Lecture 3: Basic Neural Networks: from artificial neurons to neural networks

Xuming He
SIST, ShanghaiTech
Fall, 2019

Outline

- Perceptron
  - Learning as iterative optimization
  - Loss function
- Single layer neural networks
  - Network models
  - Example: Logistic Regression
  - Learning by (sub-)gradient descent

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 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

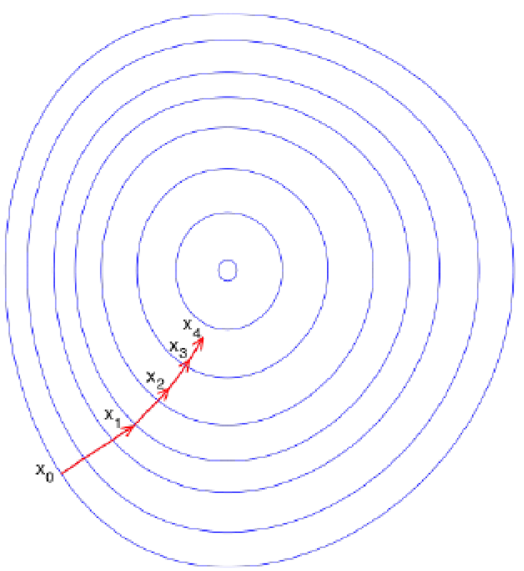
    $$w^{(t+1)} = w^{(t)} - \eta_t \cdot \nabla \mathrm{L}\left(w^{(t)}\right)$$

    until stop
  - η_t is the learning rate, and

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

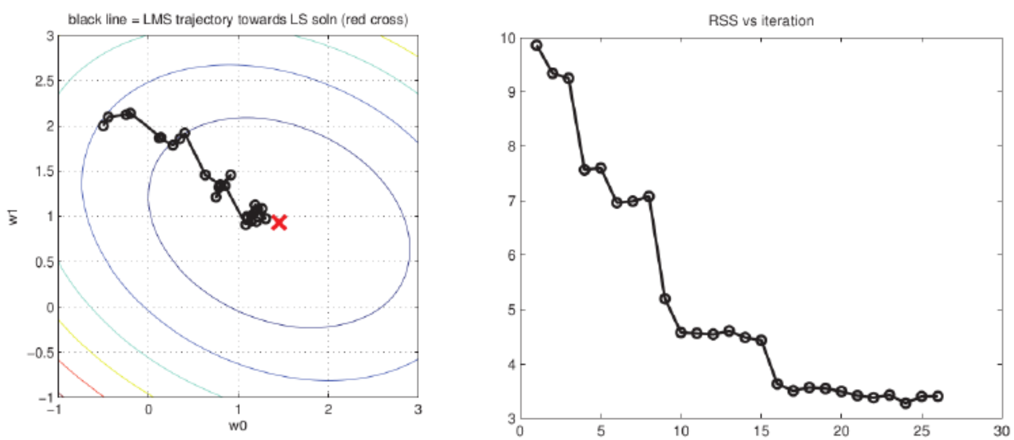
  - How to stop? $\|w^{(t+1)} - w^{(t)}\| \leq \epsilon$ or $\|\nabla \mathrm{L}(w^{(t)})\| \leq \epsilon$

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

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

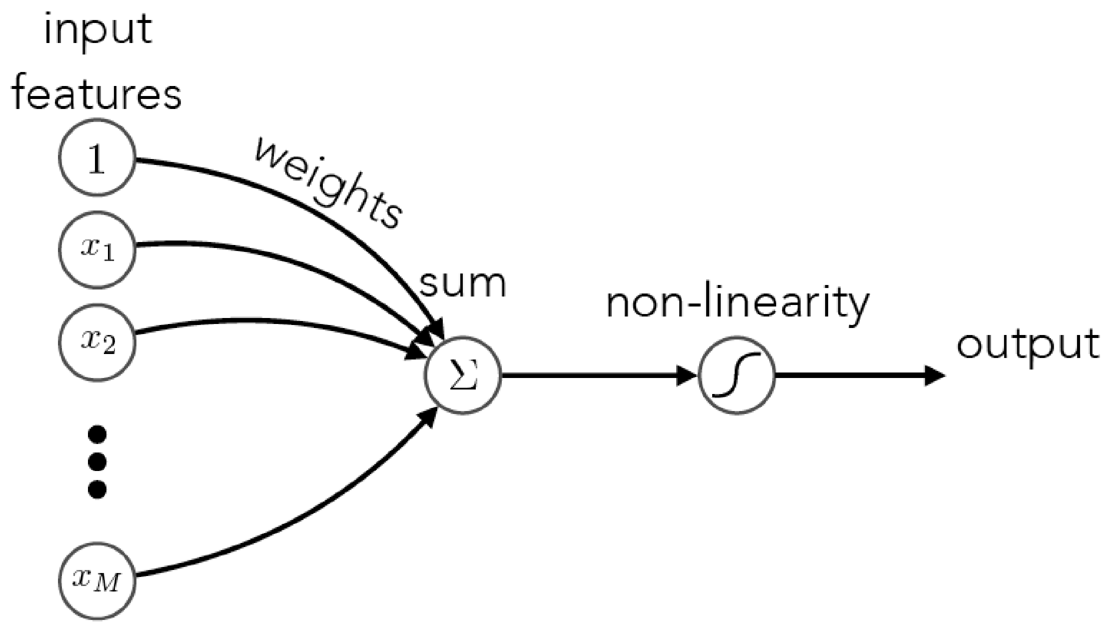
Summary: Artificial Neuron

- Representation
  - Linear inner-product + nonlinear activation function
  - Pattern matching
- Inference
  - Binary classification
- Learning
  - Perceptron: iterative optimization based on SGD

Outline

- Perceptron
  - Learning as iterative optimization
  - Loss function
- Single layer neural networks
  - Network models
  - Example: Logistic Regression
  - Learning 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

