CS 546 – Advanced Topics in NLP

Dilek Hakkani-Tür

University of Illinois Urbana-Champaign

CONVAI

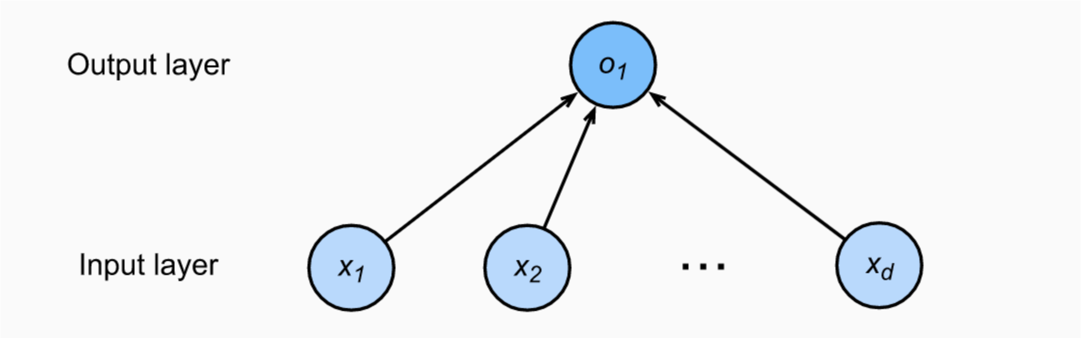
CDS Siebel School of Computing and Data Science

Topics for Today

- Gradient Descent
- Softmax Regression
- Multi-layer Perceptron

From Linear Regression to Neural Networks

- Linear regression is a single-layer neural network, consisting of just a single neuron!
- Fully connected layer (also called *dense* layer): every input is connected to every output.



Gradient Descent

- Seen the solution to linear regression last Thursday!
- Even when we cannot solve the models analytically, we can still train models effectively by **iteratively reducing the error by updating the parameters in the direction that incrementally lowers the loss function:**
  - Derivative of the true loss (i.e., average of the losses computed over all examples in the training data).
  - Expressing the updates mathematically:

$$(\mathbf{w}, b) \leftarrow (\mathbf{w}, b) - \frac{\eta}{|K|} \sum_{i \in K} \partial_{(\mathbf{w}, b)} L^{(i)}(\mathbf{w}, b)$$

Case Study: $f(x) = mx + b$

Example from the web

Gradient Descent (cont.)

- Steps of the algorithm:
  1. initialize the values of the model parameters, typically at random
  2. iteratively update the parameters in the direction of the negative gradient.

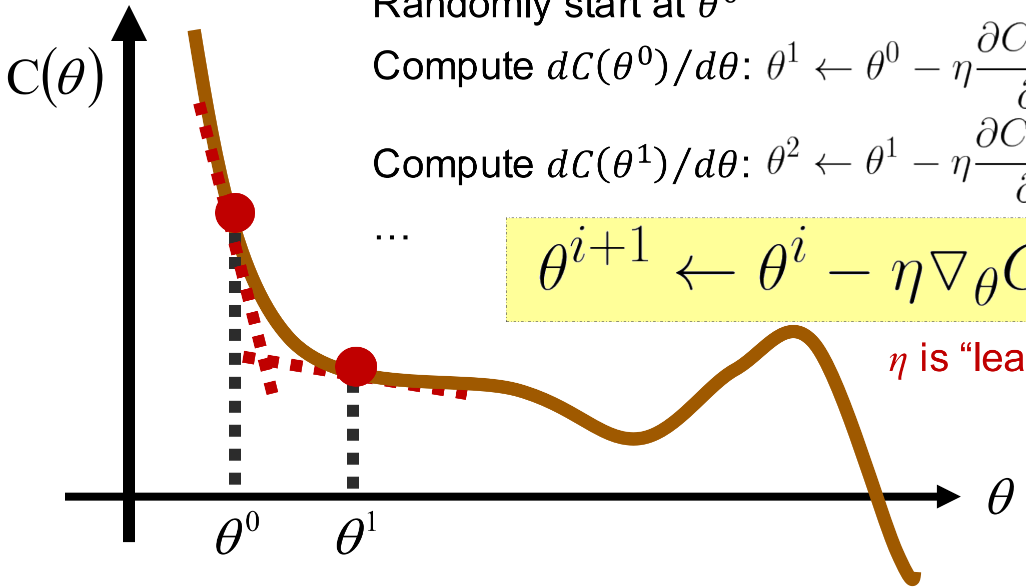
Notation:

θ: model parameters $(\mathbf{w}, b)$

$C(\theta)$: $L\ (\mathbf{w}, b)$

Gradient Descent (cont.)

- Assume that θ has only one variable



Randomly start at θ^0

Compute $dC(\theta^0)/d\theta$: $\theta^1 \leftarrow \theta^0 - \eta \frac{\partial C(\theta^0)}{\partial \theta}$

Compute $dC(\theta^1)/d\theta$: $\theta^2 \leftarrow \theta^1 - \eta \frac{\partial C(\theta^1)}{\partial \theta}$

…

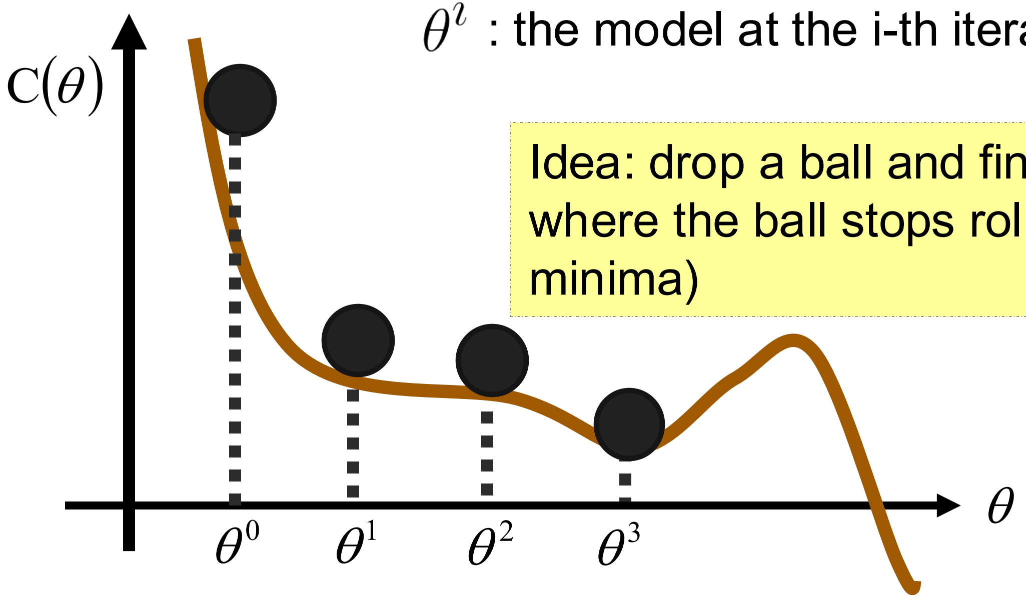
$$\theta^{i+1} \leftarrow \theta^i - \eta \nabla_\theta C(\theta^i)$$

η is "learning rate"

Gradient Descent (cont.)

- Assume that θ has only one variable

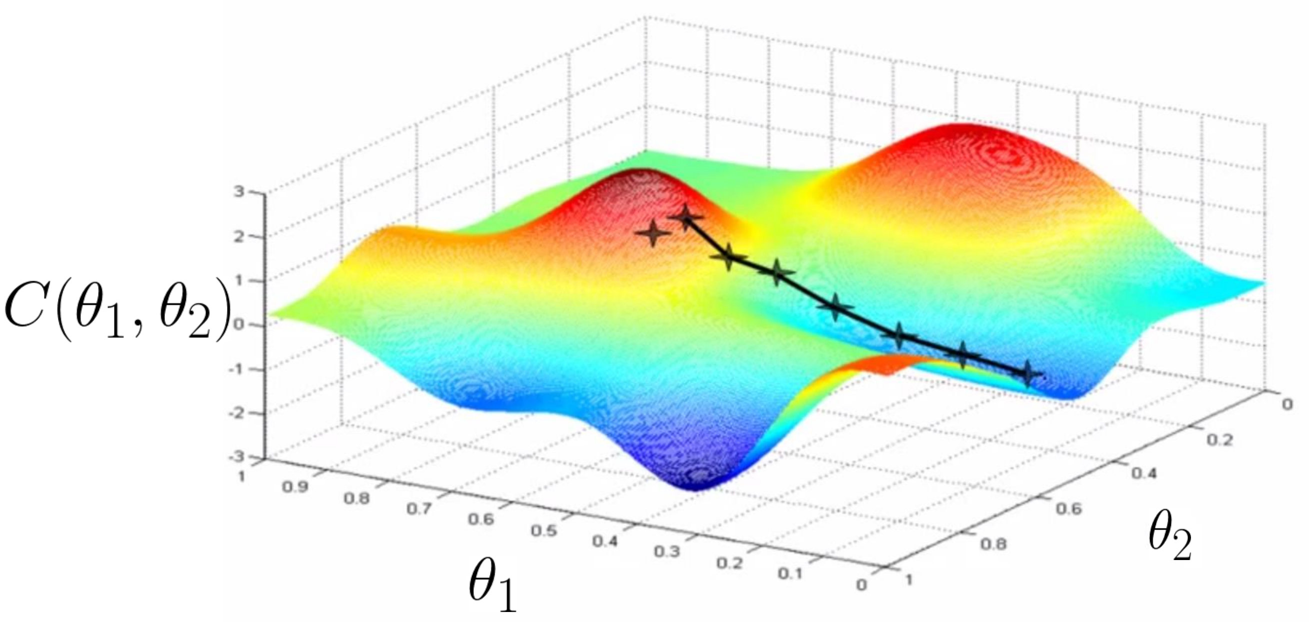
θ^i : the model at the i-th iteration



Idea: drop a ball and find the position where the ball stops rolling (local minima)

Gradient Descent (cont.)

- Assume that θ has two variables $\{\theta_1, \theta_2\}$



Gradient Descent (cont.)

