Indian Institute of Information Technology, Allahabad

Neural Networks

By

Dr. Satish Kumar Singh & Dr. Shiv Ram Dubey
Computer Vision and Biometrics Lab
Department of Information Technology
Indian Institute of Information Technology, Allahabad

TEAM

Computer Vision and Biometrics Lab (CVBL)

Department of Information Technology

Indian Institute of Information Technology Allahabad

Course Instructors

Dr. Satish Kumar Singh, Associate Professor, IIIT Allahabad (Email: sk.singh@iiita.ac.in)

Dr. Shiv Ram Dubey, Assistant Professor, IIIT Allahabad (Email: srdubey@iiita.ac.in)

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.

WE HAVE LEARNED SO FOR IN THIS MODULE

Image features and categorization

Choosing right features

Object, Scene, Action, etc.

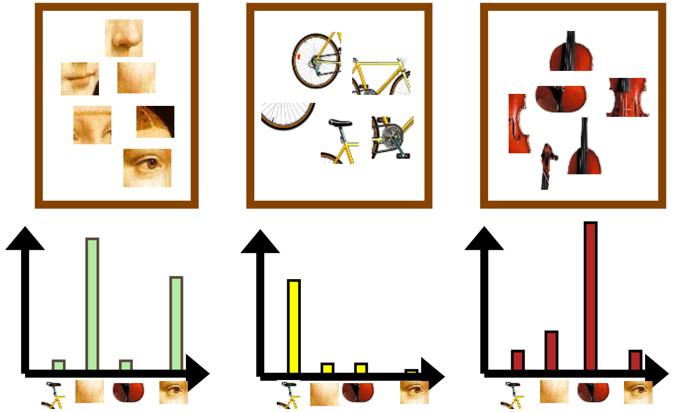
Bag-of-visual-words

Extract local features

Learn "visual vocabulary"

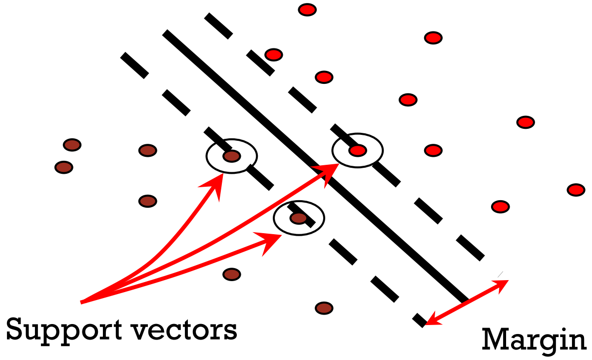
Quantize features using visual vocabulary

Represent by frequencies of "visual words"



Classifiers

Nearest neighbor, KNN, Linear classifier, SVM, Non-linear SVM, Multi-class SVM, Softmax classifier



TODAY'S CLASS

Optimization

Gradient Descent & Back propagation

Update rule

Neural networks

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

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:

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

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

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:

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

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

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

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

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:

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

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

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

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

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

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

↓

$q = x + y$ 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

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

↓

$q = x + y$ 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

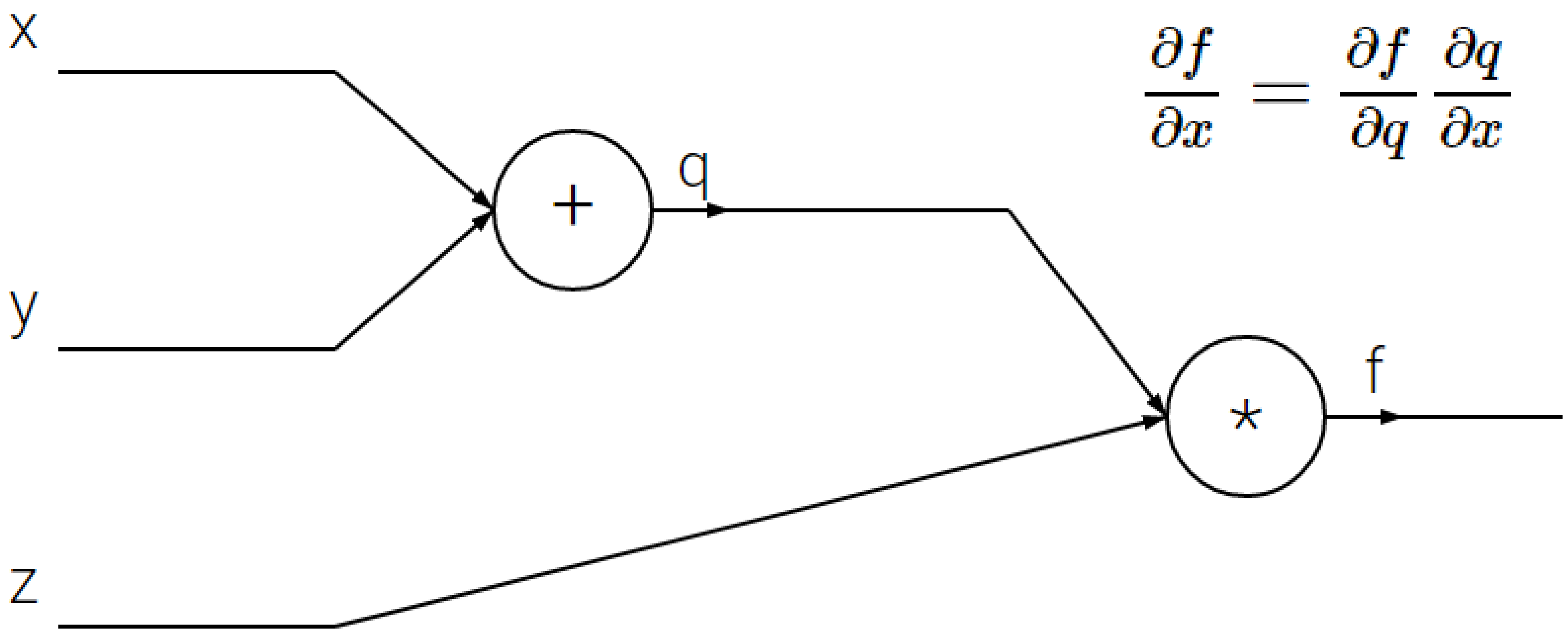
$$f(x, y, z) = (x + y)z$$

$q = x + y$ 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

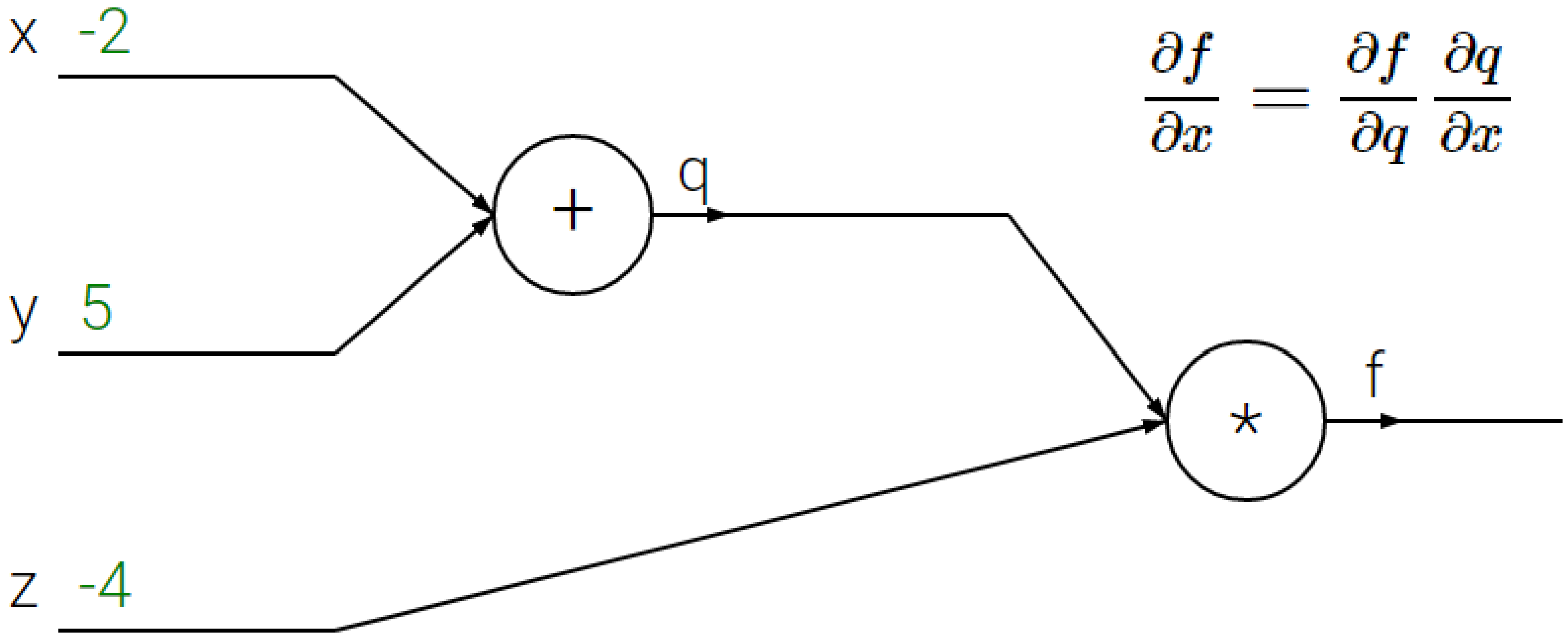
$$f(x, y, z) = (x + y)z$$

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

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

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

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



Source: http://cs231n.github.io

COMPOUND EXPRESSIONS WITH CHAIN RULE

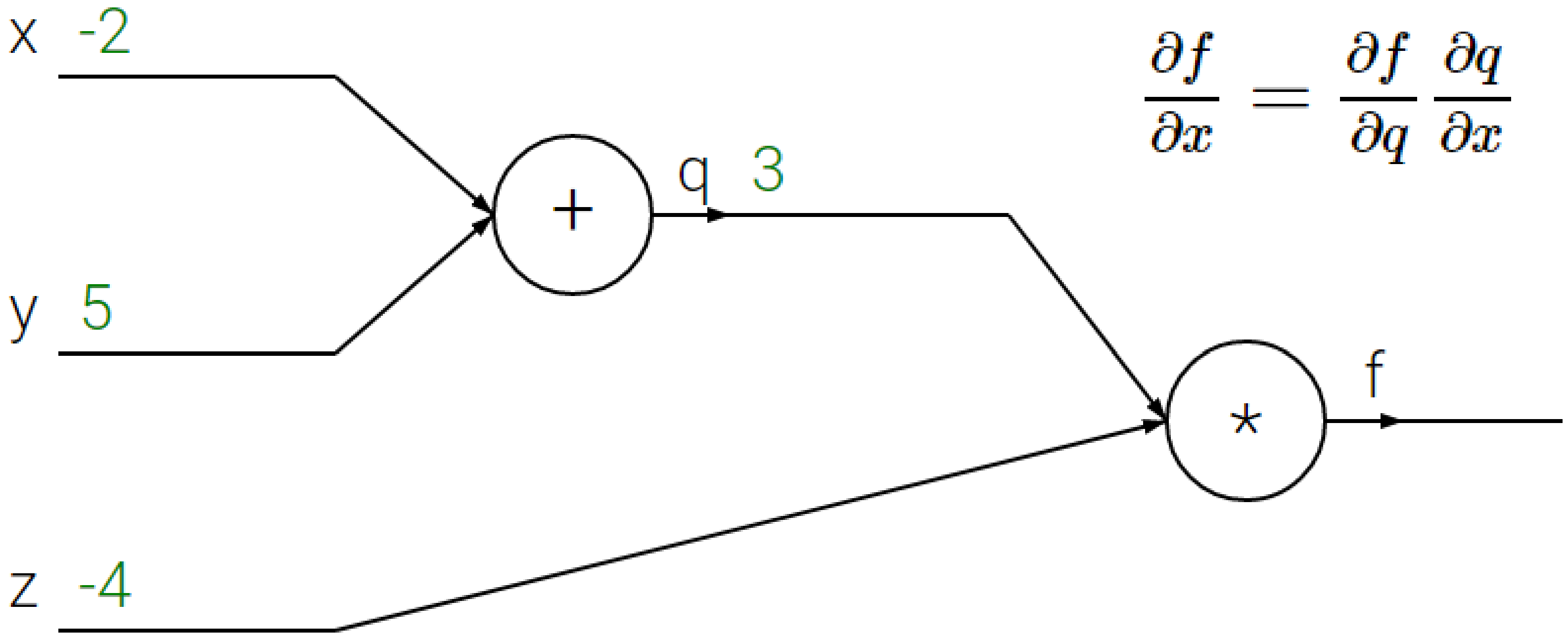
$$f(x, y, z) = (x + y)z$$

$q = x + y$ 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

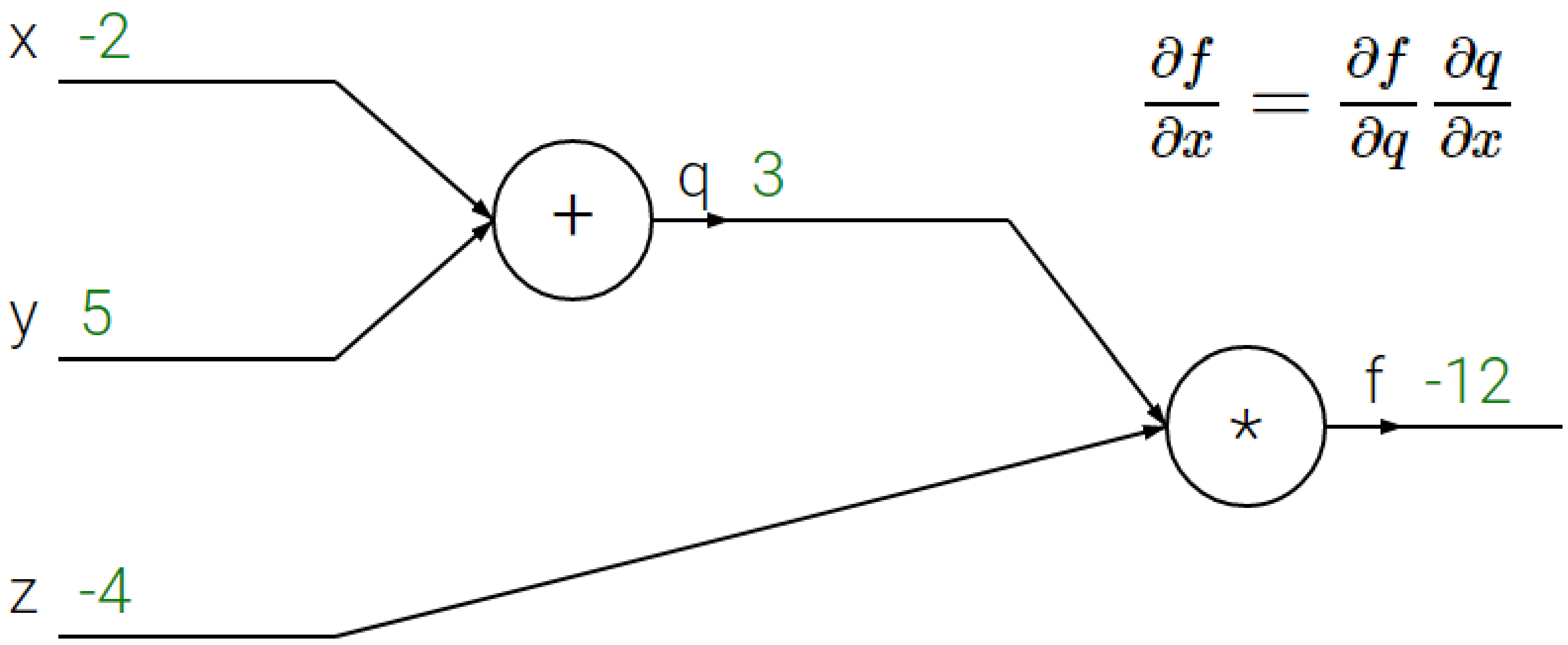
$$f(x, y, z) = (x + y)z$$

$q = x + y$ 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

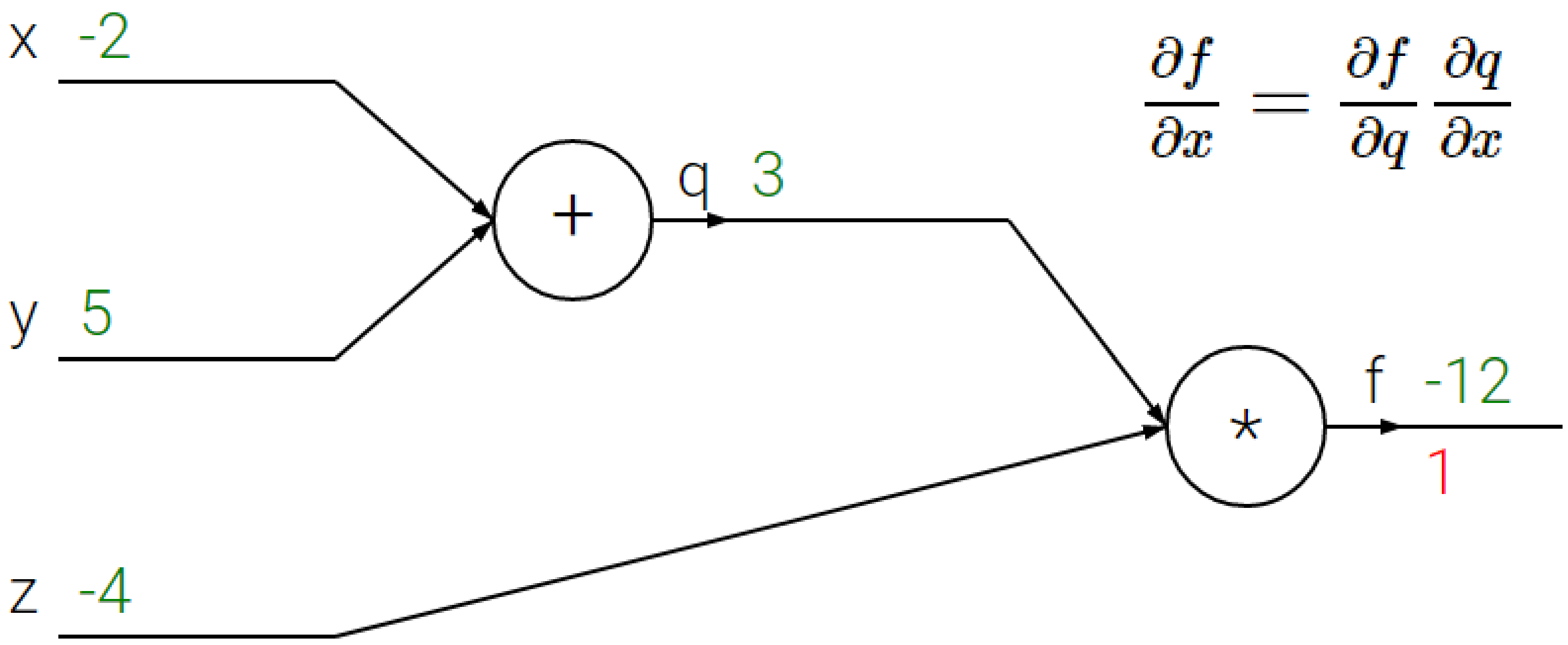
$$f(x, y, z) = (x + y)z$$

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

