Indian Institute of Information Technology, Allahabad

CVBL IIIT Allahabad

Neural Networks
Feed Forward Networks

By

Dr. Shiv Ram Dubey
Assistant Professor
Computer Vision And Biometrics Lab (CVBL)
Department Of Information Technology
Indian Institute Of Information Technology, Allahabad

Email: srdubey@iiita.ac.in Web: https://profile.iiita.ac.in/srdubey/

ABOUT IIIT ALLAHABAD (A RESEARCH LED INSTITUTE OF NATIONAL IMPORTANCE)

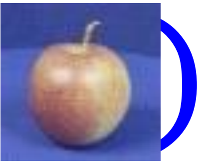
DISCLAIMER

The content (text, image, and graphics) used in this slide are adopted from many sources for Academic purposes. Broadly, the sources have been given due credit appropriately. However, there is a chance of missing out some original primary sources. The authors of this material do not claim any copyright of such material.

IMAGE CATEGORIZATION / CLASSIFICATION

THE STATISTICAL LEARNING FRAMEWORK

- Apply a prediction function to a feature representation of the image to get the desired output:

f() = “apple”

f() = “tomato”

f() = “cow”

THE STATISTICAL LEARNING FRAMEWORK

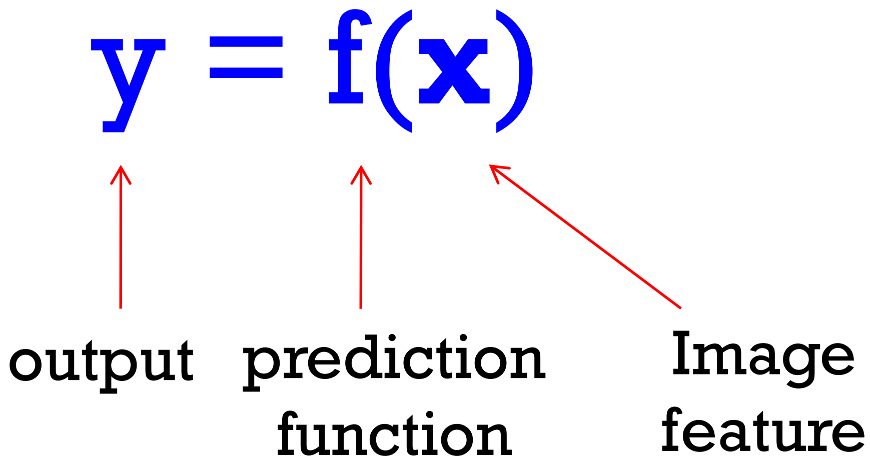
$$\mathbf{y} = f(\mathbf{x})$$

output

prediction function

Image feature

THE STATISTICAL LEARNING FRAMEWORK



$$\mathbf{y} = f(\mathbf{x})$$

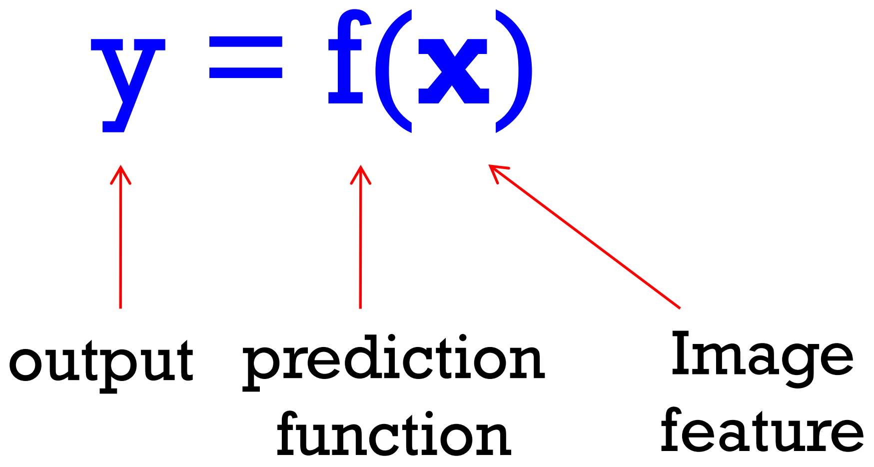
output

prediction function

Image feature

- **Training:** given a *training set* of labeled examples $\{(\mathbf{x}_1,\mathbf{y}_1), \ldots, (\mathbf{x}_N,\mathbf{y}_N)\}$, estimate the prediction function f by minimizing the prediction error on the training set

THE STATISTICAL LEARNING FRAMEWORK



$$y = f(\mathbf{x})$$

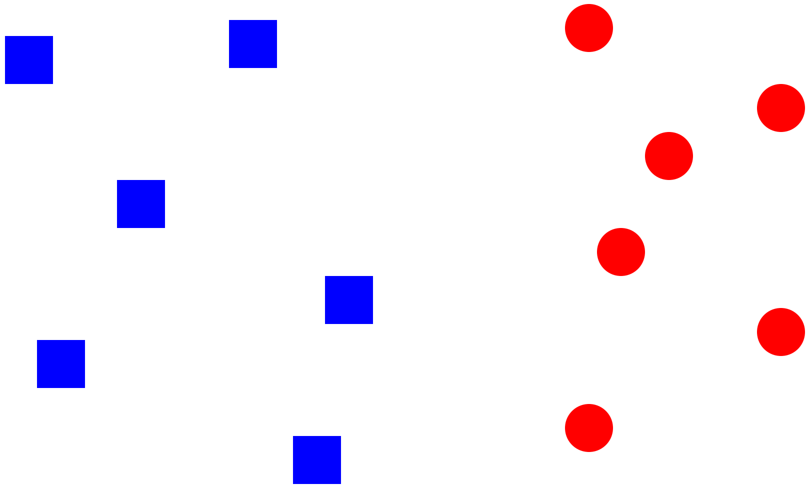
output

prediction function

Image feature

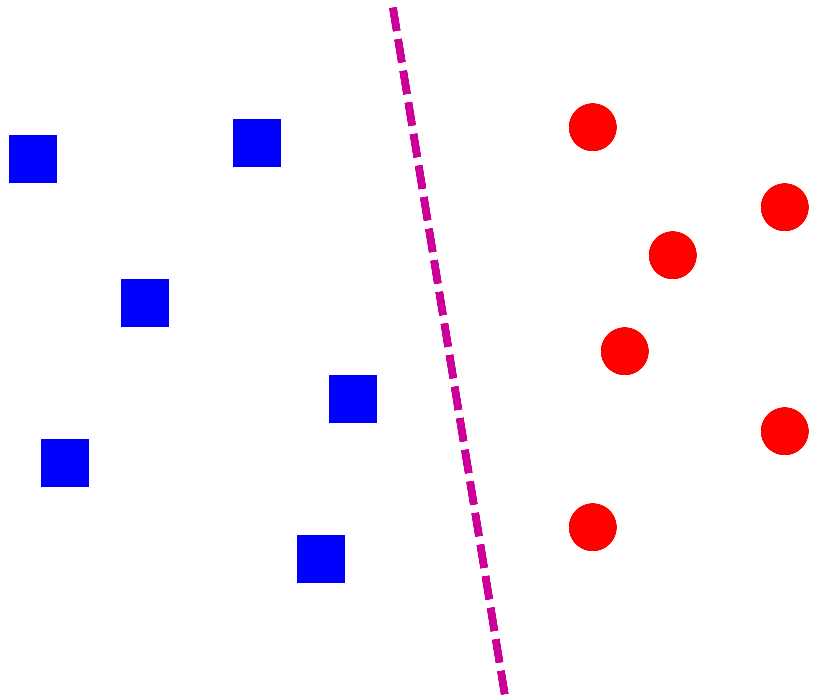
- **Training:** given a *training set* of labeled examples $\{(\mathbf{x}_1,y_1), \ldots, (\mathbf{x}_N,y_N)\}$, estimate the prediction function f by minimizing the prediction error on the training set
- **Testing:** apply f to a never before seen *test example* $\mathbf{x}$ and output the predicted value $y = f(\mathbf{x})$

LINEAR CLASSIFIERS – 2 CLASS PROBLEM



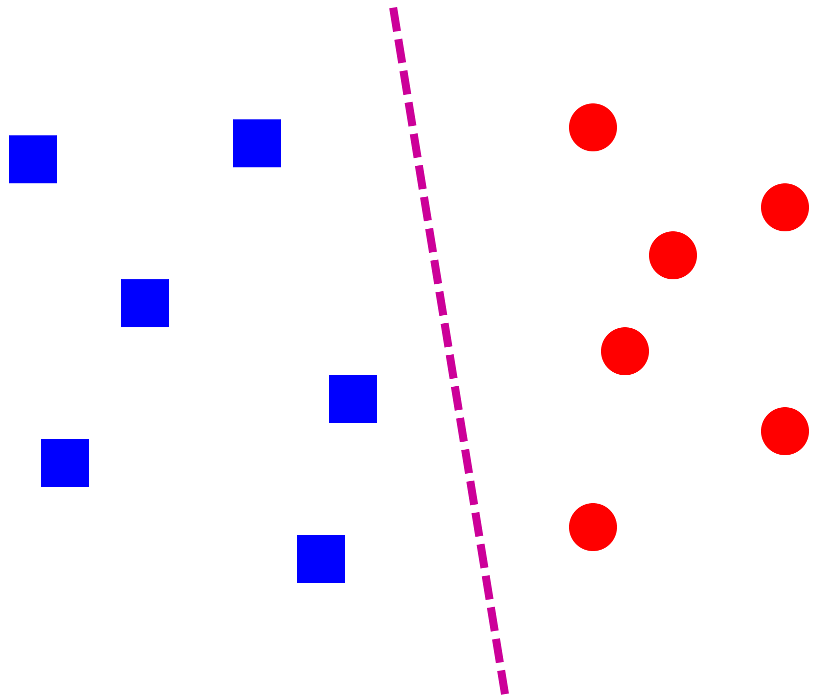
- Find a *linear function* to separate the classes:

LINEAR CLASSIFIERS – 2 CLASS PROBLEM



- Find a *linear function* to separate the classes:

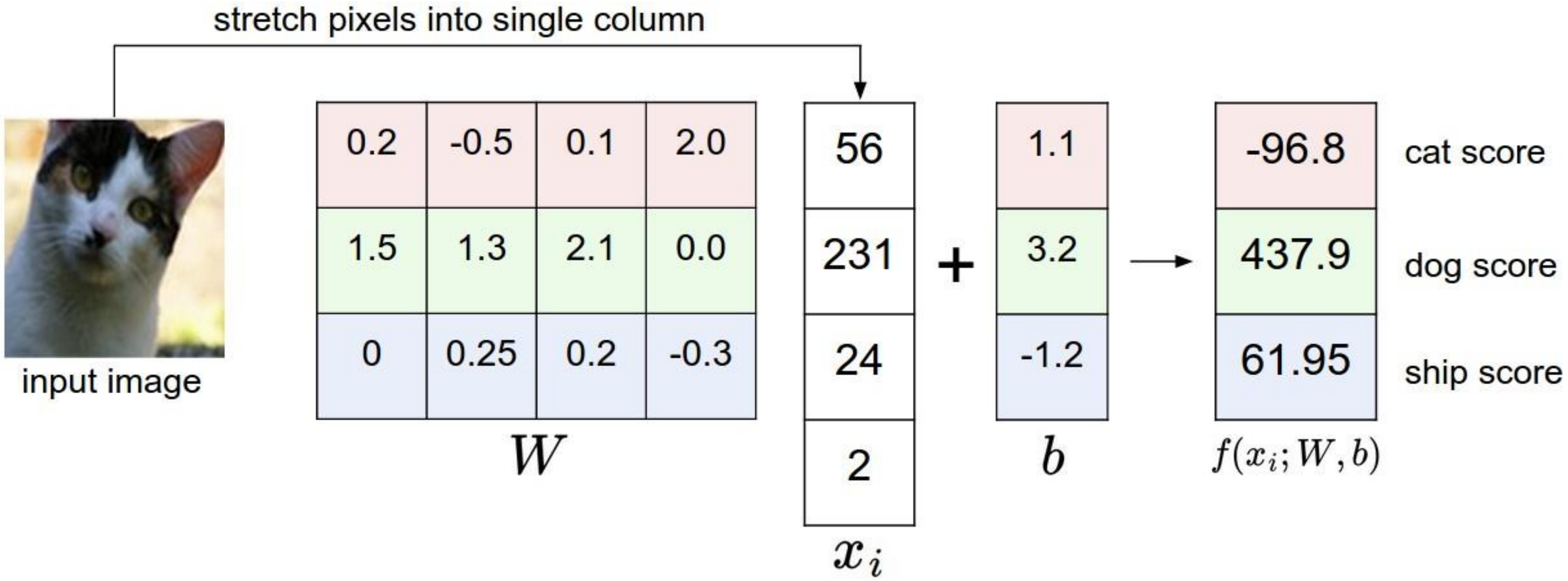
LINEAR CLASSIFIERS – 2 CLASS PROBLEM



- Find a *linear function* to separate the classes:

$$f(\mathbf{x}) = \text{sign}(\mathbf{w} \cdot \mathbf{x} + b)$$

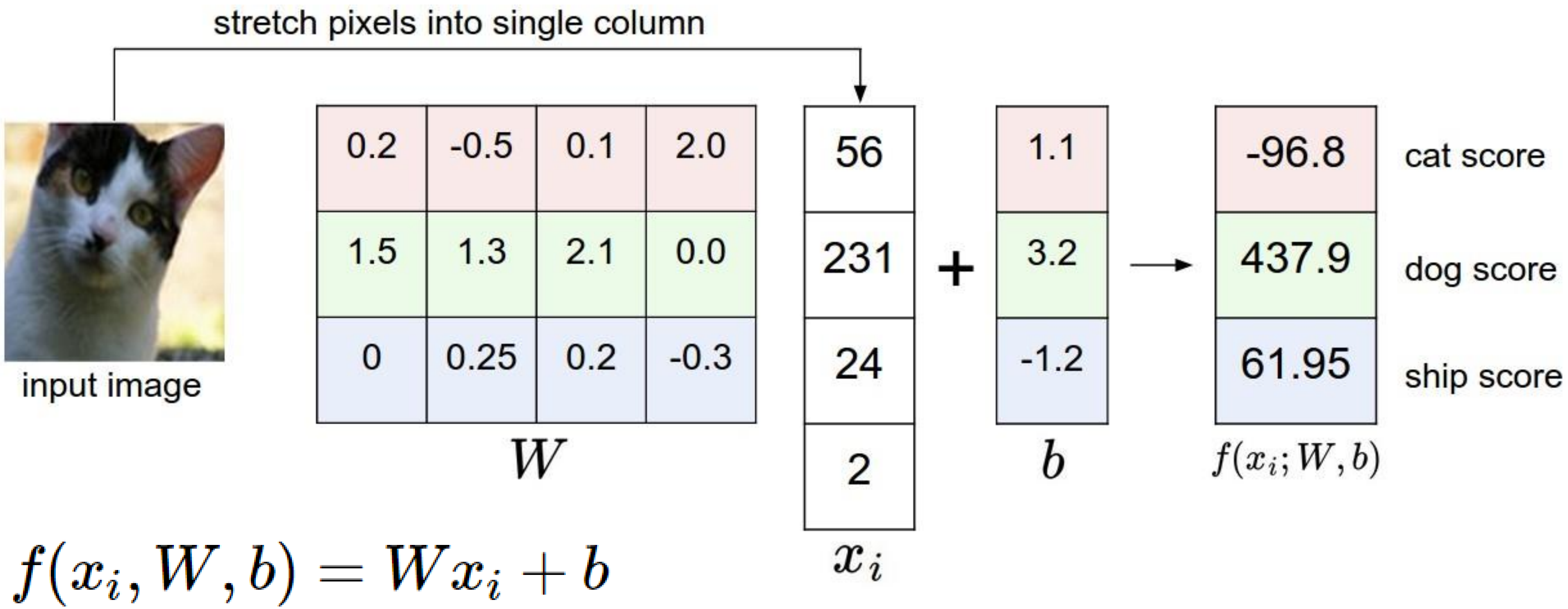
LINEAR CLASSIFIERS – MORE THAN 2 CLASS



$$\underbrace{\begin{bmatrix} 0.2 & -0.5 & 0.1 & 2.0 \\ 1.5 & 1.3 & 2.1 & 0.0 \\ 0 & 0.25 & 0.2 & -0.3 \end{bmatrix}}_{W} \underbrace{\begin{bmatrix} 56 \\ 231 \\ 24 \\ 2 \end{bmatrix}}_{x_i} + \underbrace{\begin{bmatrix} 1.1 \\ 3.2 \\ -1.2 \end{bmatrix}}_{b} \longrightarrow \underbrace{\begin{bmatrix} -96.8 \\ 437.9 \\ 61.95 \end{bmatrix}}_{f(x_i; W, b)}$$

Source: cs231n, http://cs231n.github.io/linear-classify/

LINEAR CLASSIFIERS – MORE THAN 2 CLASS



stretch pixels into single column

input image

$$\underset{W}{\begin{bmatrix}0.2 & -0.5 & 0.1 & 2.0\\ 1.5 & 1.3 & 2.1 & 0.0\\ 0 & 0.25 & 0.2 & -0.3\end{bmatrix}} \underset{x_i}{\begin{bmatrix}56\\ 231\\ 24\\ 2\end{bmatrix}} + \underset{b}{\begin{bmatrix}1.1\\ 3.2\\ -1.2\end{bmatrix}} \rightarrow \underset{f(x_i;W,b)}{\begin{bmatrix}-96.8\\ 437.9\\ 61.95\end{bmatrix}}$$

cat score

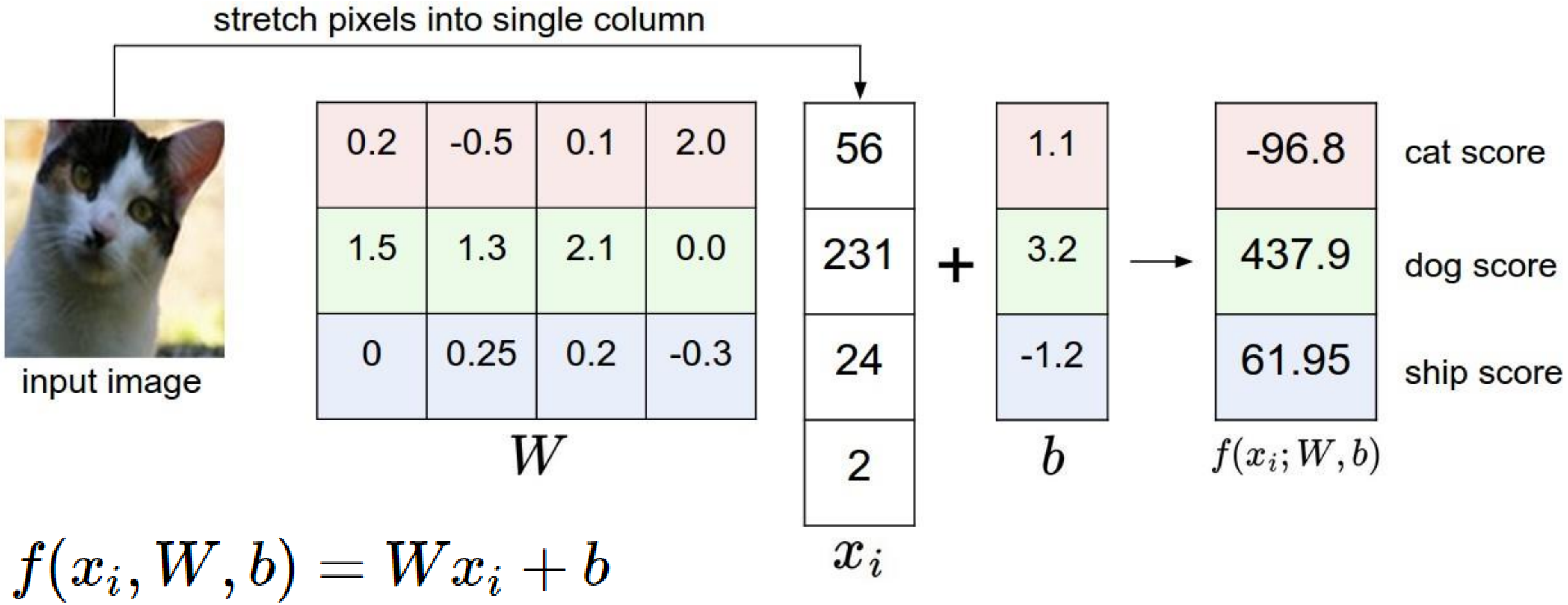
dog score

ship score

$$f(x_i, W, b) = Wx_i + b$$

Source: cs231n, http://cs231n.github.io/linear-classify/

LINEAR CLASSIFIERS – MORE THAN 2 CLASS



$$\underbrace{\begin{bmatrix} 0.2 & -0.5 & 0.1 & 2.0 \\ 1.5 & 1.3 & 2.1 & 0.0 \\ 0 & 0.25 & 0.2 & -0.3 \end{bmatrix}}_{W} \underbrace{\begin{bmatrix} 56 \\ 231 \\ 24 \\ 2 \end{bmatrix}}_{x_i} + \underbrace{\begin{bmatrix} 1.1 \\ 3.2 \\ -1.2 \end{bmatrix}}_{b} \longrightarrow \underbrace{\begin{bmatrix} -96.8 \\ 437.9 \\ 61.95 \end{bmatrix}}_{f(x_i; W, b)}$$

$$f(x_i, W, b) = W x_i + b$$

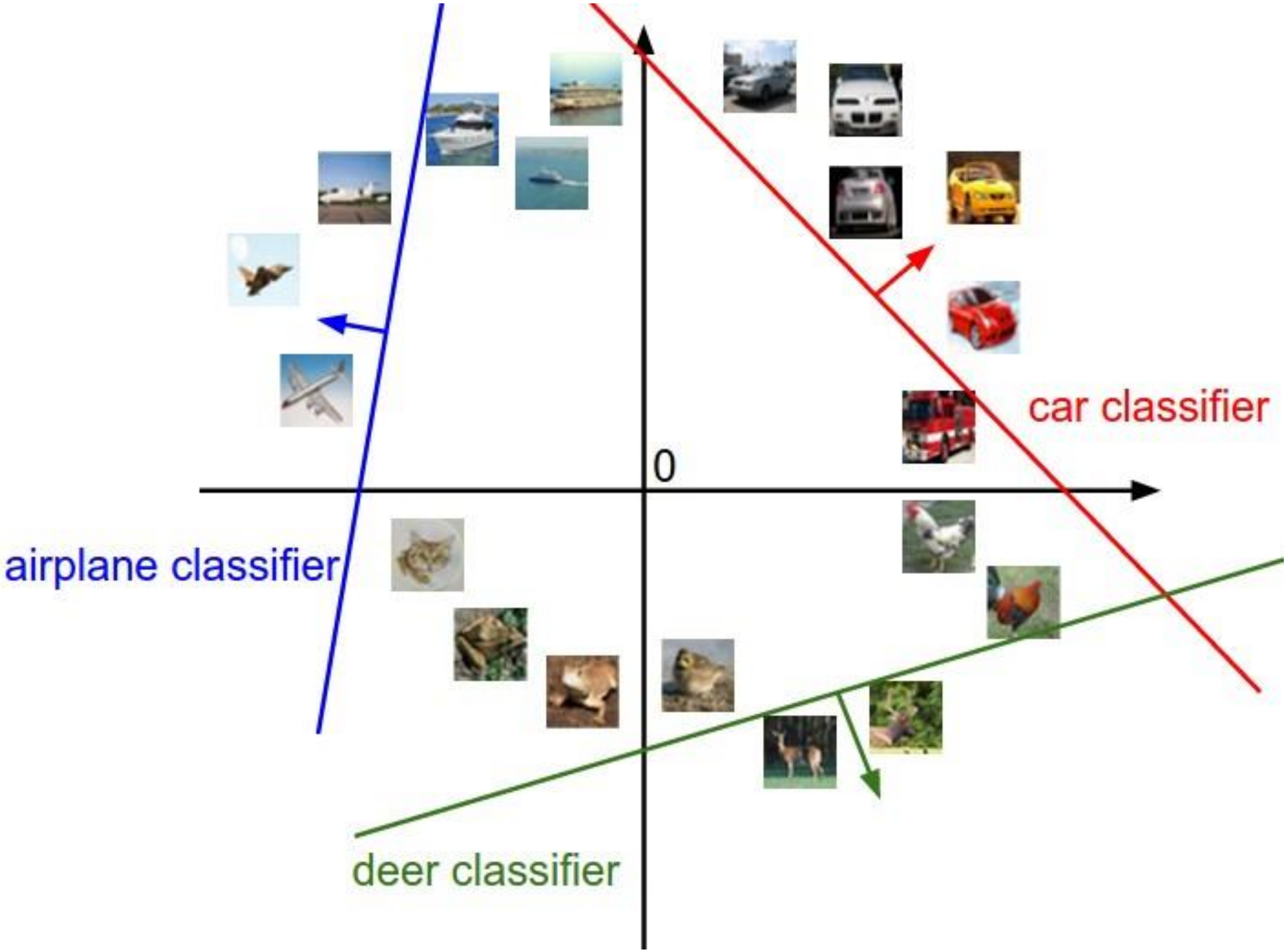
Image x_i has all of its pixels flattened out to a single column vector of shape [D x 1].

Matrix **W** (of size [K x D]), and vector **b** (of size [K x 1]) are the **parameters**.

K is the number of classes.

Source: cs231n, http://cs231n.github.io/linear-classify/

ANALOGY OF IMAGES AS HIGH-DIMENSIONAL POINTS



Source: cs231n, http://cs231n.github.io/linear-classify/

NEAREST NEIGHBOR VS. LINEAR CLASSIFIERS

- Linear pros:
  - Low-dimensional *parametric* representation
  - Very fast at test time

- Linear cons:
  - How to train the linear function?
  - What if data is not linearly separable?

SOFTMAX CLASSIFIER

- Interprets the class scores as the unnormalized log probabilities for each class and replace the *hinge loss* with a **cross-entropy loss** that has the form:

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

Source: cs231n, http://cs231n.github.io/linear-classify/

SOFTMAX CLASSIFIER

- Interprets the class scores as the unnormalized log probabilities for each class and replace the *hinge loss* with a **cross-entropy loss** that has the form:

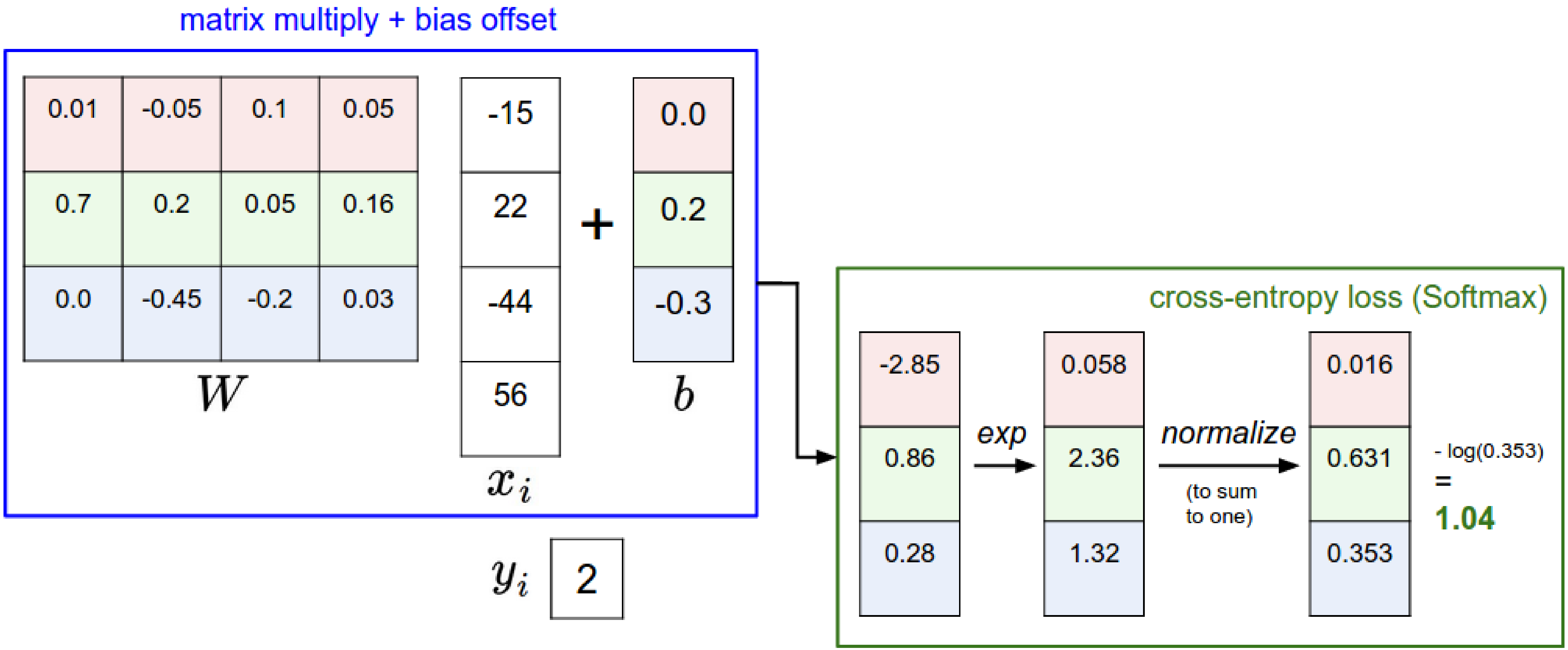
$$L_i = -log\left(\frac{e^{s_{y_i}}}{\sum_j e^{s_j}}\right) = -s_{y_i} + log\sum_j e^{s_j}$$

- Softmax Loss:

$$L = \underbrace{\frac{1}{N}\sum_i L_i}_{\text{data loss}} + \underbrace{\lambda R(W)}_{\text{regularization loss}}$$

Source: cs231n, http://cs231n.github.io/linear-classify/

HINGE VS CROSS-ENTROPY LOSS



$$\underbrace{\begin{bmatrix} 0.01 & -0.05 & 0.1 & 0.05 \\ 0.7 & 0.2 & 0.05 & 0.16 \\ 0.0 & -0.45 & -0.2 & 0.03 \end{bmatrix}}_{W} \underbrace{\begin{bmatrix} -15 \\ 22 \\ -44 \\ 56 \end{bmatrix}}_{x_i} + \underbrace{\begin{bmatrix} 0.0 \\ 0.2 \\ -0.3 \end{bmatrix}}_{b}$$

y_i $\boxed{2}$

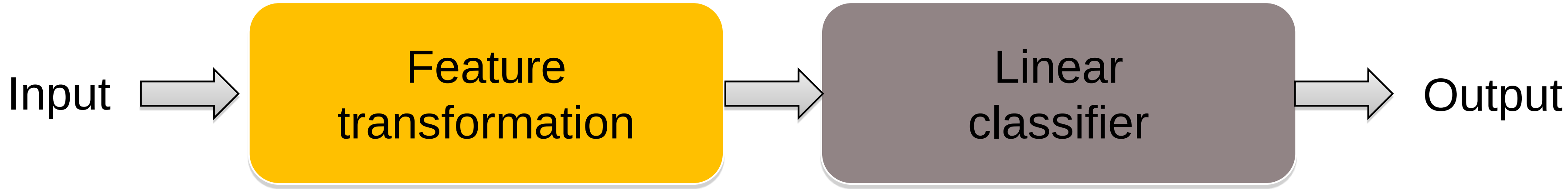
$$\begin{bmatrix} -2.85 \\ 0.86 \\ 0.28 \end{bmatrix} \xrightarrow{exp} \begin{bmatrix} 0.058 \\ 2.36 \\ 1.32 \end{bmatrix} \xrightarrow[\text{(to sum to one)}]{normalize} \begin{bmatrix} 0.016 \\ 0.631 \\ 0.353 \end{bmatrix}$$

$- \log(0.353) = 1.04$

Source: cs231n, http://cs231n.github.io/linear-classify/

FROM LINEAR TO NONLINEAR CLASSIFIERS

- To achieve good accuracy on challenging problems, we need to be able to train *nonlinear* models
- Two strategies for making nonlinear predictors out of linear ones:
  - **"Shallow" approach:** nonlinear feature transformation followed by linear classifier



Input

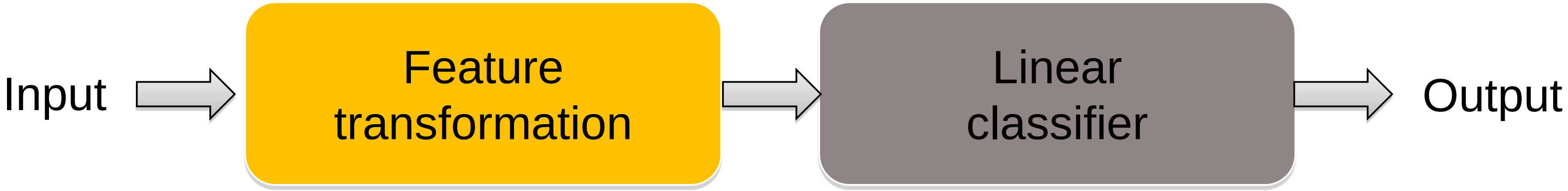
Feature transformation

Linear classifier

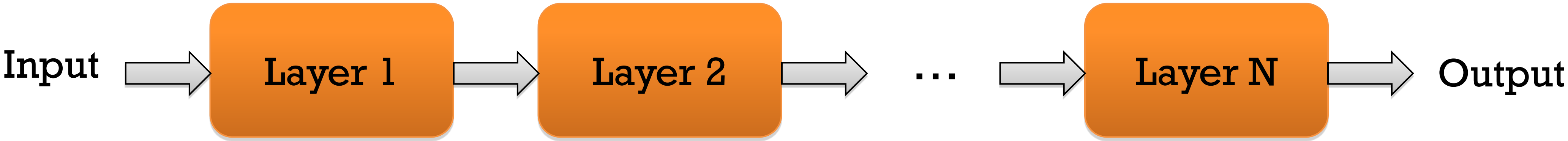
Output

FROM LINEAR TO NONLINEAR CLASSIFIERS

- To achieve good accuracy on challenging problems, we need to be able to train *nonlinear* models
- Two strategies for making nonlinear predictors out of linear ones:
  - **“Shallow” approach:** nonlinear feature transformation followed by linear classifier



  - **“Deep” approach:** stack multiple layers of linear predictors (interspersed with nonlinearities)



SHALLOW APPROACH

Input → Feature transformation → Linear classifier → Output

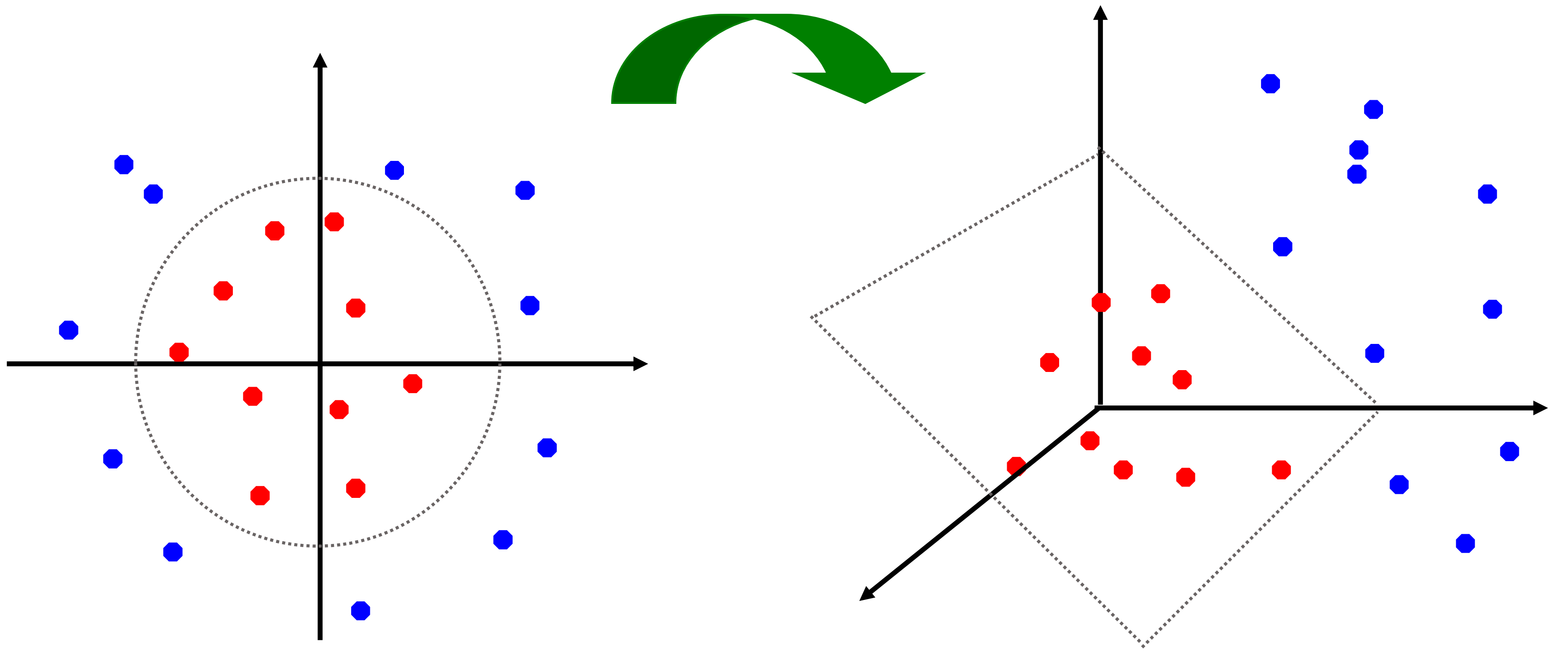


Image credit: Andrew Moore

MULTI-LAYER NEURAL NETWORKS

- “Deep” approach: stack multiple layers of linear predictors (perceptrons) interspersed with nonlinearities

