



# M2177.003100

## Deep Learning

### [2: Neuron Modeling]

Electrical and Computer Engineering  
Seoul National University

© 2018 Sungroh Yoon. this material is for educational uses only. some contents are based on the material provided by other paper/book authors and may be copyrighted by them.

(last compiled at 22:10:00 on 2018/09/05)

# Outline

Introduction

Logistic Regression

Backprop Demystified

Minibatch Processing

Summary

# References

- *Deep Learning* by Goodfellow, Bengio and Courville [▶ Link](#)
  - ▶ Chapter 6
- *Pattern Recognition and Machine Learning* by Bishop
  - ▶ Chapter 5: Neural Networks
- online resources:
  - ▶ *Deep Learning Specialization (coursera)* [▶ Link](#)
  - ▶ *Stanford CS231n: CNN for Visual Recognition* [▶ Link](#)
  - ▶ *Machine Learning Yearning* [▶ Link](#)

# Outline

Introduction

Logistic Regression

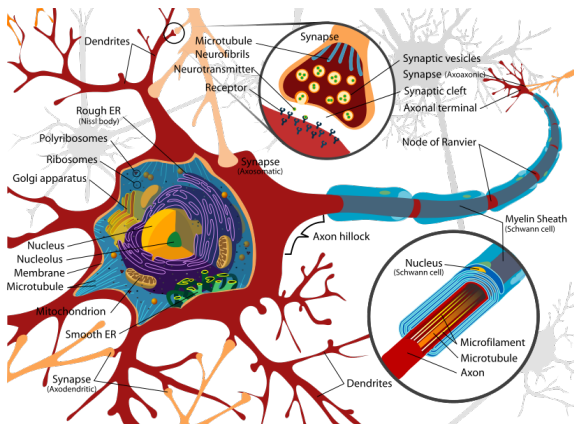
Backprop Demystified

Minibatch Processing

Summary

# Neuron

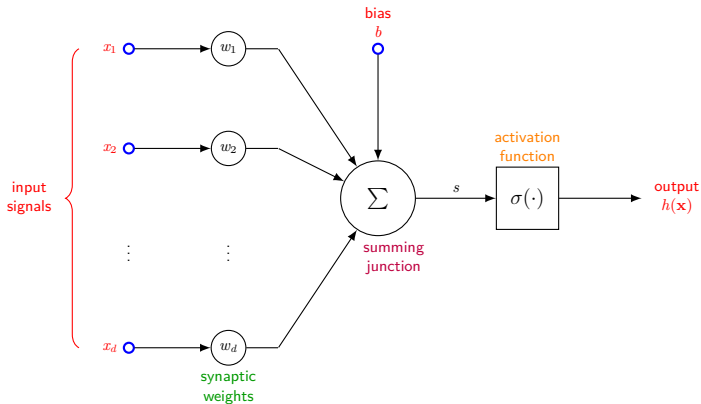
- electrically excitable cell in animal brains
  - ▶ processes and transmits information through electrical/chemical signals



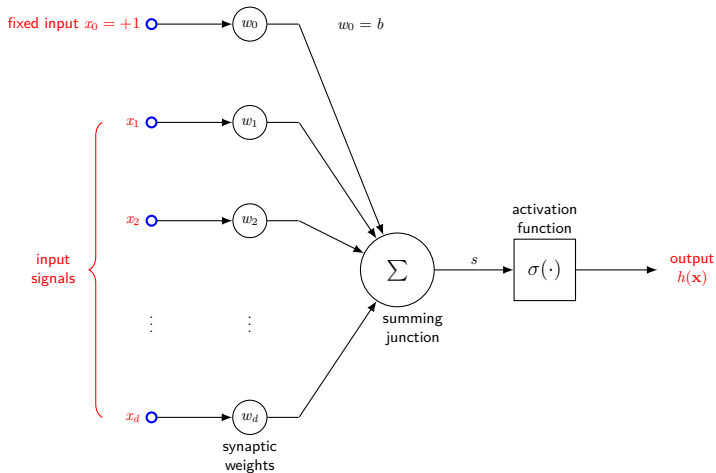
(source: <http://en.wikipedia.org/wiki/Neuron>)

# Modeling a neuron

- three basic elements
  1. **synapses (with weights)**
  2. **adder** (input vector  $\rightarrow$  scalar) adder => 가중치 계산
  3. **activation function** (possibly nonlinear)

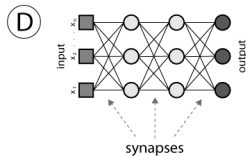
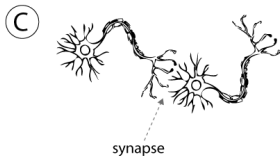
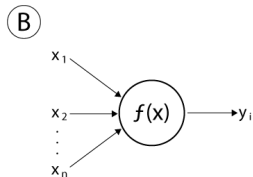
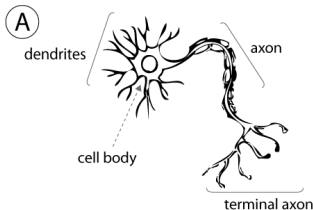


