Computer Vision

Neural Networks

Dr. Mrinmoy Ghorai

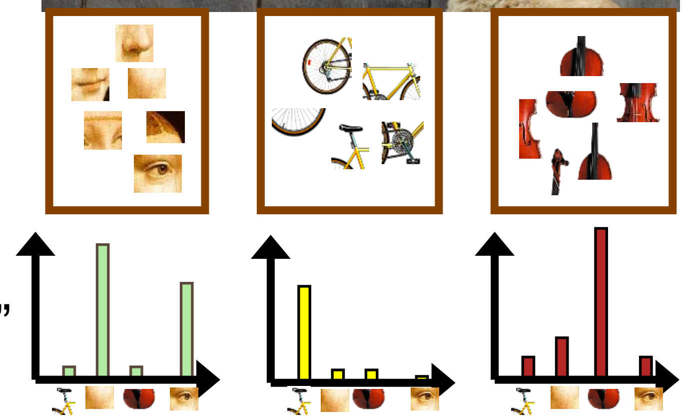
Indian Institute of Information Technology
Sri City, Chittoor

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

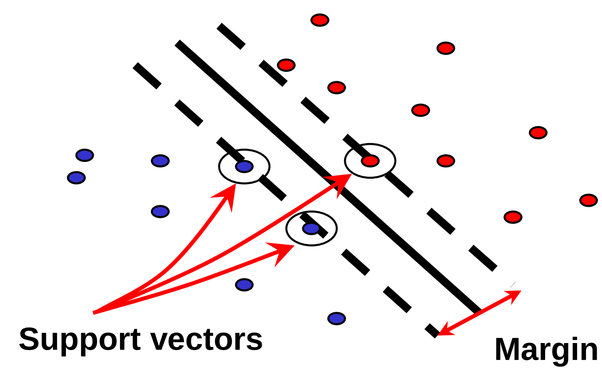
Learn “visual vocabulary”

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



Previous Class

Two key components in context of the image classification

1. A (parameterized) score function:

Mapping the raw image pixels/features to class scores
(e.g. a linear function)

Previous Class

Two key components in context of the image classification

1. A (parameterized) score function:

Mapping the raw image pixels/features to class scores
(e.g. a linear function)

2. A loss function:

Measures the goodness of parameter values in terms of how well it performs over the training data
(e.g. Softmax/SVM)

Previous Class

A linear function: $f(x_i, W) = Wx_i$

Previous Class

A linear function: $f(x_i, W) = Wx_i$

Loss: $L = \underbrace{\frac{1}{N}\sum_i L_i}_{\text{data loss}} + \underbrace{\lambda R(W)}_{\text{regularization loss}}$ $\quad R(W) = \sum_k \sum_l W_{k,l}^2$

Previous Class

A linear function: $f(x_i, W) = Wx_i$

Loss: $L = \underbrace{\frac{1}{N}\sum_i L_i}_{\text{data loss}} + \underbrace{\lambda R(W)}_{\text{regularization loss}}$ $R(W) = \sum_k \sum_l W_{k,l}^2$

SVM Loss:
Hinge Loss
Max-margin loss

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

Previous Class

A linear function: $f(x_i, W) = Wx_i$

Loss: $L = \underbrace{\frac{1}{N}\sum_i L_i}_{\text{data loss}} + \underbrace{\lambda R(W)}_{\text{regularization loss}}$ $R(W) = \sum_k \sum_l W_{k,l}^2$

SVM Loss:
Hinge Loss
Max-margin loss

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

Softmax Loss:
Cross-entropy loss

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

Today’s class

Optimization

Gradient Descent & Back propagation

Perceptron

Update rule

Neural networks

Optimization

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

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.

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+δW*** 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+δW*** 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**.