Input

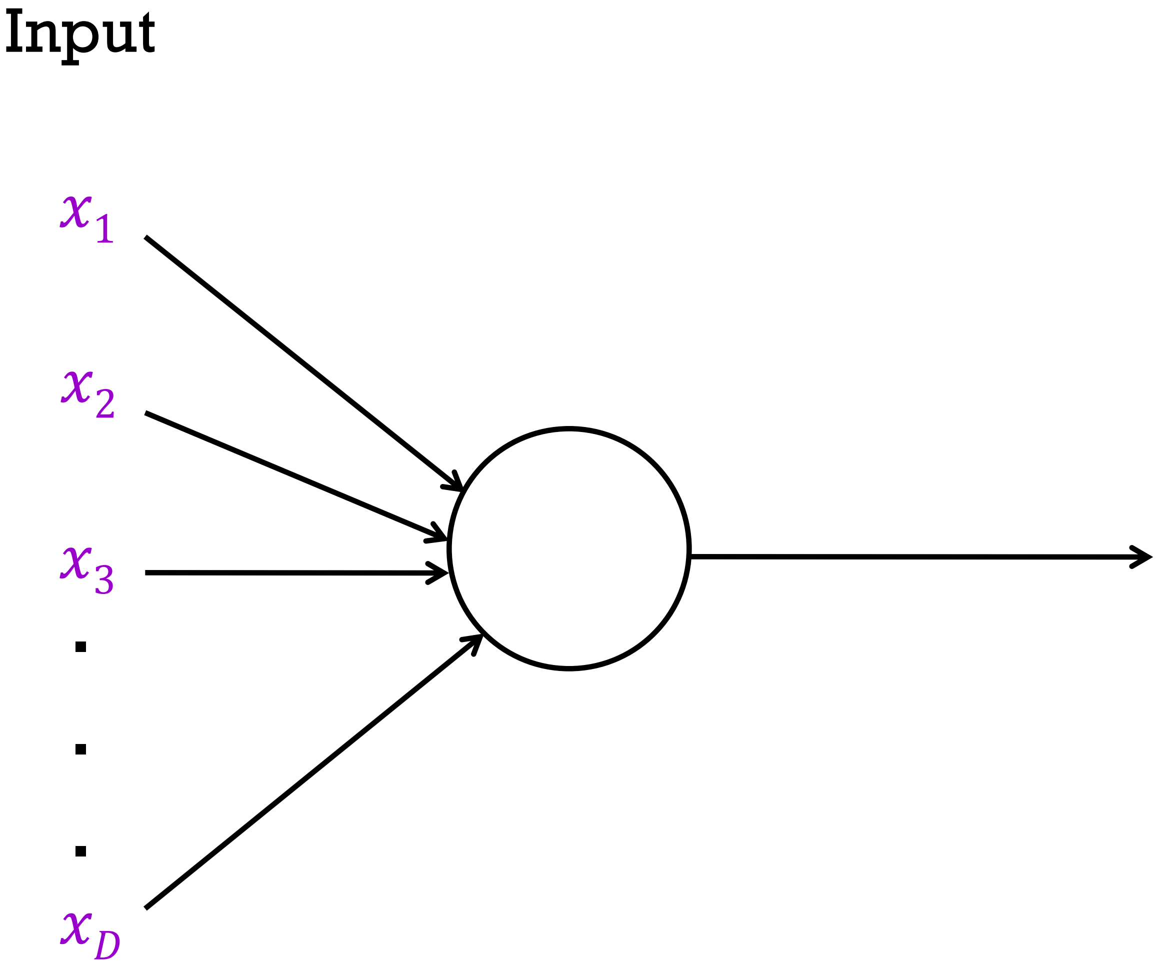
Layer 1

Layer 2

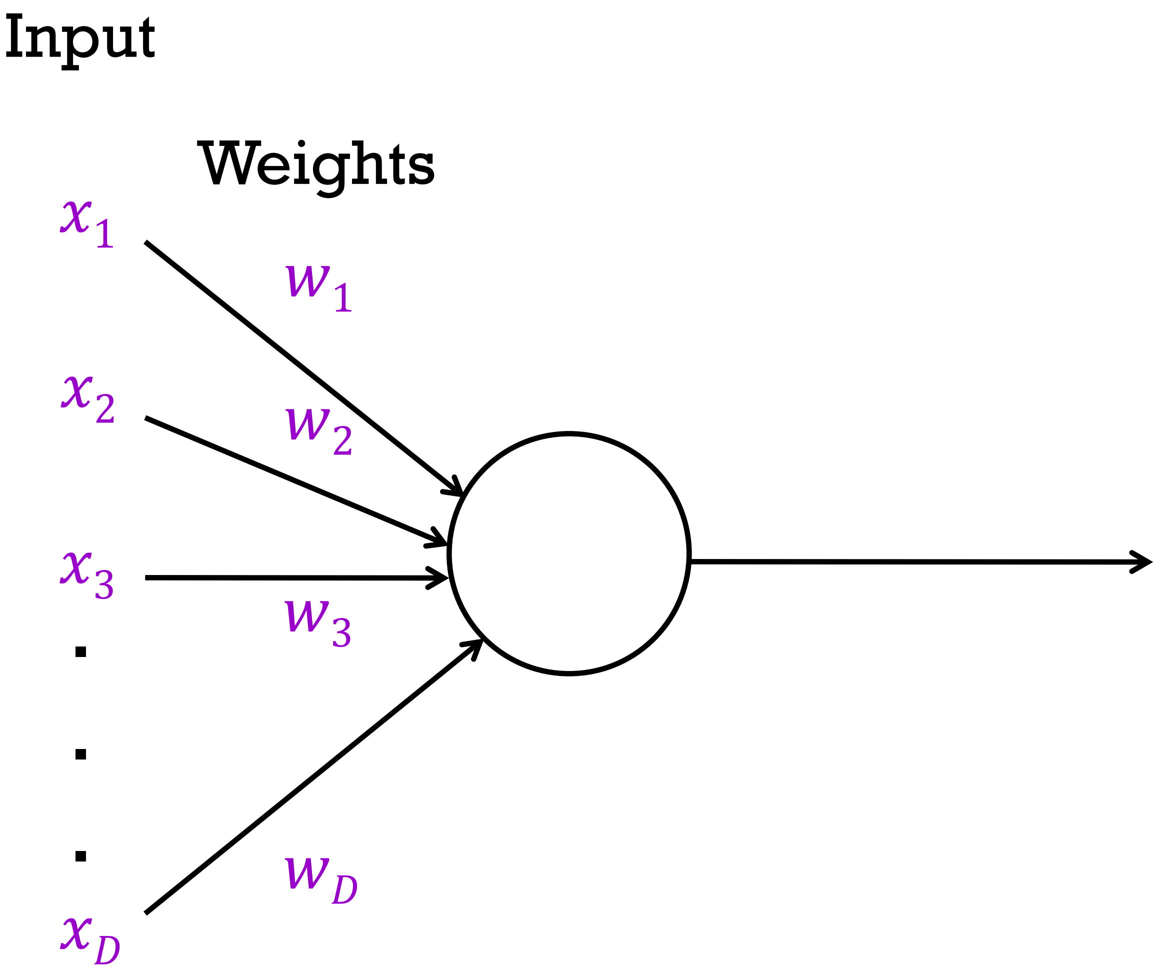
Layer 3

Output

PERCEPTRON



PERCEPTRON



Input

Weights

x_1 w_1

x_2 w_2

x_3 w_3

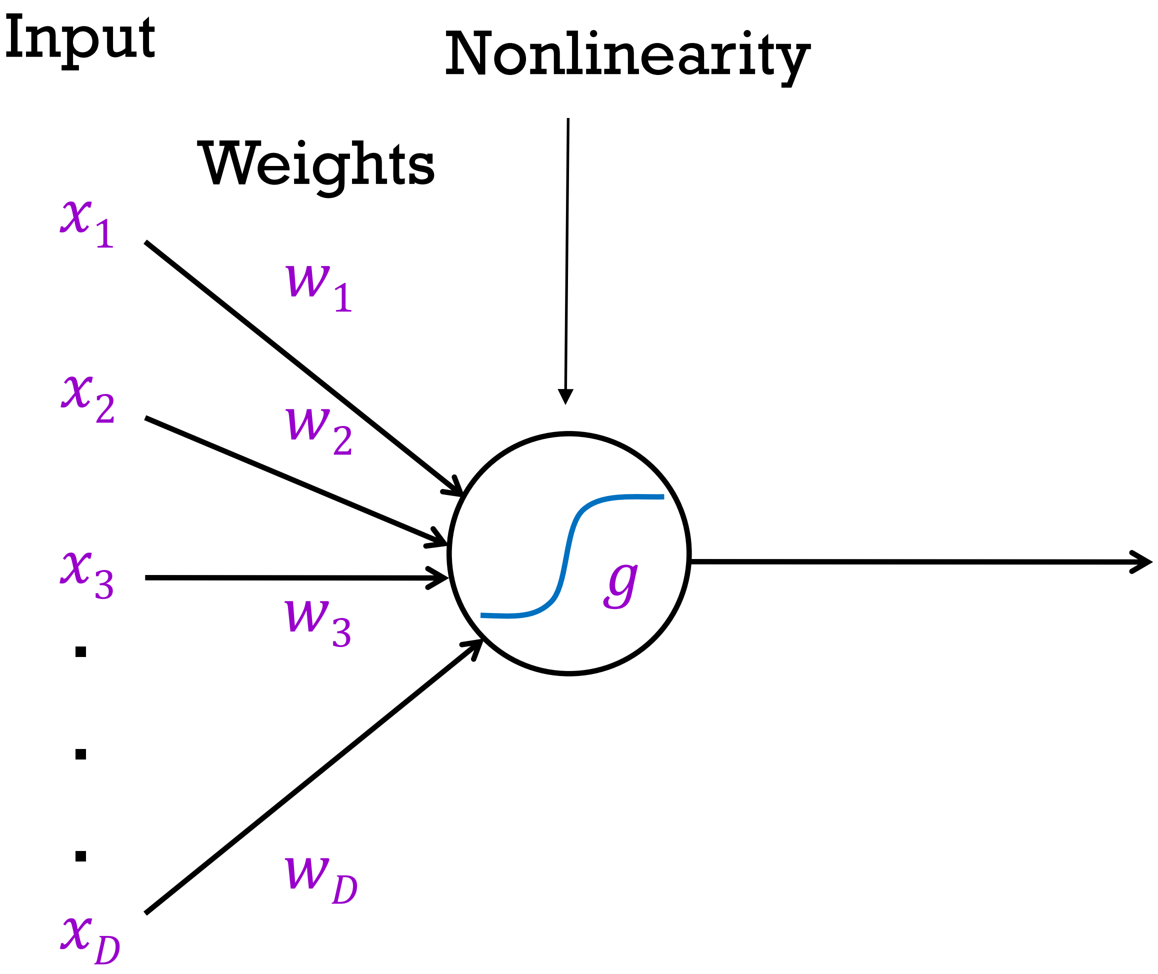
.

.

.

x_D w_D

PERCEPTRON

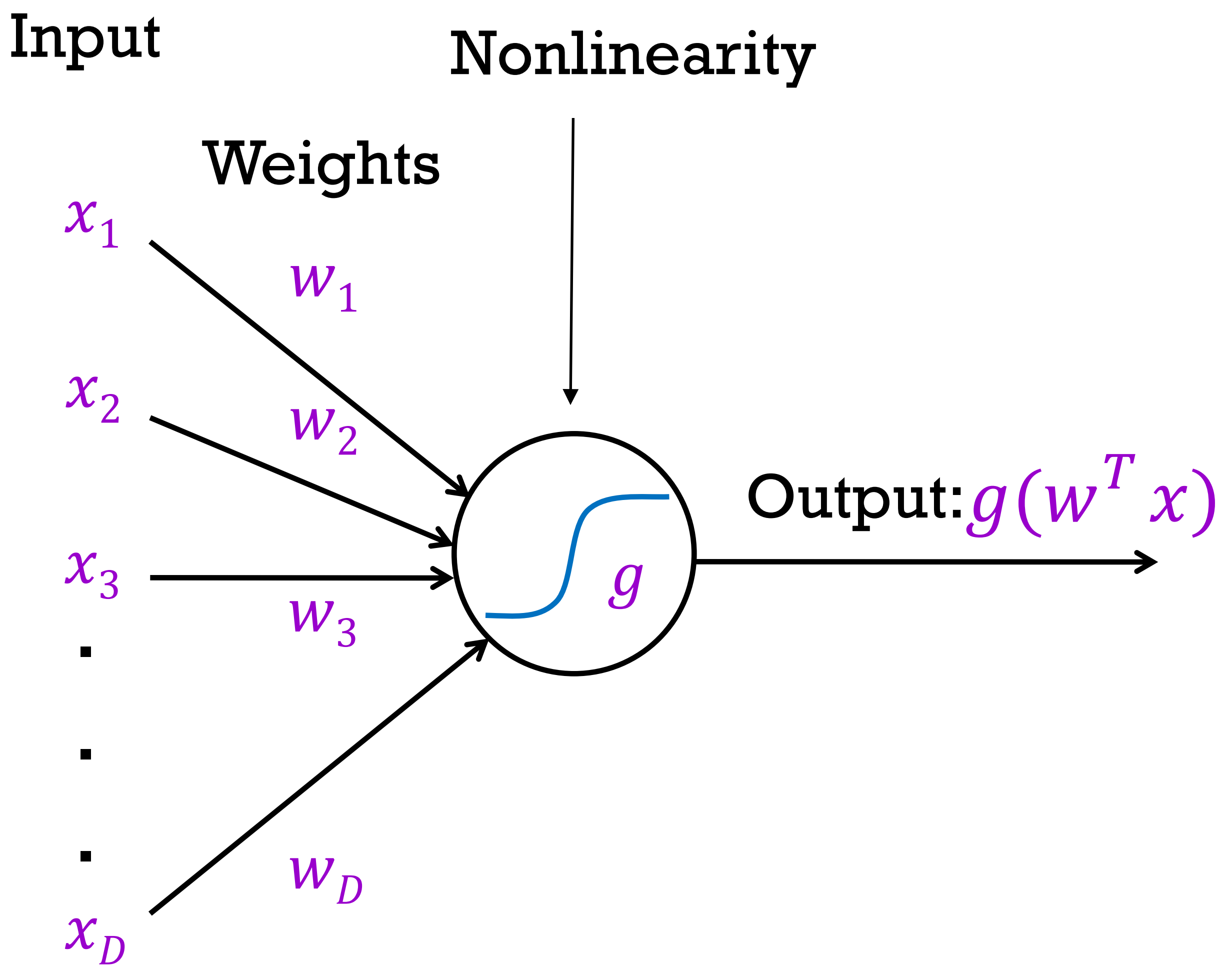


Input

Nonlinearity

Weights

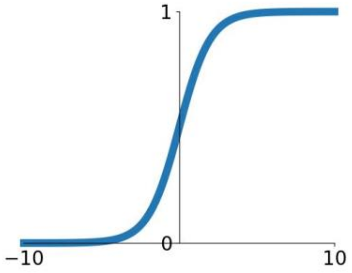
PERCEPTRON


Input
Nonlinearity
Weights
x_1
w_1
x_2
w_2
x_3
w_3
g
Output: $g(w^T x)$
x_D
w_D

COMMON NONLINEARITIES (OR *ACTIVATION FUNCTIONS*)

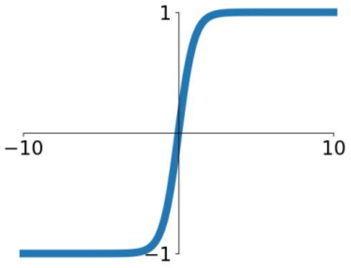
Sigmoid

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



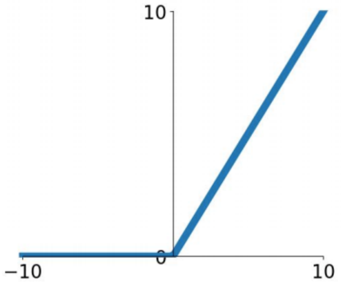
tanh

$$\tanh(x)$$



ReLU

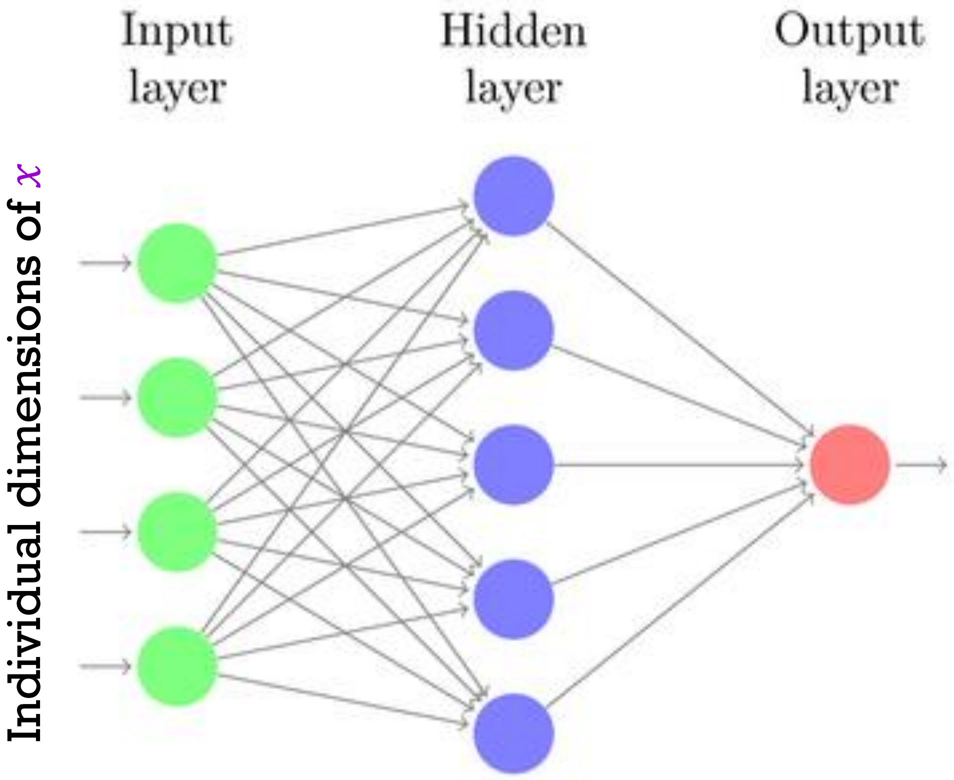
$$\max(0, x)$$



Source: Stanford 231n

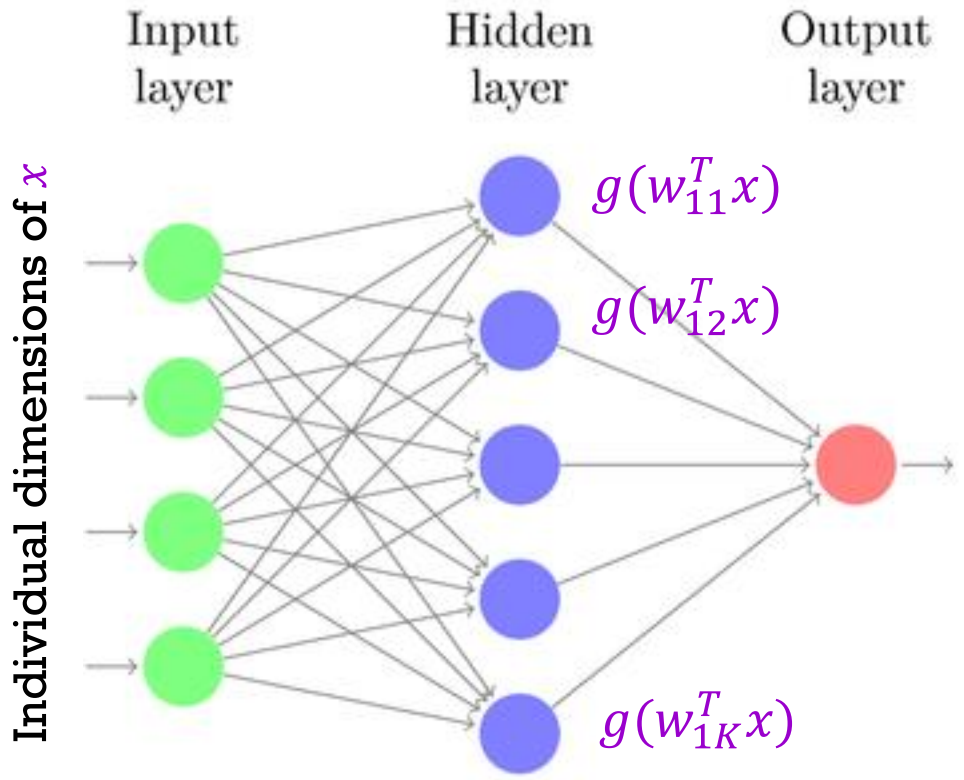
TWO-LAYER NEURAL NETWORK

- Introduce a *hidden layer* of perceptrons computing linear combinations of inputs followed by nonlinearities



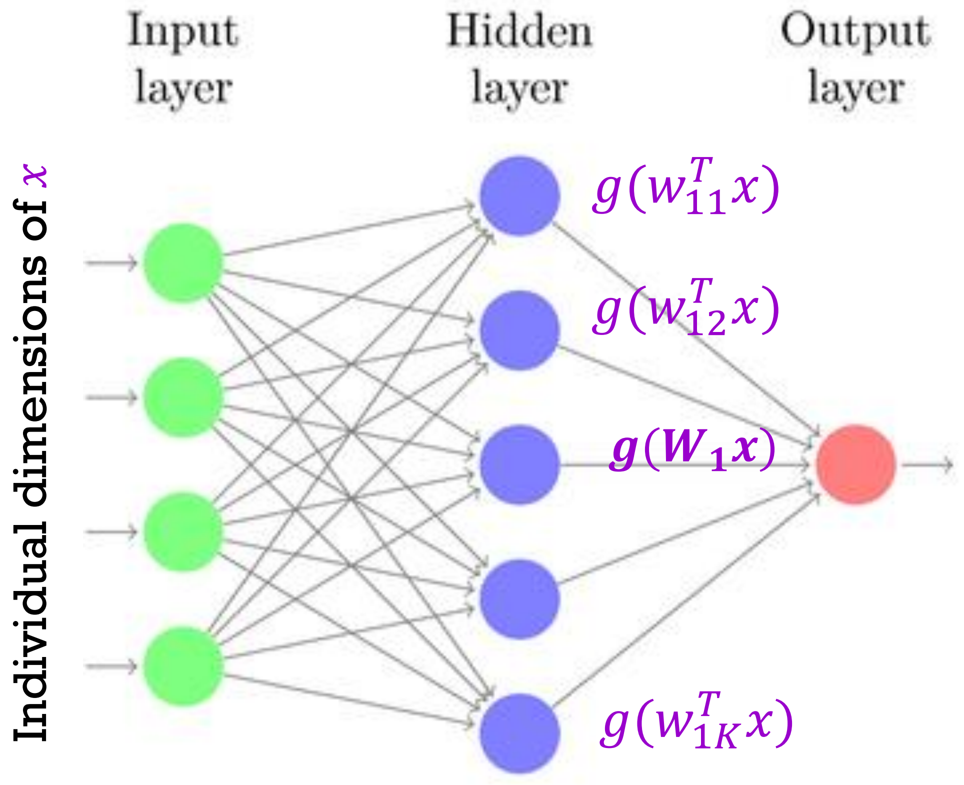
TWO-LAYER NEURAL NETWORK

- Introduce a *hidden layer* of perceptrons computing linear combinations of inputs followed by nonlinearities



TWO-LAYER NEURAL NETWORK

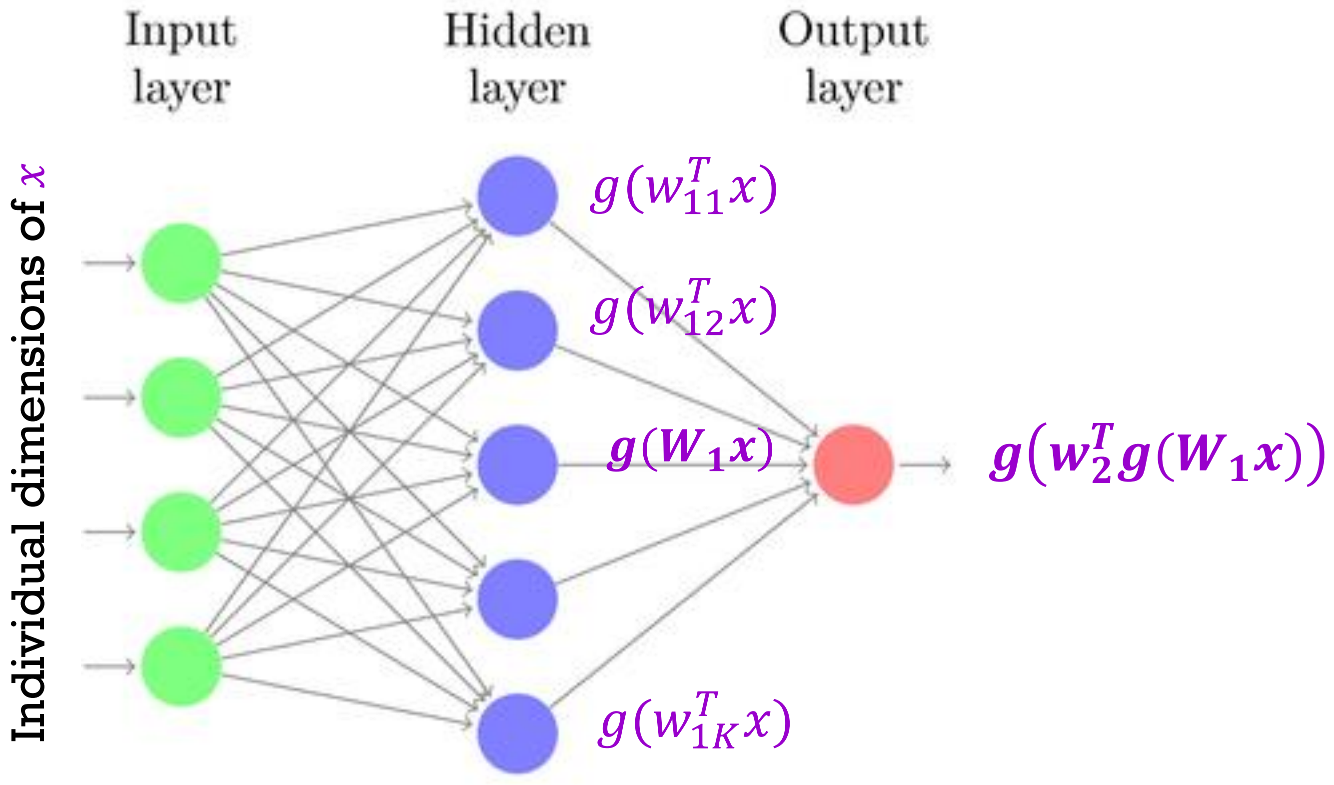
- Introduce a *hidden layer* of perceptrons computing linear combinations of inputs followed by nonlinearities



$\boldsymbol{W_1}$ – matrix with rows w_{1j}^T

TWO-LAYER NEURAL NETWORK

- Introduce a *hidden layer* of perceptrons computing linear combinations of inputs followed by nonlinearities



$\boldsymbol{W_1}$ – matrix with rows w_{1j}^T

TWO-LAYER NEURAL NETWORK

- Introduce a *hidden layer* of perceptrons computing linear combinations of inputs followed by nonlinearities
- The bigger the hidden layer, the more expressive the model

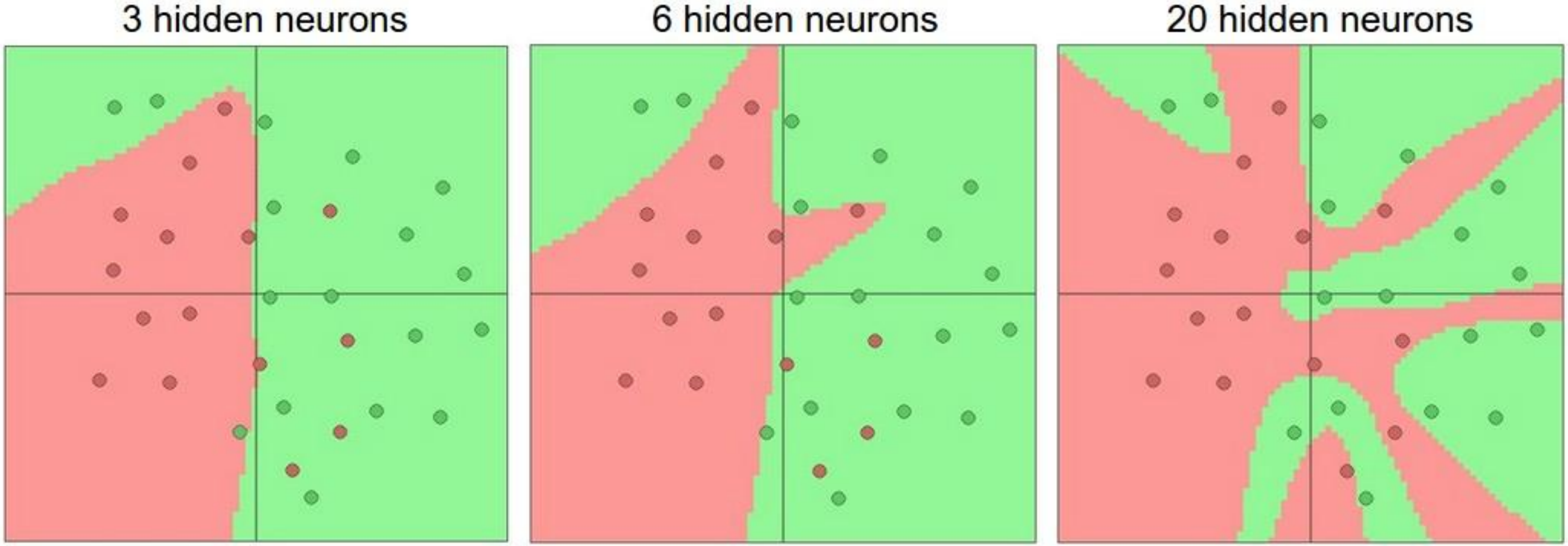
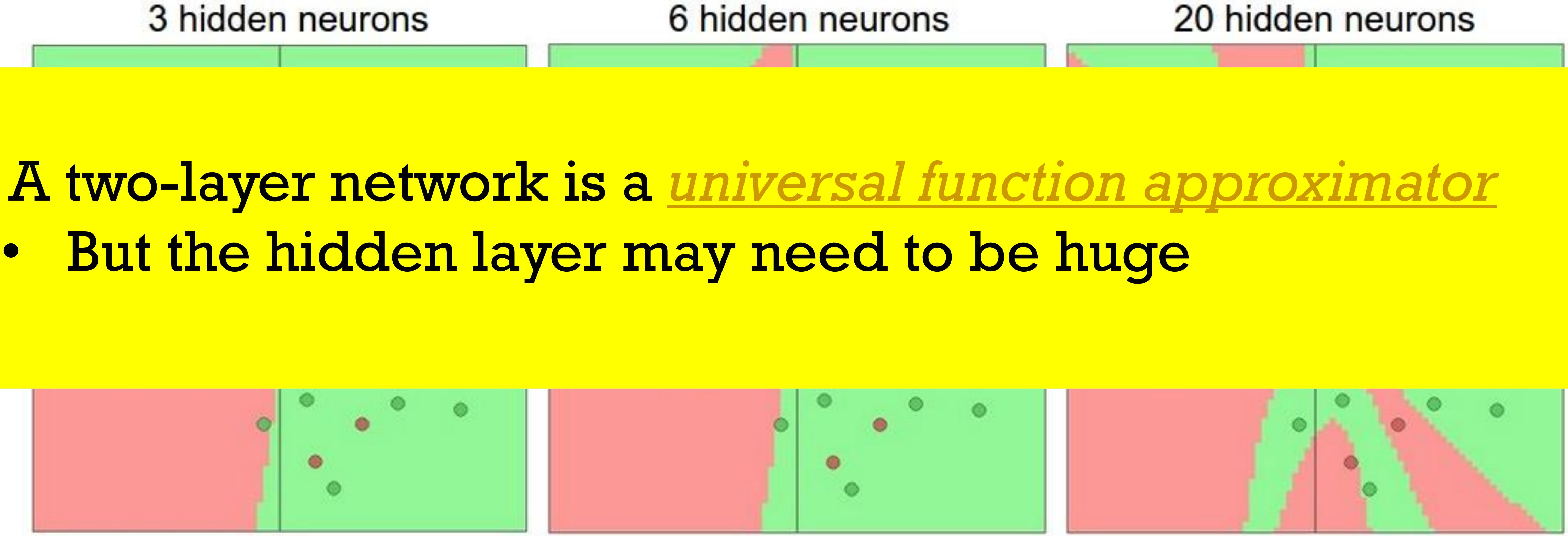


Figure source

TWO-LAYER NEURAL NETWORK

- Introduce a *hidden layer* of perceptrons computing linear combinations of inputs followed by nonlinearities
- The bigger the hidden layer, the more expressive the model



3 hidden neurons

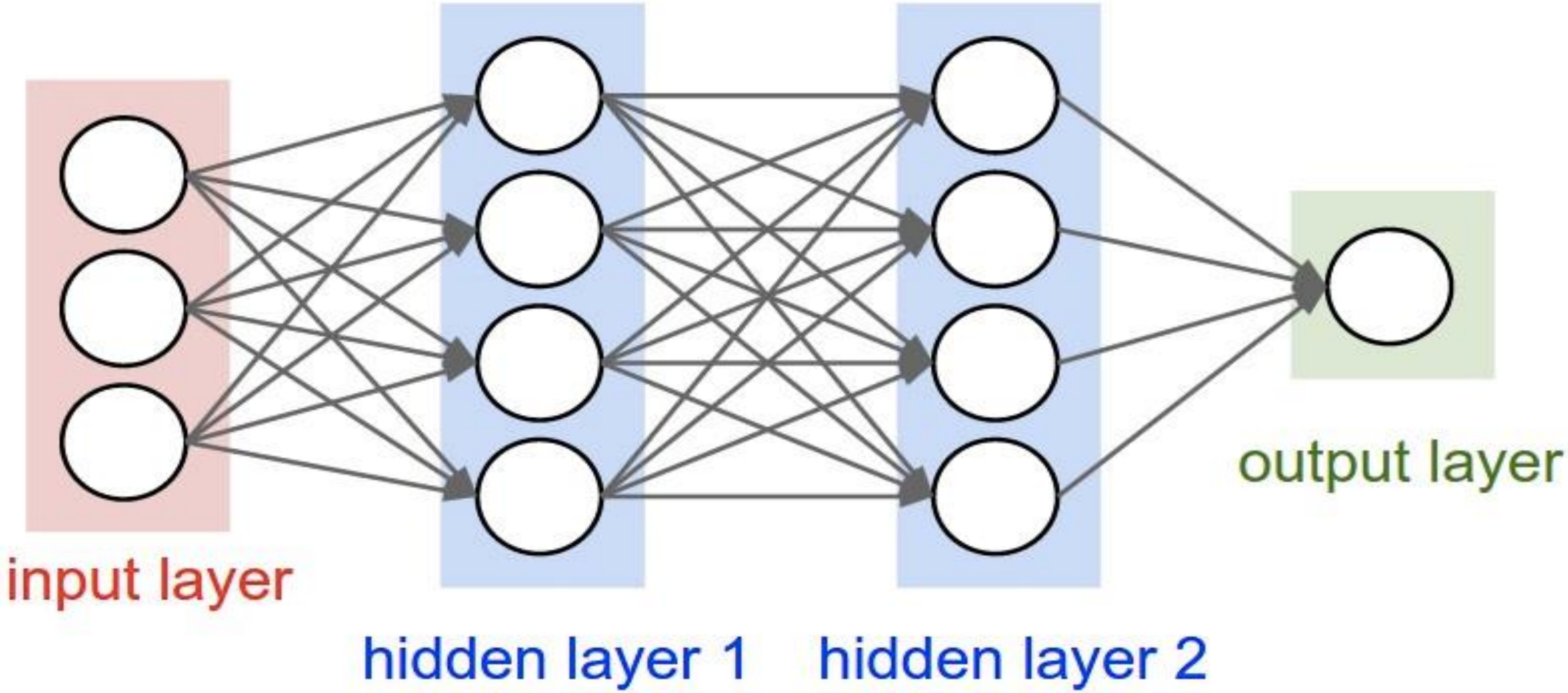
6 hidden neurons

20 hidden neurons

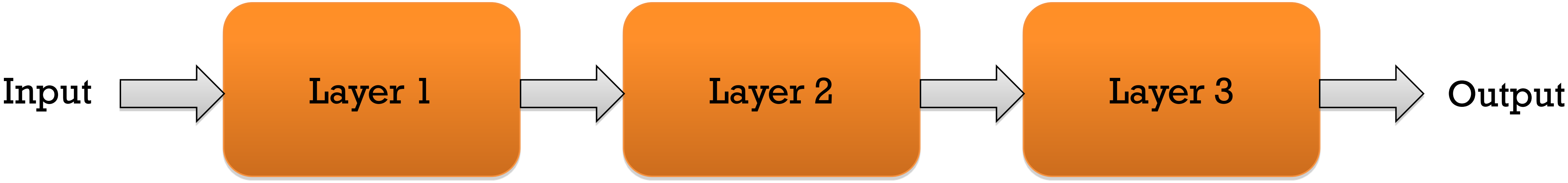
- A two-layer network is a *universal function approximator*
  - But the hidden layer may need to be huge

Figure source

BEYOND TWO LAYERS



“DEEP” PIPELINE


Input

Layer 1

Layer 2

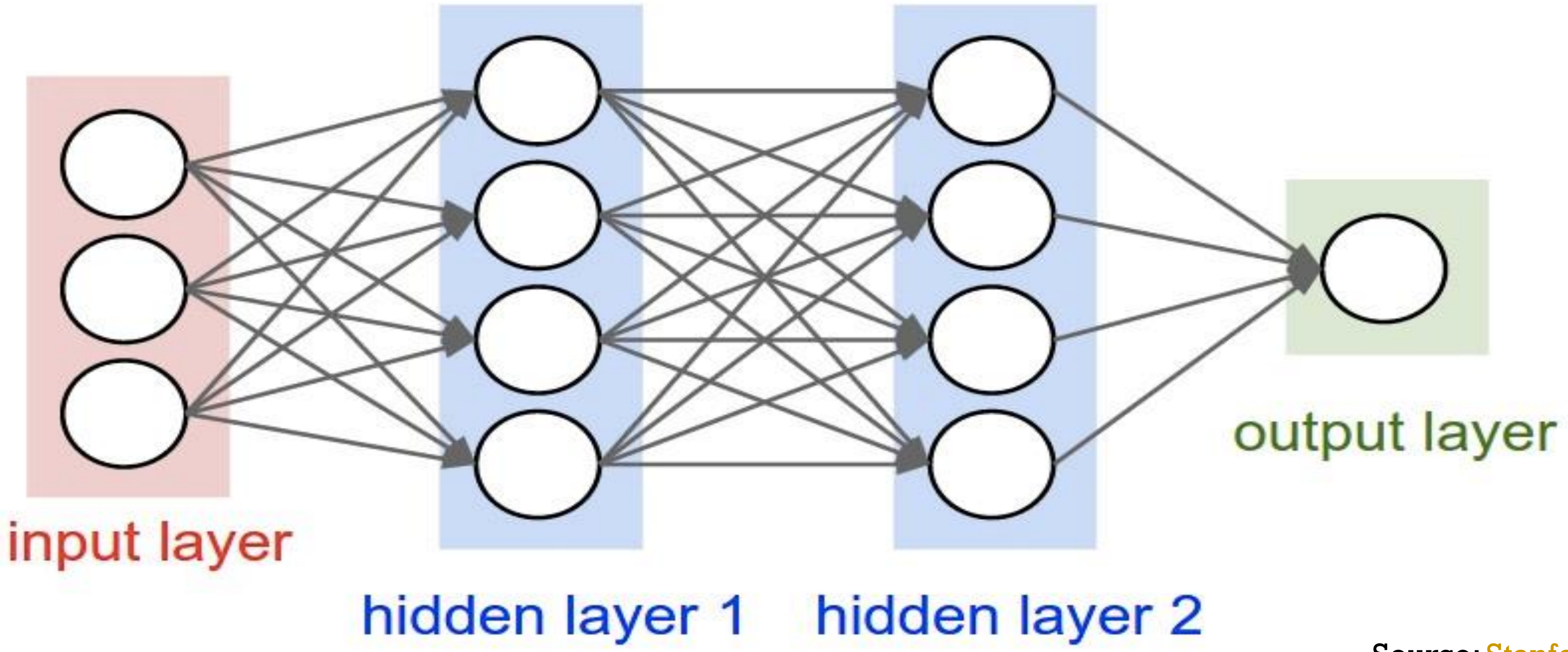
Layer 3

Output

- Learn a *feature hierarchy*
- Each layer extracts features from the output of previous layer
- All layers are trained jointly

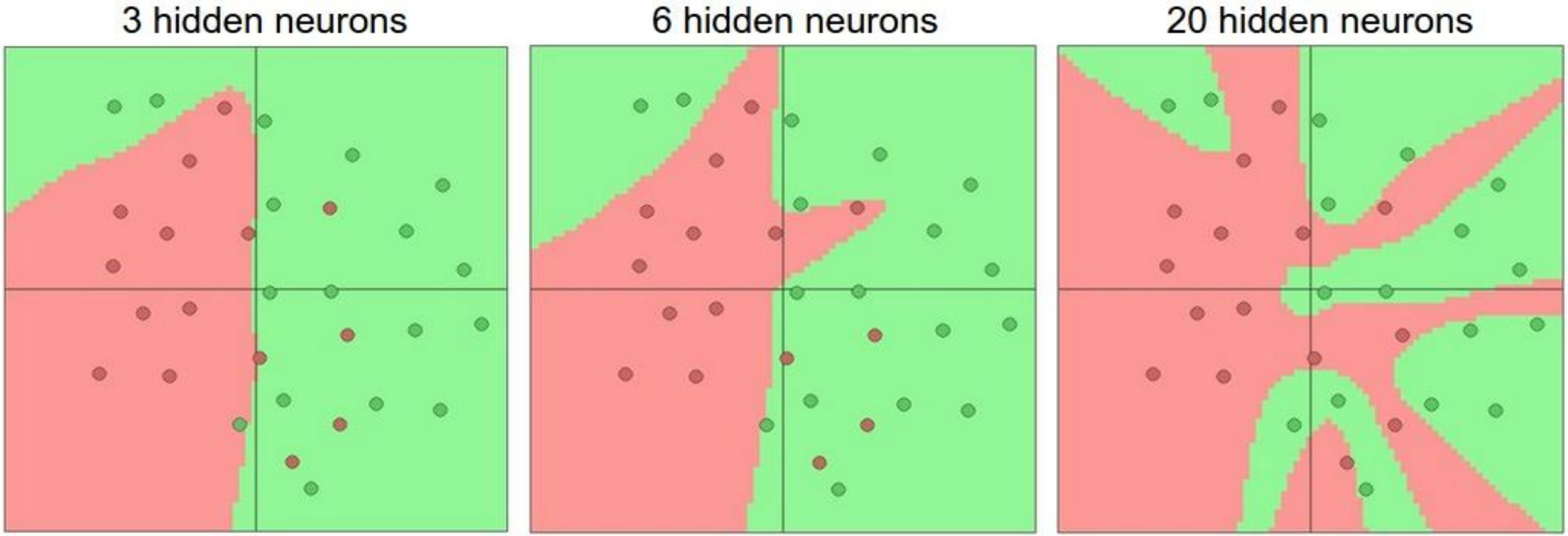
HYPERPARAMETERS IN MULTI-LAYER NETWORKS

- Number of layers, number of units per layer, activation function, loss function, regularization, regularization constant, optimizer, optimizer settings: learning rate schedule, number of epochs, batch size, etc.



