Lecture 2: Basic Artificial Neural Networks and MLP

Yujiao Shi

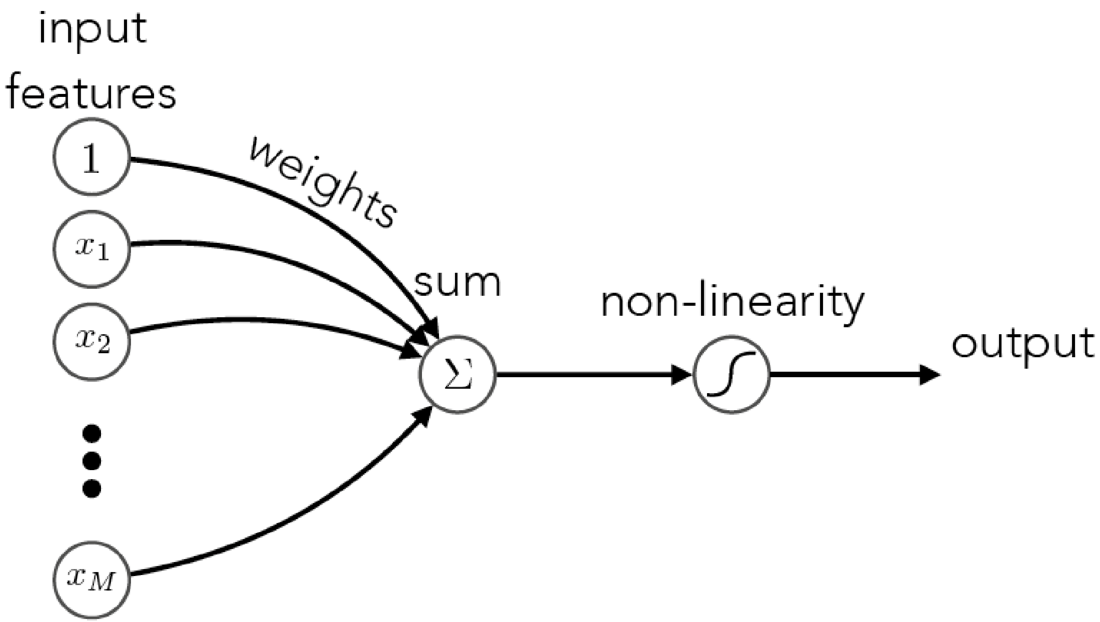
SIST, ShanghaiTech

Spring, 2024

- 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*

bias input features

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

sum weights

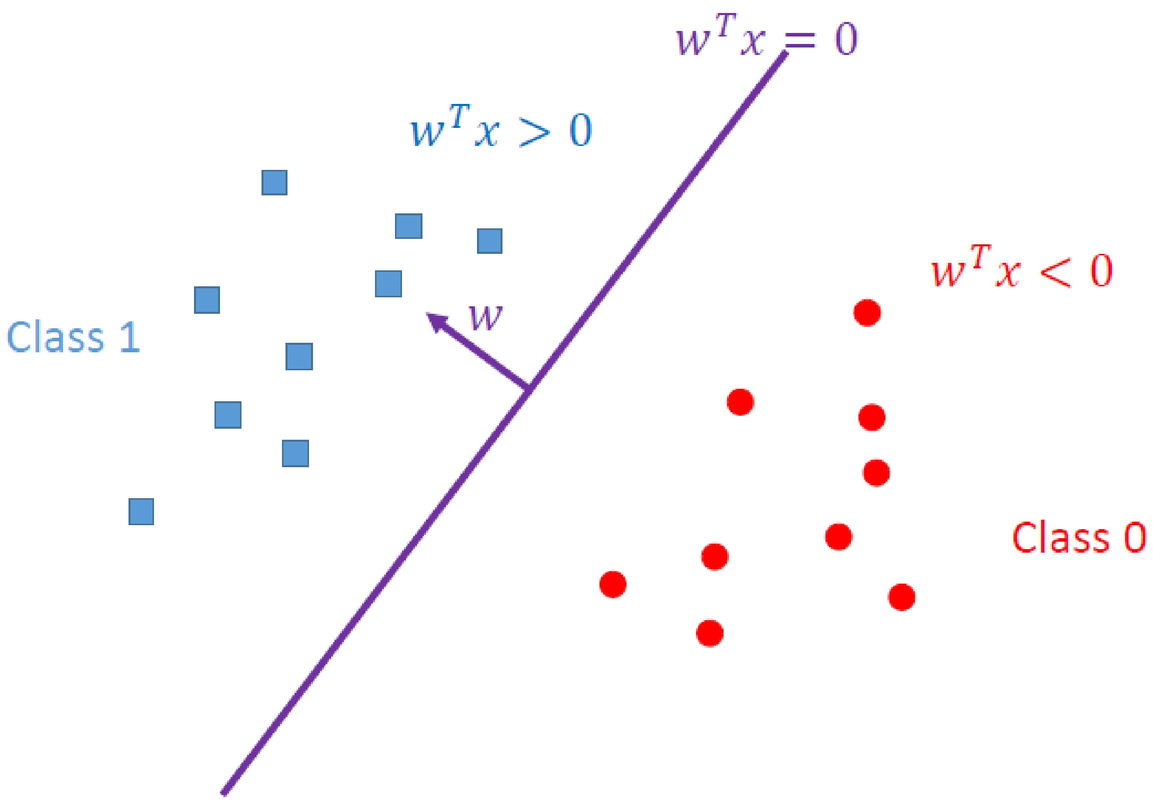
$$h = g(s)$$

output non-linearity sum

Single neuron as a linear classifier

上海科技大学 ShanghaiTech University

- Binary classification

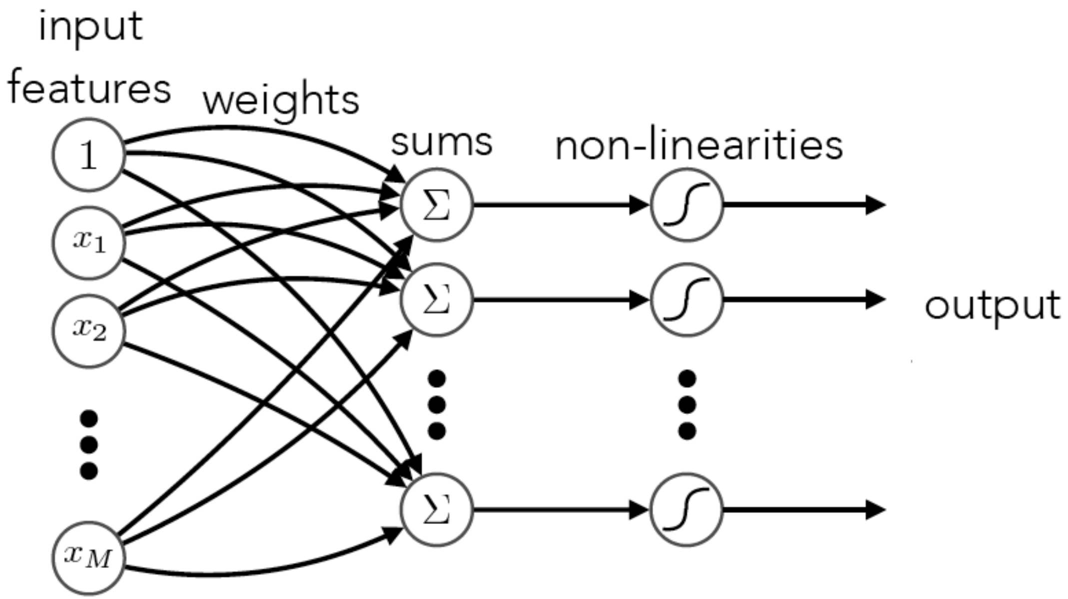


上海科技大学
ShanghaiTech University

- 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

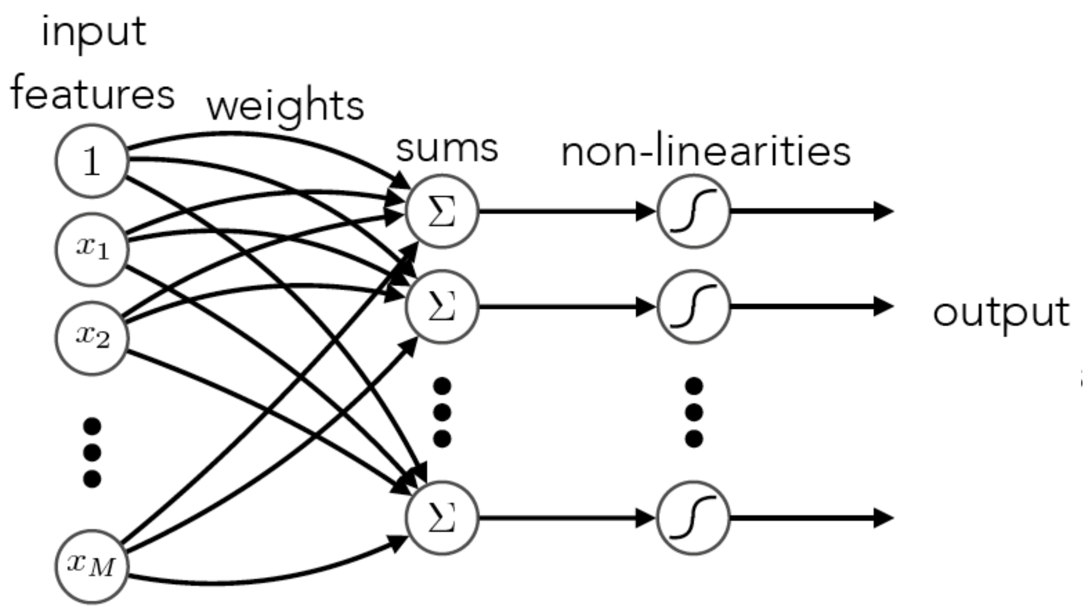


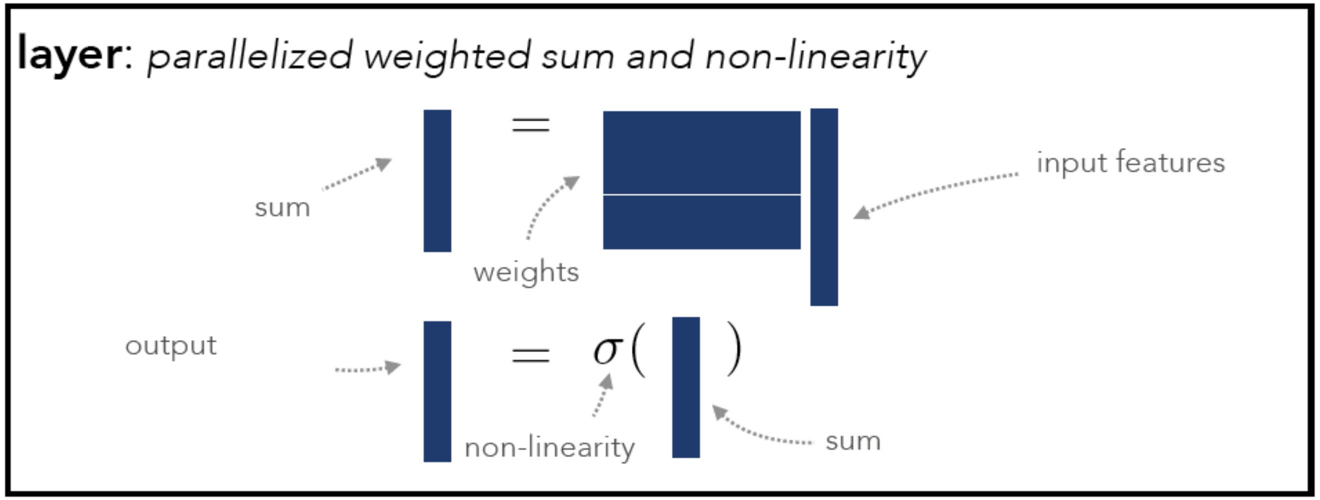
layer: *parallelized weighted sum and non-linearity*

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*

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

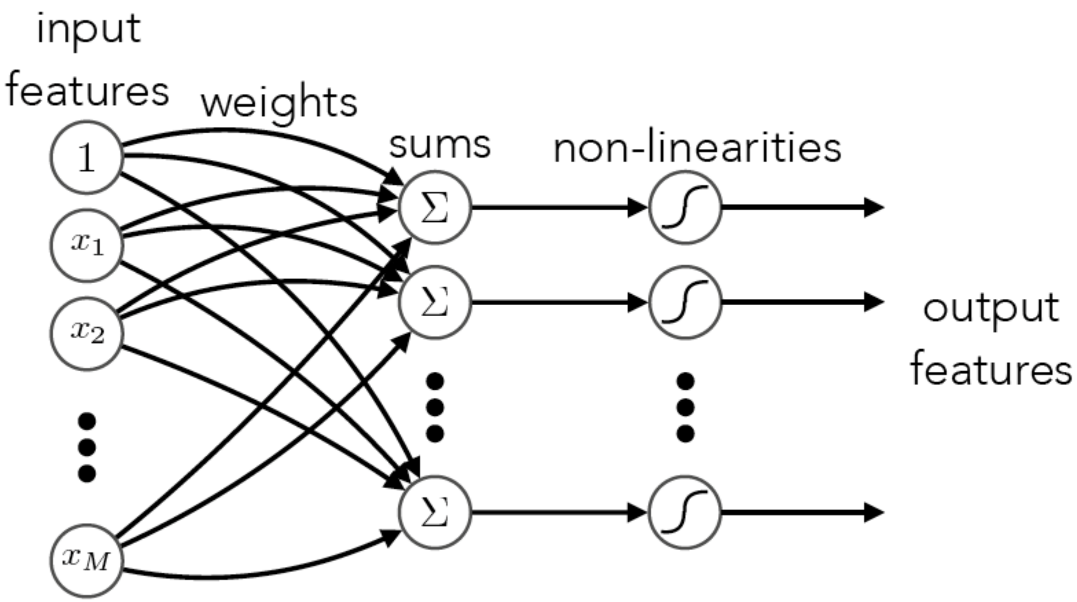
Single layer neural network





What is the output?

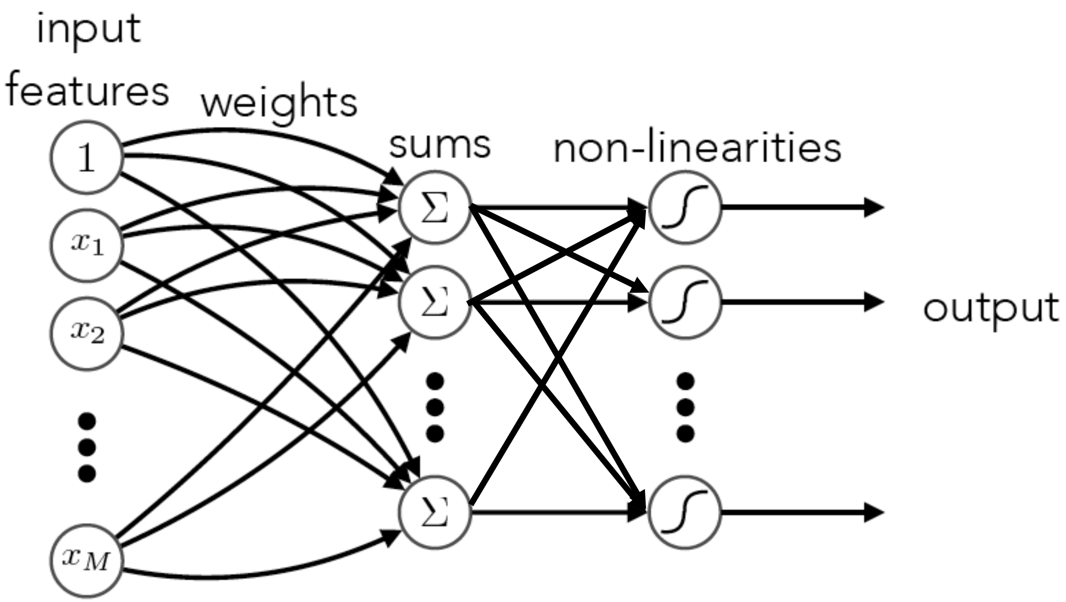
- Element-wise nonlinear functions
  - Independent feature/attribute detectors



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

What is the output?

