M2177.003100
Deep Learning

[2: Neuron Modeling]

Electrical and Computer Engineering
Seoul National University

© 2020 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 11:02:00 on 2020/09/06)

Outline

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

Outline

Neuron

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

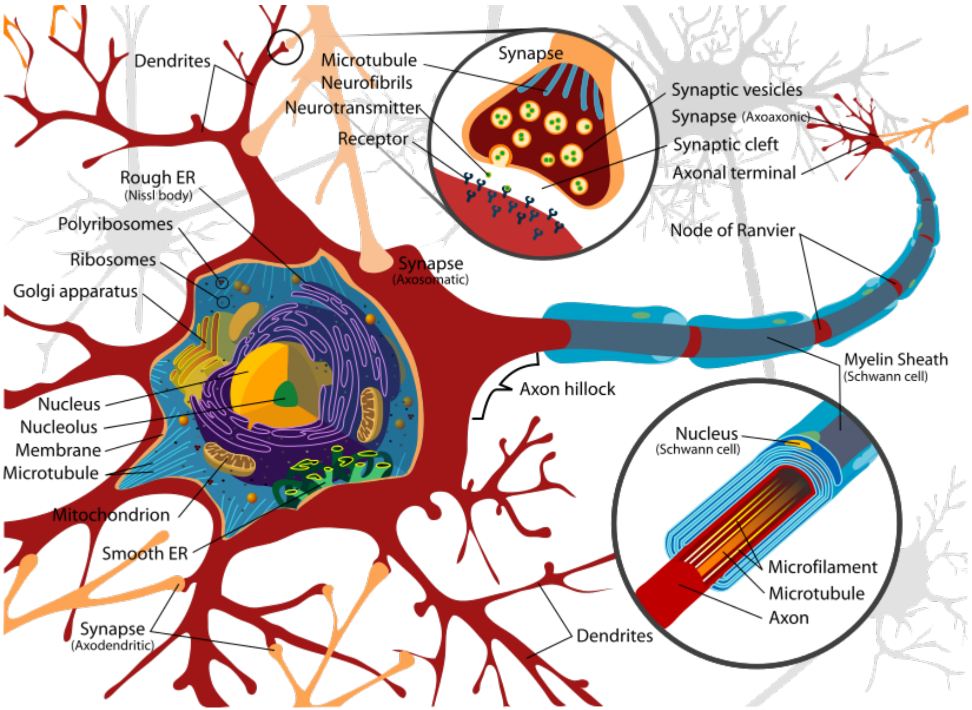


image source: `http://en.wikipedia.org/wiki/Neuron`

Modeling a neuron

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

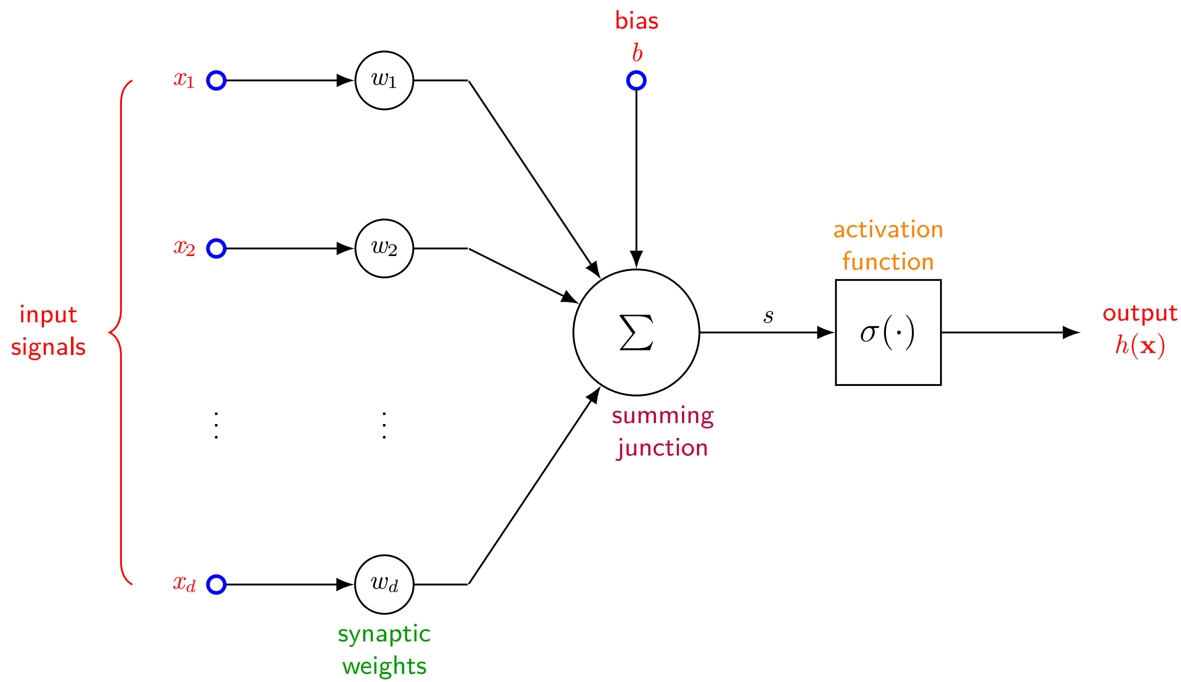


image source: S. Haykin, *Neural Networks and Learning Machines*. Pearson Education, 3rd ed., 2010

- alternative representation (w_0 for bias b):

