer neural network

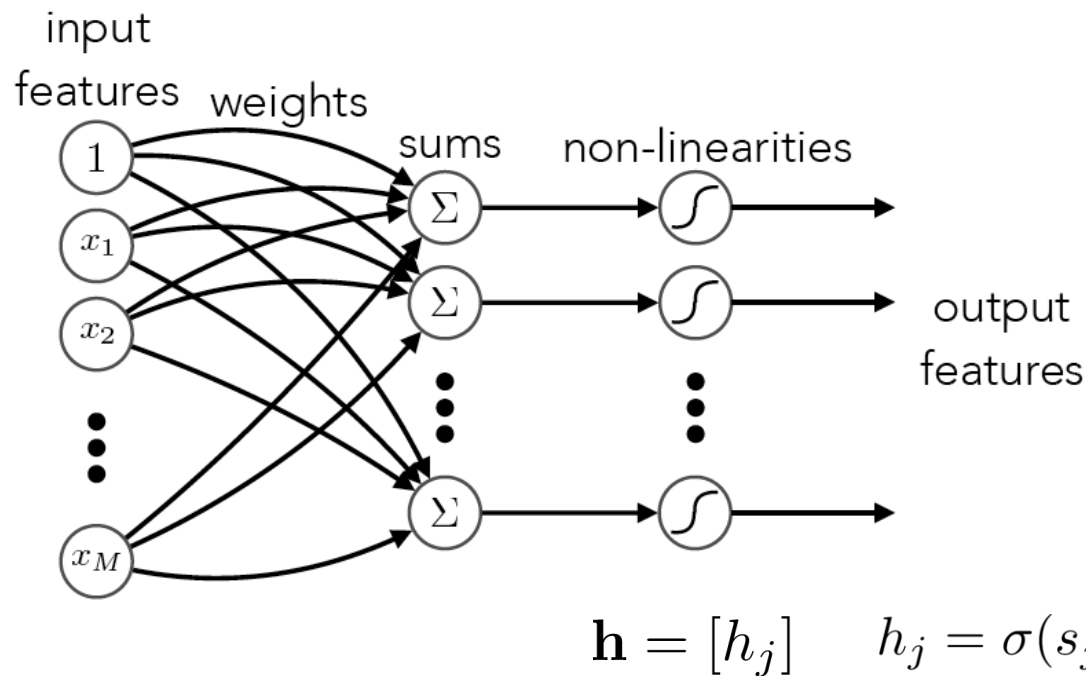


**layer:** *parallelized weighted sum and non-linearity*



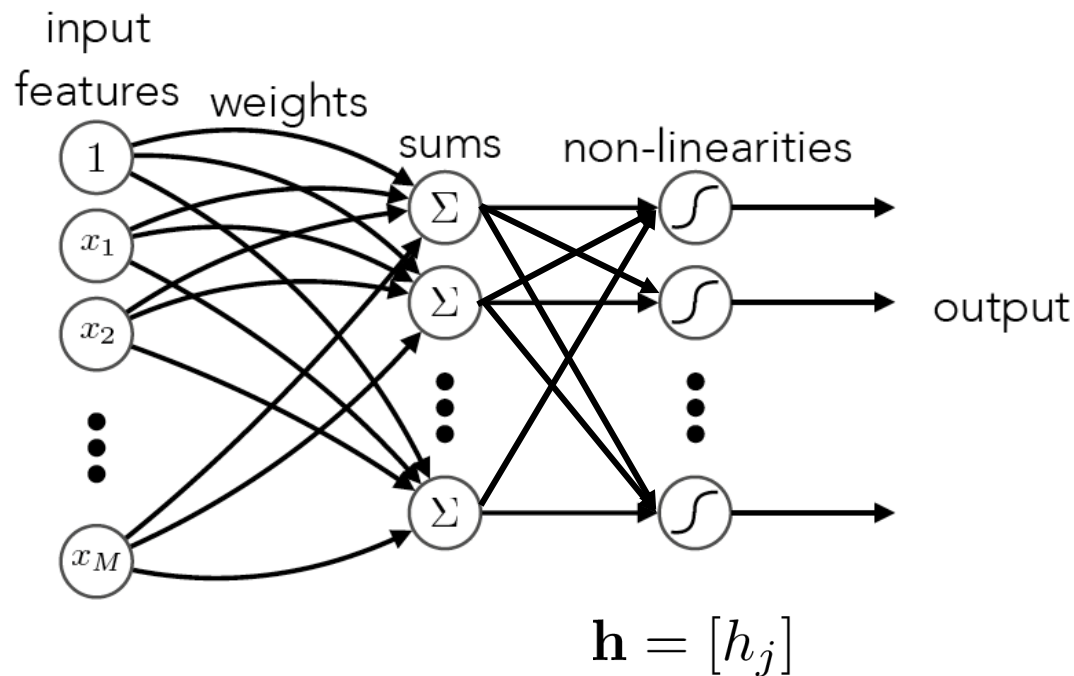
# What is the output?