- Nonlinear functions with vector input
  - Competition between neurons

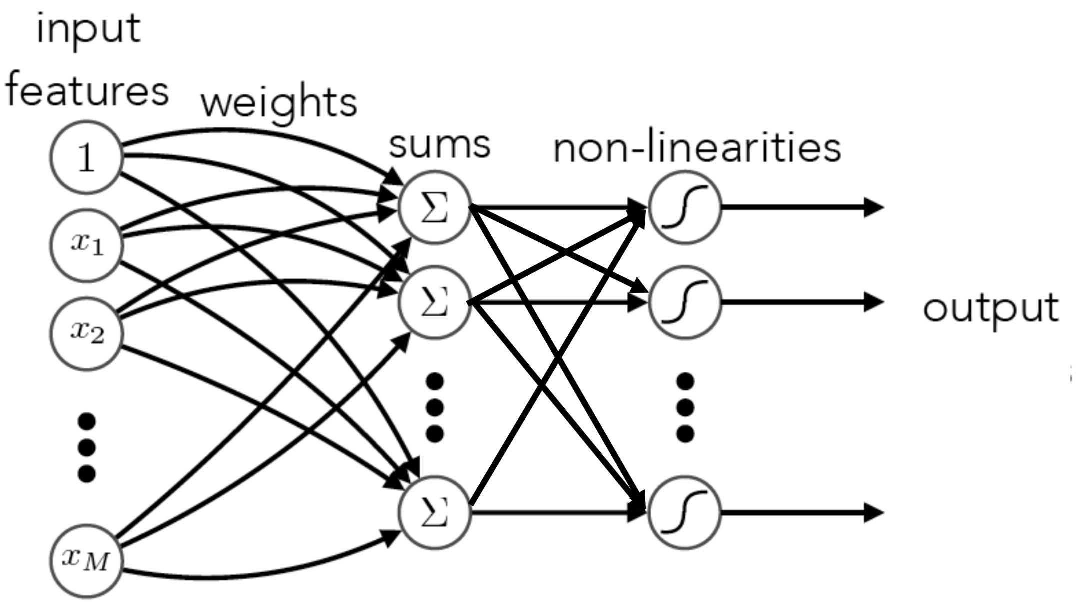


$$\mathbf{h} = [h_j]$$

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

What is the output?

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

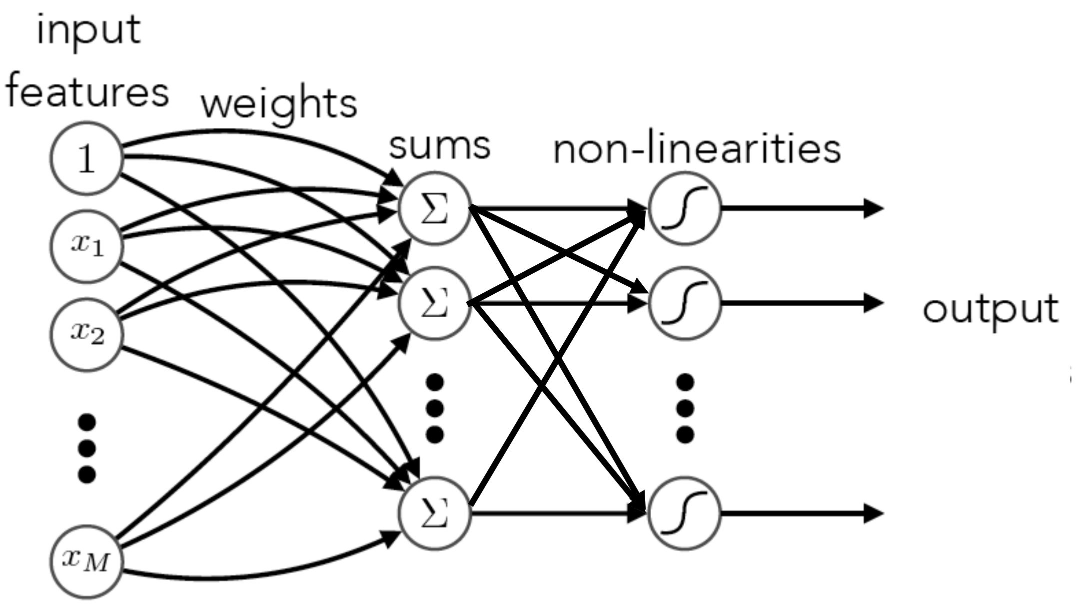


$$\mathbf{h} = [h_j]$$

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

A probabilistic perspective

- Change the output nonlinearity



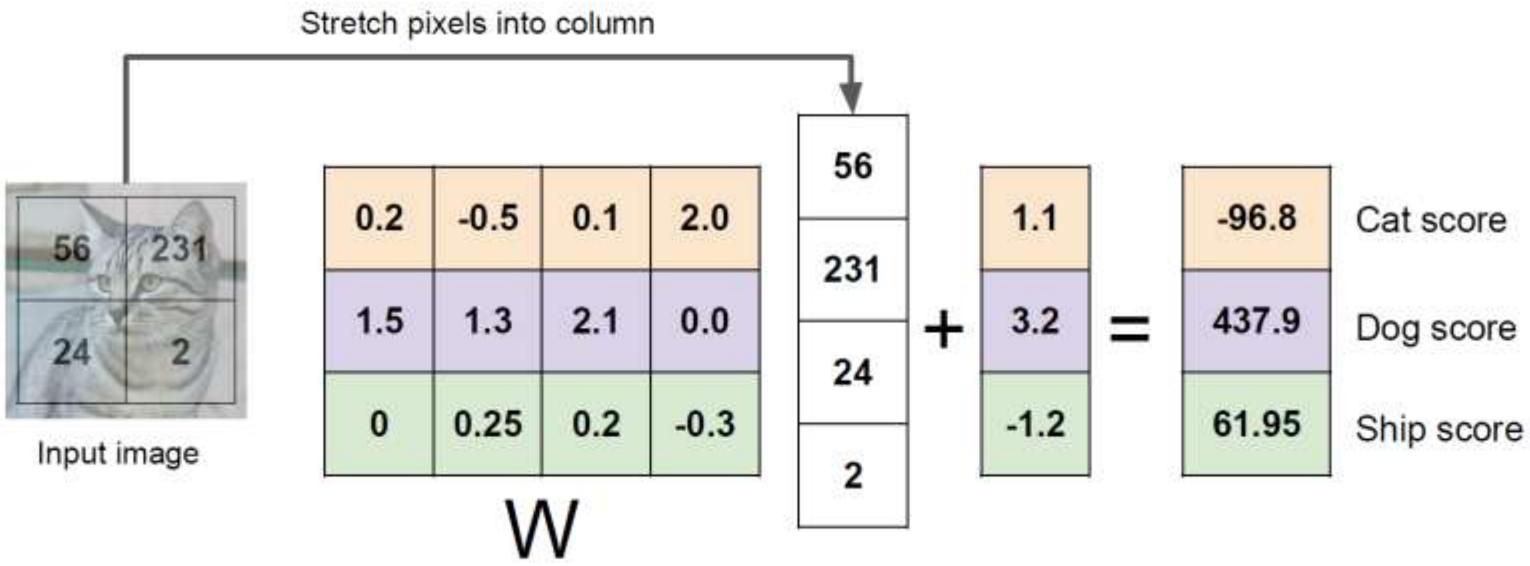
  - From WTA to Softmax function

scores = unnormalized log probabilities of the classes.

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

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

上海科技大学 ShanghaiTech University

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

How to learn a multiclass classifier?

- Define a loss function and do minimization

  - 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

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

Empirical loss

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:

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

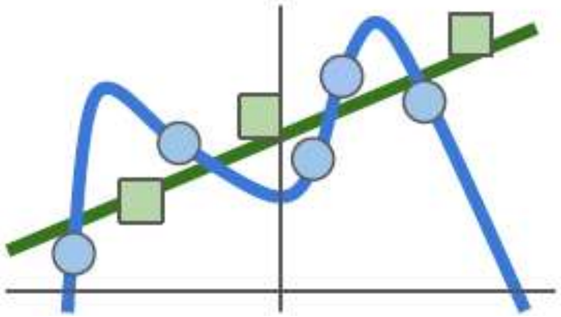
Learning with regularization

- Constraints on hypothesis space
  - Similar to Linear Regression

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

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

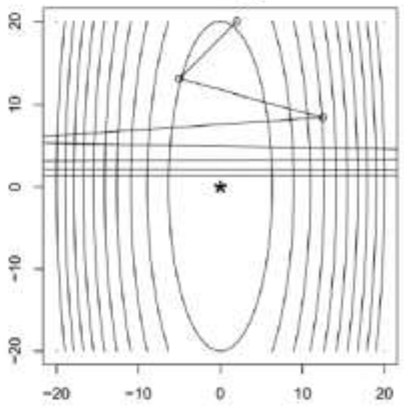
Optimization: gradient descent

- Gradient descent

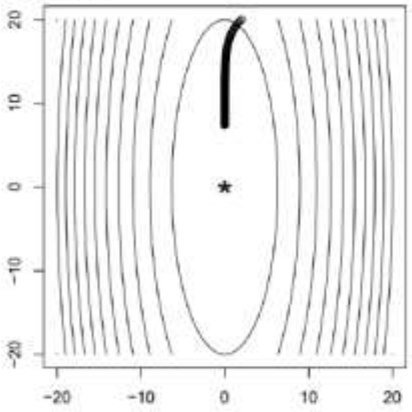
```
# Vanilla Gradient Descent

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

上海科技大学 ShanghaiTech University

- Stochastic gradient descent

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

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

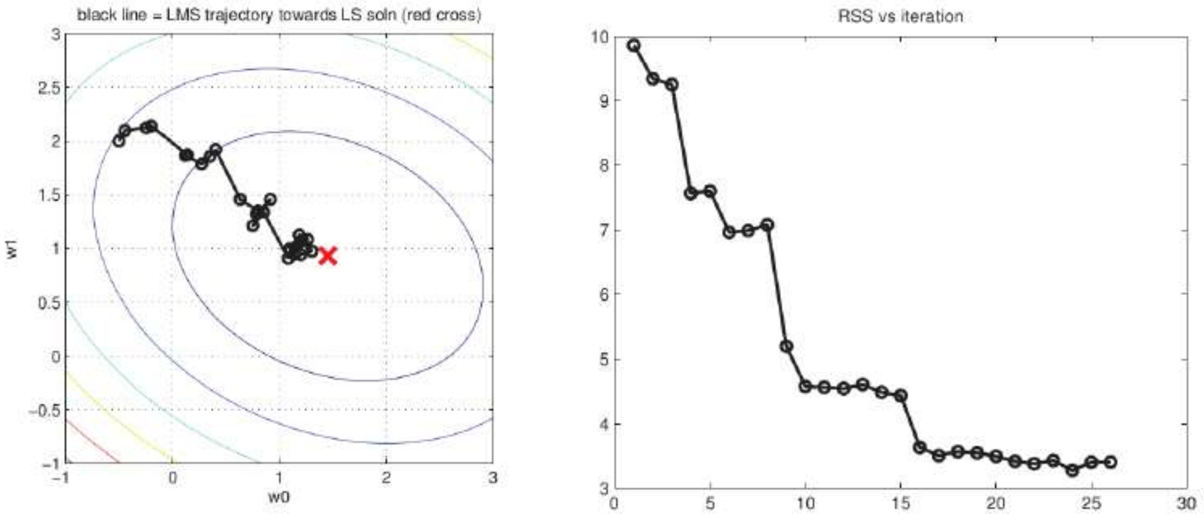
Full sum expensive when N is large!

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

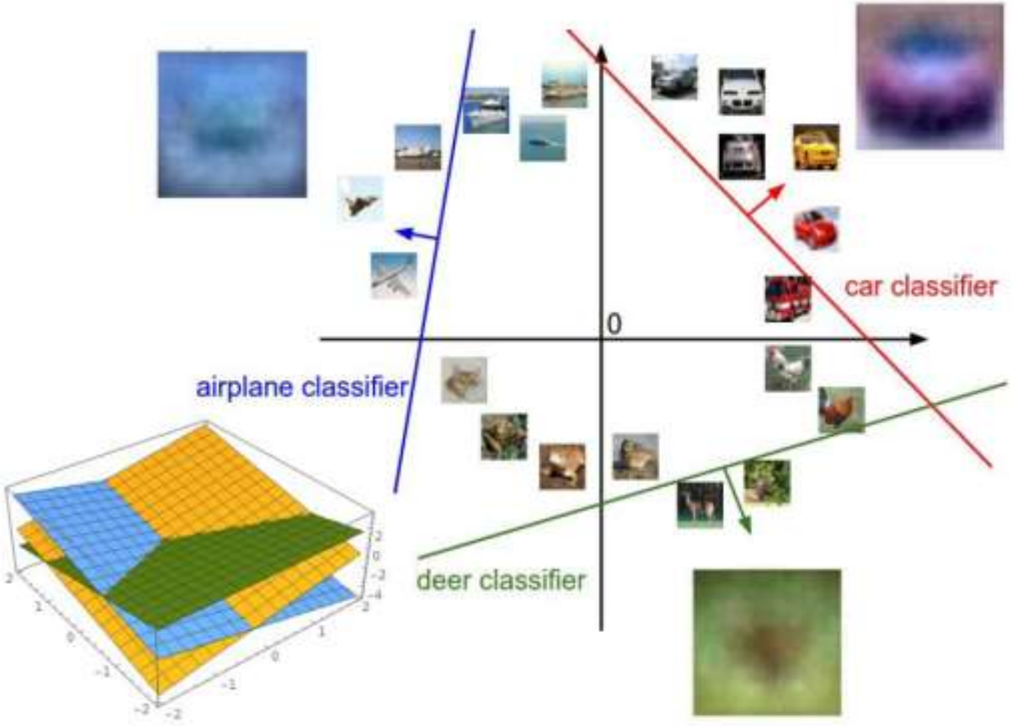
- Stochastic gradient descent



- 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

Interpreting network weights

- What are those weights?



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

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

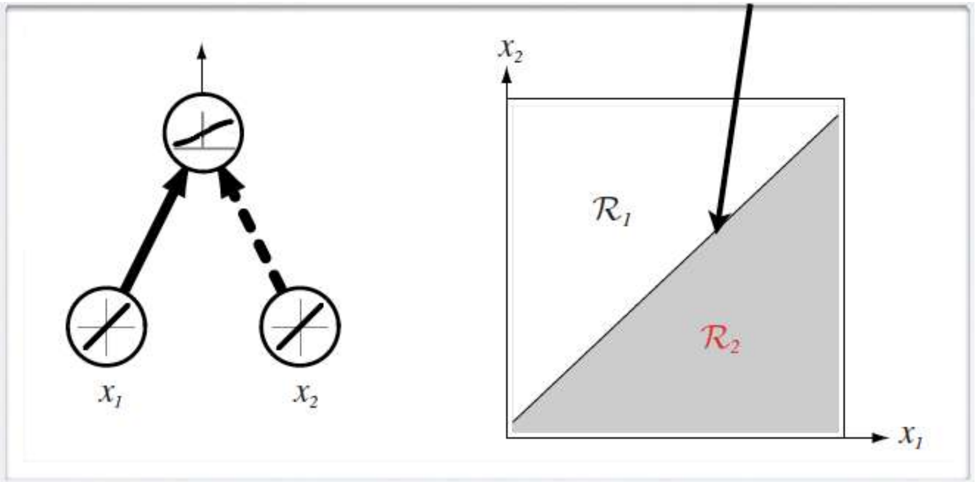
Outline

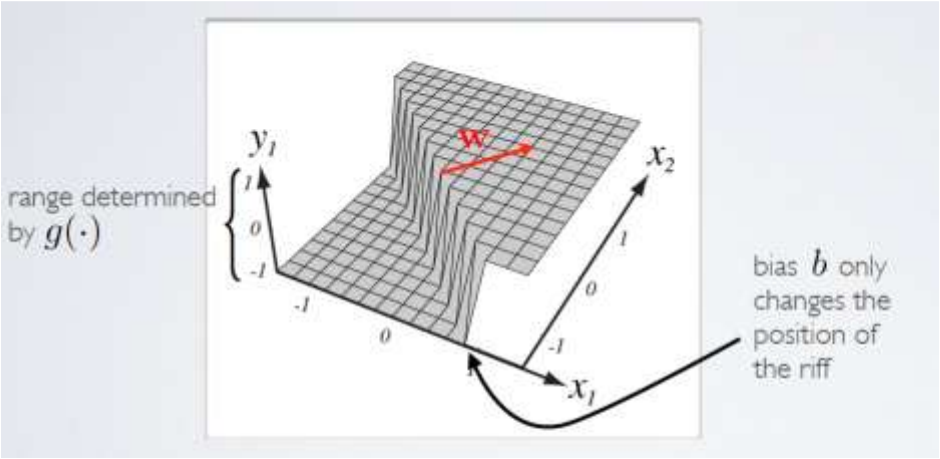
- Single layer neural networks
  - Network models
  - Example: Logistic Regression
- Multi-layer neural networks
  - Limitations of single layer networks
  - Networks with single hidden layer

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

Capacity of single neuron

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





Capacity of single neuron

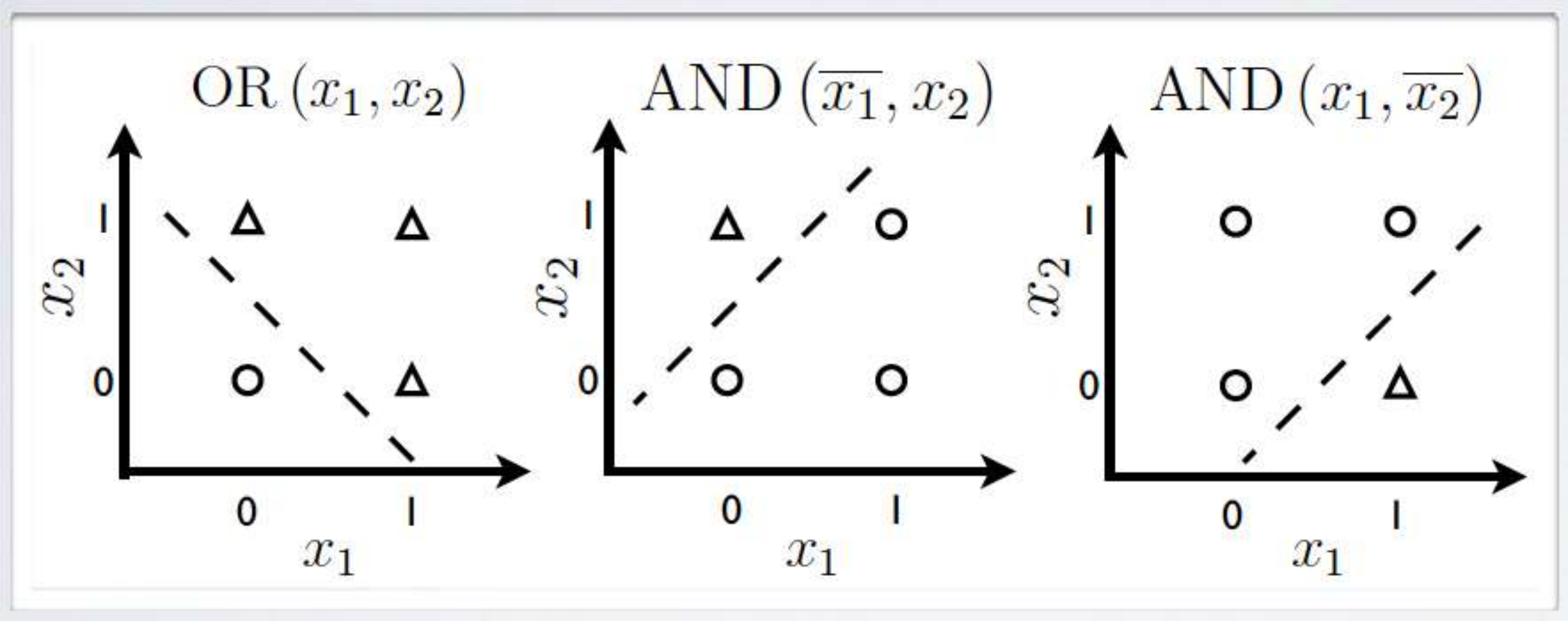
- Can solve linearly separable problems

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

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

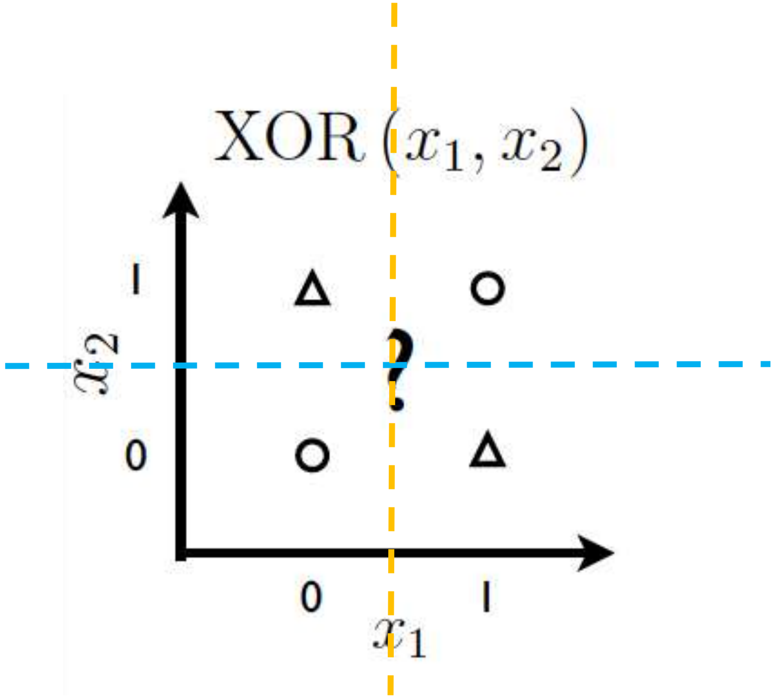
$$ {\mathbf{w}^*}^\intercal \mathbf{x} < 0,\ \forall \mathbf{x} \in \mathcal{D}^-$$

  - Examples



Capacity of single neuron

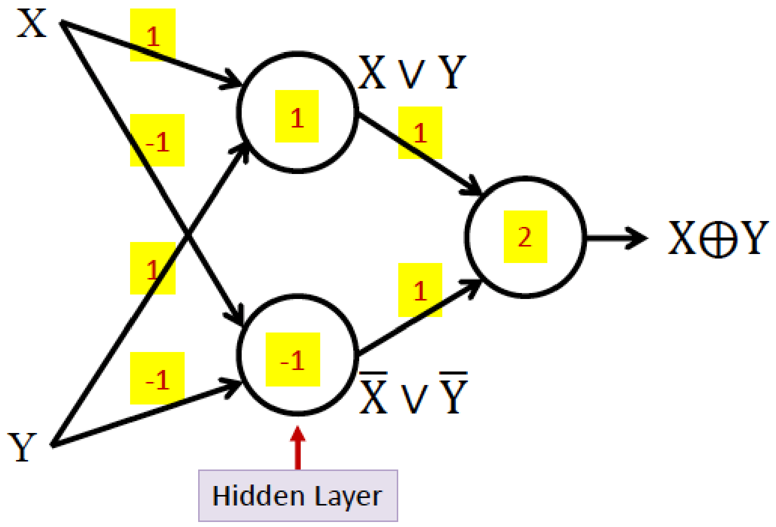
- Can't solve non linearly separable problems



- Can we use multiple neurons to achieve this?

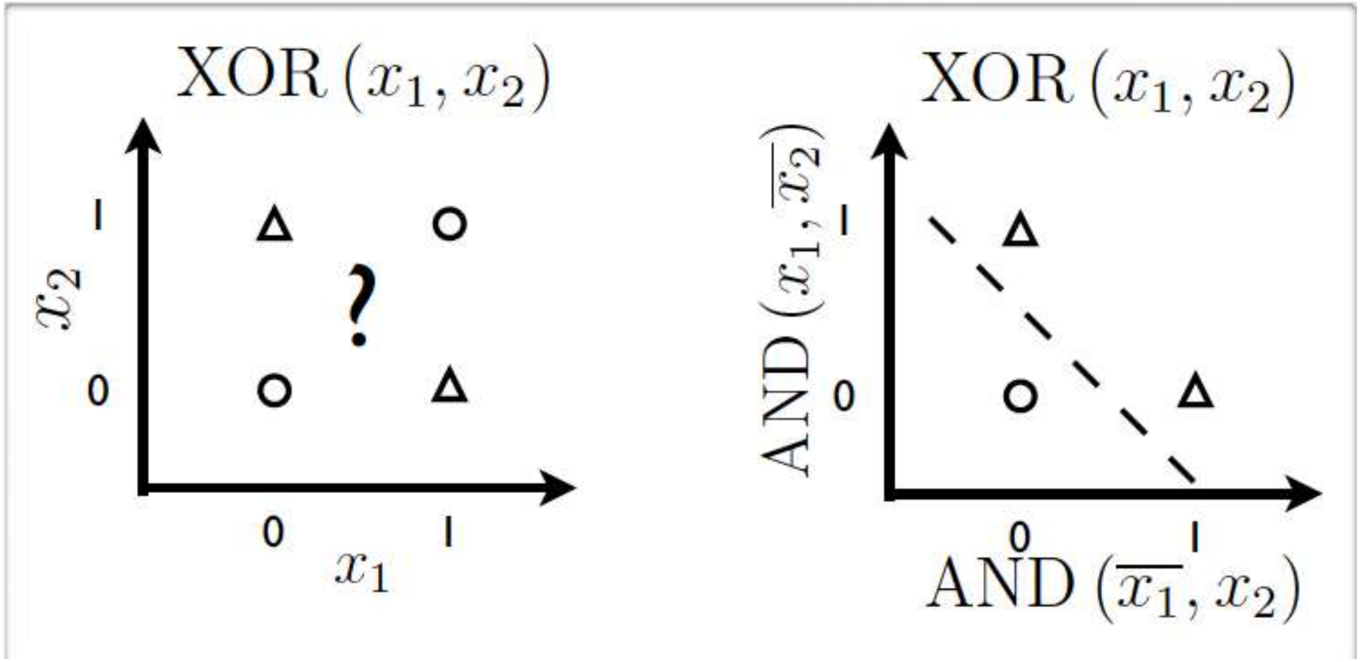
Capacity of single neuron

- 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

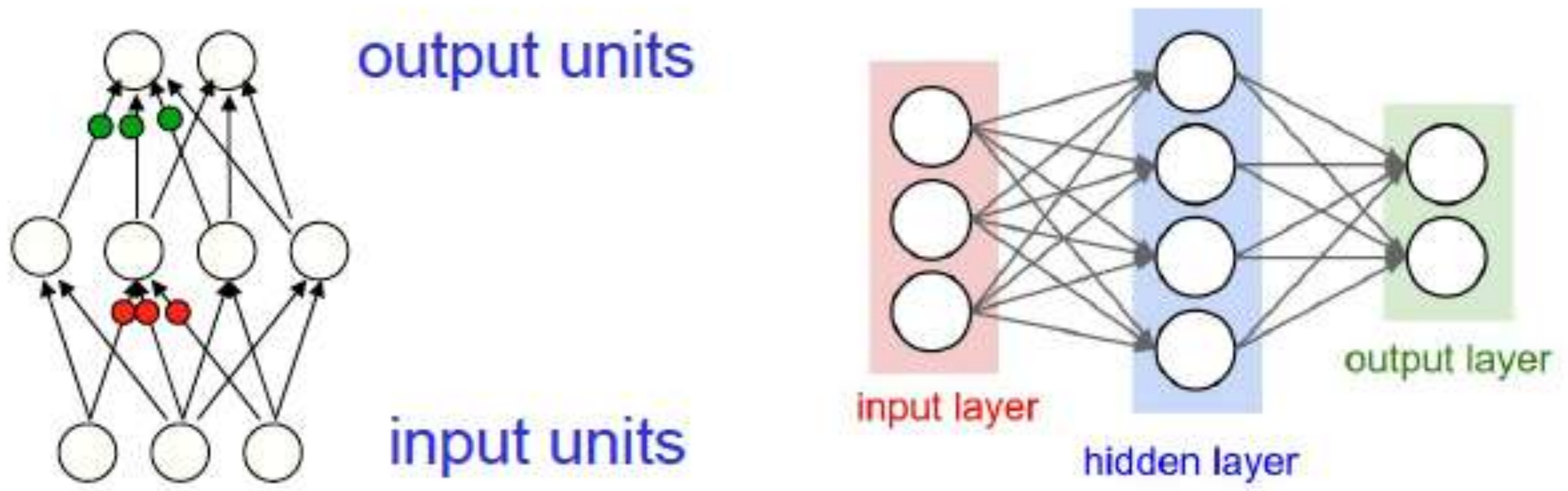
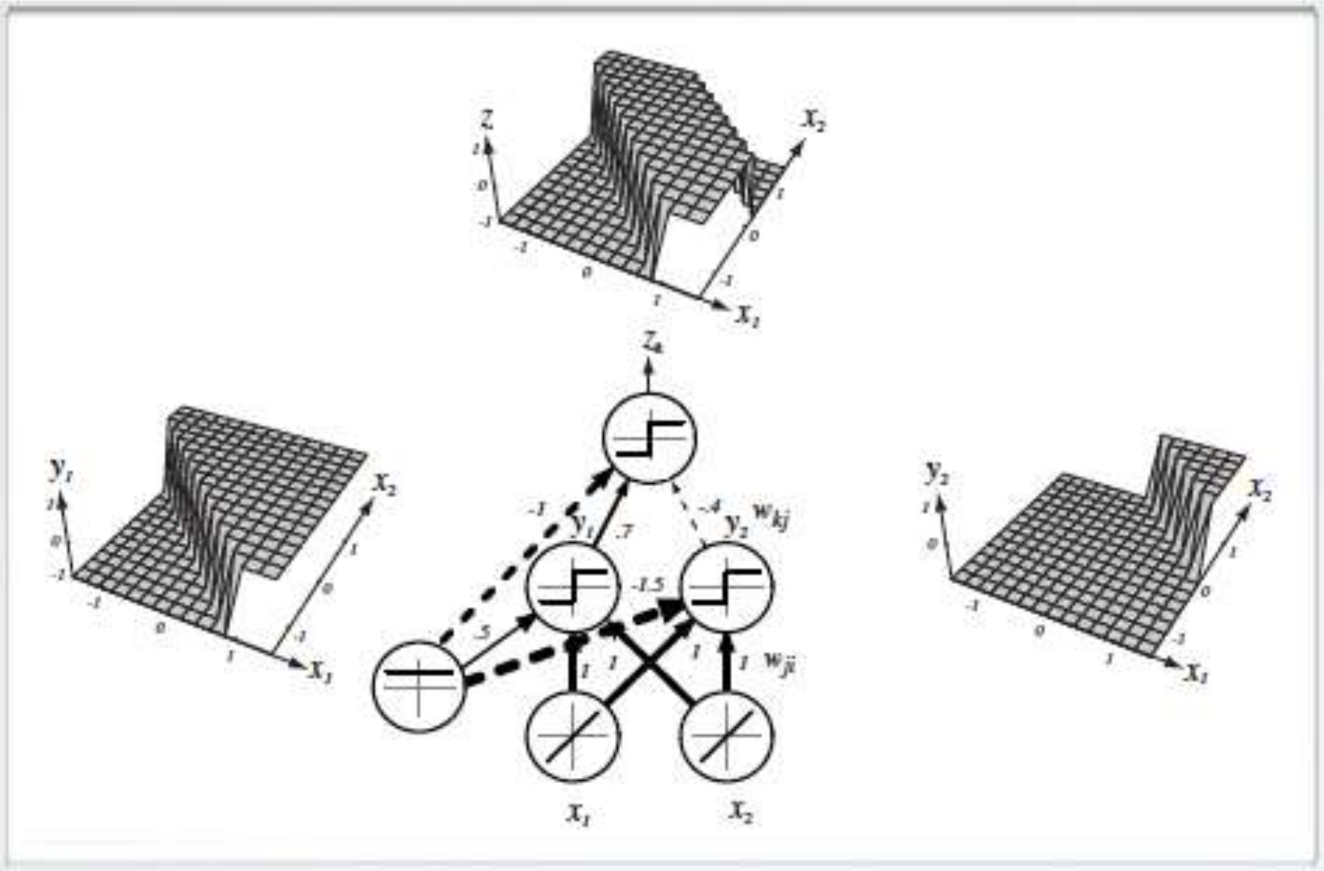