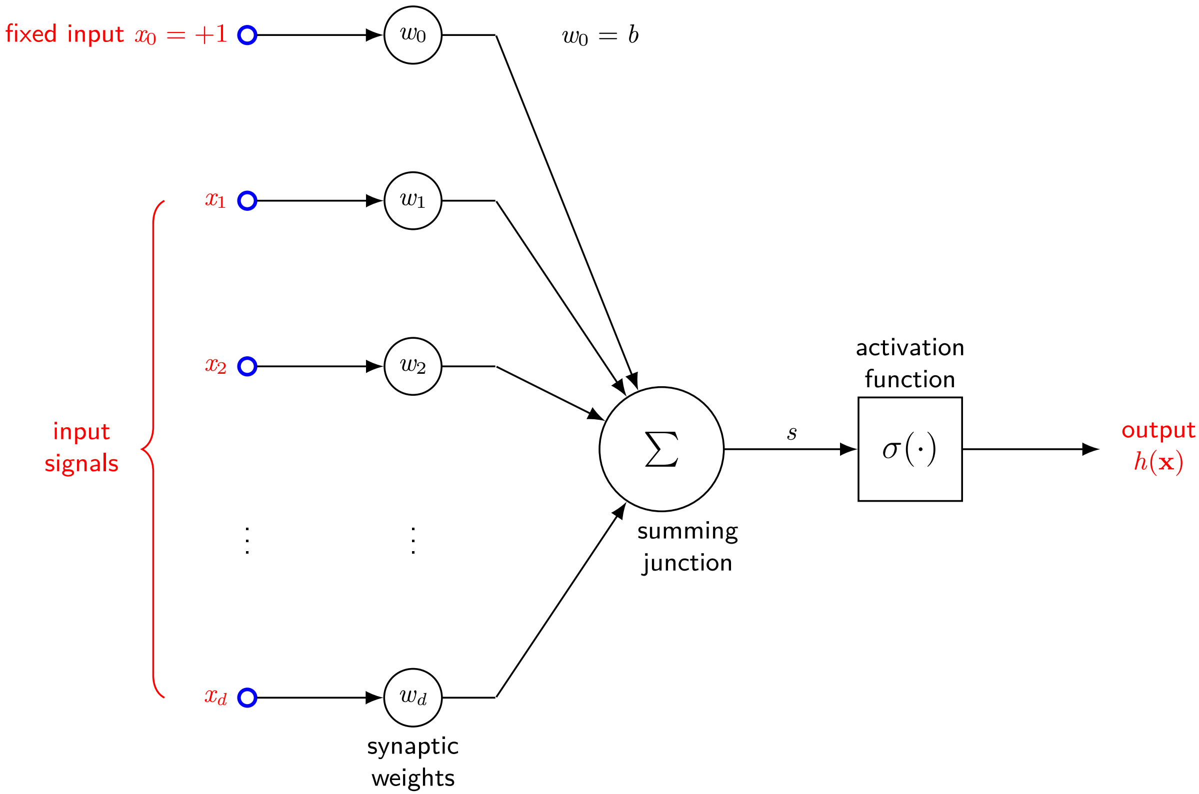
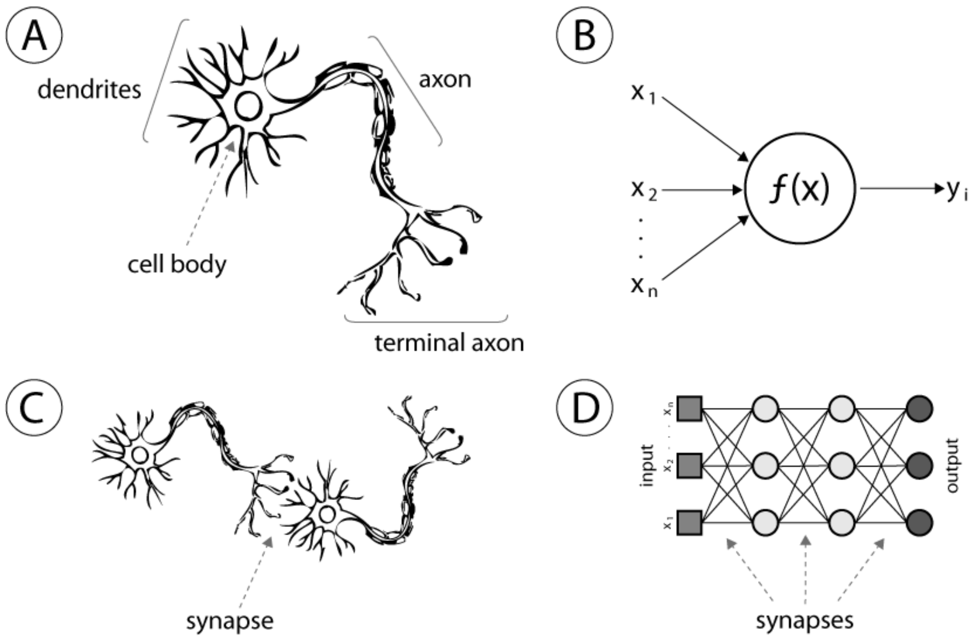


image source: S. Haykin, *Neural Networks and Learning Machines*. Pearson Education, 3rd ed., 2010

Human neuron vs artificial neuron



(a) human neuron (b) artificial neuron

(c) biological neural net (d) artificial neural net

image source: V. G. Maltarollo, K. M. Honório, and A. B. F. da Silva, "Applications of artificial neural networks in chemical problems," *Artificial Neural Networks-Architectures and Applications*, pp. 203–223, 2013

Outline

Linear models

- core of linear models
  - signal (weighted sum) $z = \underline{w^T x}$: combines input variables linearly
  - we have seen two models based on this

1. linear regression
   - signal itself = output
   - for predicting real (unbounded) response

2. linear classification
   - signal is thresholded at zero to produce ± 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

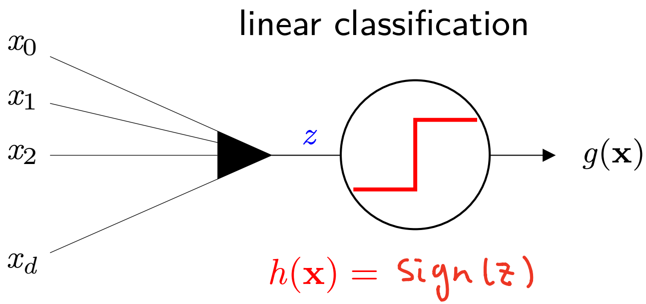
  $\Rightarrow$ intermediate values between 0 and 1 reflect this uncertainty

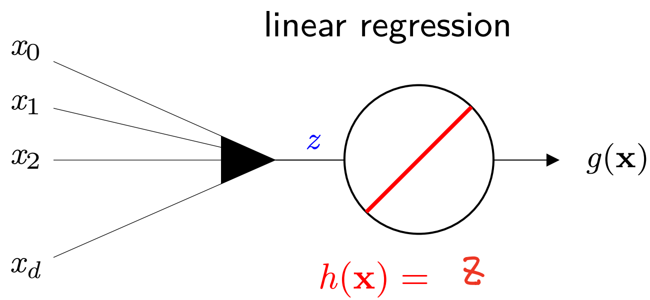
- in early neural nets: $\boxed{\text{logistic regression unit} = \text{neuron}}$

Recall: linear models

- based on "signal" z:

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





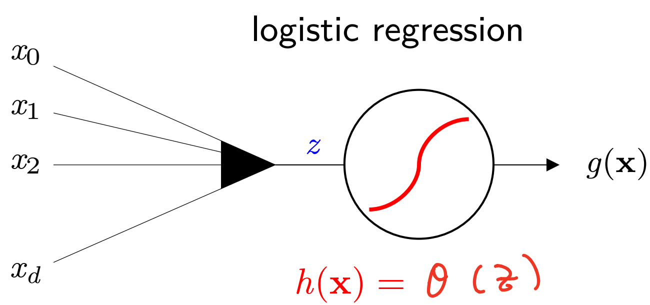


image source: Y. S. Abu-Mostafa, M. Magdon-Ismail, and H.-T. Lin, *Learning from Data*. AMLBook, 2012