- Assume that θ has two variables $\{\theta_1, \theta_2\}$
  - Randomly start at θ^0: $\theta^0 = \begin{bmatrix} \theta_1^0 \\ \theta_2^0 \end{bmatrix}$
  - Compute the gradients of $C(\theta)$ at θ^0: $\nabla_\theta C(\theta^0) = \begin{bmatrix} \frac{\partial C(\theta_1^0)}{\partial \theta_1} \\ \frac{\partial C(\theta_2^0)}{\partial \theta_2} \end{bmatrix}$
  - Update parameters:

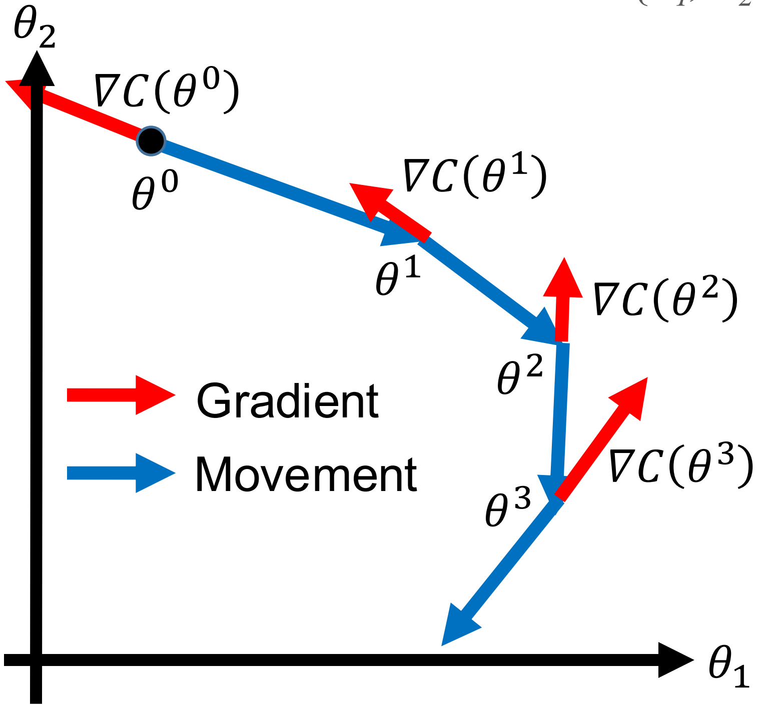
$$\begin{bmatrix} \theta_1^1 \\ \theta_2^1 \end{bmatrix} = \begin{bmatrix} \theta_1^0 \\ \theta_2^0 \end{bmatrix} - \eta \begin{bmatrix} \frac{\partial C(\theta_1^0)}{\partial \theta_1} \\ \frac{\partial C(\theta_2^0)}{\partial \theta_2} \end{bmatrix}$$

$$\theta^{i+1} \leftarrow \theta^i - \eta \nabla_\theta C(\theta^i)$$

  - Compute the gradients of $C(\theta)$ at θ^1: $\nabla_\theta C(\theta^1) = \begin{bmatrix} \frac{\partial C(\theta_1^1)}{\partial \theta_1} \\ \frac{\partial C(\theta_2^1)}{\partial \theta_2} \end{bmatrix}$
  - …

Gradient Descent (cont.)

- Assume that θ has two variables $\{\theta_1, \theta_2\}$



Algorithm

Initialization: start at θ^0

while($\theta^{(i+1)} \neq \theta^i$)

{

compute gradient at θ^i

update parameters

$$\theta^{i+1} \leftarrow \theta^i - \eta \nabla_\theta C(\theta^i)$$

}

An Issue with Gradient Descent

$$\theta^{i+1} = \theta^i - \eta \nabla C(\theta^i)$$

Training Data
$\{(x_1, \hat{y}_1), (x_2, \hat{y}_2), \ldots\}$

$$C(\theta) = \frac{1}{K} \sum_k \| f(x_k; \theta) - \hat{y}_k \| = \frac{1}{K} \sum_k C_k(\theta)$$

$$\nabla C(\theta^i) = \frac{1}{K} \sum_k \nabla C_k(\theta^i)$$

After seeing all training samples, the model can be updated → slow

Stochastic Gradient Descent (SGD)

- Gradient Descent

$$\theta^{i+1} = \theta^i - \eta \nabla C(\theta^i) \quad \nabla C(\theta^i) = \boxed{\frac{1}{K} \sum_k \nabla C_k(\theta^i)}$$

- Stochastic Gradient Descent (SGD)
  - Pick a training sample x_k

Training Data

$\{(x_1, \hat{y}_1), (x_2, \hat{y}_2), \ldots\}$

$$\theta^{i+1} = \theta^i - \eta \nabla C_k(\theta^i)$$

  - If all training samples have the same probability to be picked, then

$$E[\nabla C_k(\theta^i)] = \boxed{\frac{1}{K} \sum_k \nabla C_k(\theta^i)}$$

The model can be updated after seeing one training sample → faster

Stochastic Gradient Descent (SGD) (cont.)

- When running SGD, the model starts θ^0

Training Data
$\{(x_1, \hat{y}_1), (x_2, \hat{y}_2), \ldots\}$

pick x_1 $\quad \theta^1 = \theta^0 - \eta \nabla C_1(\theta^0)$

pick x_2 $\quad \theta^2 = \theta^1 - \eta \nabla C_2(\theta^1)$

$\vdots \qquad \vdots$

pick x_k $\quad \theta^k = \theta^{k-1} - \eta \nabla C_k(\theta^{k-1})$

$\vdots \qquad \vdots$

pick x_K $\quad \theta^K = \theta^{K-1} - \eta \nabla C_K(\theta^{K-1})$

see all training samples once

→ one epoch

pick x_1 $\quad \theta^{K+1} = \theta^K - \eta \nabla C_1(\theta^K)$

Mini-Batch SGD

- Batch Gradient Descent

Use all K samples in each iteration

$$\theta^{i+1} = \theta^i - \eta \frac{1}{K} \sum_k \nabla C_k(\theta^i)$$

- Stochastic Gradient Descent (SGD)
  - Pick a training sample x_k

Use *1* sample in each iteration

$$\theta^{i+1} = \theta^i - \eta \nabla C_k(\theta^i)$$

- Mini-Batch SGD
  - Pick a set of B training samples as a batch b

Use all B samples in each iteration

B is the "batch size"

$$\theta^{i+1} = \theta^i - \eta \frac{1}{B} \sum_{x_k \in b} \nabla C_k(\theta^i)$$

Topics for Today

- Gradient Descent
- Softmax Regression
- Multi-layer Perceptron

Softmax Regression

- Regression is useful for “how much?” and “how many?” questions, where the target can take a real value.

- Many NLP problems are about “which one?”
  - Sentiment classification: positive, negative(, neutral)
  - Sentence boundary detection from text: is the punctuation mark defining a sentence boundary or not
  - What is the part-of-speech tag of the word “word” in this sentence? NOUN, VERB, etc.?

- We can represent target labels with one-hot encodings as well. For example, for sentiment classification:

$$y \in \{\underbrace{(1,0,0)}_{\text{Positive}},\underbrace{(0,1,0)}_{\text{Negative}},\underbrace{(0,0,1)}_{\text{Neutral}}\}.$$

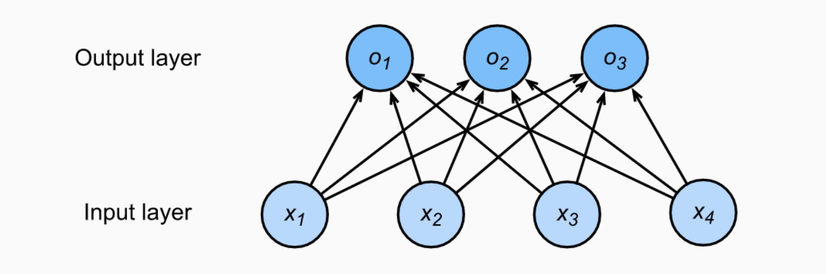
A Network Architecture for Multi-Class Problems

- To estimate the conditional probabilities associated with each class, we need a model with multiple outputs, one per class.
- As many linear functions as we have outputs.
- Example: Assume our input is represented with 4 features, then to compute the logits o_1, o_2, o_3, for each output, we need:

$$o_1 = x_1 w_{11} + x_2 w_{12} + x_3 w_{13} + x_4 w_{14} + b_1$$
$$o_2 = x_1 w_{21} + x_2 w_{22} + x_3 w_{23} + x_4 w_{24} + b_2$$
$$o_3 = x_1 w_{31} + x_2 w_{32} + x_3 w_{33} + x_4 w_{34} + b_3$$

$$\mathbf{o} = \mathbf{W}\mathbf{x} + \mathbf{b}$$

3 x 4 matrix



Multi-Class Problems

- Need to interpret model outputs as probabilities and optimize our parameters to produce probabilities that maximize the likelihood of the observed data

- To generate predictions, we can set a threshold or choose the *argmax* of the predicted probabilities

- Interpret the logits o directly as our outputs of interest, however:
    - Nothing constrains these numbers to sum to 1.
    - Depending on the inputs, they can take negative values.

Softmax

- Transforms logits such that they become nonnegative and sum to 1, while requiring that the model remains differentiable.

$$\hat{\mathbf{y}} = \mathrm{softmax}(\mathbf{o}) \quad \text{where} \quad \hat{y}_i = \frac{\exp(o_i)}{\sum_j \exp(o_j)}$$

- $\hat{y_1} + \hat{y_2} + \hat{y_3} = 1$,
- $0 \leq \hat{y_i} \leq 1$ for all i, and
- The ordering of the logits has not changed; hence we can still pick the output using argmax!

Maximum Likelihood Estimation

- The softmax function gives us a vector $\hat{\mathbf{y}}$, interpreted as estimated conditional probabilities of each class given the input x, e.g., $\hat{y_1} = P\ (y\text{=cat}|\mathbf{x})$.

$$P(Y \mid X) = \prod_{i=1}^{n} P(y^{(i)} \mid x^{(i)})$$

$$-\log P(Y \mid X) = \sum_{i=1}^{n} -\log P(y^{(i)} \mid x^{(i)})$$

- Maximizing $P(Y|X)$ (and thus equivalently minimizing $-\log P(Y|X)$) corresponds to predicting the label well!

Cross-Entropy Loss and Softmax

$$l(\mathbf{y}, \hat{\mathbf{y}}) = -\sum_{j=1}^{q} y_j \log \hat{y}_j.$$

q: the number of classes

- If we add o into the definition of the loss l and use the definition of the softmax we obtain:

$$\begin{aligned} l(\mathbf{y}, \hat{\mathbf{y}}) &= -\sum_{j=1}^{q} y_j \log \frac{\exp(o_j)}{\sum_{k=1}^{q} \exp(o_k)} \\ &= \sum_{j=1}^{q} y_j \log \sum_{k=1}^{q} \exp(o_k) - \sum_{j=1}^{q} y_j o_j \\ &= \log \sum_{k=1}^{q} \exp(o_k) - \sum_{j=1}^{q} y_j o_j. \end{aligned}$$

- If we compute its derivative with respect to o_i, we get:

$$\partial_{o_j} l(\mathbf{y}, \hat{\mathbf{y}}) = \frac{\exp(o_j)}{\sum_{k=1}^{q} \exp(o_k)} - y_j = \mathrm{softmax}(\mathbf{o})_j - y_j.$$

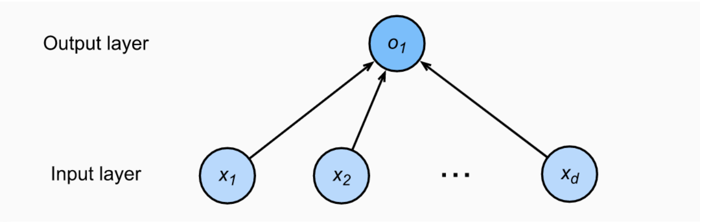
The gradient is the difference between the probability assigned to the true class by our model. This makes computing gradients very easy in practice!