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
```

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:

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

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

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:

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

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

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

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:

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

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

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

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:

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

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

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

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

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:

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

Source: http://cs231n.github.io

Compound expressions with chain rule

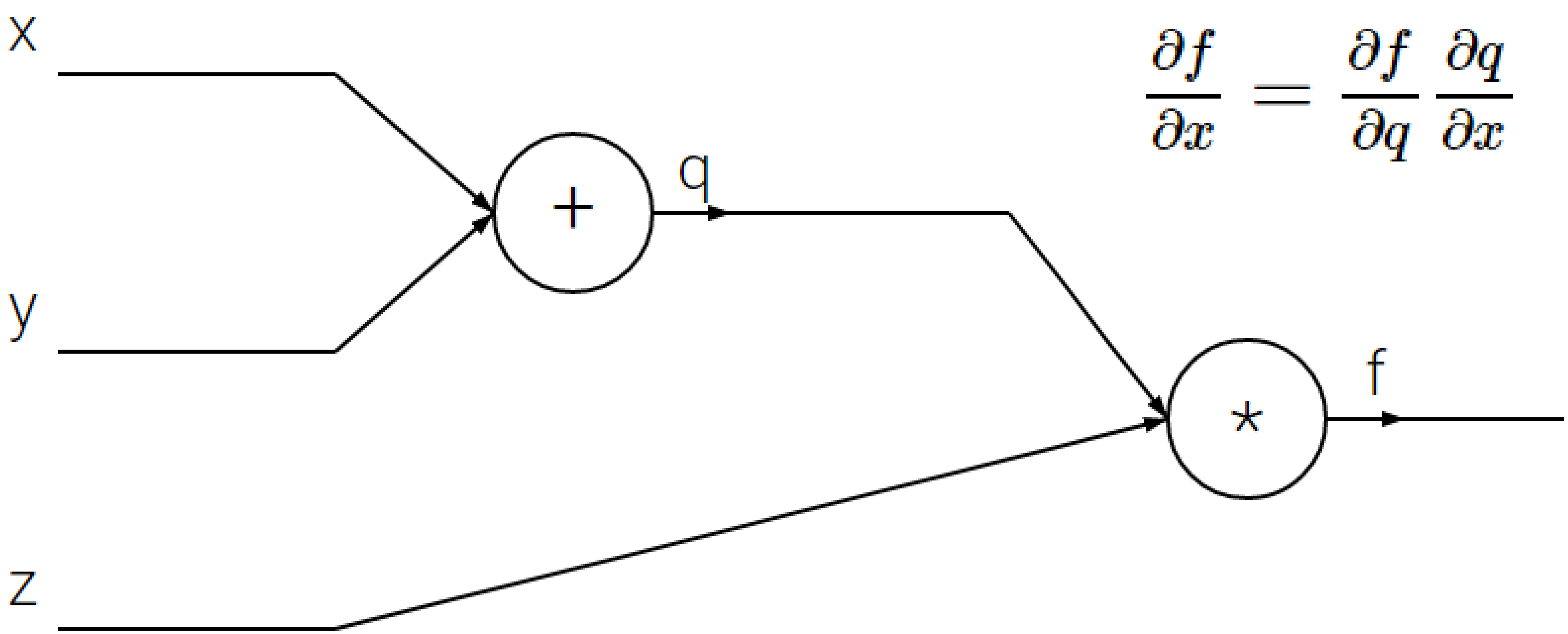
$$f(x, y, z) = (x + y)z$$

$q = x + y$ 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

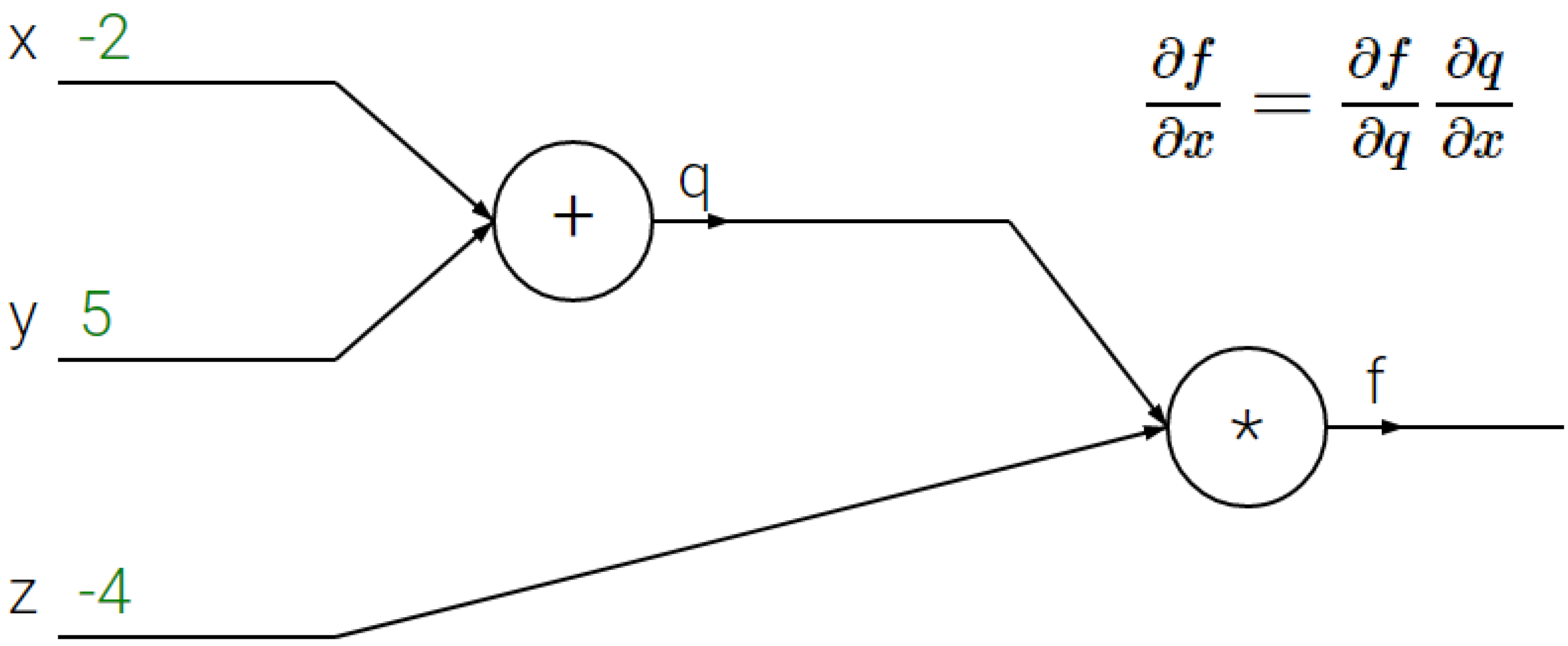
$$f(x, y, z) = (x + y)z$$

$q = x + y$ 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

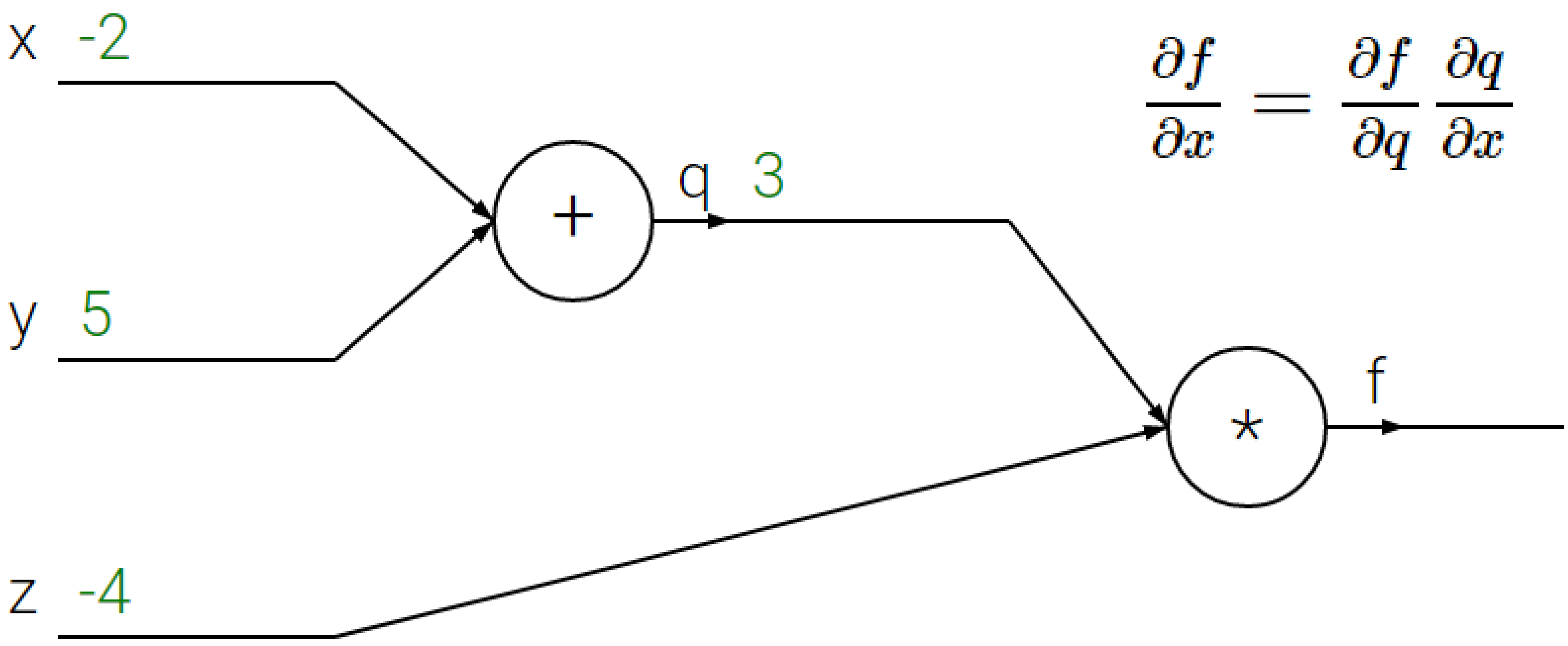
$$f(x, y, z) = (x + y)z$$