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

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

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



Source: http://cs231n.github.io

COMPOUND EXPRESSIONS WITH CHAIN RULE

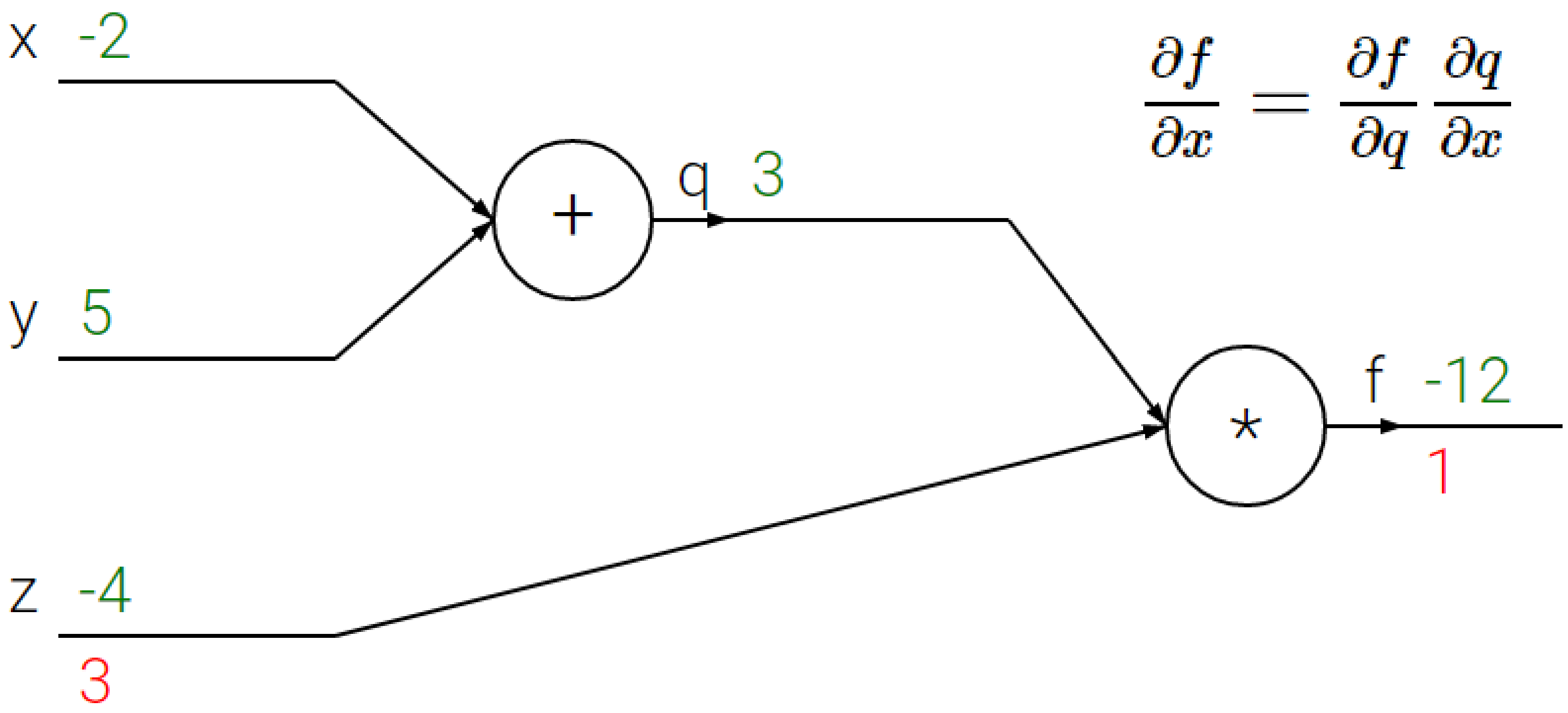
$$f(x, y, z) = (x + y)z$$

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

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

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

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



Source: http://cs231n.github.io

COMPOUND EXPRESSIONS WITH CHAIN RULE

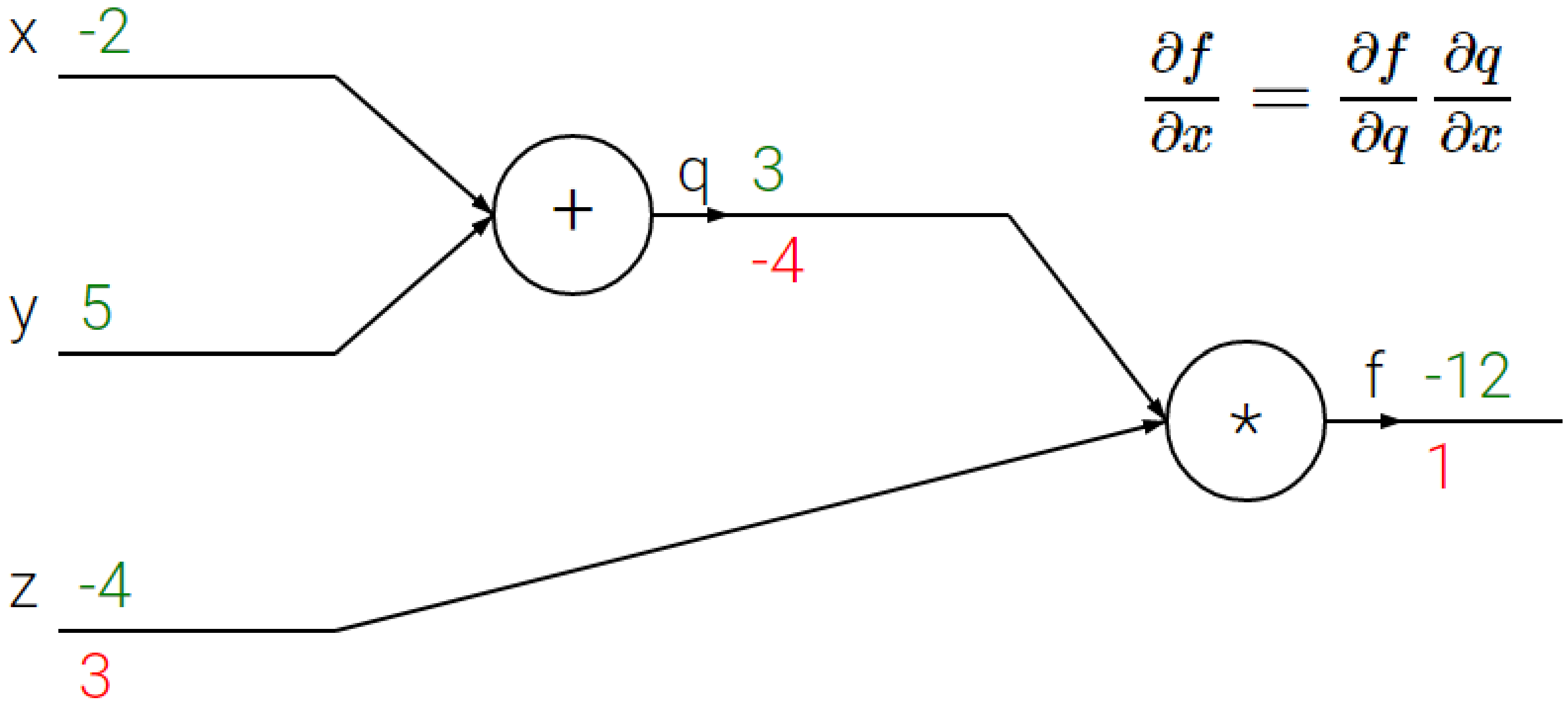
$$f(x, y, z) = (x + y)z$$

$q = x + y$ 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

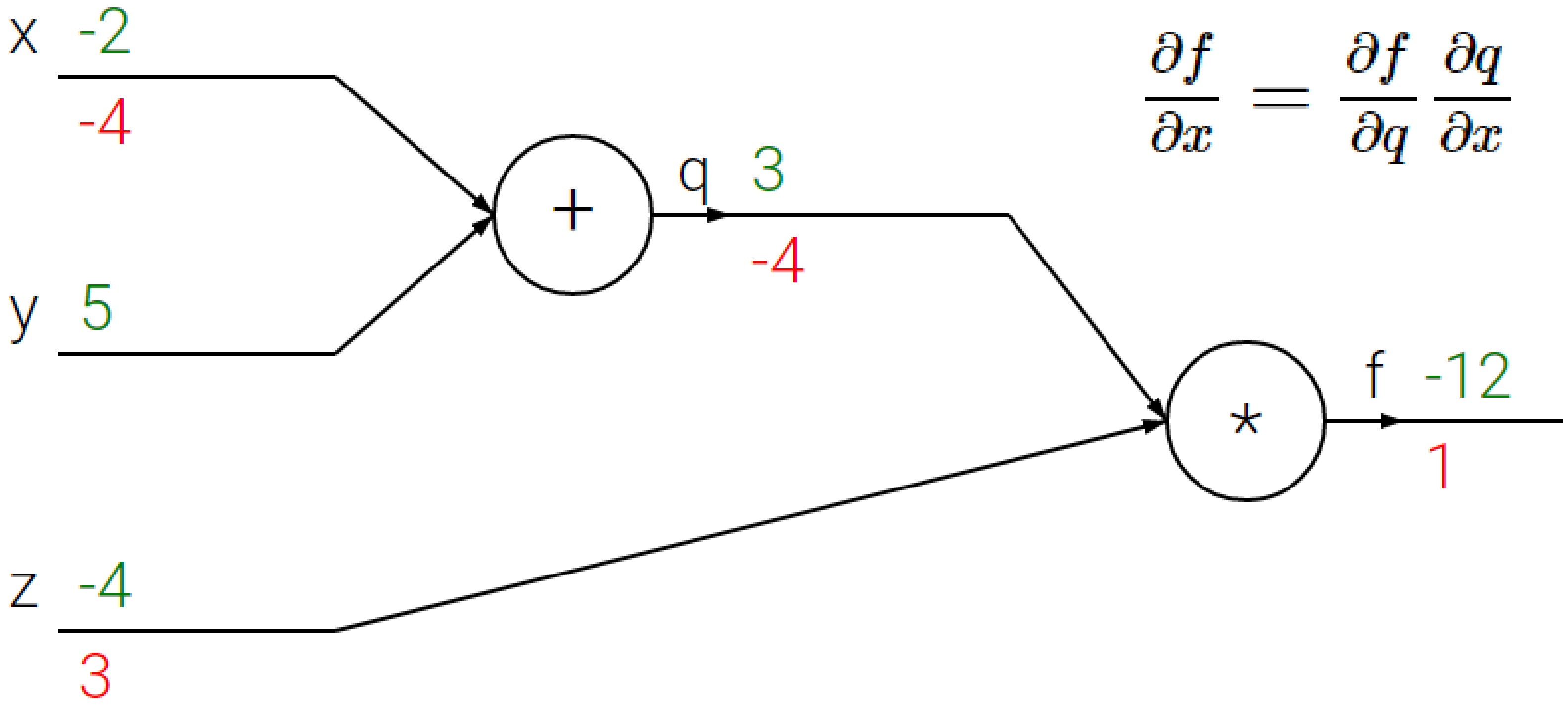
$$f(x,y,z) = (x+y)z$$

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

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

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

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



Source: http://cs231n.github.io

COMPOUND EXPRESSIONS WITH CHAIN RULE

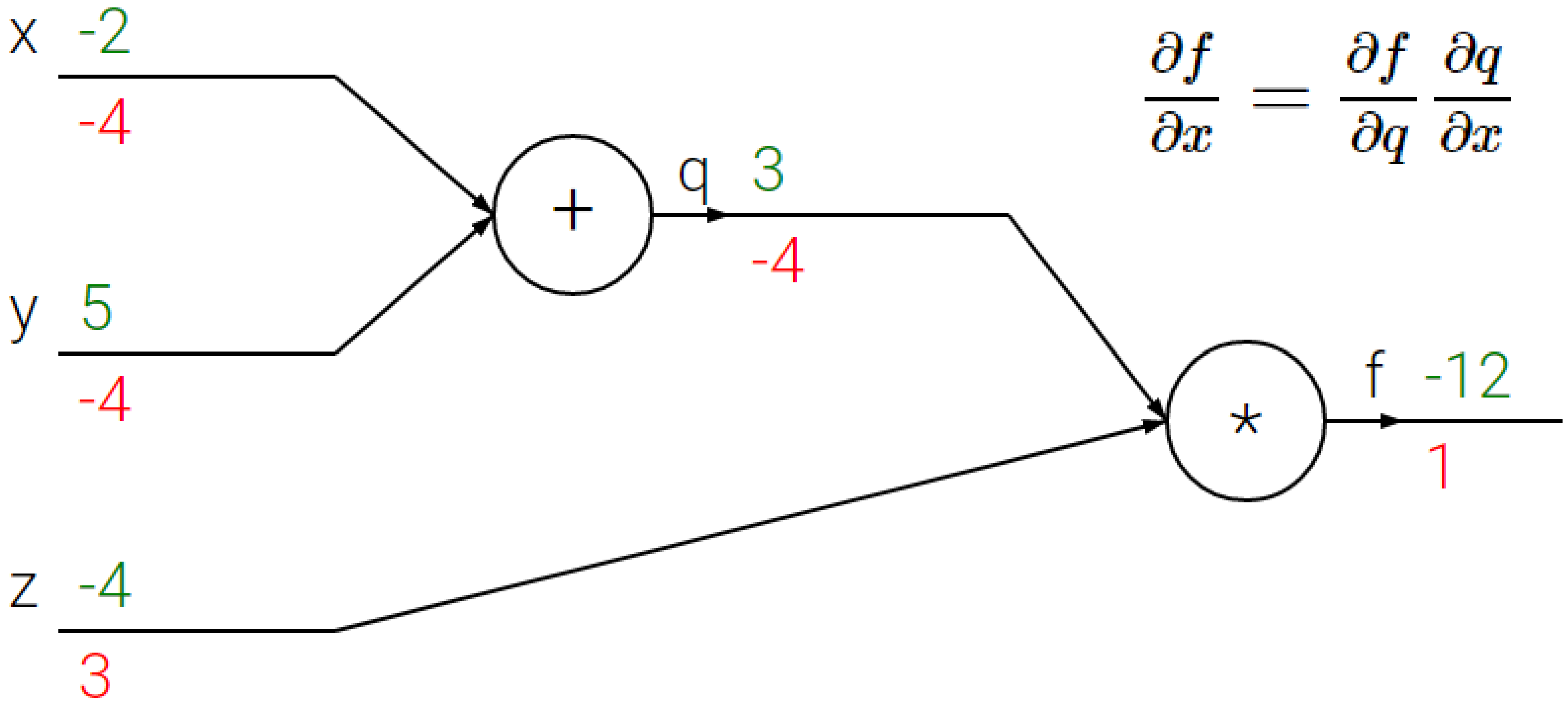
$$f(x, y, z) = (x + y)z$$

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

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

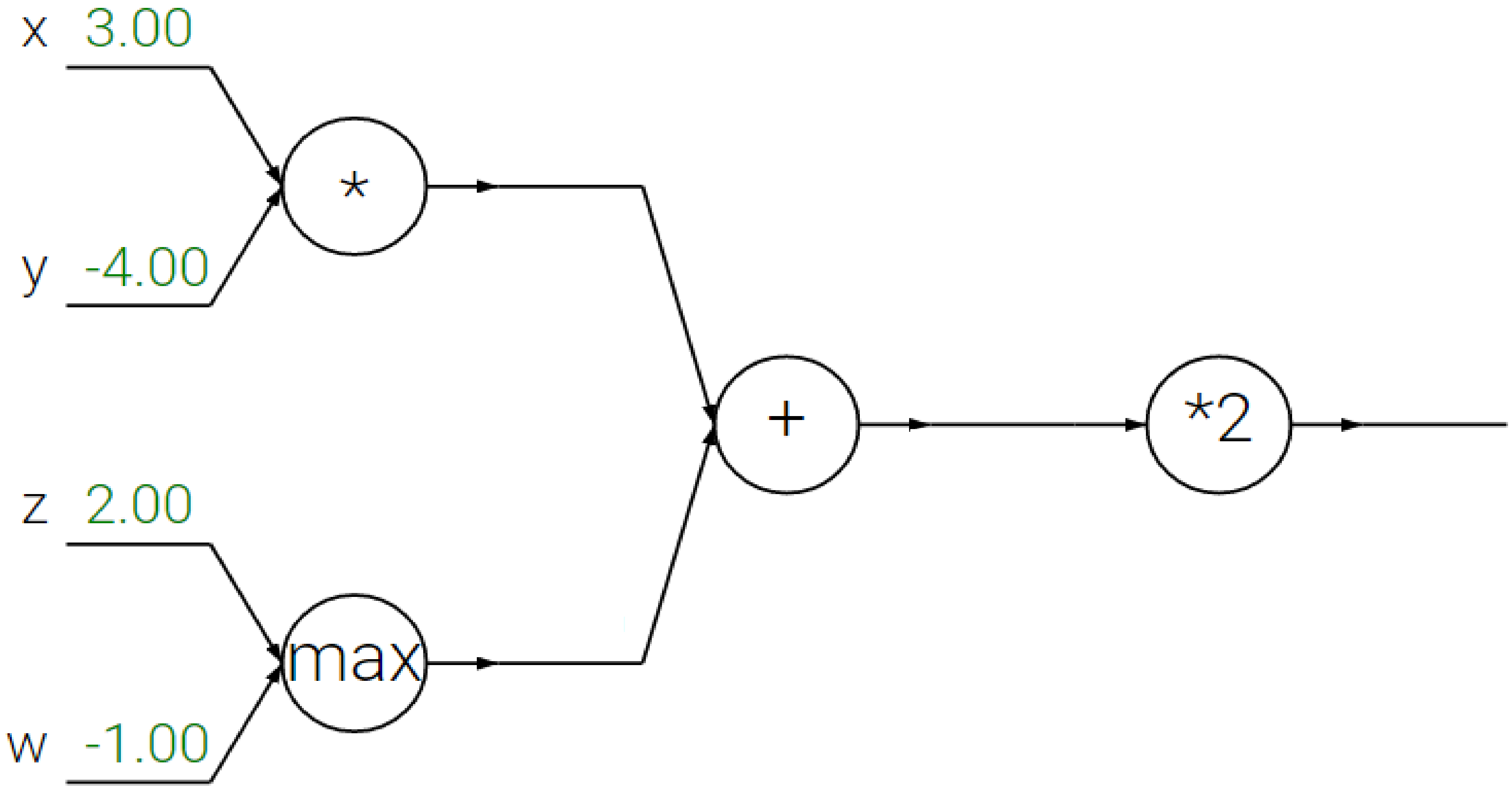
$$\frac{\partial q}{\partial x} = 1, \frac{\partial q}{\partial y} = 1$$