- alternative representation ( $w_0$  for bias  $b$ ):



# Human neuron vs artificial neuron

1개의 hidden layer가 여러개의 neuron으로 이루어져 있음



(a) human neuron

(b) artificial neuron

(c) biological neural net

(d) artificial neural net

(source: Matarollo, 2013)



# Outline

Introduction

Logistic Regression

Representation

Training by Backprop

Backprop Demystified

Minibatch Processing

Summary

# Linear models

- core of linear models
  - ▶ signal (weighted sum)  $z = \text{W}^{\text{transpose}} * \text{X}(\text{행렬 곱})$  : combines input variables linearly
  - ▶ we have seen two models based on this
- 1. linear regression    linear regression => 연속적결과
  - ▶ signal itself = output
  - ▶ for predicting real (unbounded) response
- 2. linear classification    linear classification => 0 or 1 결과
  - ▶ signal is thresholded at zero to produce  $\pm 1$  output
  - ▶ for binary decisions

# Logistic regression as a neuron model

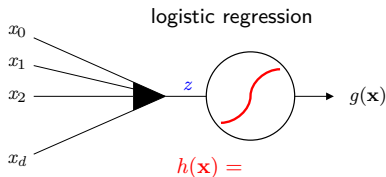
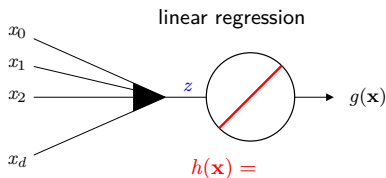
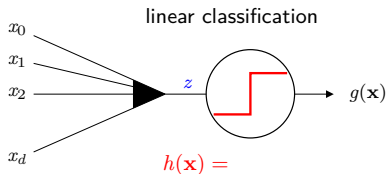
- we review the third linear model:
  - ▶ outputs *probability* of a binary response  
*e.g.* heart attack or not, dead or alive
  - ▶ returns 'soft labels' (probability)
- this model: called *logistic regression*
  - ▶ output: *real* (like regression) but *bounded* (like classification)
- comparison: linear classification vs logistic regression
  - ▶ both deal with a binary event
  - ▶ logistic regression: allowed to be uncertain
  - ⇒ intermediate values between 0 and 1 reflect this uncertainty
- in early neural nets: 

logistic regression unit = *neuron*

# Recall: linear models

- based on  
“signal”  $z$ :

$$z = \sum_{i=0}^d w_i x_i$$



# Outline

Introduction

Logistic Regression

Representation

Training by Backprop

Backprop Demystified

Minibatch Processing

Summary

# Formulation

- problem

- ▶ given:  $\mathbf{x} \in \mathbb{R}^{n_x}$
- ▶ want:  $\hat{y} = P(y = 1 \mid \mathbf{x})$

- model

- ▶  $\hat{y} =$
- ▶ parameters:

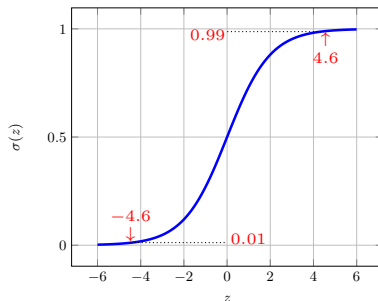
$$\mathbf{w} \in \mathbb{R}^{n_x}$$

$$b \in \mathbb{R}$$

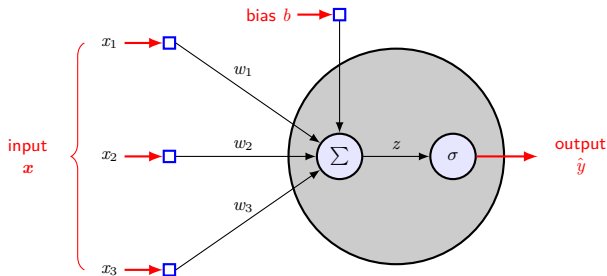
- ▶ (logistic) sigmoid:

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

$$\sigma'(z) = \sigma(z)(1 - \sigma(z))$$



- computational graph (simplified)



$$\hat{y} = \sigma(\underbrace{\mathbf{w}^\top \mathbf{x} + b}_{\triangleq z})$$

- weighted sum  $z$ : “signal”

$$\begin{aligned} z &= \sum_{i=1}^{n_x} w_i x_i + b \\ &= \mathbf{w}^\top \mathbf{x} + b \end{aligned}$$