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



# What is the output?

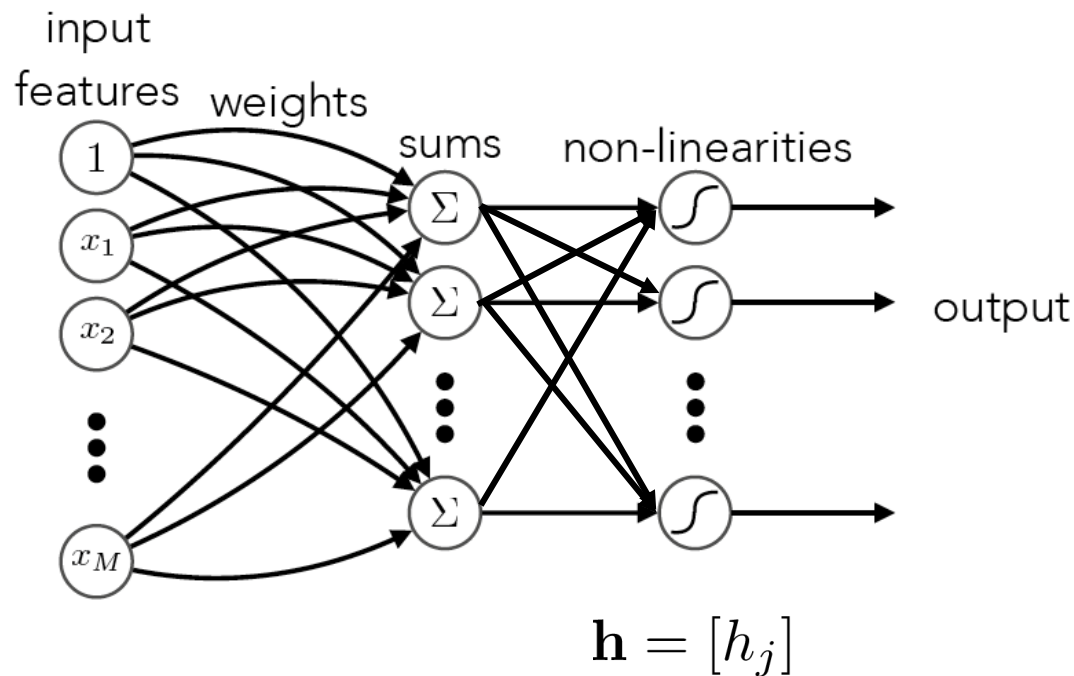
- Nonlinear functions with vector input
  - Competition between neurons