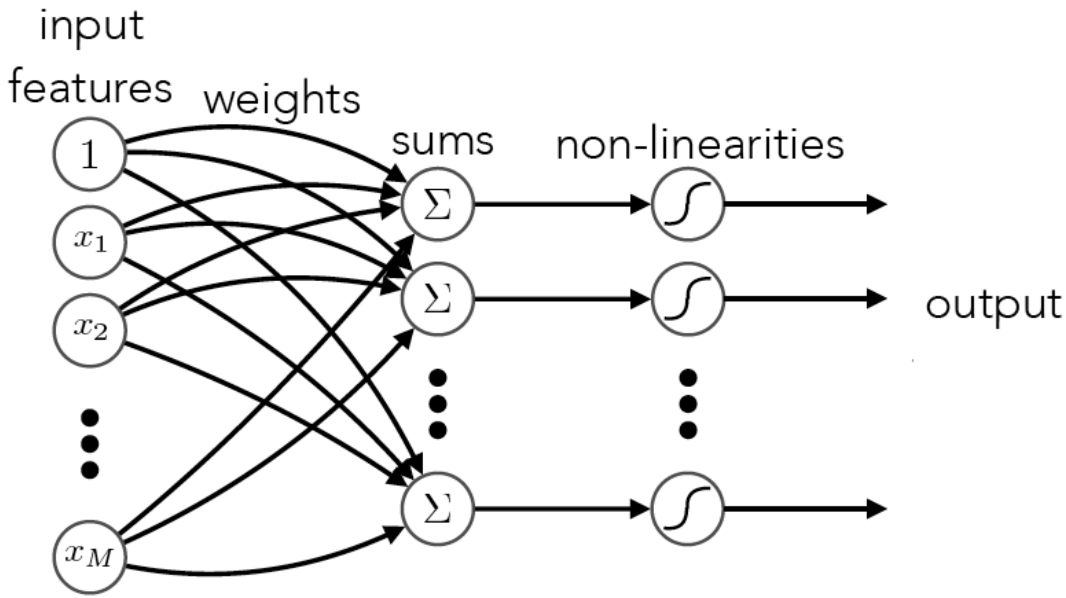
$$h = g(s)$$

output

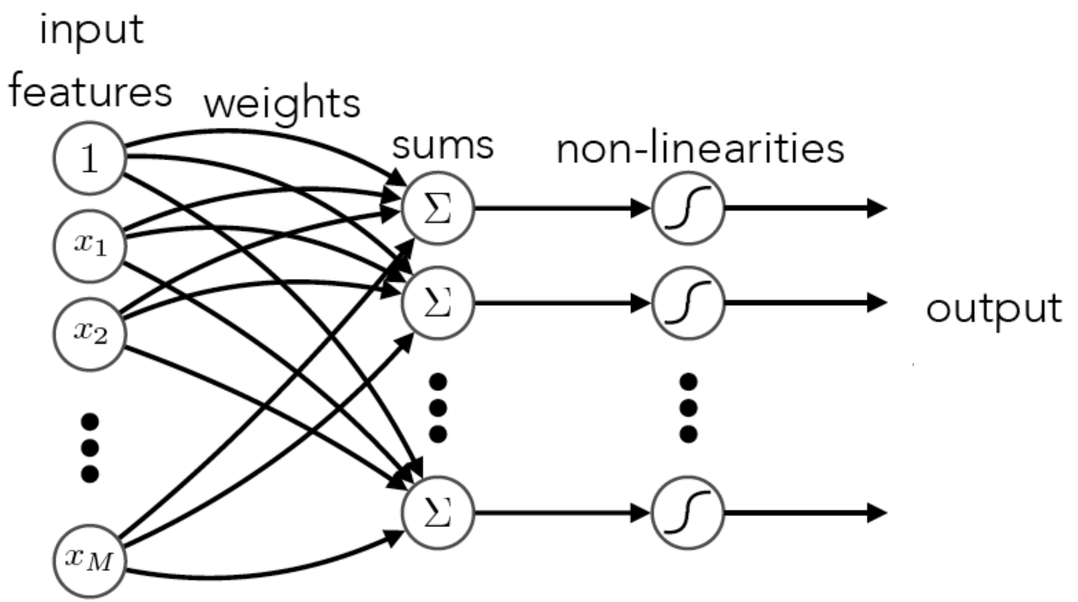
non-linearity

sum

Single layer neural network



Single layer neural network

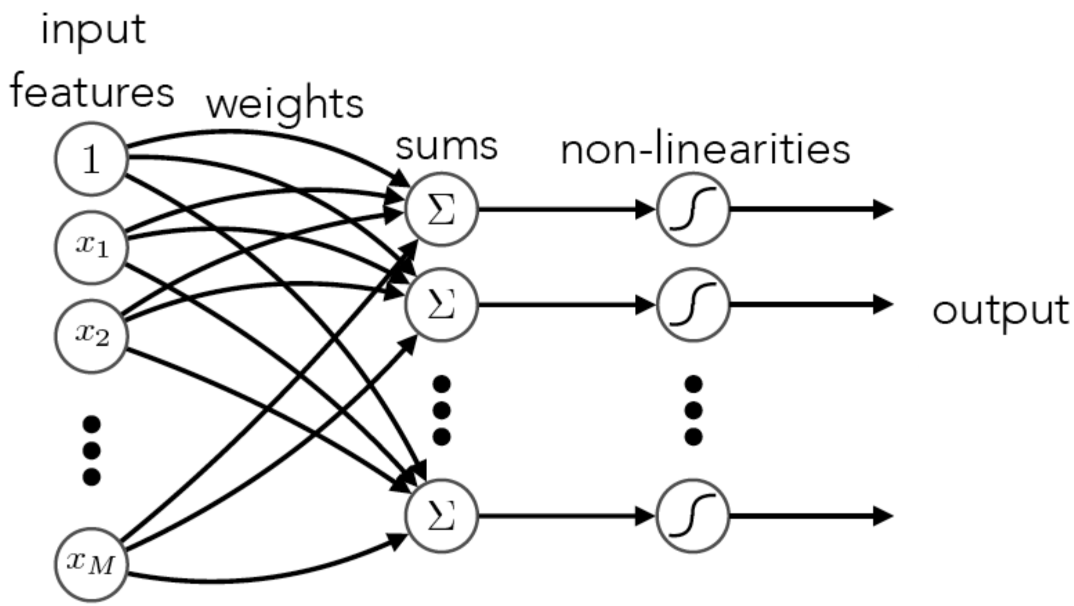


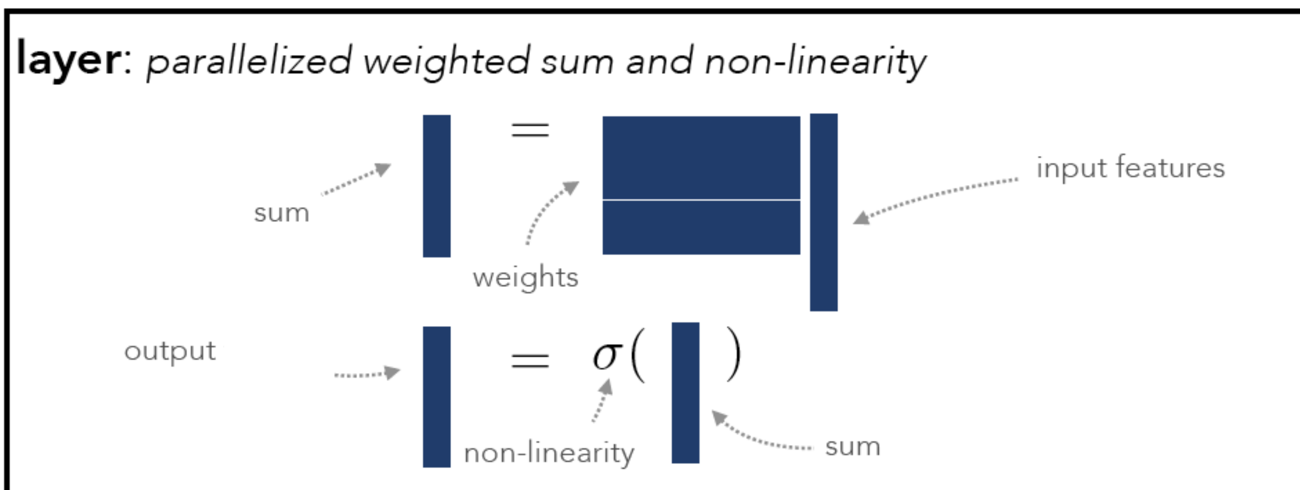
layer: *parallelized weighted sum and non-linearity*