Source: Stanford 231n

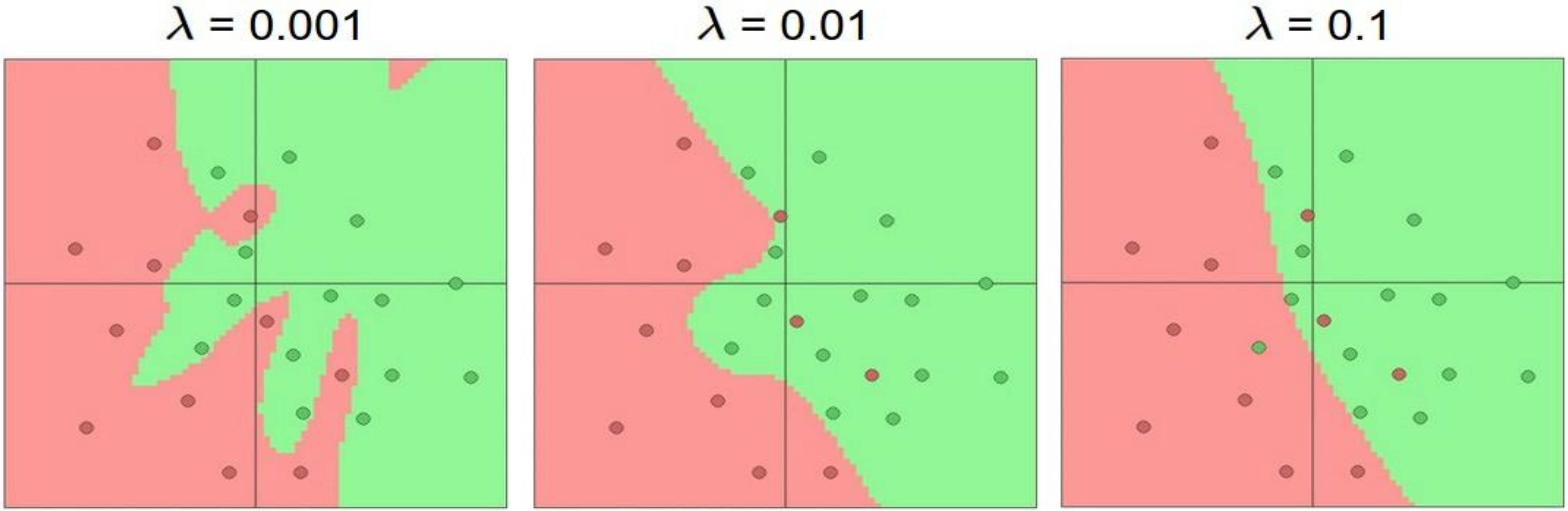
HYPERPARAMETERS IN MULTI-LAYER NETWORKS



Number of hidden units in a two-layer network

Source: Stanford 231n

HYPERPARAMETERS IN MULTI-LAYER NETWORKS

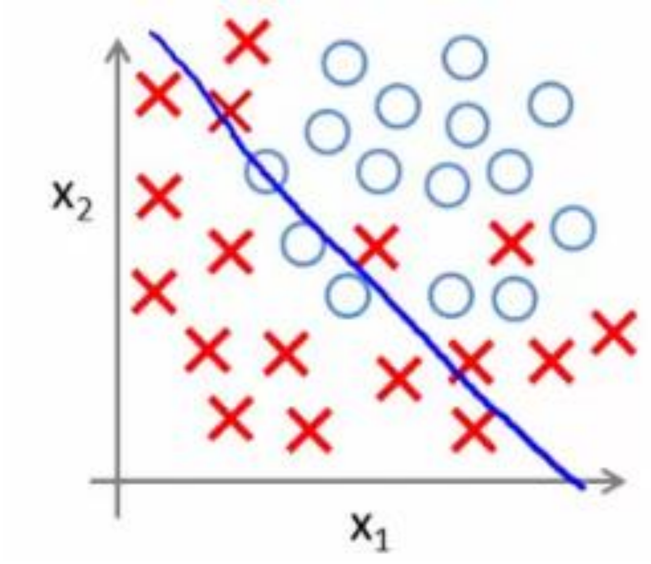


Regularization constant

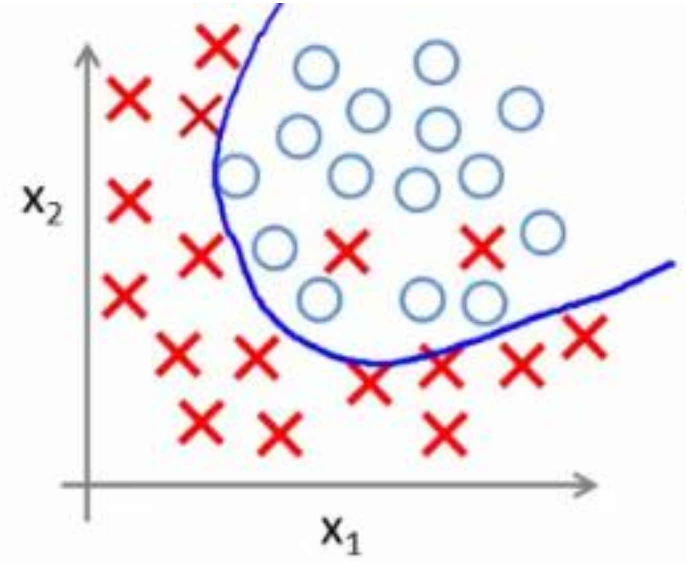
Source: Stanford 231n

MODEL COMPLEXITY AND GENERALIZATION

"Simple" model



"Intermediate" model



"Complex" model

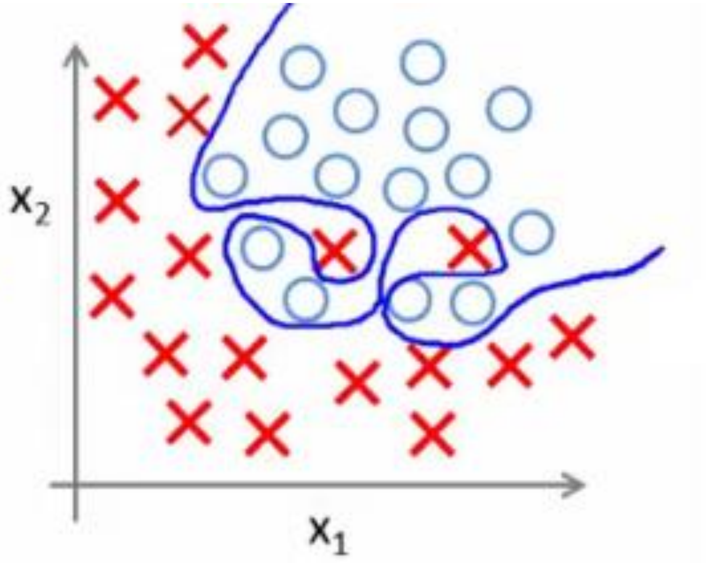


Figure source

MODEL COMPLEXITY AND GENERALIZATION

- Generalization (test) error of learning algorithms has two main components:
  - **Bias:** error due to simplifying model assumptions
  - **Variance:** error due to randomness of training set

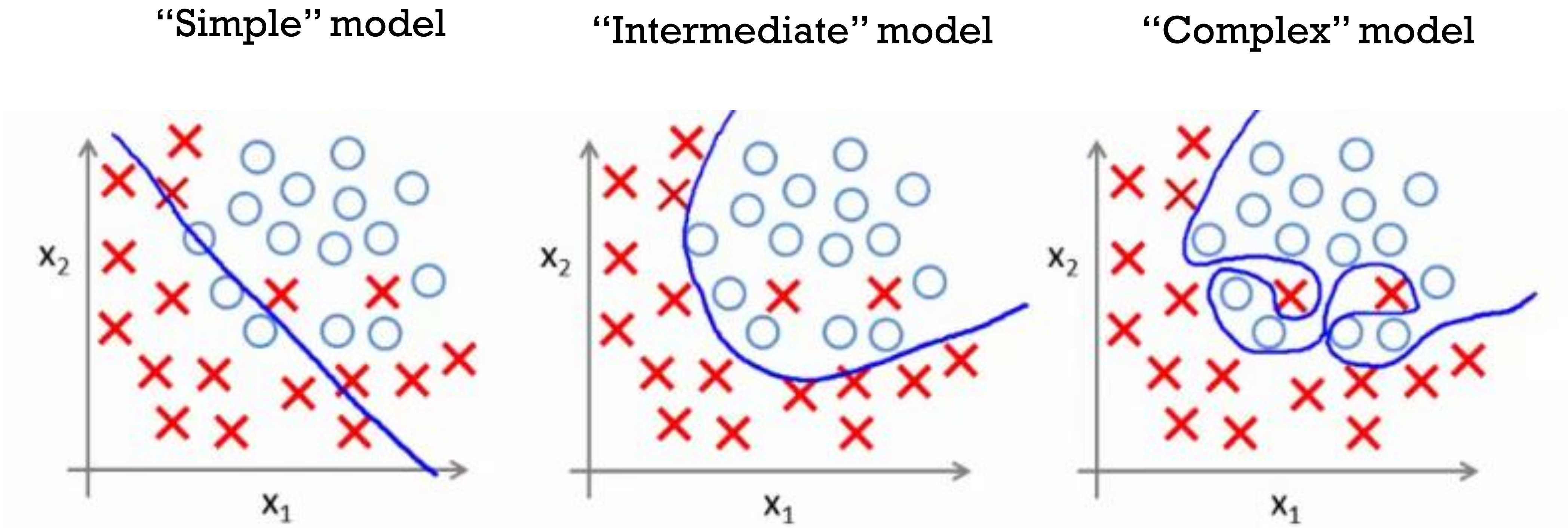


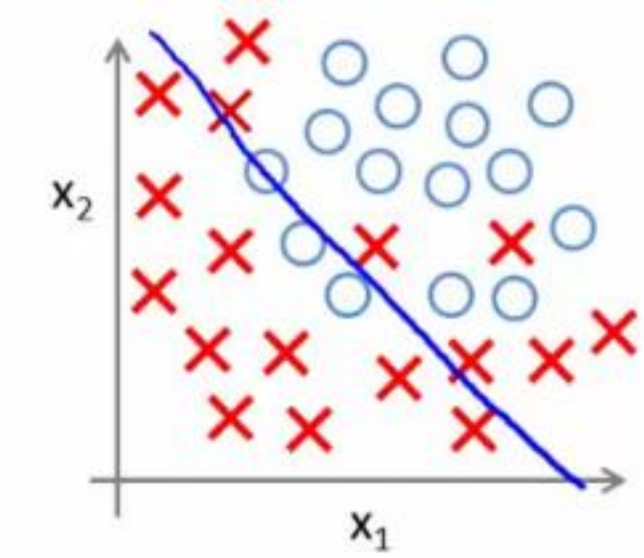
Figure source

MODEL COMPLEXITY AND GENERALIZATION

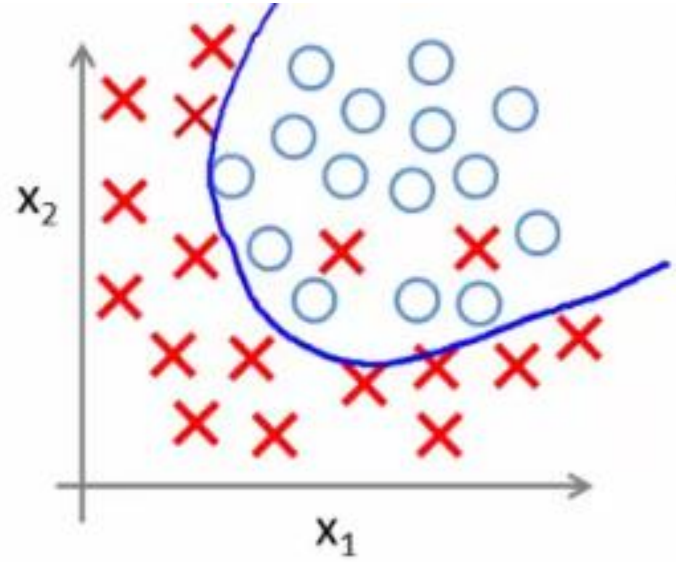
- Generalization (test) error of learning algorithms has two main components:
  - **Bias:** error due to simplifying model assumptions
  - **Variance:** error due to randomness of training set

"Simple" model

High bias, low variance



"Intermediate" model



"Complex" model

Low bias, high variance

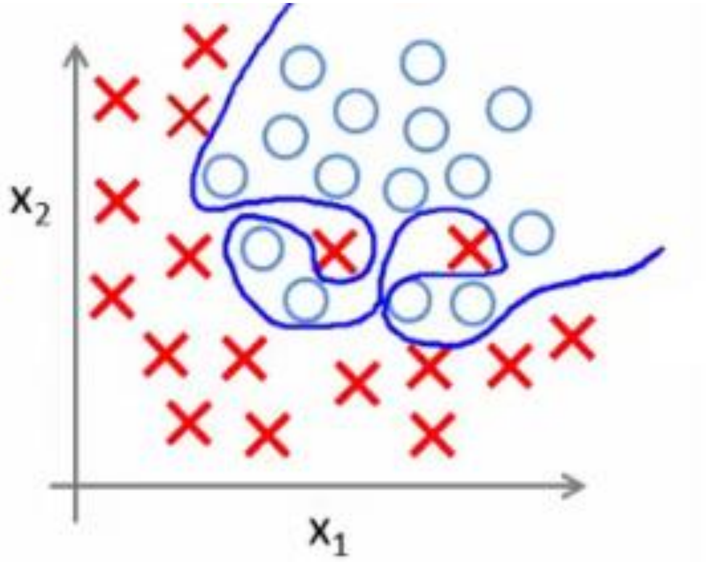


Figure source

BIAS-VARIANCE TRADEOFF

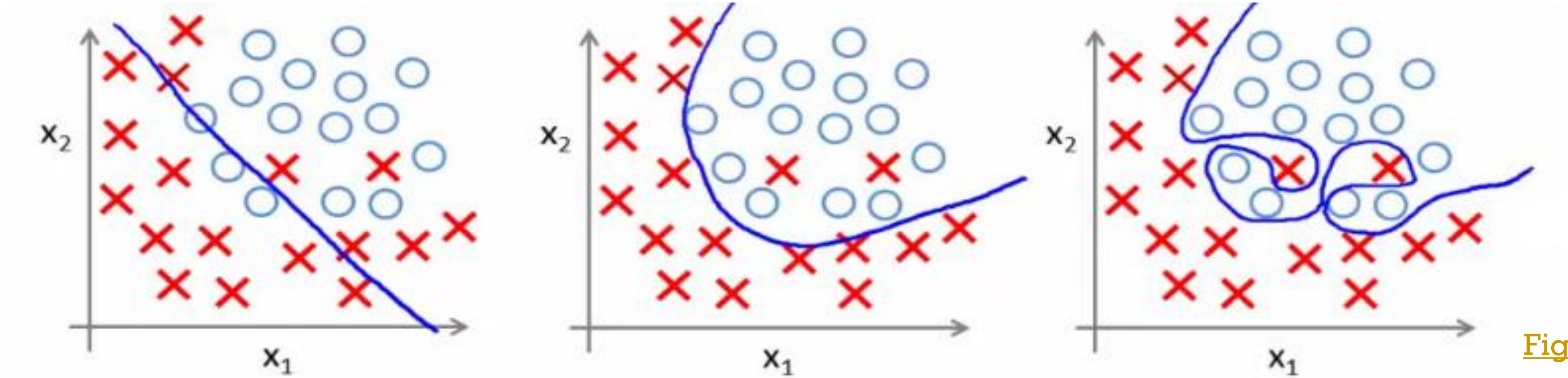


Figure source

BIAS-VARIANCE TRADEOFF

- What if your model **bias** is too high?

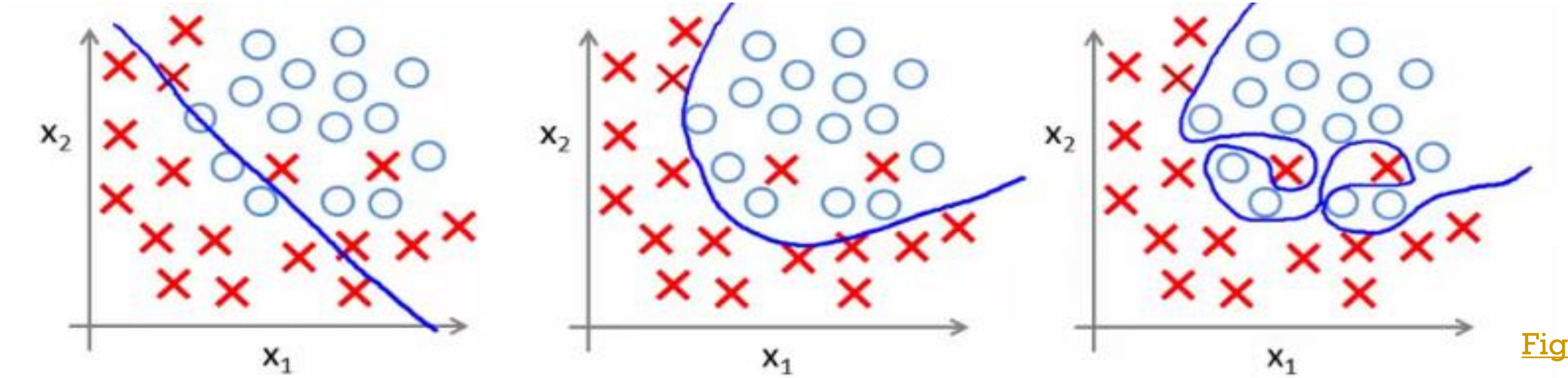


Figure source

BIAS-VARIANCE TRADEOFF

- What if your model **bias** is too high?
  - Your model is **underfitting** – it is incapable of capturing the important characteristics of the training data

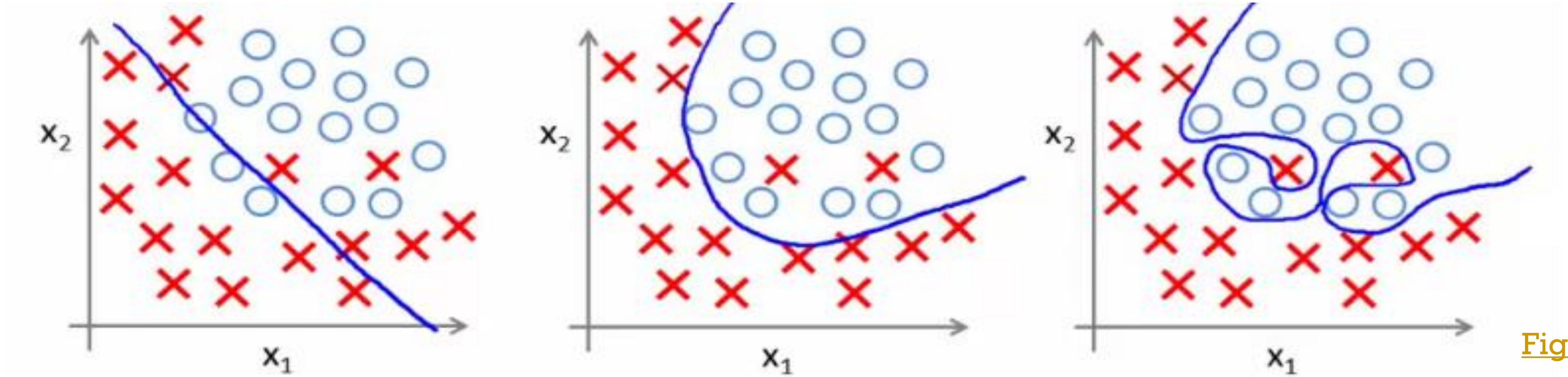
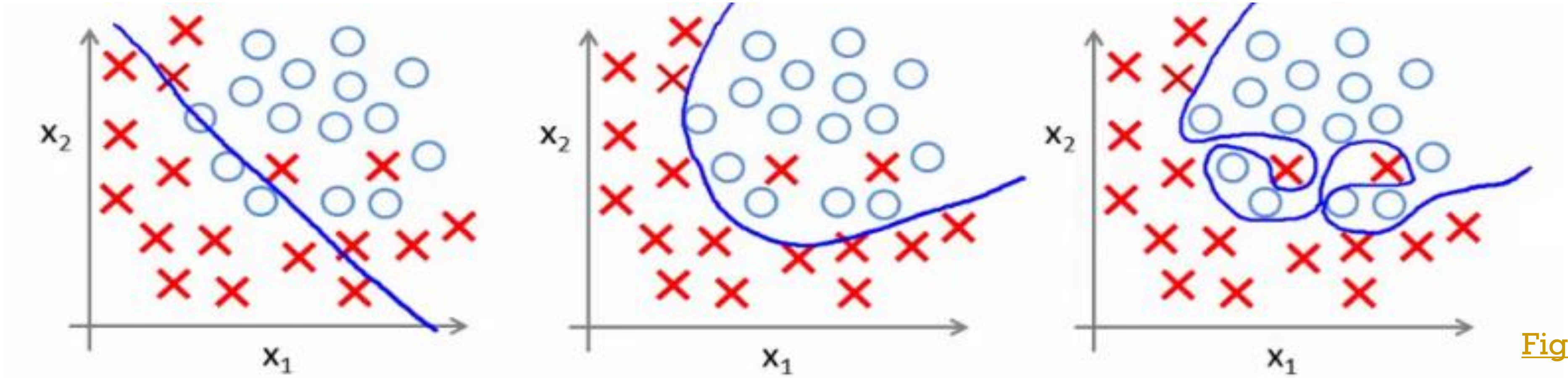

Underfitting

Figure source

BIAS-VARIANCE TRADEOFF

- What if your model **bias** is too high?
  - Your model is **underfitting** – it is incapable of capturing the important characteristics of the training data
- What if your model **variance** is too high?



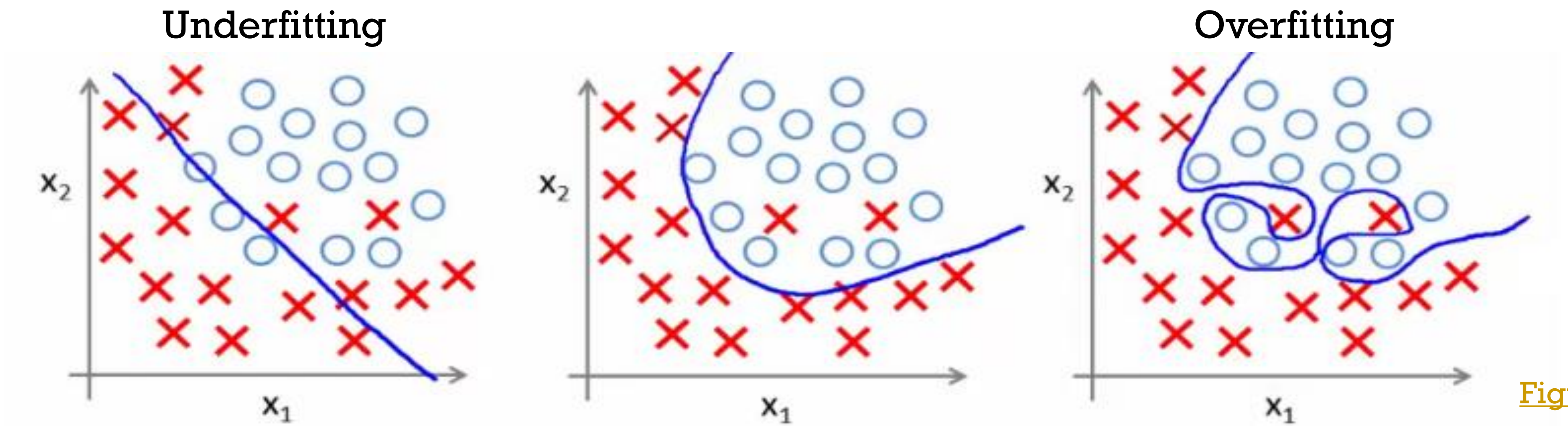
Underfitting

Figure source

BIAS-VARIANCE TRADEOFF

- What if your model **bias** is too high?
  - Your model is **underfitting** – it is incapable of capturing the important characteristics of the training data

- What if your model **variance** is too high?
  - Your model is **overfitting** – it is fitting noise and unimportant characteristics of the data


Underfitting

Overfitting

Figure source

BIAS-VARIANCE TRADEOFF

- What if your model **bias** is too high?
  - Your model is **underfitting** – it is incapable of capturing the important characteristics of the training data
- What if your model **variance** is too high?
  - Your model is **overfitting** – it is fitting noise and unimportant characteristics of the data
- How to recognize underfitting or overfitting?

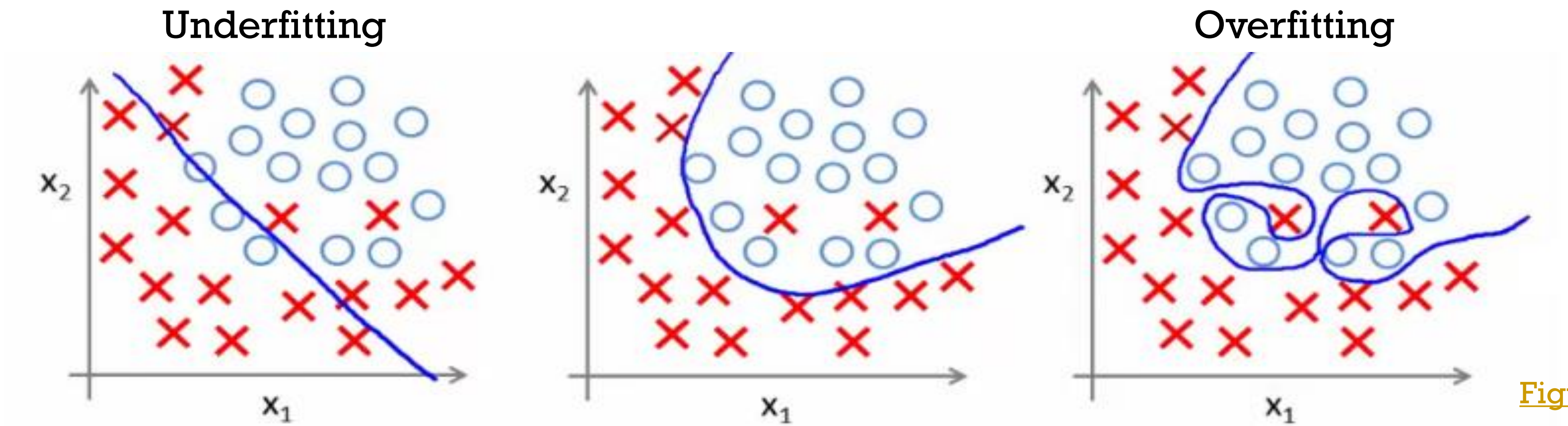


Figure source

BIAS-VARIANCE TRADEOFF

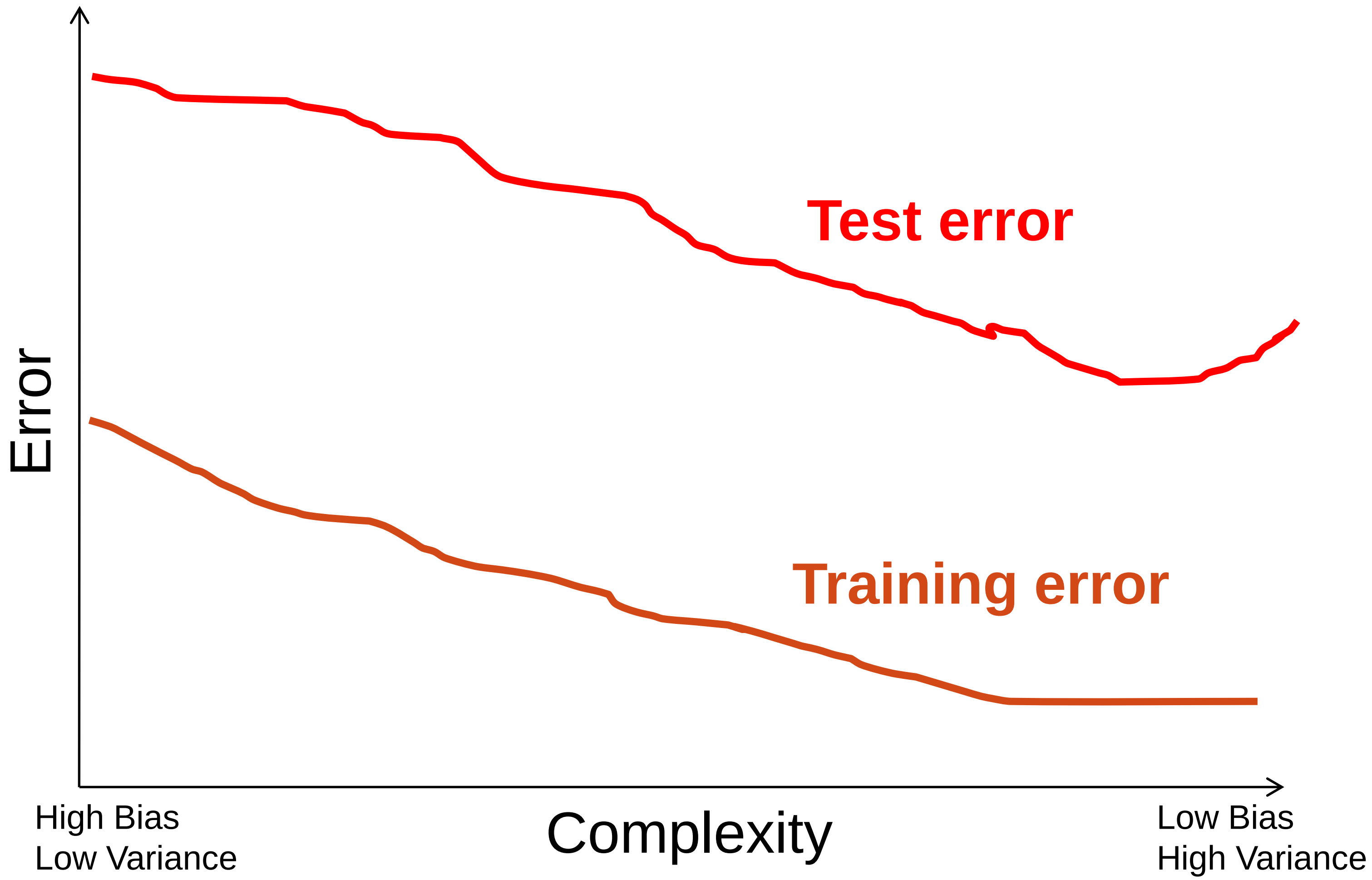
- What if your model **bias** is too high?
  - Your model is **underfitting** – it is incapable of capturing the important characteristics of the training data

- What if your model **variance** is too high?
  - Your model is **overfitting** – it is fitting noise and unimportant characteristics of the data

- How to recognize underfitting or overfitting?
  - Need to look at both training and test error
  - **Underfitting:** training and test error are both *high*
  - **Overfitting:** training error is *low*, test error is *high*

Figure source

LOOKING AT TRAINING AND TEST ERROR



Source: D. Hoiem

OPTIMIZATION

Optimization is the process of finding the set of parameters W that minimize the loss function.

Strategy #1:First very bad idea solution: Random search:

Simply try out many different random weights and keep track of what works best.

Source: http://cs231n.github.io

OPTIMIZATION

Optimization is the process of finding the set of parameters W that minimize the loss function.

Strategy #1:First very bad idea solution: Random search:

Simply try out many different random weights and keep track of what works best.

Strategy #2: Random local search:

Start out with a random W, generate random changes δW to it and if the loss at the changed $W+\delta W$ is lower, we will perform an update.

Source: http://cs231n.github.io

OPTIMIZATION

Optimization is the process of finding the set of parameters W that minimize the loss function.

Strategy #1:First very bad idea solution: Random search:

Simply try out many different random weights and keep track of what works best.

Strategy #2: Random local search:

Start out with a random W, generate random changes δW to it and if the loss at the changed $W+\delta W$ is lower, we will perform an update.

Strategy #3: Following the gradients:

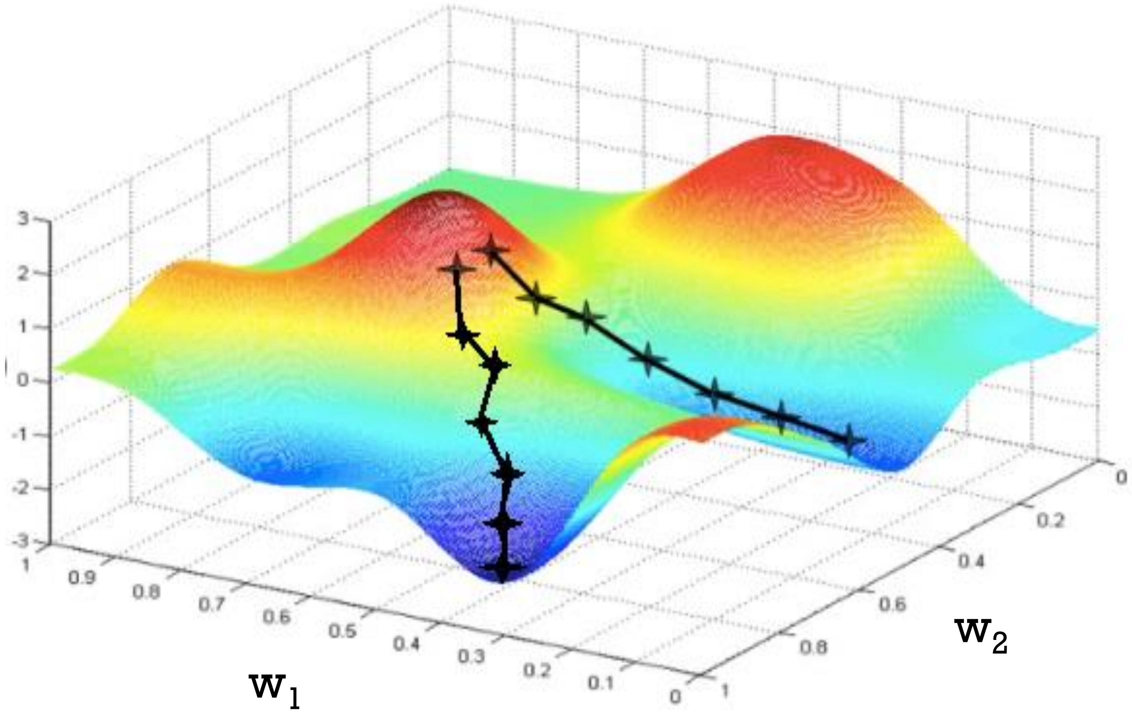
There is no need to randomly search for a good direction: this direction is related to the gradient of the loss function.

Source: http://cs231n.github.io

GRADIENT DESCENT

- Goal: find w to minimize loss $L(w)$
- Start with some initial estimate of w
- At each step, find $\nabla L(w)$, the *gradient* of the loss w.r.t. w, and take a small step in the *opposite* direction

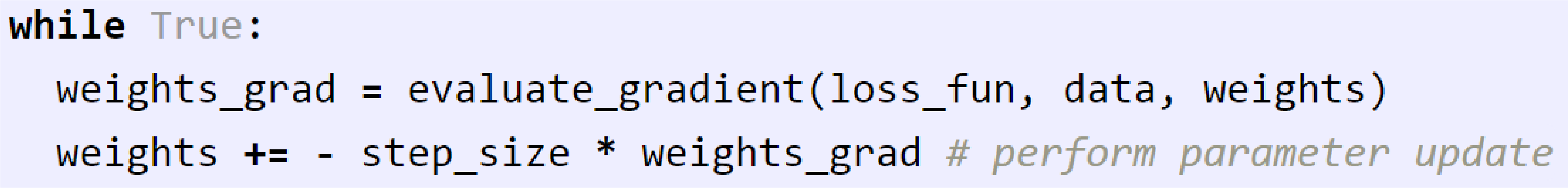
$$w \leftarrow w - \eta \, \nabla L(w)$$



GRADIENT DESCENT

The procedure of repeatedly evaluating the gradient of loss function and then performing a parameter update.

Vanilla (Original) Gradient Descent:



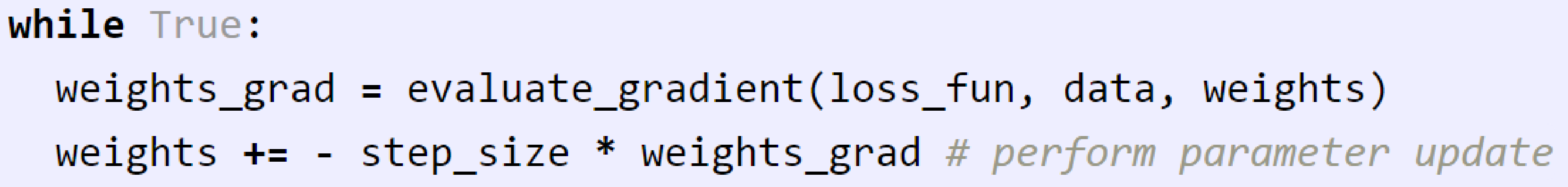
```
while True:
  weights_grad = evaluate_gradient(loss_fun, data, weights)
  weights += - step_size * weights_grad # perform parameter update
```

Source: http://cs231n.github.io

GRADIENT DESCENT

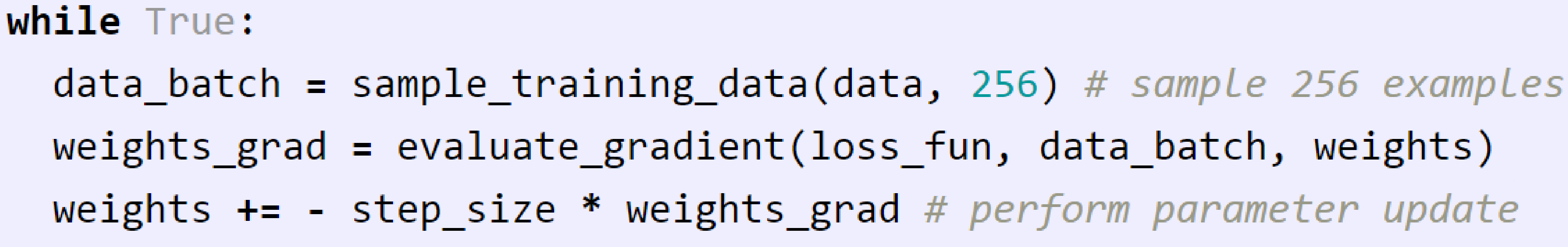
The procedure of repeatedly evaluating the gradient of loss function and then performing a parameter update.

Vanilla (Original) Gradient Descent:



```
while True:
  weights_grad = evaluate_gradient(loss_fun, data, weights)
  weights += - step_size * weights_grad # perform parameter update
```

Mini-batch Gradient Descent (MGD)*:*



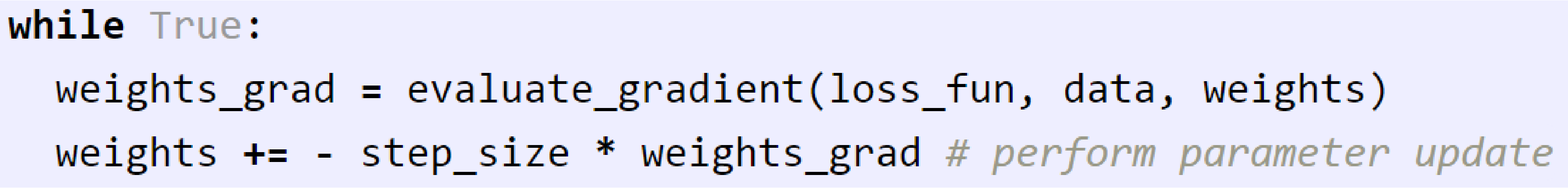
```
while True:
  data_batch = sample_training_data(data, 256) # sample 256 examples
  weights_grad = evaluate_gradient(loss_fun, data_batch, weights)
  weights += - step_size * weights_grad # perform parameter update
```

Source: http://cs231n.github.io

GRADIENT DESCENT

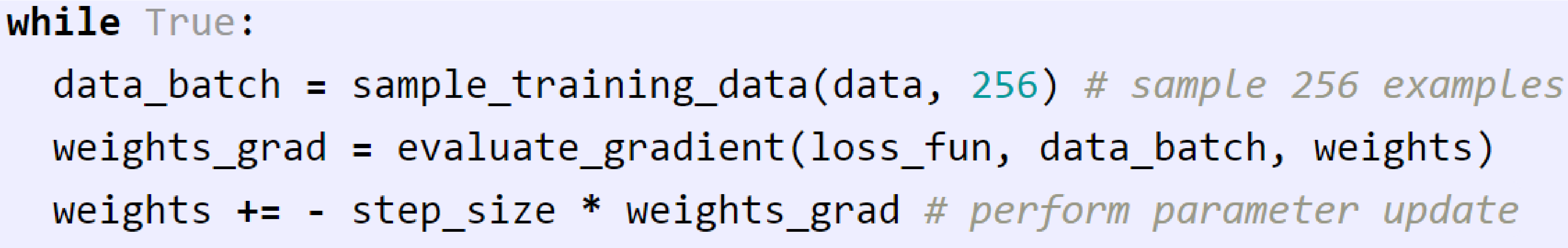
The procedure of repeatedly evaluating the gradient of loss function and then performing a parameter update.