Figure : Two different visualizations of a 2-layer neural network. In this example: 3 input units, 4 hidden units and 2 output units

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

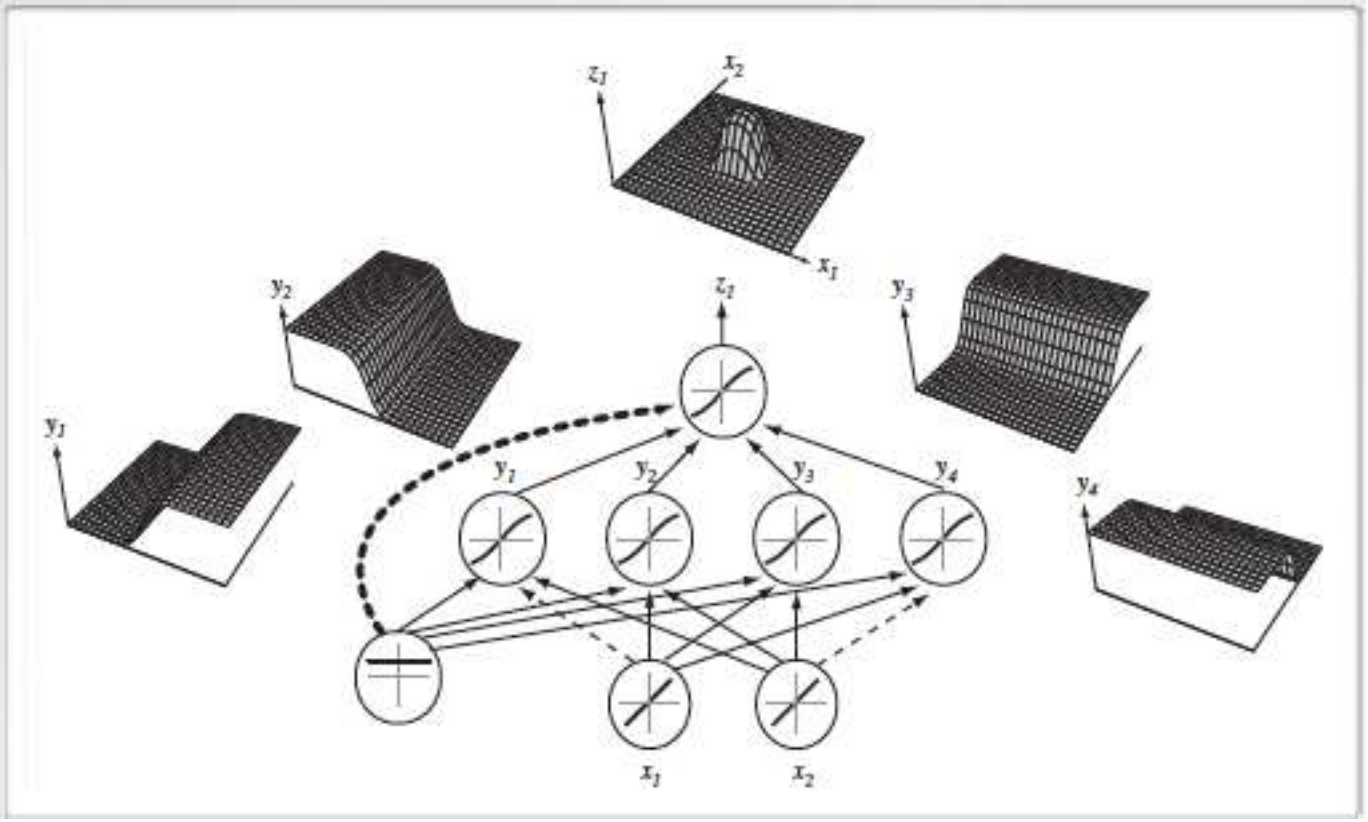
Capacity of neural network

- Single hidden layer neural network
  - Partition the input space into regions



Capacity of neural network

- Single hidden layer neural network
  - Form a stump/delta function



Capacity of neural network

- Universal approximation
  - Theorem (Hornik, 1991)

    A single hidden layer neural network with a linear output unit can approximate any continuous function arbitrarily well, given enough hidden units.
  - The result applies for sigmoid, tanh and many other hidden layer activation functions

- Caveat: good result but not useful in practice
  - How many hidden units?
  - How to find the parameters by a learning algorithm?

General neural network

- Multi-layer neural network

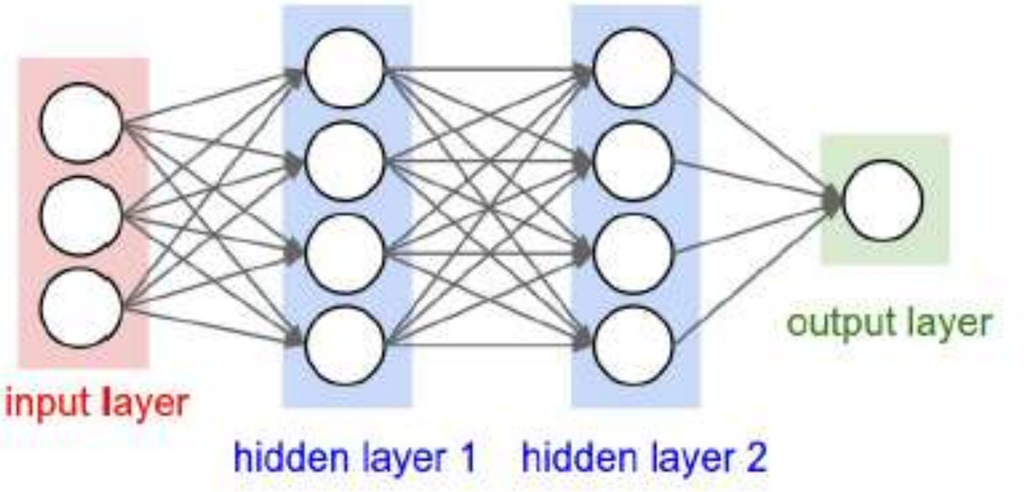
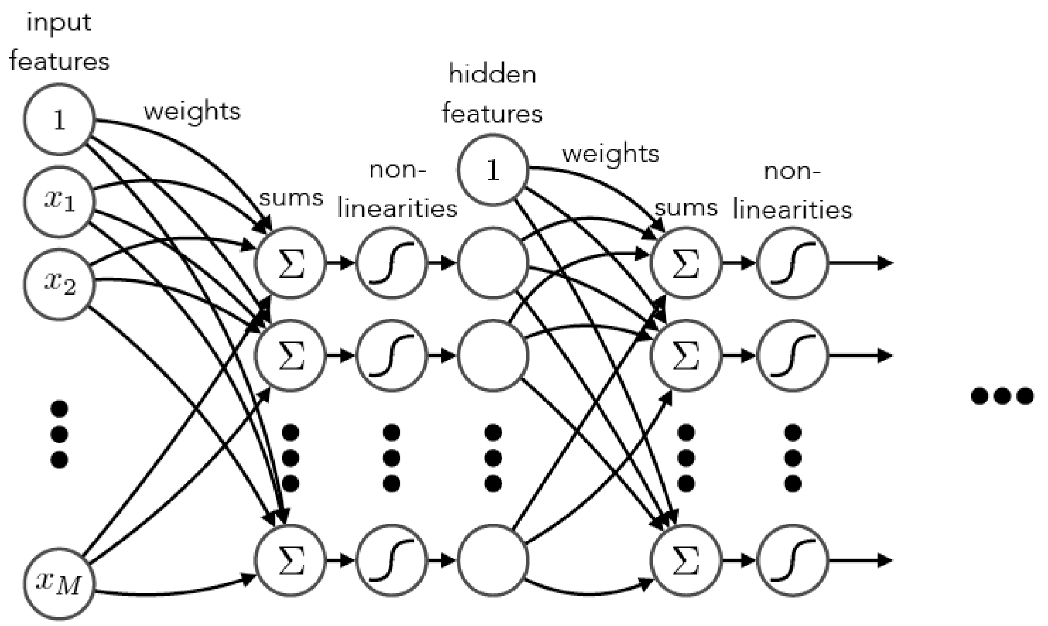


Figure : A 3-layer neural net with 3 input units, 4 hidden units in the first and second hidden layer and 1 output unit

- Naming conventions; a N-layer neural network:
  - $N - 1$ layers of hidden units
  - One output layer

Multilayer networks



network: *sequence of parallelized weighted sums and non-linearities*

DEFINE $\mathbf{x}^{(0)} \equiv \mathbf{x}, \ \mathbf{x}^{(1)} \equiv \mathbf{h},$ ETC.

1st layer

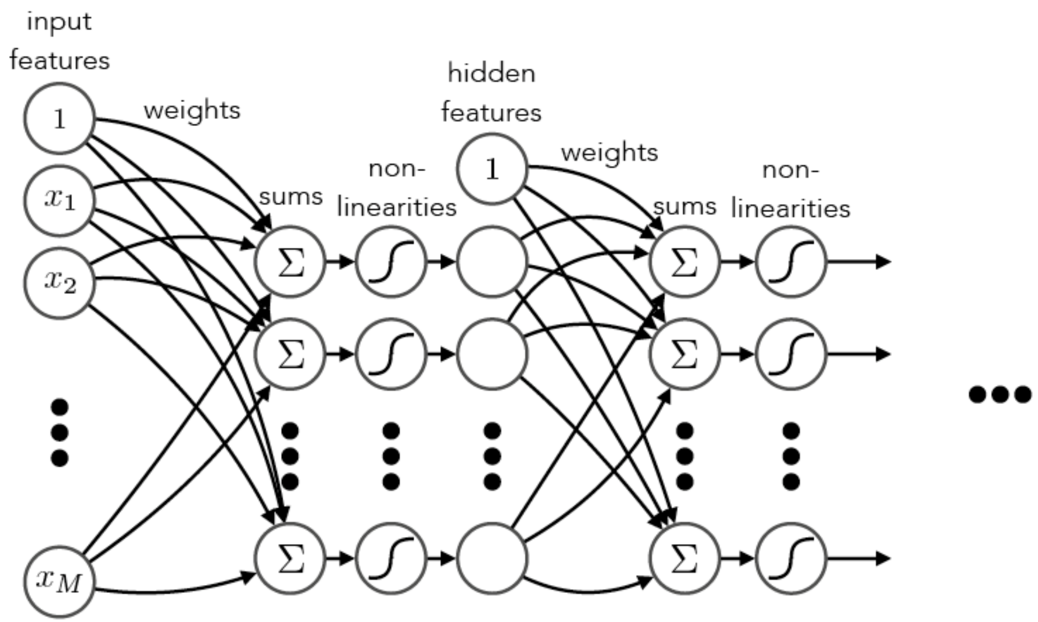
$$\mathbf{s}^{(1)} = \mathbf{W}^{(1)\intercal}\mathbf{x}^{(0)}$$
$$\mathbf{x}^{(1)} = \sigma(\mathbf{s}^{(1)})$$

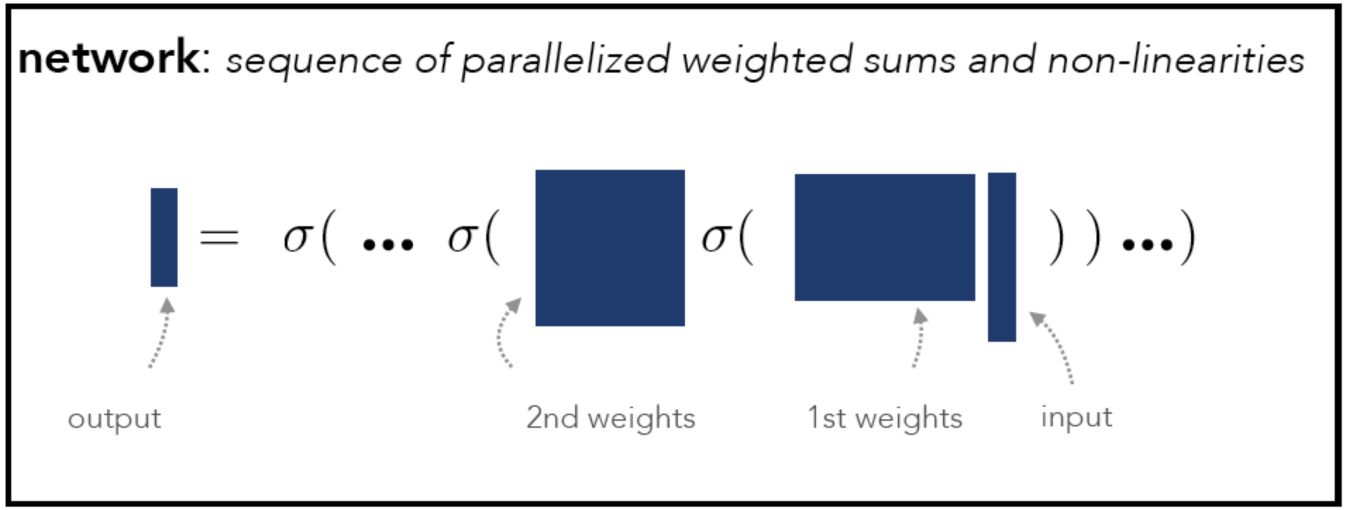
2nd layer

$$\mathbf{s}^{(2)} = \mathbf{W}^{(2)\intercal}\mathbf{x}^{(1)}$$
$$\mathbf{x}^{(2)} = \sigma(\mathbf{s}^{(2)})$$

...

Multilayer networks

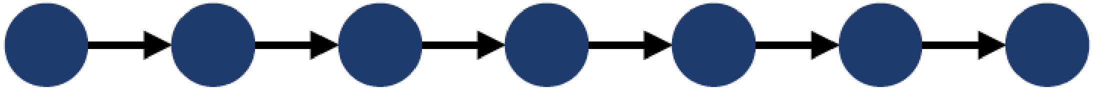




Other network connectivity

上海科技大学 ShanghaiTech University

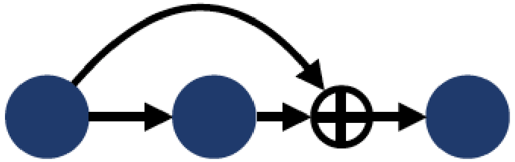
sequential connectivity: *information must flow through the entire sequence to reach the output*



information may not be able to propagate easily

→ *make shorter paths to output*

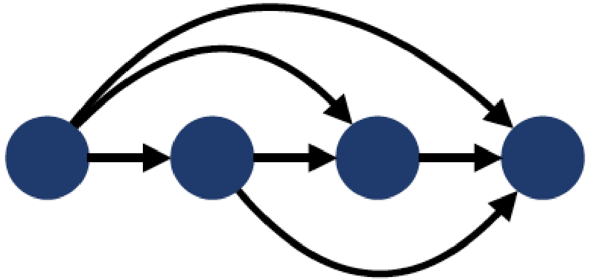
residual & highway connections



Deep residual learning for image recognition, He et al., 2016

Highway networks, Srivastava et al., 2015

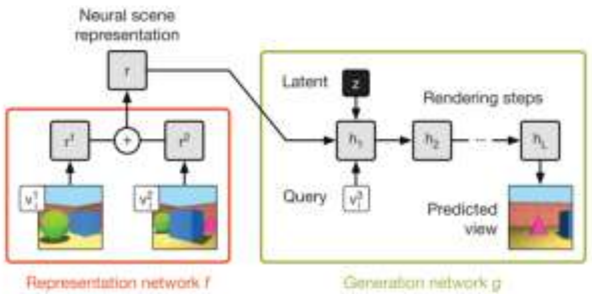
dense (concatenated) connections