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

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

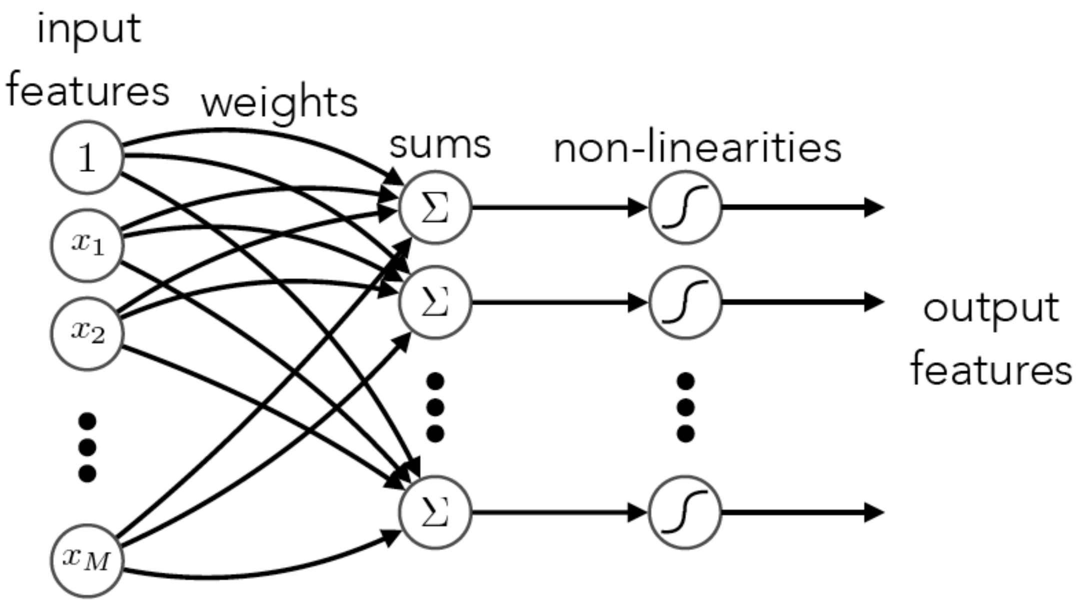
Single layer neural network





What is the output?

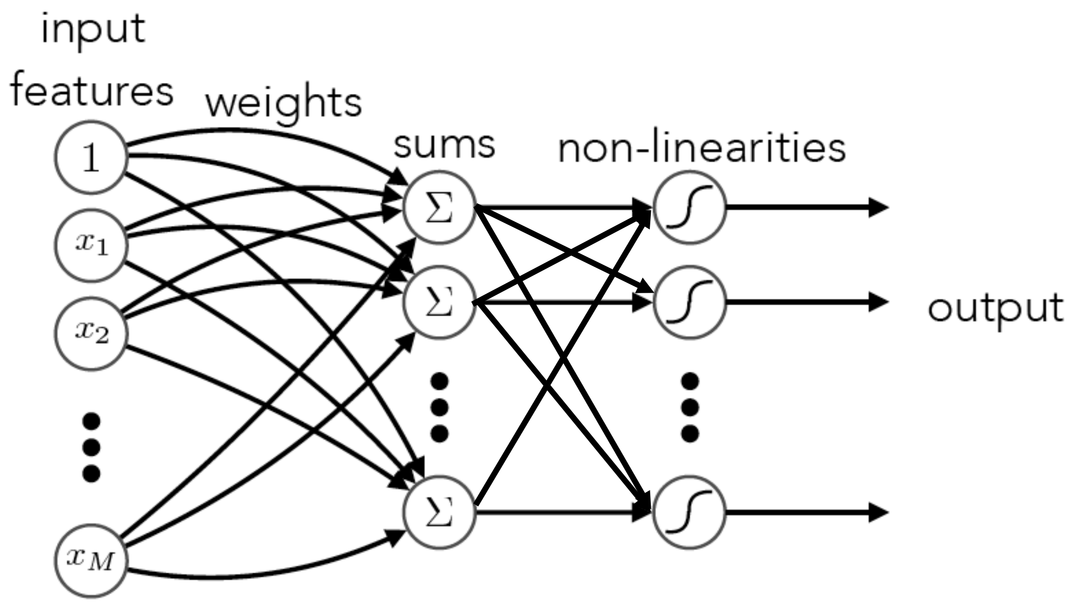
- Element-wise nonlinear functions
  - Independent feature/attribute detectors



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

What is the output?

- Nonlinear functions with vector input
  - Competition between neurons

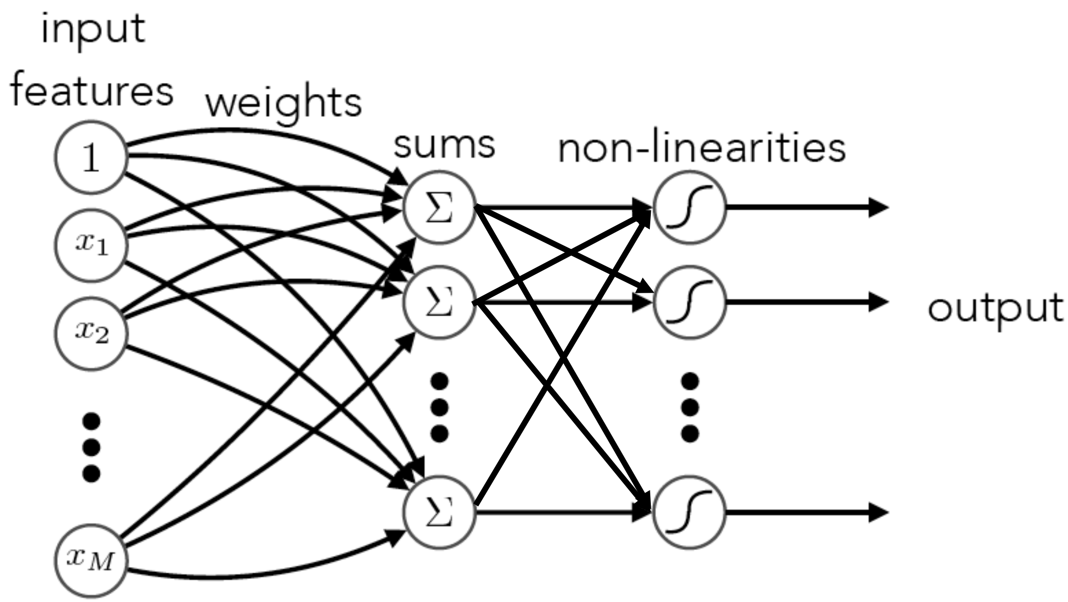


$$\mathbf{h} = [h_j]$$

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

What is the output?