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

# What is the output?

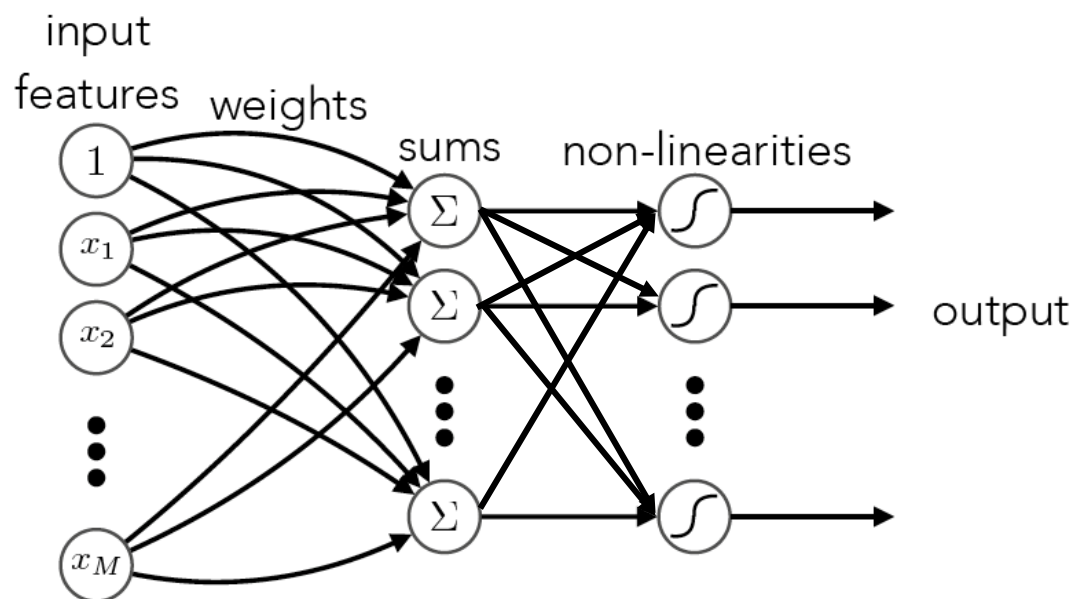
- Nonlinear functions with vector input
  - Example: Winner-Take-All (WTA)