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



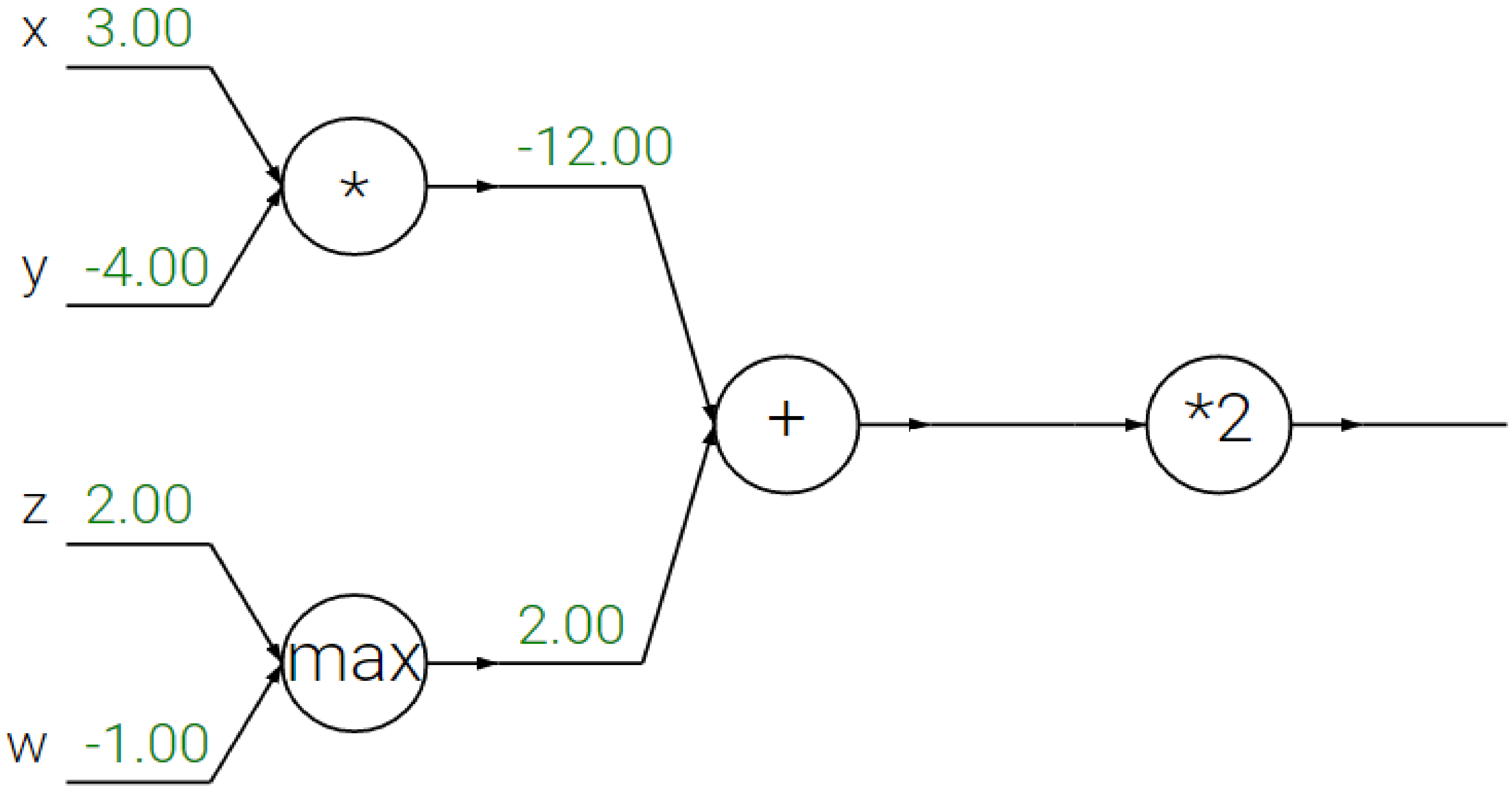
Source: http://cs231n.github.io

FORWARD AND BACKWARD PASS



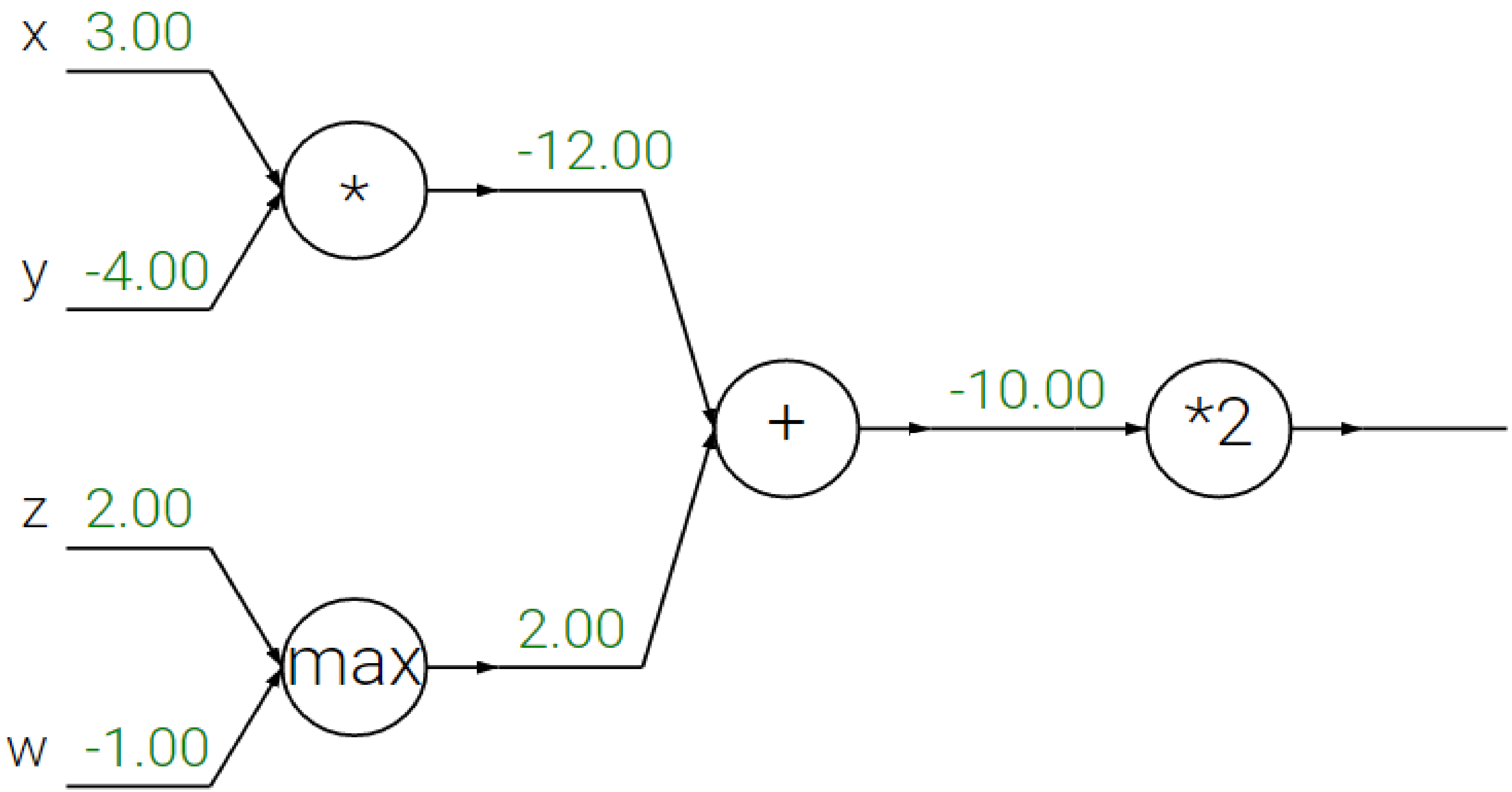
Source: http://cs231n.github.io

FORWARD AND BACKWARD PASS



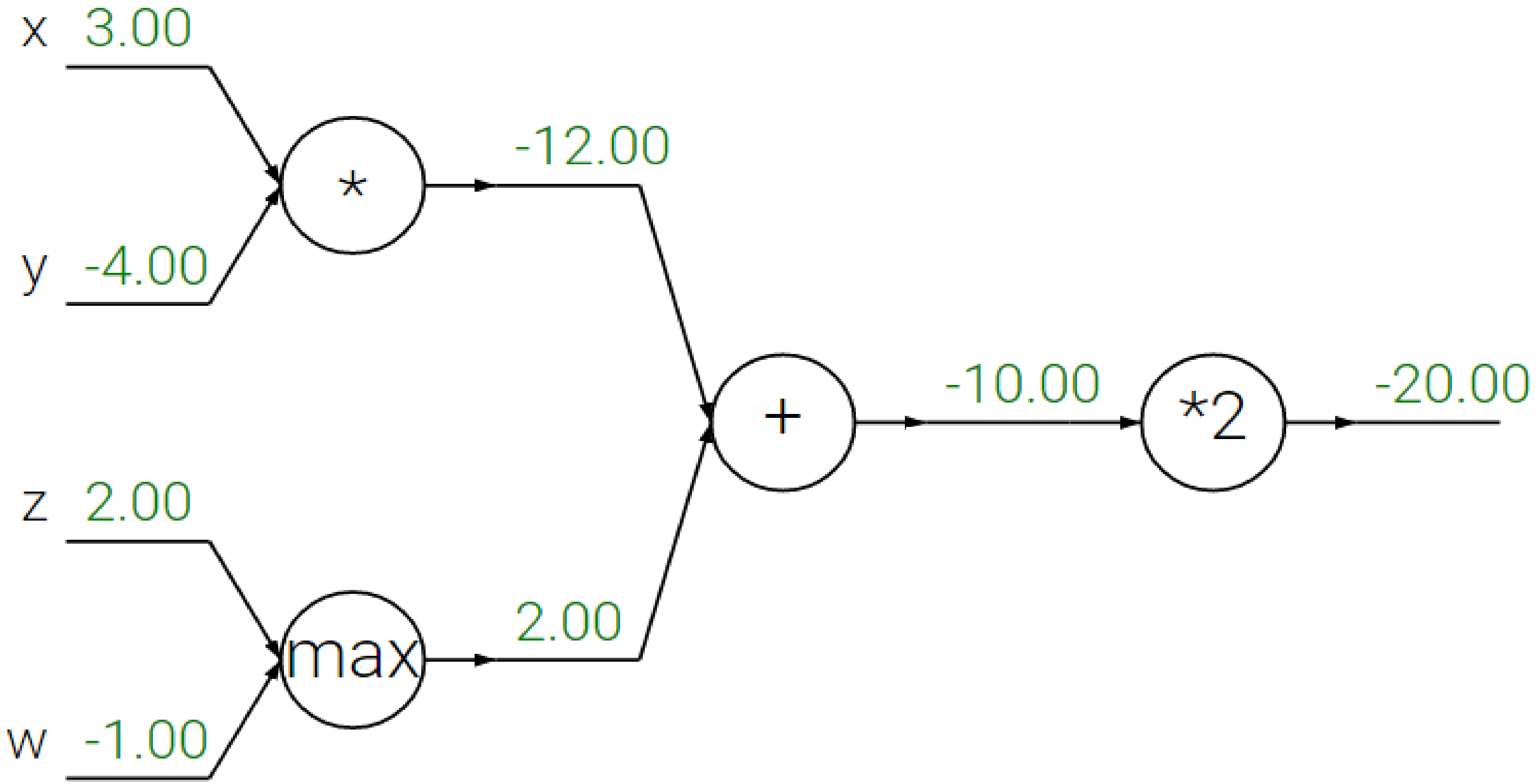
Source: http://cs231n.github.io

FORWARD AND BACKWARD PASS



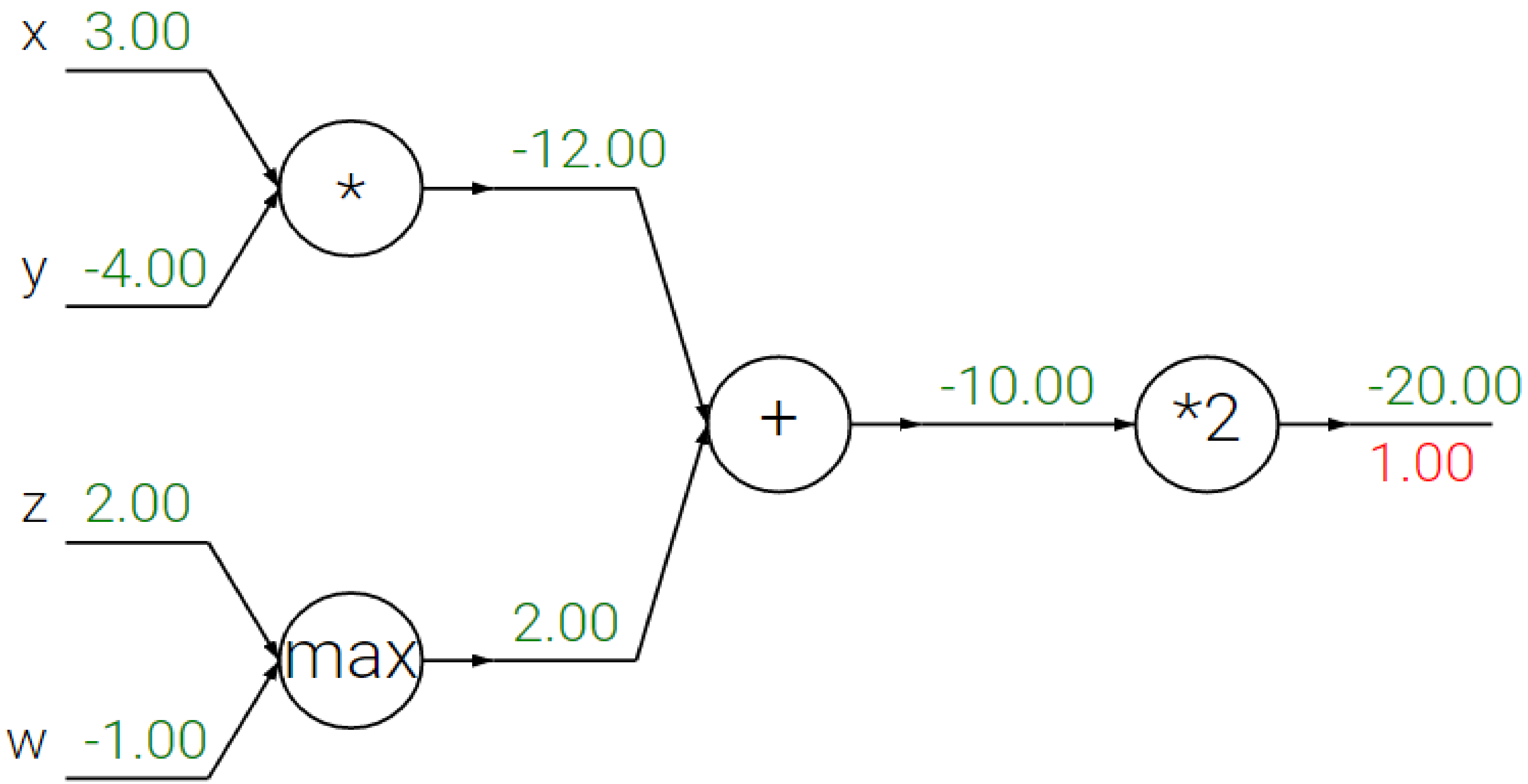
Source: http://cs231n.github.io

FORWARD AND BACKWARD PASS



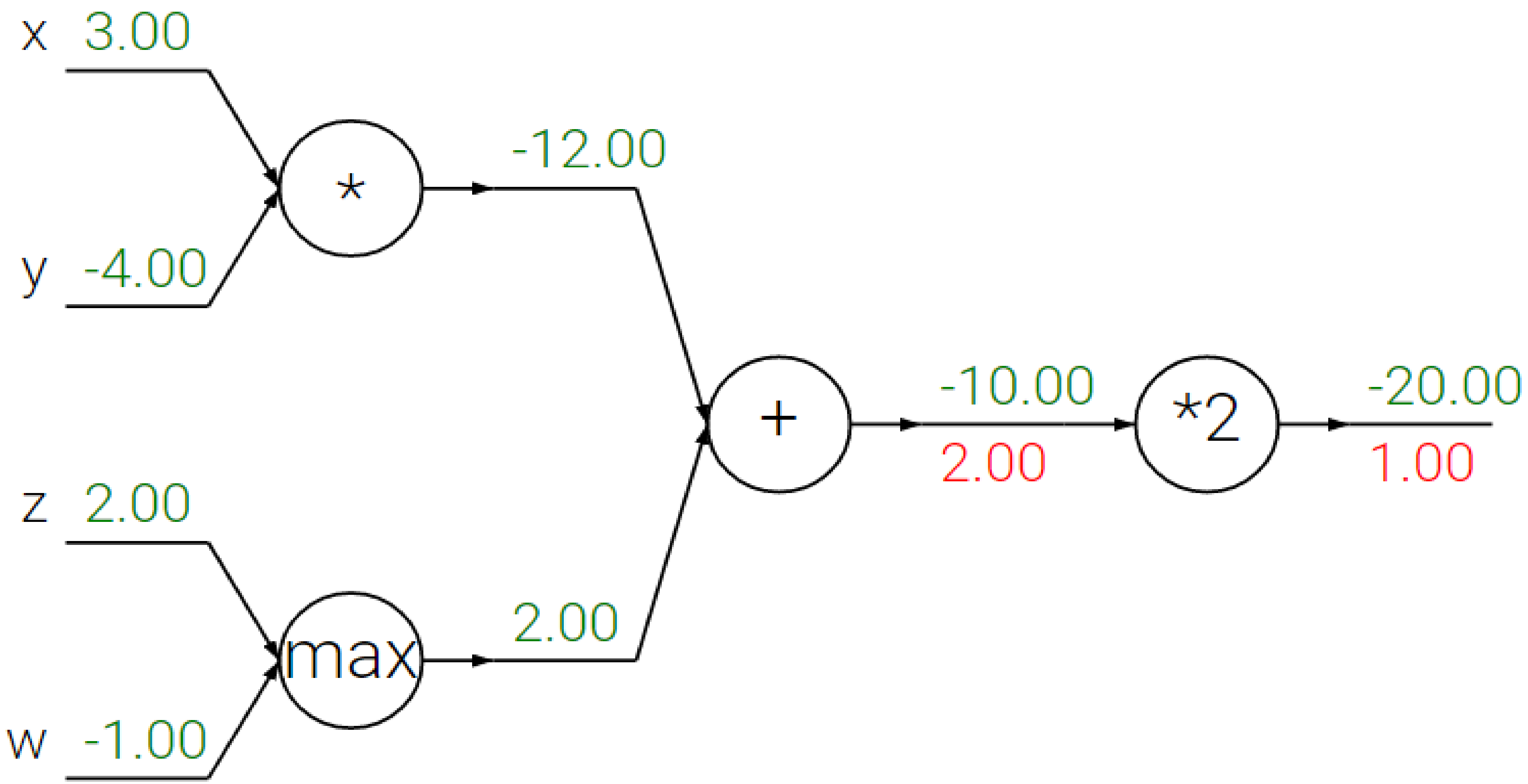
Source: http://cs231n.github.io

FORWARD AND BACKWARD PASS



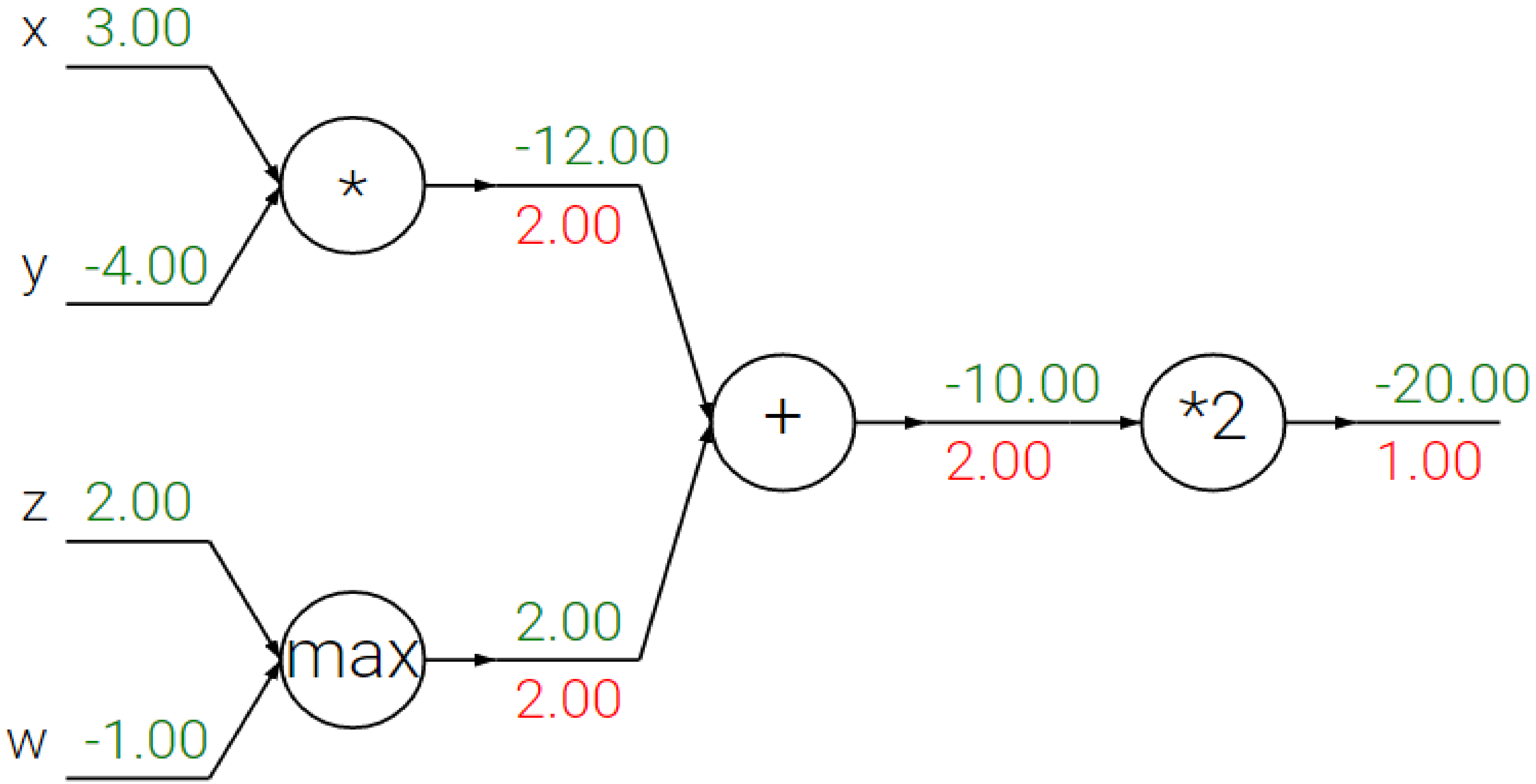
Source: http://cs231n.github.io