$q = x + y$ 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

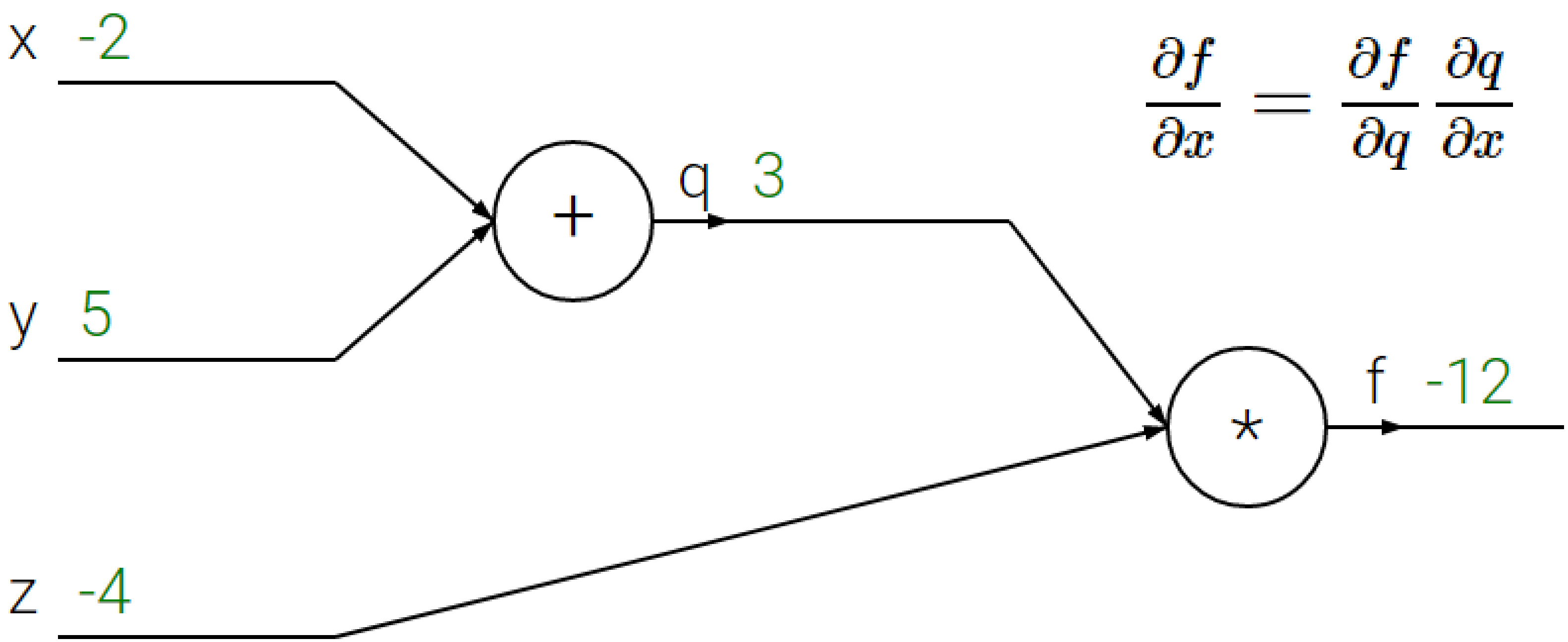
$$f(x, y, z) = (x + y)z$$

$q = x + y$ 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

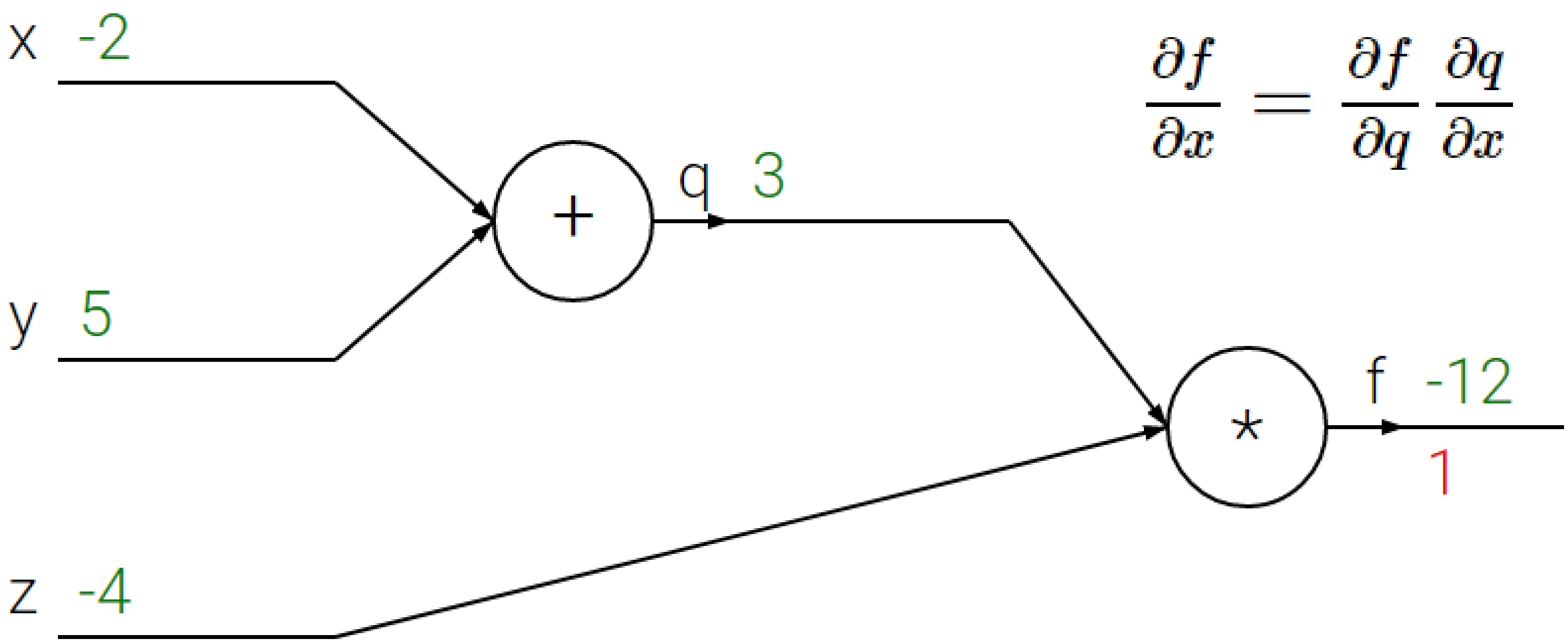
$$f(x, y, z) = (x + y)z$$

$q = x + y$ 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

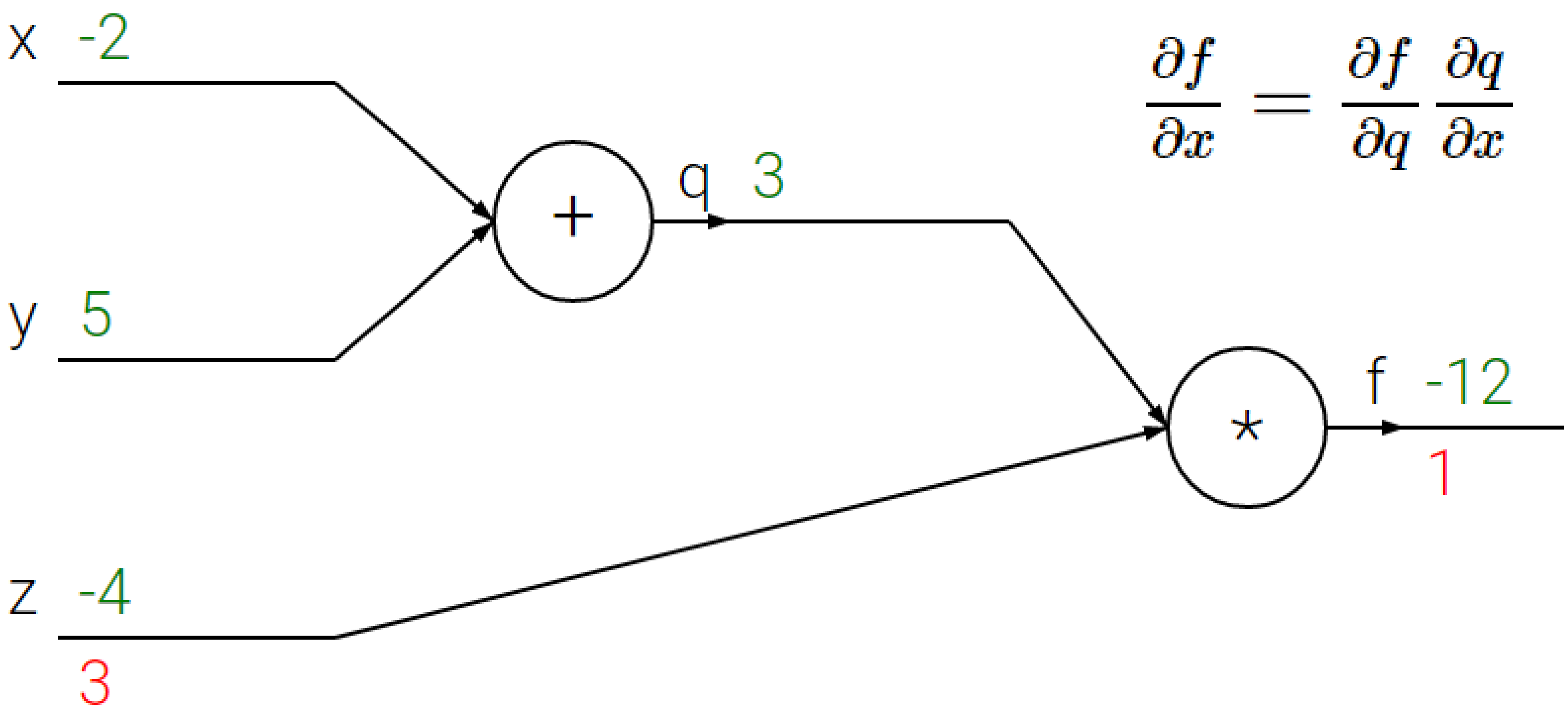