Outline

Formulation

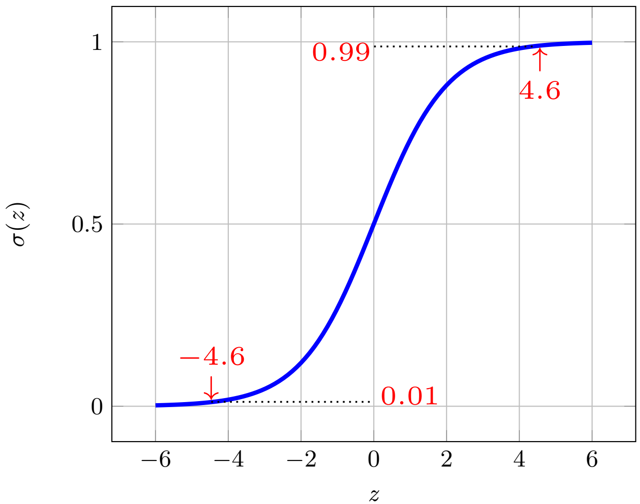
- problem
  - given: $\boldsymbol{x} \in \mathbb{R}^{n_x}$
  - want: $\hat{y} = P(y = 1 \mid \boldsymbol{x})$

- model
  - $\hat{y} =$ $\sigma(w^T x + b)$
  - parameters:

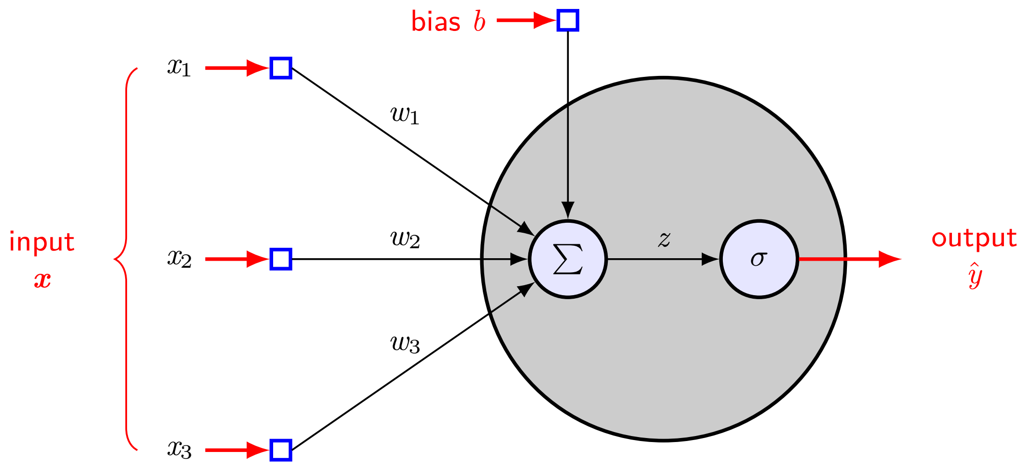
$$\theta \left\{ \begin{array}{l} \boldsymbol{w} \in \mathbb{R}^{n_x} \\ b \in \mathbb{R} \end{array} \right.$$

  - (logistic) sigmoid:

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



- computational graph (simplified)



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

- weighted sum z: "signal"

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

Probabilistic interpretation

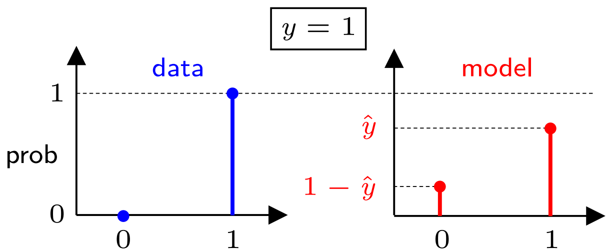
- given training set drawn independently from p_{data}

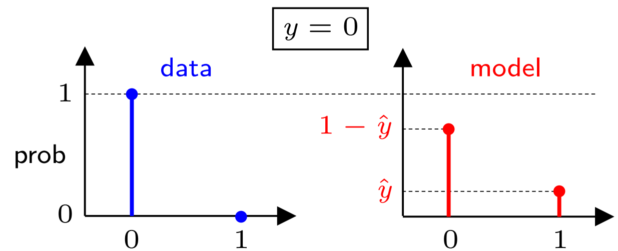
$$\mathbb{X} = \left\{ (\boldsymbol{x}^{(1)}, y^{(1)}), (\boldsymbol{x}^{(2)}, y^{(2)}), \ldots, (\boldsymbol{x}^{(m)}, y^{(m)}) \right\}$$

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

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

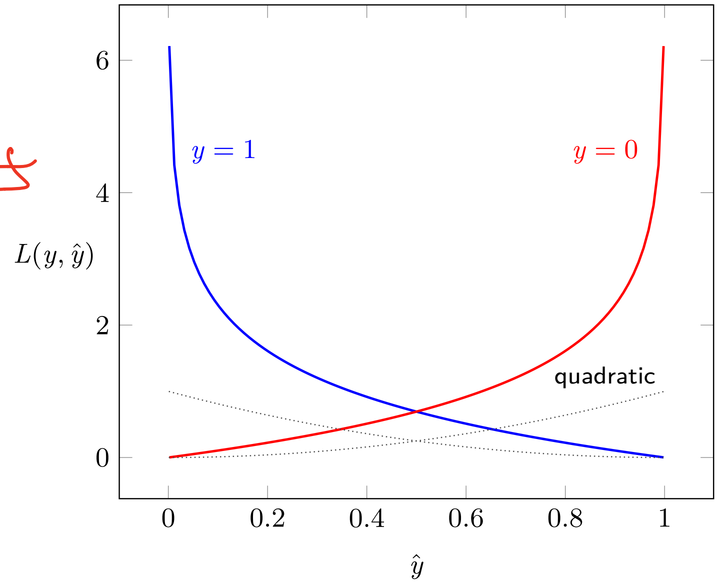
- then
  - observed pmf $\hat{p}_{\text{data}} \in \{ \underbrace{y}_{P(y=1|\boldsymbol{x})} , \underbrace{1 - y}_{P(y=0|\boldsymbol{x})} \}$ ← training data
  - fitted pmf $p_{\text{model}} \in \{ \underbrace{\hat{y}}_{P(y=1|\boldsymbol{x})} , \underbrace{1 - \hat{y}}_{P(y=0|\boldsymbol{x})} \}$ ← predicted by model