# Probabilistic interpretation

- given training set drawn independently from  $p_{\text{data}}$

$$\mathbb{X} = \{(\mathbf{x}^{(1)}, y^{(1)}), (\mathbf{x}^{(2)}, y^{(2)}), \dots, (\mathbf{x}^{(m)}, y^{(m)})\}$$

- consider label  $y^{(i)} \in \{0, 1\}$  as (hard) probability

$$y \equiv P(y = 1 | \mathbf{x}) \Rightarrow P(y = 0 | \mathbf{x}) = 1 - P(y = 1 | \mathbf{x}) = 1 - y$$

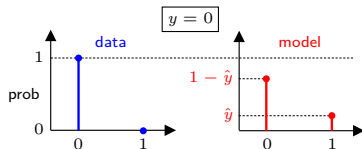
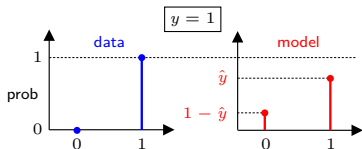
- then

- observed pmf  $\hat{p}_{\text{data}} \in \{ \underbrace{y}_{P(y=1|\mathbf{x})}, \underbrace{1-y}_{P(y=0|\mathbf{x})} \}$

← training data

- fitted pmf  $p_{\text{model}} \in \{ \underbrace{\hat{y}}_{P(y=1|\mathbf{x})}, \underbrace{1-\hat{y}}_{P(y=0|\mathbf{x})} \}$

← predicted by model

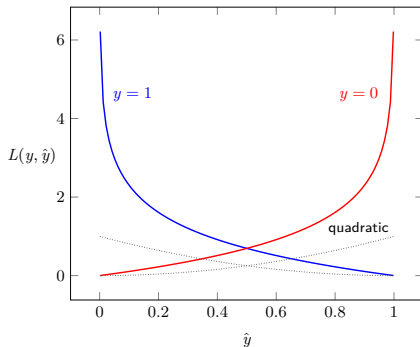




# Loss (= pointwise error)

- cross entropy for two pmfs  $p$  and  $q$

$$\begin{aligned} H(p, q) &= -\mathbb{E}_p[\log q] \\ &= -\sum_k p(k) \log q(k) \end{aligned}$$



- we use log loss (= logistic loss, cross-entropy loss):

$$L(y, \hat{y}) = -y \log \hat{y} - (1 - y) \log(1 - \hat{y})$$

- ▶ if  $y = 1 \Rightarrow L(y, \hat{y}) = -\log \hat{y} \Rightarrow$  want  $\hat{y}$  large ( $\hat{y} \rightarrow 1$ )
- ▶ if  $y = 0 \Rightarrow L(y, \hat{y}) = -\log(1 - \hat{y}) \Rightarrow$  want  $\hat{y}$  small ( $\hat{y} \rightarrow 0$ )

# Cost function

- simply average pointwise loss:

$$\begin{aligned} J(\mathbf{w}, b) &= -\mathbb{E}_{\mathbf{y} \sim \hat{p}_{\text{data}}(\mathbf{y} | \mathbf{x})} \log p_{\text{model}}(\mathbf{y} | \mathbf{x}) \\ &= \frac{1}{m} \sum_{i=1}^m L(\mathbf{y}^{(i)}, \hat{\mathbf{y}}^{(i)}) \\ &= -\frac{1}{m} \sum_{i=1}^m \left[ y^{(i)} \log \hat{y}^{(i)} + (1 - y^{(i)}) \log(1 - \hat{y}^{(i)}) \right] \end{aligned} \quad (1)$$

- bad news: no known closed-form equation to optimize
- good news: this cost function is convex  $\Rightarrow$  global minimum exists
- we will show:

$$\begin{aligned} \frac{\partial J}{\partial w_j} &= \frac{1}{m} \sum_{i=1}^m \left( \sigma(\mathbf{w}^\top \mathbf{x}^{(i)} + b) - y^{(i)} \right) x_j^{(i)} \\ \frac{\partial J}{\partial b} &= \frac{1}{m} \sum_{i=1}^m \left( \sigma(\mathbf{w}^\top \mathbf{x}^{(i)} + b) - y^{(i)} \right) \end{aligned}$$

# Outline

Introduction

Logistic Regression

Representation

Training by Backprop

Backprop Demystified

Minibatch Processing

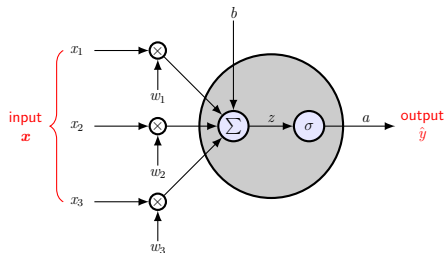
Summary

# Training a neuron

- find  $w$  and  $b$  that minimize cost function  $J(w, b)$  in (1)