$$f(x, y, z) = (x + y)z$$

$q = x + y$ 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

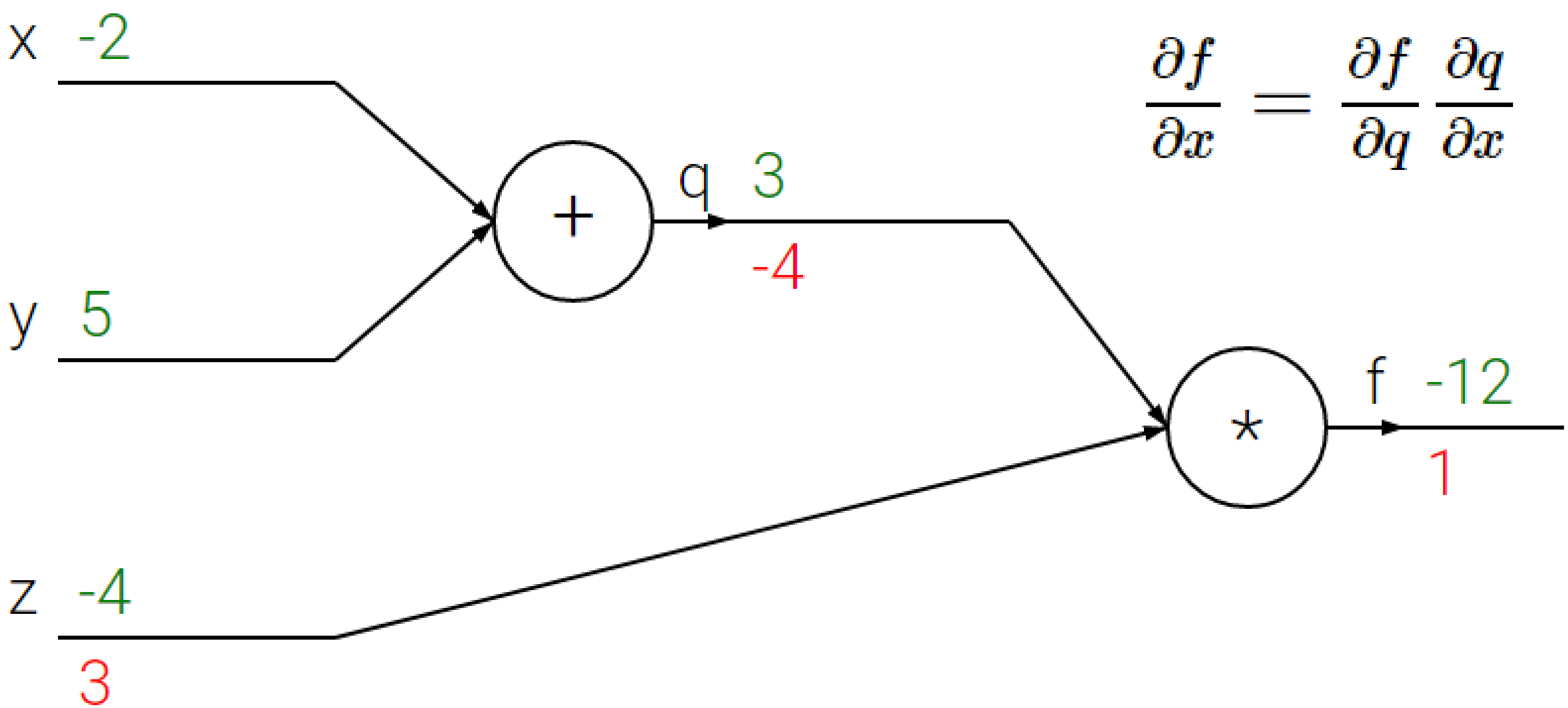
$$f(x, y, z) = (x + y)z$$

$q = x + y$ 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

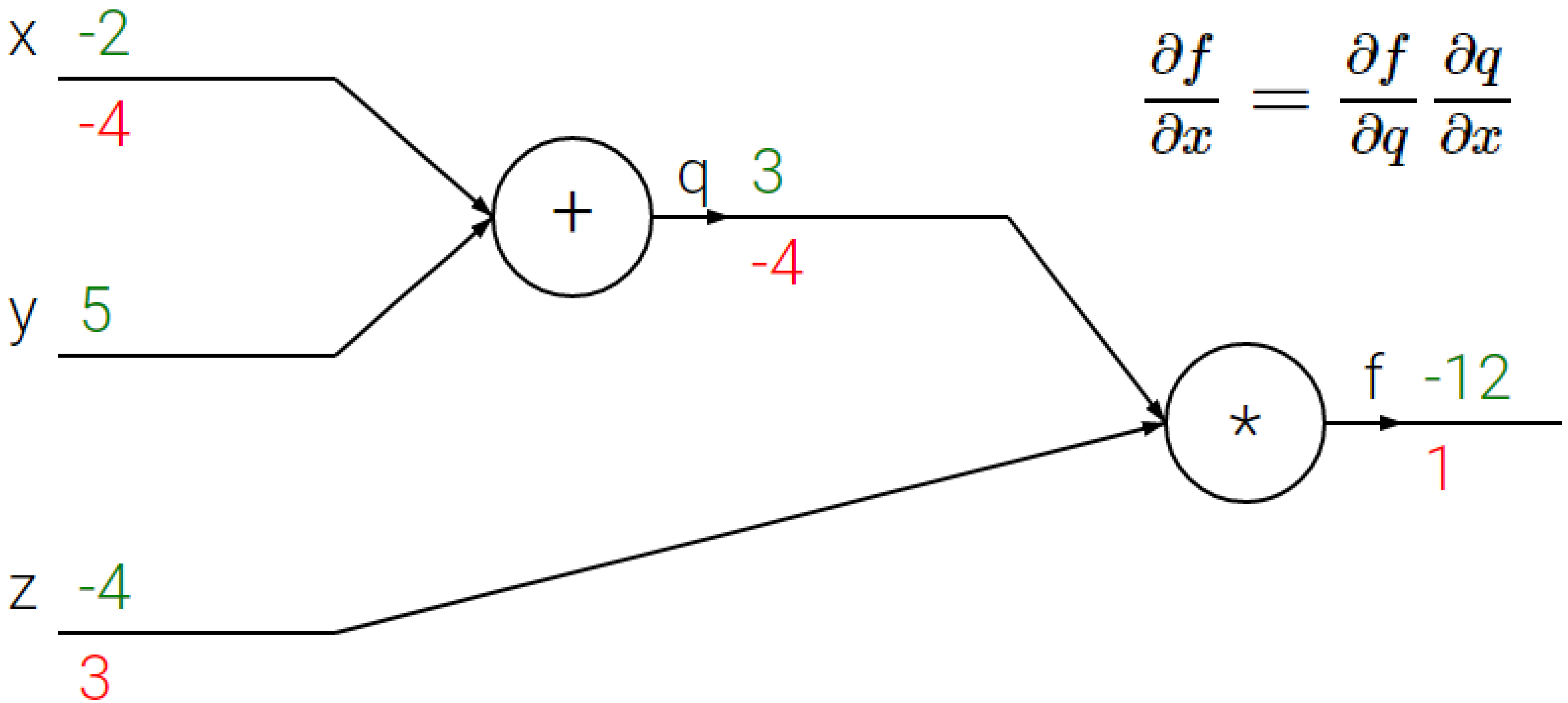
$$f(x, y, z) = (x + y)z$$

$q = x + y$ 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