FORWARD AND BACKWARD PASS



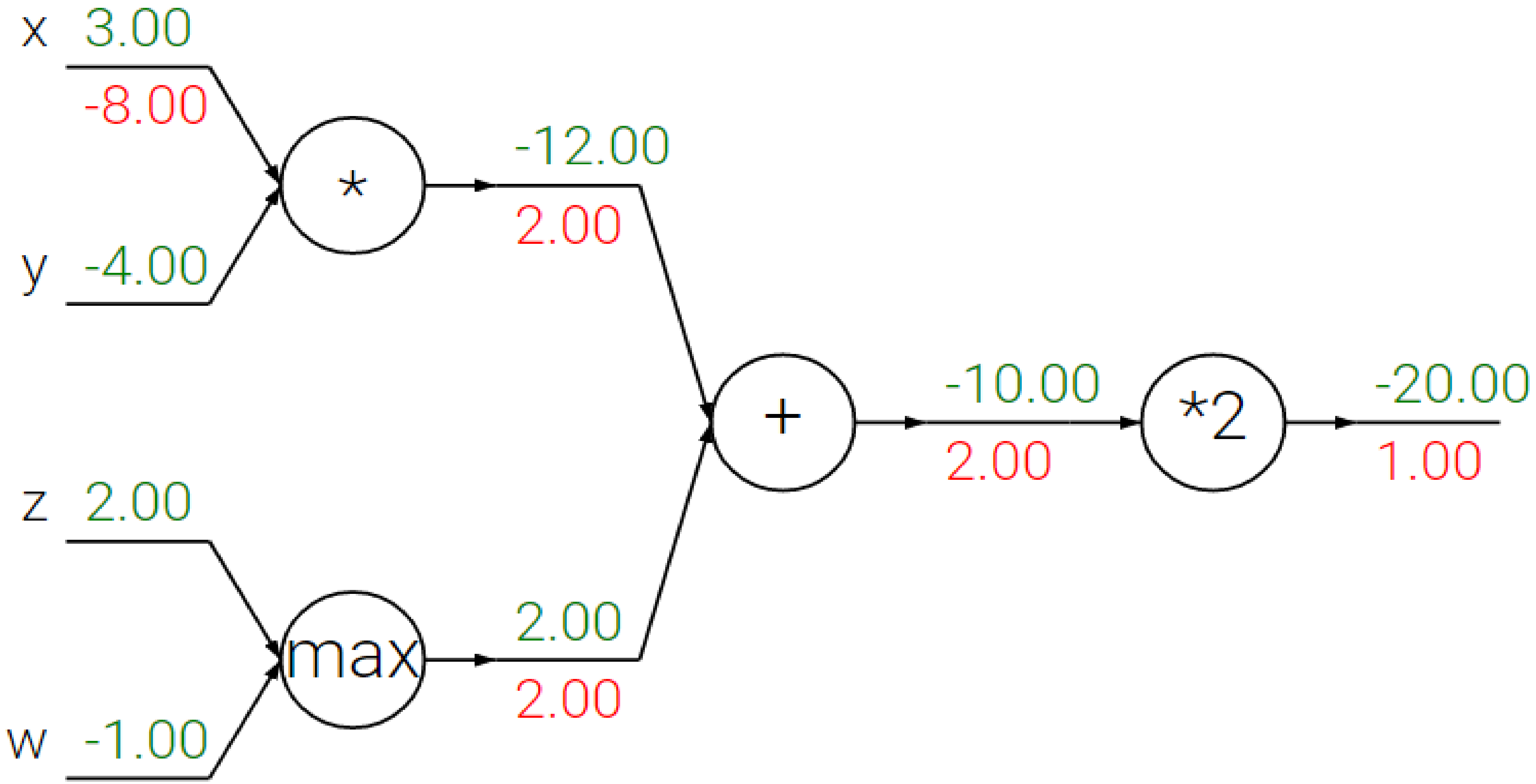
Source: http://cs231n.github.io

FORWARD AND BACKWARD PASS



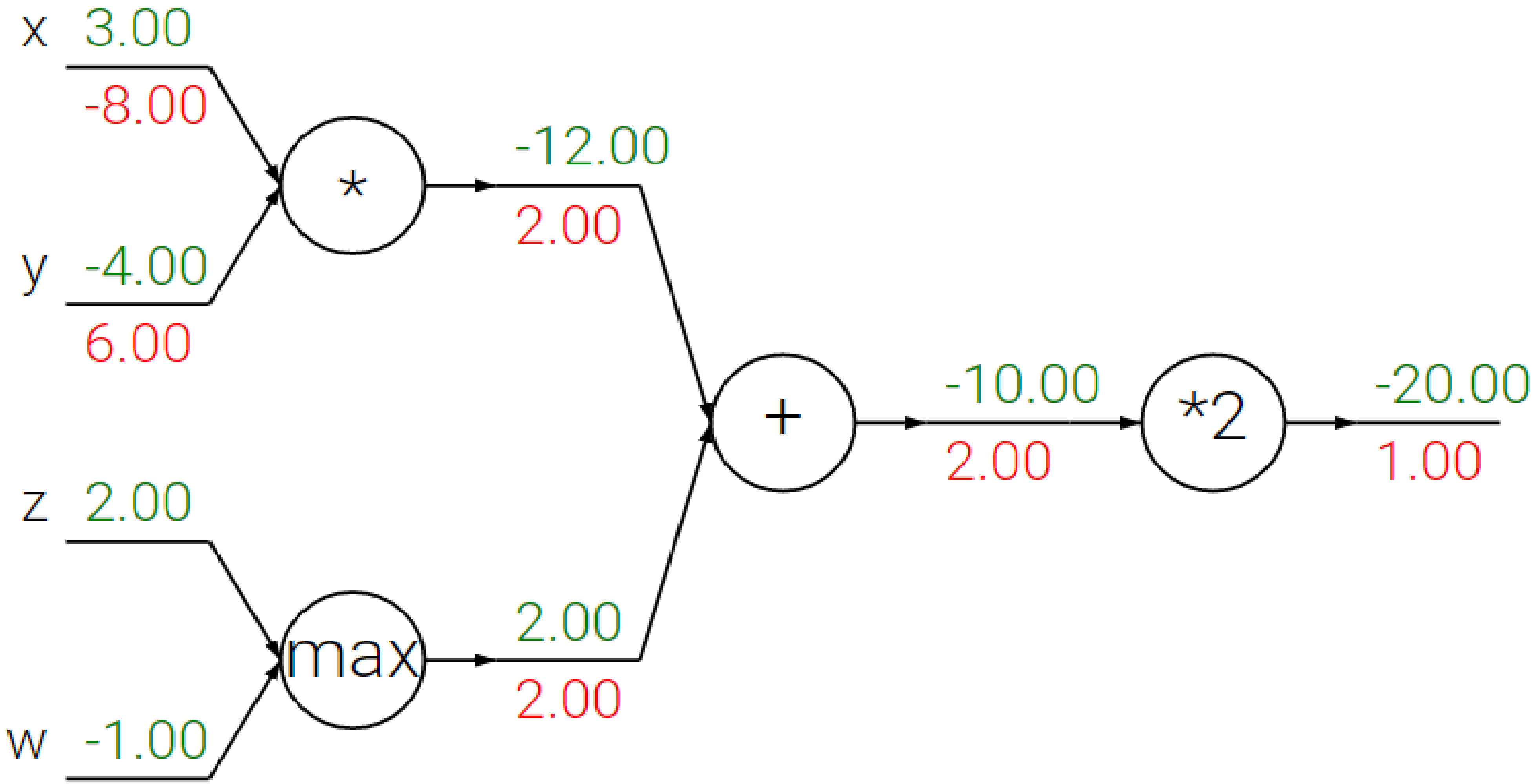
Source: http://cs231n.github.io

FORWARD AND BACKWARD PASS



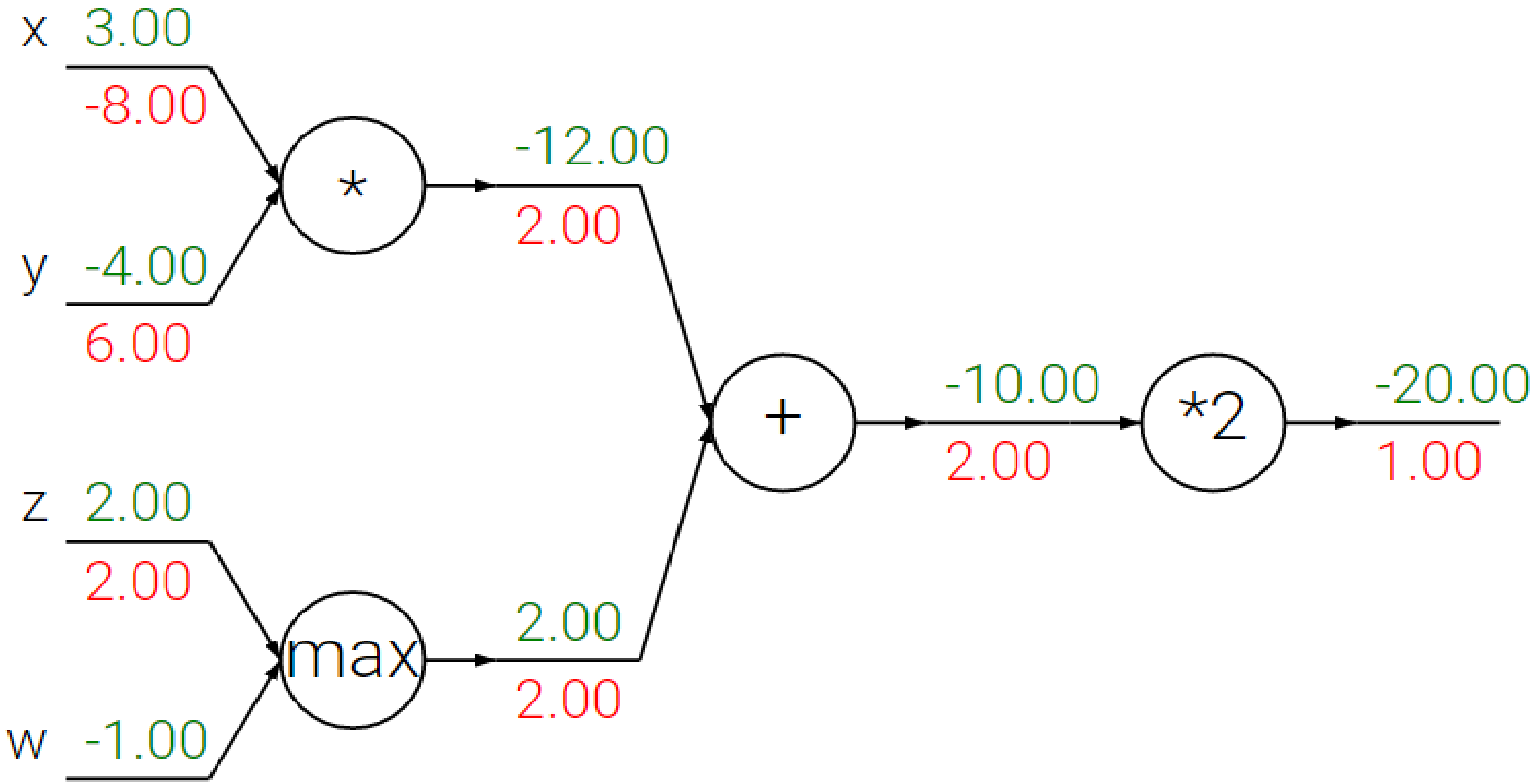
Source: http://cs231n.github.io

FORWARD AND BACKWARD PASS



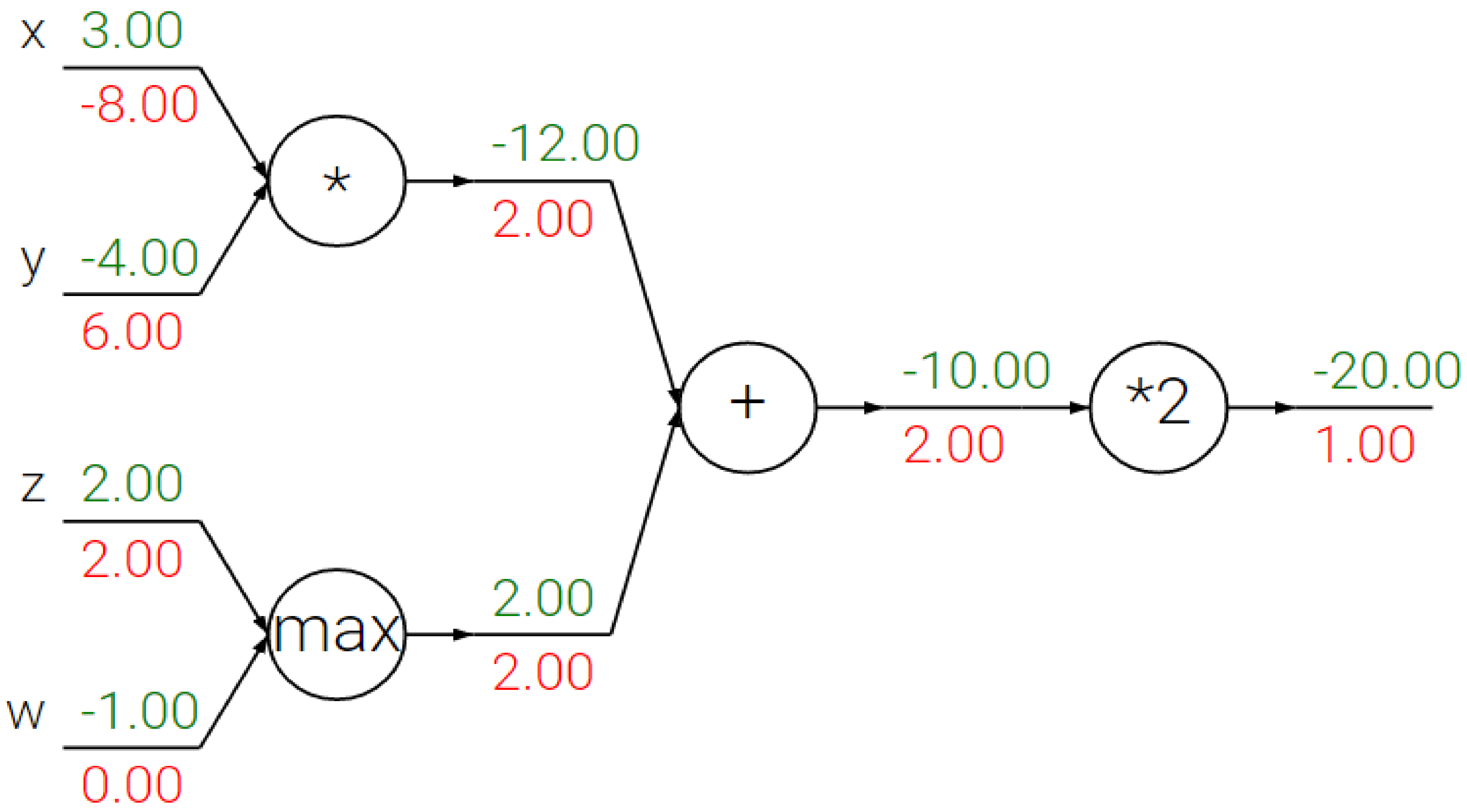
Source: http://cs231n.github.io

FORWARD AND BACKWARD PASS



Source: http://cs231n.github.io

FORWARD AND BACKWARD PASS



Source: http://cs231n.github.io

SIGMOID EXAMPLE

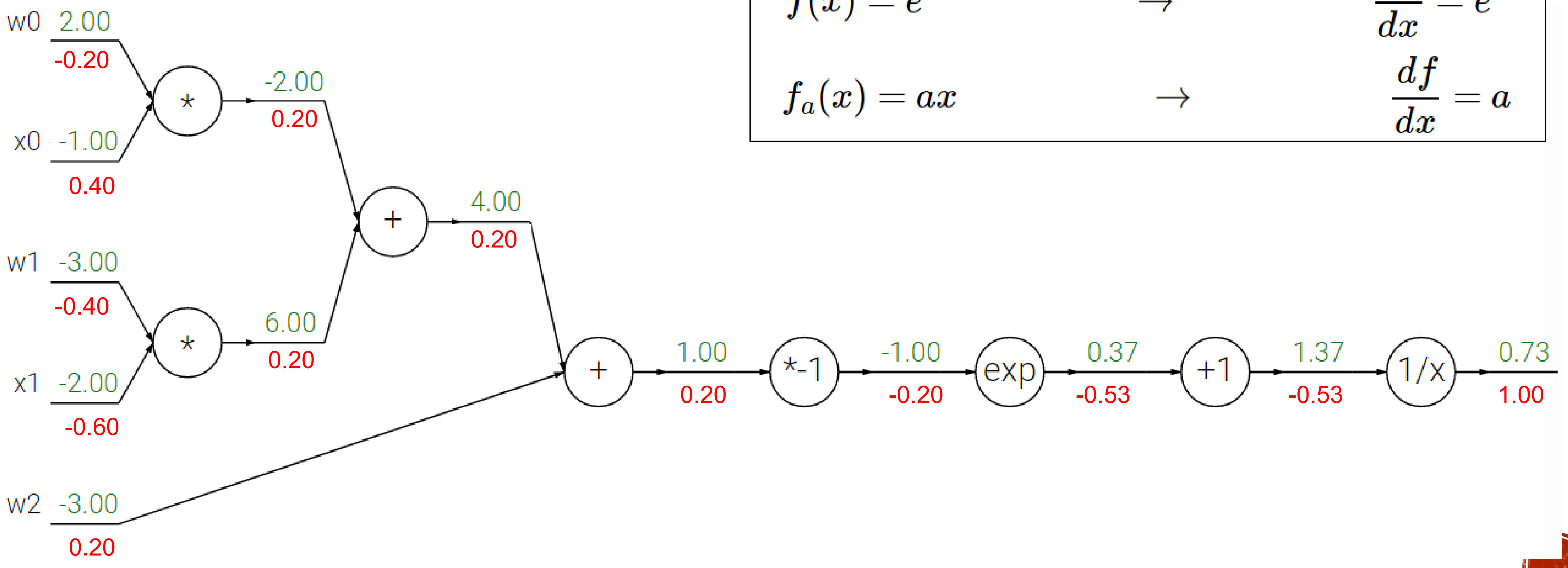
$$f(w,x) = \frac{1}{1+e^{-(w_0x_0+w_1x_1+w_2)}}$$