Loss (= pointwise error)

log loss >> MSE



- cross entropy for two pmfs p and q

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

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

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

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

Cost function

- simply average pointwise loss:

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

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

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

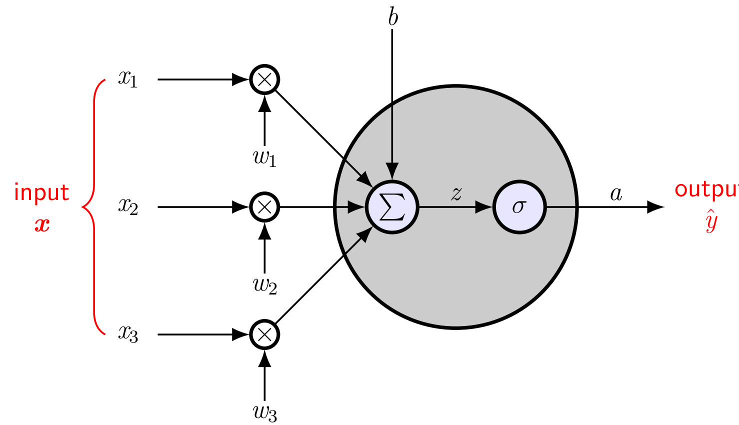
$$\frac{\partial J}{\partial b} = \frac{1}{m} \sum_{i=1}^{m} \left(\sigma(\boldsymbol{w}^\top \boldsymbol{x}^{(i)} + b) - y^{(i)} \right)$$

Outline

Training a neuron

- find $\boldsymbol{w}$ and b that minimize cost function $J(\boldsymbol{w}, b)$ in (1)
  - ▸ use iterative optimization

  i.e. gradient descent



- repeat the following:

1. forward propagation
   - ▸ pick an example $(\boldsymbol{x}, y)$ and feed $\boldsymbol{x}$ 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 $\boldsymbol{w}$ and b
   - ▸ update $\boldsymbol{w}$ and b using the propagated error

Gradient descent

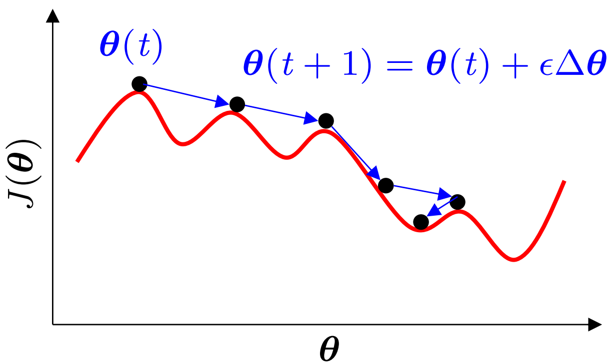
- a general technique for minimizing differentiable function

  e.g. cost function $J(\boldsymbol{\theta})$

- idea: $J(\boldsymbol{\theta})$ is a 'surface' in parameter space
  - start from somewhere on J (initial location is critical)
  - roll down the surface, decreasing J step by step

$-\frac{\partial J}{\partial w}$

- two things to decide at each step
  - which direction? $\Rightarrow \quad \Delta\boldsymbol{\theta} = -\nabla J(\boldsymbol{\theta})$
  - how much? $\Rightarrow \quad \epsilon$ (learning rate)



- parameter update:

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

or

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

image modified from: C. Angermueller, T. Pärnamaa, L. Parts, and O. Stegle, "Deep learning for computational biology," *Molecular Systems Biology*, vol. 12, no. 7, 2016

Gradient descent algorithm

algorithm 1 gradient descent

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

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

Training logistic regression by backprop

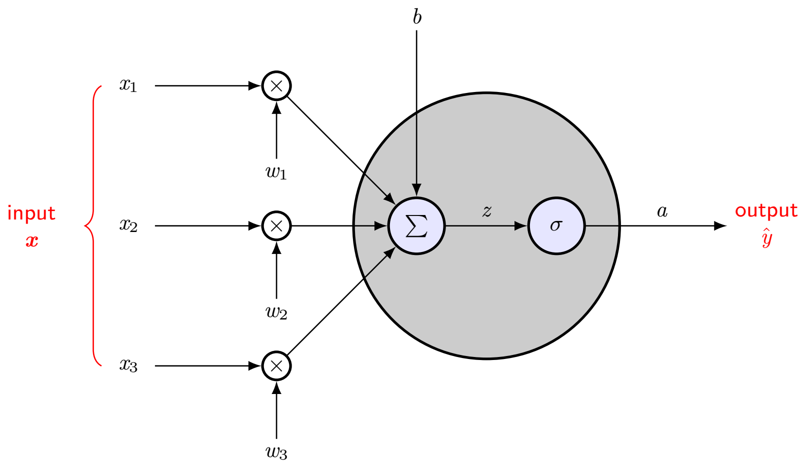
- minimize the cost function by gradient descent

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

- repeat over training examples
  - ϵ: learning rate

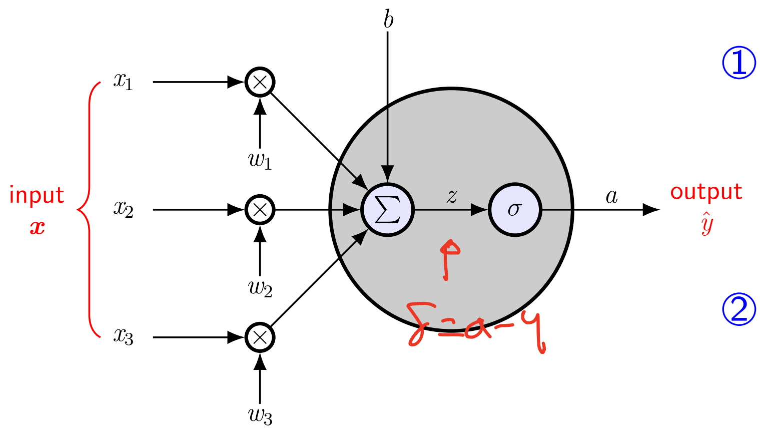
$$\boldsymbol{w} \leftarrow \boldsymbol{w} - \epsilon \nabla_{\boldsymbol{w}} J$$
$$b \leftarrow b - \epsilon \nabla_b J$$

- forward prop:



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

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



① $\boxed{\text{activation } a}$

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

② $\boxed{\text{cost function } J}$

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

③ $\boxed{\text{gradient at activation } a \text{ (output)}}$

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

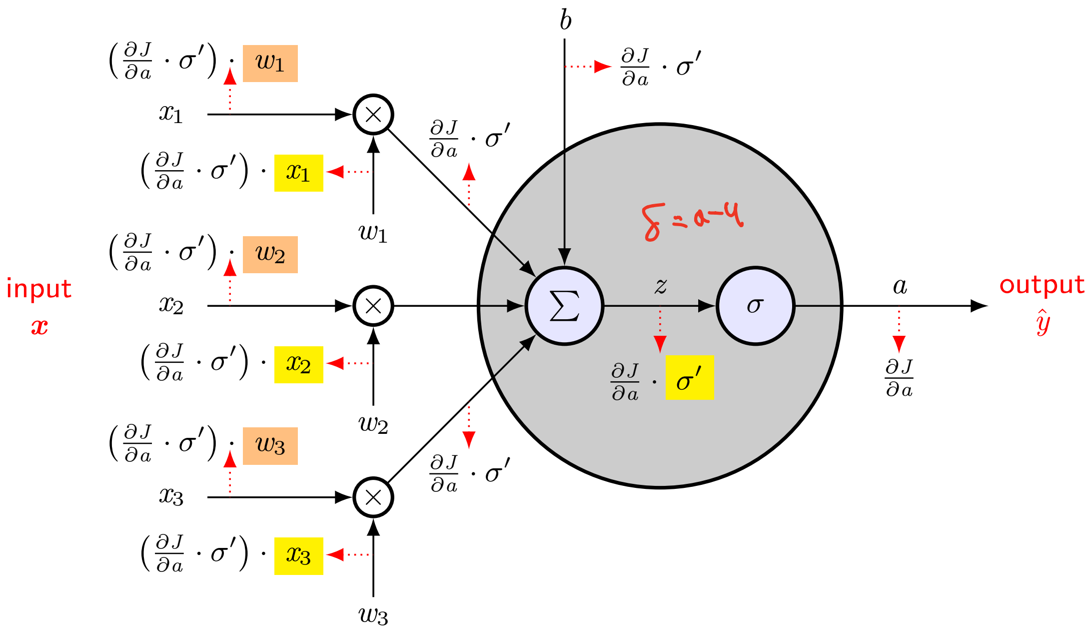
④ $\boxed{\text{local gradient at sigmoid } \sigma}$

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

⑤ $\boxed{\text{gradient at signal } z}$

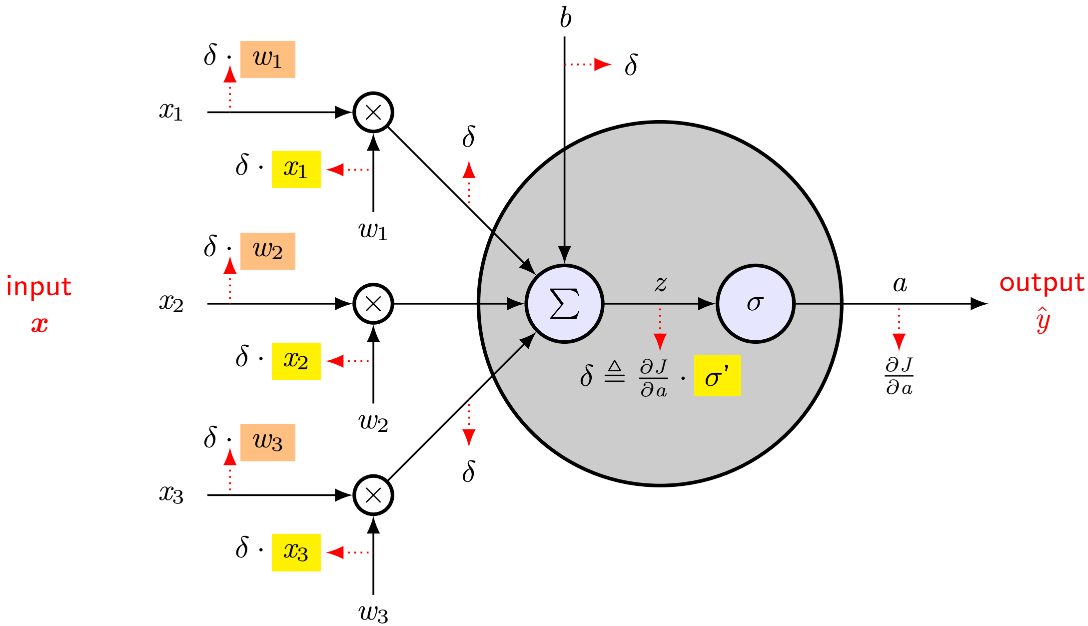
$$\begin{aligned} \frac{\partial J}{\partial z} &= \frac{\partial J}{\partial a}\frac{\partial a}{\partial z} = \frac{\partial J}{\partial a} \cdot \sigma' \\ &= \left(-\frac{y}{a} + \frac{(1-y)}{1-a}\right) \cdot a(1-a) \\ &= a - y \triangleq \delta \qquad [\text{“delta error”}] \end{aligned}$$

- 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

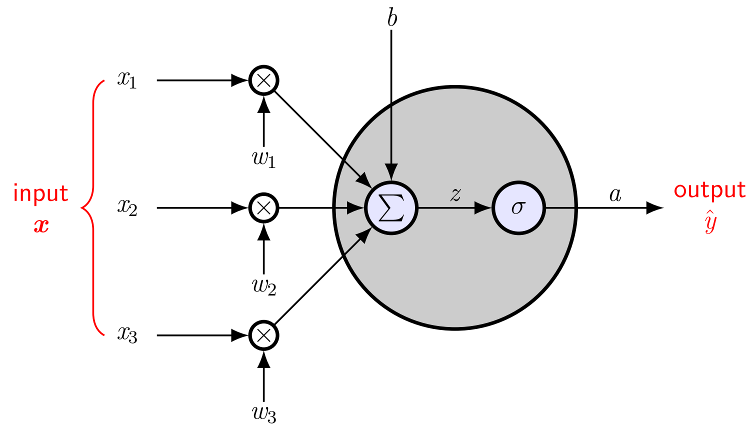
$$J(\boldsymbol{w}, b) = L(y, a) = -y \log a - (1-y)\log(1-a)$$

- compute gradient:

$$\frac{\partial J}{\partial w_1} = x_1 \cdot \frac{\partial J}{\partial z} = x_1(a-y) = x_1 \delta$$

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

$$\frac{\partial J}{\partial b} = 1 \quad \frac{\partial J}{\partial z} = 1 \ (a-y) = \delta$$



- apply update:

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

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

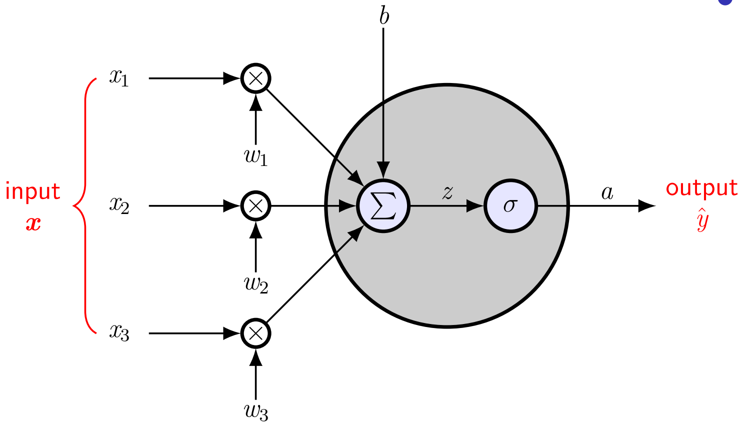
Minibatch SGD equations

- cost function

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

- signal, activation, delta error

$$z^{(i)} = \boldsymbol{w}^\top \boldsymbol{x}^{(i)} + b$$
$$a^{(i)} = \sigma(z^{(i)})$$
$$\delta^{(i)} = a^{(i)} - y^{(i)}$$



- 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:

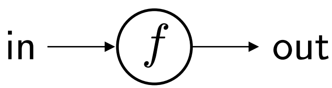
$$w_1 \leftarrow w_1 - \epsilon \frac{\partial J}{\partial w_1}$$
$$w_2 \leftarrow w_2 - \epsilon \frac{\partial J}{\partial w_2}$$
$$b \leftarrow b - \epsilon \frac{\partial J}{\partial b}$$

Outline

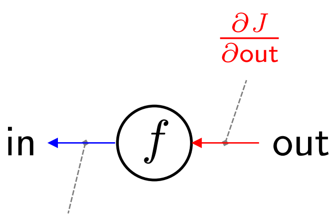
Single-gate backprop

- out $= f($in$)$

forward



backprop



$$\frac{\partial J}{\partial \text{in}} = \frac{\partial J}{\partial \text{out}} \cdot \frac{\partial \text{out}}{\partial \text{in}}$$

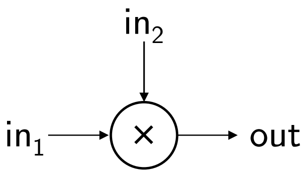
$$\frac{\partial J}{\partial \text{in}} = \underbrace{\frac{\partial J}{\partial \text{out}}}_{\substack{\text{output}\\ \text{gradient}}} \cdot \underbrace{\frac{\partial \text{out}}{\partial \text{in}}}_{\substack{\text{local}\\ \text{gradient}}}$$

$$= \frac{\partial J}{\partial \text{out}} \cdot f'(\text{in})$$

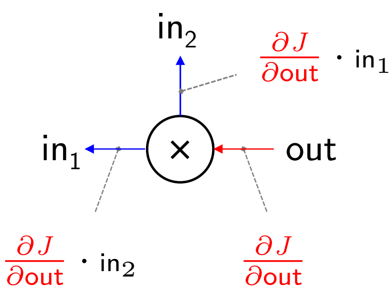
Multiplication

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

forward



backprop



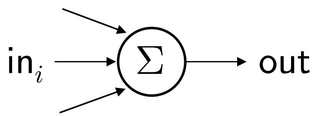
$$\frac{\partial J}{\partial \text{in}_1} = \frac{\partial J}{\partial \text{out}} \cdot \frac{\partial \text{out}}{\partial \text{in}_1} = \underbrace{\frac{\partial J}{\partial \text{out}}}_{\text{output gradient}} \cdot \underbrace{\text{in}_2}_{\text{local gradient}}$$

$$\frac{\partial J}{\partial \text{in}_2} = \frac{\partial J}{\partial \text{out}} \cdot \frac{\partial \text{out}}{\partial \text{in}_2} = \frac{\partial J}{\partial \text{out}} \cdot \text{in}_1$$

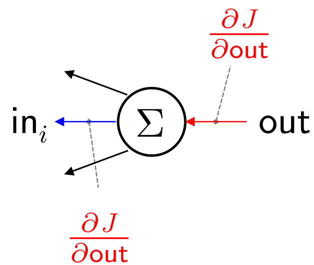
Summation

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

forward



backprop



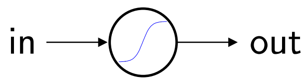
$$\frac{\partial J}{\partial \text{in}_i} = \frac{\partial J}{\partial \text{out}} \cdot \frac{\partial \text{out}}{\partial \text{in}_i} = \underbrace{\frac{\partial J}{\partial \text{out}}}_{\substack{\text{output}\\\text{gradient}}} \cdot \underbrace{1}_{\substack{\text{local}\\\text{gradient}}} = \frac{\partial J}{\partial \text{out}}$$

- sum (forward) $\Leftrightarrow$ fan out (backprop)

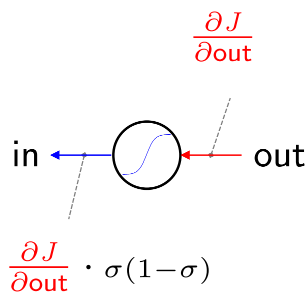
Sigmoid

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

forward



backprop

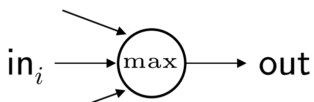


$$\begin{aligned}
\frac{\partial J}{\partial \text{in}} &= \frac{\partial J}{\partial \text{out}} \cdot \frac{\partial \text{out}}{\partial \text{in}} \\
&= \underbrace{\frac{\partial J}{\partial \text{out}}}_{\substack{\text{output} \\ \text{gradient}}} \cdot \underbrace{\sigma'(\text{in})}_{\substack{\text{local}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

Max

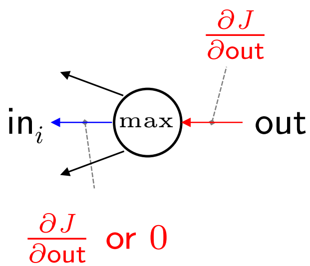
- out $= \max\limits_i \{\text{in}_i\}$

forward



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

backprop



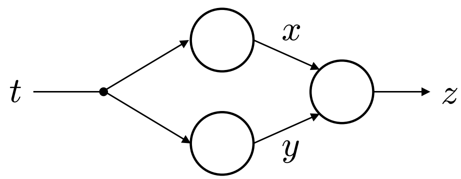
$$= \begin{cases} \frac{\partial J}{\partial \text{out}} & \text{if in}_i \text{ is max} \\ 0 & \text{otherwise} \end{cases}$$

- max (forward) $\Leftrightarrow$ mux (backprop)