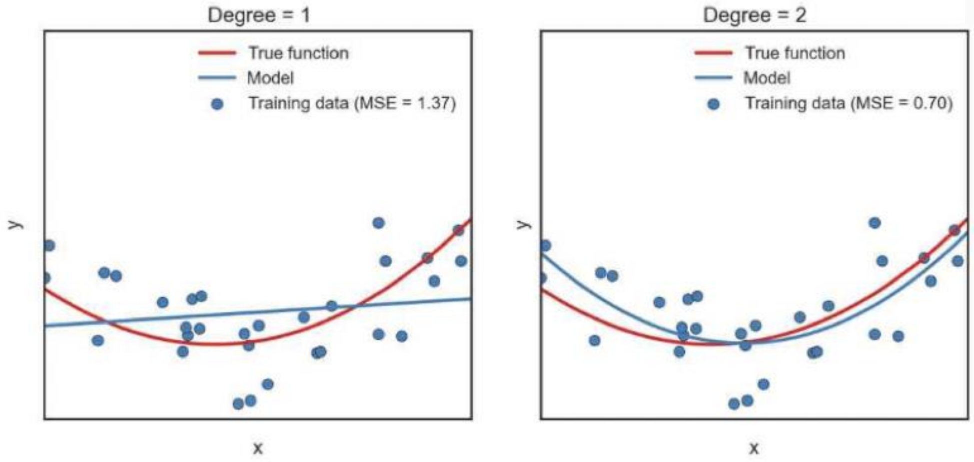
Topics for Today

- Gradient Descent
- Softmax Regression
- Multi-layer Perceptron

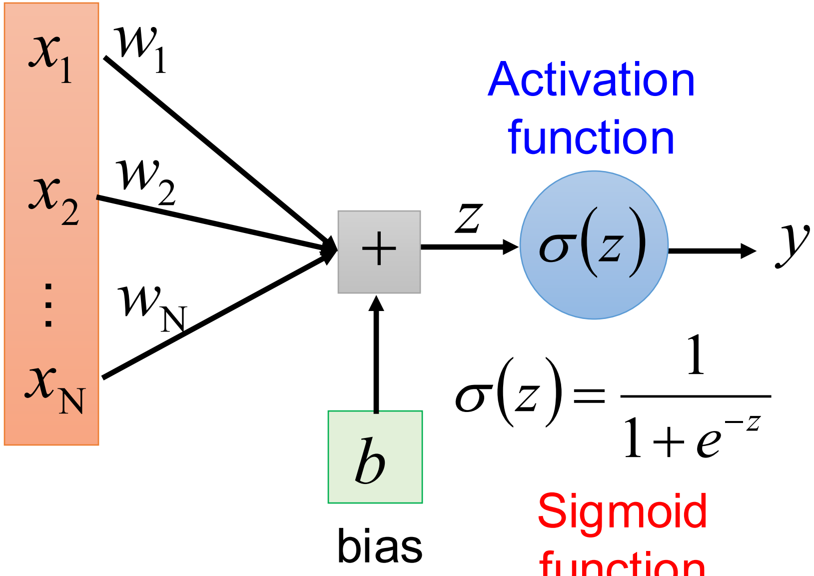
Non-Linearity

- Linear regression inputs directly to our outputs via a single linear transformation.
- But what if our data is not linear?



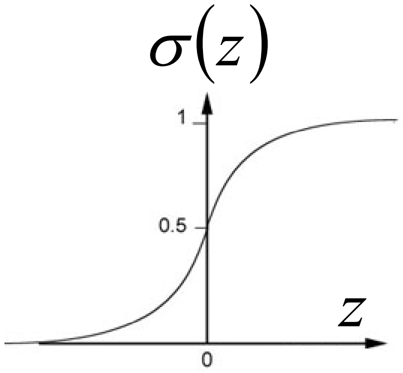


Perceptron – A Single Neuron



$$\sigma(z) = \frac{1}{1 + e^{-z}}$$

Sigmoid function



Each neuron is a very simple function

Why non-linearity?

- Function approximation
  - ***Without non-linearity***, deep neural networks work the same as linear transform

$$W_1(W_2 \cdot x) = (W_1 W_2)x = Wx$$

  - ***With non-linearity***, networks with more layers can approximate more complex functions

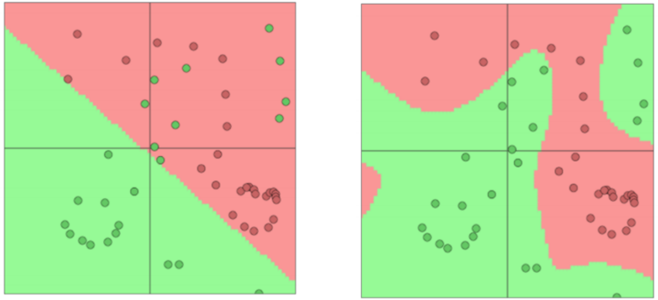
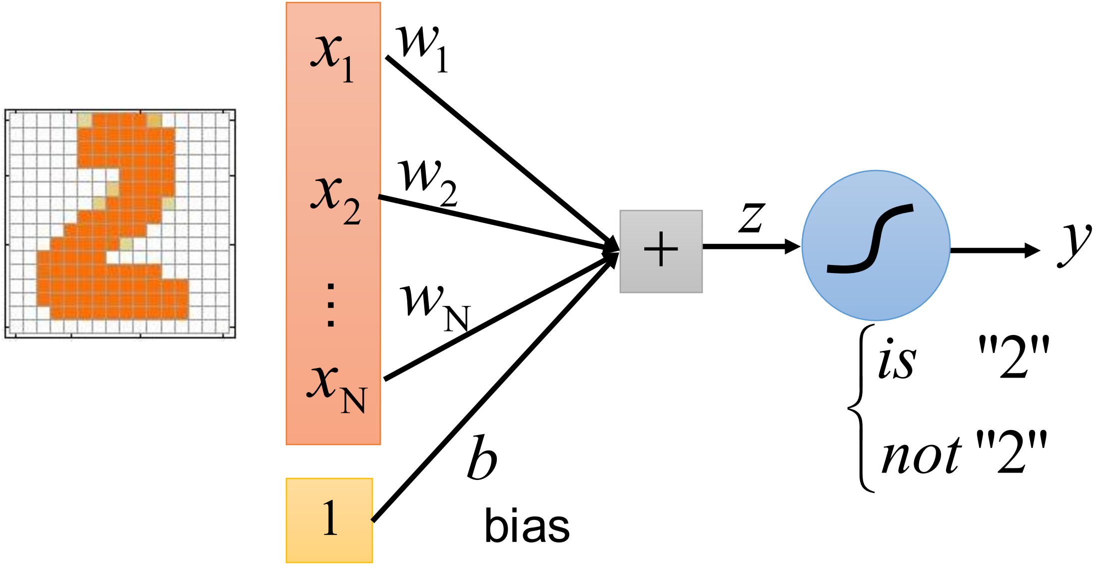


Figure from: http://cs224d.stanford.edu/lectures/CS224d-Lecture4.pdf

Perceptron – A Single Neuron

$$f: R^N \rightarrow R^M$$



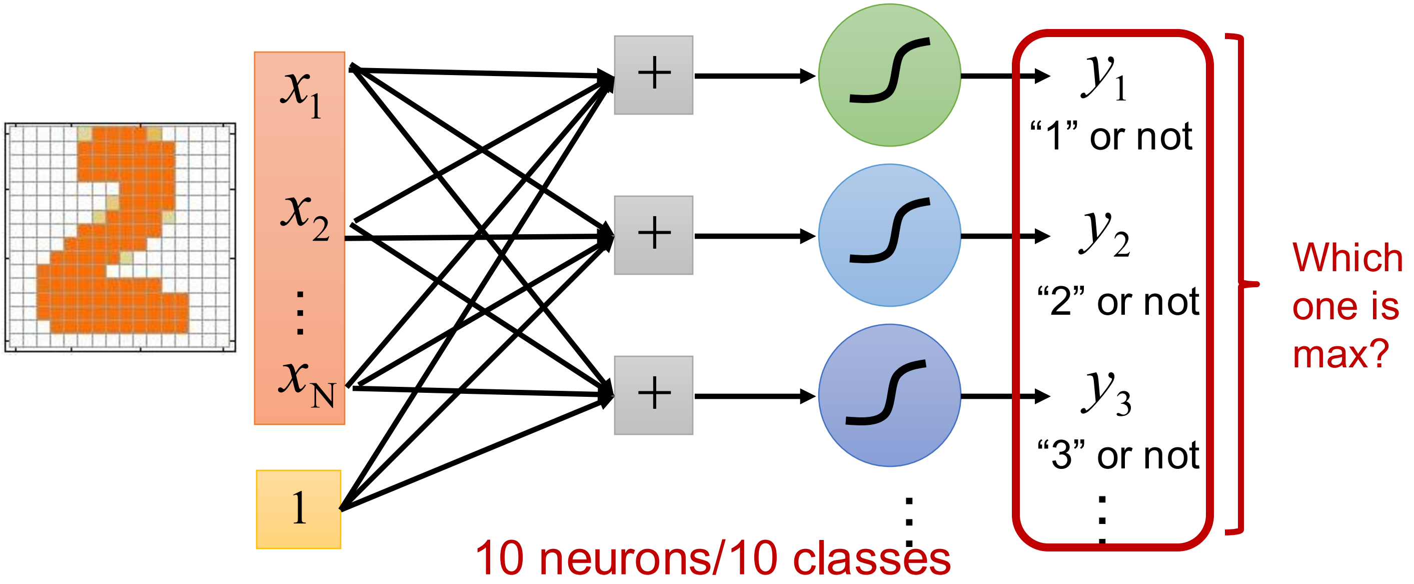
$$\begin{cases} is \quad "2" & y \geq 0.5 \\ not\ "2" & y < 0.5 \end{cases}$$

A single neuron can only handle binary classification

A Layer of Neurons

$$f : R^N \rightarrow R^M$$

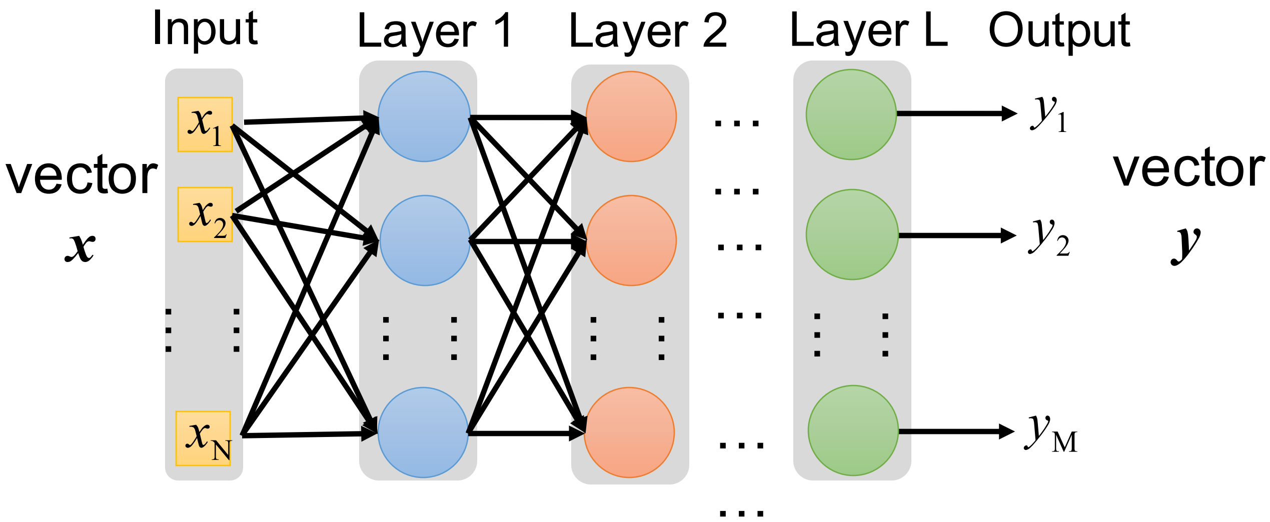
- Handwriting digit classification



A layer of neurons can handle multiple possible output, and the result depends on the max one