Vanilla (Original) Gradient Descent:



```
while True:
  weights_grad = evaluate_gradient(loss_fun, data, weights)
  weights += - step_size * weights_grad # perform parameter update
```

Mini-batch Gradient Descent (MGD)*:*



```
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
```

Stochastic Gradient Descent (SGD)*:*

Special case of MGD when mini-batch contains only a single example

Source: http://cs231n.github.io

INTERPRETATION OF THE GRADIENT

Interpretation. Derivatives indicate the rate of change of a function with respect to that variable surrounding an infinitesimally small region near a particular point:

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

Source: http://cs231n.github.io

INTERPRETATION OF THE GRADIENT

Interpretation. Derivatives indicate the rate of change of a function with respect to that variable surrounding an infinitesimally small region near a particular point:

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

$$f(x,y) = x + y \qquad \rightarrow \qquad \frac{\partial f}{\partial x} = \qquad \frac{\partial f}{\partial y} =$$

Source: http://cs231n.github.io

INTERPRETATION OF THE GRADIENT

Interpretation. Derivatives indicate the rate of change of a function with respect to that variable surrounding an infinitesimally small region near a particular point:

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

$$f(x,y) = x + y \qquad \rightarrow \qquad \frac{\partial f}{\partial x} = 1 \qquad \frac{\partial f}{\partial y} = 1$$

Source: http://cs231n.github.io

INTERPRETATION OF THE GRADIENT

Interpretation. Derivatives indicate the rate of change of a function with respect to that variable surrounding an infinitesimally small region near a particular point:

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

$$f(x,y) = x + y \qquad \to \qquad \frac{\partial f}{\partial x} = 1 \qquad \frac{\partial f}{\partial y} = 1$$

$$f(x,y) = xy \qquad \to \qquad \frac{\partial f}{\partial x} = \qquad \frac{\partial f}{\partial y} =$$

Source: http://cs231n.github.io

INTERPRETATION OF THE GRADIENT

Interpretation. Derivatives indicate the rate of change of a function with respect to that variable surrounding an infinitesimally small region near a particular point:

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

$$f(x,y) = x + y \qquad \to \qquad \frac{\partial f}{\partial x} = 1 \qquad \frac{\partial f}{\partial y} = 1$$

$$f(x,y) = xy \qquad \to \qquad \frac{\partial f}{\partial x} = y \qquad \frac{\partial f}{\partial y} = x$$

Source: http://cs231n.github.io

INTERPRETATION OF THE GRADIENT

Interpretation. Derivatives indicate the rate of change of a function with respect to that variable surrounding an infinitesimally small region near a particular point:

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

$$f(x,y) = x + y \qquad \rightarrow \qquad \frac{\partial f}{\partial x} = 1 \qquad \frac{\partial f}{\partial y} = 1$$

$$f(x,y) = xy \qquad \rightarrow \qquad \frac{\partial f}{\partial x} = y \qquad \frac{\partial f}{\partial y} = x$$

$$f(x,y) = \max(x,y) \qquad \rightarrow \qquad \frac{\partial f}{\partial x} = \qquad \frac{\partial f}{\partial y} =$$

Source: http://cs231n.github.io

INTERPRETATION OF THE GRADIENT

Interpretation. Derivatives indicate the rate of change of a function with respect to that variable surrounding an infinitesimally small region near a particular point:

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

$$f(x,y) = x + y \qquad \to \qquad \frac{\partial f}{\partial x} = 1 \qquad \frac{\partial f}{\partial y} = 1$$

$$f(x,y) = xy \qquad \to \qquad \frac{\partial f}{\partial x} = y \qquad \frac{\partial f}{\partial y} = x$$

$$f(x,y) = \max(x,y) \qquad \to \qquad \frac{\partial f}{\partial x} = 1(x >= y) \qquad \frac{\partial f}{\partial y} = 1(y >= x)$$

Source: http://cs231n.github.io

COMPOUND EXPRESSIONS WITH CHAIN RULE

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

Source: http://cs231n.github.io

COMPOUND EXPRESSIONS WITH CHAIN RULE

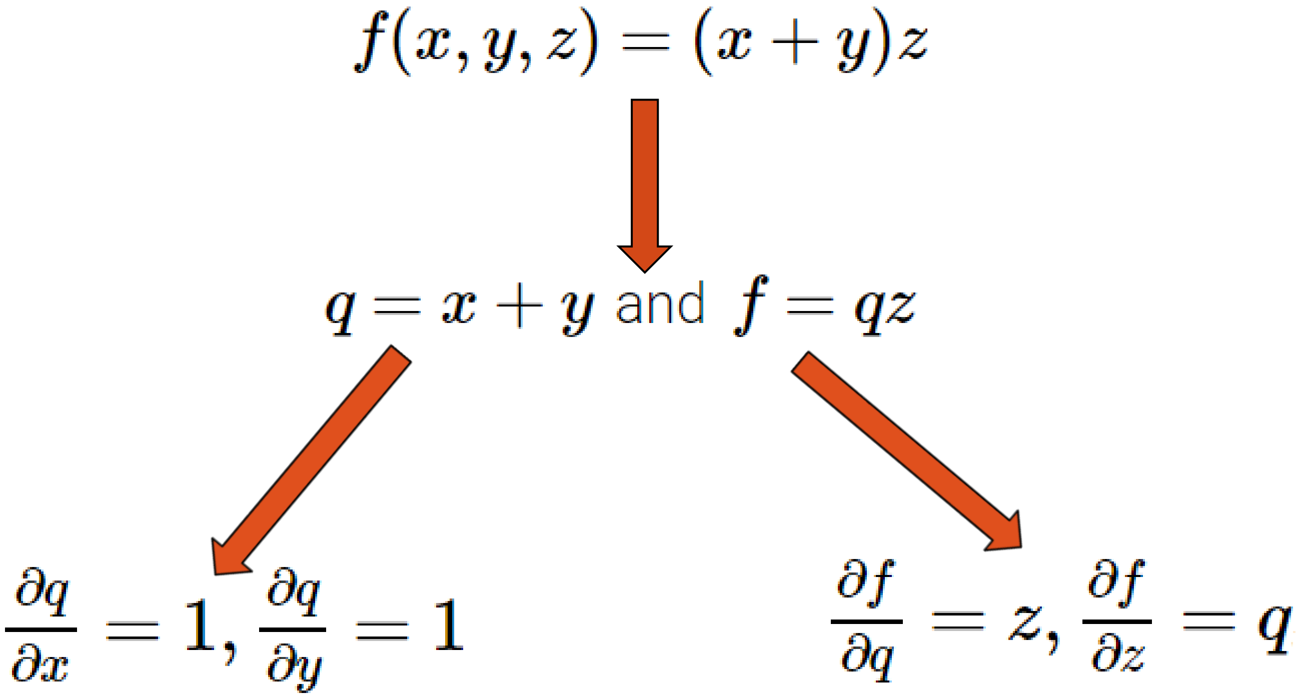
$$f(x, y, z) = (x + y)z$$

$$\downarrow$$

$q = x + y$ and $f = qz$

Source: http://cs231n.github.io

COMPOUND EXPRESSIONS WITH CHAIN RULE



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

$q = x + y$ and $f = qz$

$$\frac{\partial q}{\partial x} = 1, \frac{\partial q}{\partial y} = 1$$

$$\frac{\partial f}{\partial q} = z, \frac{\partial f}{\partial z} = q$$

Source: http://cs231n.github.io

COMPOUND EXPRESSIONS WITH CHAIN RULE

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

$q = x + y$ and $f = qz$

$$\frac{\partial q}{\partial x} = 1, \frac{\partial q}{\partial y} = 1$$

$$\frac{\partial f}{\partial q} = z, \frac{\partial f}{\partial z} = q$$

Chain rule: $$\frac{\partial f}{\partial x} = \frac{\partial f}{\partial q} \frac{\partial q}{\partial x}$$

Source: http://cs231n.github.io

COMPOUND EXPRESSIONS WITH CHAIN RULE

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

$$\downarrow$$

$q = x + y$ and $f = qz$

$$\frac{\partial f}{\partial q} = z, \frac{\partial f}{\partial z} = q$$

$$\frac{\partial q}{\partial x} = 1, \frac{\partial q}{\partial y} = 1$$

$$\frac{\partial f}{\partial x} = \frac{\partial f}{\partial q}\frac{\partial q}{\partial x}$$

Source: http://cs231n.github.io

COMPOUND EXPRESSIONS WITH CHAIN RULE

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

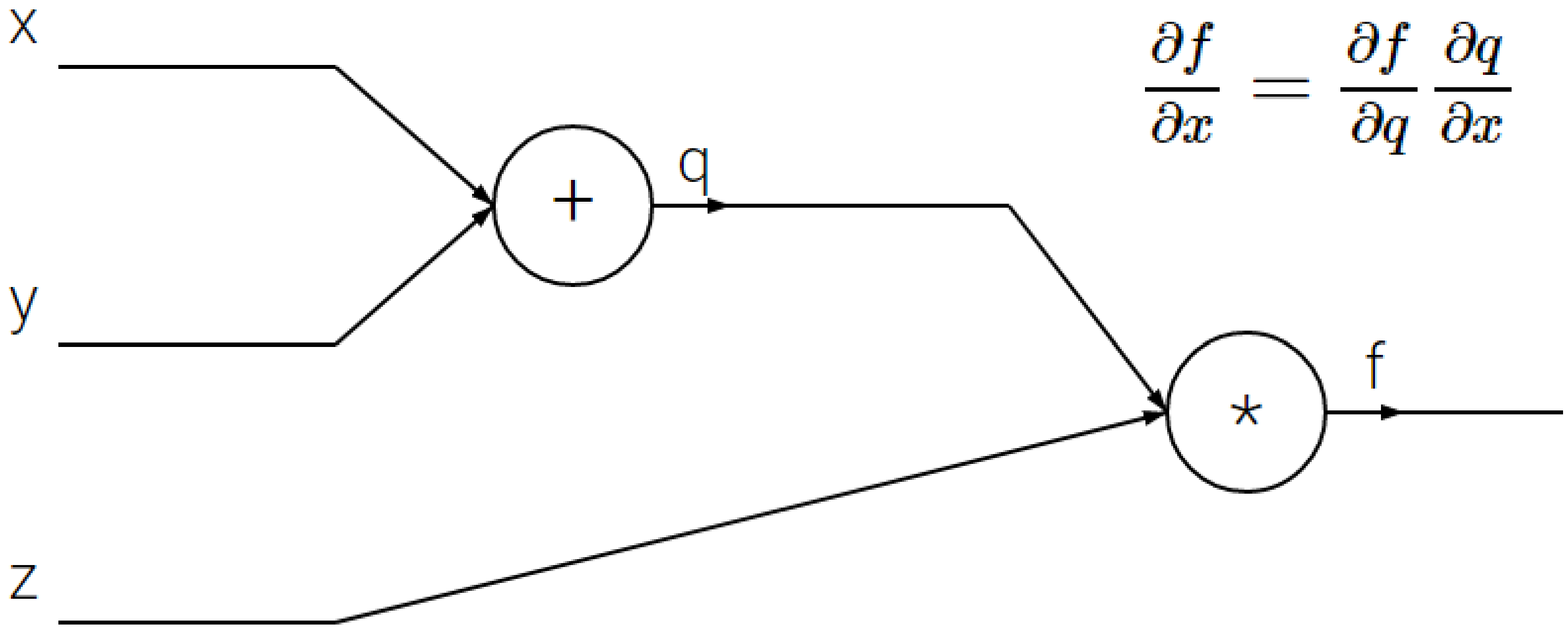
↓

$q = x + y$ and $f = qz$

$$\frac{\partial f}{\partial q} = z, \frac{\partial f}{\partial z} = q$$

$$\frac{\partial q}{\partial x} = 1, \frac{\partial q}{\partial y} = 1$$

$$\frac{\partial f}{\partial x} = \frac{\partial f}{\partial q} \frac{\partial q}{\partial x}$$



Source: http://cs231n.github.io

COMPOUND EXPRESSIONS WITH CHAIN RULE

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

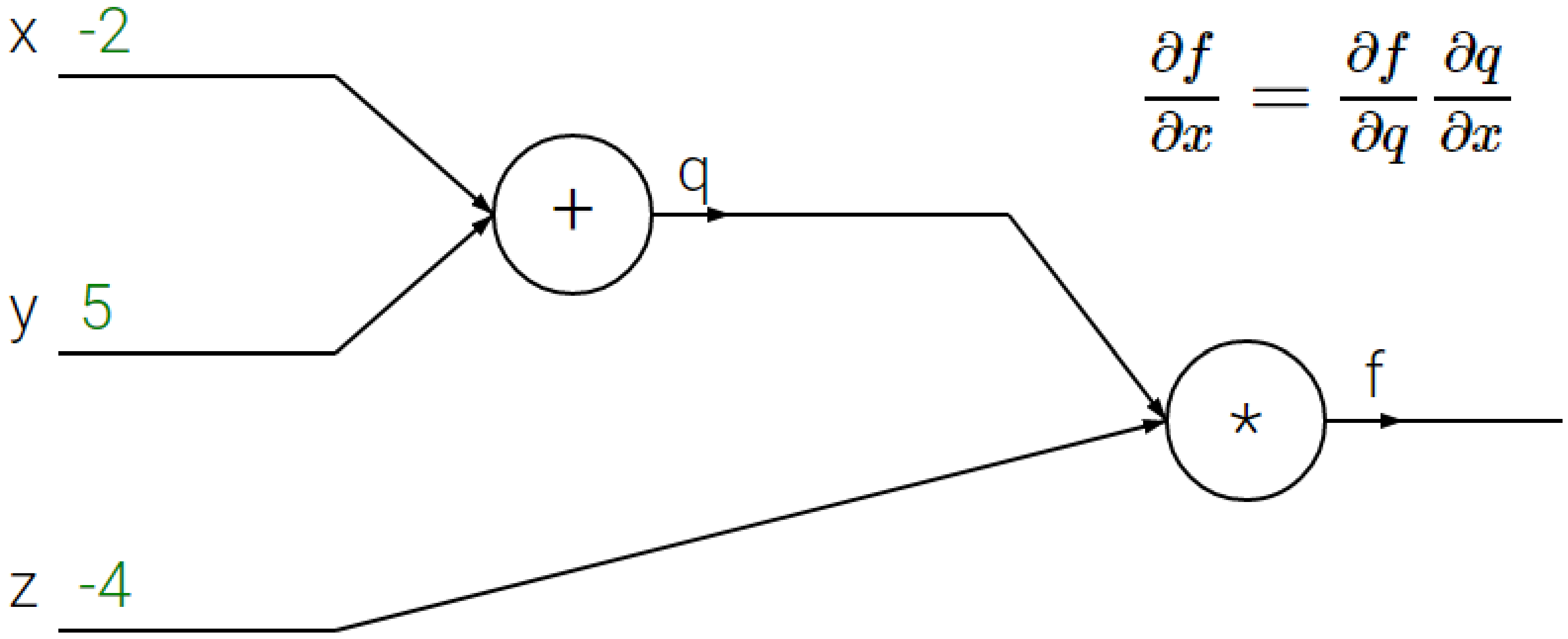
↓

$q = x + y$ and $f = qz$

$$\frac{\partial f}{\partial q} = z, \frac{\partial f}{\partial z} = q$$

$$\frac{\partial q}{\partial x} = 1, \frac{\partial q}{\partial y} = 1$$

$$\frac{\partial f}{\partial x} = \frac{\partial f}{\partial q}\frac{\partial q}{\partial x}$$



Source: http://cs231n.github.io

COMPOUND EXPRESSIONS WITH CHAIN RULE

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

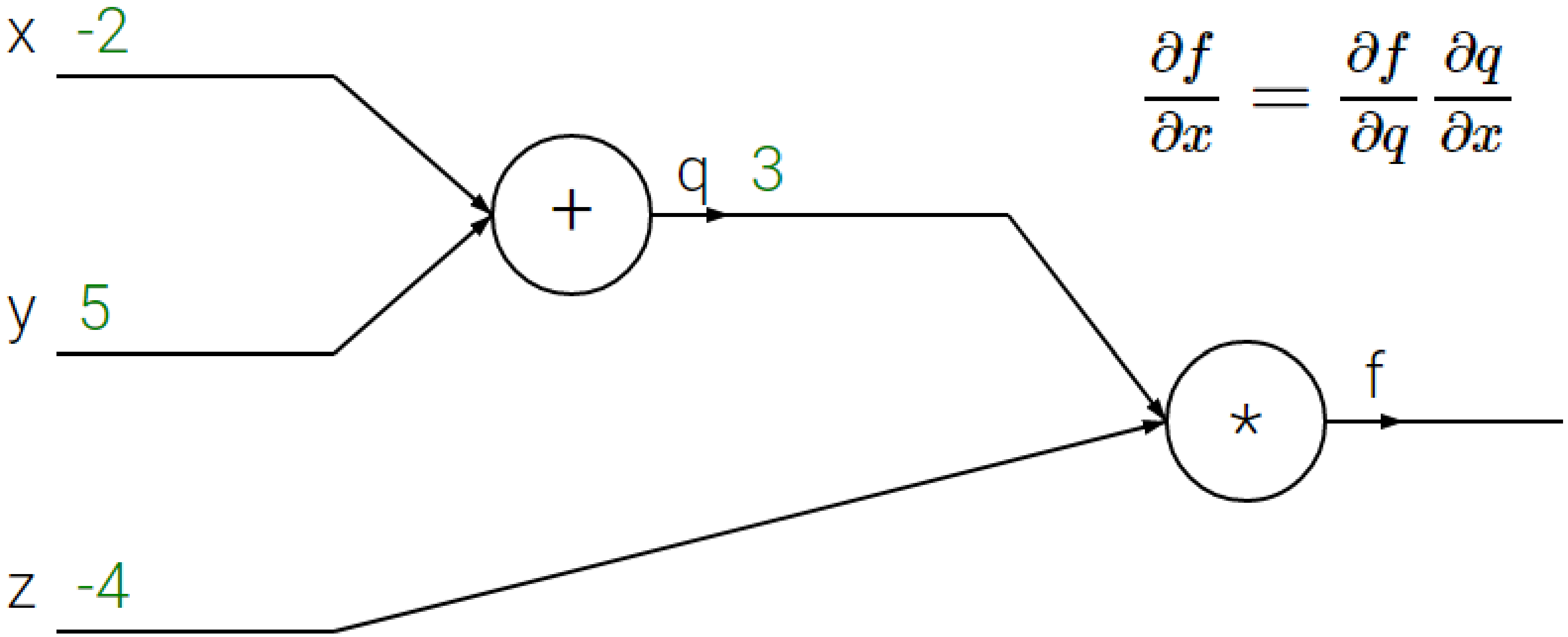
↓

$q = x + y$ and $f = qz$

$$\frac{\partial f}{\partial q} = z, \frac{\partial f}{\partial z} = q$$

$$\frac{\partial q}{\partial x} = 1, \frac{\partial q}{\partial y} = 1$$

$$\frac{\partial f}{\partial x} = \frac{\partial f}{\partial q} \frac{\partial q}{\partial x}$$



Source: http://cs231n.github.io

COMPOUND EXPRESSIONS WITH CHAIN RULE

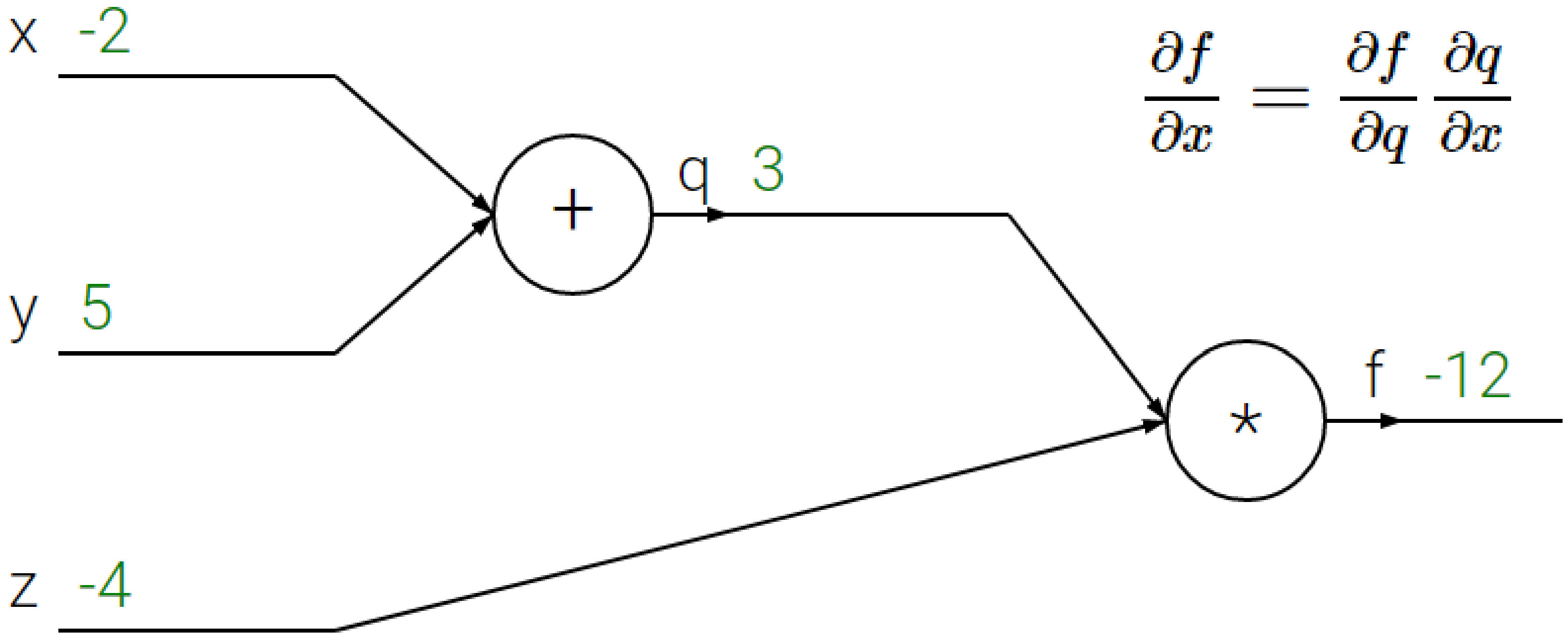
$$f(x,y,z) = (x+y)z$$

$q = x + y$ and $f = qz$

$$\frac{\partial f}{\partial q} = z, \frac{\partial f}{\partial z} = q$$

$$\frac{\partial q}{\partial x} = 1, \frac{\partial q}{\partial y} = 1$$

$$\frac{\partial f}{\partial x} = \frac{\partial f}{\partial q} \frac{\partial q}{\partial x}$$



Source: http://cs231n.github.io

COMPOUND EXPRESSIONS WITH CHAIN RULE

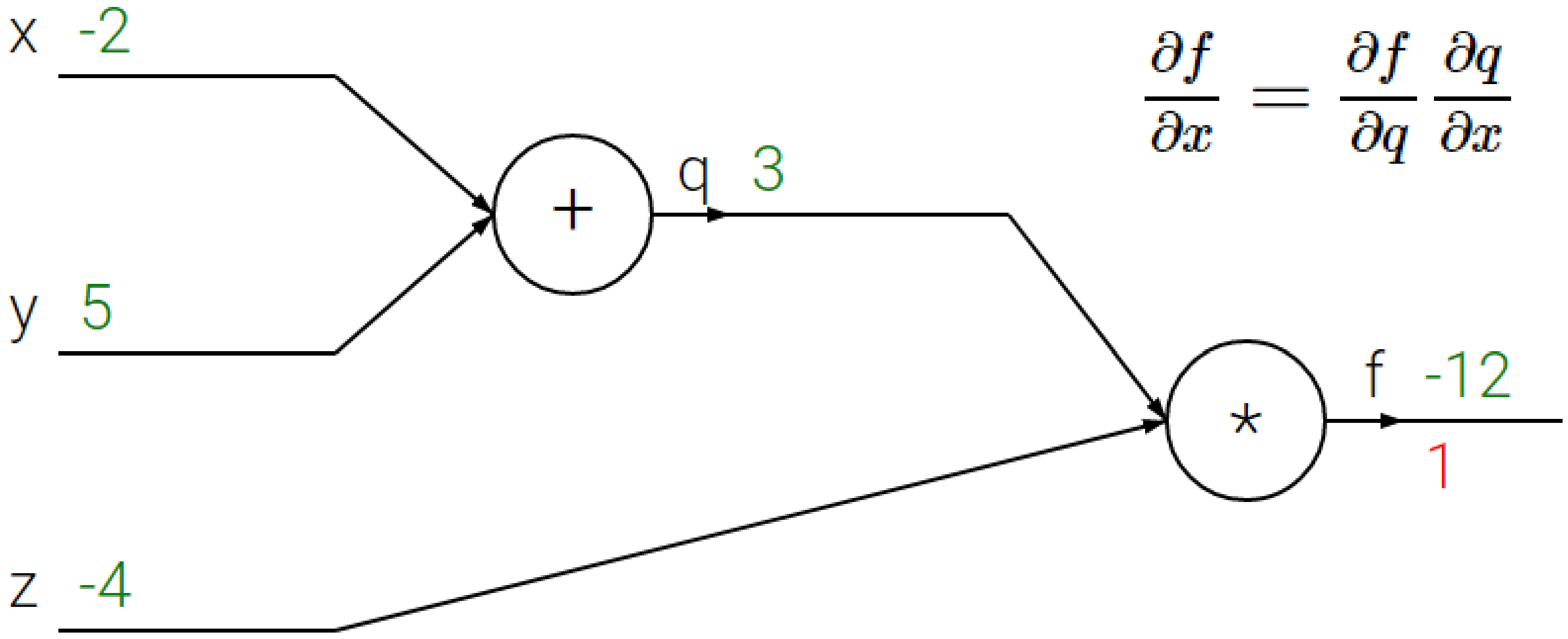
$$f(x, y, z) = (x + y)z$$

$q = x + y$ and $f = qz$

$$\frac{\partial f}{\partial q} = z, \frac{\partial f}{\partial z} = q$$

$$\frac{\partial q}{\partial x} = 1, \frac{\partial q}{\partial y} = 1$$

$$\frac{\partial f}{\partial x} = \frac{\partial f}{\partial q}\frac{\partial q}{\partial x}$$



Source: http://cs231n.github.io

COMPOUND EXPRESSIONS WITH CHAIN RULE

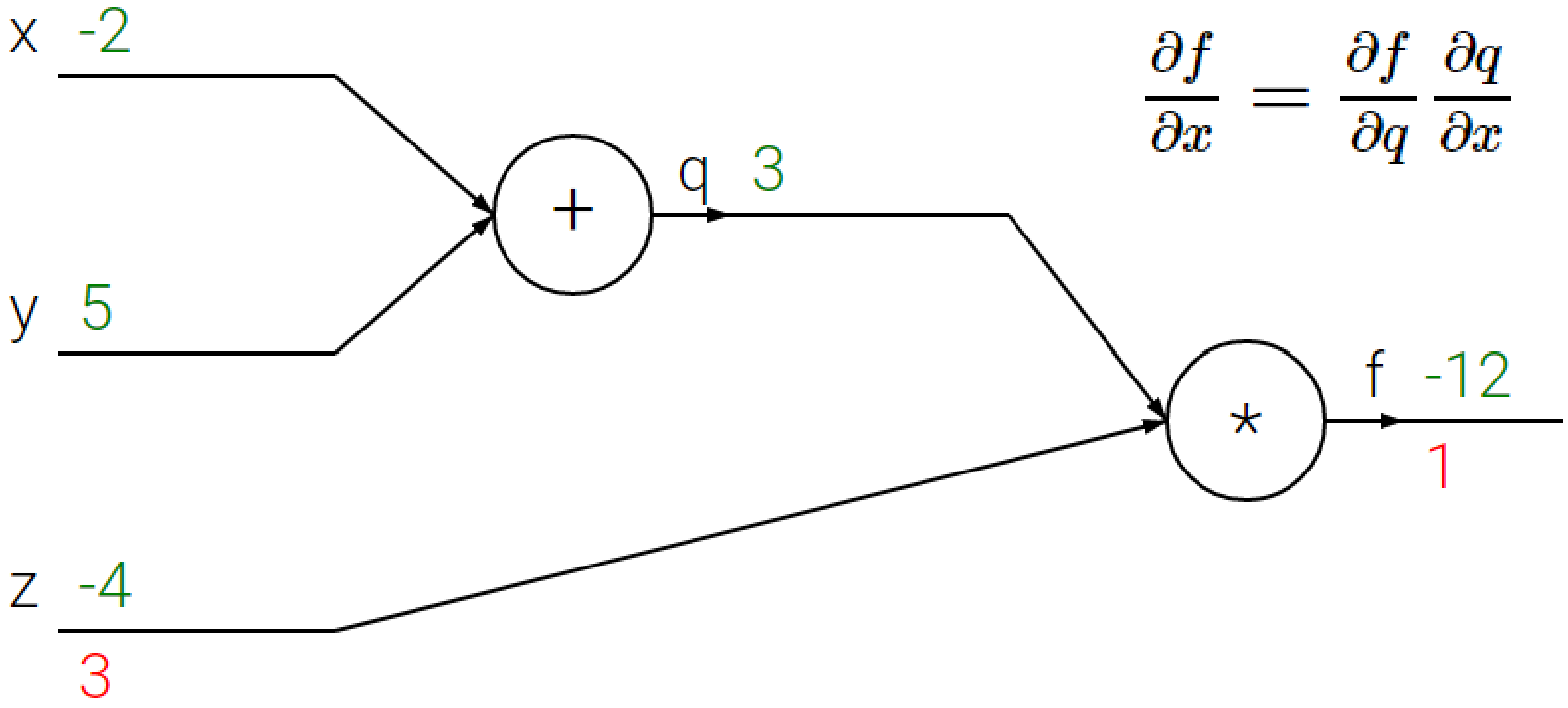
$$f(x, y, z) = (x + y)z$$

$q = x + y$ and $f = qz$

$$\frac{\partial f}{\partial q} = z, \frac{\partial f}{\partial z} = q$$

$$\frac{\partial q}{\partial x} = 1, \frac{\partial q}{\partial y} = 1$$

$$\frac{\partial f}{\partial x} = \frac{\partial f}{\partial q}\frac{\partial q}{\partial x}$$



Source: http://cs231n.github.io

COMPOUND EXPRESSIONS WITH CHAIN RULE

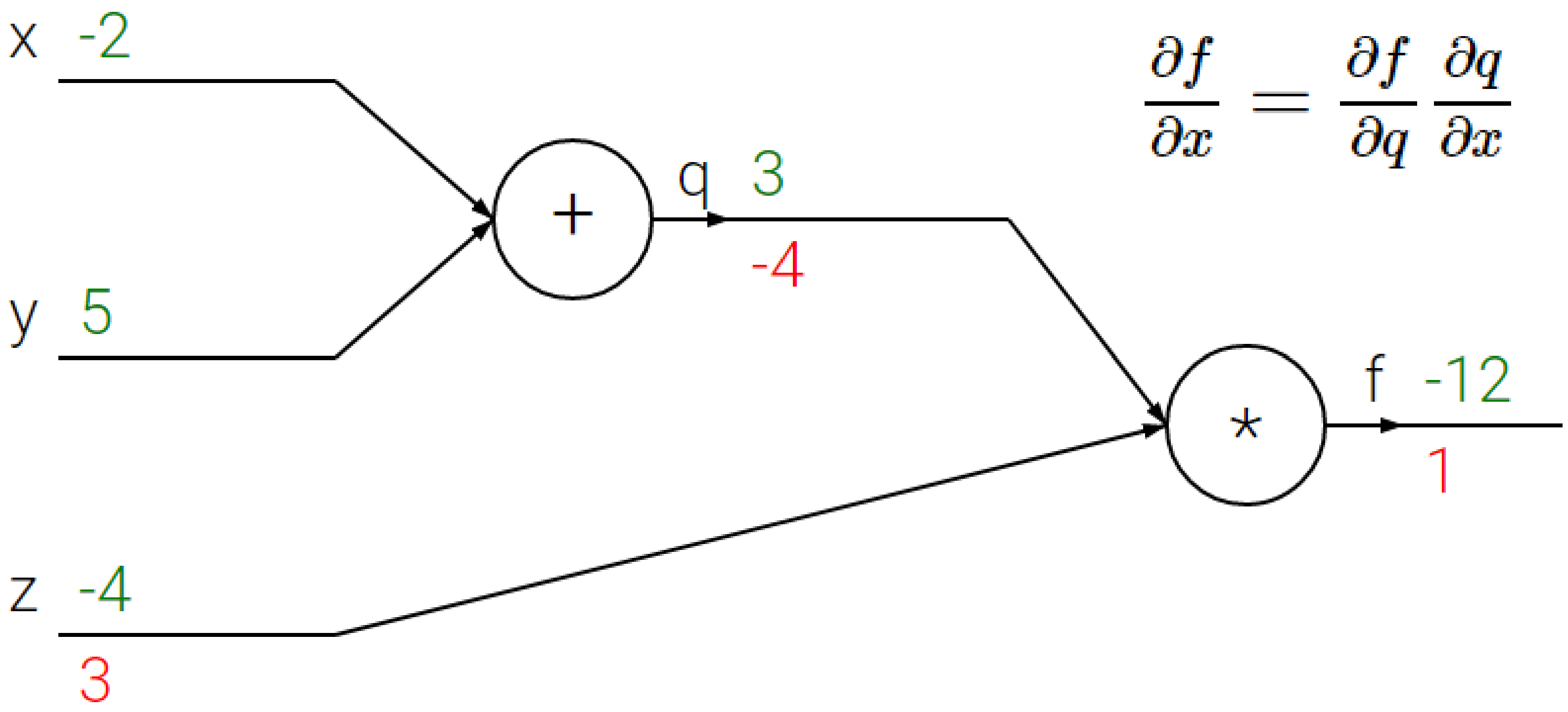
$$f(x, y, z) = (x + y)z$$

$q = x + y$ and $f = qz$

$$\frac{\partial f}{\partial q} = z, \frac{\partial f}{\partial z} = q$$

$$\frac{\partial q}{\partial x} = 1, \frac{\partial q}{\partial y} = 1$$

$$\frac{\partial f}{\partial x} = \frac{\partial f}{\partial q} \frac{\partial q}{\partial x}$$



Source: http://cs231n.github.io

COMPOUND EXPRESSIONS WITH CHAIN RULE

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

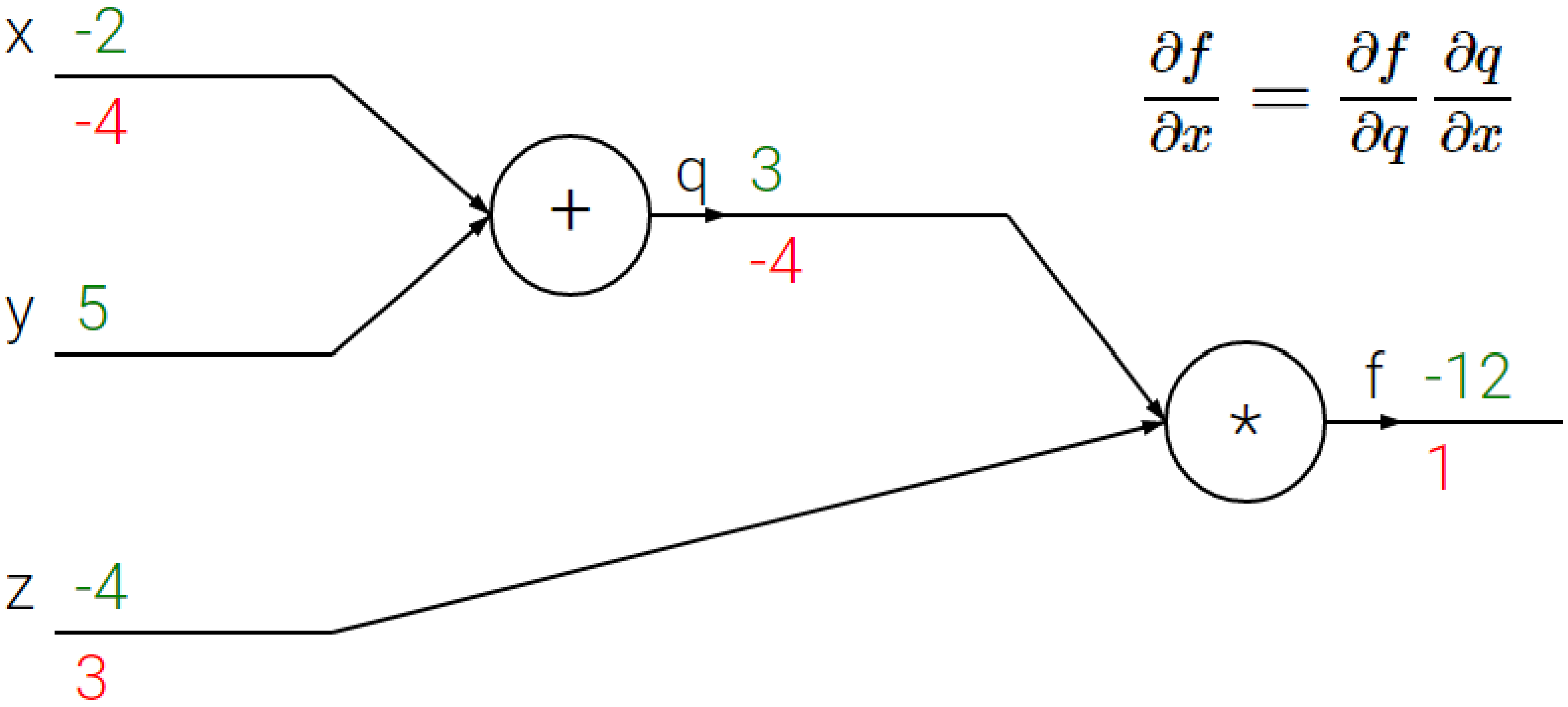
↓

$q = x + y$ and $f = qz$

$$\frac{\partial f}{\partial q} = z, \frac{\partial f}{\partial z} = q$$

$$\frac{\partial q}{\partial x} = 1, \frac{\partial q}{\partial y} = 1$$

$$\frac{\partial f}{\partial x} = \frac{\partial f}{\partial q} \frac{\partial q}{\partial x}$$



Source: http://cs231n.github.io

COMPOUND EXPRESSIONS WITH CHAIN RULE

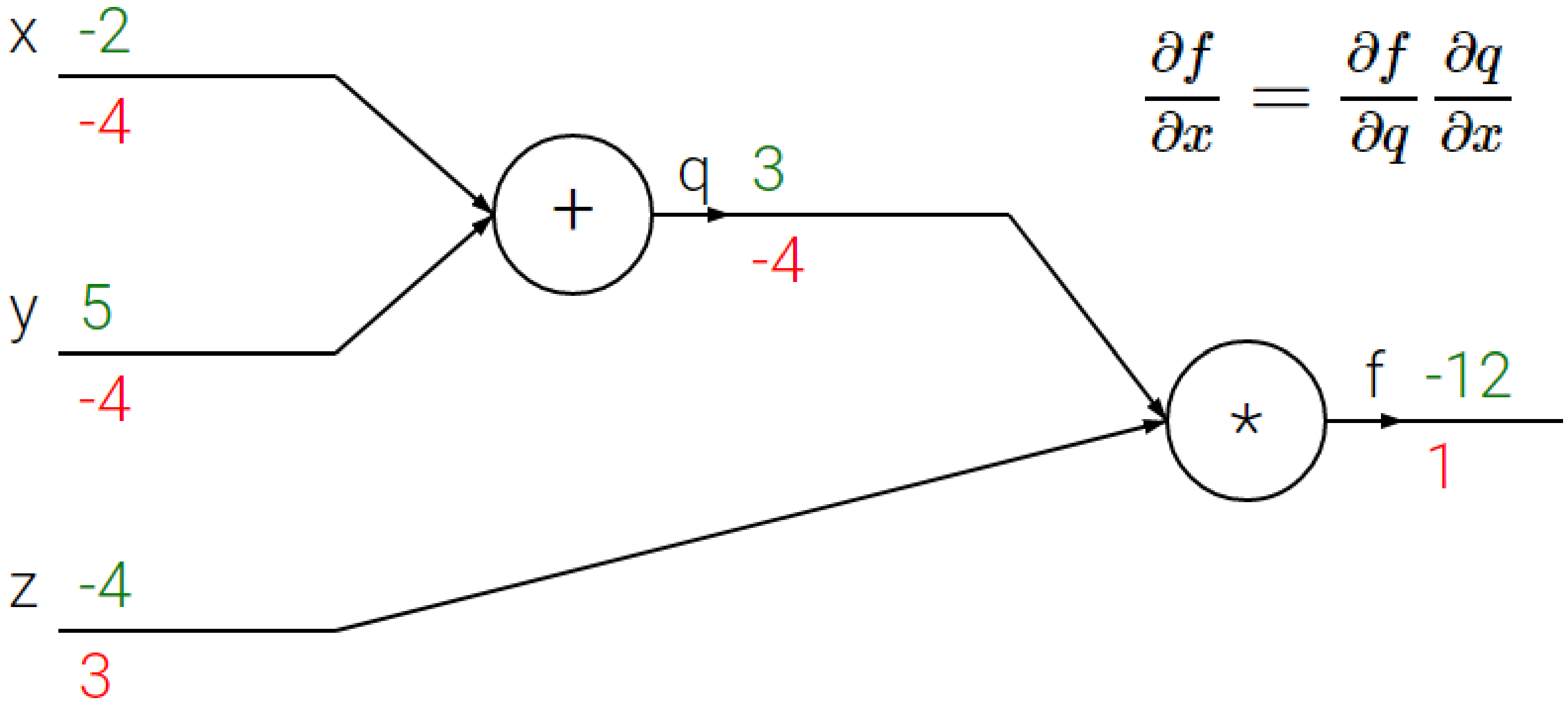
$$f(x,y,z) = (x+y)z$$

$$q = x + y \text{ and } f = qz$$

$$\frac{\partial f}{\partial q} = z, \frac{\partial f}{\partial z} = q$$

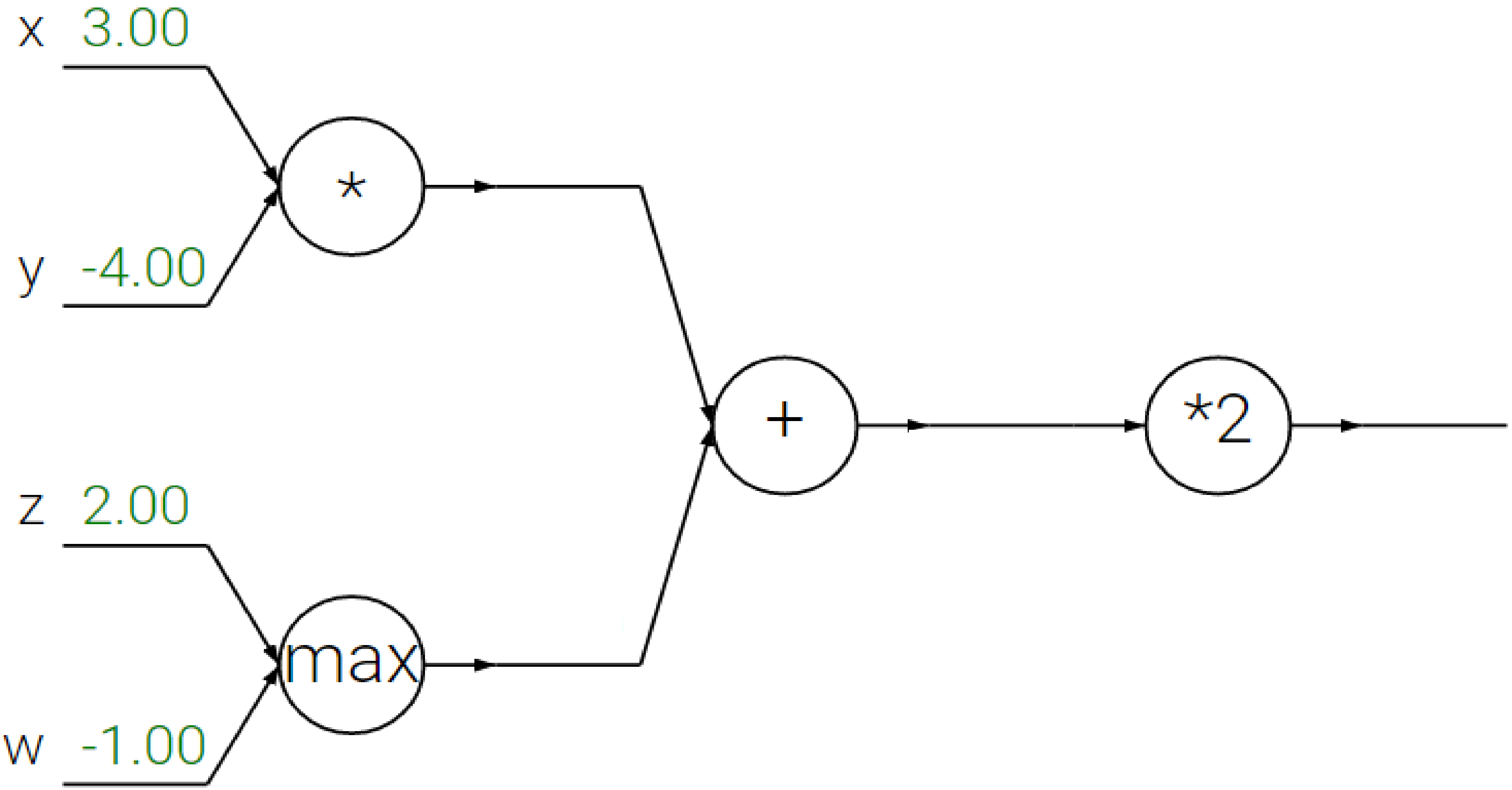
$$\frac{\partial q}{\partial x} = 1, \frac{\partial q}{\partial y} = 1$$

$$\frac{\partial f}{\partial x} = \frac{\partial f}{\partial q}\frac{\partial q}{\partial x}$$