- ▶ use iterative optimization

*i.e.* gradient descent



- repeat the following:

## 1. forward propagation

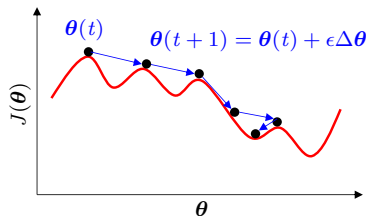
- ▶ pick an example  $(x, y)$  and feed  $x$  to the neuron
  - ▶ the net returns  $\hat{y} \Rightarrow$  generates loss  $L(y, \hat{y})$

## 2. backward propagation ("backprop"): gradient pump

- ▶ propagate error  $L(y, \hat{y})$  at the output to  $w$  and  $b$
  - ▶ update  $w$  and  $b$  using the propagated \_\_\_\_

# Gradient descent

- a general technique for minimizing twice-differentiable function  
e.g. cost function  $J(\theta)$
- idea:  $J(\theta)$  is a 'surface' in parameter space
  - ▶ start from somewhere on  $J$  (initial location is critical)
  - ▶ roll down the surface, decreasing  $J$  step by step
- two things to decide at each step
  - ▶ which direction?  $\Rightarrow \Delta\theta = -\nabla J(\theta)$
  - ▶ how much?  $\Rightarrow \epsilon$  ( learning rate )



- parameter update:

$$\begin{aligned}\theta(t+1) &= \theta(t) + \epsilon \Delta\theta \\ &= \theta(t) - \epsilon \nabla J(\theta)\end{aligned}$$

or

$$\boxed{\theta \leftarrow \theta - \epsilon \nabla J(\theta)}$$

# Gradient descent algorithm

---

**algorithm 1** gradient descent

---

- 1: initialize  $\theta$
  - 2: **while** stopping criterion not met **do**
  - 3:   sample  $m$  examples:  $\mathbb{X}_m = \{(\mathbf{x}^{(1)}, y^{(1)}), \dots, (\mathbf{x}^{(m)}, y^{(m)})\}$
  - 4:   compute gradient estimate:  $\hat{\mathbf{g}} \leftarrow \frac{1}{m} \nabla_{\theta} \sum_{i=1}^m L(y^{(m)}, \hat{y}^{(m)})$     $\triangleright m$  forward props
  - 5:   apply update:  $\theta \leftarrow \theta - \epsilon \hat{\mathbf{g}}$     $\triangleright \epsilon$ : learning rate
  - 6: **end while**
- 

- three variants ( $N$ : total number of examples)
  - ▶  $m = 1$ : stochastic gradient descent (sgd)
  - ▶  $1 < m < N$ : minibatch sgd   (typical  $m$ : 64, 128, 256, 512)
  - ▶  $m = N$ : batch gradient descent

# Training logistic regression by backprop

- minimize the cost function by gradient descent

$$J(\mathbf{w}, b) = -\frac{1}{m} \sum_{i=1}^m \left[ y^{(i)} \log \hat{y}^{(i)} + (1 - y^{(i)}) \log(1 - \hat{y}^{(i)}) \right]$$

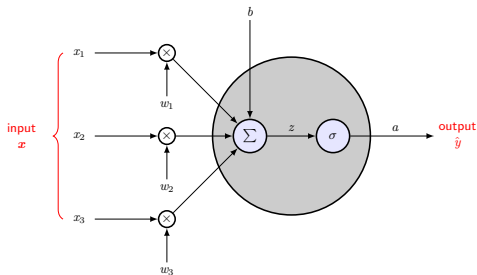
- repeat over training examples

►  $\epsilon$ : learning rate

$$\mathbf{w} \leftarrow \mathbf{w} - \epsilon \nabla_{\mathbf{w}} J$$

$$b \leftarrow b - \epsilon \nabla_b J$$

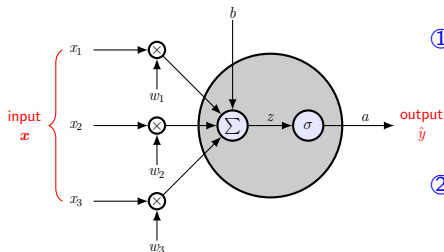
- forward prop:



$$z = \mathbf{w}^\top \mathbf{x} + b$$

$$\hat{y} = \sigma(z) \triangleq a$$

- some math (in the order of information flow):



① activation  $a$

$$\hat{y} = \sigma(\mathbf{w}^\top \mathbf{x} + b) \triangleq a$$

② cost function  $J$

$$\begin{aligned} J &= -y \log \hat{y} - (1 - y) \log(1 - \hat{y}) \\ &= -y \log a - (1 - y) \log(1 - a) \end{aligned}$$

③ gradient at activation  $a$  (output)

$$\frac{\partial J}{\partial a} = \frac{\partial J}{\partial \hat{y}} = -\frac{y}{a} + \frac{(1 - y)}{1 - a}$$