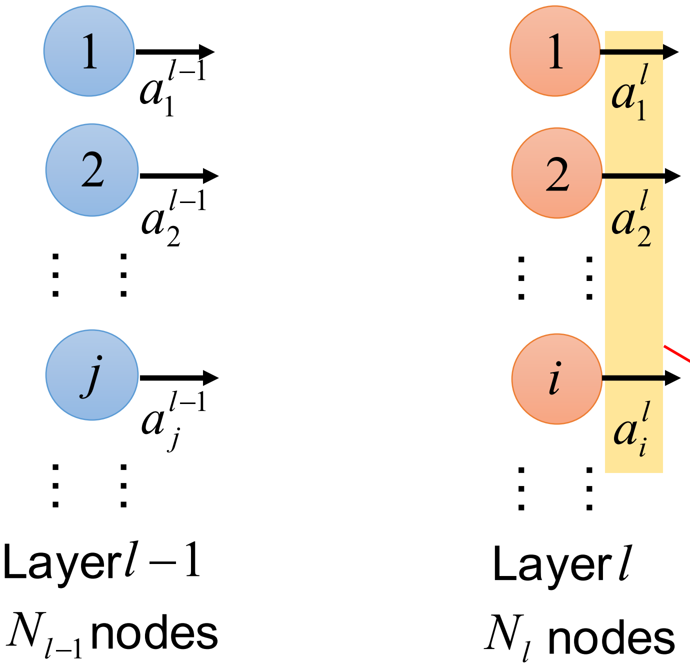
Deep Neural Networks (DNNs)

$$f : R^N \rightarrow R^M$$



Deep NN: multiple hidden layers

DNNs – Notation



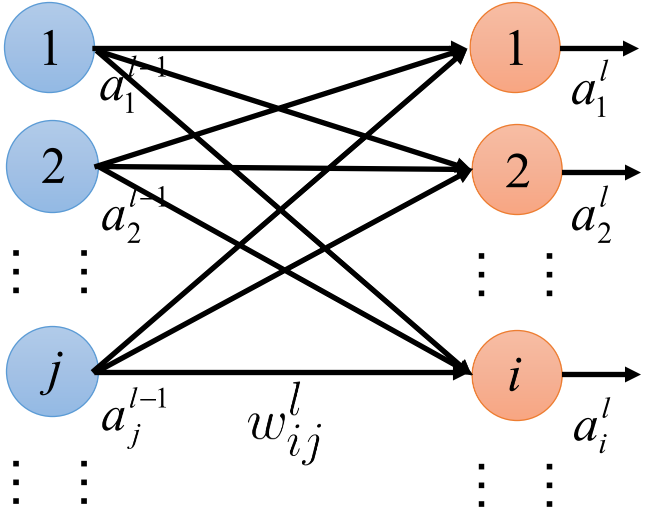
Output of a neuron:

a_i^l → layer l (superscript), neuron i (subscript)

$$a^l = \begin{bmatrix} \vdots \\ a_i^l \\ \vdots \end{bmatrix}$$

output of one layer → a vector

DNNs – Notation (cont.)



Layer $l-1$
N_{l-1} nodes

Layer l
N_l nodes

$$w_{ij}^l$$

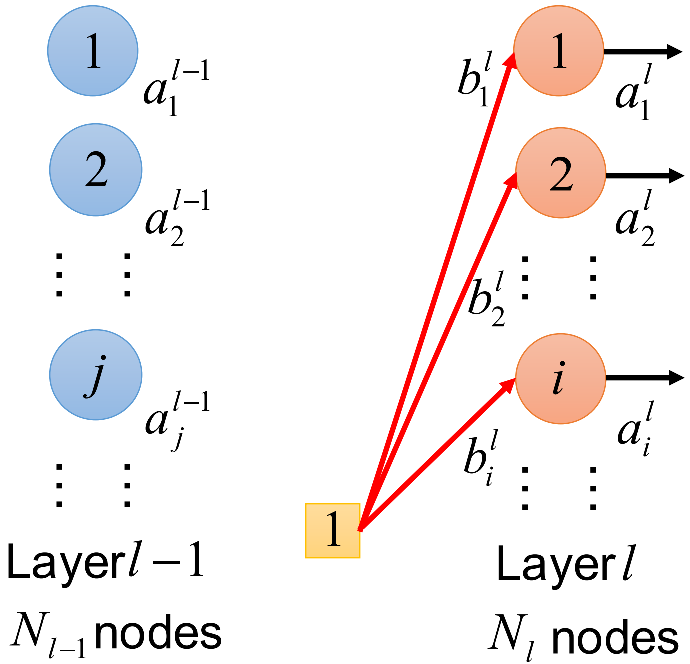
layer $l-1$ to layer l

from neuron j (layer l-1) to neuron i (layer l)

$$W^l = \begin{array}{cc} \overleftrightarrow{\quad N_{l-1} \quad} & \\ \begin{bmatrix} w_{11}^l & w_{12}^l & \cdots \\ w_{21}^l & w_{22}^l & \\ \vdots & & \ddots \end{bmatrix} & \updownarrow N_l \end{array}$$

weights between two layers → a matrix

DNNs – Notation (cont.)

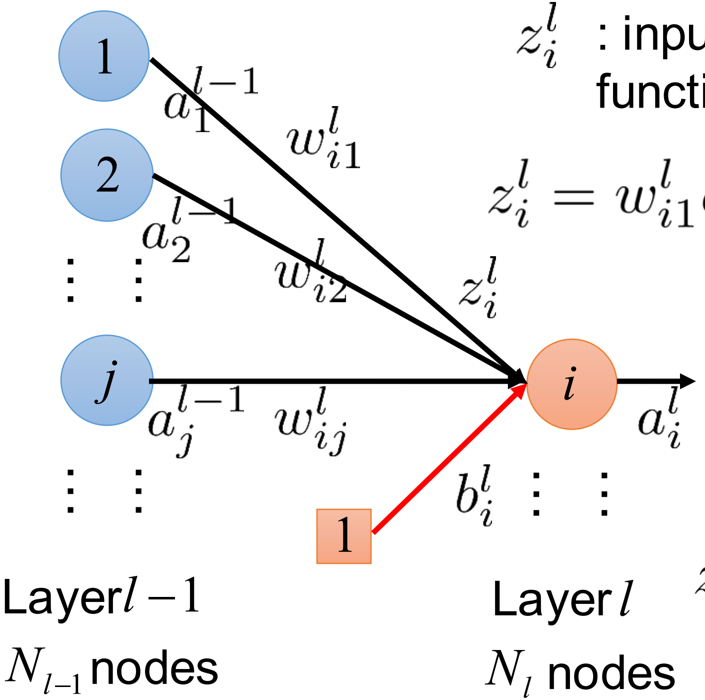


b_i^l : bias for neuron i at layer l

$$b^l = \begin{bmatrix} \vdots \\ b_i^l \\ \vdots \end{bmatrix}$$

bias of all neurons at each layer → a vector

DNNs – Notation (cont.)