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

# A probabilistic perspective

- Change the output nonlinearity



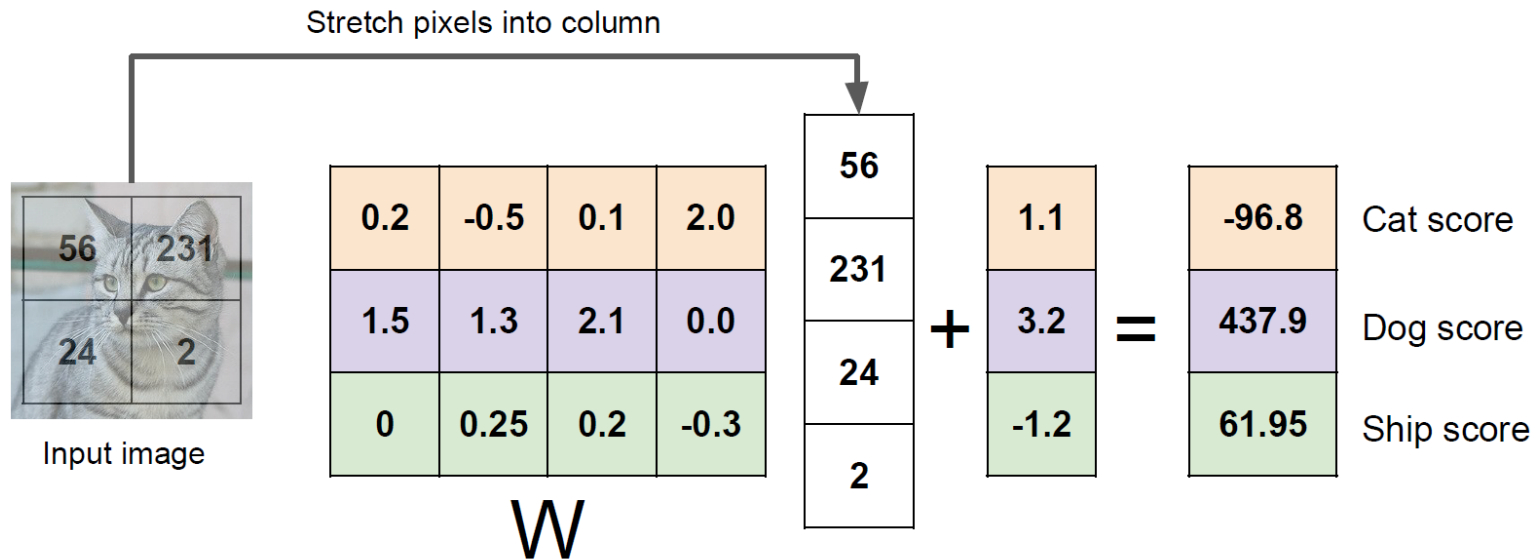
- From WTA to Softmax function

**scores = unnormalized log probabilities of the classes.**

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

# 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

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

# Probabilistic outputs

scores = unnormalized log probabilities of the classes.

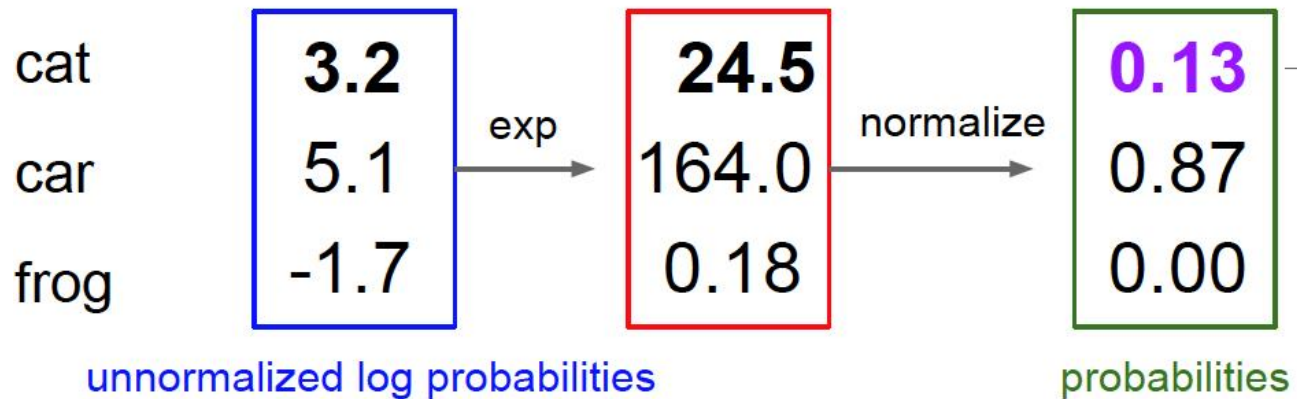


$$P(Y = k|X = x_i) = \frac{e^{s_k}}{\sum_j e^{s_j}}$$

where

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

unnormalized probabilities



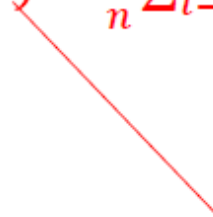


# How to learn a multiclass classifier?

## ■ 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 (see CS231n)
  - Probabilistic formulation
    - Log loss and logistic regression
- Generalization issue
  - Avoid overfitting by regularization