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

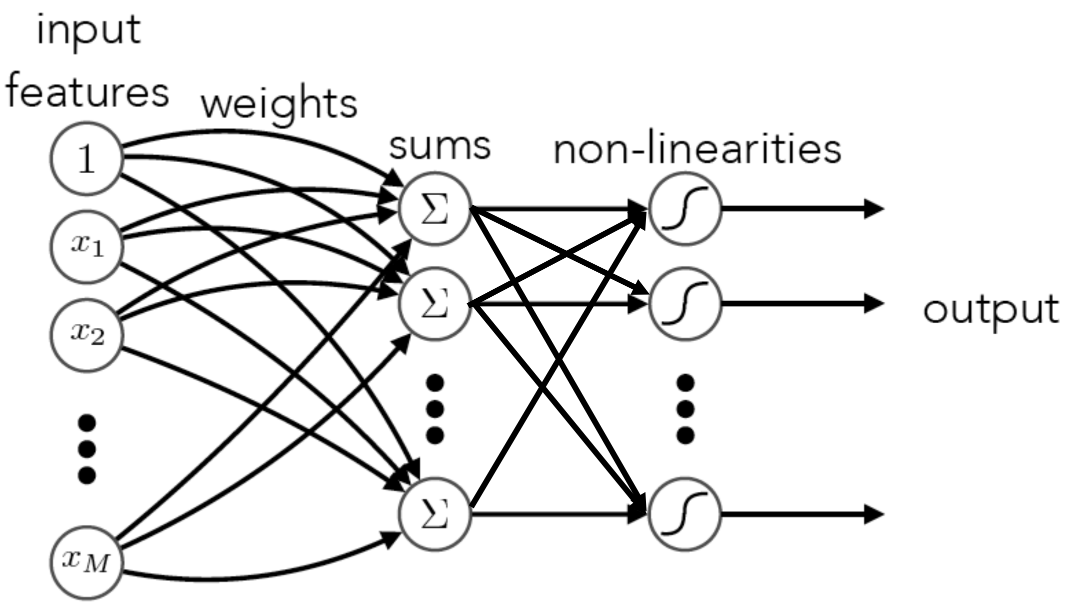


$$\mathbf{h} = [h_j]$$

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

A probabilistic perspective

- Change the output nonlinearity



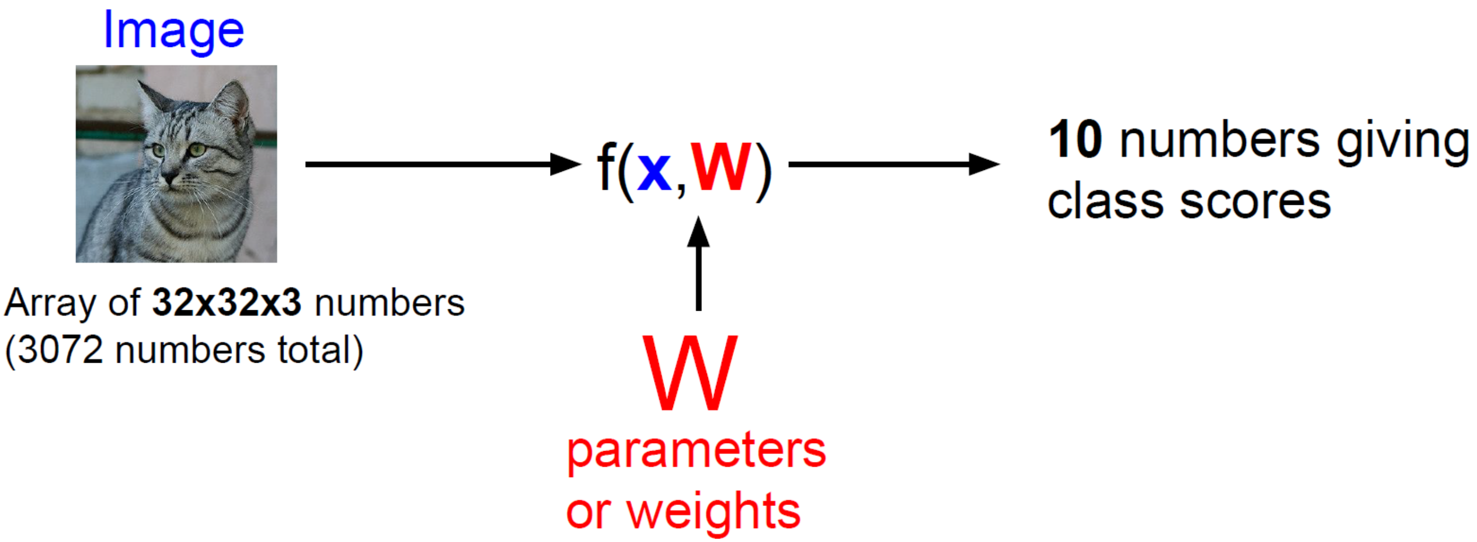
  - From WTA to Softmax function

scores = unnormalized log probabilities of the classes.

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

Example: Multiclass classification

- CIFAR10 as an example



- The output/prediction: WTA

Multiclass linear classifiers

- Extending linear classifier in binary case

Image

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

$$\underbrace{\boxed{f(x,W)}}_{10\text{x}1} = \underbrace{\boxed{W}}_{10\text{x}3072}\overbrace{\boxed{x}}^{3072\text{x}1} + \boxed{b}\ \ 10\text{x}1$$

$$\text{Image} \longrightarrow f(\mathbf{x},\mathbf{W}) \longrightarrow \text{10 numbers giving class scores}$$

10 numbers giving
class scores

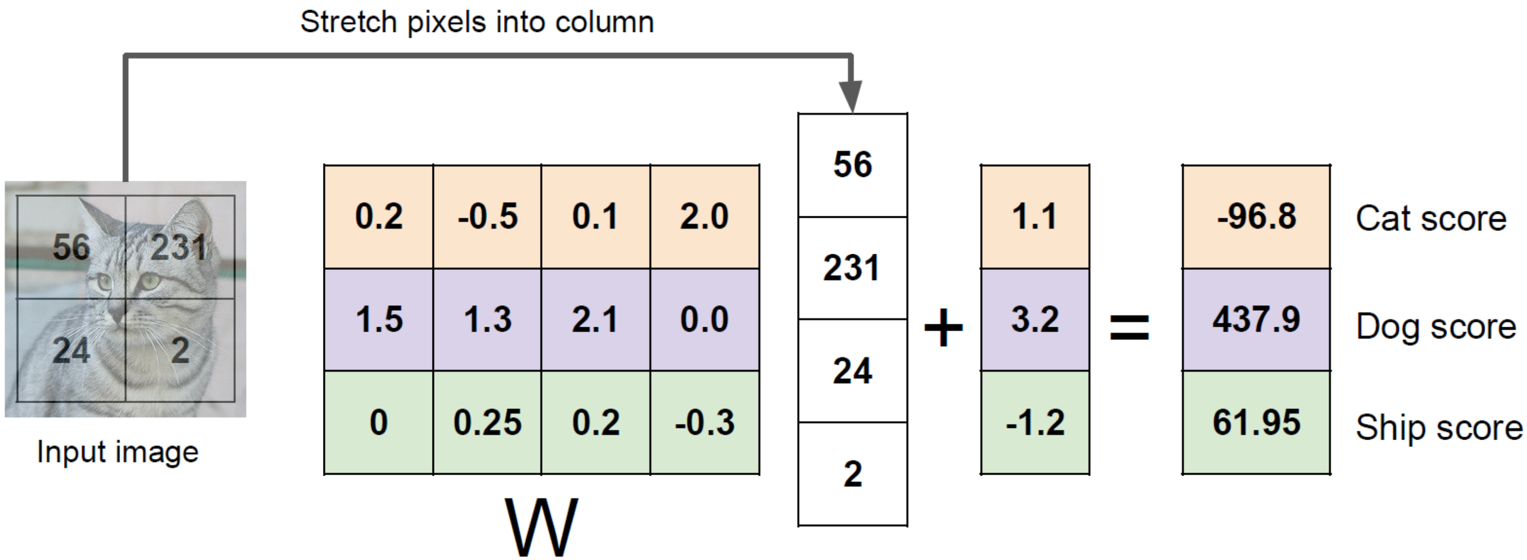
$\mathbf{s}$

$\uparrow$ 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.