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

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

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

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



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

SVM LOSS: GRADIENT

SVM loss function for a single datapoint (without regularization):

$$L_i = \sum_{j \neq y_i} \left[\max(0, w_j^T x_i - w_{y_i}^T x_i + \Delta) \right]$$

Source: http://cs231n.github.io

SVM LOSS: GRADIENT

SVM loss function for a single datapoint (without regularization):

$$L_i = \sum_{j \neq y_i} \left[\max(0, w_j^T x_i - w_{y_i}^T x_i + \Delta) \right]$$

$$L_i = \sum_{j \neq y_i} \max(0, s_j - s_{y_i} + \Delta)$$

SVM LOSS: GRADIENT

SVM loss function for a single datapoint (without regularization):

$$L_i = \sum_{j \neq y_i} \left[\max(0, w_j^T x_i - w_{y_i}^T x_i + \Delta) \right]$$

$$L_i = \sum_{j \neq y_i} \max(0, s_j - s_{y_i} + \Delta)$$

Gradient w.r.t. w_{y_i} :

$$\nabla_{w_{y_i}} L_i = -\left(\sum_{j \neq y_i} 1(w_j^T x_i - w_{y_i}^T x_i + \Delta > 0) \right) x_i$$

Source: http://cs231n.github.io

SVM LOSS: GRADIENT

SVM loss function for a single datapoint (without regularization):

$$L_i = \sum_{j \neq y_i} \left[\max(0, w_j^T x_i - w_{y_i}^T x_i + \Delta) \right]$$

$$L_i = \sum_{j \neq y_i} \max(0, s_j - s_{y_i} + \Delta)$$

Gradient w.r.t. w_{y_i} :

$$\nabla_{w_{y_i}} L_i = - \left(\sum_{j \neq y_i} 1(w_j^T x_i - w_{y_i}^T x_i + \Delta > 0) \right) x_i$$

Count of the number of classes that didn't meet the desired margin

Source: http://cs231n.github.io

SVM LOSS: GRADIENT

SVM loss function for a single datapoint (without regularization):

$$L_i = \sum_{j \neq y_i} \left[\max(0, w_j^T x_i - w_{y_i}^T x_i + \Delta) \right]$$

$$L_i = \sum_{j \neq y_i} \max(0, s_j - s_{y_i} + \Delta)$$

Gradient w.r.t. w_{y_i} :

$$\nabla_{w_{y_i}} L_i = -\left(\sum_{j \neq y_i} \mathbb{1}(w_j^T x_i - w_{y_i}^T x_i + \Delta > 0) \right) x_i$$

Count of the number of classes that didn't meet the desired margin

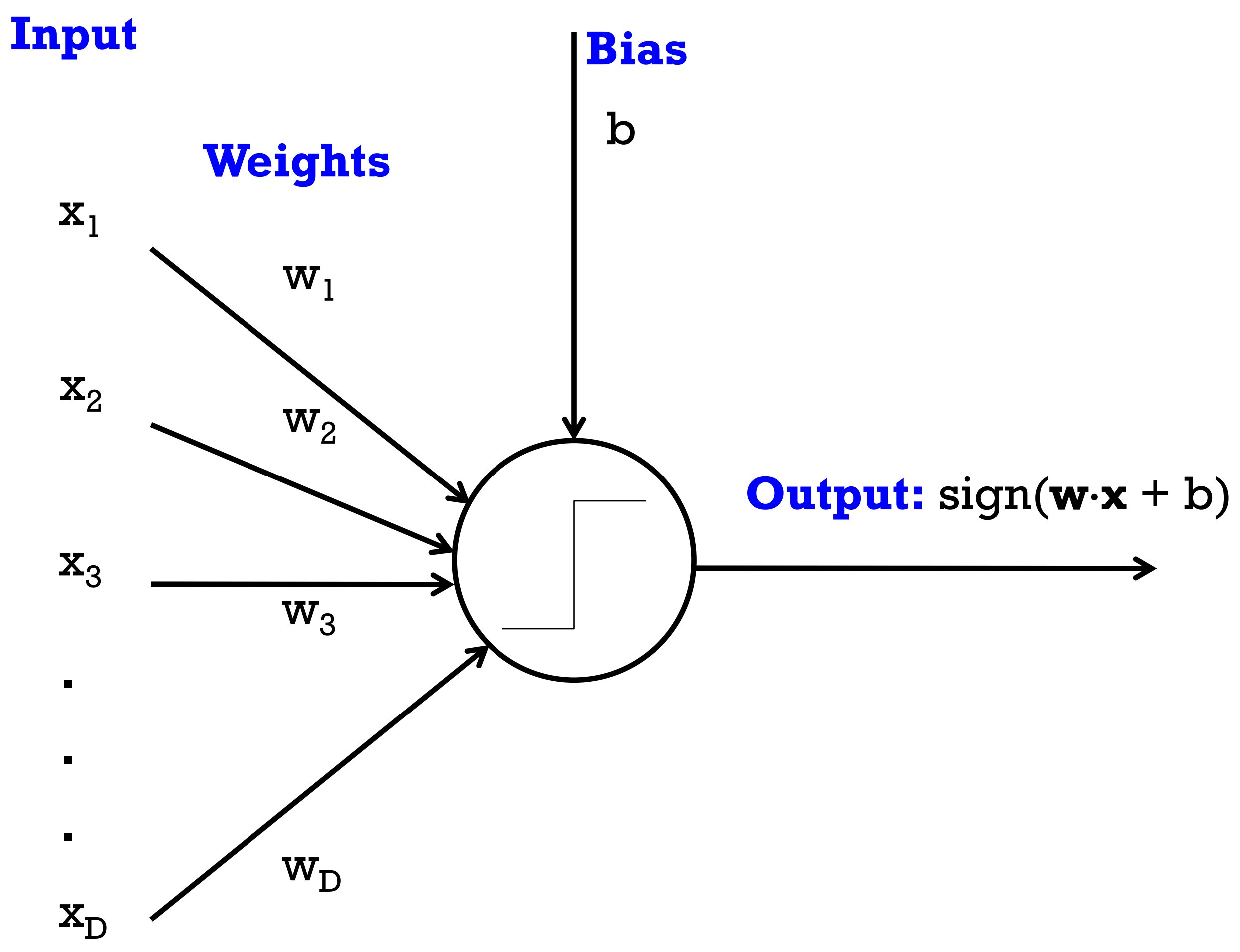
Gradient for the other rows where $j \neq y_i$:

$$\nabla_{w_j} L_i = \mathbb{1}(w_j^T x_i - w_{y_i}^T x_i + \Delta > 0) x_i$$

Source: http://cs231n.github.io

PERCEPTRON

- Supervised learning of binary classifier



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

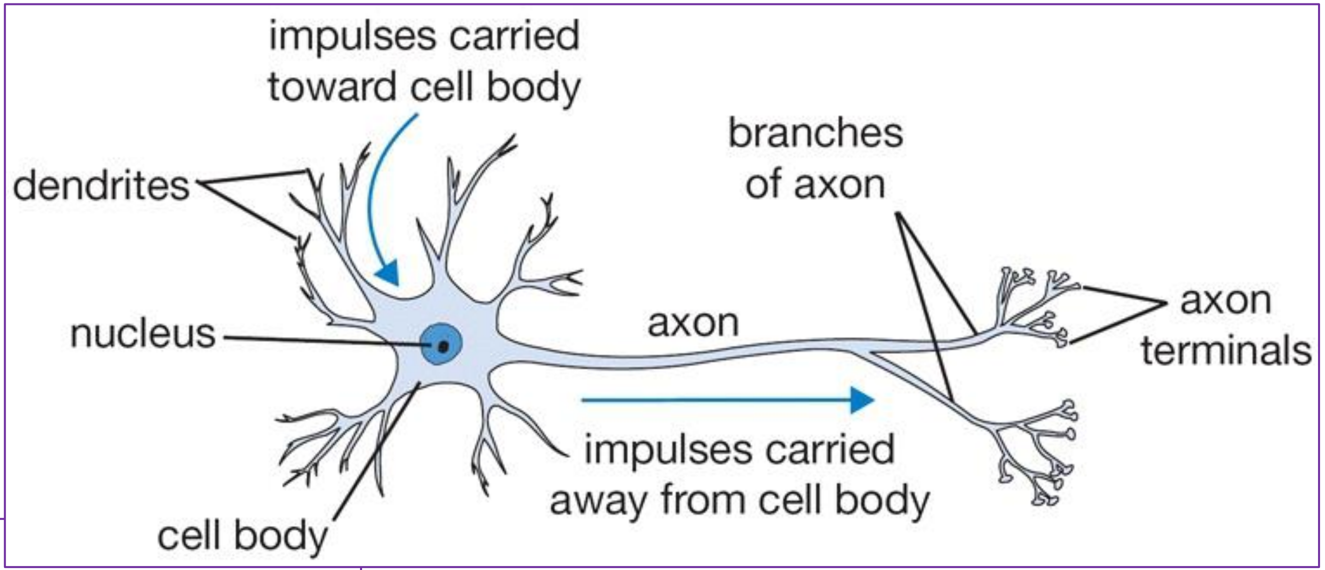
Probability of the other class would be:

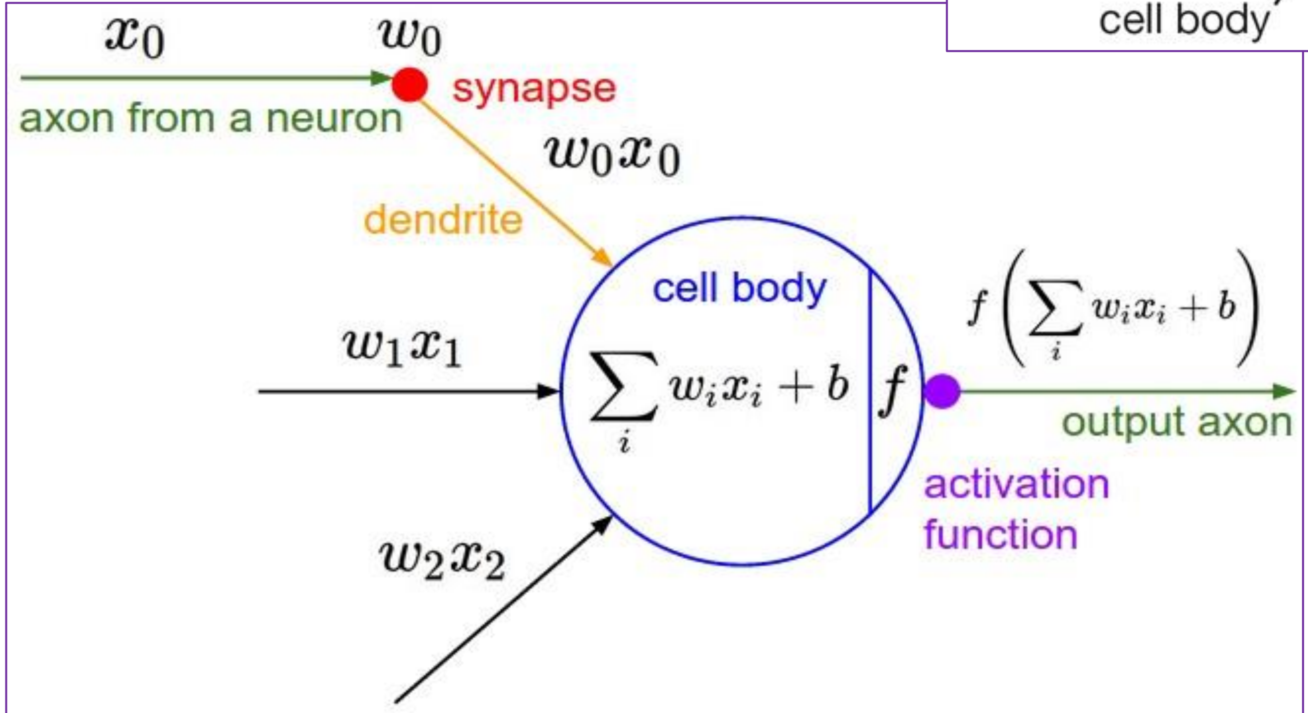
$$P(y_i = 0 \mid x_i; w) = 1 - P(y_i = 1 \mid x_i; w)$$