Densely connected convolutional networks, Huang et al., 2017

Modern MLP as Implicit Representation

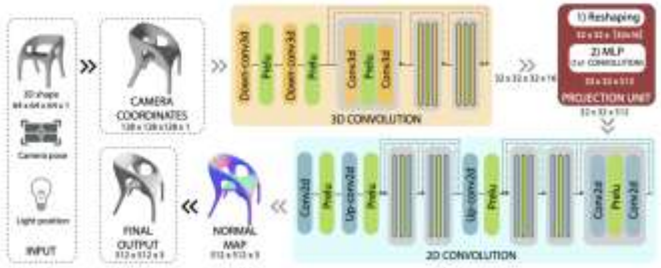
上海科技大学
ShanghaiTech University



Generative Query Networks
[Eslami et al. 2018]

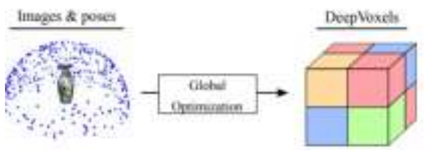
[Flynn et al., 2016; Zhou et al., 2018b;
Mildenhall et al. 2019]

Multiplane Images (MPIs)

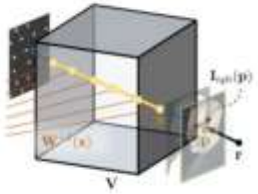


RenderNet [Nguyen-Phuoc et al. 2018]

Voxel Grids + CNN decoder



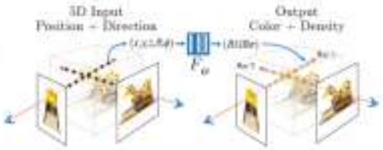
DeepVoxels
[Sitzmann et al. 2019]



Neural Volumes
[Lombardi et al. 2019]

Voxel Grids + Ray Marching

SRN
[Sitzmann et al. 2019b]



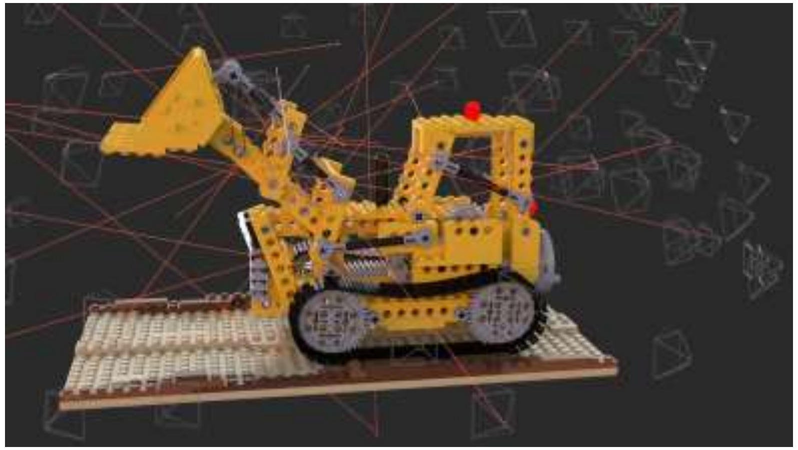
NeRF
[Mildenhall et al. 2020]

IDR
[Yariv et al. 2020]

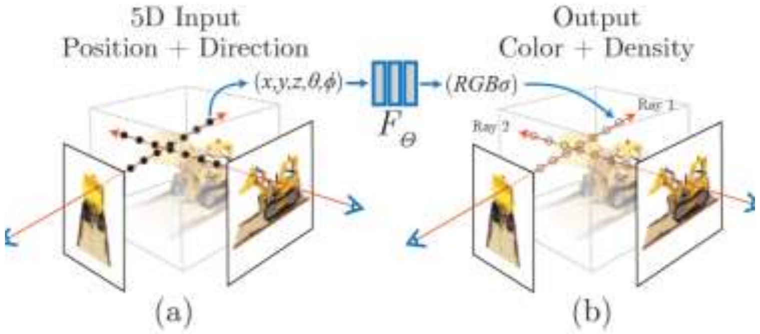
Implicit Fields

Modern MLP in NeRF

- \- Color + Density
- Positional Encoding
- Volume Rendering



Representing Scenes as Neural Radiance Fields for View Synthesis, Mildenhall et al., *ECCV 2020 Oral - Best Paper Honorable Mention*



Outline

- Single layer neural networks
  - Network models; Example: Logistic Regression
- Multi-layer neural networks
  - Limitations of single layer networks
  - Neural networks with single hidden layer
  - Sequential network architecture and variants
- Inference and learning
  - Forward and Backpropagation
  - Examples: one-layer network
  - General BP algorithm

Computation in neural network

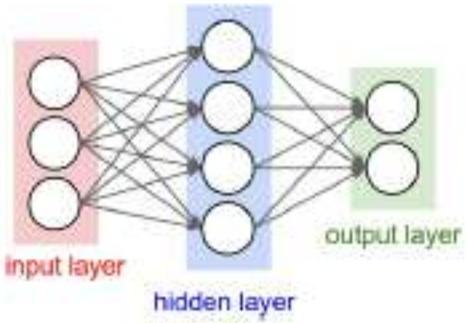
- We only need to know two algorithms
  - Inference/prediction: simply forward pass
  - Parameter learning: needs backward pass
- Basic fact:
  - A neural network is a function of composed operations

$$f_L(\mathbf{w}_L, f_{L-1}(\mathbf{w}_{L-1}, \dots f_1(\mathbf{w}_1, \mathbf{x}) \dots))$$

  - All the f functions are linear + (simple) nonlinear (differentiable a.e.) operators

Inference example: Forward Pass

- What does the network compute?



- Output of the network can be written as:

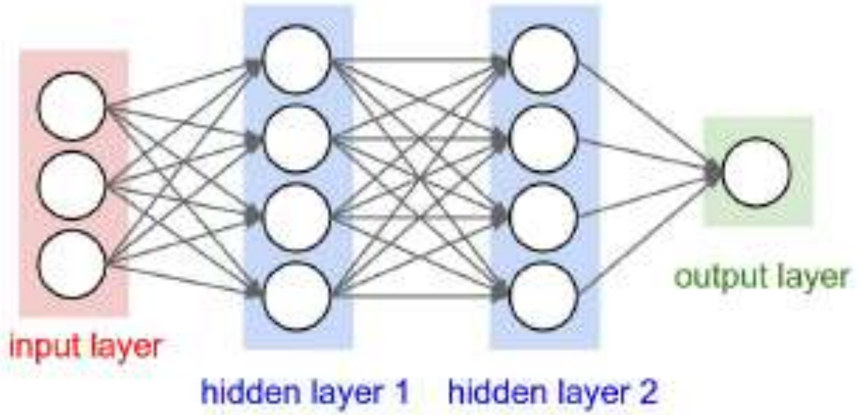
$$
\begin{aligned}
h_j(\mathbf{x}) &= f(v_{j0} + \sum_{i=1}^{D} x_i v_{ji}) \\
o_k(\mathbf{x}) &= g(w_{k0} + \sum_{j=1}^{J} h_j(\mathbf{x}) w_{kj})
\end{aligned}
$$

(j indexing hidden units, k indexing the output units, D number of inputs)

Forward Pass in Python

上海科技大学 ShanghaiTech University

- Example code for a forward pass for a 3-layer network in Python:




```
# forward-pass of a 3-layer neural network:
f = lambda x: 1.0/(1.0 + np.exp(-x)) # activation function (use sigmoid)
x = np.random.randn(3, 1) # random input vector of three numbers (3x1)
h1 = f(np.dot(W1, x) + b1) # calculate first hidden layer activations (4x1)
h2 = f(np.dot(W2, h1) + b2) # calculate second hidden layer activations (4x1)
out = np.dot(W3, h2) + b3 # output neuron (1x1)
```

- Can be implemented efficiently using matrix operations

Parameter learning: Backward Pass

上海科技大学 ShanghaiTech University

- Supervised learning framework

  - Find weights:

$$\mathbf{w}^* = \underset{\mathbf{w}}{\operatorname{argmin}} \sum_{n=1}^{N} \operatorname{loss}(\mathbf{o}^{(n)}, \mathbf{t}^{(n)})$$

    where $\mathbf{o} = f(\mathbf{x}; \mathbf{w})$ is the output of a neural network

  - Define a loss function, eg:
    - Squared loss: $\sum_k \frac{1}{2}(o_k^{(n)} - t_k^{(n)})^2$
    - Cross-entropy loss: $-\sum_k t_k^{(n)} \log o_k^{(n)}$

  - Gradient descent:

$$\mathbf{w}^{t+1} = \mathbf{w}^t - \eta \frac{\partial E}{\partial \mathbf{w}^t}$$

    where η is the learning rate (and E is error/loss)

Gradient descent iteration

上海科技大学 ShanghaiTech University

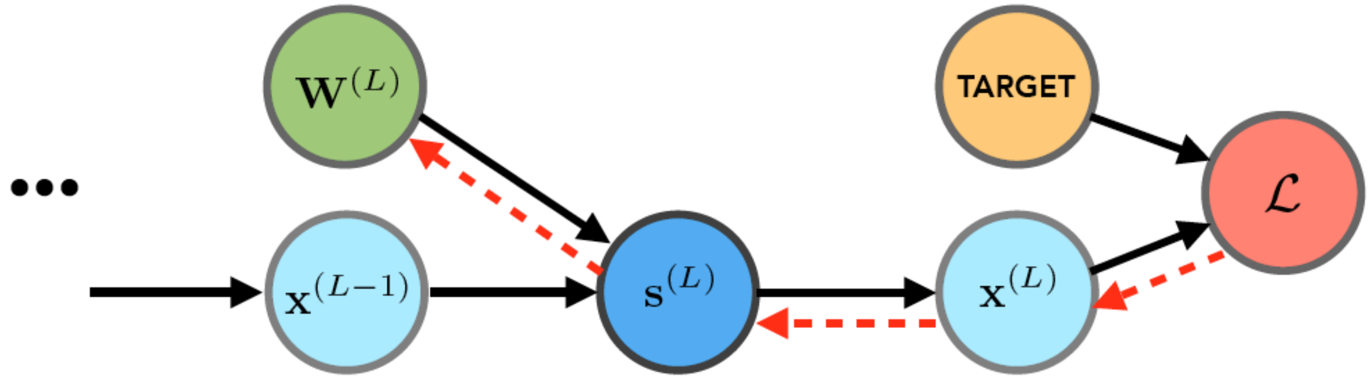
- Forward pass

1st layer	2nd layer		Loss
$\mathbf{s}^{(1)} = \mathbf{W}^{(1)\intercal}\mathbf{x}^{(0)}$ $\mathbf{x}^{(1)} = \sigma(\mathbf{s}^{(1)})$	$\mathbf{s}^{(2)} = \mathbf{W}^{(2)\intercal}\mathbf{x}^{(1)}$ $\mathbf{x}^{(2)} = \sigma(\mathbf{s}^{(2)})$	•••	$\mathcal{L}$

- Backward pass

calculate $\nabla_{W^{(1)}}\mathcal{L}, \nabla_{W^{(2)}}\mathcal{L}, \ldots$ let's start with the final layer: $\nabla_{W^{(L)}}\mathcal{L}$

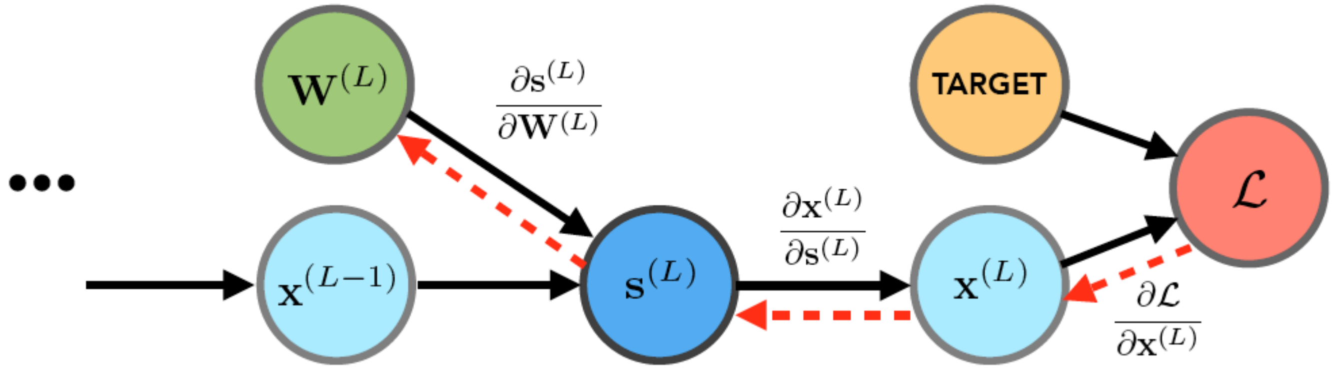
to determine the chain rule ordering, we'll draw the dependency graph



Gradient descent iteration

上海科技大学 ShanghaiTech University

- Backward pass



$$\frac{\partial \mathcal{L}}{\partial \mathbf{W}^{(L)}} = \frac{\partial \mathcal{L}}{\partial \mathbf{x}^{(L)}} \frac{\partial \mathbf{x}^{(L)}}{\partial \mathbf{s}^{(L)}} \frac{\partial \mathbf{s}^{(L)}}{\partial \mathbf{W}^{(L)}}$$

The order needs to be reversed for Jacobians!

depends on the form of the loss

derivative of the non-linearity

$$\frac{\partial}{\partial \mathbf{W}^{(L)}}(\mathbf{W}^{(L)\intercal}\mathbf{x}^{(L-1)}) = \mathbf{x}^{(L-1)\intercal}$$

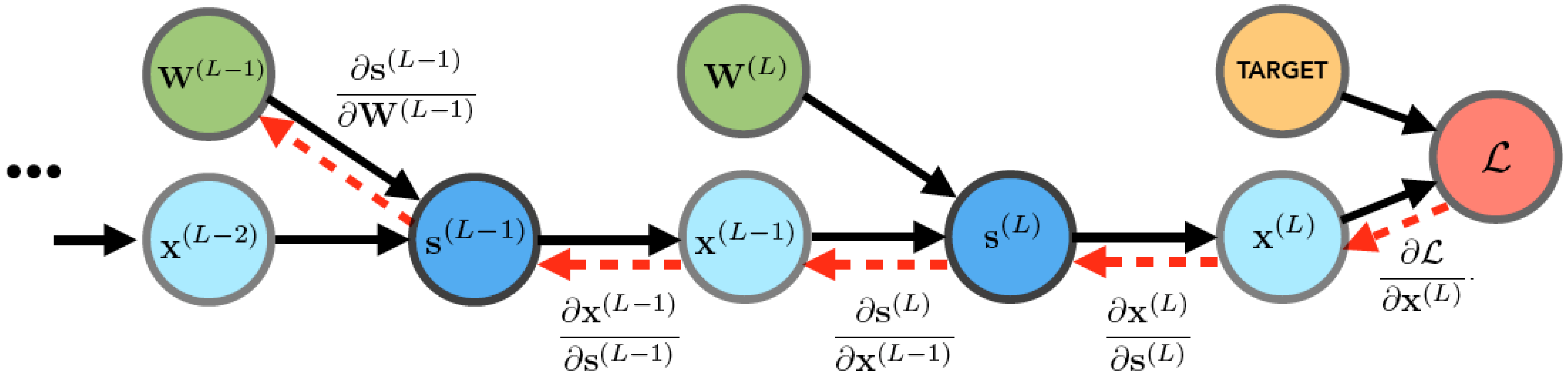
note $\nabla_{\mathbf{W}^{(L)}}\mathcal{L} \equiv \frac{\partial \mathcal{L}}{\partial \mathbf{W}^{(L)}}$ is notational convention

Gradient descent iteration

- Backward pass

now let's go back one more layer...

again we'll draw the dependency graph:

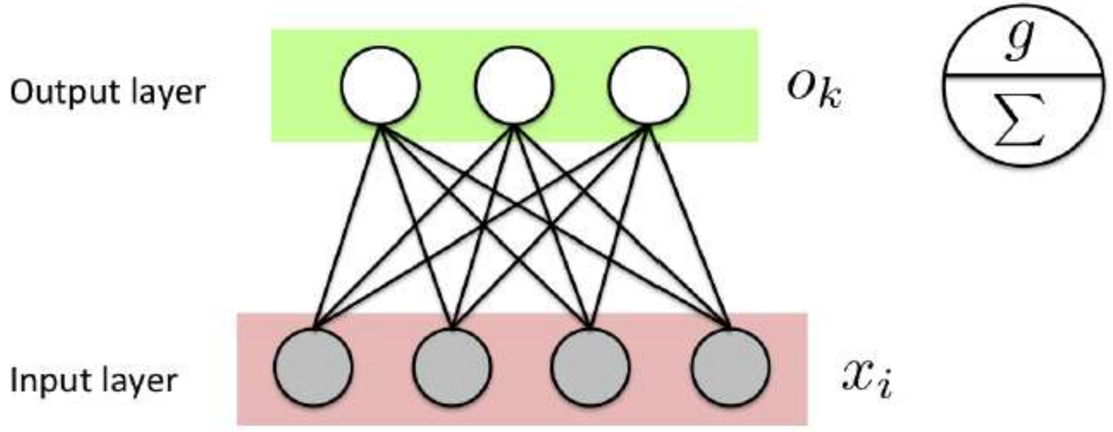


$$\frac{\partial \mathcal{L}}{\partial \mathbf{W}^{(L)}} = \frac{\partial \mathcal{L}}{\partial \mathbf{x}^{(L)}} \frac{\partial \mathbf{x}^{(L)}}{\partial \mathbf{s}^{(L)}} \frac{\partial \mathbf{s}^{(L)}}{\partial \mathbf{x}^{(L-1)}} \frac{\partial \mathbf{x}^{(L-1)}}{\partial \mathbf{s}^{(L-1)}} \frac{\partial \mathbf{s}^{(L-1)}}{\partial \mathbf{W}^{(L-1)}}$$

The order needs to be reversed for Jacobians!

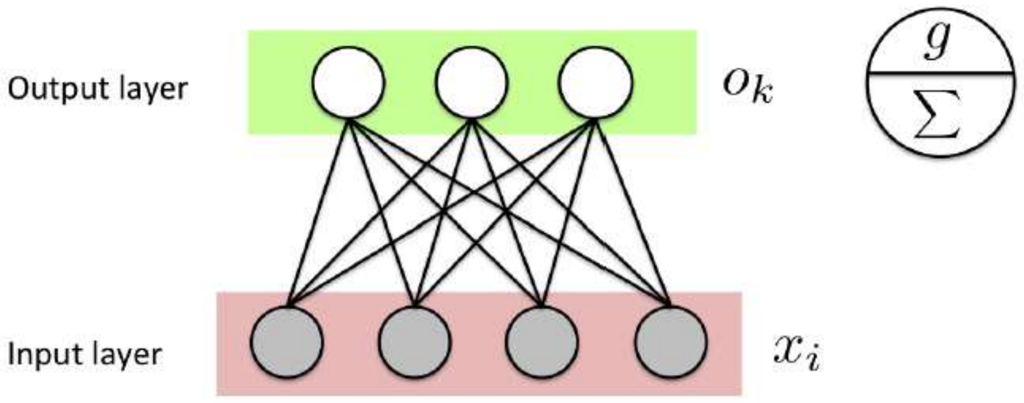
Example: Single Layer Network

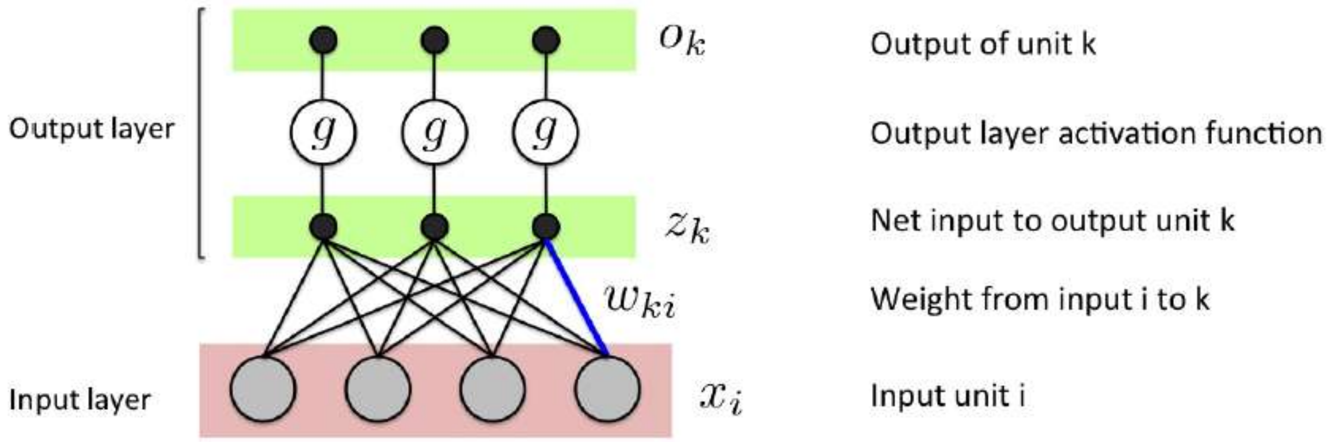
- Let's take a single layer network



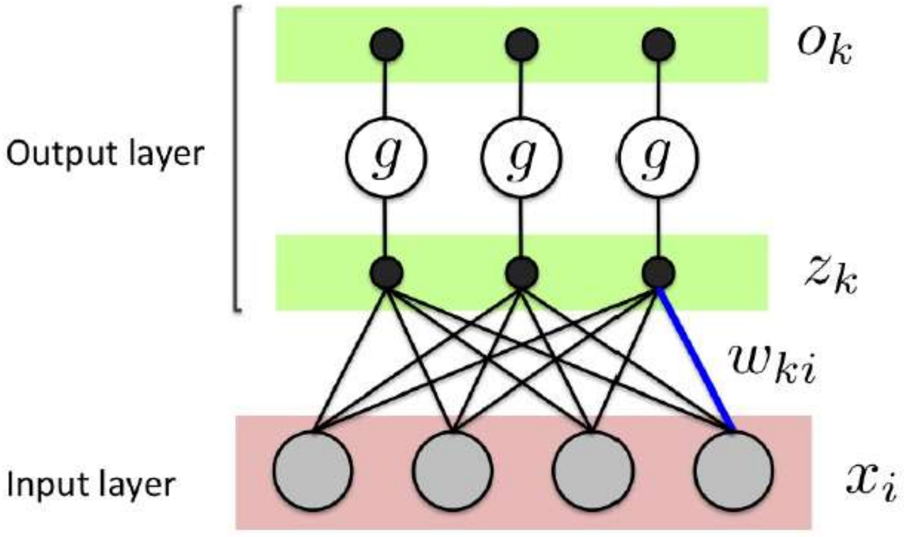
Example: Single Layer Network

- Let's take a single layer network and draw it a bit differently





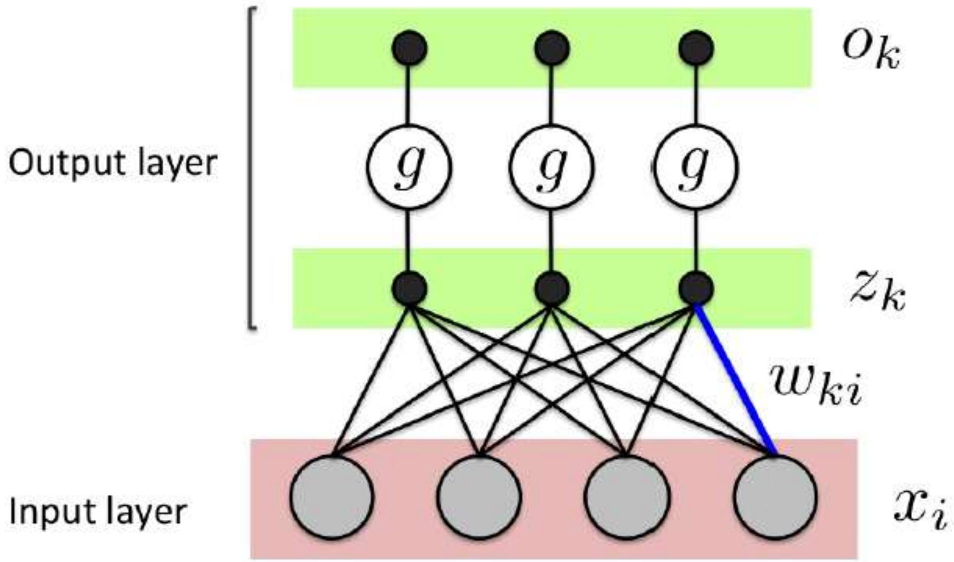
Example: Single Layer Network



- Error gradients for single layer network:

$$\frac{\partial E}{\partial w_{ki}} =$$

Example: Single Layer Network



- Error gradients for single layer network:

$$\frac{\partial E}{\partial w_{ki}} = \frac{\partial E}{\partial o_k} \frac{\partial o_k}{\partial z_k} \frac{\partial z_k}{\partial w_{ki}}$$

- Error gradient is computable for any continuous activation function $g()$, and any continuous error function

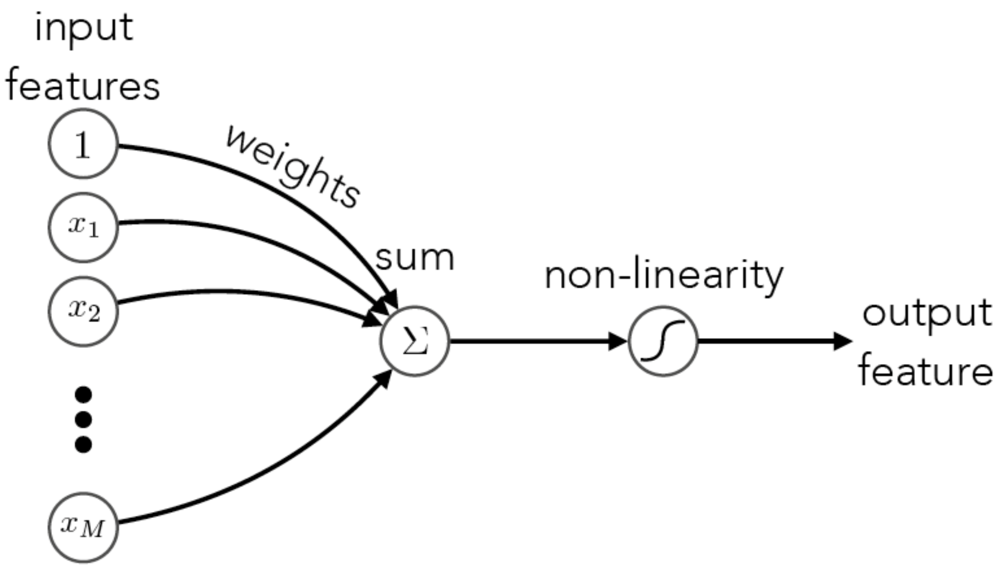
Outline

- Multi-layer neural networks
  - Limitations of single layer networks
  - Neural networks with single hidden layer
  - Sequential network architecture and variants
- Inference and learning
  - Forward and Backpropagation
  - Examples: one-layer network
  - General BP algorithm

An implementation perspective

- Example: Univariate logistic least square model

$$s = wx + b$$
$$y = \sigma(s)$$
$$\mathcal{L} = \frac{1}{2}(y - t)^2$$



Univariate chain rule

- A structured way to implement it
  - The goal is to write a program that efficiently computes the derivatives

Computing the loss:

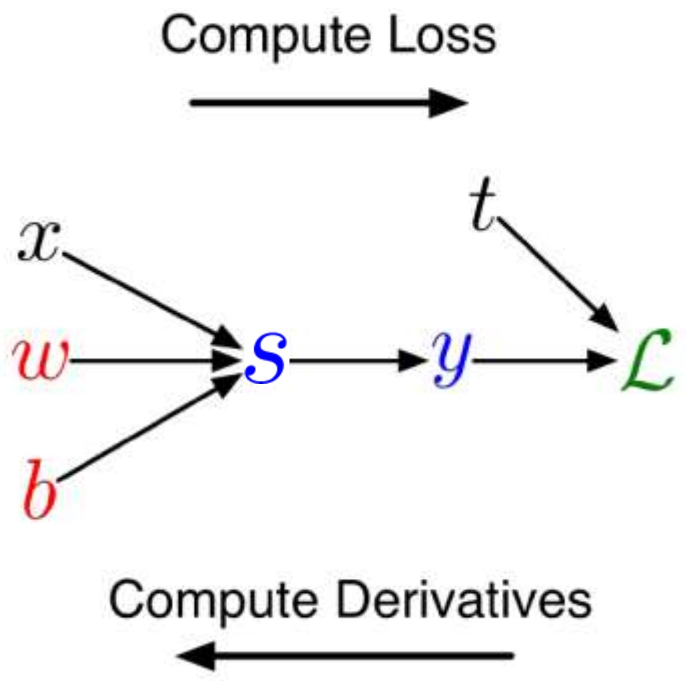
$$s = wx + b$$
$$y = \sigma(s)$$
$$\mathcal{L} = \frac{1}{2}(y - t)^2$$

Computing the derivatives:

$$\frac{d\mathcal{L}}{dy} = y - t$$
$$\frac{d\mathcal{L}}{ds} = \frac{d\mathcal{L}}{dy}\sigma'(s)$$
$$\frac{d\mathcal{L}}{dw} = \frac{d\mathcal{L}}{ds}x$$
$$\frac{d\mathcal{L}}{db} = \frac{d\mathcal{L}}{ds}$$

Computation graph

- Represent the computations using a computation graph
  - Nodes: inputs & computed quantities
  - Edges: which nodes are computed directly as function of which other nodes



Univariate chain rule

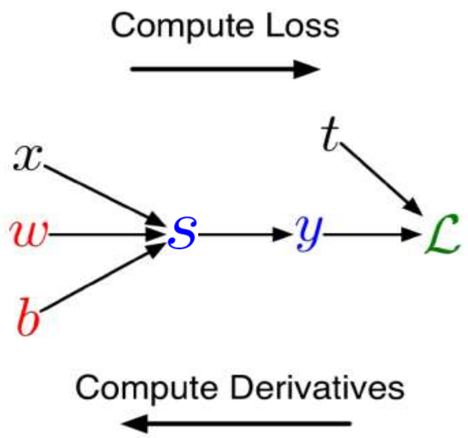
- A shorthand notation
  - Use $\delta_y := d\mathcal{L}/dy$, called the error signal
  - Note that the error signals are values computed by the program

Computing the loss:

$$s = wx + b$$
$$y = \sigma(s)$$
$$\mathcal{L} = \frac{1}{2}(y - t)^2$$

Computing the derivatives:

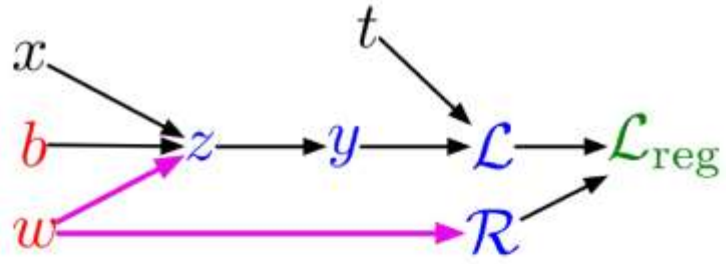
$$\delta_y = y - t$$
$$\delta_s = \delta_y \sigma'(s)$$
$$\delta_w = \delta_s x$$
$$\delta_b = \delta_s$$



Multivariate chain rule

- The computation graph has fan-out > 1

L_2-Regularized regression



$$z = wx + b$$

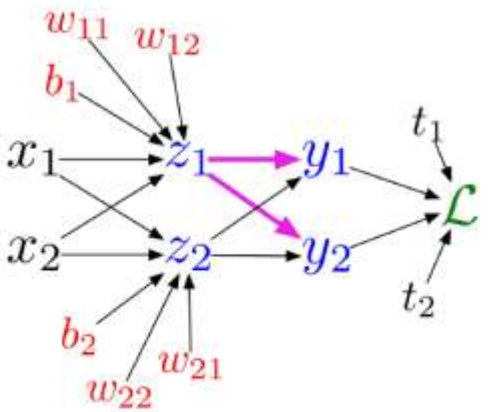
$$y = \sigma(z)$$

$$\mathcal{L} = \frac{1}{2}(y - t)^2$$

$$\mathcal{R} = \frac{1}{2}w^2$$

$$\mathcal{L}_{\mathrm{reg}} = \mathcal{L} + \lambda\mathcal{R}$$

Multiclass logistic regression



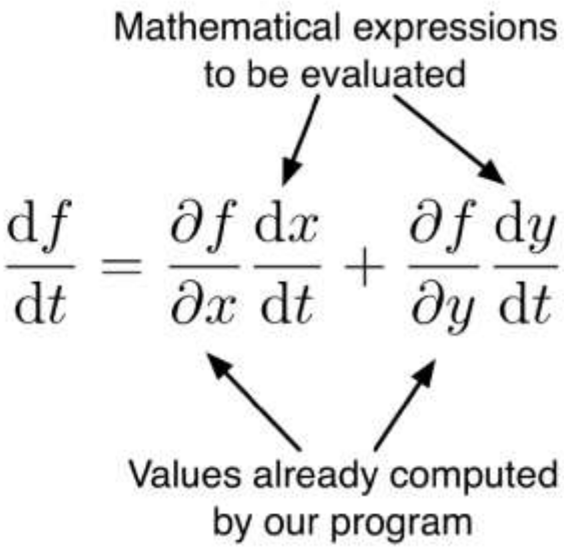
$$z_\ell = \sum_j w_{\ell j} x_j + b_\ell$$

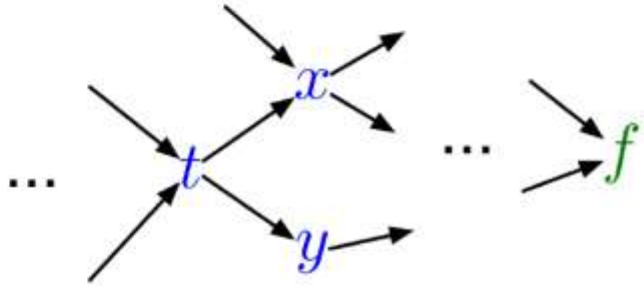
$$y_k = \frac{e^{z_k}}{\sum_\ell e^{z_\ell}}$$

$$\mathcal{L} = -\sum t_k \log y_k$$

Multivariable chain rule

- Recall the distributed chain rule



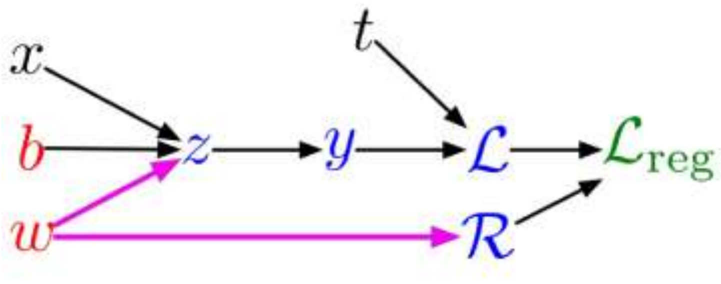


- The shorthand notation:

$$\delta_t = \delta_x \frac{dx}{dt} + \delta_y \frac{dy}{dt}$$

General Backpropagation

- Example: univariate logistic least square regression



Forward pass:

$$
\begin{aligned}
z &= wx + b \\
y &= \sigma(z) \\
\mathcal{L} &= \frac{1}{2}(y - t)^2 \\
\mathcal{R} &= \frac{1}{2}w^2 \\
\mathcal{L}_{\text{reg}} &= \mathcal{L} + \lambda\mathcal{R}
\end{aligned}
$$

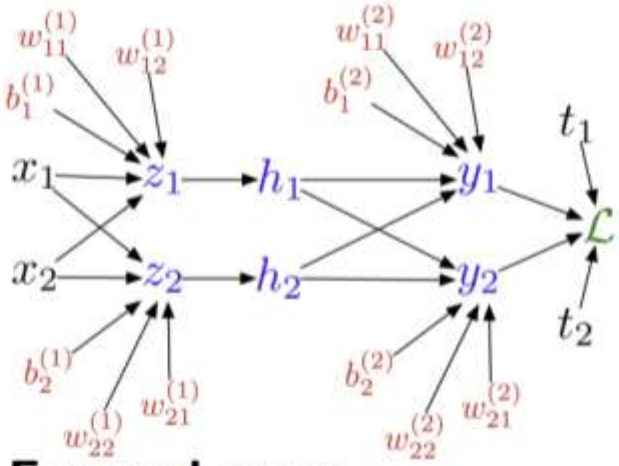
Backward pass:

$$
\begin{aligned}
\delta_{\mathcal{L}_{\text{reg}}} &= 1 \\
\delta_{\mathcal{R}} &= \delta_{\mathcal{L}_{\text{reg}}} \frac{d\mathcal{L}_{\text{reg}}}{d\mathcal{R}} \\
&= \delta_{\mathcal{L}_{\text{reg}}} \lambda \\
\delta_{\mathcal{L}} &= \delta_{\mathcal{L}_{\text{reg}}} \frac{d\mathcal{L}_{\text{reg}}}{d\mathcal{L}} \\
&= \delta_{\mathcal{L}_{\text{reg}}} \\
\delta y &= \delta_{\mathcal{L}} \frac{d\mathcal{L}}{dy} \\
&= \delta_{\mathcal{L}} (y - t)
\end{aligned}
$$

$$
\begin{aligned}
\delta_z &= \delta_y \frac{dy}{dz} \\
&= \delta_y \sigma'(z) \\
\delta_w &= \delta_z \frac{dz}{dw} + \delta_{\mathcal{R}} \frac{d\mathcal{R}}{dw} \\
&= \delta_z x + \delta_{\mathcal{R}} w \\
\delta_b &= \delta_z \frac{dz}{db} \\
&= \delta_z
\end{aligned}
$$

General Backpropagation

- Example: Multilayer Perceptron (multiple outputs)



Forward pass:

$$z_i = \sum_j w_{ij}^{(1)} x_j + b_i^{(1)}$$

$$h_i = \sigma(z_i)$$

$$y_k = \sum_i w_{ki}^{(2)} h_i + b_k^{(2)}$$