z_i^l : input of the activation function for neuron i at layer l

$$z_i^l = w_{i1}^l a_1^{l-1} + w_{i2}^l a_2^{l-1} + \ldots + b_i^l$$

$$z_i^l = \sum_{j=1}^{N_{l-1}} w_{ij}^l a_j^{l-1} + b_i^l$$

$$z^l = \begin{bmatrix} \vdots \\ z_i^l \\ \vdots \end{bmatrix}$$

activation function input at each layer
→ a vector

DNNs – Notation Summary

a_i^l : output of a neuron

a^l : output vector of a layer

z_i^l : input of activation function

z^l : input vector of activation function for a layer

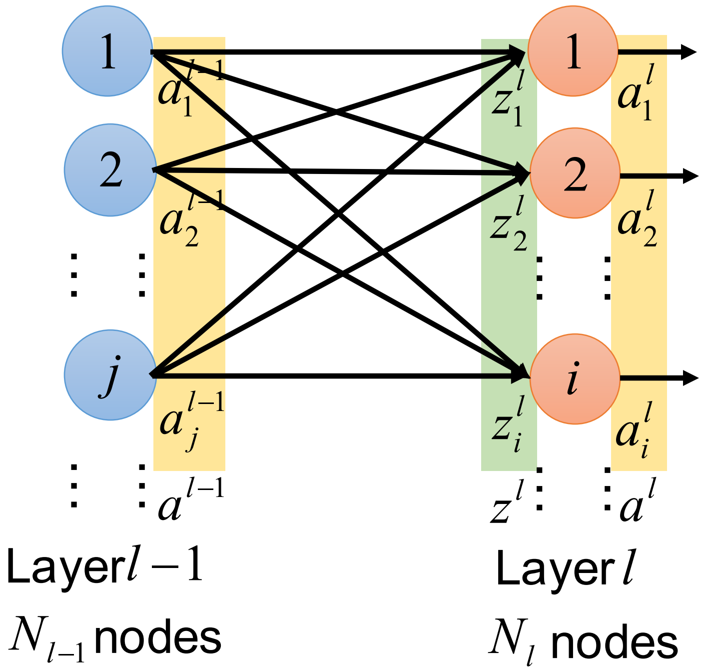
w_{ij}^l : a weight

W^l : a weight matrix

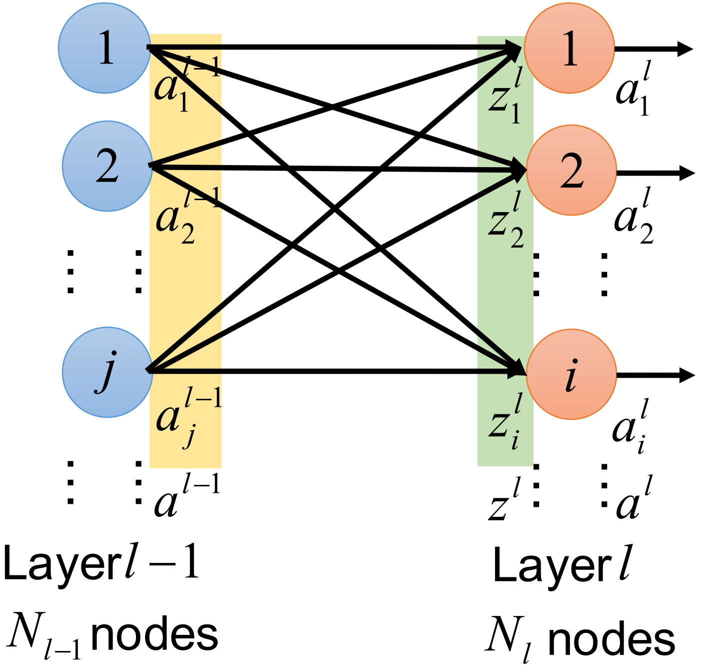
b_i^l : a bias

b^l : a bias vector

Output of a Layer



Output of a Layer – From a to z



$$z_1^l = w_{11}^1 a_1^{l-1} + w_{12}^1 a_2^{l-1} + \cdots + b_1^l$$

$$\vdots$$

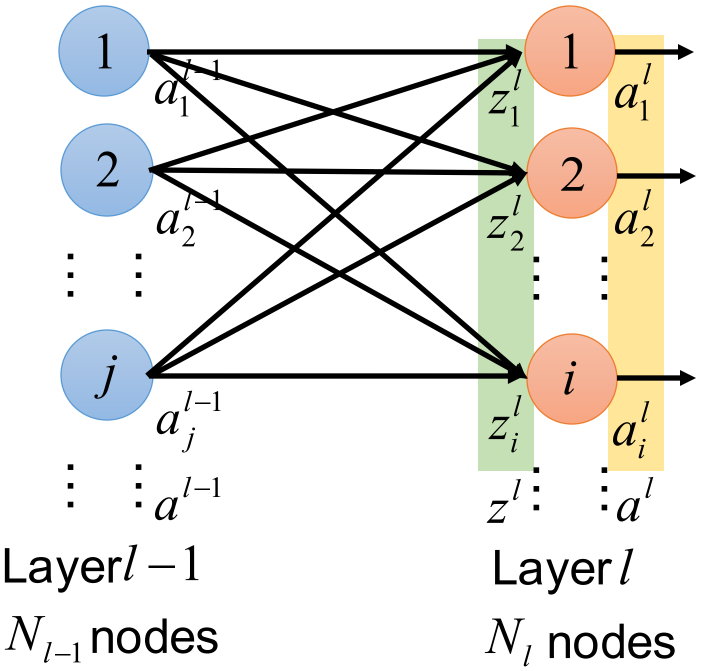
$$z_i^l = w_{i1}^1 a_1^{l-1} + w_{i2}^1 a_2^{l-1} + \cdots + b_i^l$$

$$\vdots$$

$$\begin{bmatrix} z_1^l \\ \vdots \\ z_i^l \\ \vdots \end{bmatrix} = \begin{bmatrix} w_{11}^l & w_{12}^l & \cdots \\ w_{21}^l & w_{22}^l & \\ \vdots & & \ddots \end{bmatrix} \begin{bmatrix} a_1^{l-1} \\ \vdots \\ a_i^{l-1} \\ \vdots \end{bmatrix} + \begin{bmatrix} b_1^l \\ \vdots \\ b_i^l \\ \vdots \end{bmatrix}$$

$$z^l = W^l a^{l-1} + b^l$$

Output of a Layer – From z to a

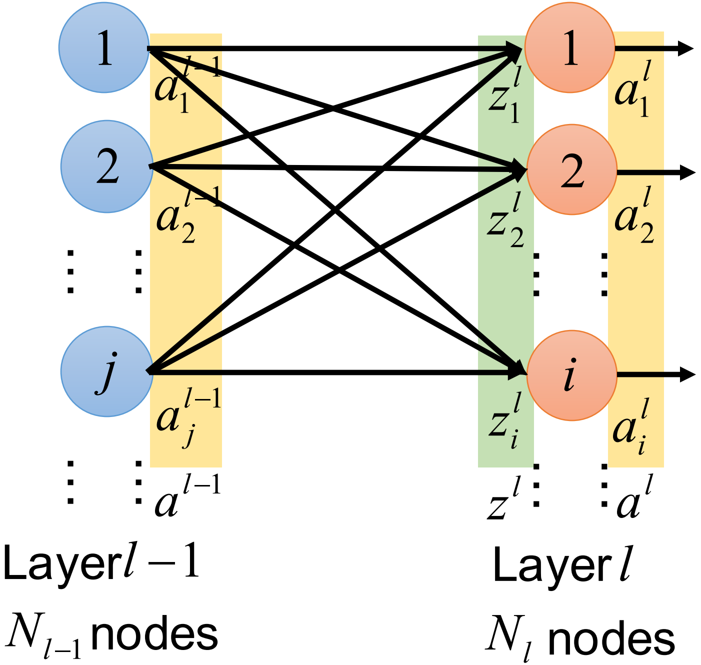


$$a_i^l = \sigma\left(z_i^l\right)$$

$$\begin{bmatrix} a_1^l \\ a_2^l \\ \vdots \\ a_i^l \\ \vdots \end{bmatrix} = \begin{bmatrix} \sigma\left(z_1^l\right) \\ \sigma\left(z_2^l\right) \\ \vdots \\ \sigma\left(z_i^l\right) \\ \vdots \end{bmatrix}$$

$$a^l = \sigma\left(z^l\right)$$

Output of a Layer



$$z^l = W^l a^{l-1} + b^l$$

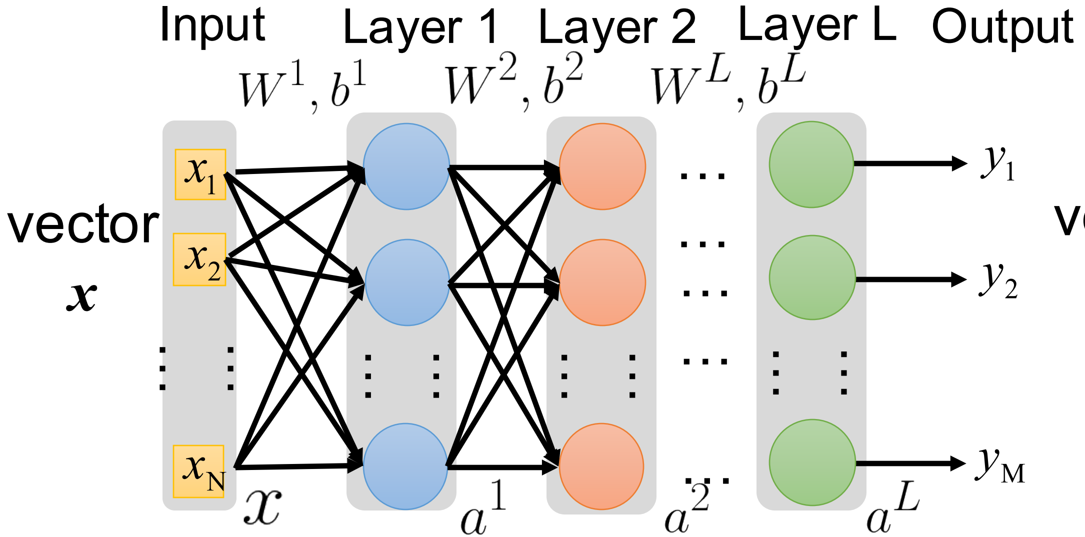
$$a^l = \sigma\left(z^l\right)$$

$$\boxed{a^l = \sigma(W^l a^{l-1} + b^l)}$$

DNN Formulation

$$f: R^N \rightarrow R^M$$

- Fully connected feedforward network

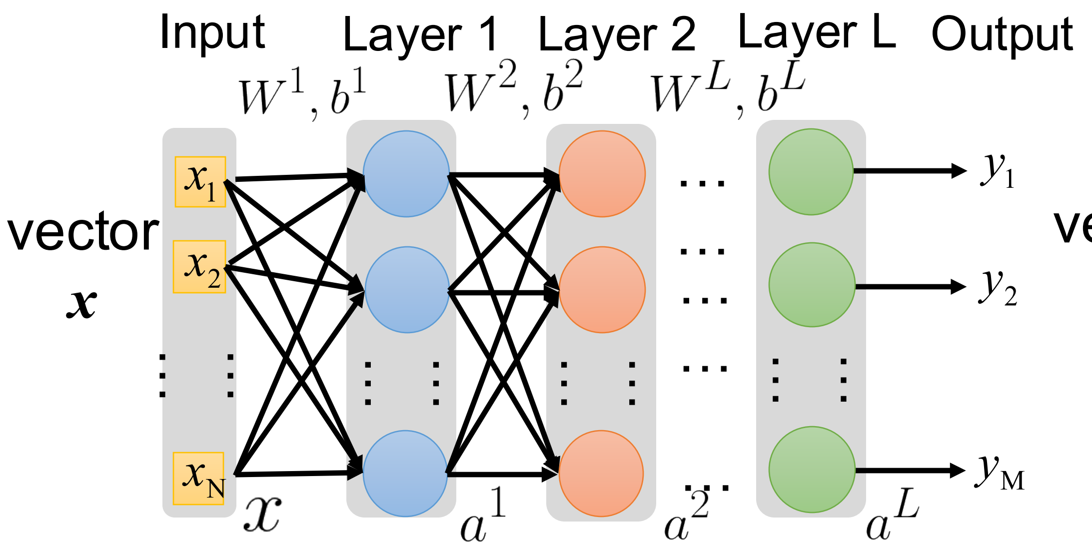


$$a^1 = \sigma(W^1 x + b^1) \quad a^2 = \sigma(W^2 a^1 + b^2) \quad a^L = \sigma(W^L a^{L-1} + b^L) = y$$

DNN Formulation

$$f: R^N \to R^M$$

- Fully connected feedforward network



$$y = f(x) = \sigma(W^L \cdots \sigma(W^2 \sigma(W^1 x + b^1) + b^2) \ldots + b^L)$$

Loss Function for Training

Back to an earlier slide!

x : "It makes too much noise."
function input

$\hat{y}$: **-** (negative)
function output

Model: Hypothesis Function Set
$f_1, f_2 \cdots$

Training Data
$\{(x_1, \hat{y}_1), (x_2, \hat{y}_2), \ldots\}$

Training: Pick the best function f^*

"Best" Function f^*

A "Good" function: $f(x;\theta) \sim \hat{y} \Rightarrow \|\hat{y} - f(x;\theta)\| \approx 0$

Define an example loss function:

$$C(\theta) = \sum_k \|\hat{y}_k - f(x_k;\theta)\|$$

sum over the error of all training samples

Gradient Descent for DNNs

$$y = f(x) = \sigma(W^L \cdots \sigma(W^2\sigma(W^1x + b^1) + b^2) \ldots + b^L)$$

$$\theta = \left\{W^1, b^1, W^2, b^2, \cdots W^L, b^L\right\}$$

$$W^l = \begin{bmatrix} w_{11}^l & w_{12}^l & \cdots \\ w_{21}^l & w_{22}^l & \\ \vdots & & \ddots \end{bmatrix} \quad b^l = \begin{bmatrix} \vdots \\ b_i^l \\ \vdots \end{bmatrix}$$

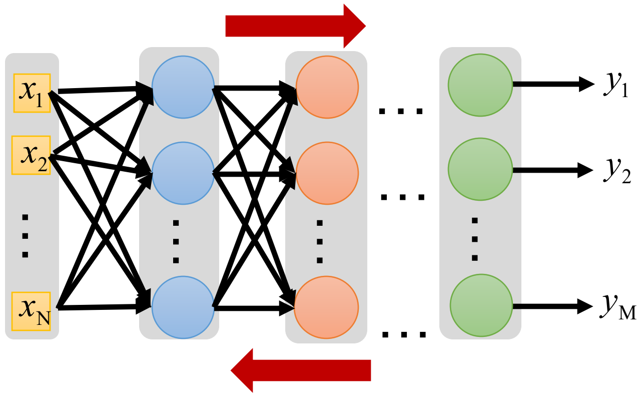
$$\nabla C(\theta) = \begin{bmatrix} \vdots \\ \frac{\partial C(\theta)}{\partial w_{ij}^l} \\ \vdots \\ \frac{\partial C(\theta)}{\partial b_i^l} \end{bmatrix}$$

Algorithm
Initialization: start at θ^0
while($\theta^{(i+1)} \neq \theta^i$)
{
compute gradient at θ^i
update parameters
$\theta^{i+1} \leftarrow \theta^i - \eta \nabla_\theta C(\theta^i)$
}

To update weights efficiently, we use **backpropagation**.