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

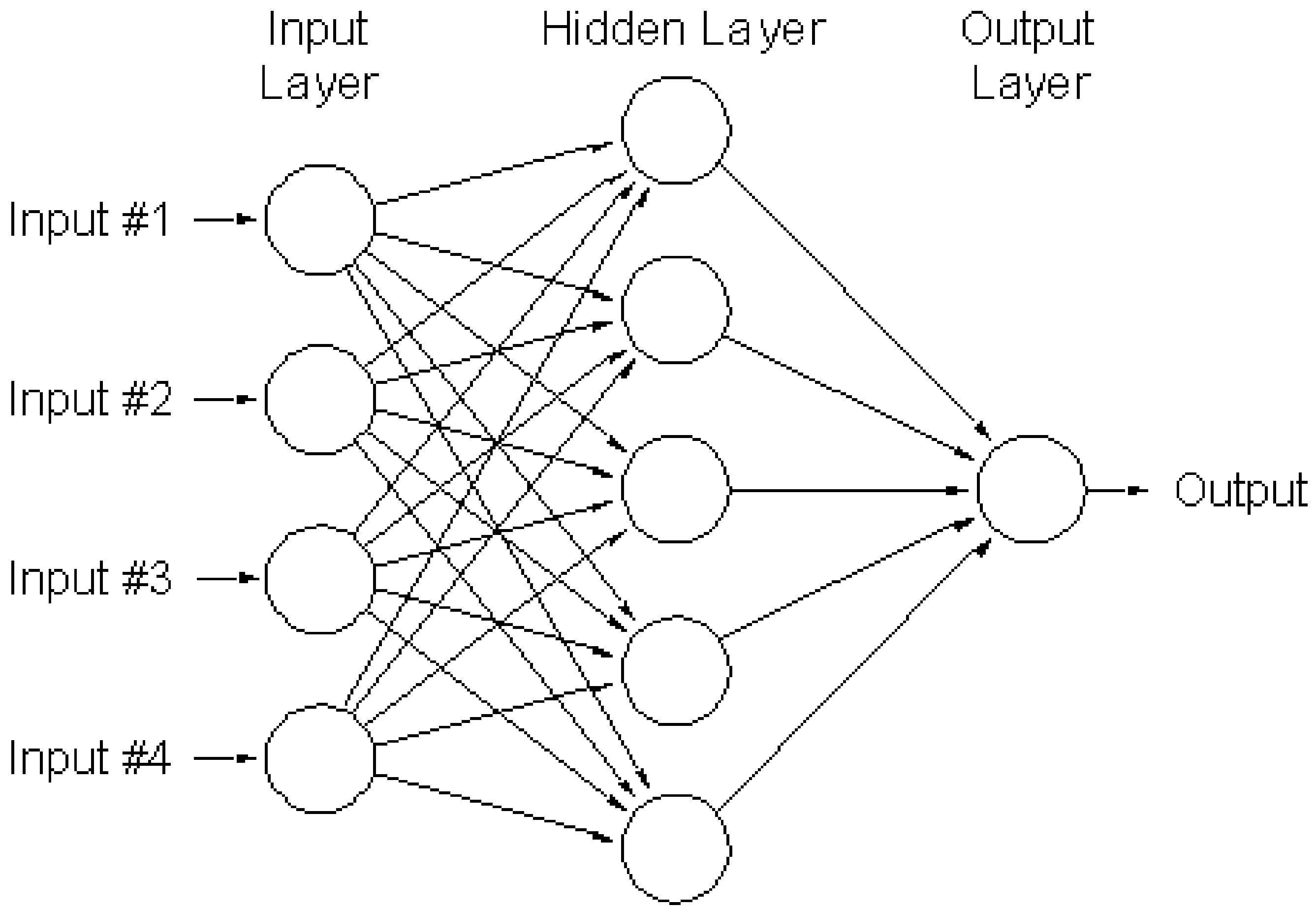




Source: http://cs231n.github.io

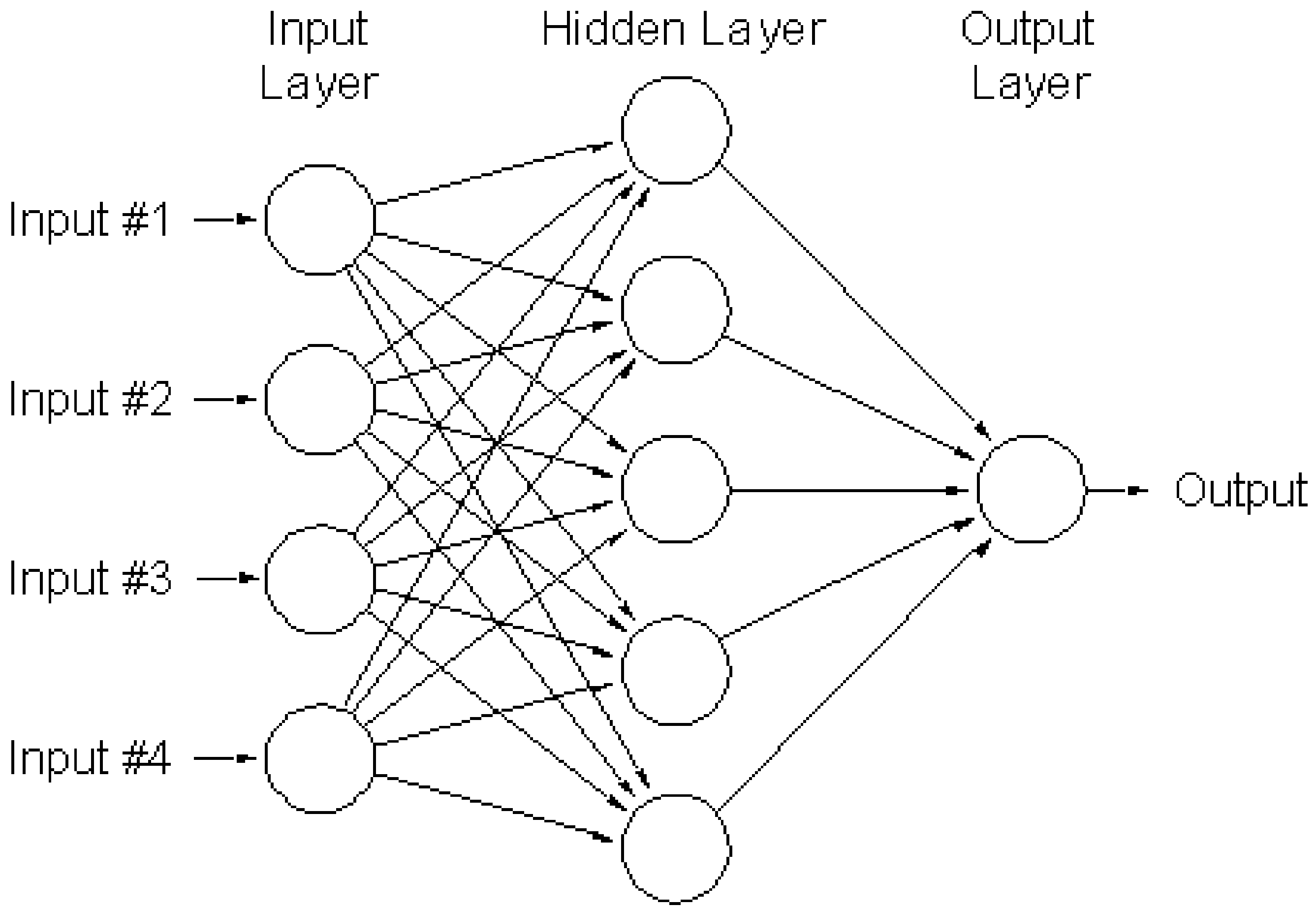
MULTI-LAYER NEURAL NETWORKS

- Network with a hidden layer:



MULTI-LAYER NEURAL NETWORKS

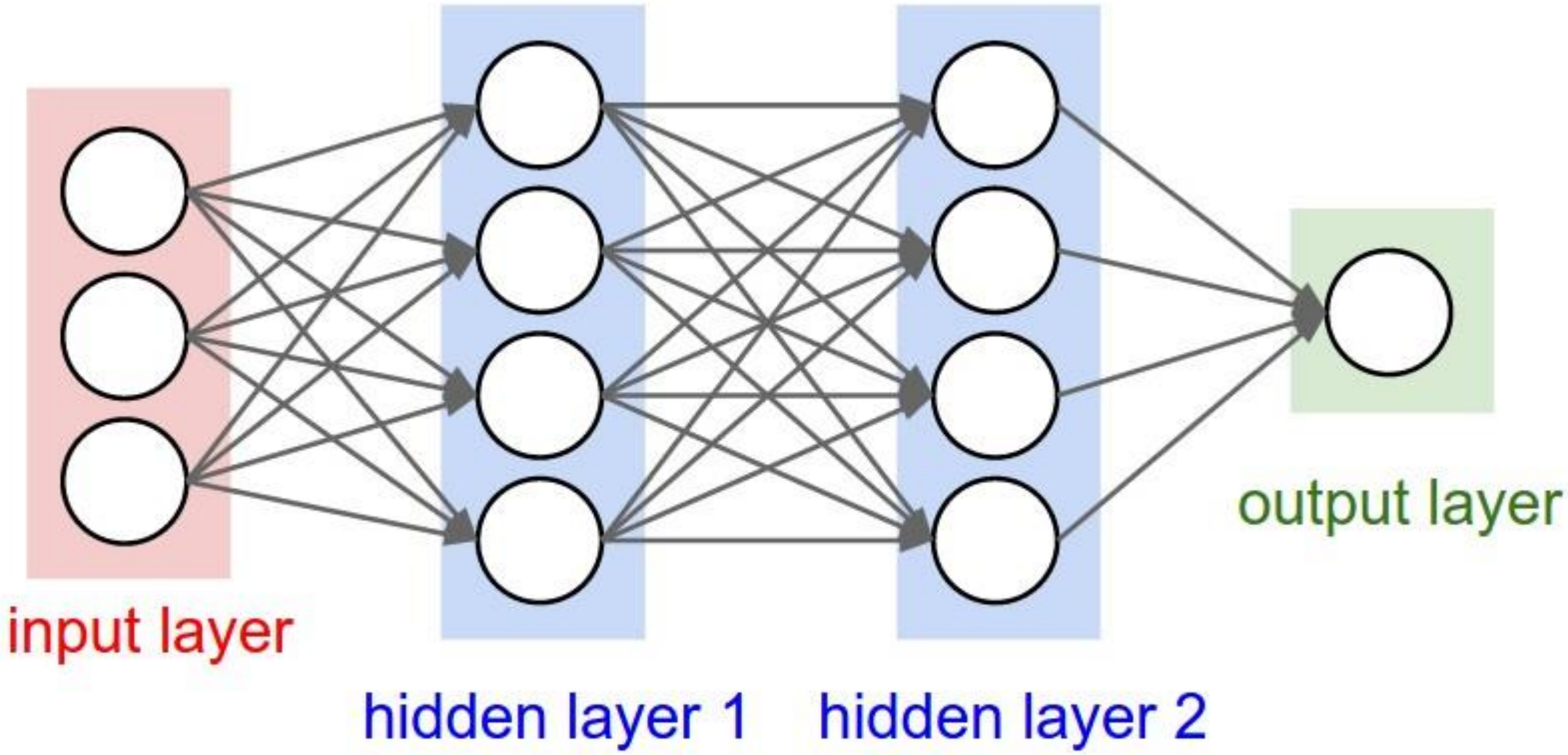
- Network with a hidden layer:



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

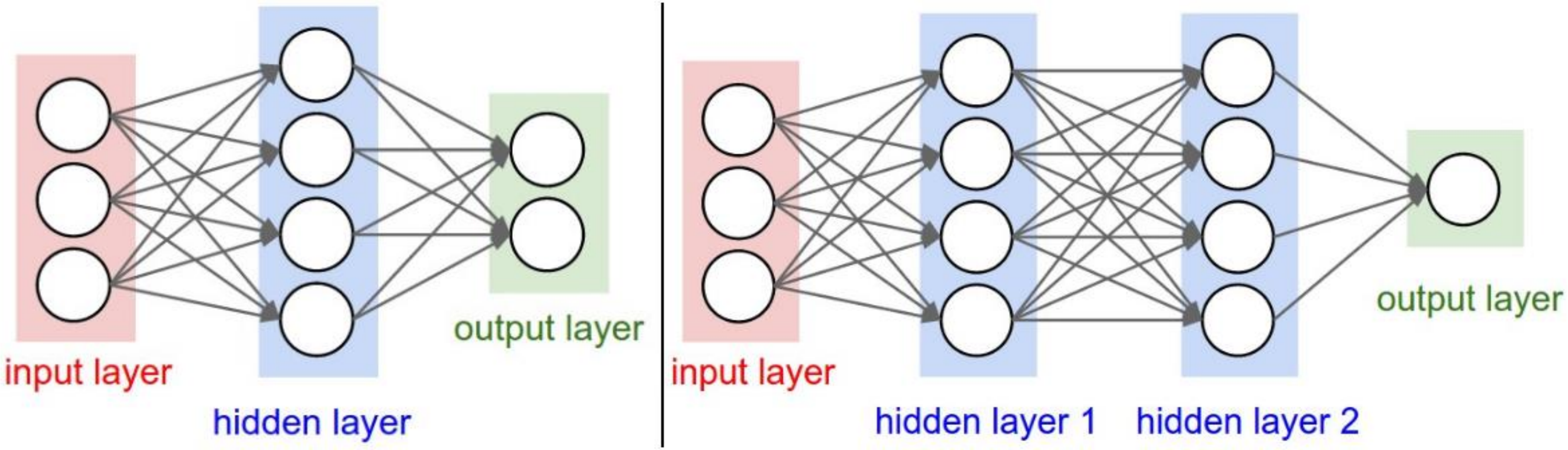
MULTI-LAYER NEURAL NETWORKS

- Beyond a single hidden layer:



Source: http://cs231n.github.io

SIZING NEURAL NETWORKS



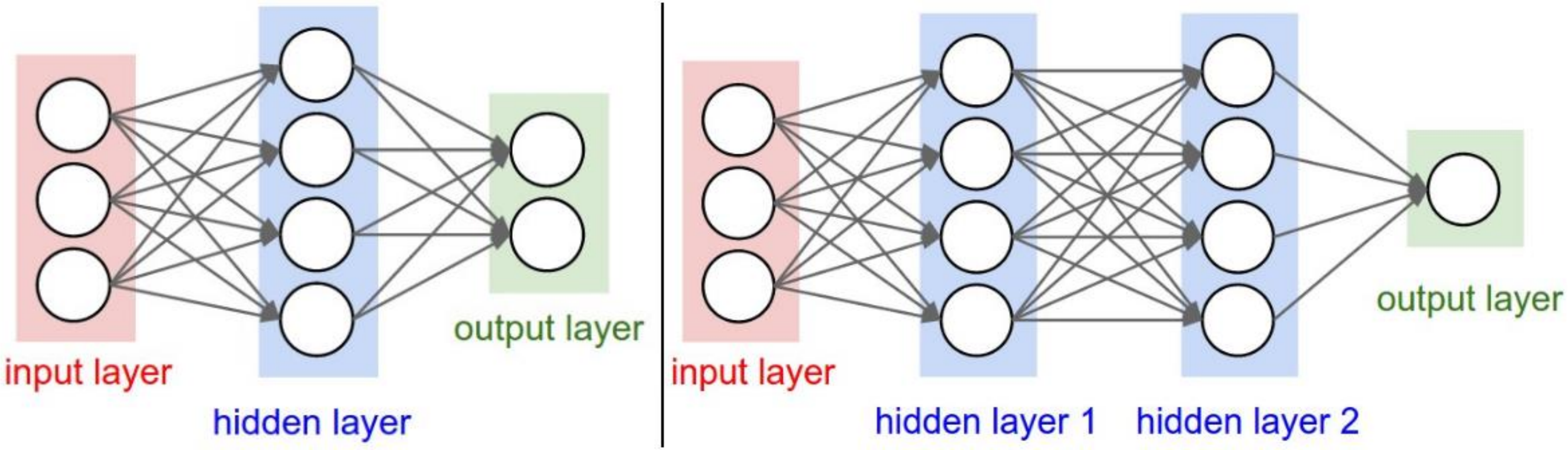
First network (left):

No. of neurons (not counting the inputs):

No. of learnable parameters:

Source: http://cs231n.github.io

SIZING NEURAL NETWORKS



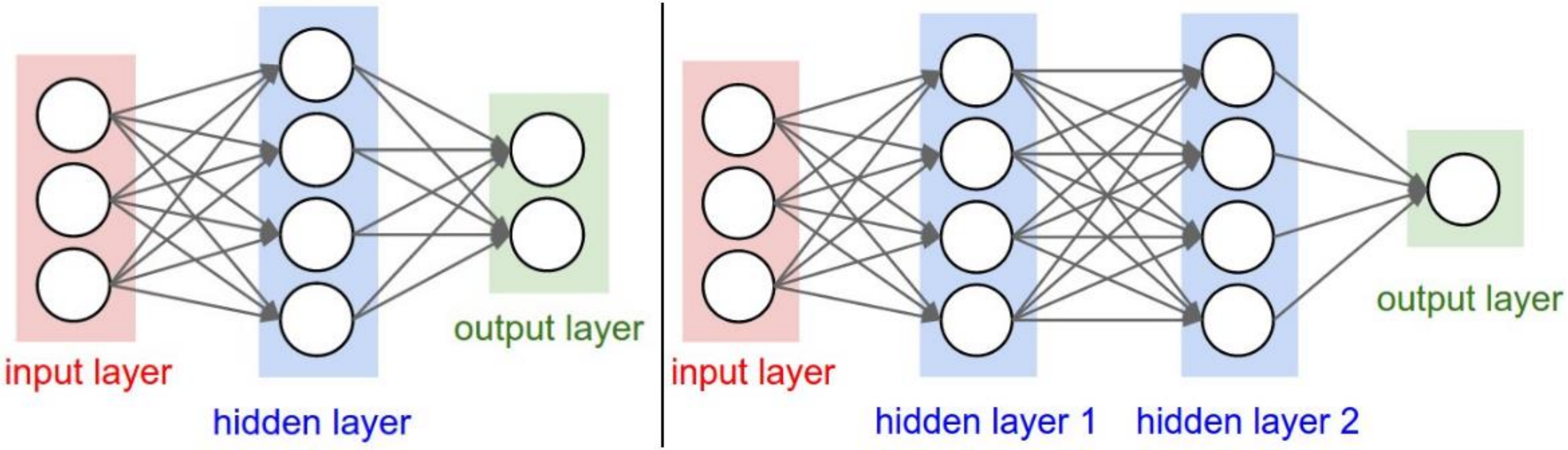
First network (left):