④ local gradient at sigmoid  $\sigma$

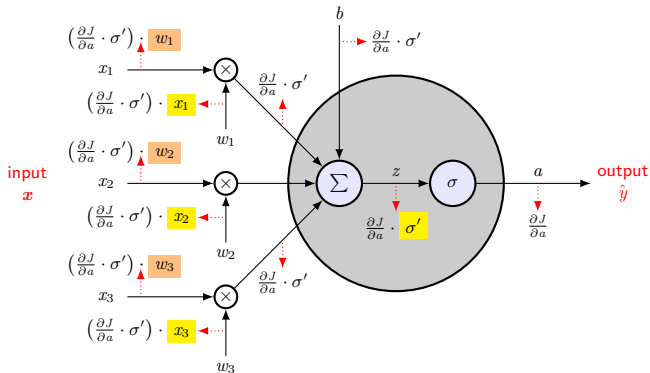
$$\sigma' = \sigma(1 - \sigma) = a(1 - a)$$

⑤ gradient at signal  $z$

$$\begin{aligned} \frac{\partial J}{\partial z} &= \frac{\partial J}{\partial a} \frac{\partial a}{\partial z} = \frac{\partial J}{\partial a} \cdot \sigma' \\ &= \left( -\frac{y}{a} + \frac{(1 - y)}{1 - a} \right) \cdot a(1 - a) \\ &= a - y \triangleq \delta \quad \text{[ "error" ]} \end{aligned}$$

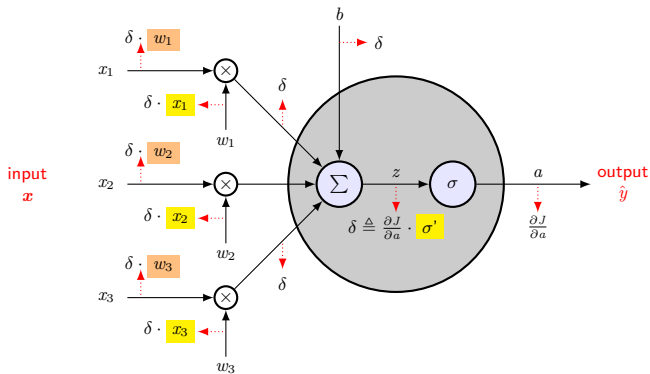


- backprop:



$$\frac{\partial J}{\partial z} = \frac{\partial J}{\partial a} \cdot \sigma' = a - y \triangleq \delta$$

- backprop (simplified):



$$\frac{\partial J}{\partial z} = \frac{\partial J}{\partial a} \cdot \sigma' = a - y \triangleq \delta$$

# SGD equations (1 example)

- cost function

$$J(\mathbf{w}, b) = L(\mathbf{y}, \mathbf{a}) = -y \log a - (1 - y) \log(1 - a)$$

- compute gradient:

$$\frac{\partial J}{\partial w_1} = x_1 \cdot \frac{\partial J}{\partial z} = x_1 ( \quad ) = x_1 \delta$$

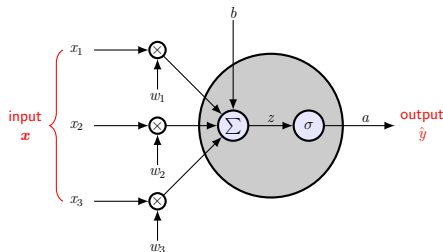
$$\frac{\partial J}{\partial w_2} = x_2 \cdot \frac{\partial J}{\partial z} = x_2 (a - y) = x_2 \delta$$

$$\frac{\partial J}{\partial b} = \frac{\partial J}{\partial z} = (a - y) = \delta$$

- apply update:

$$w_i \leftarrow w_i - \epsilon \frac{\partial J}{\partial w_i} = w_i - \epsilon \cdot x_i \cdot \delta = w_i - \epsilon x_i (a - y)$$

$$b \leftarrow b - \epsilon \frac{\partial J}{\partial b} = b - \epsilon \cdot \delta = b - \epsilon (a - y)$$



# Minibatch SGD equations

- cost function

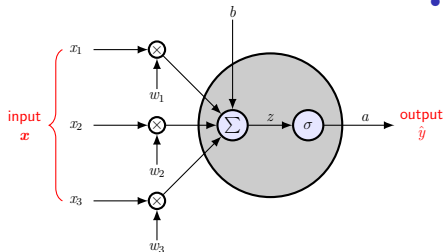
$$J(\mathbf{w}, b) = \frac{1}{m} \sum_{i=1}^m L(y^{(i)}, \mathbf{a}^{(i)})$$