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

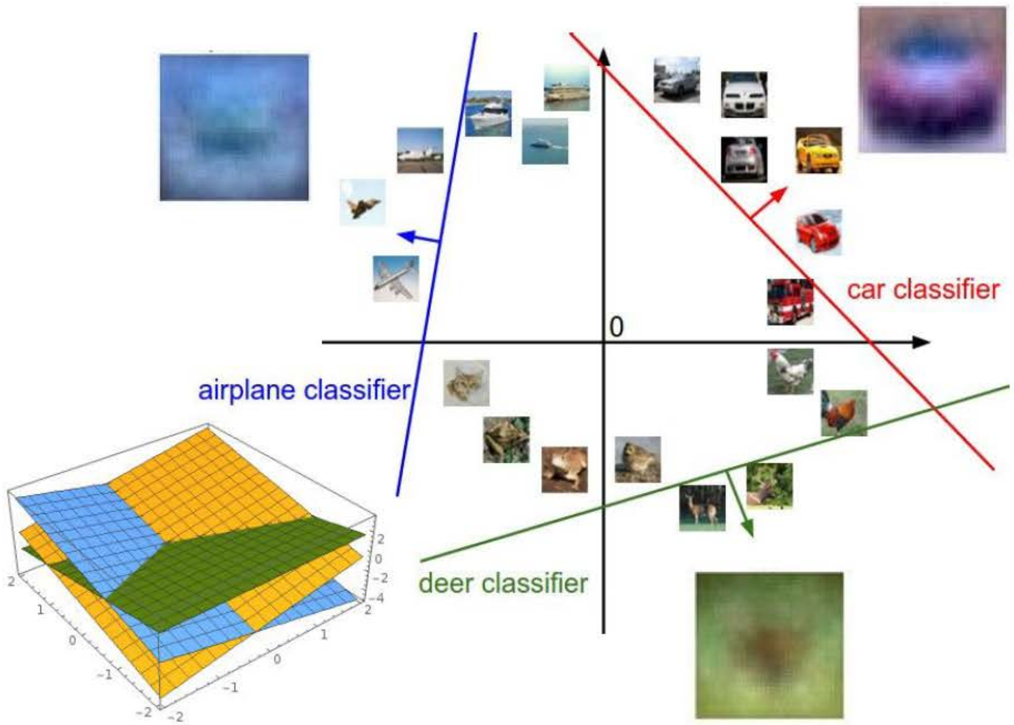
where

$$\boxed{s=f(x_i;W)}$$

	unnormalized log probabilities	$\xrightarrow{\text{exp}}$ unnormalized probabilities	$\xrightarrow{\text{normalize}}$ probabilities
cat	**3.2**	**24.5**	**0.13**
car	5.1	164.0	0.87
frog	-1.7	0.18	0.00

Interpreting network weights

- What are those weights?



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

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

How to learn a single-layer network?

- 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 single-layer network

- 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.

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

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

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

Logistic Regression

- Learning loss: example

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

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

→ L_i = -log(0.13)
= **0.89**

Logistic Regression

- Learning loss: questions

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

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

unnormalized probabilities

cat	**3.2**	**24.5**	**0.13**
car	5.1	164.0	0.87
frog	-1.7	0.18	0.00

$\xrightarrow{\text{exp}}$ $\xrightarrow{\text{normalize}}$

L_i = -log(0.13)
= **0.89**

unnormalized log probabilities

probabilities

Logistic Regression

- Learning loss: questions

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

unnormalized probabilities

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

	unnormalized log probabilities	exp →	normalize →	probabilities
cat	**3.2**	**24.5**	**0.13**	→ L_i = -log(0.13) = **0.89**
car	5.1	164.0	0.87	
frog	-1.7	0.18	0.00	

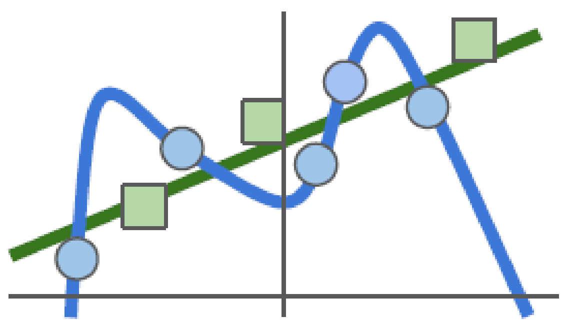
Learning with regularization

- Constraints on hypothesis space
  - Similar to Linear Regression

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

Data loss: Model predictions should match training data

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

- Sparsity

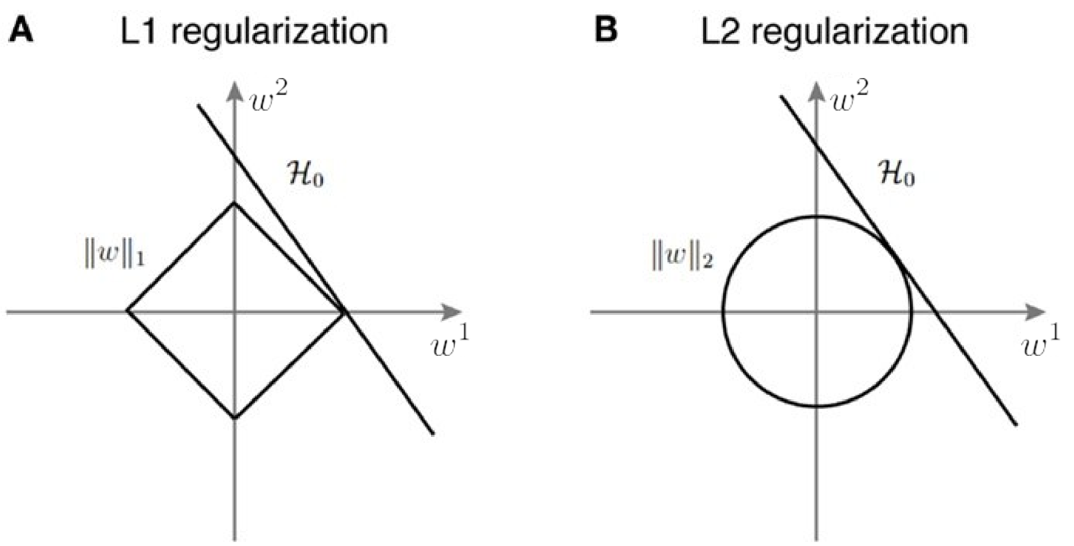
$$x = [1, 1, 1, 1]$$

$$w_1 = [1, 0, 0, 0]$$
$$w_2 = [0.25, 0.25, 0.25, 0.25]$$
$$w_3 = [0.5, 0.5, 0, 0]$$

$$f(x) = w^\mathsf{T} x$$

$$w_1^\mathsf{T} x = w_2^\mathsf{T} x = w_3^\mathsf{T} x$$

$$\|w_1\|^2 = |w_1| = 1$$
$$\|w_2\|^2 = 4/16 = 1/4, |w_2| = 1$$
$$\|w_3\|^2 = 2/4 = 1/2, |w_3| = 1$$



Optimization of loss functions

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

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

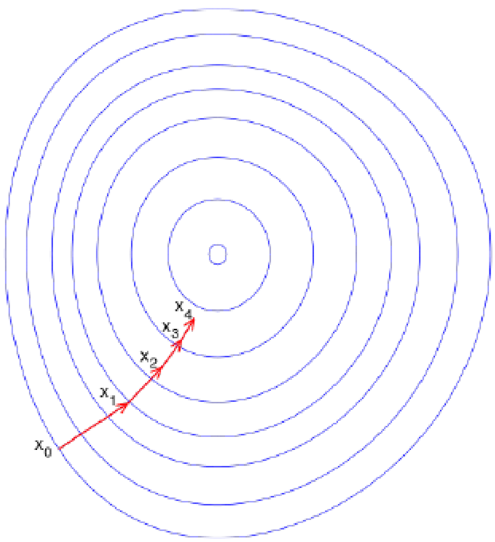
until stop

  - η_t is the learning rate, and

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

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

Two dimensional example:



Optimization: gradient descent

- Numerical gradient

current W:	**W + h** (first dim):	**gradient dW:**
[0.34,	[0.34 + **0.0001**,	[-2.5,
-1.11,	-1.11,	?,
0.78,	0.78,	?,
0.12,	0.12,	
0.55,	0.55,	
2.81,	2.81,	
-3.1,	-3.1,	
-1.5,	-1.5,	?,
0.33,…]	0.33,…]	?,…]
loss 1.25347	**loss 1.25322**	