# Example: Logistic Regression

- Learning loss: negative log likelihood

**scores = unnormalized log probabilities of the classes.**

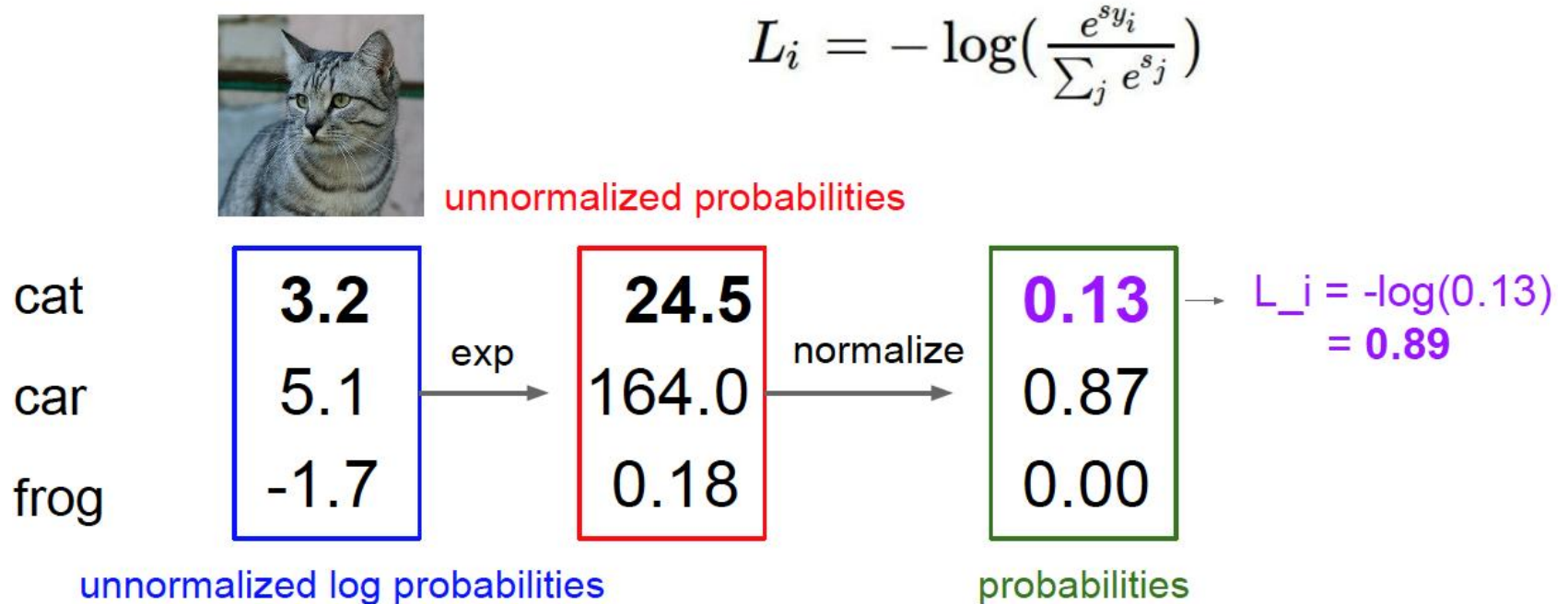
$$P(Y = k|X = x_i) = \frac{e^{s_k}}{\sum_j e^{s_j}} \quad \text{where} \quad s = f(x_i; W)$$

Want to maximize the log likelihood, or (for a loss function) to minimize the negative log likelihood of the correct class:

$$L_i = -\log P(Y = y_i|X = x_i)$$

# Logistic Regression

- Learning loss: example



# Logistic Regression

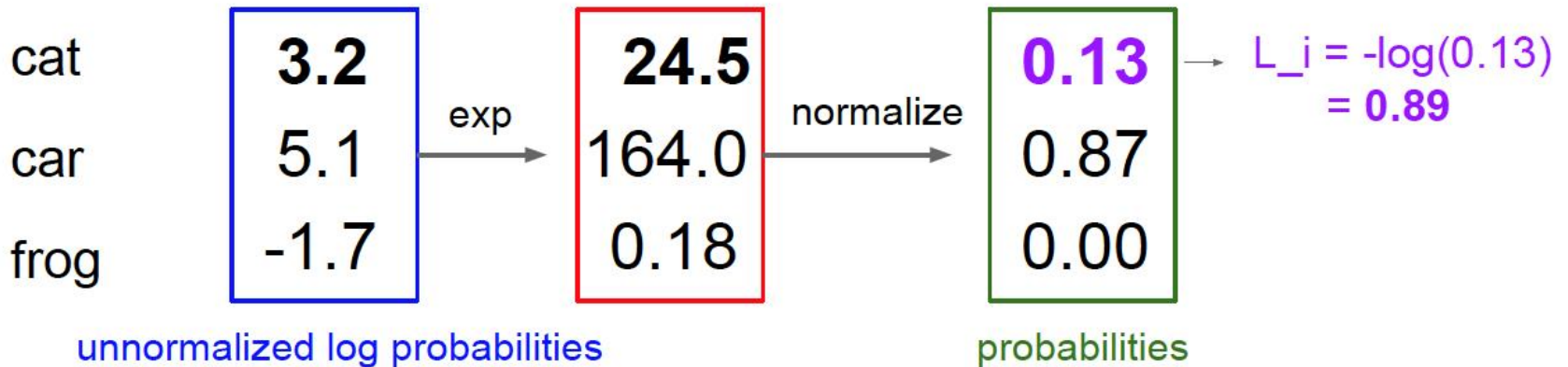
- Learning loss: questions



$$L_i = -\log\left(\frac{e^{sy_i}}{\sum_j e^{s_j}}\right)$$

unnormalized probabilities

Q: What is the min/max possible loss  $L_i$ ?



# Logistic Regression

- Learning loss: questions



$$L_i = -\log\left(\frac{e^{sy_i}}{\sum_j e^{s_j}}\right)$$

unnormalized probabilities

Q2: Usually at initialization  $W$  is small so all  $s \approx 0$ . What is the loss?

cat  
car  
frog

3.2

5.1

-1.7

exp

24.5

164.0

0.18

normalize

0.13

0.87

0.00

→  $L_i = -\log(0.13)$   
 $= 0.89$

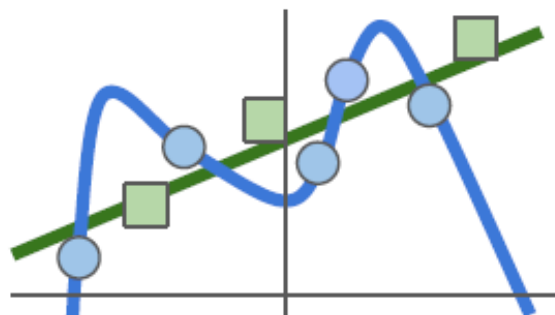
unnormalized log probabilities

probabilities

# Learning with regularization

- Constraints on hypothesis space
  - Similar to Linear Regression

$$L(W) = \underbrace{\frac{1}{N} \sum_{i=1}^N L_i(f(x_i, W), y_i)}_{\text{Data loss: Model predictions should match training data}} + \underbrace{\lambda R(W)}_{\text{Regularization: Model should be "simple", so it works on test data}}$$



# Learning with regularization

## ■ Regularization terms

In common use:

**L2 regularization**  $R(W) = \sum_k \sum_l W_{k,l}^2$

**L1 regularization**  $R(W) = \sum_k \sum_l |W_{k,l}|$

**Elastic net (L1 + L2)**  $R(W) = \sum_k \sum_l \beta W_{k,l}^2 + |W_{k,l}|$

**Max norm regularization** (might see later)

## ■ Priors on the weights

- Bayesian: integrating out weights
- Empirical: computing MAP estimate of  $W$



# L1 vs L2 regularization



<https://www.youtube.com/watch?v=jEVh0uheCPk>

# L1 vs L2 regularization

## ■ Sparsity

$$x = [1, 1, 1, 1]$$

$$w_1 = [1, 0, 0, 0]$$

$$w_2 = [0.25, 0.25, 0.25, 0.25]$$

$$w_3 = [0.5, 0.5, 0, 0]$$

$$f(x) = w^\top x$$

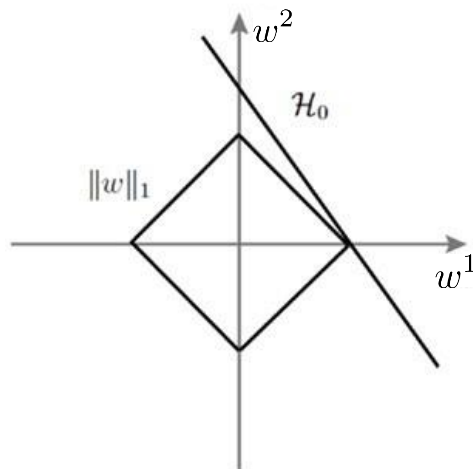
$$w_1^\top x = w_2^\top x = w_3^\top x$$