- signal, activation, delta error

$$z^{(i)} = \mathbf{w}^\top \mathbf{x}^{(i)} + b$$

$$a^{(i)} = \sigma(z^{(i)})$$

$$\delta^{(i)} = a^{(i)} - y^{(i)}$$



- compute gradient:

$$\frac{\partial J}{\partial w_1} = \frac{1}{m} \sum_{i=1}^m x_1^{(i)} \delta^{(i)}$$

$$\frac{\partial J}{\partial w_2} = \frac{1}{m} \sum_{i=1}^m x_2^{(i)} \delta^{(i)}$$

$$\frac{\partial J}{\partial b} = \frac{1}{m} \sum_{i=1}^m \delta^{(i)}$$

- apply update:

$$w_1 \leftarrow w_1 - \epsilon \frac{\partial J}{\partial w_1}$$

$$w_2 \leftarrow w_2 - \epsilon \frac{\partial J}{\partial w_2}$$

$$b \leftarrow b - \epsilon \frac{\partial J}{\partial b}$$

# Outline

Introduction

Logistic Regression

Backprop Demystified

Minibatch Processing

Summary

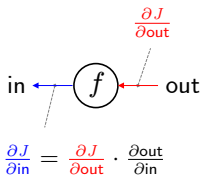
# Single-gate backprop

- $\text{out} = f(\text{in})$

forward



backprop

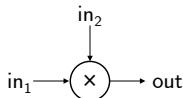


$$\begin{aligned} \frac{\partial J}{\partial \text{in}} &= \underbrace{\frac{\partial J}{\partial \text{out}}}_{\text{output gradient}} \cdot \underbrace{\frac{\partial \text{out}}{\partial \text{in}}}_{\text{local gradient}} \\ &= \frac{\partial J}{\partial \text{out}} \cdot f'(\text{in}) \end{aligned}$$

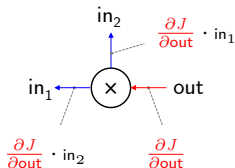
# Multiplication

- $\text{out} = \text{in}_1 \cdot \text{in}_2$

forward



backprop



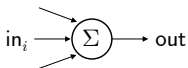
$$\begin{aligned}\frac{\partial J}{\partial \text{in}_1} &= \frac{\partial J}{\partial \text{out}} \cdot \frac{\partial \text{out}}{\partial \text{in}_1} \\ &= \underbrace{\frac{\partial J}{\partial \text{out}}}_{\text{output gradient}} \cdot \underbrace{\text{in}_2}_{\text{local gradient}}\end{aligned}$$

$$\begin{aligned}\frac{\partial J}{\partial \text{in}_2} &= \frac{\partial J}{\partial \text{out}} \cdot \frac{\partial \text{out}}{\partial \text{in}_2} \\ &= \frac{\partial J}{\partial \text{out}} \cdot \text{in}_1\end{aligned}$$

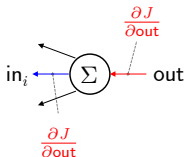
# Summation

- $\text{out} = \sum_i \text{in}_i$

forward



backprop



$$\begin{aligned}\frac{\partial J}{\partial \text{in}_i} &= \frac{\partial J}{\partial \text{out}} \cdot \frac{\partial \text{out}}{\partial \text{in}_i} \\ &= \underbrace{\frac{\partial J}{\partial \text{out}}}_{\text{output gradient}} \cdot \underbrace{1}_{\text{local gradient}} \\ &= \frac{\partial J}{\partial \text{out}}\end{aligned}$$

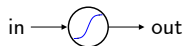
- $\text{sum (forward)} \Leftrightarrow \text{_____ (backprop)}$



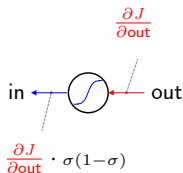
# Sigmoid

- $\text{out} = \sigma(\text{in})$

forward



backprop

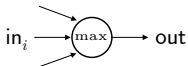


$$\begin{aligned}\frac{\partial J}{\partial \text{in}} &= \frac{\partial J}{\partial \text{out}} \cdot \frac{\partial \text{out}}{\partial \text{in}} \\ &= \underbrace{\frac{\partial J}{\partial \text{out}}}_{\text{output gradient}} \cdot \underbrace{\sigma'(\text{in})}_{\text{local gradient}} \\ &= \frac{\partial J}{\partial \text{out}} \cdot [\sigma(\text{in}) (1 - \sigma(\text{in}))]\end{aligned}$$

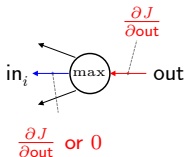
# Max

- $\text{out} = \max_i \{\text{in}_i\}$

forward



backprop



$$\frac{\partial J}{\partial \text{in}_i} = \frac{\partial J}{\partial \text{out}} \cdot \underbrace{\frac{\partial \text{out}}{\partial \text{in}}}_{1 \text{ or } 0}$$