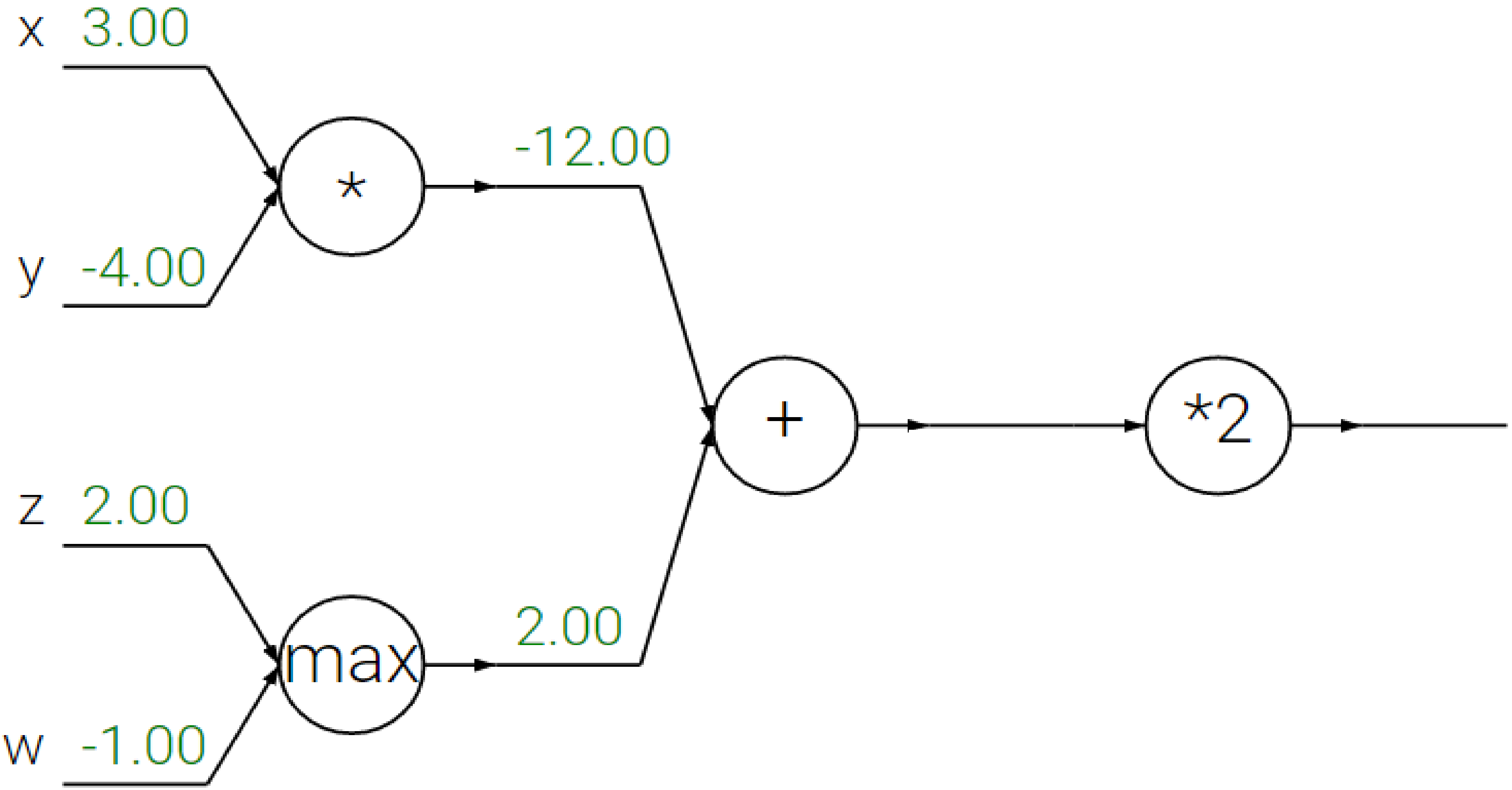
Source: http://cs231n.github.io

FORWARD AND BACKWARD PASS



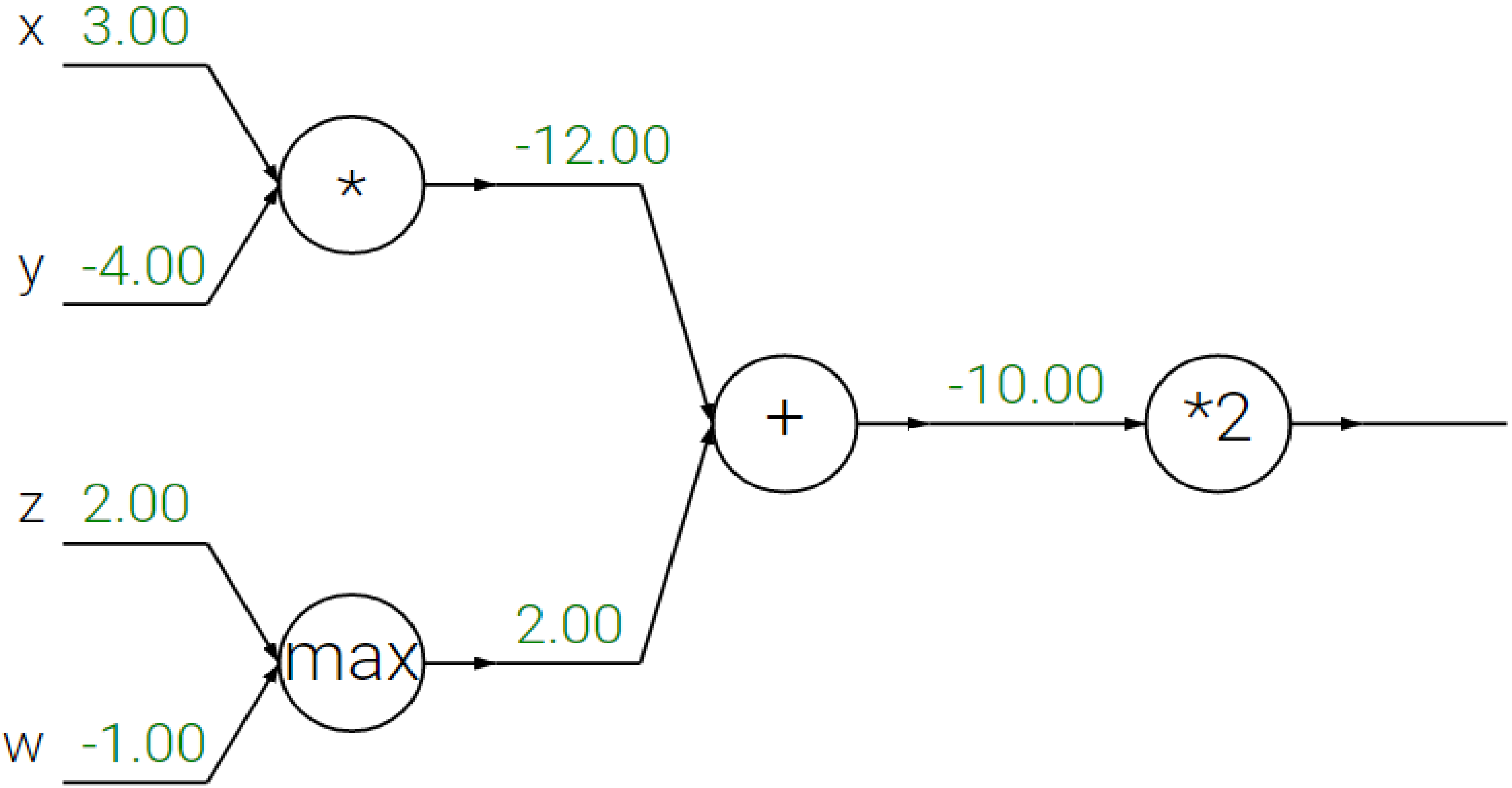
Source: http://cs231n.github.io

FORWARD AND BACKWARD PASS



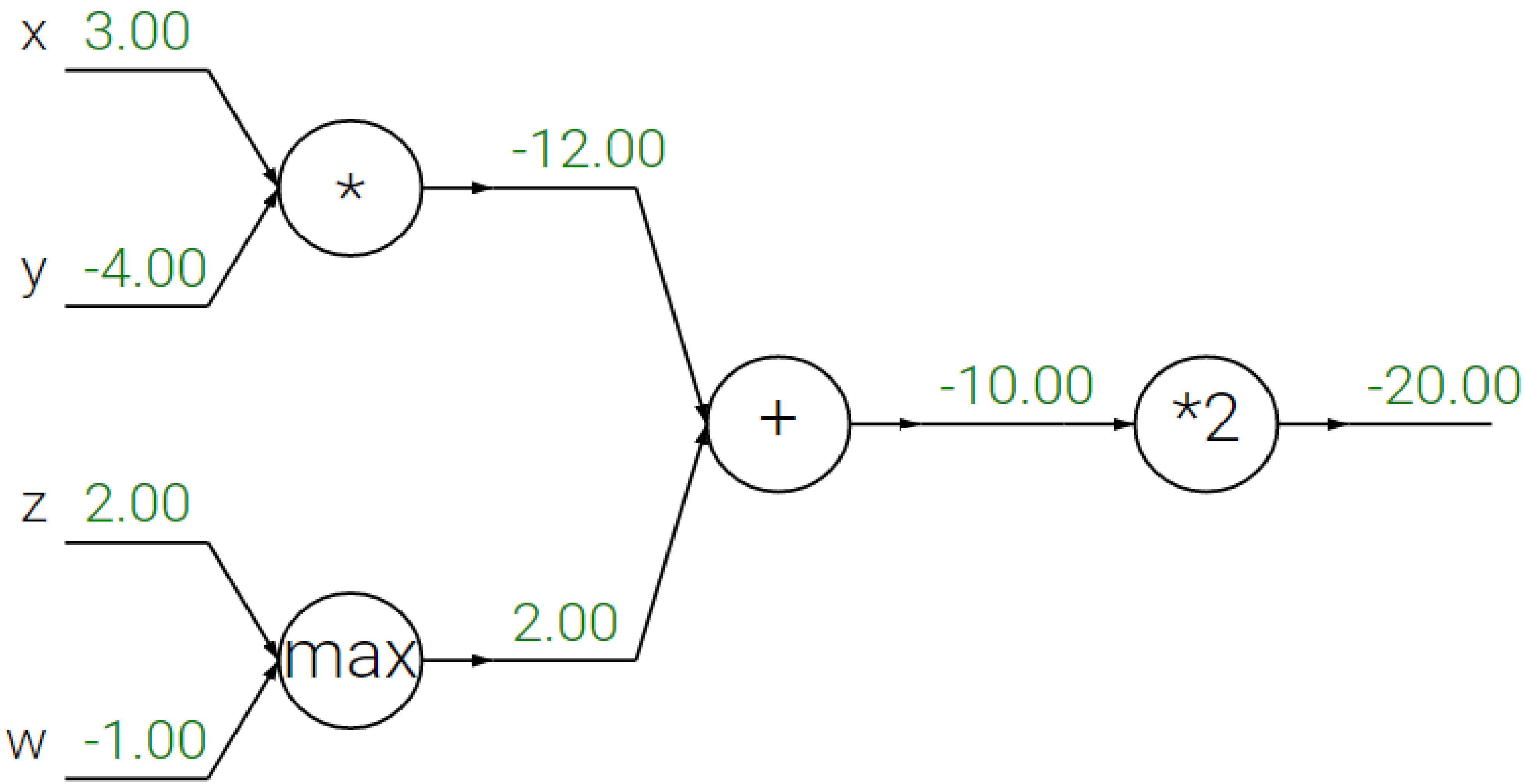
Source: http://cs231n.github.io

FORWARD AND BACKWARD PASS



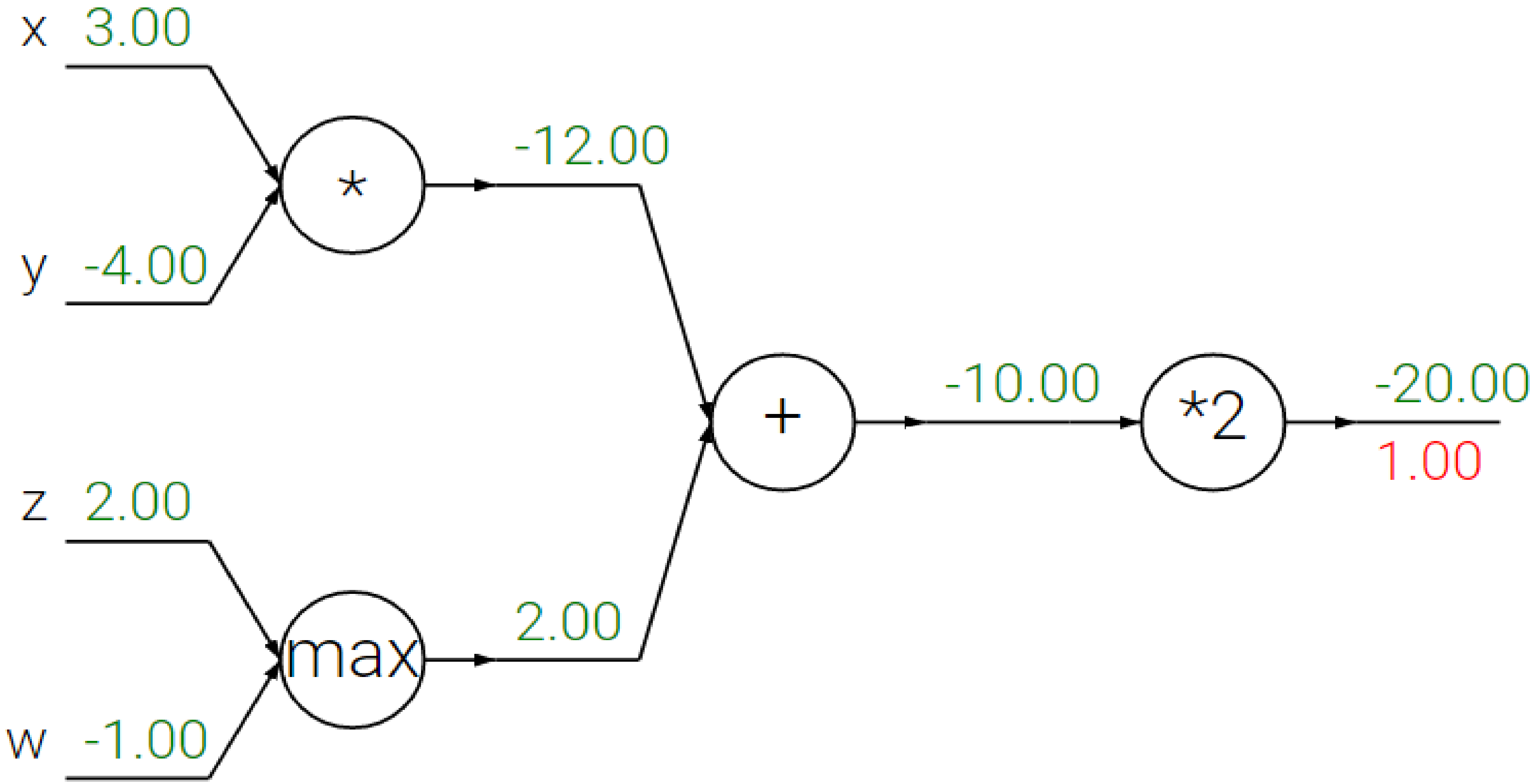
Source: http://cs231n.github.io

FORWARD AND BACKWARD PASS



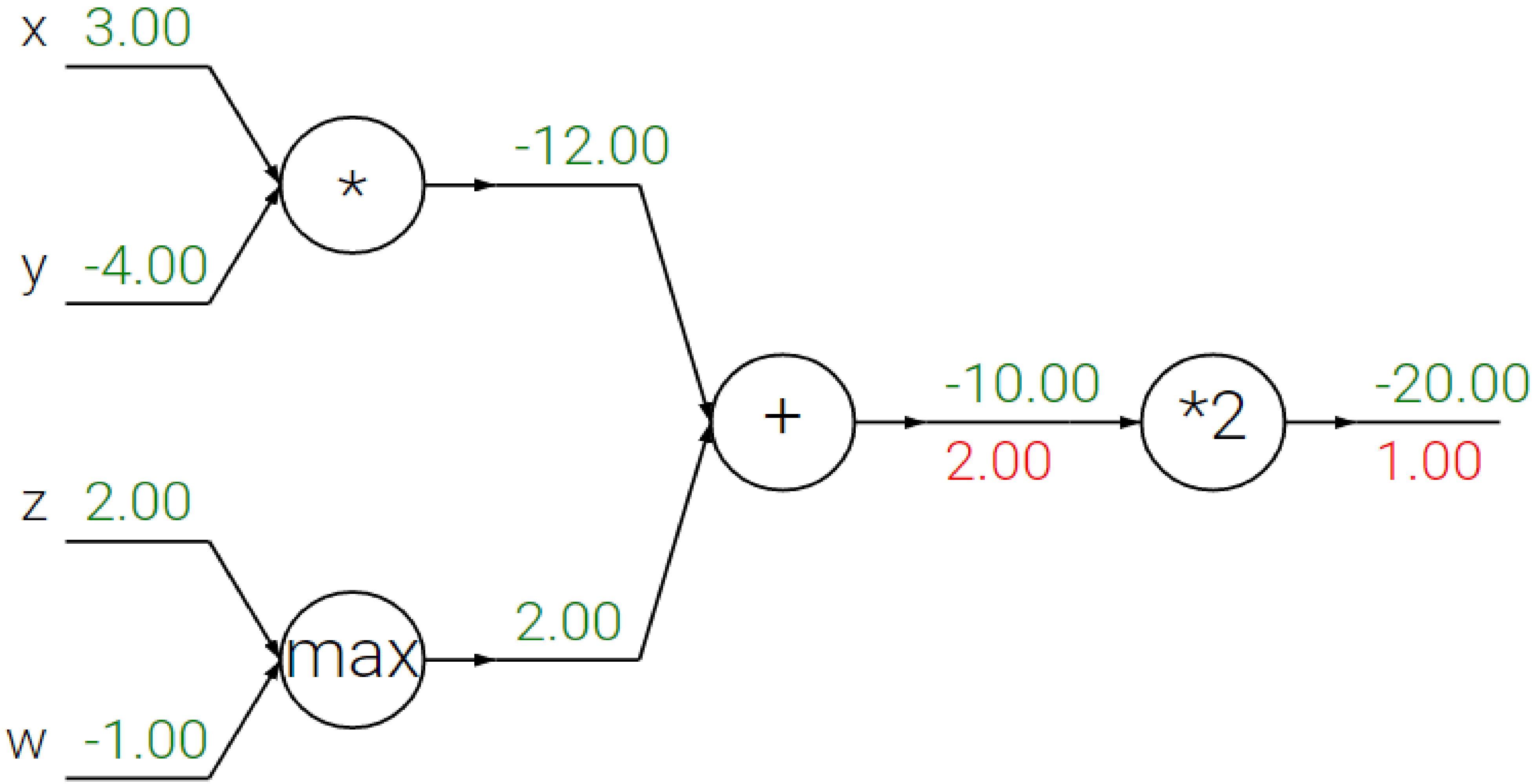
Source: http://cs231n.github.io

FORWARD AND BACKWARD PASS



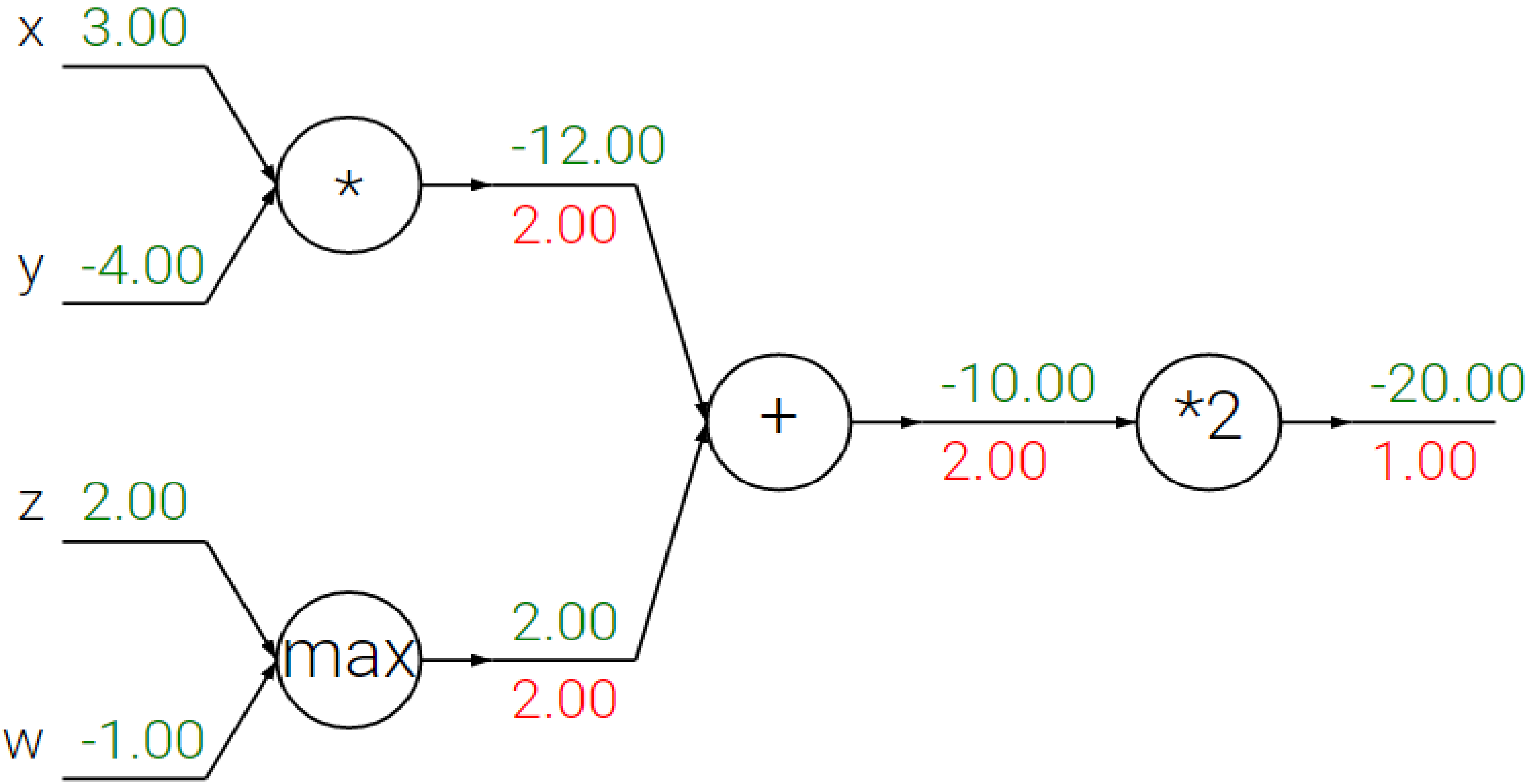
Source: http://cs231n.github.io

FORWARD AND BACKWARD PASS



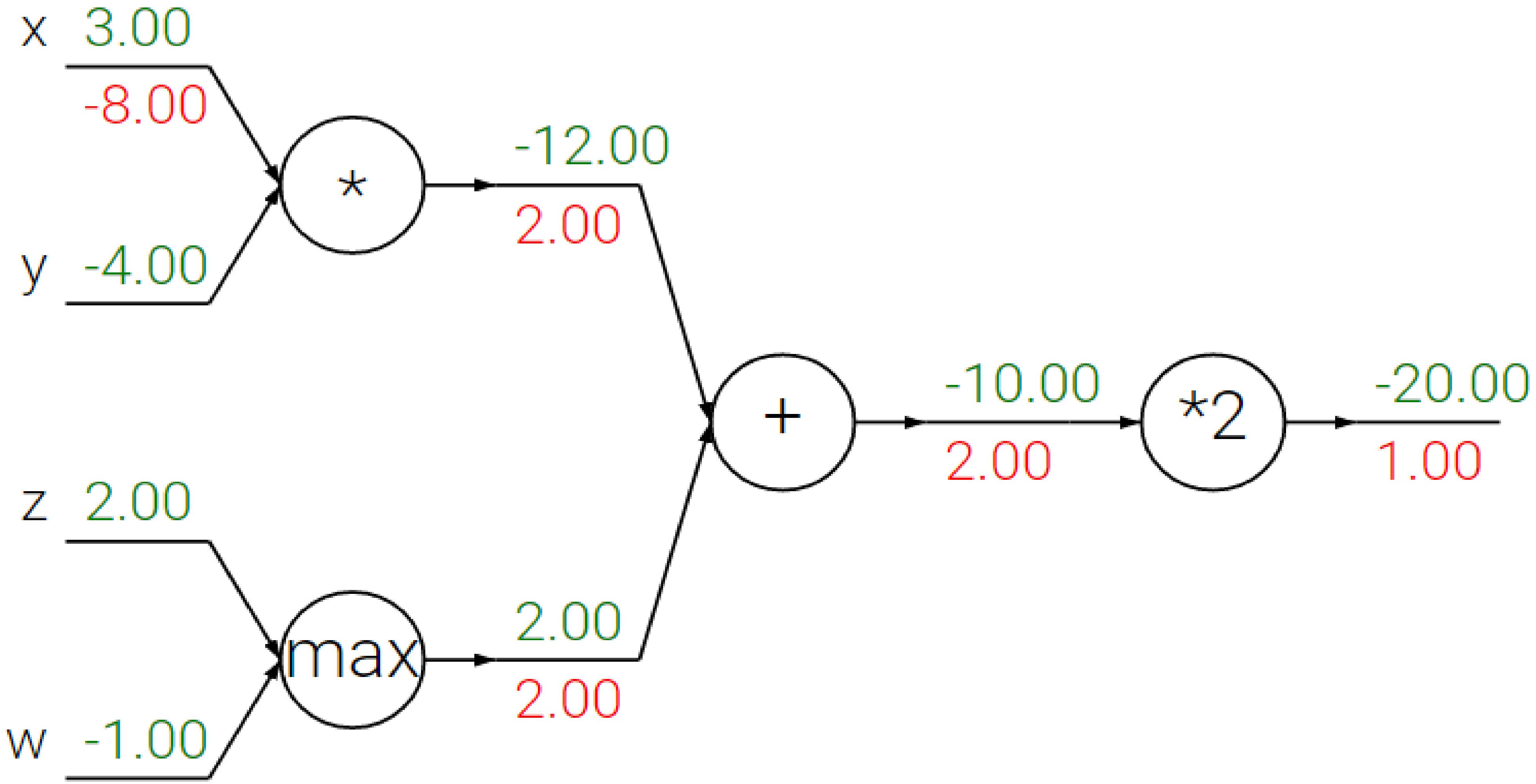
Source: http://cs231n.github.io

FORWARD AND BACKWARD PASS



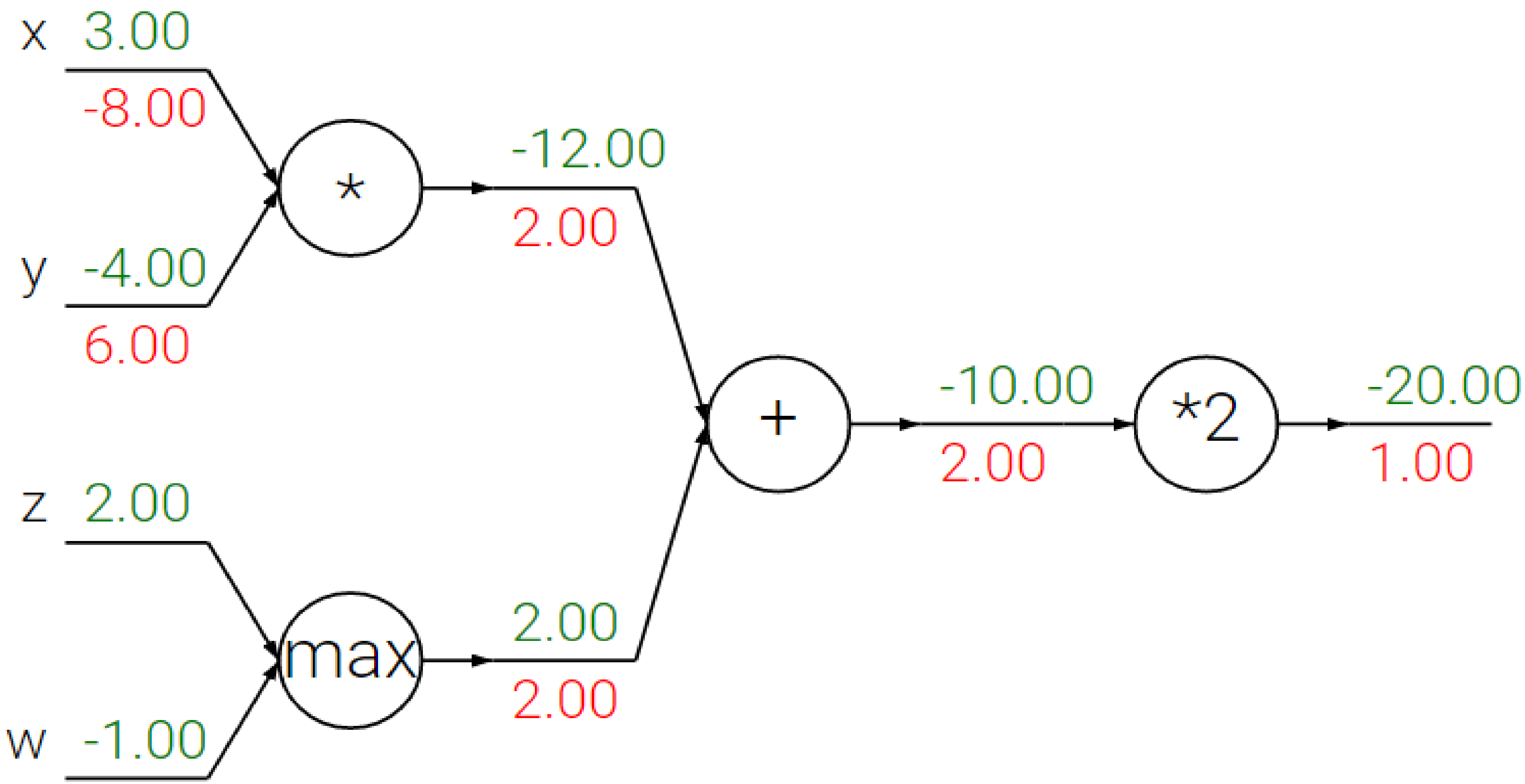
Source: http://cs231n.github.io

FORWARD AND BACKWARD PASS



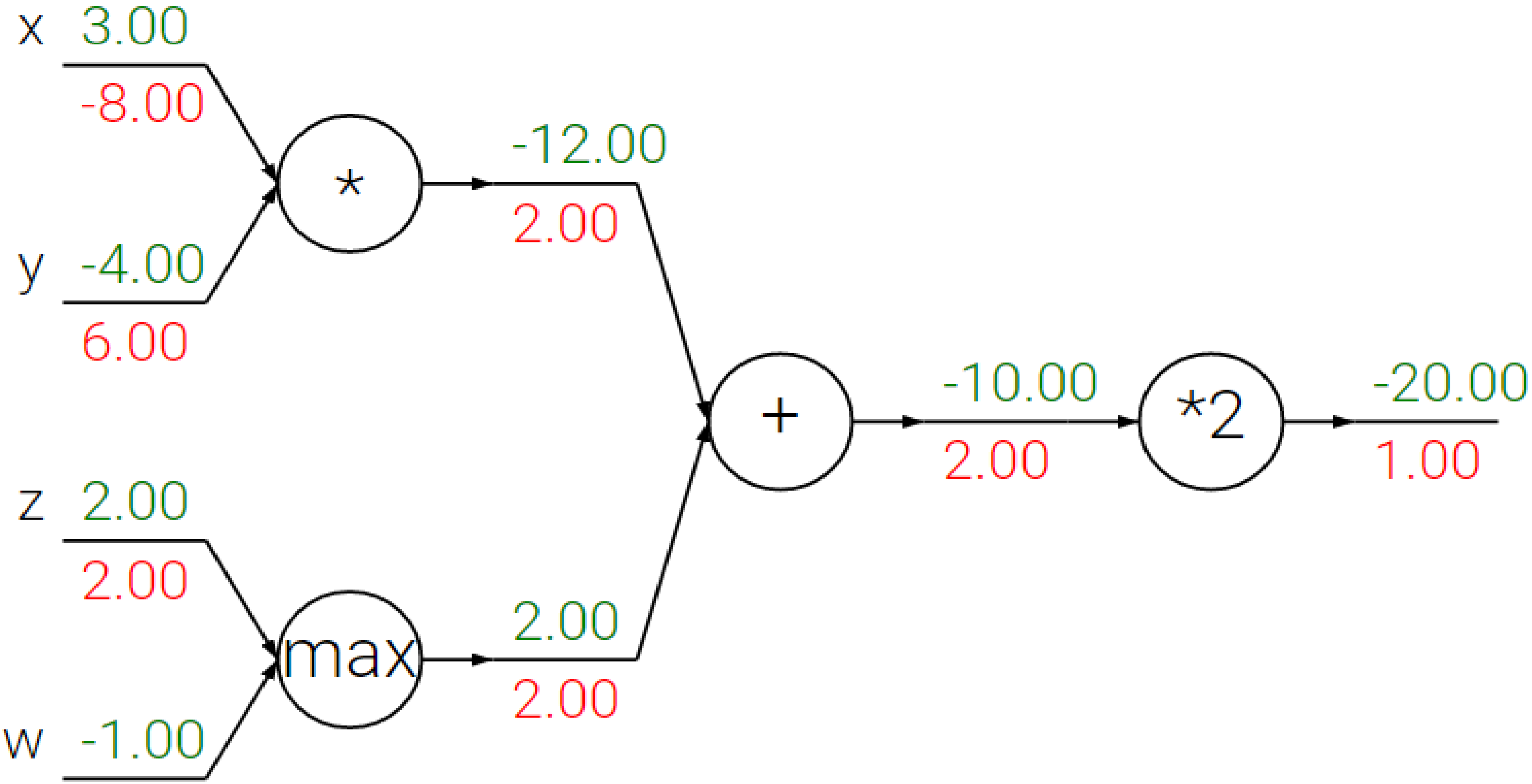
Source: http://cs231n.github.io

FORWARD AND BACKWARD PASS



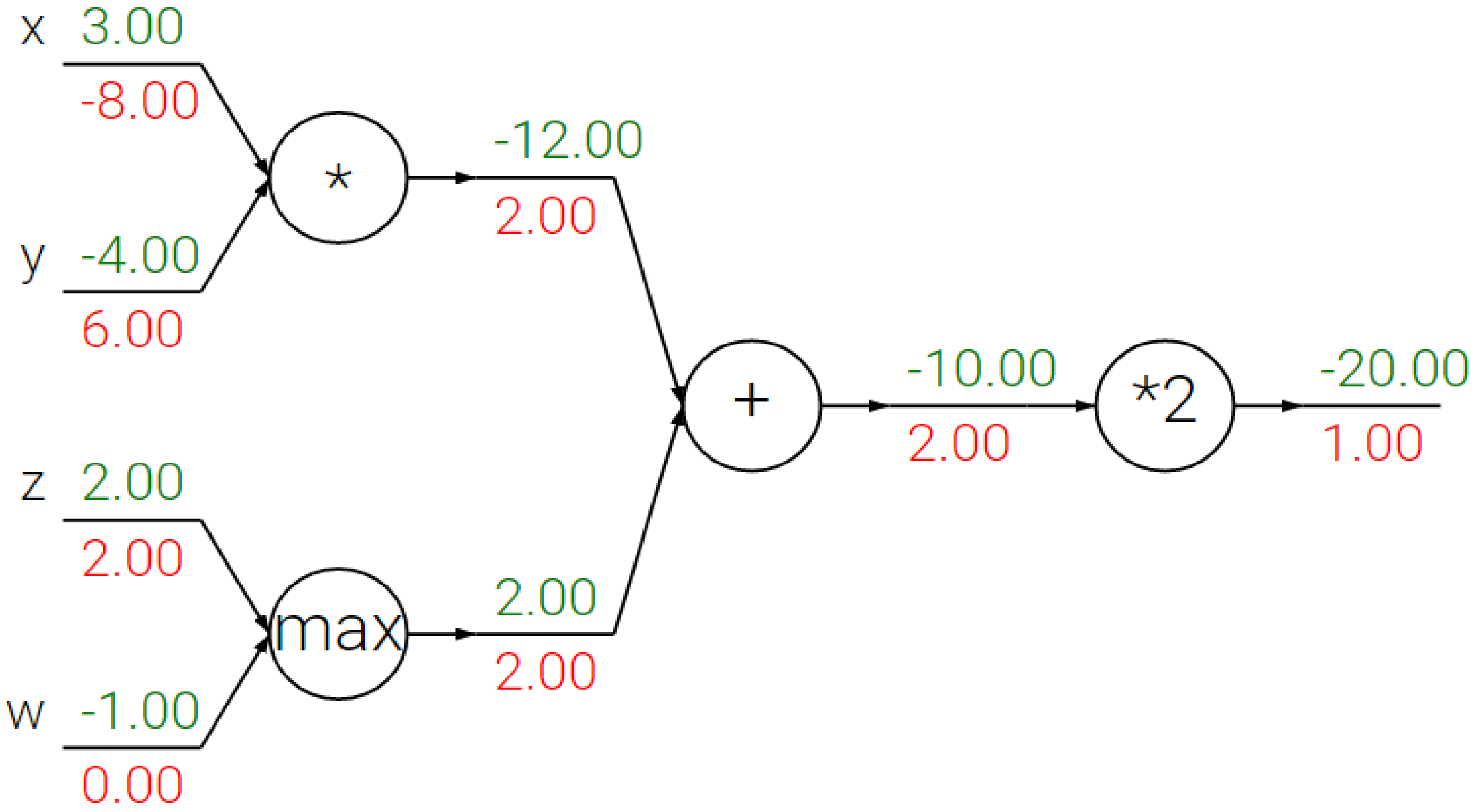
Source: http://cs231n.github.io

FORWARD AND BACKWARD PASS



Source: http://cs231n.github.io

FORWARD AND BACKWARD PASS



Source: http://cs231n.github.io

SIGMOID EXAMPLE

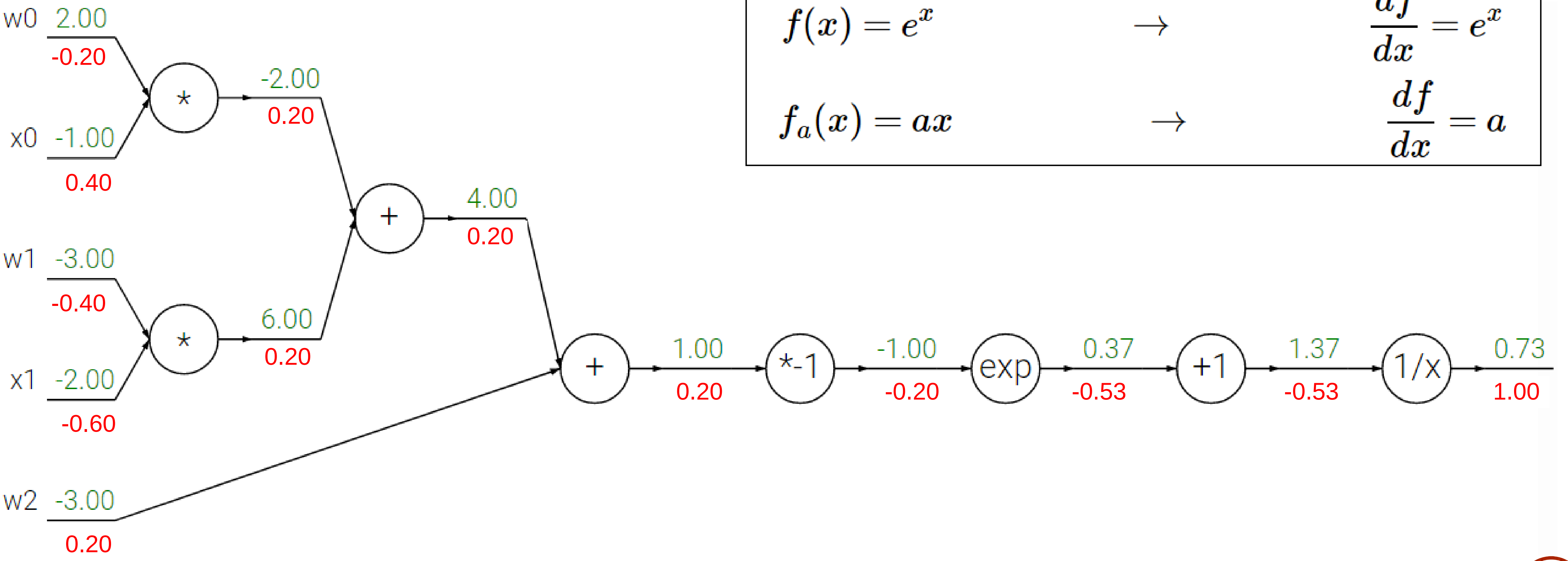
$$f(w,x) = \frac{1}{1+e^{-(w_0x_0+w_1x_1+w_2)}}$$

$$f(x) = \frac{1}{x} \quad \rightarrow \quad \frac{df}{dx} = -1/x^2$$

$$f_c(x) = c + x \quad \rightarrow \quad \frac{df}{dx} = 1$$

$$f(x) = e^x \quad \rightarrow \quad \frac{df}{dx} = e^x$$

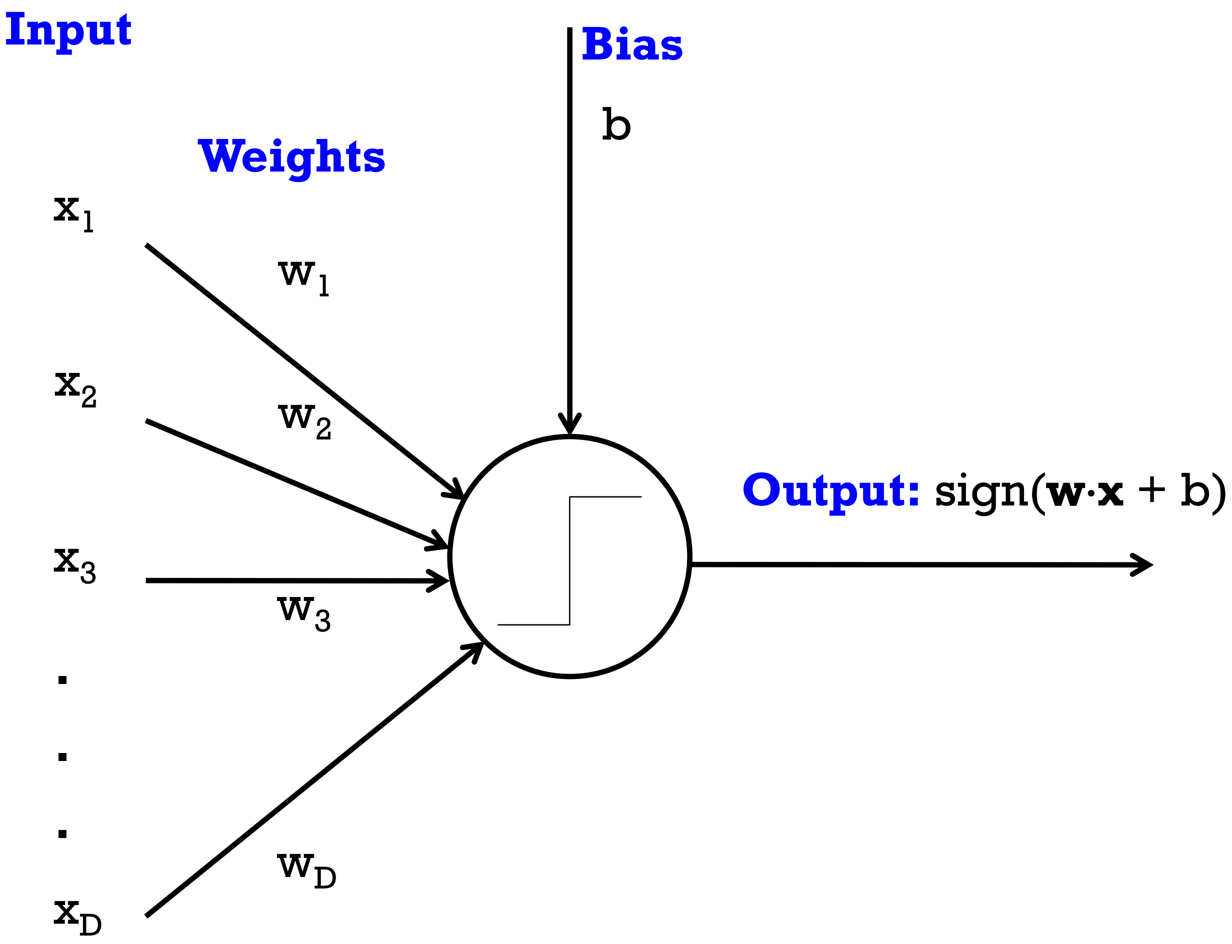
$$f_a(x) = ax \quad \rightarrow \quad \frac{df}{dx} = a$$