(1.25322 - 1.25347)/0.0001
= -2.5

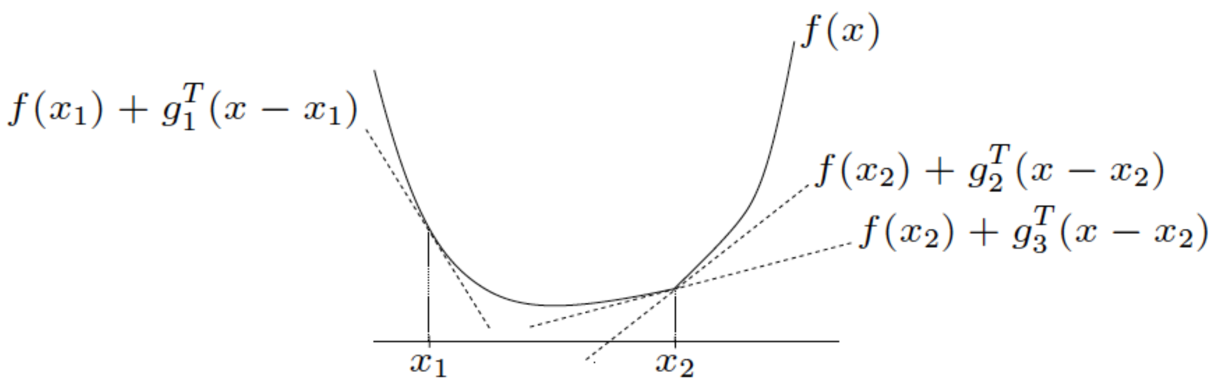
$$\frac{df(x)}{dx} = \lim_{h \to 0} \frac{f(x+h) - f(x)}{h}$$

Optimization: gradient descent

- Analytic gradient
  - Differentiable function: Calculus
  - Non-differentiable function: Sub-gradient

g is a **subgradient** of f (not necessarily convex) at x if

$$f(y) \geq f(x) + g^T(y - x) \quad \text{for all } y$$

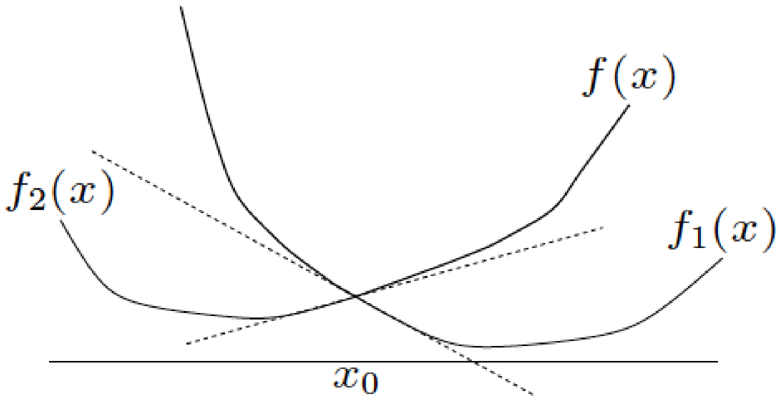


g_2, g_3 are subgradients at x_2; g_1 is a subgradient at x_1

Optimization: gradient descent

- Sub-gradient example

$f = \max\{f_1, f_2\}$, with f_1, f_2 convex and differentiable



- $f_1(x_0) > f_2(x_0)$: unique subgradient $g = \nabla f_1(x_0)$
- $f_2(x_0) > f_1(x_0)$: unique subgradient $g = \nabla f_2(x_0)$
- $f_1(x_0) = f_2(x_0)$: subgradients form a line segment $[\nabla f_1(x_0), \nabla f_2(x_0)]$