$$\mathcal{L} = \frac{1}{2}\sum_k (y_k - t_k)^2$$

Backward pass:

$$\overline{\mathcal{L}} = 1$$

$$\overline{y_k} = \overline{\mathcal{L}}\,(y_k - t_k)$$

$$\overline{w_{ki}^{(2)}} = \overline{y_k}\, h_i$$

$$\overline{b_k^{(2)}} = \overline{y_k}$$

$$\overline{h_i} = \sum_k \overline{y_k} w_{ki}^{(2)}$$

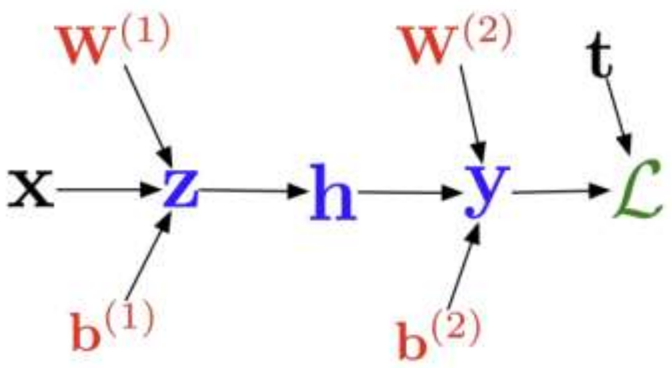
$$\overline{z_i} = \overline{h_i}\, \sigma'(z_i)$$

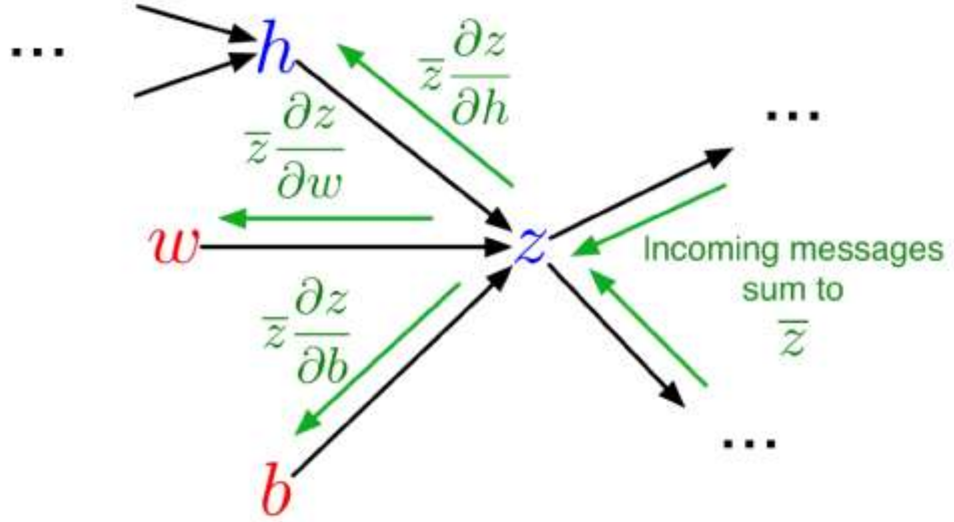
$$\overline{w_{ij}^{(1)}} = \overline{z_i}\, x_j$$

$$\overline{b_i^{(1)}} = \overline{z_i}$$

General Backpropagation

- Backprop as message passing:

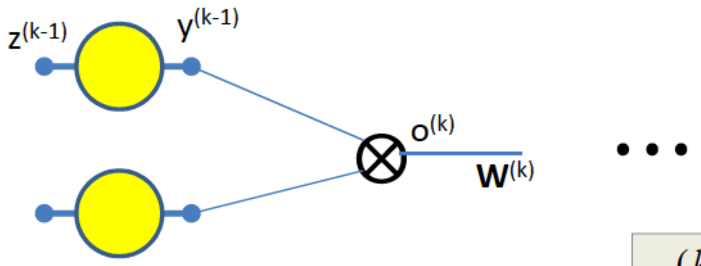




  - Each node receives a set of messages from its children, which are aggregated into its error signal, then it passes messages to its parents
  - Modularity: each node only has to know how to compute derivatives w.r.t. its arguments – local computation in the graph

Patterns in backward flow

- Multiplicative node



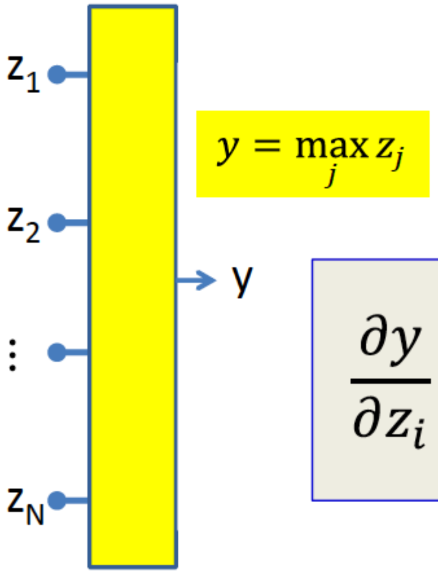
Forward: $o_i^{(k)} = y_j^{(k-1)} y_l^{(k-1)}$

$$\frac{\partial L}{\partial y_j^{(k-1)}} = \frac{\partial L}{\partial o_i^{(k)}} \frac{\partial o_i^{(k)}}{\partial y_j^{(k-1)}} = y_l^{(k-1)} \frac{\partial L}{\partial o_i^{(k)}}$$

Patterns in backward flow

上海科技大学 ShanghaiTech University

- Max node



$$\frac{\partial y}{\partial z_i} = \begin{cases} 1, & i = \operatorname{argmax}_j z_j \\ 0, & otherwise \end{cases}$$

- Vector equivalent of subgradient
  - 1 w.r.t. the largest incoming input
    - Incremental changes in this input will change the output
  - 0 for the rest
    - Incremental changes to these inputs will not change the output

Differentiation Quiz

Speed Quiz:
2 minute time limit.

Differentiation Quiz #1:

Suppose x = 2 and z = 3, what are dy/dx and dy/dz for the function below? **Round your answer to the nearest integer**.

$$y = \exp(xz) + \frac{xz}{\log(x)} + \frac{\sin(\log(x))}{xz}$$

Answer: *Answers below are in the form [dy/dx, dy/dz]*

A. [42, -72]
B. [72, -42]
C. [100, 127]
D. [127, 100]
E. [1208, 810]
F. [810, 1208]
G. [1505, 94]
H. [94, 1505]

Algorithm

BACKPROPAGATION FOR BINARY LOGISTIC REGRESSION

Derivative of a Sigmoid

First suppose that

$$s = \frac{1}{1 + \exp(-b)} \tag{1}$$

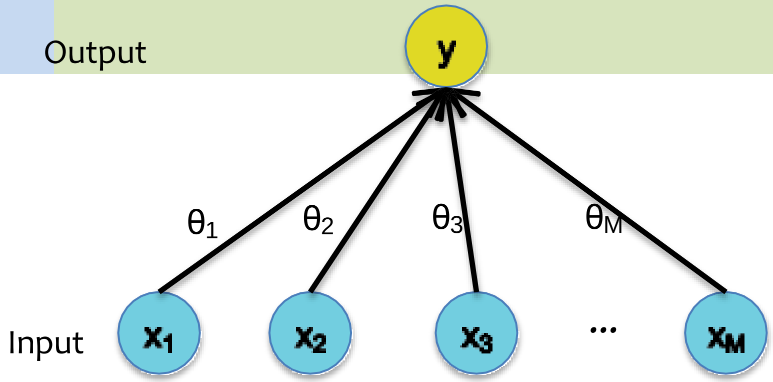
To obtain the simplified form of the derivative of a sigmoid.

$$\begin{aligned}
\frac{ds}{db} &= \frac{\exp(-b)}{(\exp(-b)+1)^2} && (2)\\
&= \frac{\exp(-b)+1-1}{(\exp(-b)+1+1-1)^2} && (3)\\
&= \frac{\exp(-b)+1-1}{(\exp(-b)+1)^2} && (4)\\
&= \frac{\exp(-b)+1}{(\exp(-b)+1)^2} - \frac{1}{(\exp(-b)+1)^2} && (5)\\
&= \frac{1}{(\exp(-b)+1)} - \frac{1}{(\exp(-b)+1)^2} && (6)\\
&= \frac{1}{(\exp(-b)+1)} - \left(\frac{1}{(\exp(-b)+1)}\frac{1}{(\exp(-b)+1)}\right) && (7)\\
&= \frac{1}{(\exp(-b)+1)}\left(1 - \frac{1}{(\exp(-b)+1)}\right) && (8)\\
&= s(1-s) && (9)
\end{aligned}$$

Training

Backpropagation

Case 1: Logistic Regression



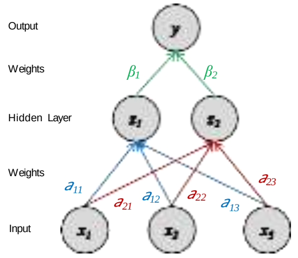
Question: How do we compute this?
Answer:

Computation Graph

Forward	Backward
$J = y^* \log y + (1 - y^*) \log(1 - y)$	$g_y = \frac{y^*}{y} + \frac{(1 - y^*)}{y - 1}$
$y = \frac{1}{1 + \exp(-a)}$	$g_a = g_y \frac{\partial y}{\partial a}, \quad \frac{\partial y}{\partial a} = \frac{\exp(-a)}{(\exp(-a) + 1)^2}$
$a = \sum_{j=0}^{D} \theta_j x_j$	$g_{\theta_j} = \boxed{}$ $g_{x_j} =$

Case 2: Neural Network

Backpropagation



	Forward	Backward
Loss	$J = y^* \log y + (1 - y^*) \log(1 - y)$	$\frac{dJ}{dy} = \frac{y^*}{y} + \frac{(1 - y^*)}{y - 1}$
Sigmoid	$y = \frac{1}{1 + \exp(-b)}$	$\frac{dJ}{db} = \frac{dJ}{dy}\frac{dy}{db}, \quad \frac{dy}{db} = \frac{\exp(-b)}{(\exp(-b) + 1)^2}$
Linear	$b = \sum_{j=0}^{D} \beta_j z_j$	$\frac{dJ}{d\beta_j} = \frac{dJ}{db}\frac{db}{d\beta_j}, \quad \frac{db}{d\beta_j} = z_j$ $\frac{dJ}{dz_j} = \frac{dJ}{db}\frac{db}{dz_j}, \quad \frac{db}{dz_j} = \beta_j$
Sigmoid	$z_j = \frac{1}{1 + \exp(-a_j)}$	$\frac{dJ}{da_j} = \frac{dJ}{dz_j}\frac{dz_j}{da_j}, \quad \frac{dz_j}{da_j} = \frac{\exp(-a_j)}{(\exp(-a_j) + 1)^2}$
Linear	$a_j = \sum_{i=0}^{M} \alpha_{ji} x_i$	$\frac{dJ}{d\alpha_{ji}} = \frac{dJ}{da_j}\frac{da_j}{d\alpha_{ji}}, \quad \frac{da_j}{d\alpha_{ji}} = x_i$ $\frac{dJ}{dx_i} = \sum_{j=0}^{D} \frac{dJ}{da_j}\frac{da_j}{dx_i}, \quad \frac{da_j}{dx_i} = \alpha_{ji}$

MATRIX CALCULUS

Matrix Calculus

Let $y, x \in \mathbb{R}$ be scalars, $\mathbf{y} \in \mathbb{R}^M$ and $\mathbf{x} \in \mathbb{R}^P$ be vectors, and $\mathbf{Y} \in \mathbb{R}^{M \times N}$ and $\mathbf{X} \in \mathbb{R}^{P \times Q}$ be matrices

Numerator (columns) / *Denominator* (rows)

Types of Derivatives	**scalar**	**vector**	**matrix**
scalar	$\frac{\partial y}{\partial x}$	$\frac{\partial \mathbf{y}}{\partial x}$	$\frac{\partial \mathbf{Y}}{\partial x}$
vector	$\frac{\partial y}{\partial \mathbf{x}}$	$\frac{\partial \mathbf{y}}{\partial \mathbf{x}}$	$\frac{\partial \mathbf{Y}}{\partial \mathbf{x}}$
matrix	$\frac{\partial y}{\partial \mathbf{X}}$	$\frac{\partial \mathbf{y}}{\partial \mathbf{X}}$	$\frac{\partial \mathbf{Y}}{\partial \mathbf{X}}$

Matrix Calculus

上海科技大学
ShanghaiTech University

Types of Derivatives	**scalar**
scalar	$\frac{\partial y}{\partial x} = \left[\frac{\partial y}{\partial x}\right]$
vector	$\frac{\partial y}{\partial \mathbf{x}} = \begin{bmatrix} \frac{\partial y}{\partial x_1} \\ \frac{\partial y}{\partial x_2} \\ \vdots \\ \frac{\partial y}{\partial x_P} \end{bmatrix}$
matrix	$\frac{\partial y}{\partial \mathbf{X}} = \begin{bmatrix} \frac{\partial y}{\partial X_{11}} & \frac{\partial y}{\partial X_{12}} & \cdots & \frac{\partial y}{\partial X_{1Q}} \\ \frac{\partial y}{\partial X_{21}} & \frac{\partial y}{\partial X_{22}} & \cdots & \frac{\partial y}{\partial X_{2Q}} \\ \vdots & & & \vdots \\ \frac{\partial y}{\partial X_{P1}} & \frac{\partial y}{\partial X_{P2}} & \cdots & \frac{\partial y}{\partial X_{PQ}} \end{bmatrix}$

Matrix Calculus

上海科技大学 ShanghaiTech University

Types of Derivatives	**scalar**	**vector**
scalar	$\frac{\partial y}{\partial x} = \left[\frac{\partial y}{\partial x}\right]$	$\frac{\partial \mathbf{y}}{\partial x} = \left[\frac{\partial y_1}{\partial x} \quad \frac{\partial y_2}{\partial x} \quad \cdots \quad \frac{\partial y_N}{\partial x}\right]$
vector	$\frac{\partial y}{\partial \mathbf{x}} = \begin{bmatrix} \frac{\partial y}{\partial x_1} \\ \frac{\partial y}{\partial x_2} \\ \vdots \\ \frac{\partial y}{\partial x_P} \end{bmatrix}$	$\frac{\partial \mathbf{y}}{\partial \mathbf{x}} = \begin{bmatrix} \frac{\partial y_1}{\partial x_1} & \frac{\partial y_2}{\partial x_1} & \cdots & \frac{\partial y_N}{\partial x_1} \\ \frac{\partial y_1}{\partial x_2} & \frac{\partial y_2}{\partial x_2} & \cdots & \frac{\partial y_N}{\partial x_2} \\ \vdots & & & \\ \frac{\partial y_1}{\partial x_P} & \frac{\partial y_2}{\partial x_P} & \cdots & \frac{\partial y_N}{\partial x_P} \end{bmatrix}$

Matrix Calculus

Whenever you read about matrix calculus, you'll be confronted with two layout conventions:

Let $y, x \in \mathbb{R}$ be scalars, $\mathbf{y} \in \mathbb{R}^M$ and $\mathbf{x} \in \mathbb{R}^P$ be vectors.

1. In numerator layout:

$$\frac{\partial y}{\partial \mathbf{x}} \text{ is a } 1 \times P \text{ matrix, i.e. a row vector}$$

$$\frac{\partial \mathbf{y}}{\partial \mathbf{x}} \text{ is an } M \times P \text{ matrix}$$

2. In denominator layout:

$$\frac{\partial y}{\partial \mathbf{x}} \text{ is a } P \times 1 \text{ matrix, i.e. a column vector}$$

$$\frac{\partial \mathbf{y}}{\partial \mathbf{x}} \text{ is an } P \times M \text{ matrix}$$

In this course, **we use denominator layout.**

Why? This ensures that our gradients of the objective function with respect to some subset of parameters are the same shape as those parameters.