Modified from: http://cs231n.github.io

PERCEPTRON

- Supervised learning of binary classifier



Output: sign($\mathbf{w}\cdot\mathbf{x}$ + b)

SINGLE NEURON AS A LINEAR CLASSIFIER

Binary Softmax classifier (*Logistic Regression*) $\sigma(\sum_i w_i x_i + b)$

Source: http://cs231n.github.io

SINGLE NEURON AS A LINEAR CLASSIFIER

Binary Softmax classifier (*Logistic Regression*) $\sigma(\sum_i w_i x_i + b)$

Probability of one of the classes: $P(y_i = 1 \mid x_i; w)$

Source: http://cs231n.github.io

SINGLE NEURON AS A LINEAR CLASSIFIER

Binary Softmax classifier (*Logistic Regression*) $\sigma(\sum_i w_i x_i + b)$

↓

Probability of one of the classes: $P(y_i = 1 \mid x_i; w)$

Probability of the other class would be:

$$P(y_i = 0 \mid x_i; w) = 1 - P(y_i = 1 \mid x_i; w)$$

Source: http://cs231n.github.io

SINGLE NEURON AS A LINEAR CLASSIFIER

Binary Softmax classifier (*Logistic Regression*) $\sigma(\sum_i w_i x_i + b)$

Probability of one of the classes: $P(y_i = 1 \mid x_i; w)$

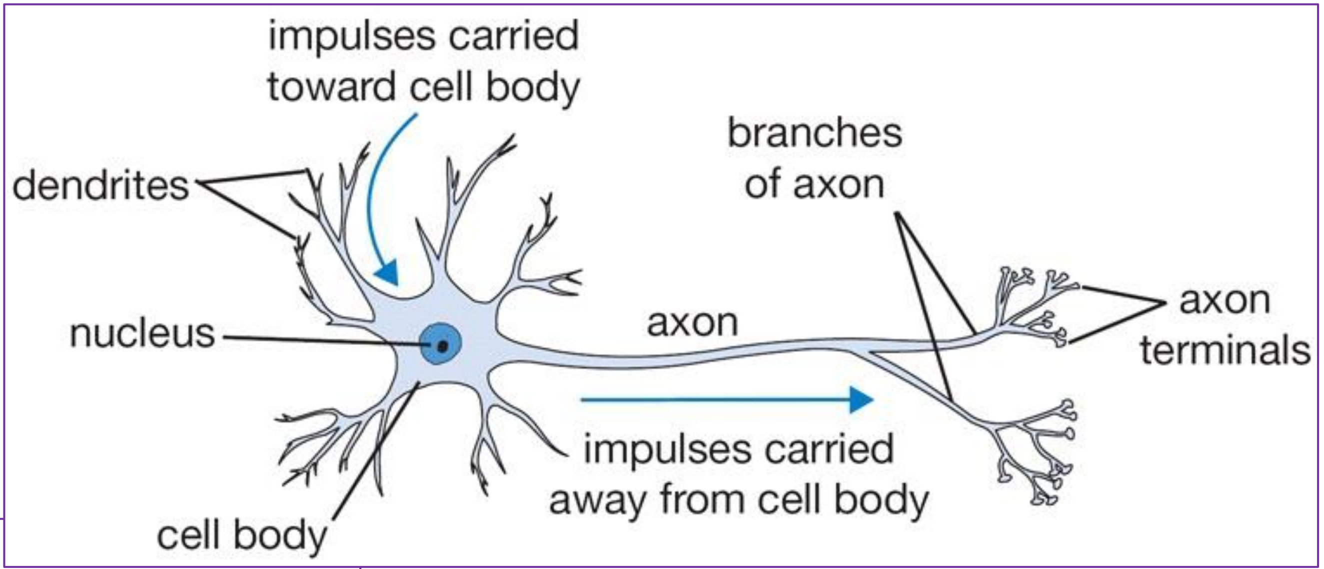
Probability of the other class would be:

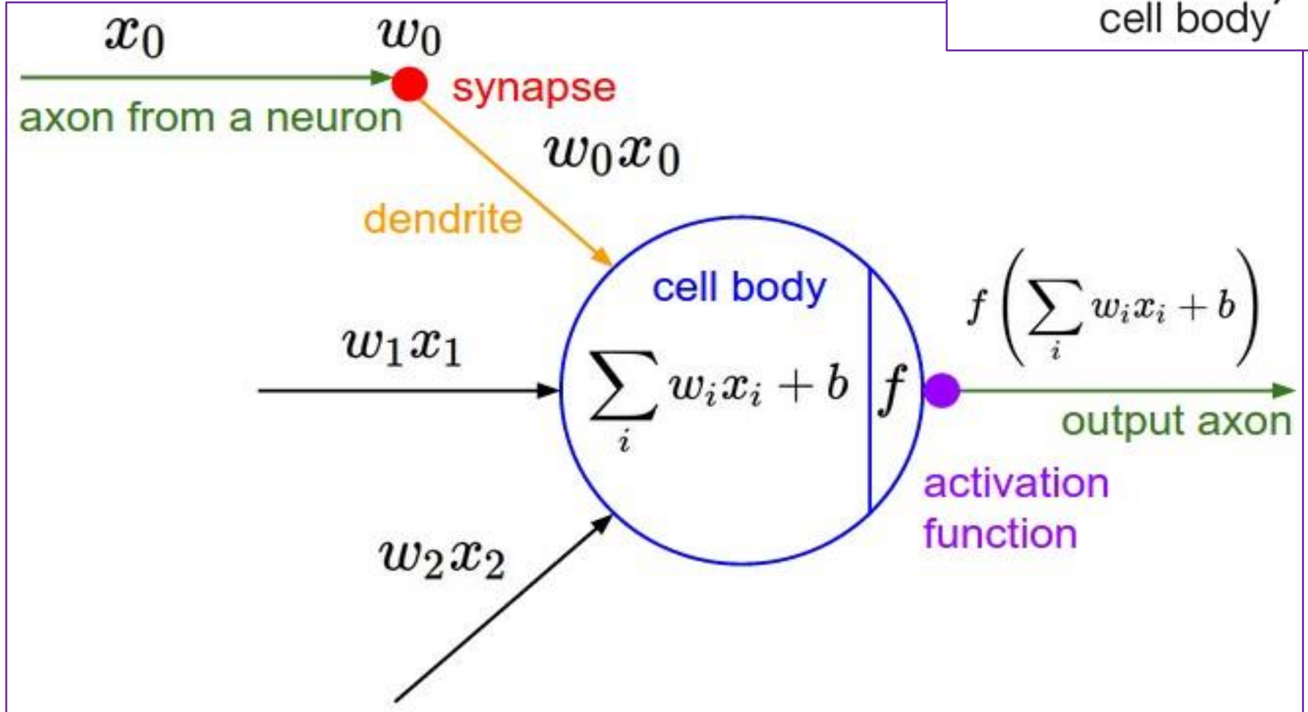
$$P(y_i = 0 \mid x_i; w) = 1 - P(y_i = 1 \mid x_i; w)$$

Binary SVM classifier:
Alternatively, we could attach a max-margin hinge loss to the output of the neuron and train it to become a binary Support Vector Machine.

Source: http://cs231n.github.io

LOOSE INSPIRATION: HUMAN NEURONS

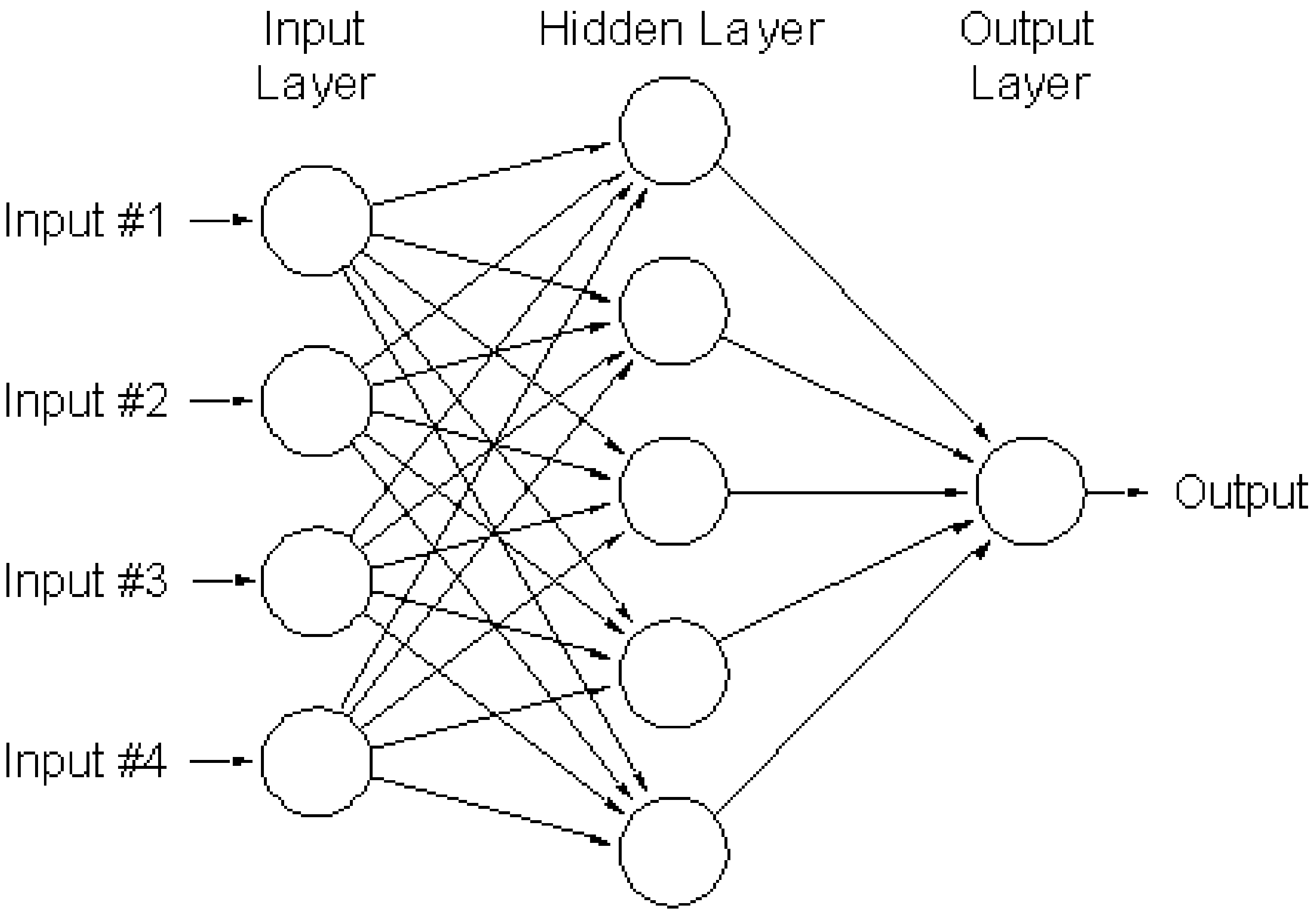




Source: http://cs231n.github.io

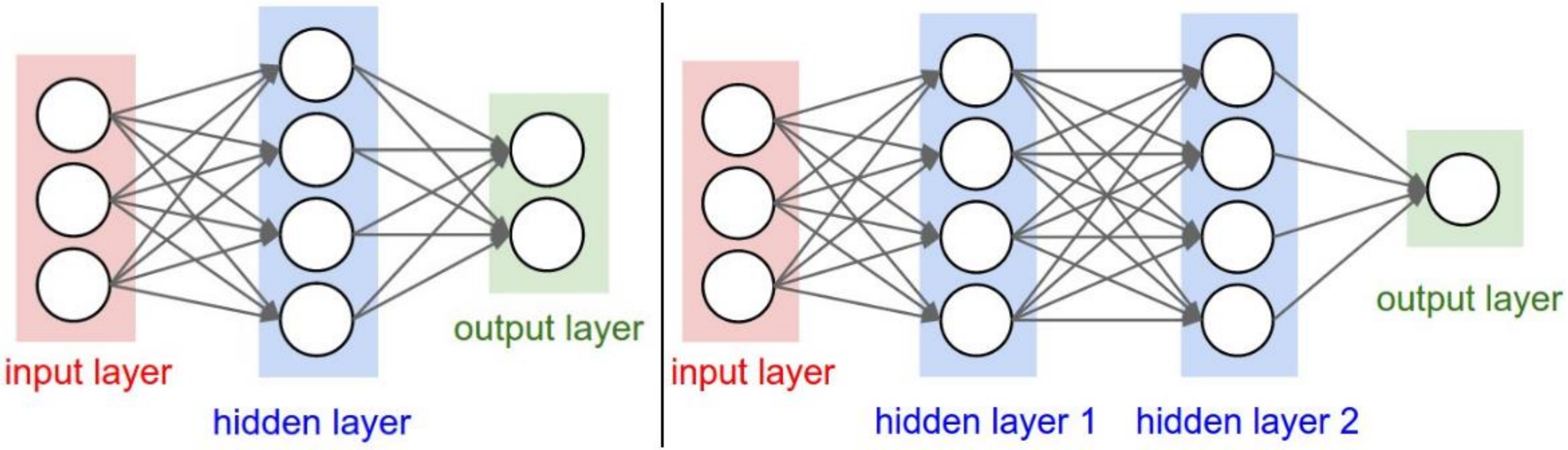
MULTI-LAYER NEURAL NETWORKS

Network with a hidden layer:



Can represent nonlinear functions (provided each perceptron has a nonlinearity)

SIZING NEURAL NETWORKS



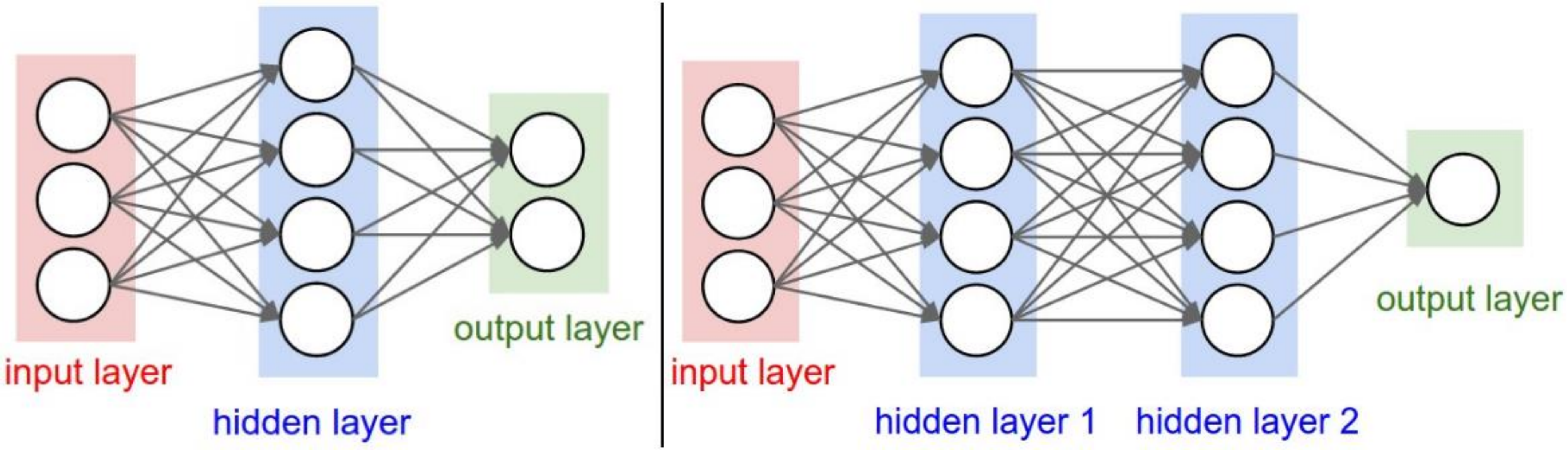
First network (left):

No. of neurons (not counting the inputs):

No. of learnable parameters:

Source: http://cs231n.github.io

SIZING NEURAL NETWORKS



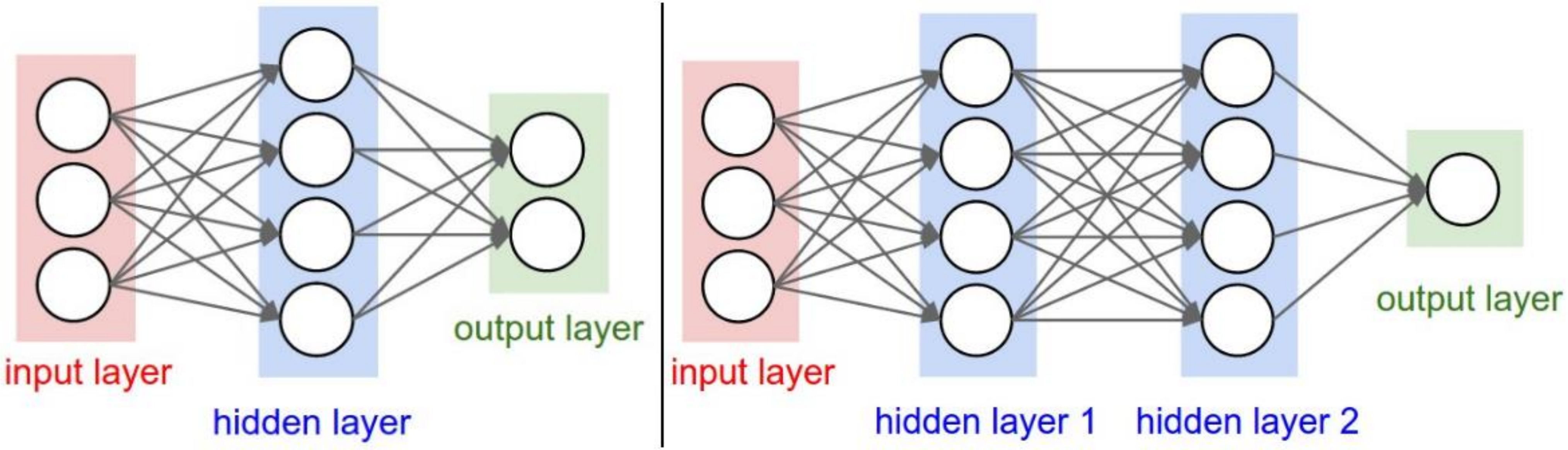
First network (left):

No. of neurons (not counting the inputs): 4 + 2 = 6

No. of learnable parameters:

Source: http://cs231n.github.io

SIZING NEURAL NETWORKS



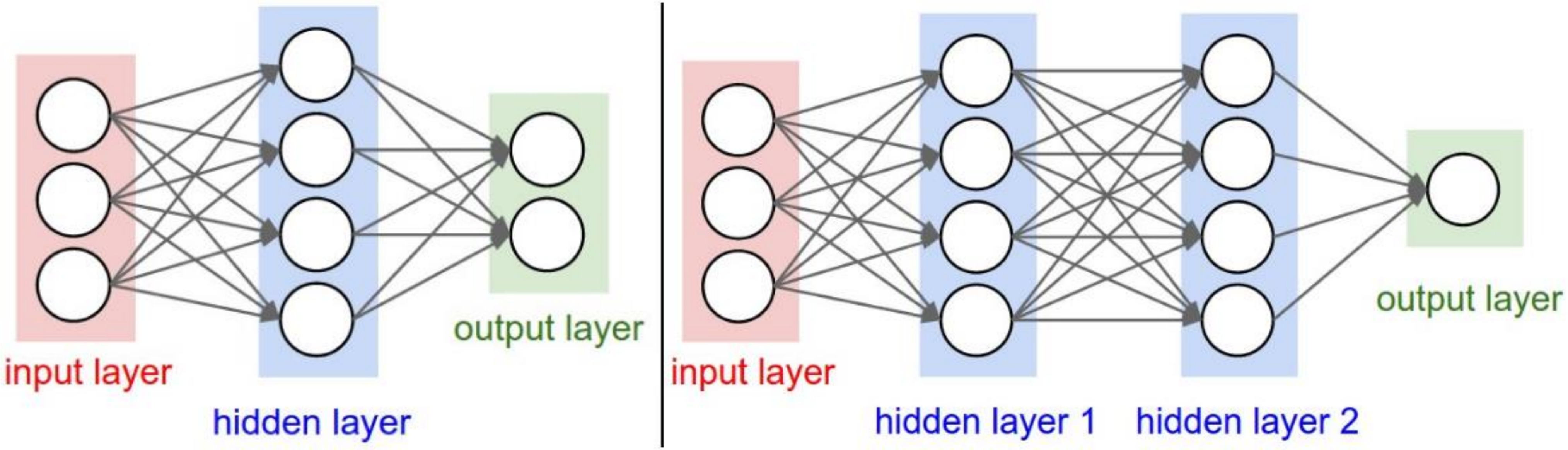
First network (left):

No. of neurons (not counting the inputs): 4 + 2 = 6

No. of learnable parameters: [3 x 4] + [4 x 2] = 20 weights +
4 + 2 = 6 biases = 26.

Source:
http://cs231n.github.io

SIZING NEURAL NETWORKS



First network (left):

No. of neurons (not counting the inputs): 4 + 2 = 6

No. of learnable parameters: [3 x 4] + [4 x 2] = 20 weights +
4 + 2 = 6 biases = 26.

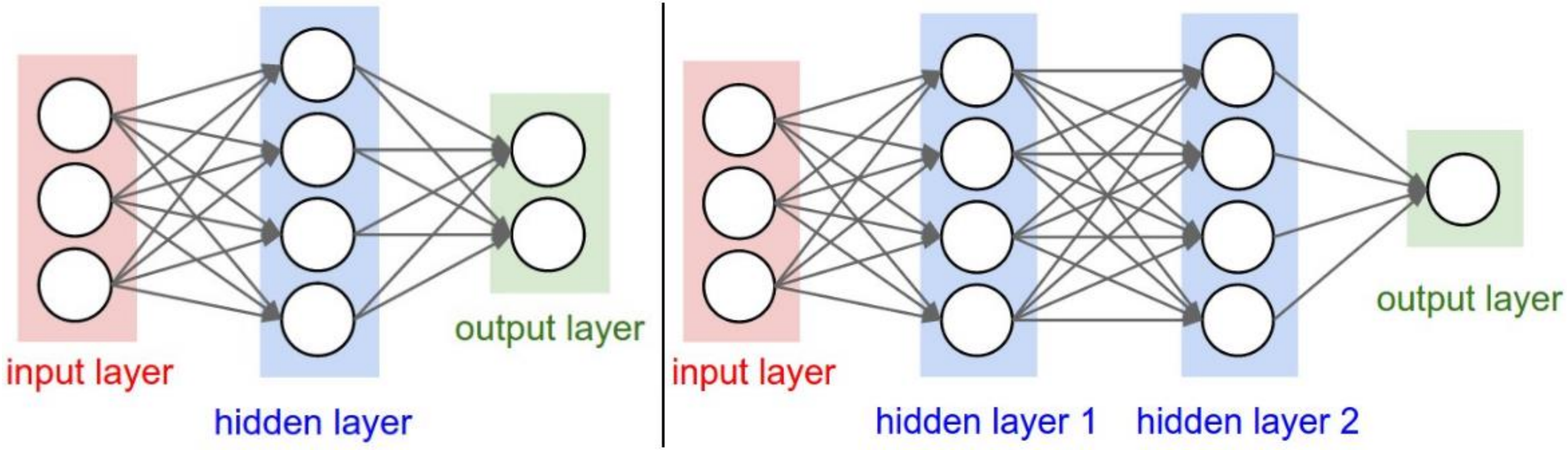
Second network (right):

No. of neurons (not counting the inputs):

No. of learnable parameters:

Source: http://cs231n.github.io

SIZING NEURAL NETWORKS



First network (left):

No. of neurons (not counting the inputs): 4 + 2 = 6

No. of learnable parameters: [3 x 4] + [4 x 2] = 20 weights + 4 + 2 = 6 biases = 26.

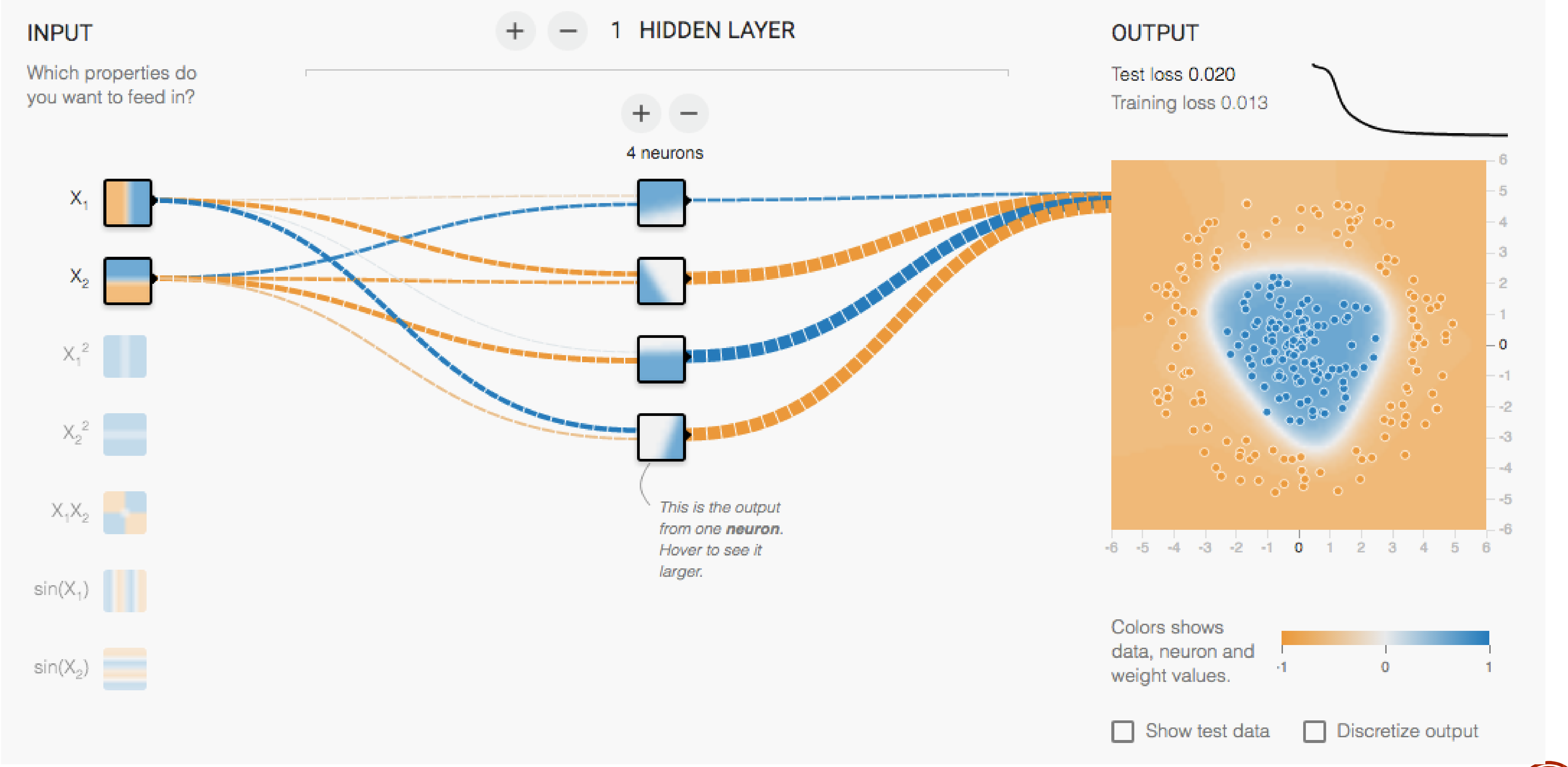
Second network (right):

No. of neurons (not counting the inputs): 4 + 4 + 1 = 9

No. of learnable parameters: [3x4]+[4x4]+[4x1] = 32 weights + 4 + 4 + 1 = 9 biases = 41.

Source: http://cs231n.github.io

MULTI-LAYER NETWORK DEMO



INPUT

Which properties do you want to feed in?

1 HIDDEN LAYER

4 neurons

OUTPUT

Test loss 0.020

Training loss 0.013

This is the output from one ***neuron****. Hover to see it larger.*

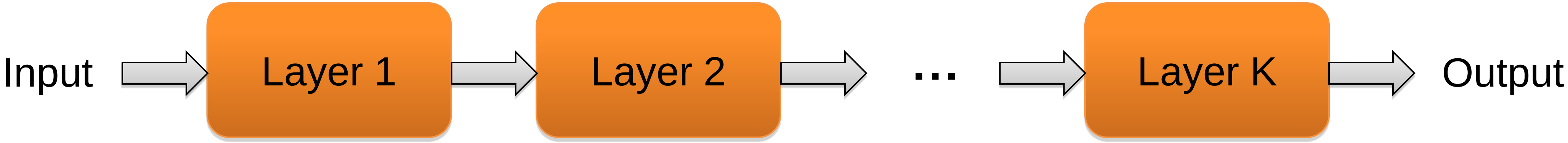
Colors shows data, neuron and weight values.

☐ Show test data ☐ Discretize output

http://playground.tensorflow.org/

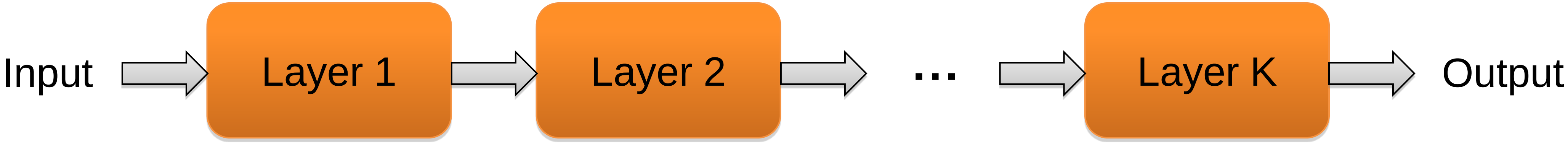
RECALL: MULTI-LAYER NEURAL NETWORKS

- The function computed by the network is a composition of the functions computed by individual layers (e.g., linear layers and nonlinearities):



RECALL: MULTI-LAYER NEURAL NETWORKS

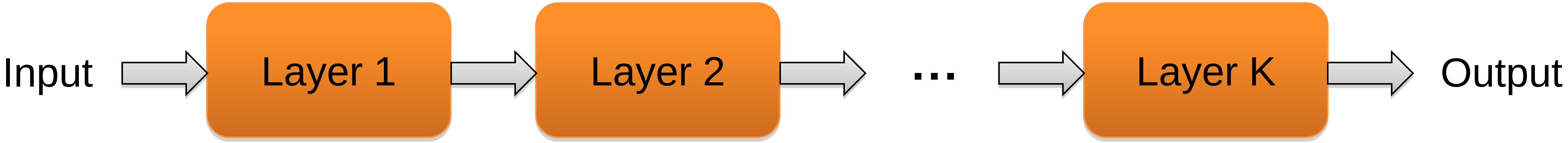
- The function computed by the network is a composition of the functions computed by individual layers (e.g., linear layers and nonlinearities):



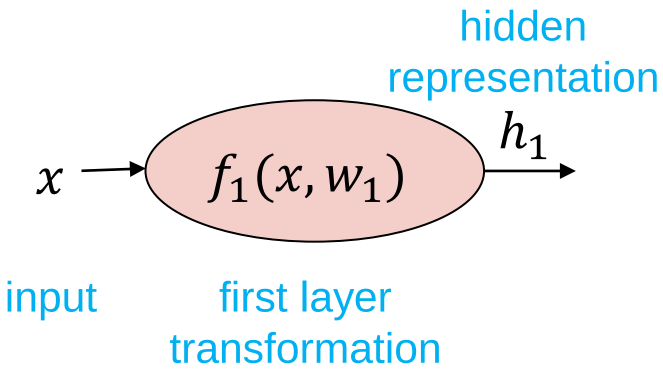
- More precisely:

RECALL: MULTI-LAYER NEURAL NETWORKS

- The function computed by the network is a composition of the functions computed by individual layers (e.g., linear layers and nonlinearities):

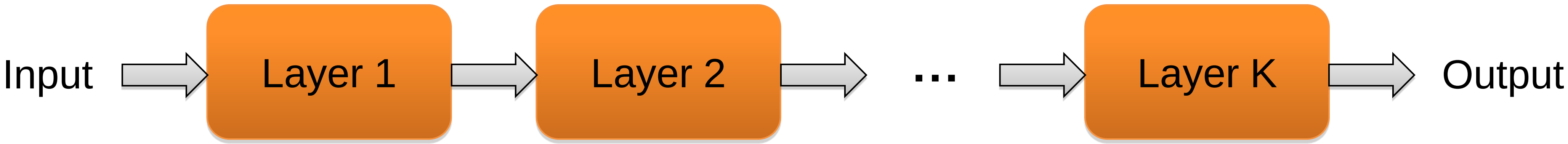


- More precisely:



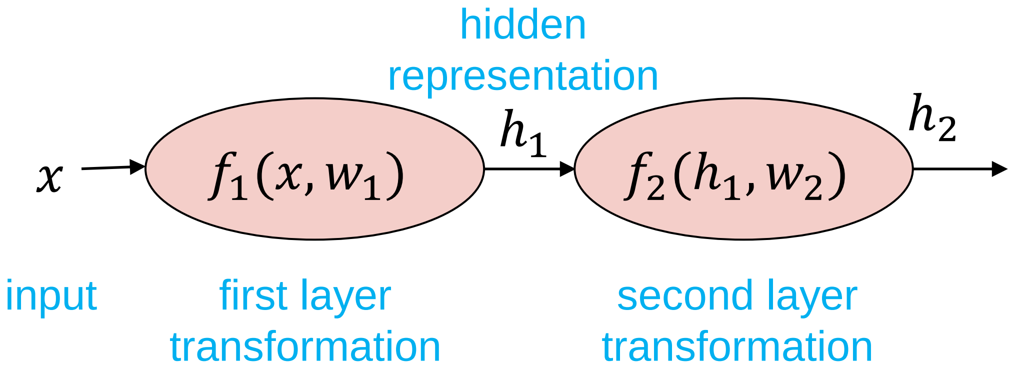
RECALL: MULTI-LAYER NEURAL NETWORKS

- The function computed by the network is a composition of the functions computed by individual layers (e.g., linear layers and nonlinearities):



Input → Layer 1 → Layer 2 → … → Layer K → Output

- More precisely:



$$x \longrightarrow \boxed{f_1(x, w_1)} \xrightarrow{h_1} \boxed{f_2(h_1, w_2)} \xrightarrow{h_2}$$

hidden representation

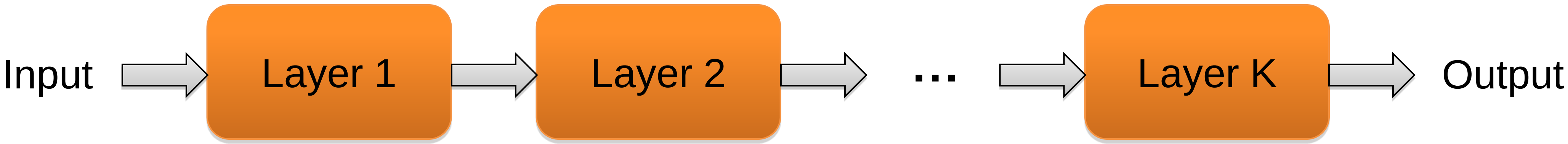
input

first layer transformation

second layer transformation

RECALL: MULTI-LAYER NEURAL NETWORKS

- The function computed by the network is a composition of the functions computed by individual layers (e.g., linear layers and nonlinearities):