Forward vs. Backward Propagation

- In a feedforward neural network
  - forward propagation
    - from input x to output y information <u>flows forward</u> through the network
    - during training, forward propagation can continue onward until it produces a scalar loss $C(\theta)$
  - back-propagation
    - allows the information from the cost to then <u>flow backwards</u> through the network, in order to compute the **gradient**
    - can be applied to any function



Chain Rule

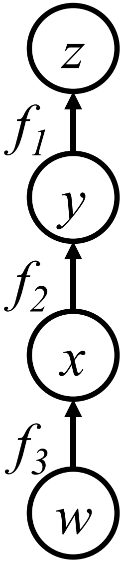
$$\frac{\partial z}{\partial w} = \frac{\partial z}{\partial y}\frac{\partial y}{\partial x}\frac{\partial x}{\partial w}$$

$$= f_1'(y)\, f_2'(x)\, f_3'(w)$$

forward propagation for computing loss

$$= f_1'(f_2(f_3(w)))\, f_2'(f_3(w))\, f_3'(w)$$

back-propagation of gradients for updating weights



$z=f_1(f_2(f_3(w)))$ or

$z=f_1(y)$

$y=f_2(x)$

$x=f_3(w)$

Gradient Descent for DNNs

$$y = f(x) = \sigma(W^L \cdots \sigma(W^2\sigma(W^1 x + b^1) + b^2)\ldots + b^L)$$

$$\theta = \left\{W^1, b^1, W^2, b^2, \cdots W^L, b^L\right\}$$

$$W^l = \begin{bmatrix} w_{11}^l & w_{12}^l & \cdots \\ w_{21}^l & w_{22}^l & \\ \vdots & & \ddots \end{bmatrix} \quad b^l = \begin{bmatrix} \vdots \\ b_i^l \\ \vdots \end{bmatrix}$$

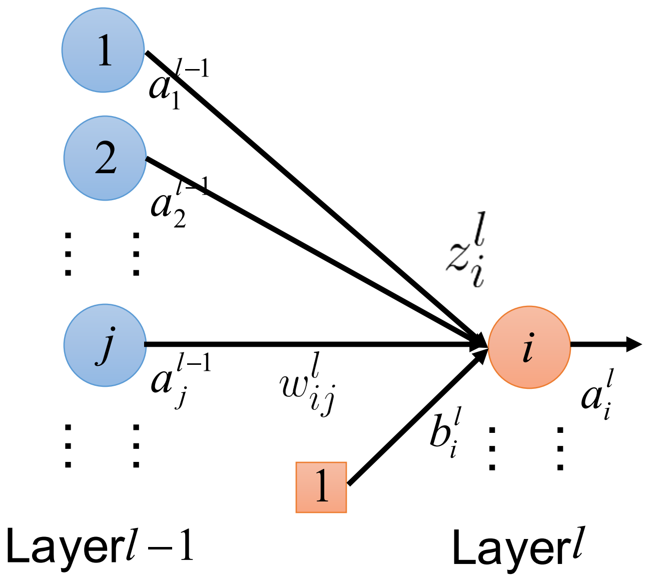
$$\nabla C(\theta) = \begin{bmatrix} \vdots \\ \frac{\partial C(\theta)}{\partial w_{ij}^l} \\ \vdots \\ \frac{\partial C(\theta)}{\partial b_i^l} \end{bmatrix}$$

Algorithm
Initialization: start at θ^0
while($\theta^{(i+1)} \neq \theta^i$)
{
compute gradient at θ^i
update parameters
$\theta^{i+1} \leftarrow \theta^i - \eta\nabla_\theta C(\theta^i)$
}

To update weights efficiently, we use **backpropagation**.

Gradient Descent for DNNs (cont.)

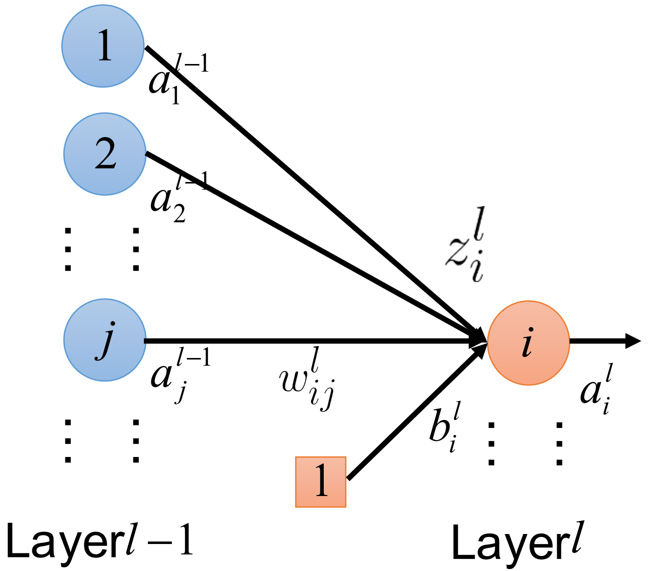
$$\partial C(\theta) / \partial w_{ij}^{l}$$



$$\frac{\partial C(\theta)}{\partial w_{ij}^{l}} = \frac{\partial C(\theta)}{\partial z_i^{l}} \frac{\partial z_i^{l}}{\partial w_{ij}^{l}}$$

Gradient Descent for DNNs (cont.)

$$\partial z_i^l / \partial w_{ij}^l \ (l > 1)$$



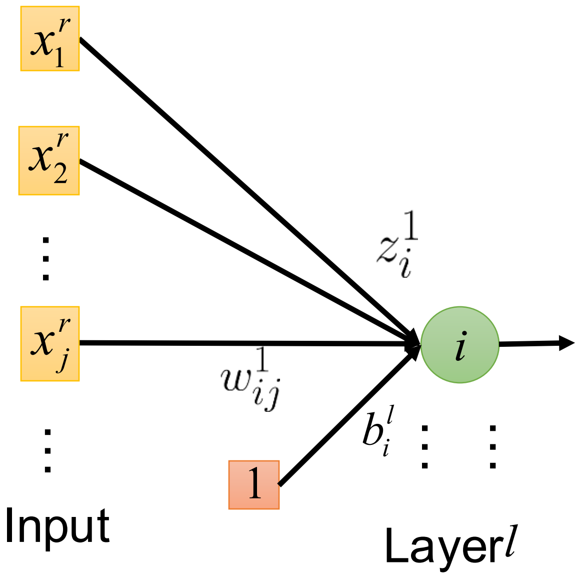
$$z^l = W^l a^{l-1} + b^l$$

$$z_i^l = \sum_j w_{ij}^l a_j^{l-1} + b_i^l$$

$$\frac{\partial z_i^l}{\partial w_{ij}^l} = a_j^{l-1}$$

Gradient Descent for DNNs (cont.)

$$\partial z_i^l / \partial w_{ij}^l \ (l = 1)$$



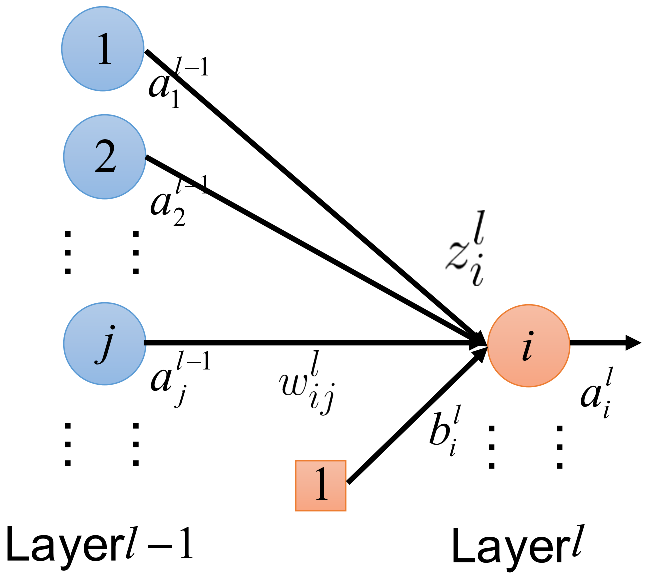
$$z^1 = W^1 x + b^1$$

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

$$\frac{\partial z_i^1}{\partial w_{ij}^1} = x_j$$

Gradient Descent for DNNs (cont.)

$$\partial C(\theta) / \partial w_{ij}^{l}$$

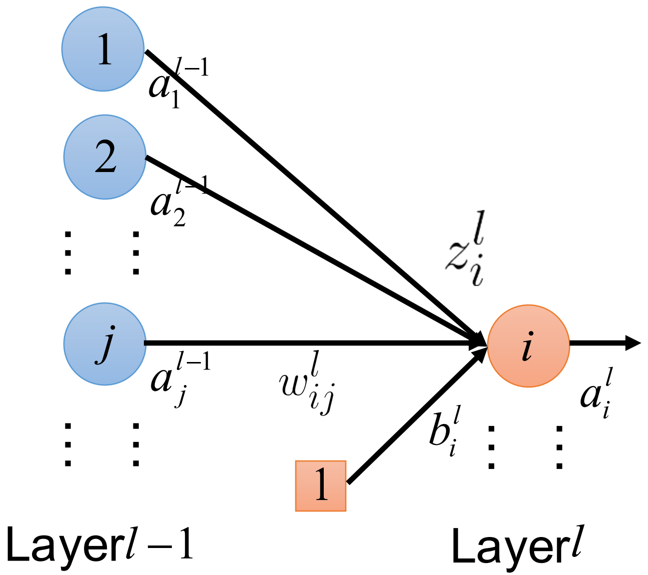


$$\frac{\partial C(\theta)}{\partial w_{ij}^{l}} = \frac{\partial C(\theta)}{\partial z_i^{l}} \frac{\partial z_i^{l}}{\partial w_{ij}^{l}}$$

$$\frac{\partial z_i^{l}}{\partial w_{ij}^{l}} = \begin{cases} a_j^{l-1} & , l > 1 \\ x_j & , l = 1 \end{cases}$$