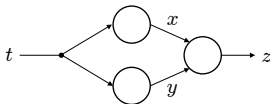
$$= \begin{cases} \frac{\partial J}{\partial \text{out}} & \text{if } \text{in}_i \text{ is max} \\ 0 & \text{otherwise} \end{cases}$$

- $\text{max (forward)} \Leftrightarrow \text{mux (backprop)}$

# Backprop through fanout

- multivariable chain rule

forward

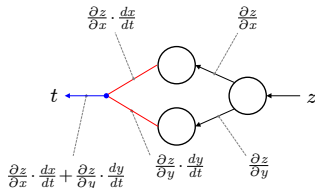


let

$$x = x(t), \quad y = y(t)$$

$$z = f(x, y)$$

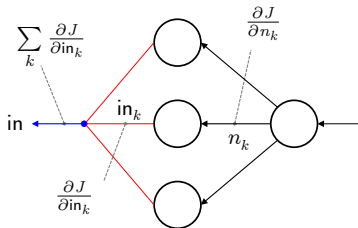
backprop



then

$$\frac{\partial z}{\partial t} = \frac{\partial z}{\partial x} \cdot \frac{\partial x}{\partial t} + \frac{\partial z}{\partial y} \cdot \frac{\partial y}{\partial t}$$

- fanout



assuming

$$\mathcal{E} = f(n_1, \dots, n_k, \dots)$$

and

$$n_k = n_k(\text{in})$$

gives

$$\begin{aligned} \frac{\partial J}{\partial in} &= \sum_k \frac{\partial J}{\partial n_k} \cdot \frac{\partial n_k}{\partial in} \\ &= \sum_k \frac{\partial J}{\partial in_k} \end{aligned}$$

where

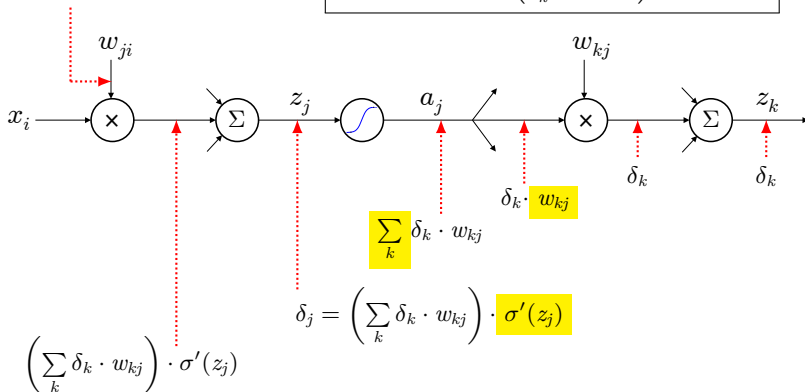
$$in_k \triangleq \text{input to } n_k$$

- fanout (forward)  $\Leftrightarrow$  \_\_\_\_ (backprop)

# Example

- computing  $\frac{\partial J}{\partial w_{ji}}$

$$\left( \sum_k \delta_k \cdot w_{kj} \right) \cdot \sigma'(z_j) \cdot x_i \implies \boxed{\frac{\partial J}{\partial w_{ji}} = \delta_j \cdot x_i = \left( \sum_k \delta_k \cdot w_{kj} \right) \cdot \sigma'(z_j) \cdot x_i}$$



# Outline

Introduction

Logistic Regression

Backprop Demystified

Minibatch Processing

Summary

# Arranging minibatch

- two options to arrange  $m$  examples

▶ in **columns**

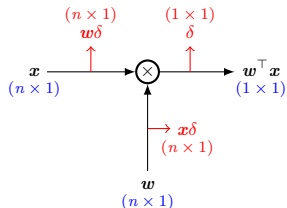
$$\mathbf{X}_{||} = \begin{bmatrix} \left. \begin{array}{c} | \\ \mathbf{x}^{(1)} \\ | \end{array} \right| & \left. \begin{array}{c} | \\ \mathbf{x}^{(2)} \\ | \end{array} \right| & \dots & \left. \begin{array}{c} | \\ \mathbf{x}^{(m)} \\ | \end{array} \right| \end{bmatrix}$$

▶ in **rows** (“ ”)

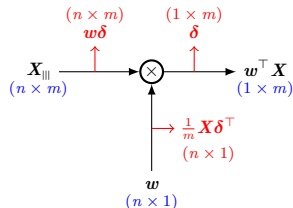
$$\mathbf{X}_{\equiv} = \begin{bmatrix} \text{---} & \mathbf{x}^{(1)\top} & \text{---} \\ \text{---} & \mathbf{x}^{(2)\top} & \text{---} \\ & \vdots & \\ \text{---} & \mathbf{x}^{(m)\top} & \text{---} \end{bmatrix}$$