$$\|w_1\|^2 = |w_1| = 1$$

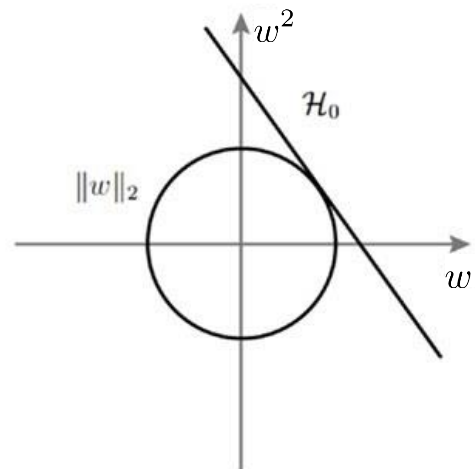
$$\|w_2\|^2 = 4/16 = 1/4, |w_2| = 1$$

$$\|w_3\|^2 = 2/4 = 1/2, |w_3| = 1$$

**A** L1 regularization



**B** L2 regularization



# Optimization: gradient descent

## ■ Gradient descent

```
# Vanilla Gradient Descent
```

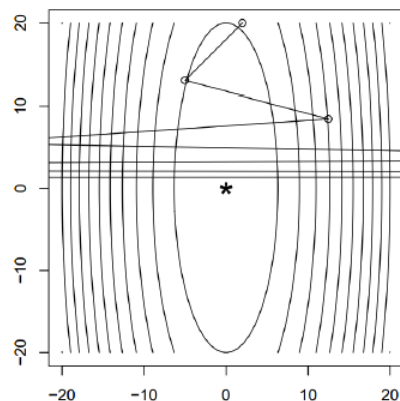
```
while True:
```

```
    weights_grad = evaluate_gradient(loss_fun, data, weights)
```

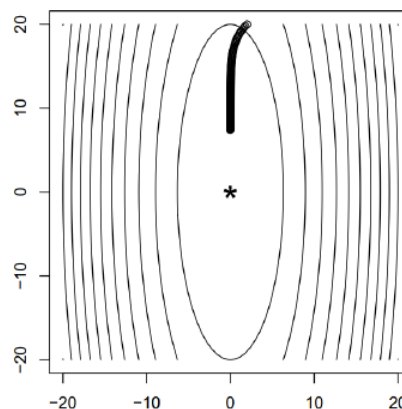
```
    weights += - step_size * weights_grad # perform parameter update
```

## ■ Learning rate matters

$\eta_t = t$ , it is too big



too small  $\eta_t$ , after 100 iterations



# Optimization: gradient descent

## ■ Stochastic gradient descent

$$L(W) = \frac{1}{N} \sum_{i=1}^N L_i(x_i, y_i, W) + \lambda R(W)$$

Full sum expensive  
when N is large!

$$\nabla_W L(W) = \frac{1}{N} \sum_{i=1}^N \nabla_W L_i(x_i, y_i, W) + \lambda \nabla_W R(W)$$

Approximate sum  
using a **minibatch** of  
examples  
32 / 64 / 128 common

```
# Vanilla Minibatch Gradient Descent
```

```
while True:
```