Gradient Descent for DNNs (cont.)

$$\partial C(\theta) / \partial w_{ij}^{l}$$

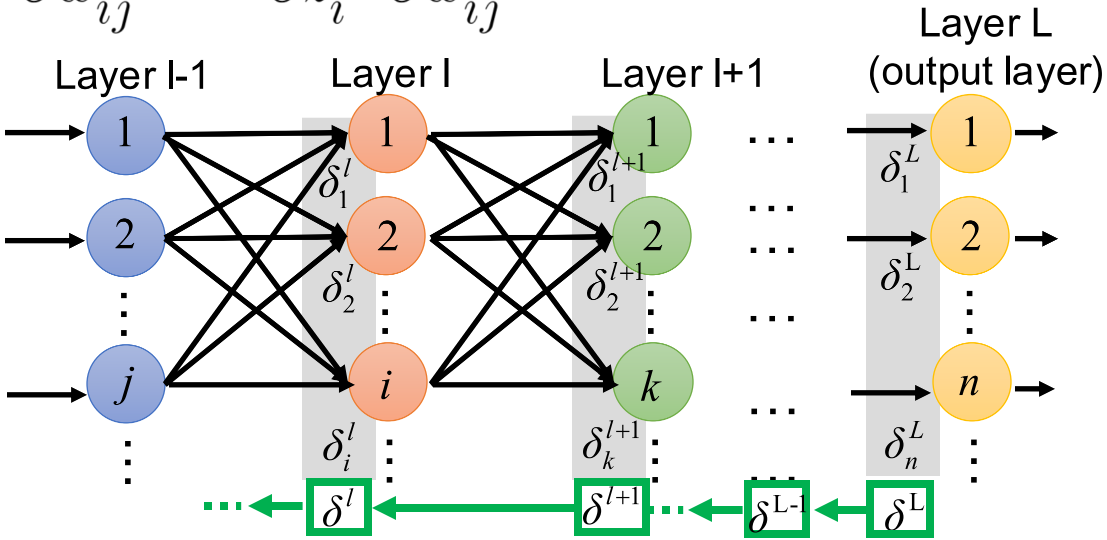


$$\frac{\partial C(\theta)}{\partial w_{ij}^{l}} = \frac{\partial C(\theta)}{\partial z_i^{l}} \frac{\partial z_i^{l}}{\partial w_{ij}^{l}}$$

Gradient Descent for DNNs (cont.)

$$\frac{\partial C(\theta)}{\partial w_{ij}^l} = \frac{\partial C(\theta)}{\partial z_i^l} \frac{\partial z_i^l}{\partial w_{ij}^l}$$

δ_i^l : the propagated gradient corresponding to the l-th layer



Idea: computing δ^l layer by layer (from δ^L to δ^1) is more efficient

Gradient Descent for DNNs (cont.)

$$\partial C(\theta)/\partial z_i^l = \delta_i^l$$

- Idea: from L to 1

① Initialization: compute δ^L

② Compute δ^l based on δ^{l+1}

Gradient Descent for DNNs (cont.)

$$\partial C(\theta) / \partial z_i^l = \delta_i^l$$

- Idea: from L to 1

① **Initialization: compute δ^L**

② Compute δ^l based on δ^{l+1}

$$\delta_i^L = \frac{\partial C}{\partial z_i^L} \qquad \Delta z_i^L \rightarrow \Delta a_i^L = \Delta y_i \rightarrow \Delta C$$

$$= \frac{\boxed{\partial C}\,\partial y_i}{\boxed{\partial y_i}\,\partial z_i^L}$$

$\partial C / \partial y_i$ depends on the loss function

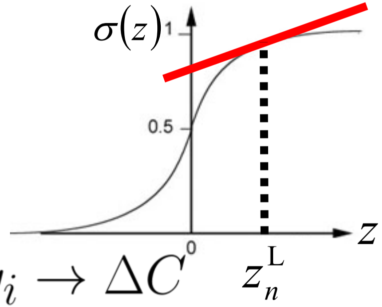
Gradient Descent for DNNs (cont.)

$$\partial C(\theta)/\partial z_i^l = \delta_i^l$$

- Idea: from L to 1

① **Initialization: compute δ^L**

② Compute δ^l based on δ^{l+1}



$$\Delta z_i^L \rightarrow \Delta a_i^L = \Delta y_i \rightarrow \Delta C$$

$$\delta_i^L = \frac{\partial C}{\partial z_i^L}$$

$$= \frac{\partial C}{\partial y_i}\frac{\partial y_i}{\partial z_i^L} \qquad y_i = a_i^L = \sigma(z_i^L)$$

$$= \frac{\partial C}{\partial y_i}\sigma'(z_i^L)$$

$$\sigma'(z^L) = \begin{bmatrix} \sigma'(z_1^L) \\ \sigma'(z_2^L) \\ \vdots \\ \sigma'(z_i^L) \\ \vdots \end{bmatrix} \qquad \nabla C(y) = \begin{bmatrix} \frac{\partial C}{\partial y_1} \\ \frac{\partial C}{\partial y_2} \\ \vdots \\ \frac{\partial C}{\partial y_i} \\ \vdots \end{bmatrix}$$

$$\boxed{\delta^L = \sigma'(z^L) \odot \nabla C(y)}$$

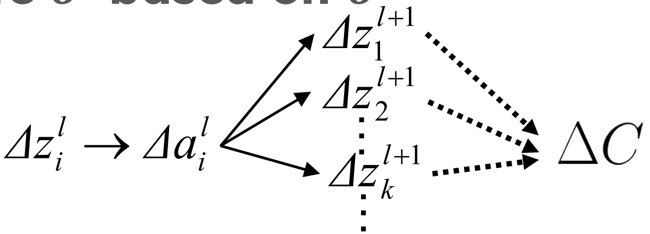
Gradient Descent for DNNs (cont.)

$$\partial C(\theta)/\partial z_i^l = \delta_i^l$$

- Idea: from L to 1

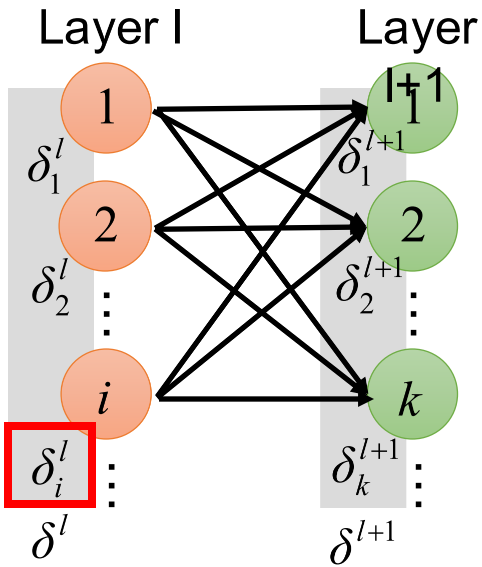
① Initialization: compute δ^L

② **Compute δ^l based on δ^{l+1}**



$$\delta_i^l = \frac{\partial C}{\partial z_i^l} = \sum_k \left(\frac{\partial C}{\partial z_k^{l+1}} \frac{\partial z_k^{l+1}}{\partial a_i^l} \frac{\partial a_i^l}{\partial z_i^l}\right)$$

$$= \frac{\partial a_i^l}{\partial z_i^l} \sum_k \left(\boxed{\frac{\partial C}{\partial z_k^{l+1}}} \frac{\partial z_k^{l+1}}{\partial a_i^l}\right) \qquad \delta_i^{l+1}$$



Gradient Descent for DNNs (cont.)

$$\partial C(\theta)/\partial z_i^l = \delta_i^l$$

- Idea: from L to 1

① Initialization: compute δ^L

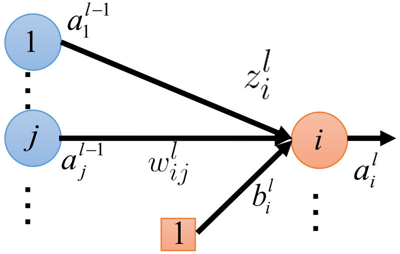
② **Compute δ^l based on δ^{l+1}**

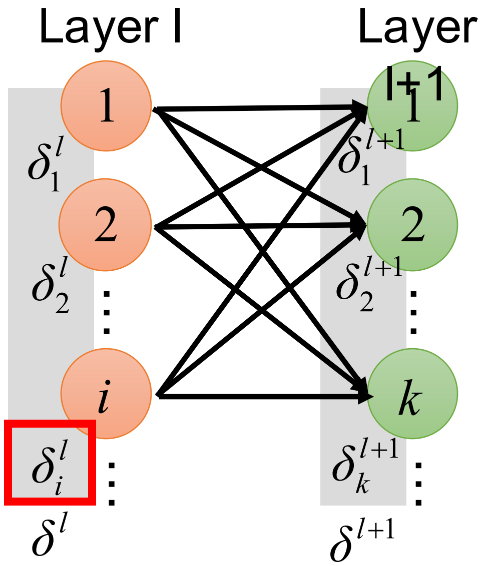
$$\delta_i^l = \frac{\partial a_i^l}{\partial z_i^l} \sum_k \frac{\partial z_k^{l+1}}{\partial a_i^l} \delta_k^{l+1}$$

$$= \sum_i w_{ki}^{l+1} a_i^l + b_k^{l+1}$$

$$= \sigma'(z_i) \sum_k \frac{\partial z_k^{l+1}}{\partial a_i^l} \delta_k^{l+1}$$