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

No. of learnable parameters:

Source:
http://cs231n.github.io

SIZING NEURAL NETWORKS



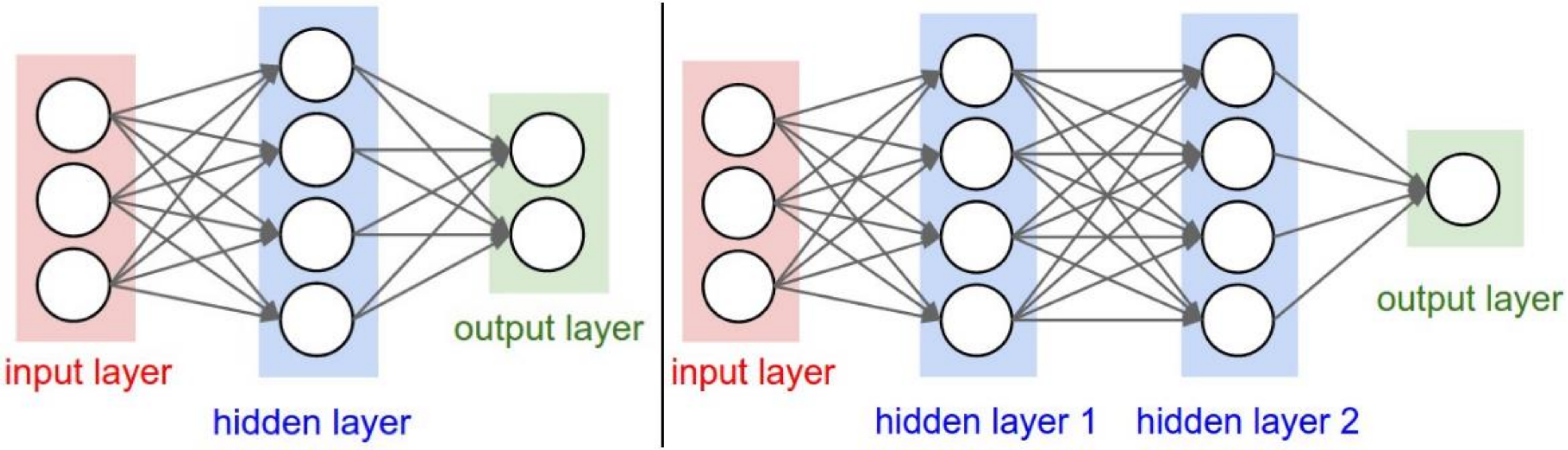
First network (left):

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.

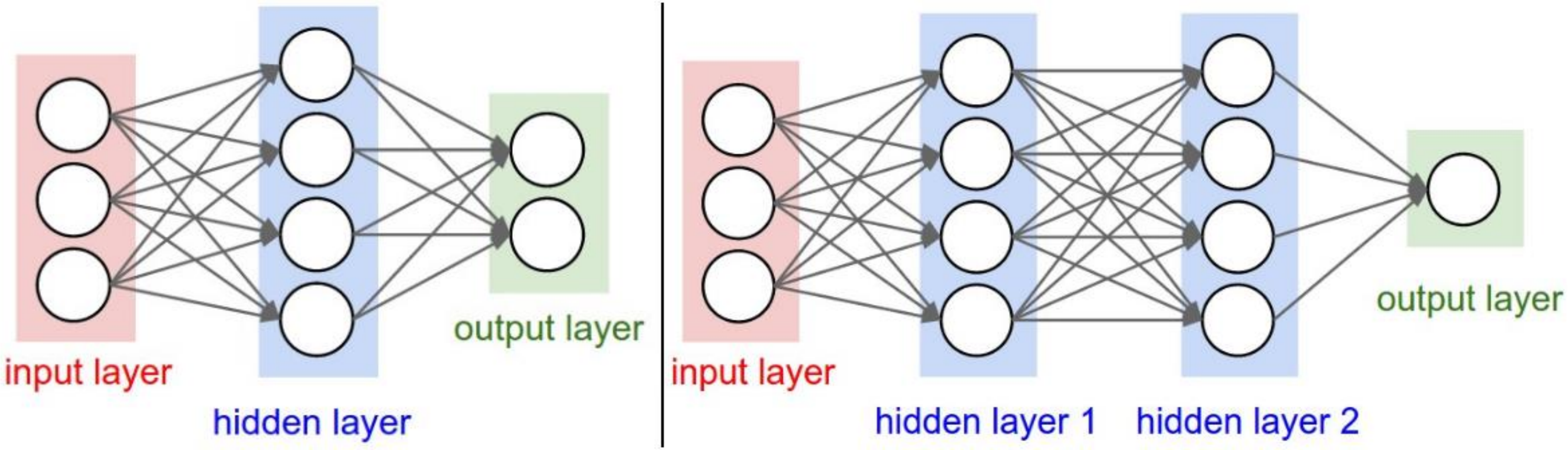
Second network (right):

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.

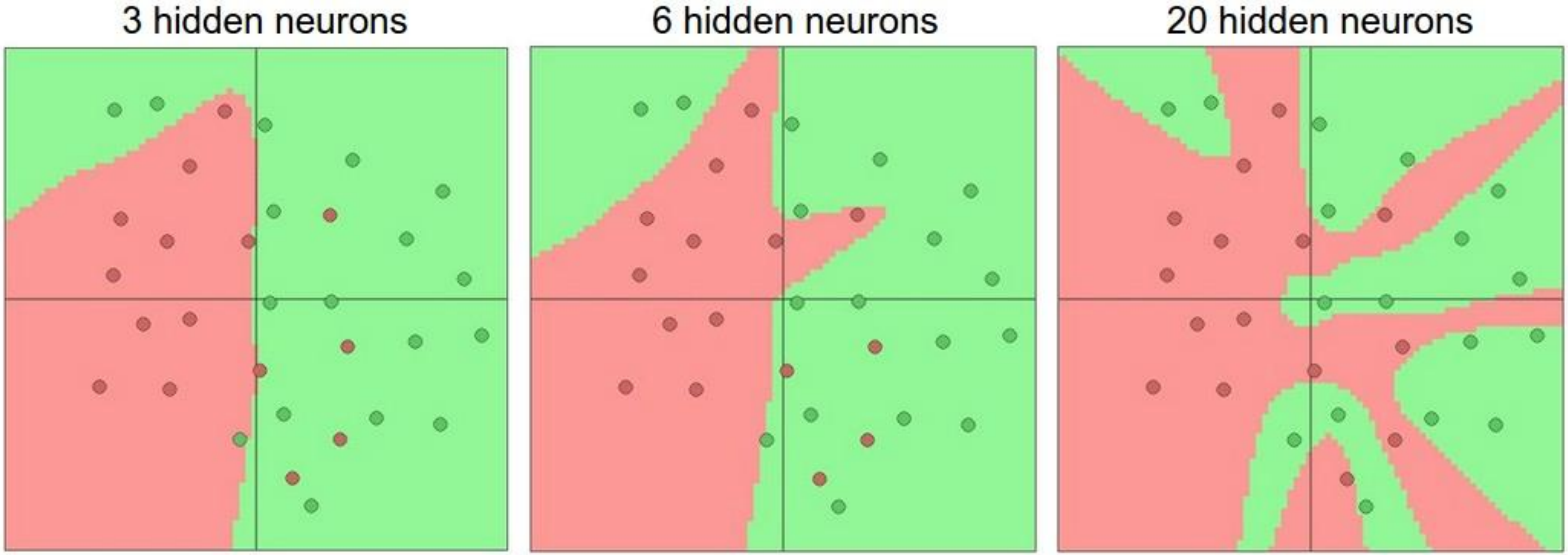
Second network (right):

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.

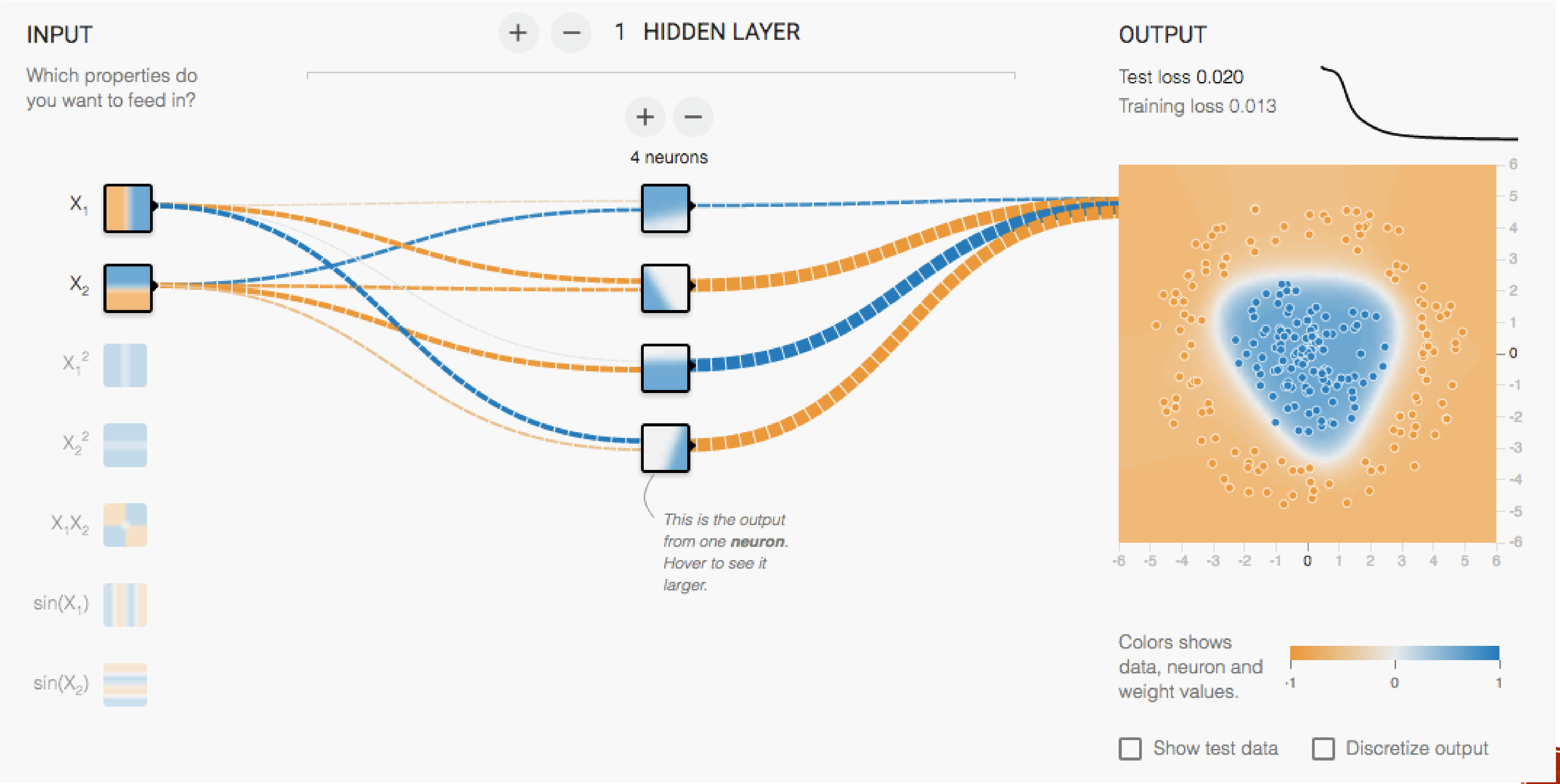
Source: http://cs231n.github.io

MULTI-LAYER NEURAL NETWORKS



Source: http://cs231n.github.io

MULTI-LAYER NETWORK DEMO



http://playground.tensorflow.org/

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

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)

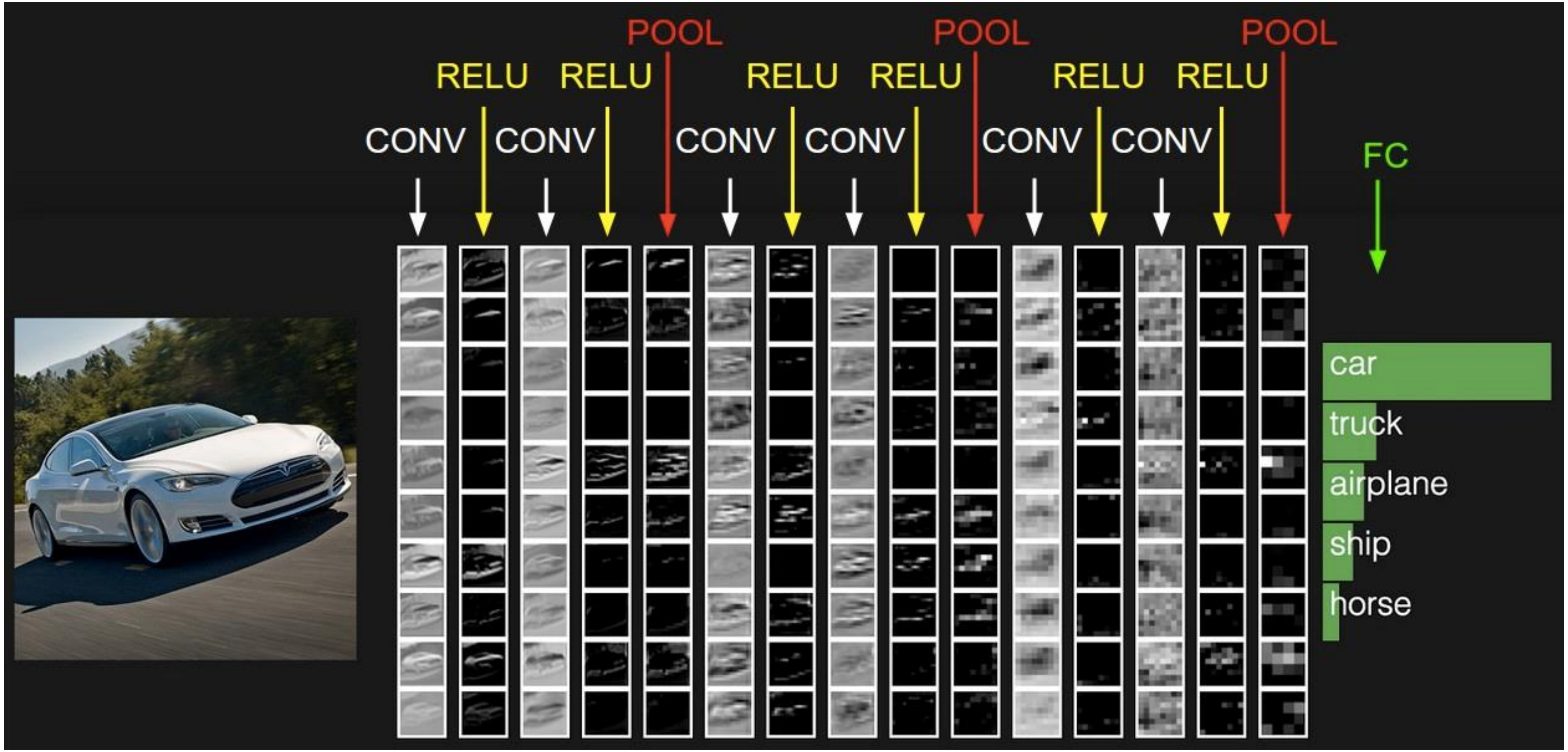
ACKNOWLEDGEMENT

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

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

NEXT LECTURE

Convolutional Neural Networks



Questions?