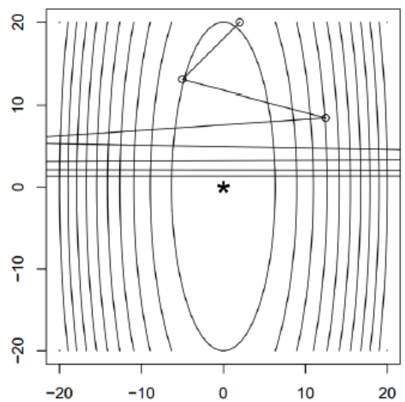
Optimization: gradient descent

- Gradient descent

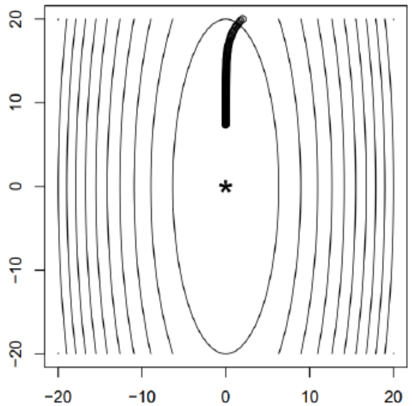
```
# Vanilla Gradient Descent

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 η_t, after 100 iterations



Optimization: gradient descent

- Stochastic gradient descent

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

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

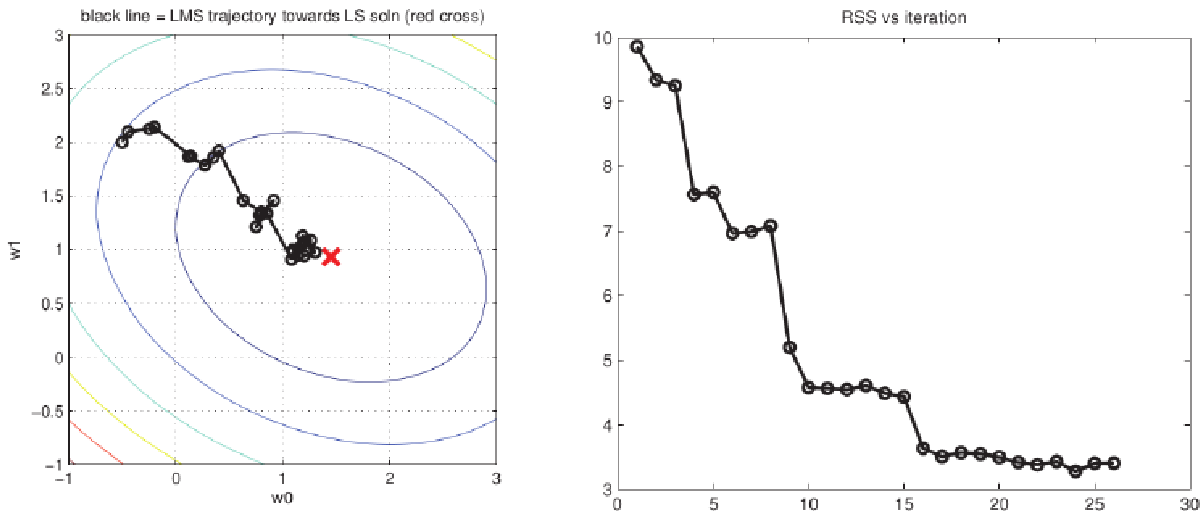
Full sum expensive when N is large!

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

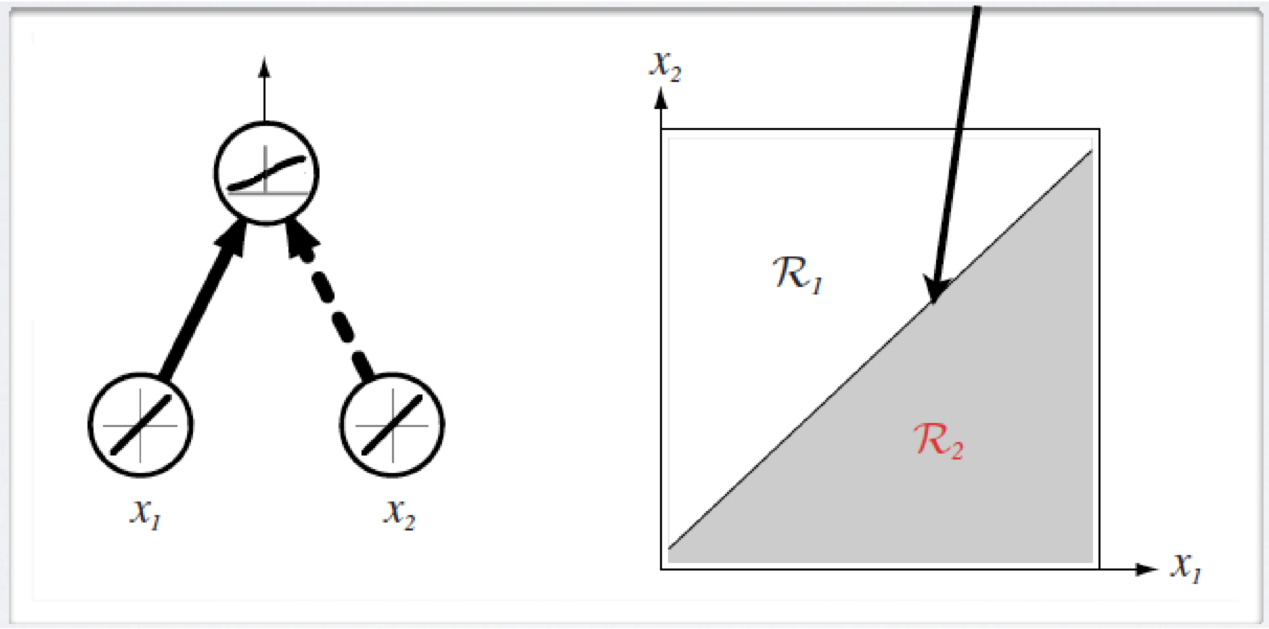
- Stochastic gradient descent



- 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

Capacity of single neuron

- Binary classification
  - A neuron estimates $P(y = 1|\mathbf{x}) = \sigma(\mathbf{w}^\intercal \mathbf{x})$
  - Its decision boundary is linear, determined by its weights



Capacity of single neuron

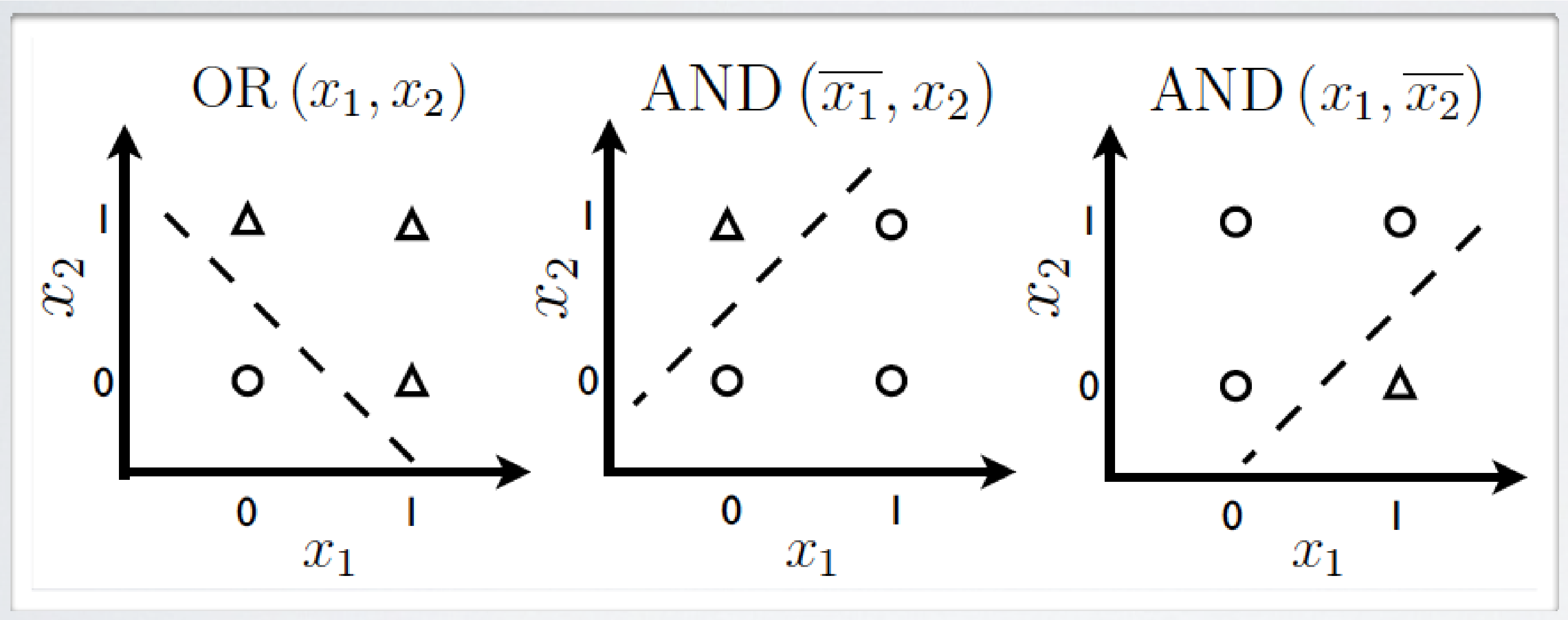
- Can solve linearly separable problems

$$\mathcal{D} = \mathcal{D}^+ \cup \mathcal{D}^-$$

$$\exists\ \mathbf{w}^*, \mathbf{w}^{*\mathsf{T}}\mathbf{x} > 0,\ \forall \mathbf{x} \in \mathcal{D}^+$$