Vector Derivatives

上海科技大学
ShanghaiTech University

Scalar Derivatives

Suppose $x \in \mathbb{R}$
and $f : \mathbb{R} \to \mathbb{R}$

$f(x)$	$\frac{\partial f(x)}{\partial x}$
bx	b
xb	b
x^2	$2x$
bx^2	$2bx$

Vector Derivatives

Suppose $\mathbf{x} \in \mathbb{R}^m$, $\mathbf{b} \in \mathbb{R}^m$,
$\mathbf{B} \in \mathbb{R}^{m \times n}$, $\mathbf{Q} \in \mathbb{R}^{m \times m}$
and $\mathbf{Q}$ is symmetric.

$f(\mathbf{x})$	$\frac{\partial f(\mathbf{x})}{\partial \mathbf{x}}$	type of f
$\mathbf{b}^T\mathbf{x}$	$\mathbf{b}$	$f : \mathbb{R}^m \to \mathbb{R}$
$\mathbf{x}^T\mathbf{b}$	$\mathbf{b}$	$f : \mathbb{R}^m \to \mathbb{R}$
$\mathbf{x}^T\mathbf{B}$	$\mathbf{B}$	$f : \mathbb{R}^m \to \mathbb{R}^n$
$\mathbf{B}^T\mathbf{x}$	$\mathbf{B}$	$f : \mathbb{R}^m \to \mathbb{R}^n$
$\mathbf{x}^T\mathbf{x}$	$2\mathbf{x}$	$f : \mathbb{R}^m \to \mathbb{R}$
$\mathbf{x}^T\mathbf{Q}\mathbf{x}$	$2\mathbf{Q}\mathbf{x}$	$f : \mathbb{R}^m \to \mathbb{R}$

Vector Derivatives

Scalar Derivatives

Suppose $x \in \mathrm{R}^m$ and we have constants $a \in \mathrm{R}$, $b \in \mathrm{R}$

$f(x)$	$\frac{\partial f(x)}{\partial x}$
$g(x) + h(x)$	$\frac{\partial g(x)}{\partial x} + \frac{\partial h(x)}{\partial x}$
$ag(x)$	$a\frac{\partial g(x)}{\partial x}$
$g(x)b$	$\frac{\partial g(x)}{\partial x}b$

Vector Derivatives

Suppose $\mathrm{x} \in \mathrm{R}^m$ and we have constants $\mathrm{a} \in \mathrm{R}$, $\mathrm{b} \in \mathrm{R}^n$

$f(\mathrm{x})$	$\frac{\partial f(\mathrm{x})}{\partial \mathrm{x}}$
$g(\mathrm{x}) + h(\mathrm{x})$	$\frac{\partial \mathbf{g}(\mathrm{x})}{\partial \mathrm{x}} + \frac{\partial \mathbf{h}(\mathrm{x})}{\partial \mathrm{x}}$
$ag(\mathrm{x})$	$a\frac{\partial \mathbf{g}(\mathrm{x})}{\partial \mathrm{x}}$
$g(\mathrm{x})\mathrm{b}$	$\frac{\partial \mathbf{g}(\mathrm{x})}{\partial \mathrm{x}}\mathrm{b}^T$

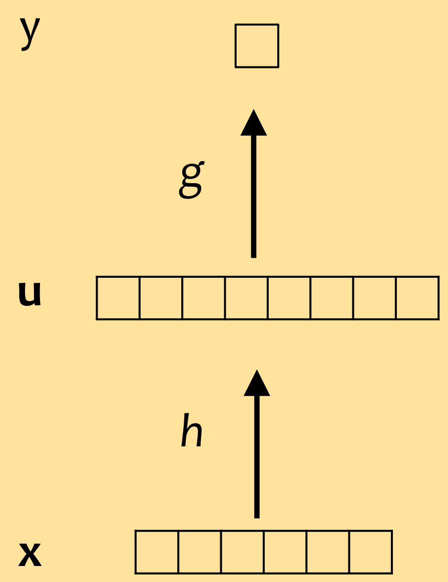
Matrix Calculus

Recall:

$$\frac{\partial y}{\partial \mathbf{x}} = \begin{bmatrix} \frac{\partial y}{\partial x_1} \\ \frac{\partial y}{\partial x_2} \\ \vdots \\ \frac{\partial y}{\partial x_P} \end{bmatrix} \qquad \frac{\partial \mathbf{y}}{\partial \mathbf{x}} = \begin{bmatrix} \frac{\partial y_1}{\partial x_1} & \frac{\partial y_2}{\partial x_1} & \dots & \frac{\partial y_N}{\partial x_1} \\ \frac{\partial y_1}{\partial x_2} & \frac{\partial y_2}{\partial x_2} & \dots & \frac{\partial y_N}{\partial x_2} \\ \vdots & & & \\ \frac{\partial y_1}{\partial x_P} & \frac{\partial y_2}{\partial x_P} & \dots & \frac{\partial y_N}{\partial x_P} \end{bmatrix}$$

Question:

Suppose y = g(**u**) and **u** = h(**x**)



Which of the following is the correct definition of the chain rule?

Answer:

$$\frac{\partial y}{\partial \mathbf{x}} = \dots$$

A. $\frac{\partial y}{\partial \mathbf{u}} \frac{\partial \mathbf{u}}{\partial \mathbf{x}}$

B. $\frac{\partial y}{\partial \mathbf{u}}^T \frac{\partial \mathbf{u}}{\partial \mathbf{x}}$

C. $\frac{\partial y}{\partial \mathbf{u}} \frac{\partial \mathbf{u}}{\partial \mathbf{x}}^T$

D. $\frac{\partial y}{\partial \mathbf{u}}^T \frac{\partial \mathbf{u}}{\partial \mathbf{x}}^T$

E. $(\frac{\partial y}{\partial \mathbf{u}} \frac{\partial \mathbf{u}}{\partial \mathbf{x}})^T$

F. None of the above

Gradient Descent for Neural Network Training

- Input: $\mathcal{D} = \{(\boldsymbol{x}^{(n)}, y^{(n)})\}_{n=1}^{N}, \eta^{(0)}$
- Initialize all weights $W_{(0)}^{(1)}, \ldots, W_{(0)}^{(L)}$ to small, random numbers and set $t = 0$ (???)
- While TERMINATION CRITERION is not satisfied (???)
  - For $l = 1, \ldots, L$
    - Compute $G^{(l)} = \nabla_{W^{(l)}} \ell_{\mathcal{D}}\left(W_{(t)}^{(1)}, \ldots, W_{(t)}^{(L)}\right)$ (???)
    - Update $W^{(l)}$: $W_{(t+1)}^{(l)} = W_{(t)}^{(l)} - \eta_0 G^{(l)}$
  - Increment t: $t = t + 1$
- Output: $W_{(t)}^{(1)}, \ldots, W_{(t)}^{(L)}$

Computing Gradients

$$\ell_{\mathcal{D}}\left(W_{(t)}^{(1)}, \dots, W_{(t)}^{(L)}\right) = \sum_{n=1}^{N} \ell^{(n)}\left(W_{(t)}^{(1)}, \dots, W_{(t)}^{(L)}\right)$$

$$\nabla_{W^{(l)}} \ell_{\mathcal{D}}\left(W_{(t)}^{(1)}, \dots, W_{(t)}^{(L)}\right)$$

$$= \begin{bmatrix} \frac{\partial \ell_{\mathcal{D}}}{\partial w_{1,0}^{(l)}} & \frac{\partial \ell_{\mathcal{D}}}{\partial w_{1,1}^{(l)}} & \cdots & \frac{\partial \ell_{\mathcal{D}}}{\partial w_{1,d^{(l-1)}}^{(l)}} \\ \frac{\partial \ell_{\mathcal{D}}}{\partial w_{2,0}^{(l)}} & \frac{\partial \ell_{\mathcal{D}}}{\partial w_{2,1}^{(l)}} & \cdots & \frac{\partial \ell_{\mathcal{D}}}{\partial w_{2,d^{(l-1)}}^{(l)}} \\ \vdots & \vdots & \ddots & \vdots \\ \frac{\partial \ell_{\mathcal{D}}}{\partial w_{d^{(l)},0}^{(l)}} & \frac{\partial \ell_{\mathcal{D}}}{\partial w_{d^{(l)},1}^{(l)}} & \cdots & \frac{\partial \ell_{\mathcal{D}}}{\partial w_{d^{(l)},d^{(l-1)}}^{(l)}} \end{bmatrix}$$

$$\frac{\partial \ell_{\mathcal{D}}}{\partial w_{b,a}^{(l)}} = \sum_{n=1}^{N} \frac{\partial \ell^{(n)}\left(W_{(t)}^{(1)}, \dots, W_{(t)}^{(L)}\right)}{\partial w_{b,a}^{(l)}}$$

Finite Difference Method

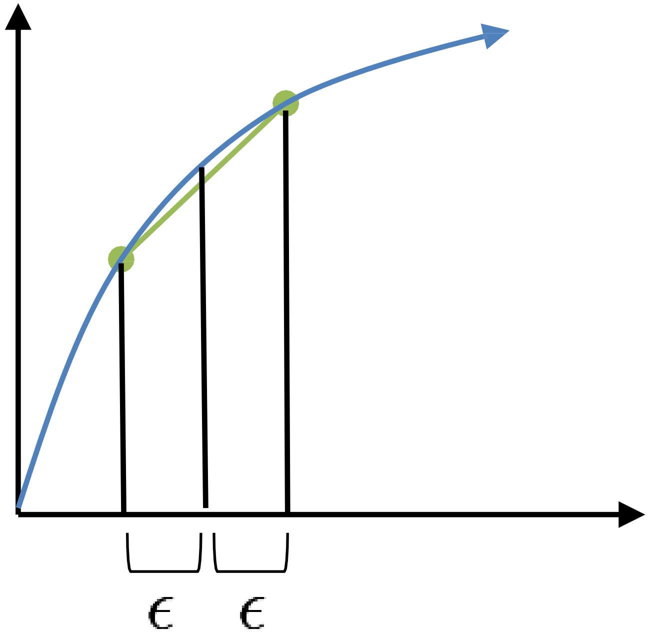
The *centered* finite difference approximation is:

$$\frac{\partial}{\partial \theta_i} J(\boldsymbol{\theta}) \approx \frac{(J(\boldsymbol{\theta} + \epsilon \cdot \boldsymbol{d}_i) - J(\boldsymbol{\theta} - \epsilon \cdot \boldsymbol{d}_i))}{2\epsilon} \tag{1}$$

where $\boldsymbol{d}_i$ is a 1-hot vector consisting of all zeros except for the ith entry of $\boldsymbol{d}_i$, which has value 1.

Notes:

- Suffers from issues of floating point precision, in practice
- Typically only appropriate to use on small examples with an appropriately chosen epsilon



Summary

1. **Neural Networks…**
   - provide a way of learning features
   - are highly nonlinear prediction functions
   - (can be) a highly parallel network of logistic regression classifiers
   - discover useful hidden representations of the input
2. **Backpropagation…**
   - Backprop is used to train the majority of neural nets
   - Even generative network learning, or advanced optimization algorithms (second-order) use backprop to compute the update of weights

Summary

上海科技大学 ShanghaiTech University

1. **Neural Networks…**
   - provide a way of learning features
   - are highly nonlinear prediction functions
   - (can be) a highly parallel network of logistic regression classifiers
   - discover useful hidden representations of the input
2. **Backpropagation…**
   - However, backprop seems biologically implausible
   - No evidence for biological signals analogous to error derivatives
   - All the existing biologically plausible alternatives learn much more slowly on computers.

Backprop Objectives

上海科技大学
ShanghaiTech University

- You should be able to…

- Differentiate between a neural network diagram and a computation graph
- Construct a computation graph for a function as specified by an algorithm
- Carry out the backpropagation on an arbitrary computation graph
- Construct a computation graph for a neural network, identifying all the given and intermediate quantities that are relevant
- Instantiate the backpropagation algorithm for a neural network
- Instantiate an optimization method (e.g. SGD) and a regularizer (e.g. L2) when the parameters of a model are comprised of several matrices corresponding to different layers of a neural network
- Apply the empirical risk minimization framework to learn a neural network
- Use the finite difference method to evaluate the gradient of a function
- Identify when the gradient of a function can be computed at all and when it can be computed efficiently
- Employ basic matrix calculus to compute vector/matrix/tensor derivatives.