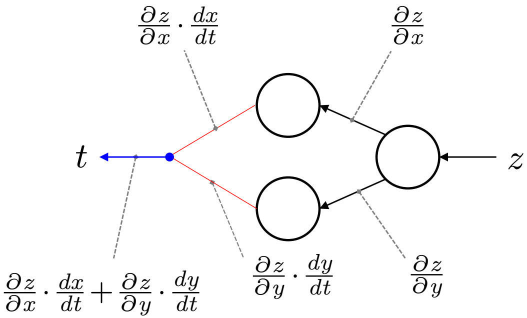
Backprop through fanout

- multivariable chain rule

forward



backprop



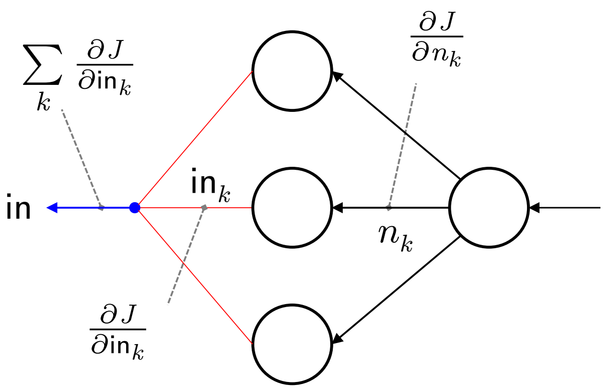
let

$$x = x(t), \;\; y = y(t)$$
$$z = f(x, y)$$

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

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

and

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

gives

$$\frac{\partial J}{\partial \text{in}} = \sum_k \frac{\partial J}{\partial n_k} \cdot \frac{\partial n_k}{\partial \text{in}} = \sum_k \frac{\partial J}{\partial \text{in}_k}$$

where

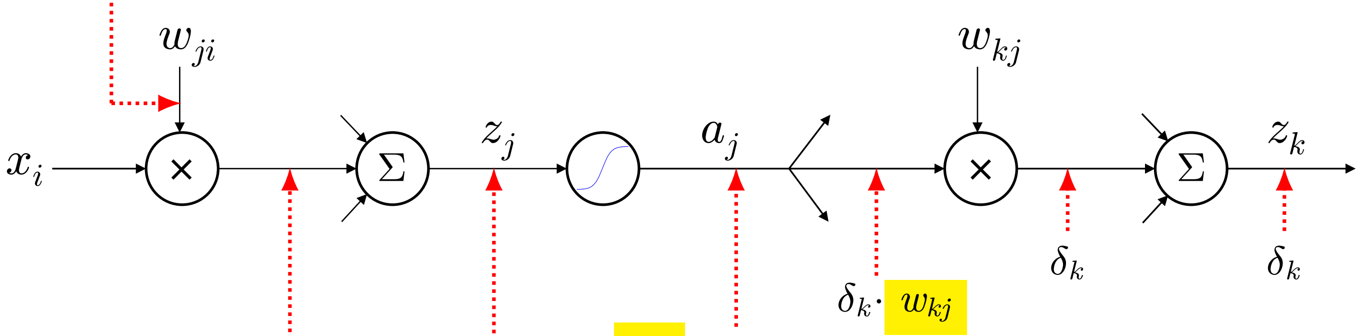
$$\text{in}_k \triangleq \text{ input to } n_k$$

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

Example

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

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



$$\sum_k \delta_k \cdot w_{kj}$$

$$\delta_j = \left(\sum_k \delta_k \cdot w_{kj}\right) \cdot \sigma'(z_j)$$

$$\left(\sum_k \delta_k \cdot w_{kj}\right) \cdot \sigma'(z_j)$$

Outline

Arranging minibatch

- two options to arrange m examples
  - in columns

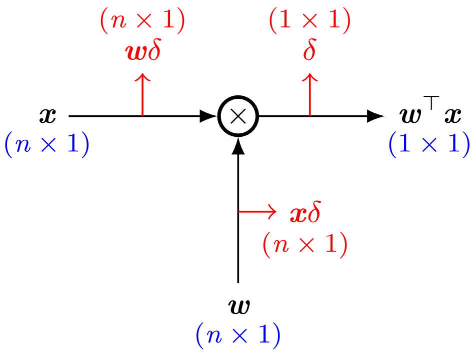
$$\boldsymbol{X}_{|||} = \begin{bmatrix} | & | & & | \\ \boldsymbol{x}^{(1)} & \boldsymbol{x}^{(2)} & \cdots & \boldsymbol{x}^{(m)} \\ | & | & & | \end{bmatrix}$$

  - in rows ("design matrix")

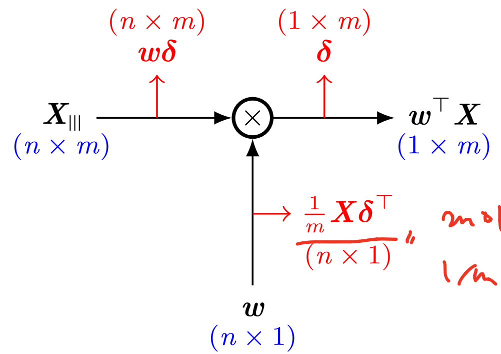
$$\boldsymbol{X}_{\equiv} = \begin{bmatrix} \text{——} & \boldsymbol{x}^{(1)\top} & \text{——} \\ \text{——} & \boldsymbol{x}^{(2)\top} & \text{——} \\ & \vdots & \\ \text{——} & \boldsymbol{x}^{(m)\top} & \text{——} \end{bmatrix}$$

Weighted sum

- 1 example (column):

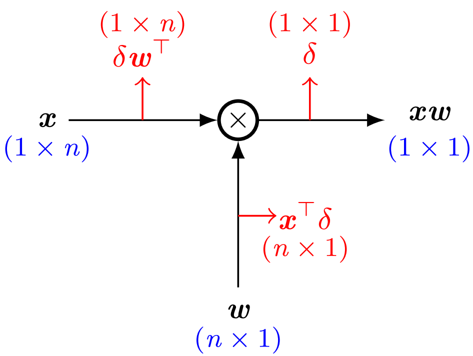


- size-m minibatch (column):

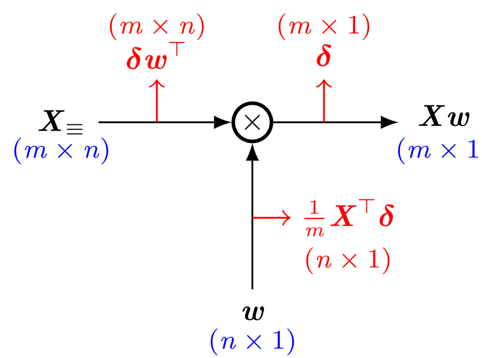


m이 사라지게 해서
1/m 붙음

- 1 example (row):