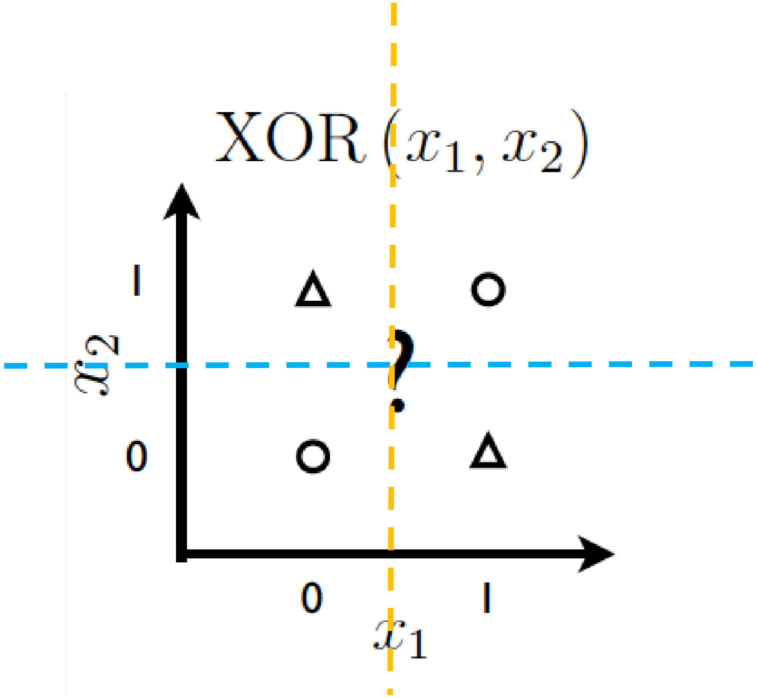
$$\mathbf{w}^{*\mathsf{T}}\mathbf{x} < 0,\ \forall \mathbf{x} \in \mathcal{D}^-$$

  - Examples



Capacity of single neuron

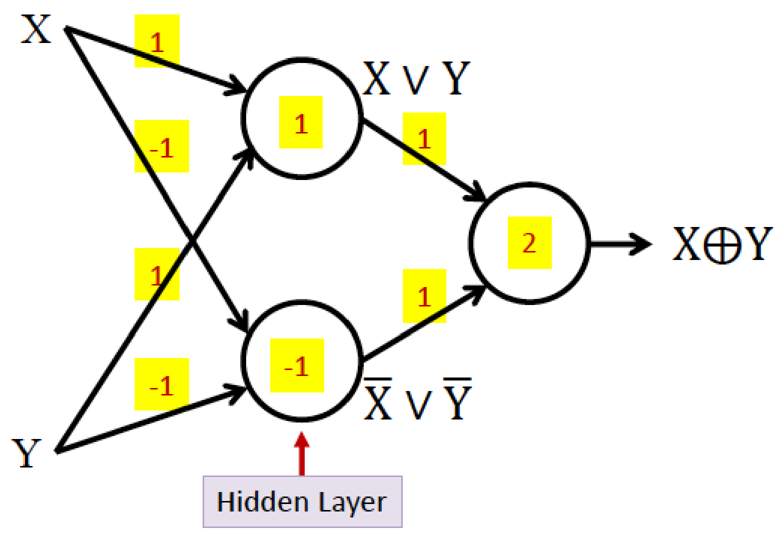
- Can't solve non linearly separable problems



- Can we use multiple neurons to achieve this?

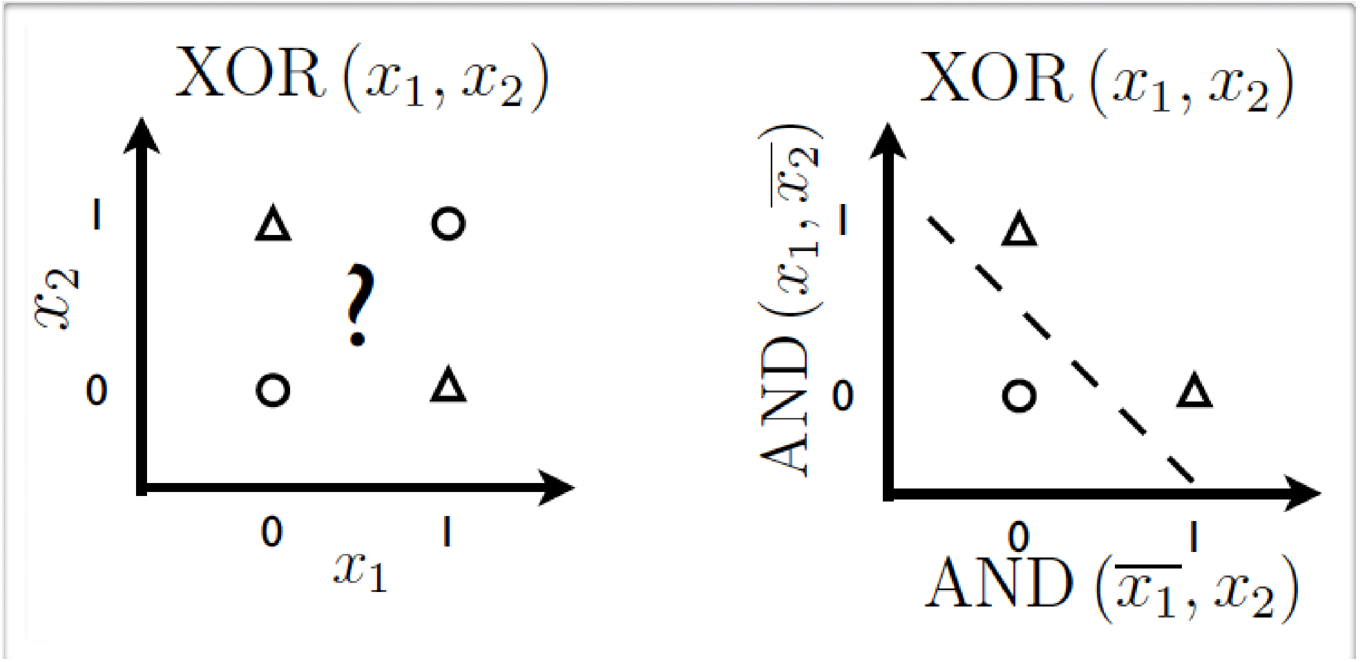
Capacity of single neuron

- Can't solve non linearly separable problems
- Unless the input is transformed in a better representation



Capacity of single neuron

- Can't solve non linearly separable problems



- Unless the input is transformed in a better representation

Adding one more layer

- Single hidden layer neural network
  - 2-layer neural network: ignoring input units

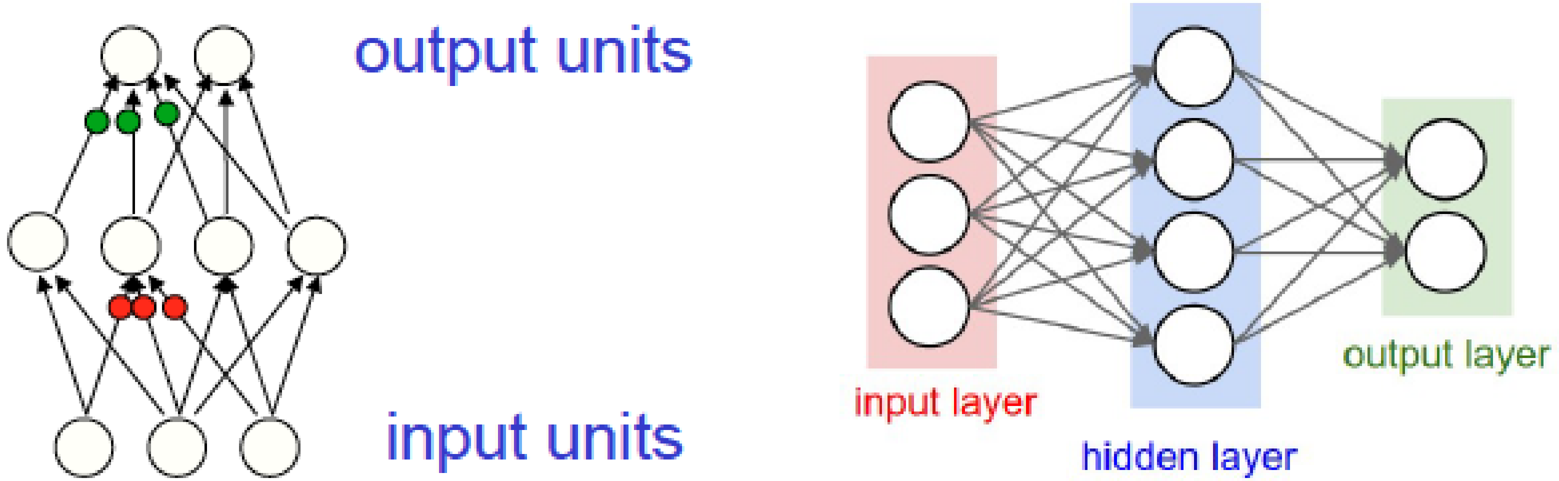


Figure : Two different visualizations of a 2-layer neural network. In this example: 3 input units, 4 hidden units and 2 output units

- Q: What if using linear activation in hidden layer?

Summary: Single-layer neural network

- Single layer neural networks
  - Multiclass linear classifiers
  - Loss functions with regularization
  - Optimization via gradient descent
  - Limited representation power