Input

Layer 1

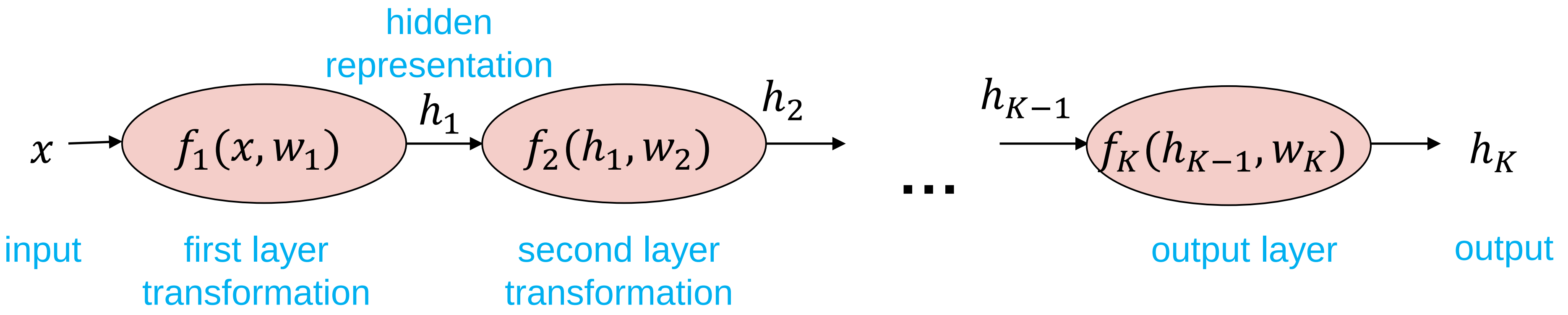
Layer 2

…

Layer K

Output

- More precisely:



hidden representation

x

$f_1(x, w_1)$

h_1

$f_2(h_1, w_2)$

h_2

…

h_{K-1}

$f_K(h_{K-1}, w_K)$

h_K

input

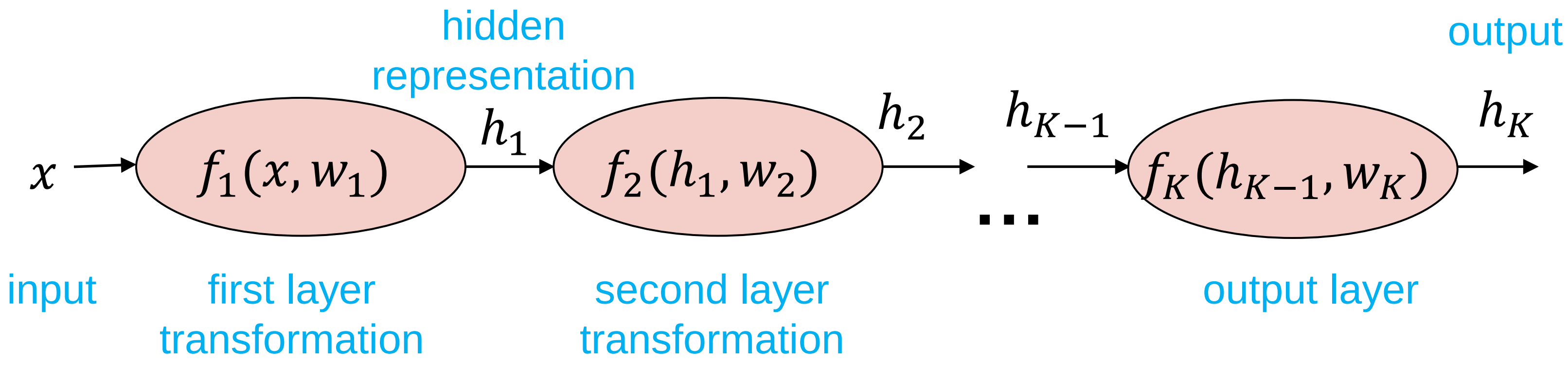
first layer transformation

second layer transformation

output layer

output

TRAINING A MULTI-LAYER NETWORK



hidden representation

output

x $f_1(x, w_1)$ h_1 $f_2(h_1, w_2)$ h_2 … h_{K-1} $f_K(h_{K-1}, w_K)$ h_K

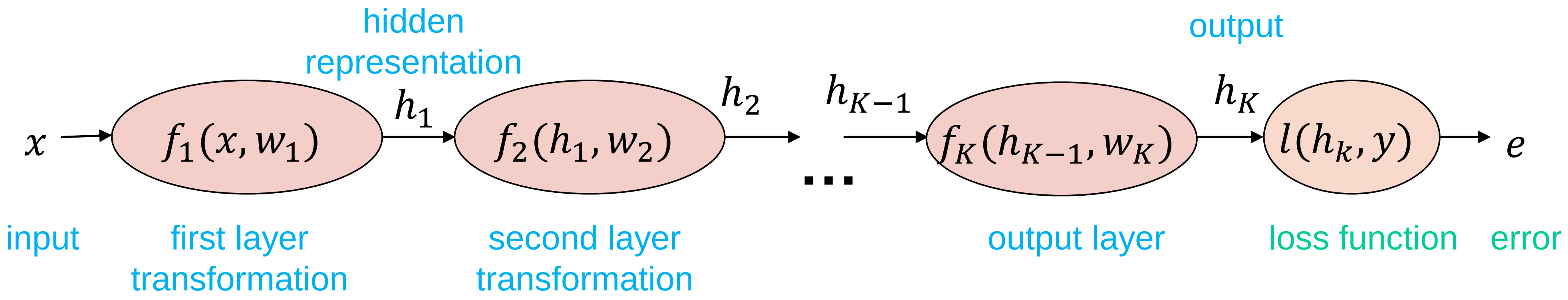
input

first layer transformation

second layer transformation

output layer

TRAINING A MULTI-LAYER NETWORK



hidden representation

output

x → $f_1(x, w_1)$ → h_1 → $f_2(h_1, w_2)$ → h_2 → … → h_{K-1} → $f_K(h_{K-1}, w_K)$ → h_K → $l(h_k, y)$ → e

input

first layer transformation

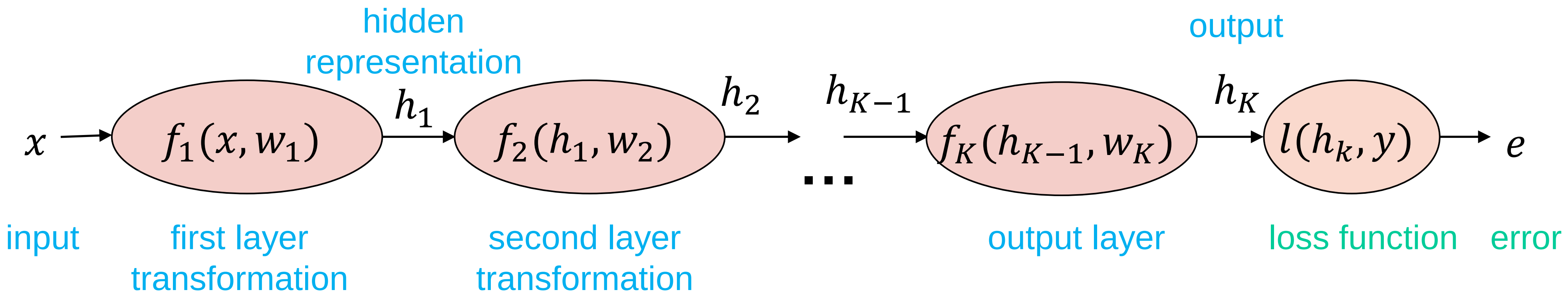
second layer transformation

output layer

loss function

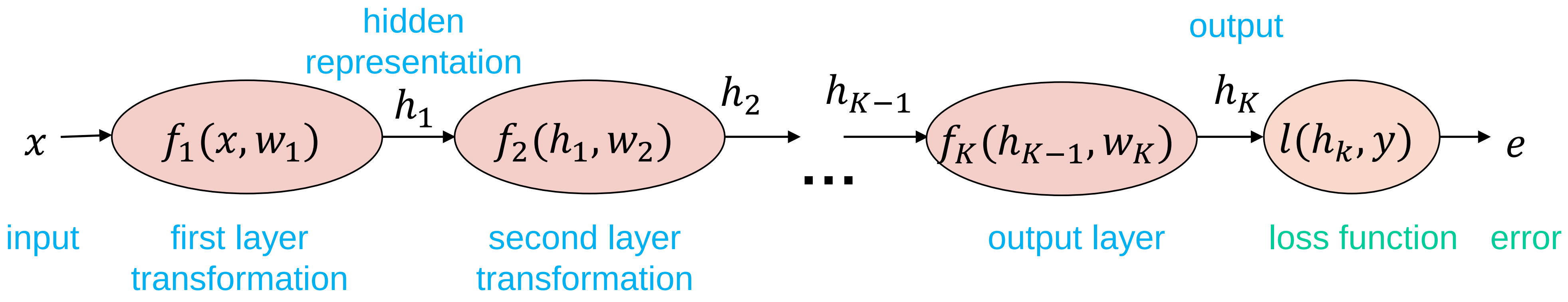
error

TRAINING A MULTI-LAYER NETWORK



- What is the SGD update for the parameters w_k of the kth layer?

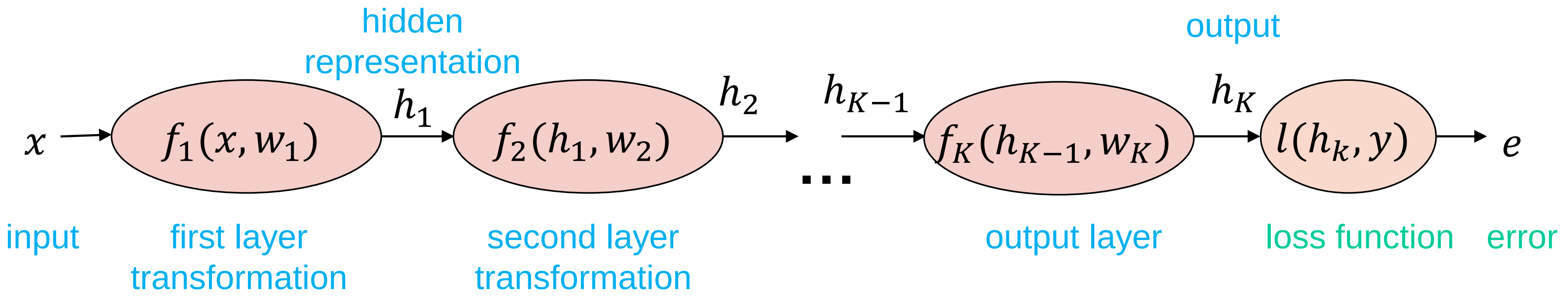
TRAINING A MULTI-LAYER NETWORK



- What is the SGD update for the parameters w_k of the kth layer?

$$w_k \leftarrow w_k - \eta \frac{\partial e}{\partial w_k}$$

TRAINING A MULTI-LAYER NETWORK

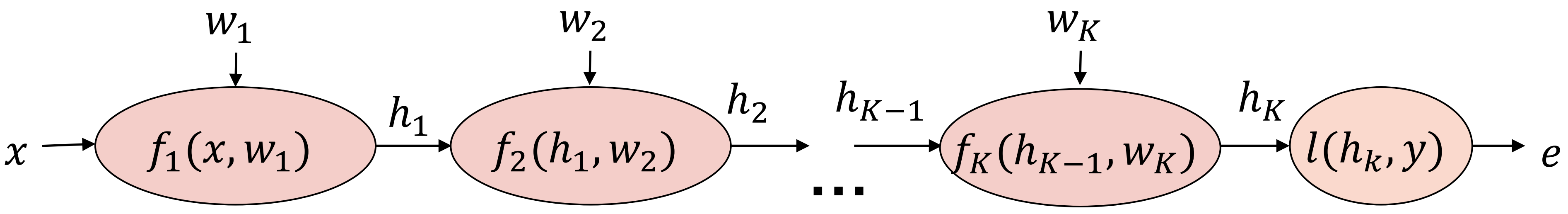


- What is the SGD update for the parameters w_k of the kth layer?

$$w_k \leftarrow w_k - \eta \frac{\partial e}{\partial w_k}$$

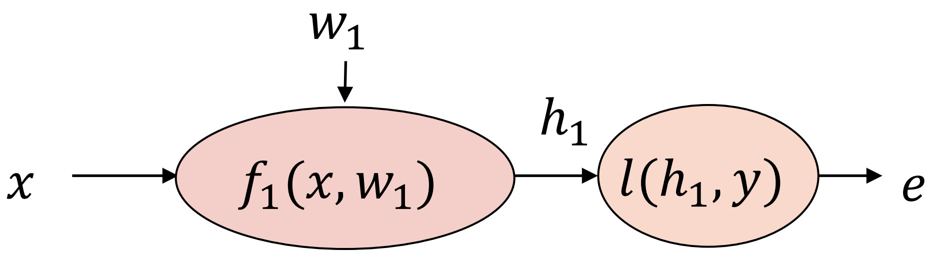
- To train the network, we need to find the gradient of the error w.r.t. the parameters of each layer, $\frac{\partial e}{\partial w_k}$

COMPUTATION GRAPH



CHAIN RULE

Let's start with $k = 1$

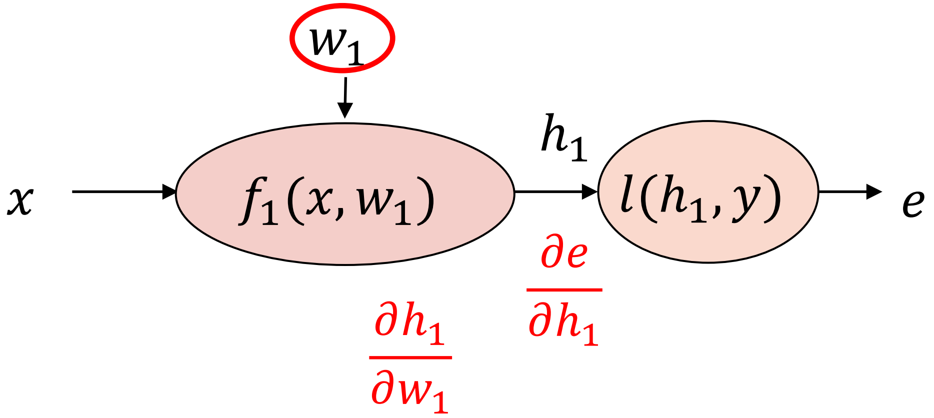


$$e = l(f_1(x, w_1), y)$$

CHAIN RULE

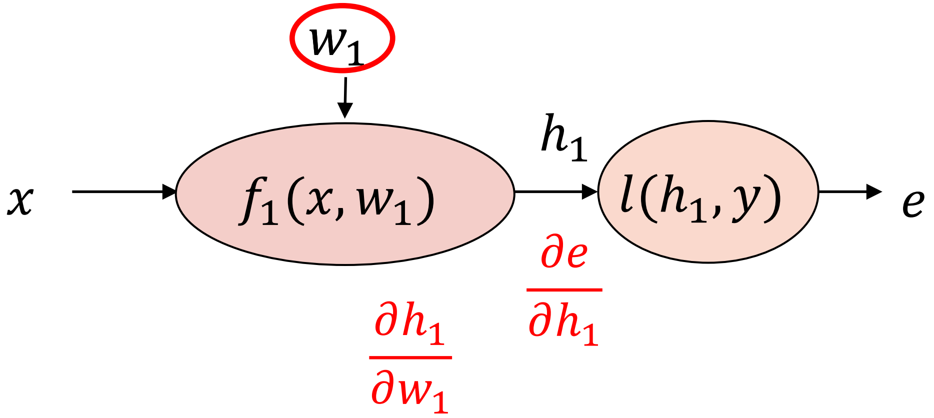
Let's start with $k = 1$

$$e = l(f_1(x, w_1), y)$$



CHAIN RULE

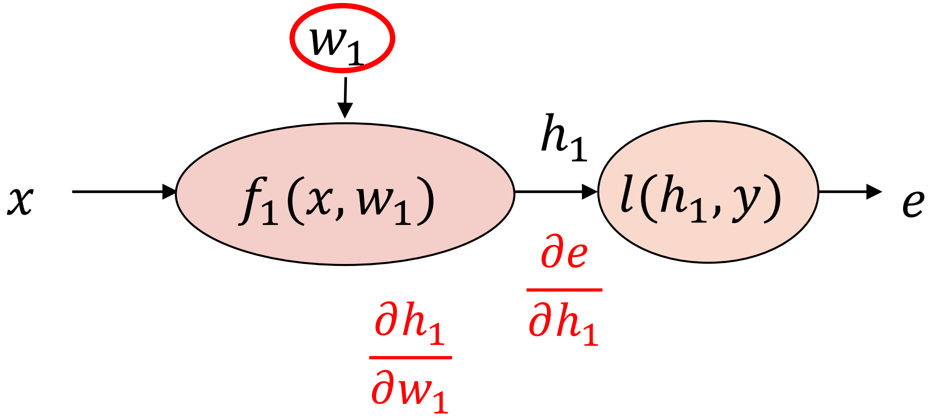
Let's start with $k = 1$



$$e = l(f_1(x, w_1), y)$$

$$\frac{\partial}{\partial w_1} l(f_1(x, w_1), y) = \frac{\partial e}{\partial h_1} \frac{\partial h_1}{\partial w_1}$$

CHAIN RULE



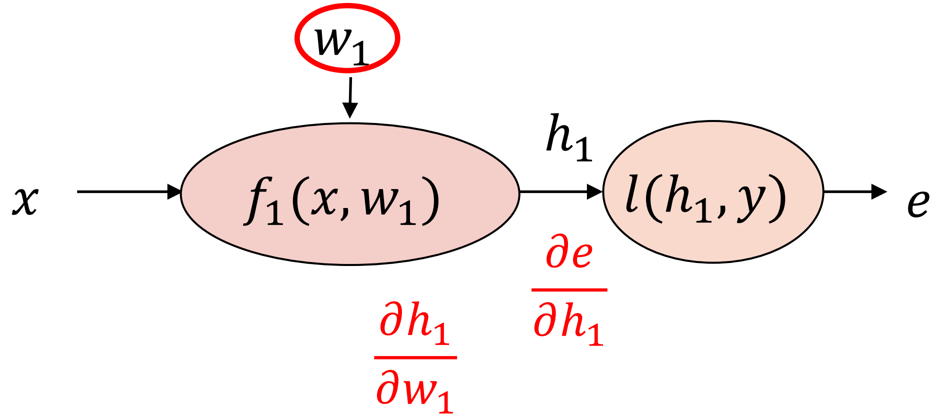
Let’s start with $k = 1$

$$e = l(f_1(x, w_1), y)$$

$$\frac{\partial}{\partial w_1} l(f_1(x, w_1), y) = \frac{\partial e}{\partial h_1} \frac{\partial h_1}{\partial w_1}$$

Example: $e = (y - w_1^T x)^2$

CHAIN RULE



Let’s start with $k = 1$

$$e = l(f_1(x, w_1), y)$$

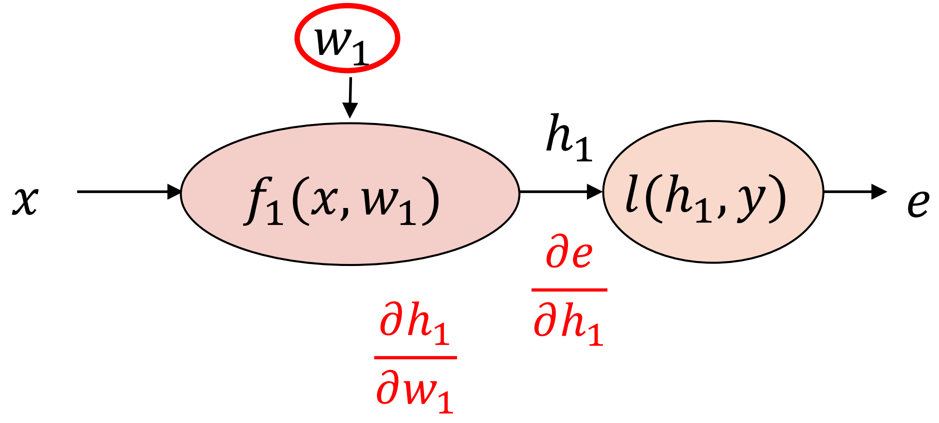
$$\frac{\partial}{\partial w_1} l(f_1(x, w_1), y) = \frac{\partial e}{\partial h_1} \frac{\partial h_1}{\partial w_1}$$

Example: $e = (y - w_1^T x)^2$

$$h_1 = f_1(x, w_1) = w_1^T x$$

$$e = l(h_1, y) = (y - h_1)^2$$

CHAIN RULE



Let's start with $k = 1$

$$e = l(f_1(x, w_1), y)$$

$$\frac{\partial}{\partial w_1} l(f_1(x, w_1), y) = \frac{\partial e}{\partial h_1} \frac{\partial h_1}{\partial w_1}$$

Example: $e = (y - w_1^T x)^2$

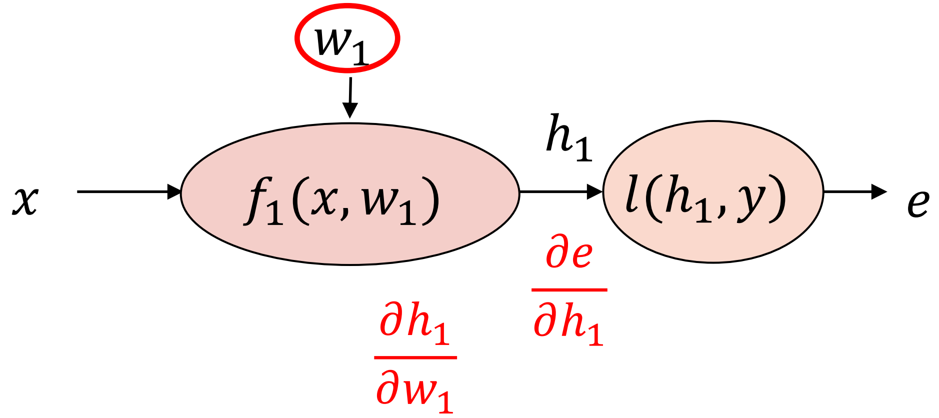
$$h_1 = f_1(x, w_1) = w_1^T x$$

$$e = l(h_1, y) = (y - h_1)^2$$

$$\frac{\partial h_1}{\partial w_1} = ??????$$

$$\frac{\partial e}{\partial h_1} = ??????$$

CHAIN RULE



Let's start with $k = 1$

$$e = l(f_1(x, w_1), y)$$

$$\frac{\partial}{\partial w_1} l(f_1(x, w_1), y) = \frac{\partial e}{\partial h_1} \frac{\partial h_1}{\partial w_1}$$

Example: $e = (y - w_1^T x)^2$

$$h_1 = f_1(x, w_1) = w_1^T x$$

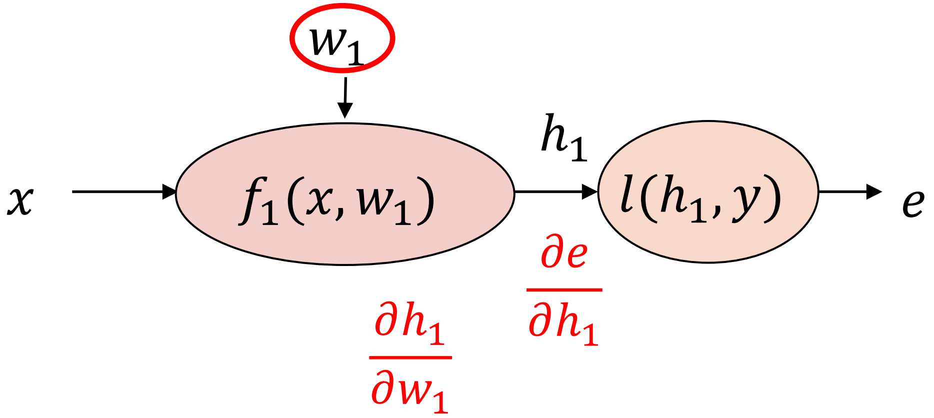
$$e = l(h_1, y) = (y - h_1)^2$$

$$\frac{\partial h_1}{\partial w_1} = x$$

$$\frac{\partial e}{\partial h_1} = -2(y - h_1) = -2(y - w_1^T x)$$

CHAIN RULE

Let's start with $k = 1$



$$e = l(f_1(x, w_1), y)$$

$$\frac{\partial}{\partial w_1} l(f_1(x, w_1), y) = \frac{\partial e}{\partial h_1} \frac{\partial h_1}{\partial w_1}$$

Example: $e = (y - w_1^T x)^2$

$$h_1 = f_1(x, w_1) = w_1^T x$$

$$e = l(h_1, y) = (y - h_1)^2$$

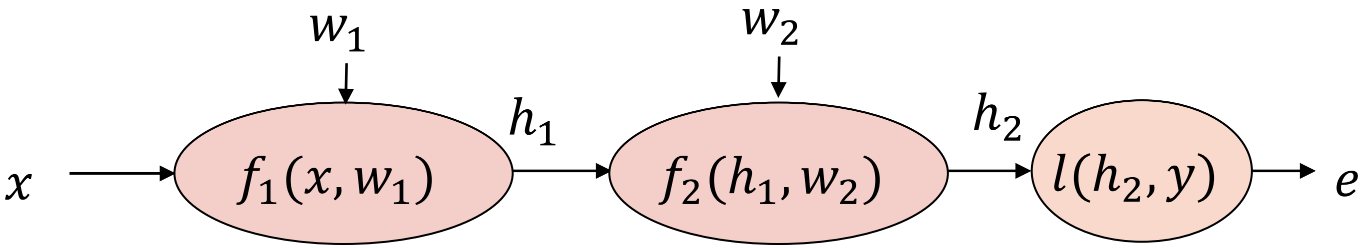
$$\frac{\partial h_1}{\partial w_1} = x$$

$$\frac{\partial e}{\partial h_1} = -2(y - h_1) = -2(y - w_1^T x)$$

$$\frac{\partial e}{\partial w_1} = \frac{\partial e}{\partial h_1} \frac{\partial h_1}{\partial w_1} = -2x(y - w_1^T x)$$

CHAIN RULE

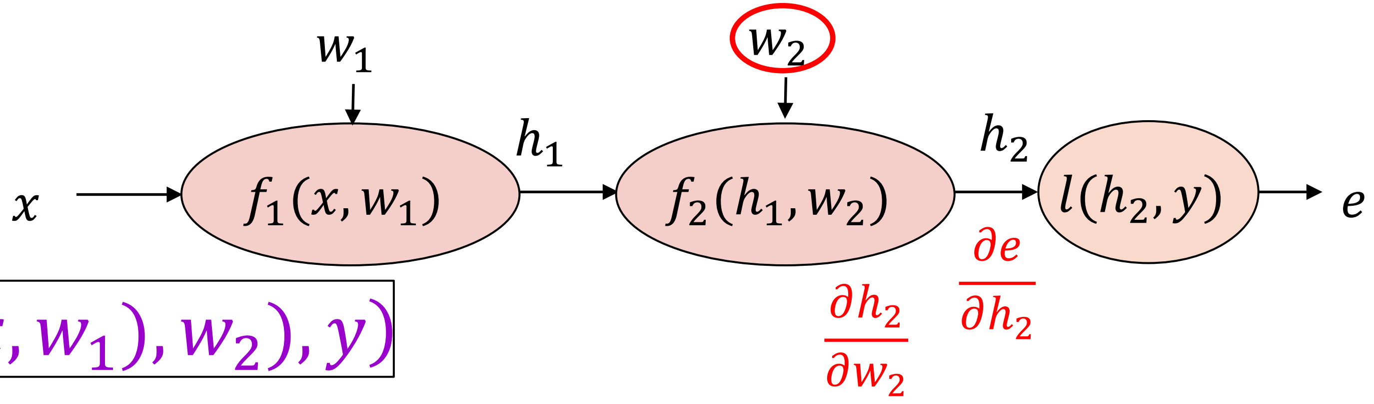
$k = 2$



$$\boxed{e = l(f_2(f_1(x, w_1), w_2), y)}$$

CHAIN RULE

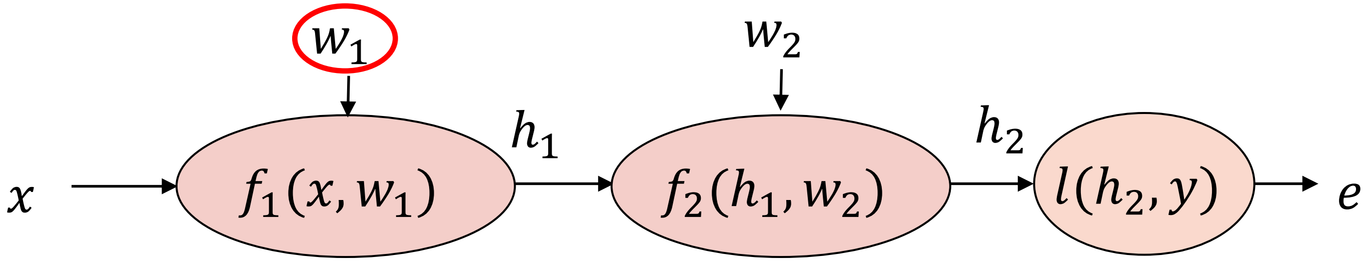
$k = 2$



$$\boxed{e = l(f_2(f_1(x, w_1), w_2), y)}$$

$$\frac{\partial e}{\partial w_2} = \frac{\partial e}{\partial h_2} \frac{\partial h_2}{\partial w_2}$$

CHAIN RULE

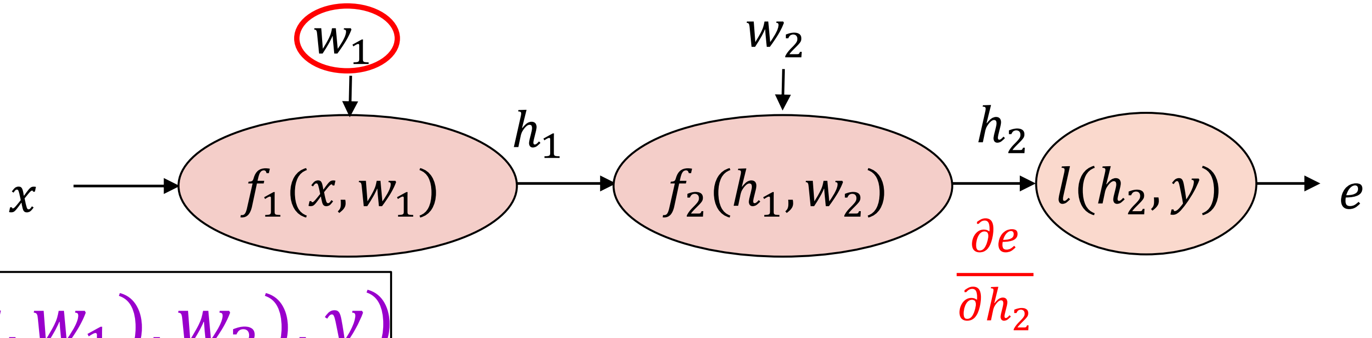

w_1
w_2
x
$f_1(x, w_1)$
h_1
$f_2(h_1, w_2)$
h_2
$l(h_2, y)$
e

$k = 2$

$$\boxed{e = l(f_2(f_1(x, w_1), w_2), y)}$$

$$\frac{\partial e}{\partial w_2} = \frac{\partial e}{\partial h_2}\frac{\partial h_2}{\partial w_2} \qquad \frac{\partial e}{\partial w_1} =$$

CHAIN RULE



$k = 2$

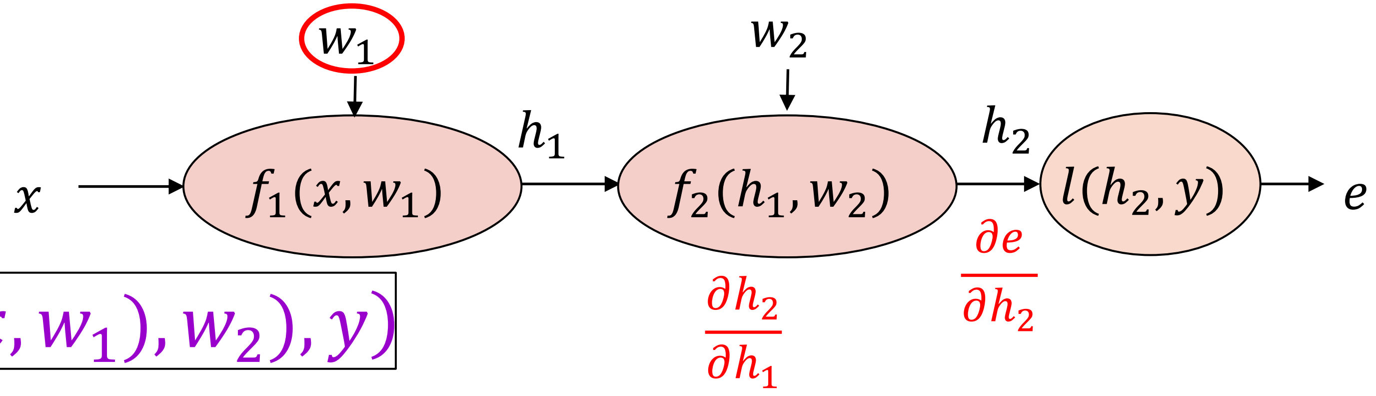
$$\boxed{e = l(f_2(f_1(x, w_1), w_2), y)}$$

$$\frac{\partial e}{\partial w_2} = \frac{\partial e}{\partial h_2}\frac{\partial h_2}{\partial w_2}$$

$$\frac{\partial e}{\partial w_1} = \frac{\partial e}{\partial h_2}$$

CHAIN RULE

$k = 2$



$$\boxed{e = l(f_2(f_1(x, w_1), w_2), y)}$$

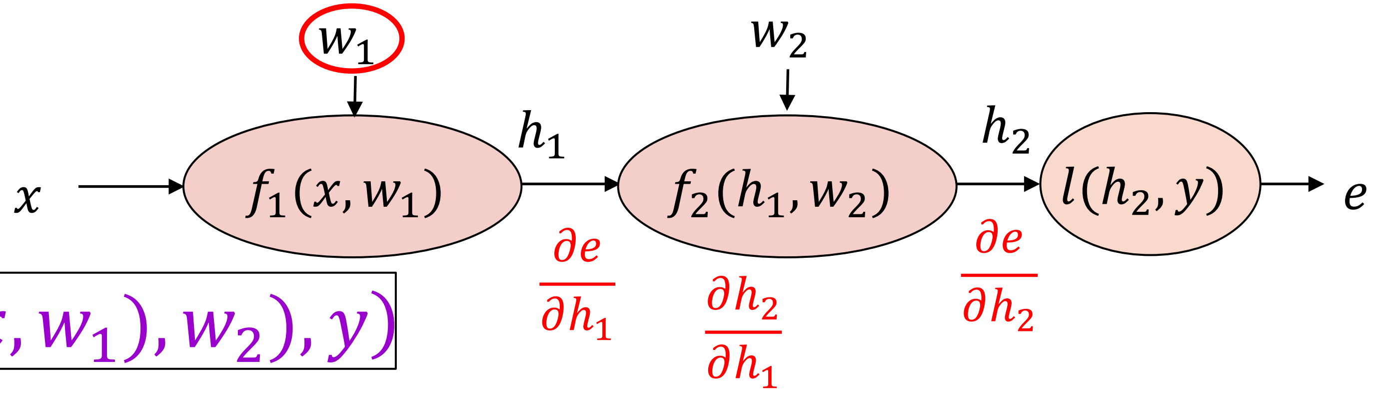
$$\frac{\partial e}{\partial w_2} = \frac{\partial e}{\partial h_2}\frac{\partial h_2}{\partial w_2}$$

$$\frac{\partial e}{\partial w_1} = \frac{\partial e}{\partial h_2}\frac{\partial h_2}{\partial h_1}$$

CHAIN RULE

$k = 2$

$$\boxed{e = l(f_2(f_1(x, w_1), w_2), y)}$$

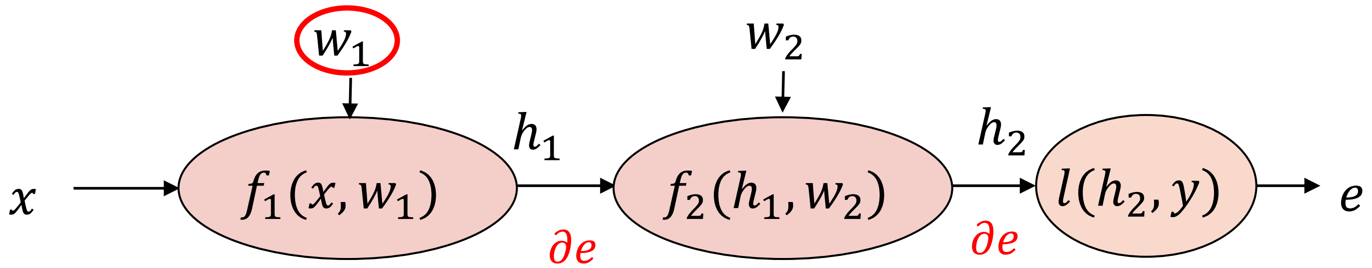


$$\frac{\partial e}{\partial w_2} = \frac{\partial e}{\partial h_2}\frac{\partial h_2}{\partial w_2}$$

$$\frac{\partial e}{\partial w_1} = \boxed{\frac{\partial e}{\partial h_2}\frac{\partial h_2}{\partial h_1}}$$

CHAIN RULE

$k = 2$



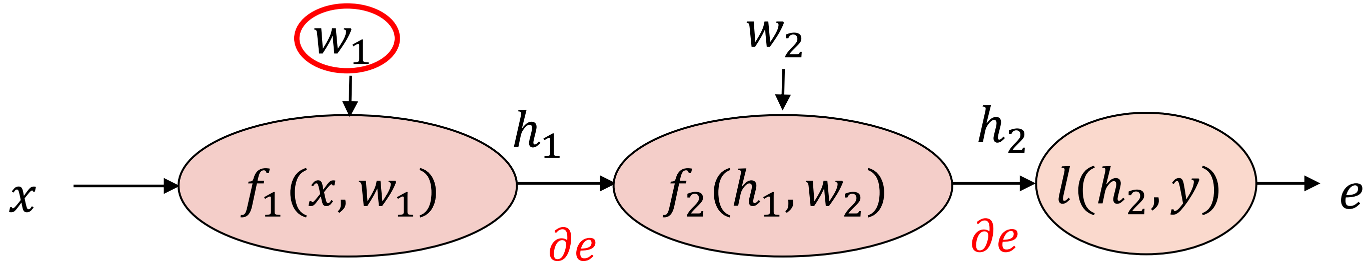
$$\boxed{e = l(f_2(f_1(x, w_1), w_2), y)}$$

$$\frac{\partial e}{\partial w_2} = \frac{\partial e}{\partial h_2}\frac{\partial h_2}{\partial w_2}$$

$$\frac{\partial e}{\partial w_1} = \boxed{\frac{\partial e}{\partial h_2}\frac{\partial h_2}{\partial h_1}}\frac{\partial h_1}{\partial w_1}$$

CHAIN RULE

$k = 2$



$$\boxed{e = l(f_2(f_1(x, w_1), w_2), y)}$$

$\frac{\partial h_1}{\partial w_1}$ $\frac{\partial e}{\partial h_1}$ $\frac{\partial h_2}{\partial h_1}$ $\frac{\partial e}{\partial h_2}$

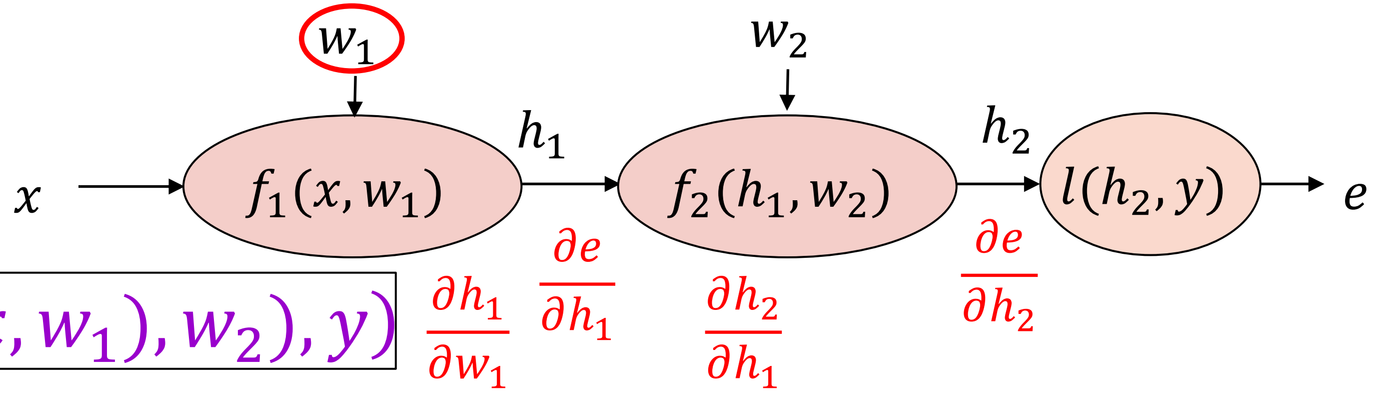
$$\frac{\partial e}{\partial w_2} = \frac{\partial e}{\partial h_2}\frac{\partial h_2}{\partial w_2}$$

$$\frac{\partial e}{\partial w_1} = \boxed{\frac{\partial e}{\partial h_2}\frac{\partial h_2}{\partial h_1}}\frac{\partial h_1}{\partial w_1}$$

Example: $e = -\log\left(\sigma(w_1^T x)\right)$ (assume $y = 1$)

CHAIN RULE

$$k = 2$$



$$\boxed{e = l(f_2(f_1(x,w_1),w_2),y)}$$

$$\frac{\partial e}{\partial w_2} = \frac{\partial e}{\partial h_2}\frac{\partial h_2}{\partial w_2} \qquad \frac{\partial e}{\partial w_1} = \boxed{\frac{\partial e}{\partial h_2}\frac{\partial h_2}{\partial h_1}}\frac{\partial h_1}{\partial w_1}$$

Example: $e = -\log\left(\sigma(w_1^T x)\right)$ (assume $y = 1$)

$$h_1 = f_1(x,w_1) = w_1^T x$$

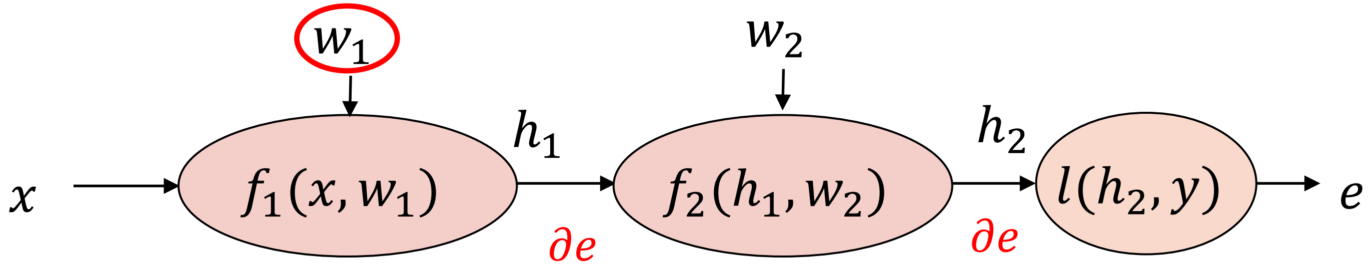
$$h_2 = f_2(h_1) = \sigma(h_1)$$

$$e = l(h_2,1) = -\log(h_2)$$

$$\frac{\partial e}{\partial w_1} = \frac{\partial e}{\partial h_2}\frac{\partial h_2}{\partial h_1}\frac{\partial h_1}{\partial w_1} = ??????$$