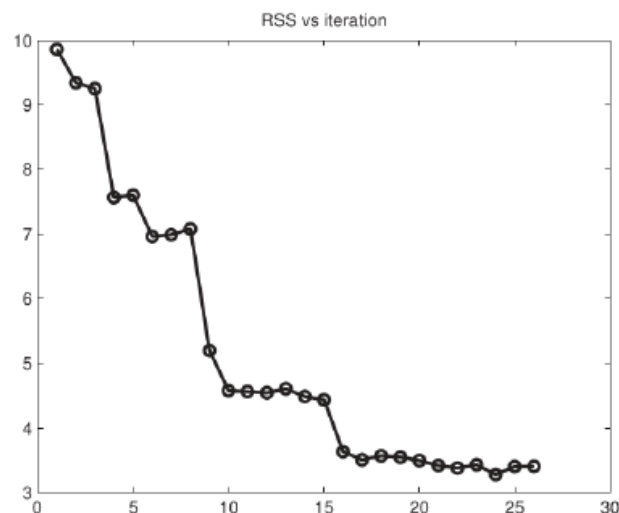
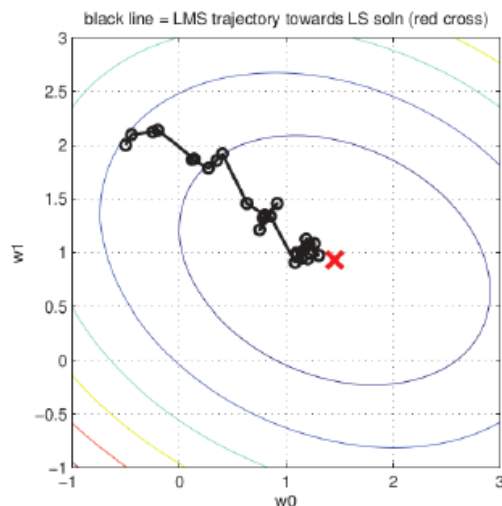
```
    data_batch = sample_training_data(data, 256) # sample 256 examples
```

```
    weights_grad = evaluate_gradient(loss_fun, data_batch, weights)
```

```
    weights += - step_size * weights_grad # perform parameter update
```

# Optimization: gradient descent

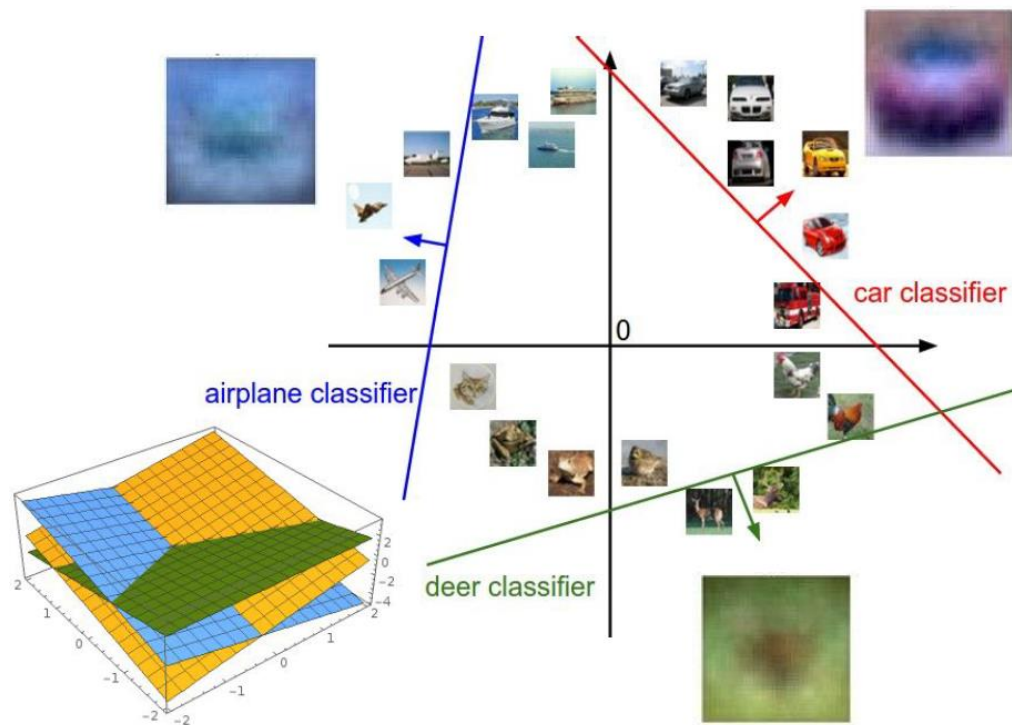
## ■ Stochastic gradient descent



- ▶ the objective does not always decrease for each step
- ▶ comparing to GD, SGD needs more steps, but each step is cheaper
- ▶ mini-batch, say pick up 100 samples and do average, may accelerate the convergence

# Interpreting network weights

- What are those weights?



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



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

# Summary

- Artificial neurons
- Single-layer network
- Next time
  - Multi-layer neural networks
  - Computation in neural networks