# Weighted sum

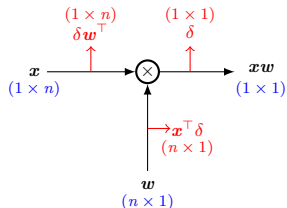
- 1 example (column):



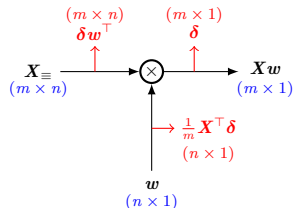
- size- $m$  minibatch (column):



- 1 example (row):



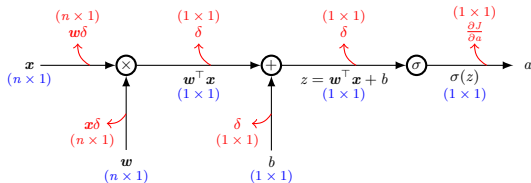
- size- $m$  minibatch (row):



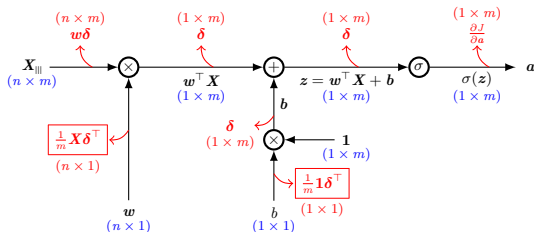


# Forward/backward prop (column-wise)

- 1 example:



- size- $m$  minibatch:




---

## algorithm 2

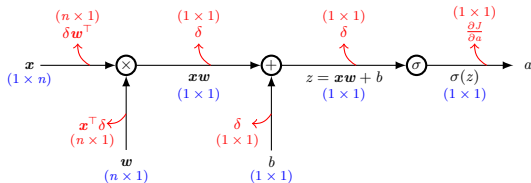
### logistic regression (col)

---

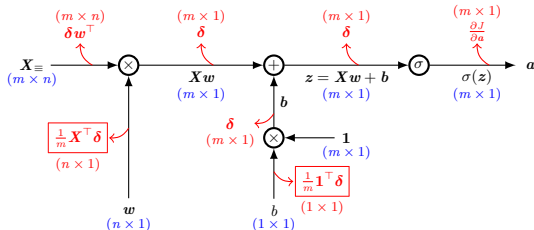
- 1: initialize  $w, b$
  - 2: **while** necessary **do**
  - 3:    $z = w^\top X + b$
  - 4:    $a = \sigma(z)$
  - 5:    $\frac{\partial J}{\partial z} \triangleq \delta = a - y$
  - 6:    $\frac{\partial J}{\partial w} = \frac{1}{m} X \delta^\top$
  - 7:    $\frac{\partial J}{\partial b} = \frac{1}{m} \mathbf{1} \delta^\top$
  - 8:    $w \leftarrow w - \epsilon \frac{\partial J}{\partial w}$
  - 9:    $b \leftarrow b - \epsilon \frac{\partial J}{\partial b}$
  - 10: **end while**
  - 11: **return**  $w, b$
-

# Forward/backward prop (row-wise)

- 1 example:



- size- $m$  minibatch:




---

## algorithm 3

### logistic regression (row)

---

- 1: initialize  $w, b$
  - 2: **while** necessary **do**
  - 3:      $z = Xw + b$
  - 4:      $a = \sigma(z)$
  - 5:      $\frac{\partial J}{\partial z} \triangleq \delta = a - y$
  - 6:      $\frac{\partial J}{\partial w} =$
  - 7:      $\frac{\partial J}{\partial b} =$
  - 8:      $w \leftarrow w - \epsilon \frac{\partial J}{\partial w}$
  - 9:      $b \leftarrow b - \epsilon \frac{\partial J}{\partial b}$
  - 10: **end while**
  - 11: return  $w, b$
-

# Outline

Introduction

Logistic Regression

Backprop Demystified

Minibatch Processing

Summary

# Summary

- neuron: brain cell for information processing
  - ▶ model: synaptic weights, adder, nonlinear activation function
- logistic regression: a linear model to probability estimation
  - ▶ parameterized by weights and bias:  $\theta = (w, b)$
  - ▶ used as a neuron model in early neural nets
  - ▶ log loss:  $L(y, \hat{y}) = -y \log \hat{y} - (1 - y) \log(1 - \hat{y})$
  - ▶ cost function  $J(\theta)$ : average loss from training examples
  - ▶ training: iterative optimization (such as gradient descent)
- gradient descent: a general, iterative optimization technique
  - ▶ update equation:  $\theta \leftarrow \theta - \epsilon \nabla_{\theta} J(\theta)$
  - ▶ unit of gradient estimation: batch (all), minibatch ( $m$ ), stochastic (1)
  - ▶ neural nets: gradients are provided by back propagation (backprop)