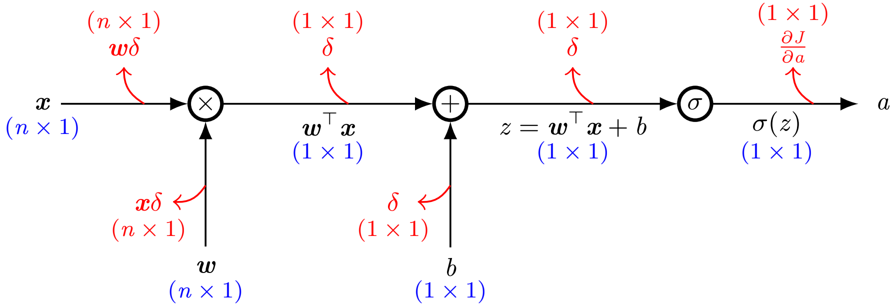


- size-m minibatch (row):

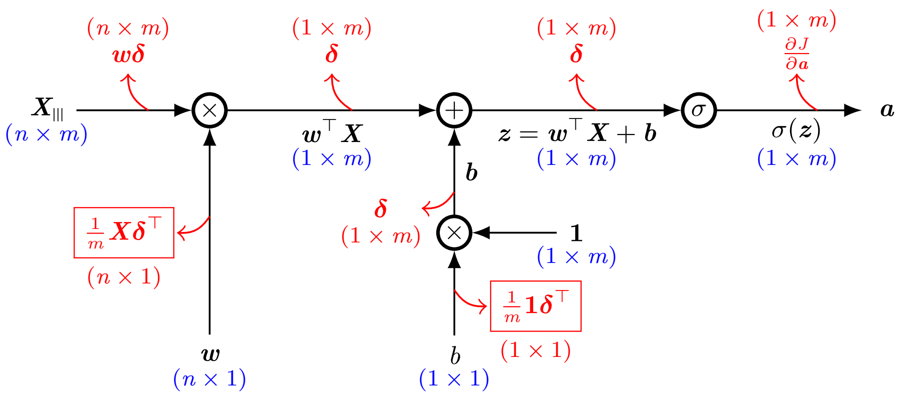


Forward/backward prop (column-wise)

- 1 example:



- size-m minibatch:

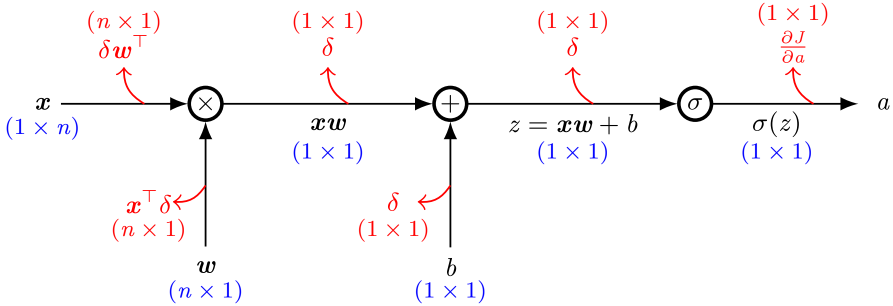


algorithm 2
logistic regression (col)

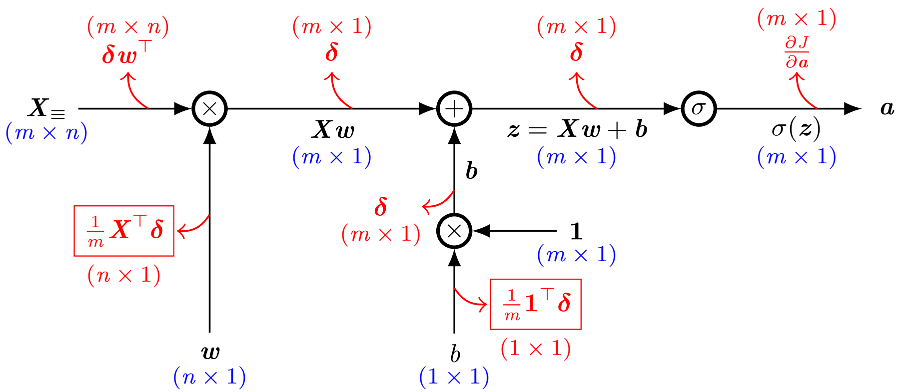
initialize $\boldsymbol{w}$, b
while necessary **do**
  $\boldsymbol{z} = \boldsymbol{w}^\top \boldsymbol{X} + \boldsymbol{b}$
  $\boldsymbol{a} = \sigma(\boldsymbol{z})$
  $\frac{\partial J}{\partial \boldsymbol{z}} \triangleq \boldsymbol{\delta} = \boldsymbol{a} - \boldsymbol{y}$
  $\frac{\partial J}{\partial \boldsymbol{w}} = \frac{1}{m}\boldsymbol{X}\boldsymbol{\delta}^\top$
  $\frac{\partial J}{\partial b} = \frac{1}{m}\mathbf{1}\boldsymbol{\delta}^\top$
  $\boldsymbol{w} \leftarrow \boldsymbol{w} - \epsilon \frac{\partial J}{\partial \boldsymbol{w}}$
  $b \leftarrow b - \epsilon \frac{\partial J}{\partial b}$
end while
return $\boldsymbol{w}$, b

Forward/backward prop (row-wise)

- 1 example:



- size-m minibatch:



algorithm 3
logistic regression (row)

initialize w, b
while necessary **do**
$\quad z = Xw + b$
$\quad a = \sigma(z)$
$\quad \frac{\partial J}{\partial z} \triangleq \delta = a - y$
$\quad \frac{\partial J}{\partial w} =$
$\quad \frac{\partial J}{\partial b} =$
$\quad w \leftarrow w - \epsilon \frac{\partial J}{\partial w}$
$\quad b \leftarrow b - \epsilon \frac{\partial J}{\partial b}$
end while
return w, b

Outline

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: $\boldsymbol{\theta} = (\boldsymbol{w}, b)$
  - 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 $J(\boldsymbol{\theta})$: average loss from training examples
  - training: iterative optimization (such as gradient descent)
- gradient descent: a general, iterative optimization technique
  - update equation: $\boldsymbol{\theta} \leftarrow \boldsymbol{\theta} - \epsilon \nabla_{\boldsymbol{\theta}} J(\boldsymbol{\theta})$
  - unit of gradient estimation: batch (all), minibatch (m), stochastic (1)
  - neural nets: gradients are provided by back propagation (backprop)