$$= \sigma'(z_i) \sum_k w_{ki}^{l+1} \delta_k^{l+1}$$



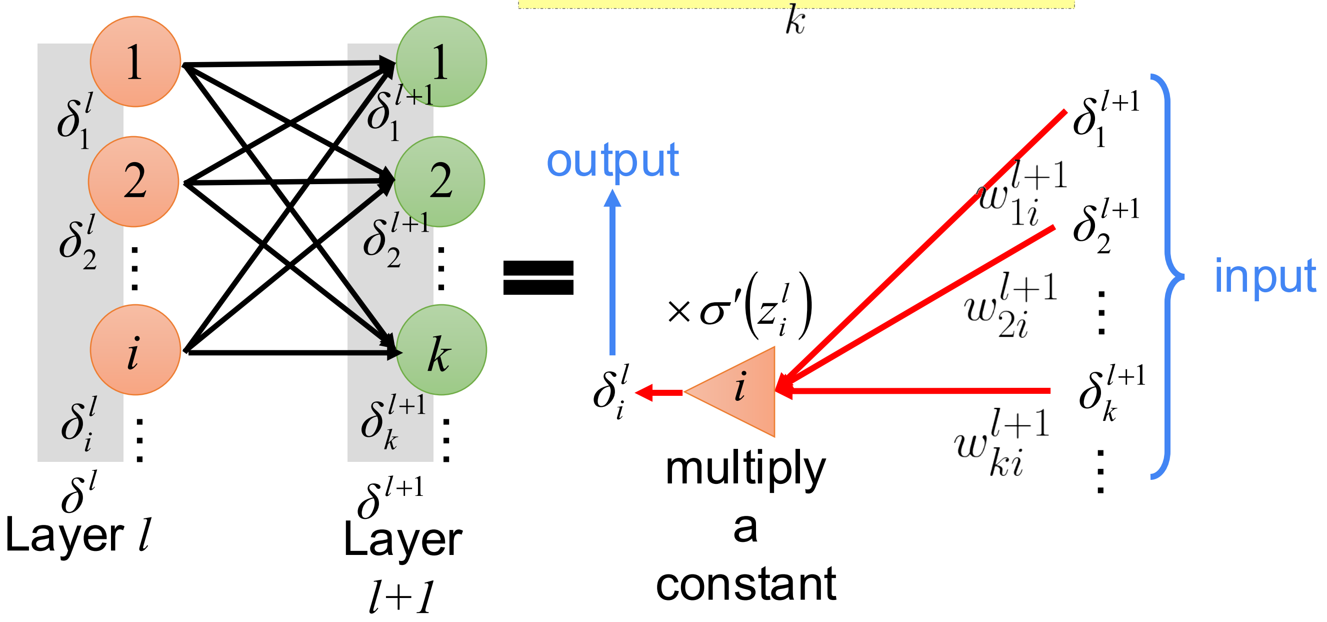


Gradient Descent for DNNs (cont.)

$$\partial C(\theta)/\partial z_i^l = \delta_i^l$$

- Rethink the propagation

$$\delta_i^l = \sigma'(z_i) \sum_k w_{ki}^{l+1} \delta_k^{l+1}$$



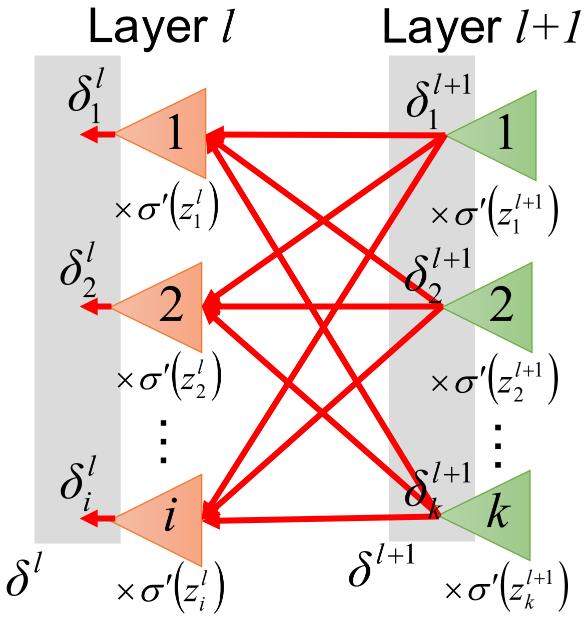
Gradient Descent for DNNs (cont.)

$$\partial C(\theta)/\partial z_i^l = \delta_i^l$$

$$\delta_i^l = \sigma'(z_i) \sum_k w_{ki}^{l+1} \delta_k^{l+1}$$

$$\sigma'(z^l) = \begin{bmatrix} \sigma'(z_1^l) \\ \sigma'(z_2^l) \\ \vdots \\ \sigma'(z_i^l) \\ \vdots \end{bmatrix}$$

$$\boxed{\delta^l = \sigma'(z^l) \odot (W^{l+1})^T \delta^{l+1}}$$



Gradient Descent for DNNs (cont.)

$$\partial C(\theta)/\partial z_i^l = \delta_i^l$$

$$\frac{\partial C(\theta)}{\partial w_{ij}^l} = \boxed{\frac{\partial C(\theta)}{\partial z_i^l}} \frac{\partial z_i^l}{\partial w_{ij}^l}$$

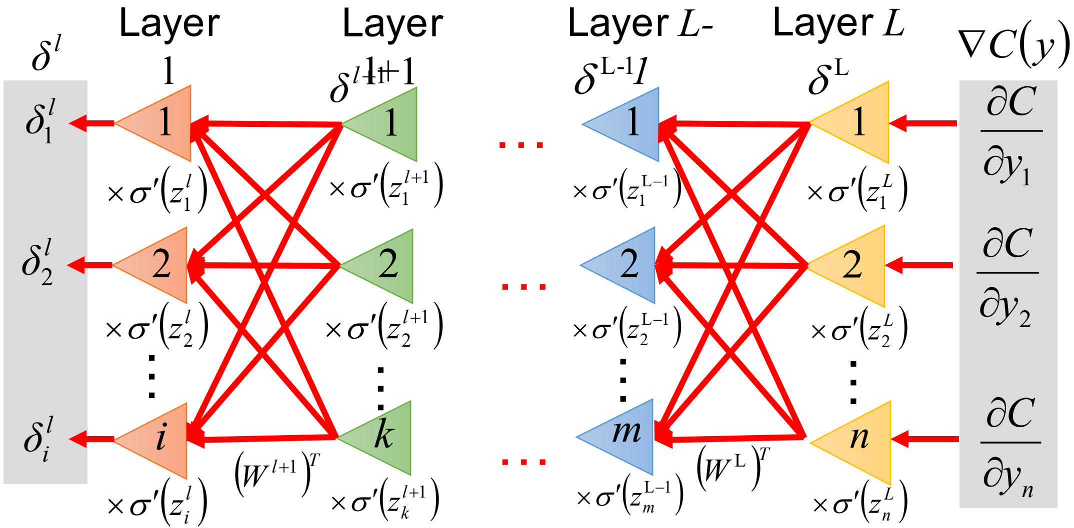
- Idea: from L to 1

① Initialization: compute δ^L

② Compute δ^{l-1} based on δ^l

$$\delta^L = \sigma'(z^L) \odot \nabla C(y)$$

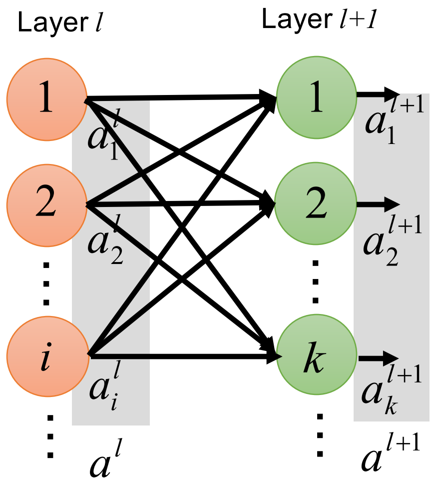
$$\delta^l = \sigma'(z^l) \odot (W^{l+1})^T \delta^{l+1}$$



Backpropagation

$$\frac{\partial C(\theta)}{\partial w_{ij}^l} = \frac{\partial C(\theta)}{\partial z_i^l} \boxed{\frac{\partial z_i^l}{\partial w_{ij}^l}}$$

$$\frac{\partial z_i^l}{\partial w_{ij}^l} = \begin{cases} a_j^{l-1} & , l > 1 \\ x_j & , l = 1 \end{cases}$$



Forward Pass

$$z^1 = W^1 x + b^1 \qquad a^1 = \sigma(z^1)$$

$$\vdots$$

$$z^l = W^l a^{l-1} + b^l \quad a^l = \sigma(z^l)$$

$$\vdots$$

Backpropagation

$$\frac{\partial C(\theta)}{\partial w_{ij}^l} = \frac{\partial C(\theta)}{\partial z_i^l} \frac{\partial z_i^l}{\partial w_{ij}^l}$$

$$\frac{\partial C(\theta)}{\partial z_i^l} = \delta_i^l$$

Backward Pass

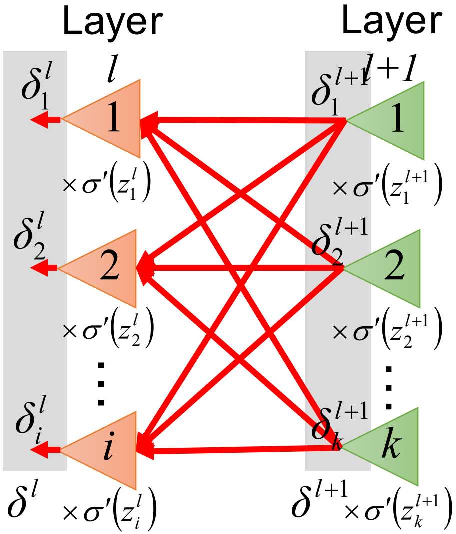
$$\delta^L = \sigma'(z^L) \odot \nabla C(y)$$

$$\delta^{L-1} = \sigma'(z^{L-1}) \odot (W^L)^T \delta^L$$

$$\vdots$$

$$\delta^l = \sigma'(z^l) \odot (W^{l+1})^T \delta^{l+1}$$

$$\vdots$$

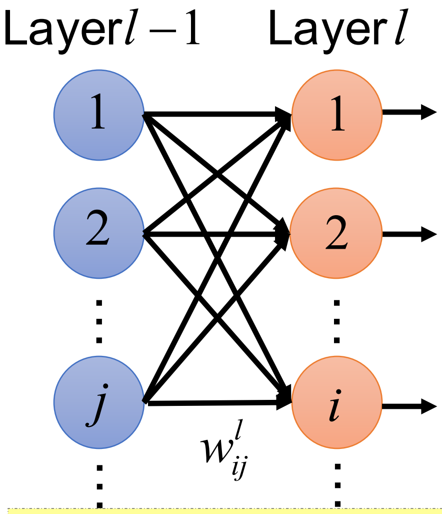


Gradient Descent – Summary

$$\frac{\partial C(\theta)}{\partial w_{ij}^l} = \boxed{\frac{\partial C(\theta)}{\partial z_i^l}} \boxed{\frac{\partial z_i^l}{\partial w_{ij}^l}}$$

$$\boxed{\delta_i^l}$$

$$\boxed{\begin{cases} a_j^{l-1} & l > 1 \\ x_j & l = 1 \end{cases}}$$



Backward Pass

$$\delta^L = \sigma'(z^L) \odot \nabla C(y)$$
$$\delta^{L-1} = \sigma'(z^{L-1}) \odot (W^L)^T \delta^L$$
$$\vdots$$
$$\delta^l = \sigma'(z^l) \odot (W^{l+1})^T \delta^{l+1}$$
$$\vdots$$

Forward Pass

$$z^1 = W^1 x + b^1$$
$$a^1 = \sigma(z^1)$$
$$\vdots$$
$$z^l = W^l a^{l-1} + b^l$$
$$a^l = \sigma(z^l)$$
$$\vdots$$

Compute the gradient based on terms from backward and forward passes

Topics for Thursday

- Distributional Similarity
- Sparse Word Representations
- Word Embeddings and Word2Vec
- Language Modeling
- Unexpected things we learn with word embeddings