CHAIN RULE

$k = 2$



$\boxed{e = l(f_2(f_1(x,w_1),w_2),y)}$ $\frac{\partial h_1}{\partial w_1}$ $\frac{\partial e}{\partial h_1}$ $\frac{\partial h_2}{\partial h_1}$ $\frac{\partial e}{\partial h_2}$

$$\frac{\partial e}{\partial w_2} = \frac{\partial e}{\partial h_2}\frac{\partial h_2}{\partial w_2} \qquad \frac{\partial e}{\partial w_1} = \boxed{\frac{\partial e}{\partial h_2}\frac{\partial h_2}{\partial h_1}}\frac{\partial h_1}{\partial w_1}$$

Example: $e = -\log\left(\sigma(w_1^T x)\right)$ (assume $y = 1$)

$h_1 = f_1(x,w_1) = w_1^T x$

$h_2 = f_2(h_1) = \sigma(h_1)$

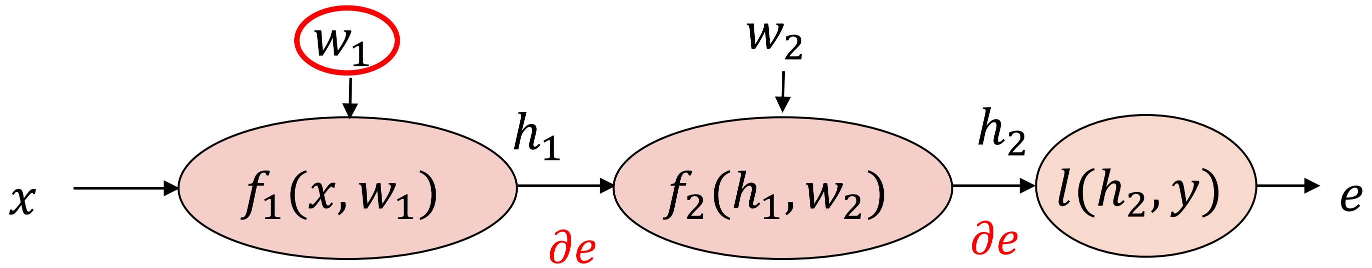
$e = l(h_2,1) = -\log(h_2)$

$\frac{\partial h_1}{\partial w_1} = ???$ $\frac{\partial h_2}{\partial h_1} = ???$ $\frac{\partial e}{\partial h_2} = ???$

$$\frac{\partial e}{\partial w_1} = \frac{\partial e}{\partial h_2}\frac{\partial h_2}{\partial h_1}\frac{\partial h_1}{\partial w_1} = ??????$$

CHAIN RULE

$k = 2$



$\boxed{e = l(f_2(f_1(x, w_1), w_2), y)}$ $\frac{\partial h_1}{\partial w_1}$ $\frac{\partial e}{\partial h_1}$ $\frac{\partial h_2}{\partial h_1}$ $\frac{\partial e}{\partial h_2}$

$$\frac{\partial e}{\partial w_2} = \frac{\partial e}{\partial h_2}\frac{\partial h_2}{\partial w_2}$$

$$\frac{\partial e}{\partial w_1} = \boxed{\frac{\partial e}{\partial h_2}\frac{\partial h_2}{\partial h_1}}\frac{\partial h_1}{\partial w_1}$$

Example: $e = -\log\left(\sigma(w_1^T x)\right)$ (assume $y = 1$)

$h_1 = f_1(x, w_1) = w_1^T x$

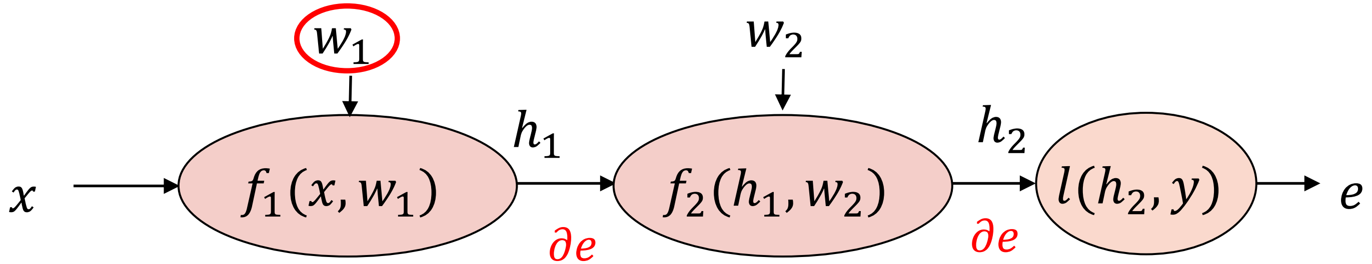
$h_2 = f_2(h_1) = \sigma(h_1)$

$e = l(h_2, 1) = -\log(h_2)$

$\frac{\partial h_1}{\partial w_1} = x$ $\frac{\partial h_2}{\partial h_1} = \sigma'(h_1) = \sigma(h_1)(1 - \sigma(h_1))$ $\frac{\partial e}{\partial h_2} = -\frac{1}{h_2}$

$$\frac{\partial e}{\partial w_1} = \frac{\partial e}{\partial h_2}\frac{\partial h_2}{\partial h_1}\frac{\partial h_1}{\partial w_1} = ??????$$

CHAIN RULE



$k = 2$

$$\boxed{e = l(f_2(f_1(x, w_1), w_2), y)}$$

$\frac{\partial h_1}{\partial w_1}$ $\frac{\partial e}{\partial h_1}$ $\frac{\partial h_2}{\partial h_1}$ $\frac{\partial e}{\partial h_2}$

$$\frac{\partial e}{\partial w_2} = \frac{\partial e}{\partial h_2}\frac{\partial h_2}{\partial w_2}$$

$$\frac{\partial e}{\partial w_1} = \boxed{\frac{\partial e}{\partial h_2}\frac{\partial h_2}{\partial h_1}}\frac{\partial h_1}{\partial w_1}$$

Example: $e = -\log\left(\sigma(w_1^T x)\right)$ (assume $y = 1$)

$h_1 = f_1(x, w_1) = w_1^T x$

$h_2 = f_2(h_1) = \sigma(h_1)$

$e = l(h_2, 1) = -\log(h_2)$

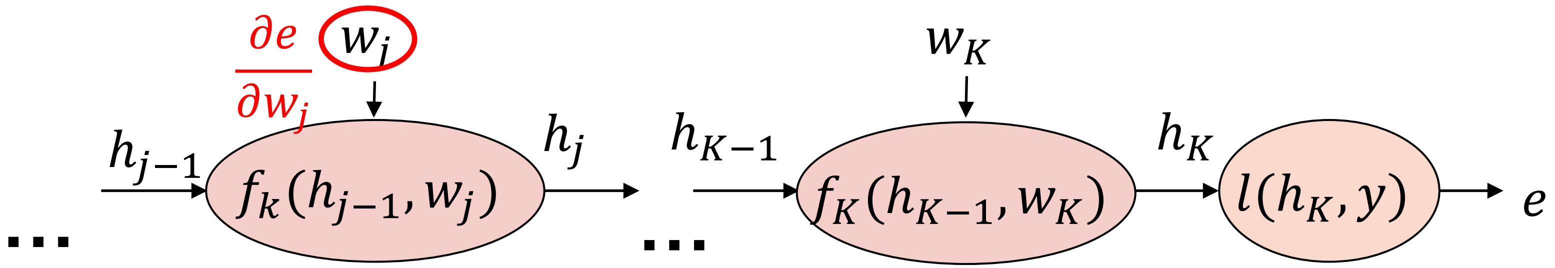
$\frac{\partial h_1}{\partial w_1} =$

$\frac{\partial h_2}{\partial h_1} = \sigma'(h_1) = \sigma(h_1)(1 - \sigma(h_1))$

$\frac{\partial e}{\partial h_2} = -\frac{1}{h_2}$

$$\frac{\partial e}{\partial w_1} = \frac{\partial e}{\partial h_2}\frac{\partial h_2}{\partial h_1}\frac{\partial h_1}{\partial w_1} = -\frac{1}{\sigma(w_1^T x)}\sigma(w_1^T x)\left(1 - \sigma(w_1^T x)\right)x = -\sigma(-w_1^T x)x$$

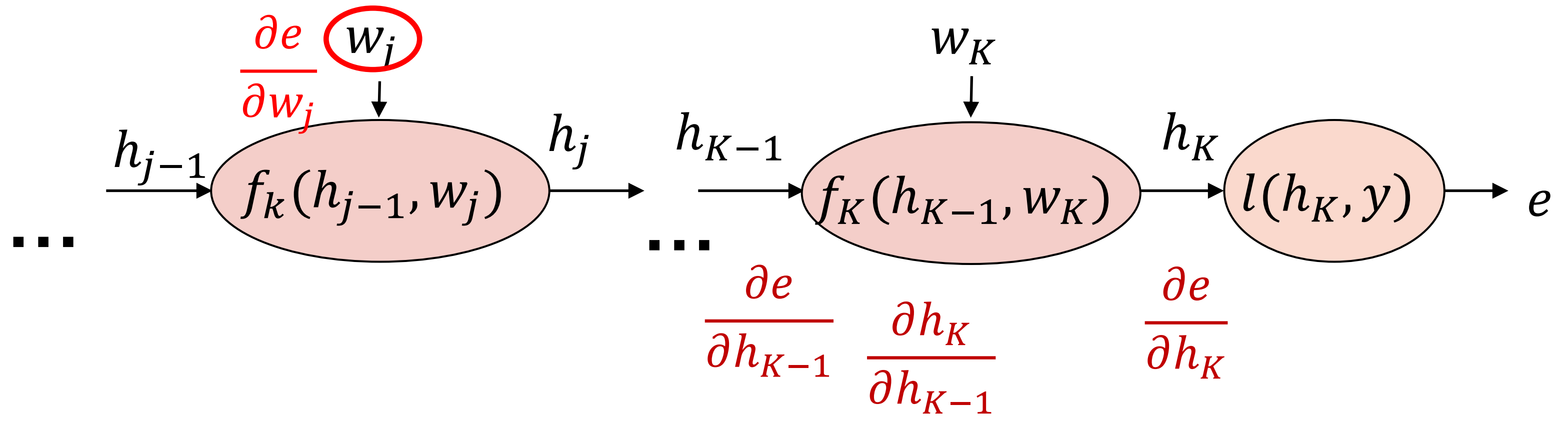
CHAIN RULE



General case:

$$\frac{\partial e}{\partial w_j} = \frac{\partial e}{\partial h_K} \frac{\partial h_K}{\partial h_{K-1}} \cdots \frac{\partial h_{j+1}}{\partial h_j} \frac{\partial h_j}{\partial w_j}$$

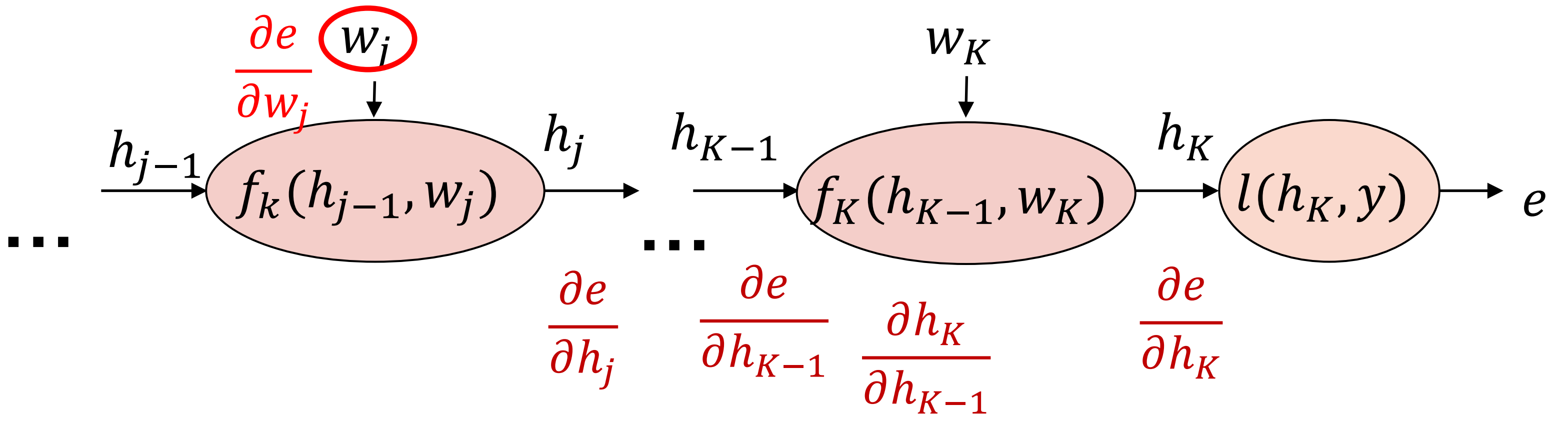
CHAIN RULE



General case:

$$\frac{\partial e}{\partial w_j} = \frac{\partial e}{\partial h_K} \frac{\partial h_K}{\partial h_{K-1}} \cdots \frac{\partial h_{j+1}}{\partial h_j} \frac{\partial h_j}{\partial w_j}$$

CHAIN RULE



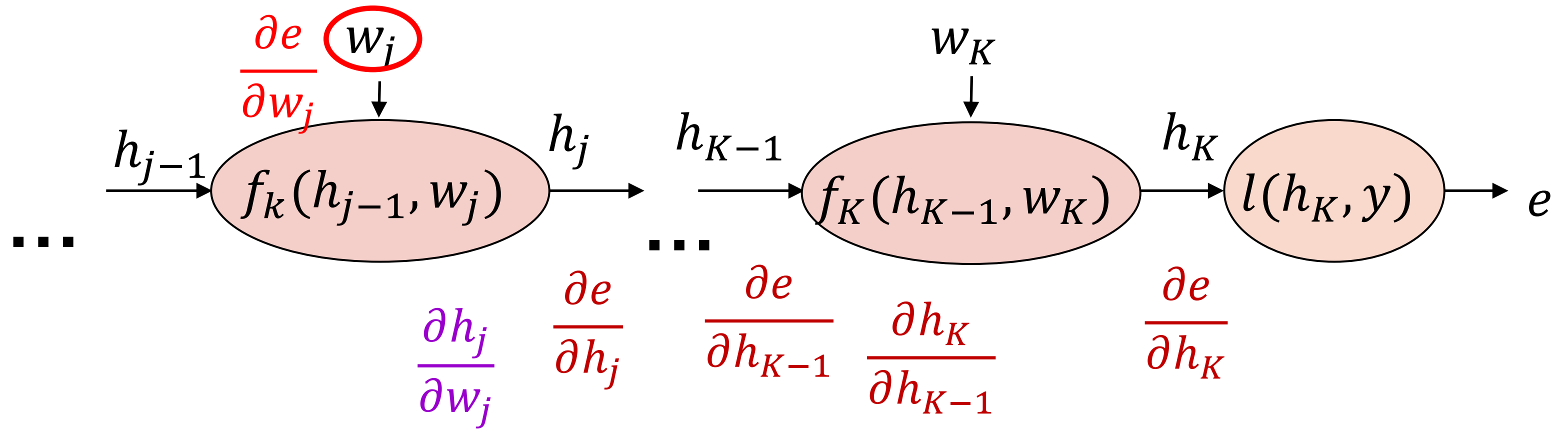
General case:

$$\frac{\partial e}{\partial w_j} = \boxed{\frac{\partial e}{\partial h_K} \frac{\partial h_K}{\partial h_{K-1}} \cdots \frac{\partial h_{j+1}}{\partial h_j}} \frac{\partial h_j}{\partial w_j}$$

Upstream gradient

$$\frac{\partial e}{\partial h_j}$$

CHAIN RULE



General case:

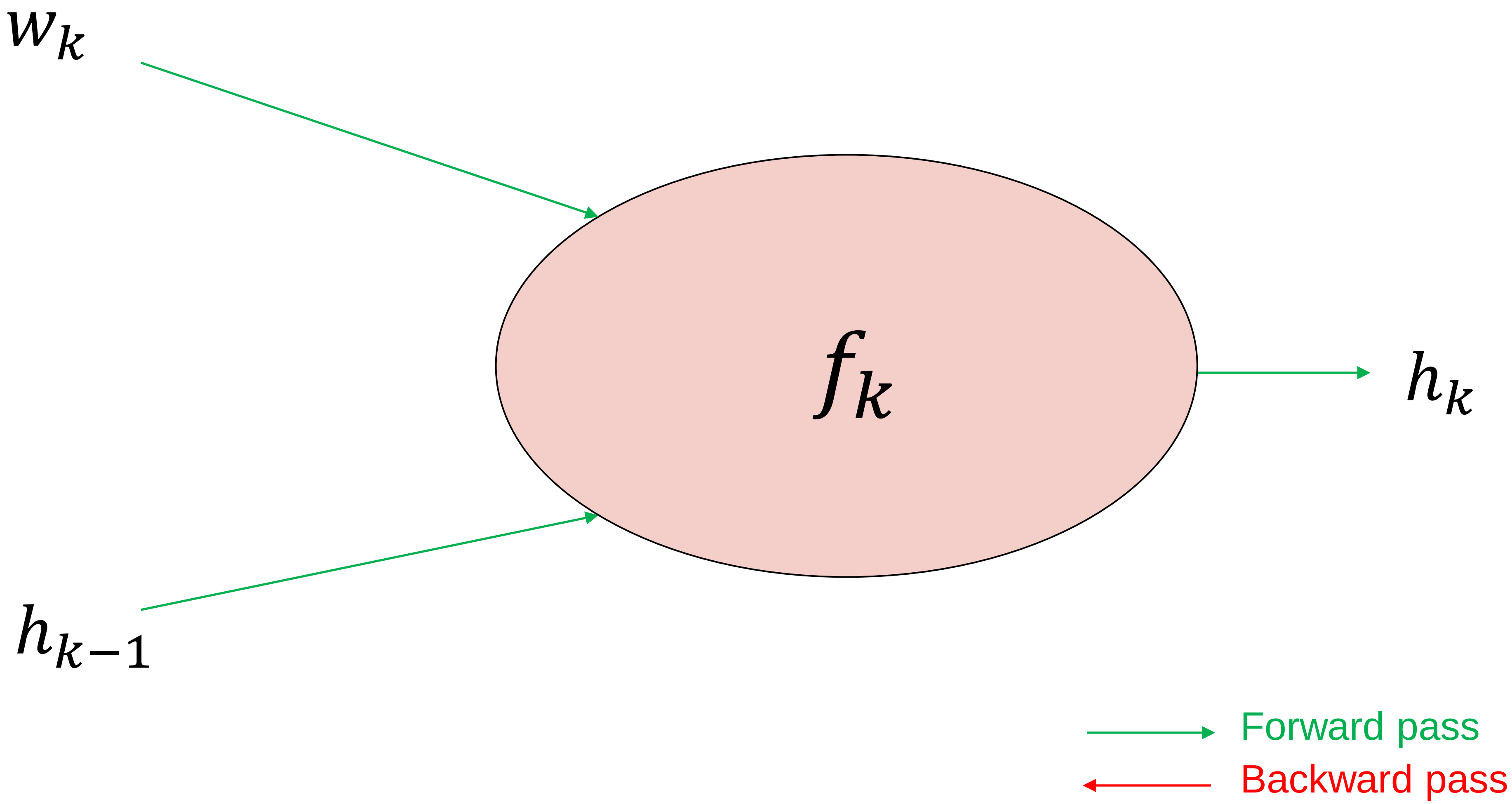
$$\frac{\partial e}{\partial w_j} = \boxed{\frac{\partial e}{\partial h_K} \frac{\partial h_K}{\partial h_{K-1}} \cdots \frac{\partial h_{j+1}}{\partial h_j}} \frac{\partial h_j}{\partial w_j}$$

Upstream gradient

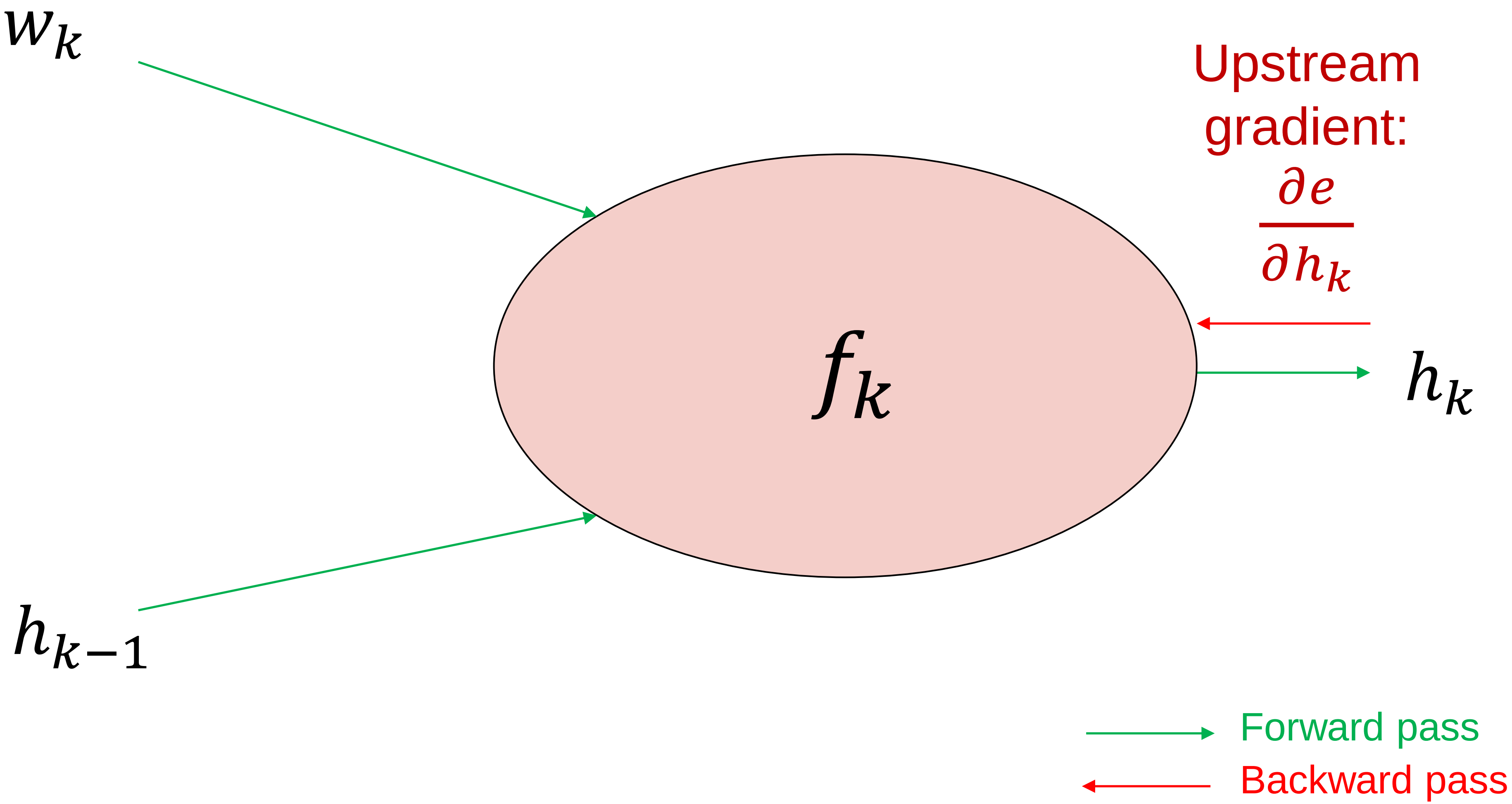
$$\frac{\partial e}{\partial h_j}$$

Local gradient

BACKPROPAGATION SUMMARY



BACKPROPAGATION SUMMARY



w_k

h_{k-1}

f_k

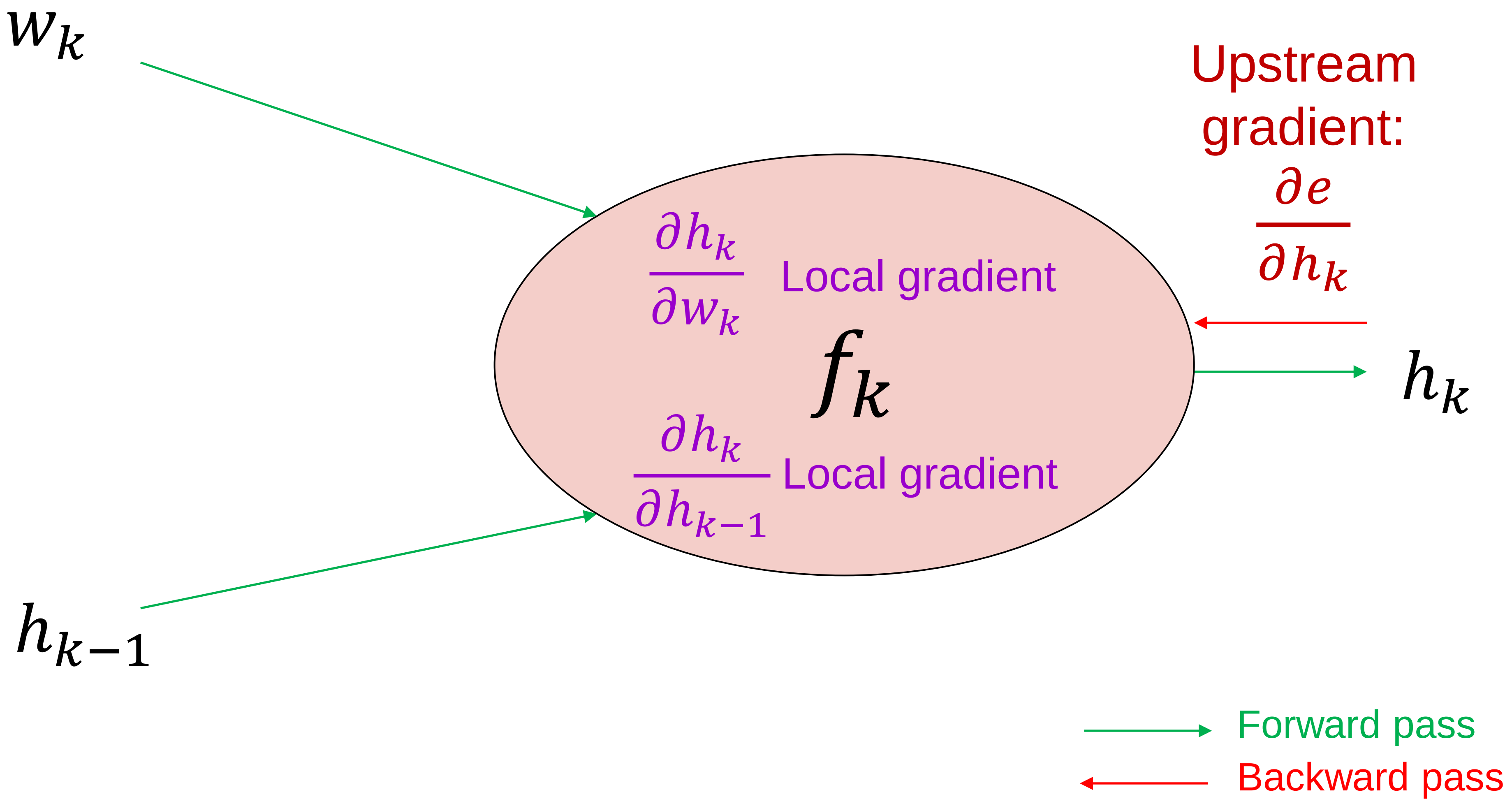
Upstream gradient: $\frac{\partial e}{\partial h_k}$

h_k

→ Forward pass

← Backward pass

BACKPROPAGATION SUMMARY



w_k

h_{k-1}

f_k

$\frac{\partial h_k}{\partial w_k}$ Local gradient

$\frac{\partial h_k}{\partial h_{k-1}}$ Local gradient

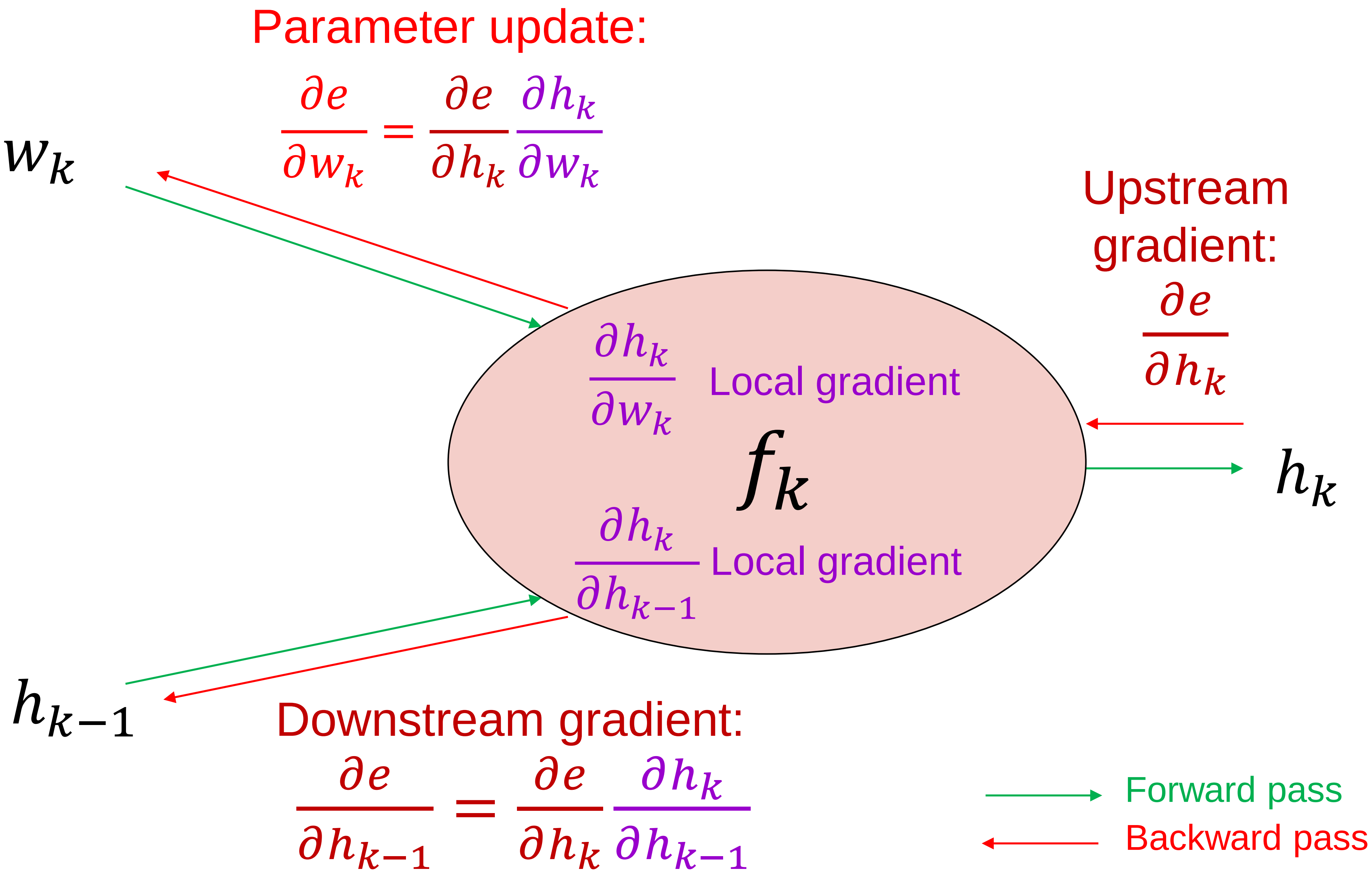
Upstream gradient: $\frac{\partial e}{\partial h_k}$

h_k

Forward pass

Backward pass

BACKPROPAGATION SUMMARY



Parameter update:

$$\frac{\partial e}{\partial w_k} = \frac{\partial e}{\partial h_k}\frac{\partial h_k}{\partial w_k}$$

w_k

Upstream gradient:

$$\frac{\partial e}{\partial h_k}$$

h_k

$$\frac{\partial h_k}{\partial w_k}$$ Local gradient

f_k

$$\frac{\partial h_k}{\partial h_{k-1}}$$ Local gradient

h_{k-1}

Downstream gradient:

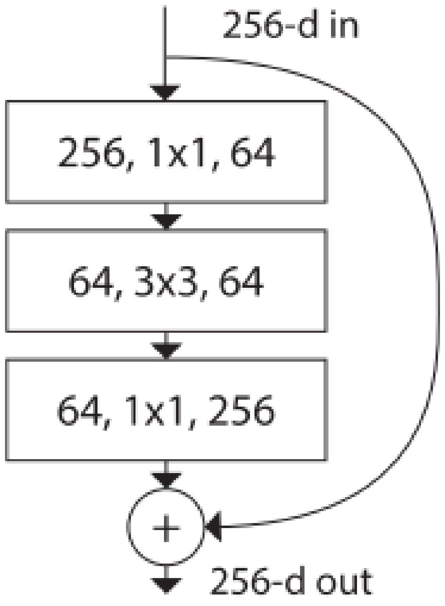
$$\frac{\partial e}{\partial h_{k-1}} = \frac{\partial e}{\partial h_k}\frac{\partial h_k}{\partial h_{k-1}}$$

Forward pass

Backward pass

WHAT ABOUT MORE GENERAL COMPUTATION GRAPHS?

ResNet



ResNeXt

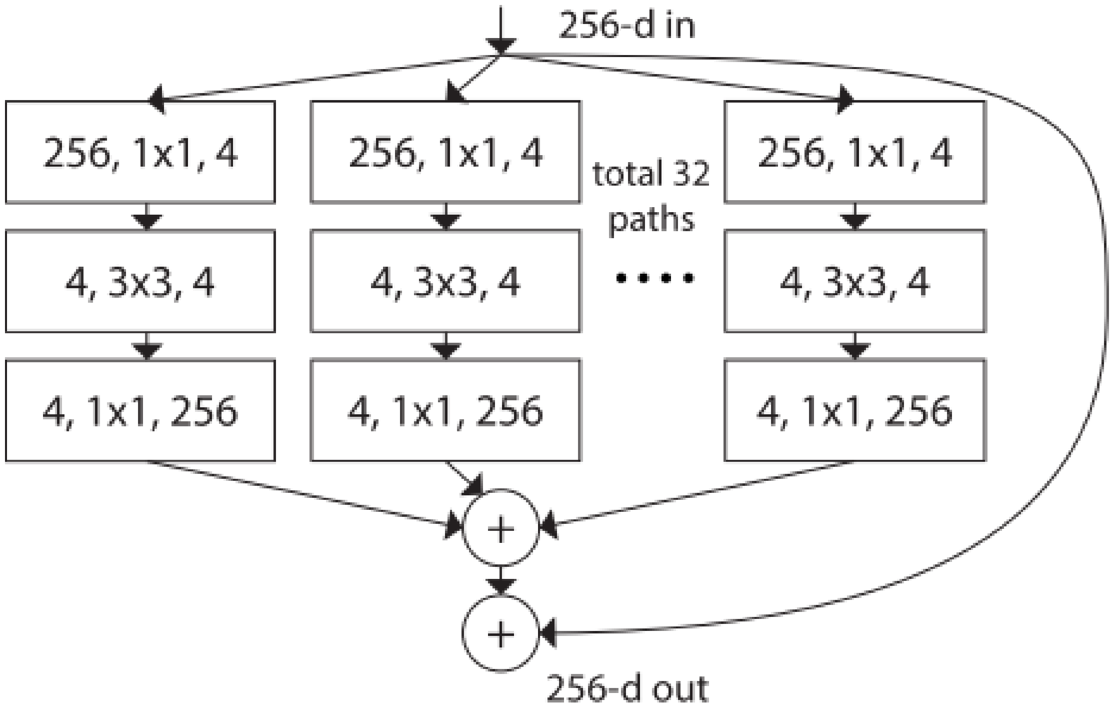
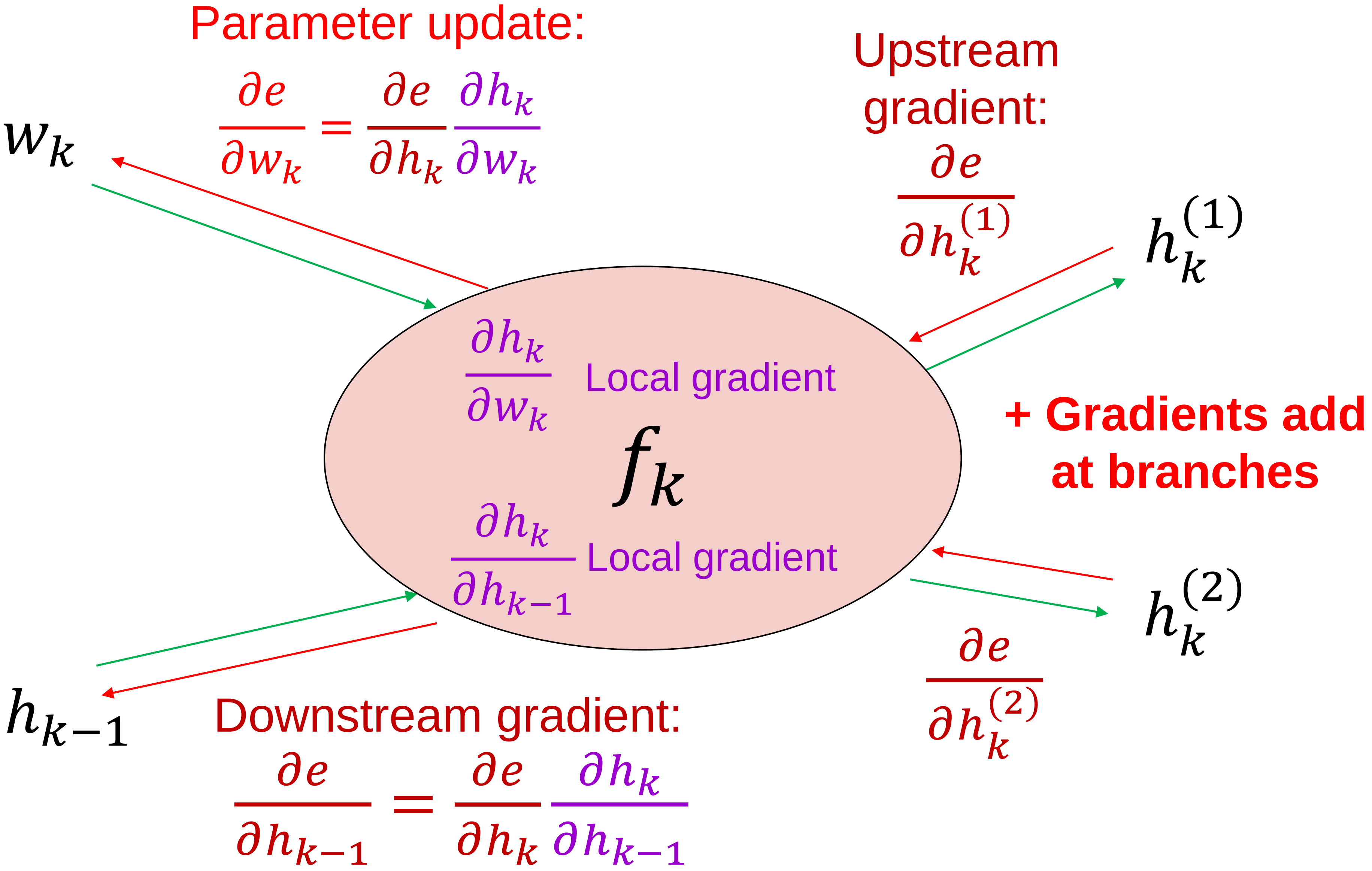


Figure source

WHAT ABOUT MORE GENERAL COMPUTATION GRAPHS?



Parameter update:

$$\frac{\partial e}{\partial w_k} = \frac{\partial e}{\partial h_k}\frac{\partial h_k}{\partial w_k}$$

Upstream gradient:

$$\frac{\partial e}{\partial h_k^{(1)}}$$

$$\frac{\partial e}{\partial h_k^{(2)}}$$

$$\frac{\partial h_k}{\partial w_k}$$ Local gradient

$$f_k$$

$$\frac{\partial h_k}{\partial h_{k-1}}$$ Local gradient

+ Gradients add at branches

Downstream gradient:

$$\frac{\partial e}{\partial h_{k-1}} = \frac{\partial e}{\partial h_k}\frac{\partial h_k}{\partial h_{k-1}}$$

TRAINING OF MULTI-LAYER NETWORKS

- Find network weights to minimize the error between true and estimated outputs of training examples:

$$E(\mathbf{w}) = \sum_{j=1}^{N} \left(y_j - f_{\mathbf{w}}(\mathbf{x}_j)\right)^2$$

TRAINING OF MULTI-LAYER NETWORKS

- Find network weights to minimize the error between true and estimated outputs of training examples:

$$E(\mathbf{w}) = \sum_{j=1}^{N} \left(y_j - f_{\mathbf{w}}(\mathbf{x}_j) \right)^2$$

- Update weights by **gradient descent:** $\mathbf{w} \leftarrow \mathbf{w} - \alpha \frac{\partial E}{\partial \mathbf{w}}$

TRAINING OF MULTI-LAYER NETWORKS

- Find network weights to minimize the error between true and estimated outputs of training examples:

$$E(\mathbf{w}) = \sum_{j=1}^{N} \left(y_j - f_{\mathbf{w}}(\mathbf{x}_j) \right)^2$$

- Update weights by **gradient descent:** $\mathbf{w} \leftarrow \mathbf{w} - \alpha \frac{\partial E}{\partial \mathbf{w}}$

- **Back-propagation:** gradients are computed in the direction from output to input layers and combined using chain rule

NEURAL NETWORKS: PROS AND CONS

Pros

- Flexible and general function approximation framework
- Can build extremely powerful models by adding more layers

Cons

- Hard to analyze theoretically (e.g., training is prone to local optima)
- Huge amount of training data, computing power may be required to get good performance
- The space of implementation choices are huge (network architectures, parameters)

ACKNOWLEDGEMENT

Thanks to the following courses and corresponding researchers for making their teaching/research material online

- Deep Learning, Stanford University
- Introduction to Deep Learning, University of Illinois at Urbana-Champaign
- Introduction to Deep Learning, Carnegie Mellon University
- Convolutional Neural Networks for Visual Recognition, Stanford University
- Natural Language Processing with Deep Learning, Stanford University
- And Many More ……

Thank You