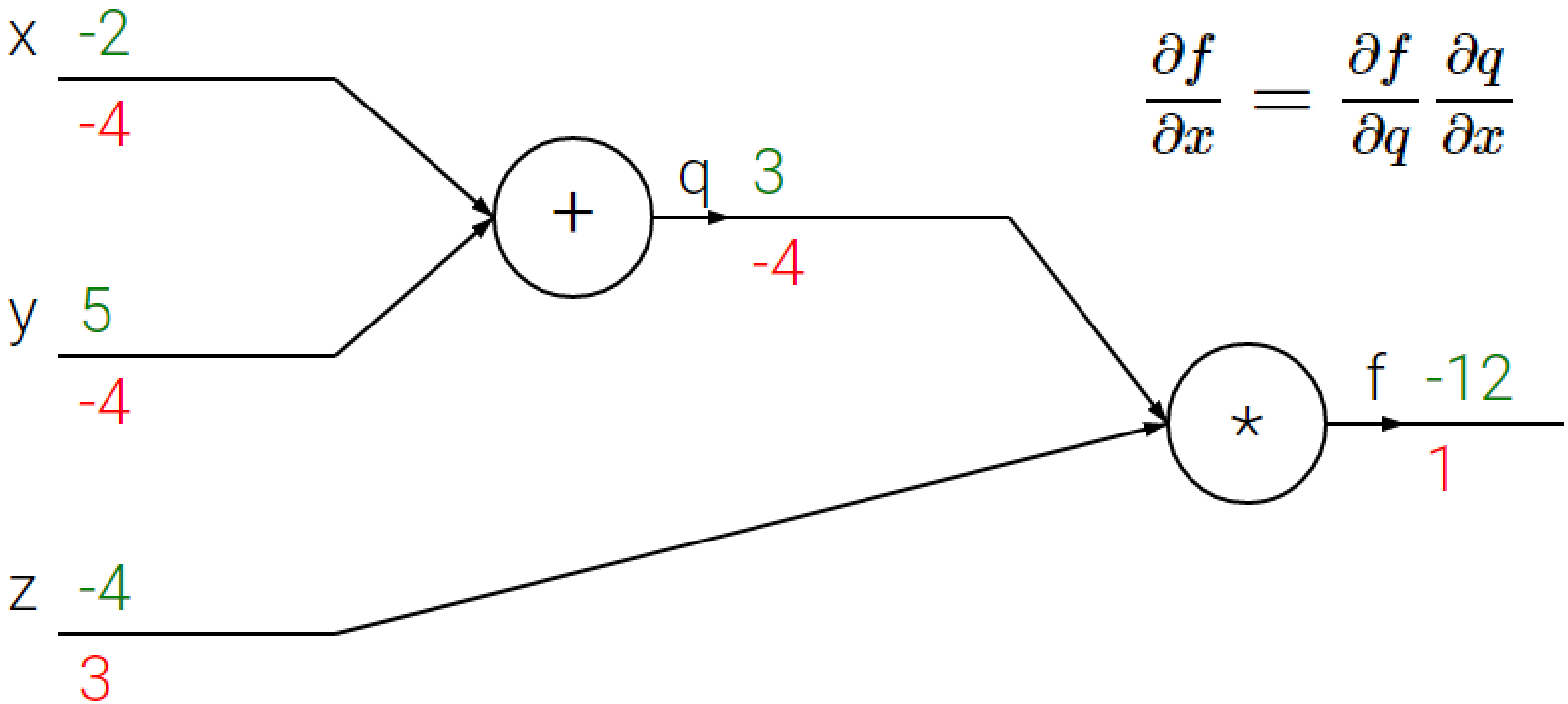
$$f(x, y, z) = (x + y)z$$

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

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

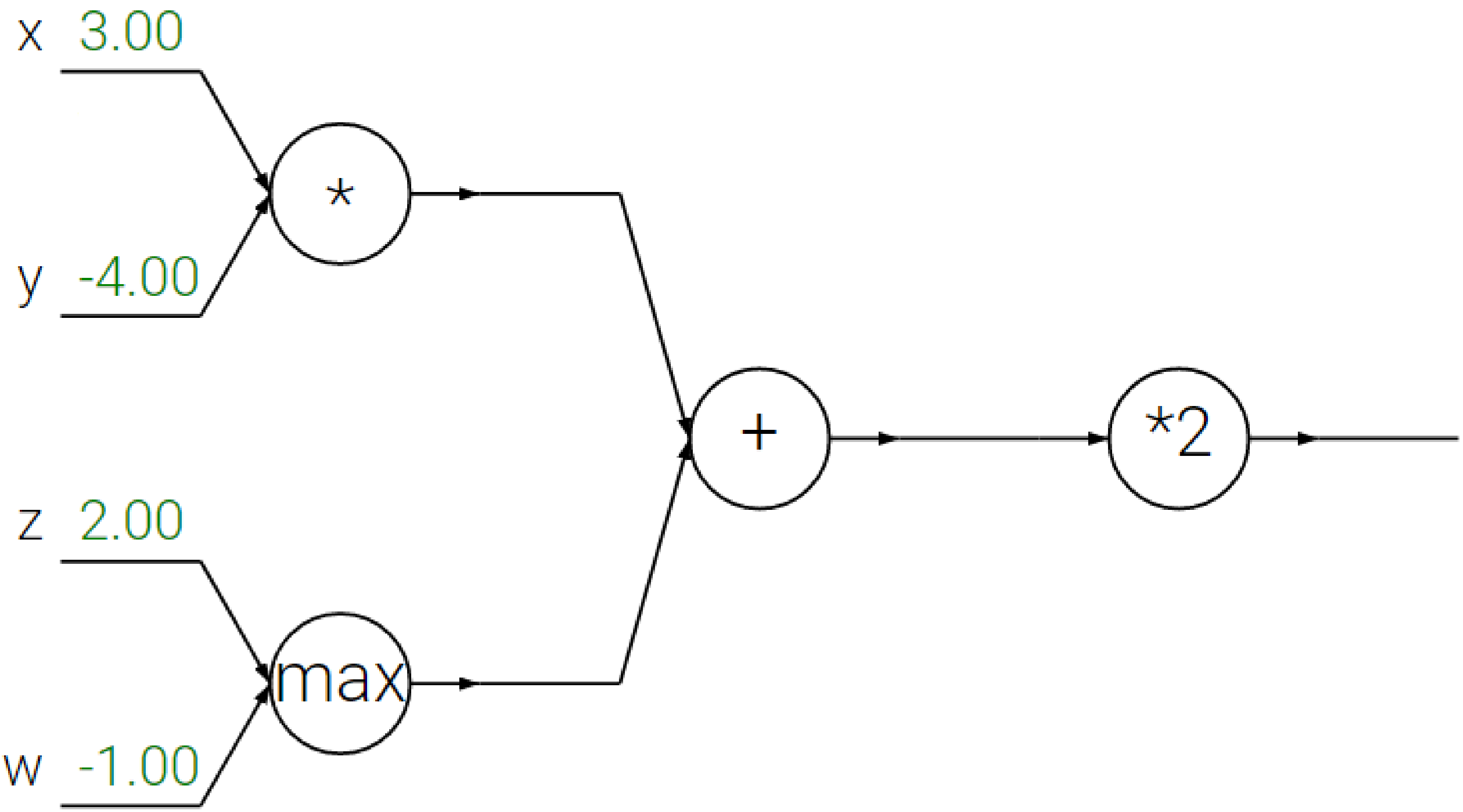
$$\frac{\partial q}{\partial x} = 1, \frac{\partial q}{\partial y} = 1$$

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



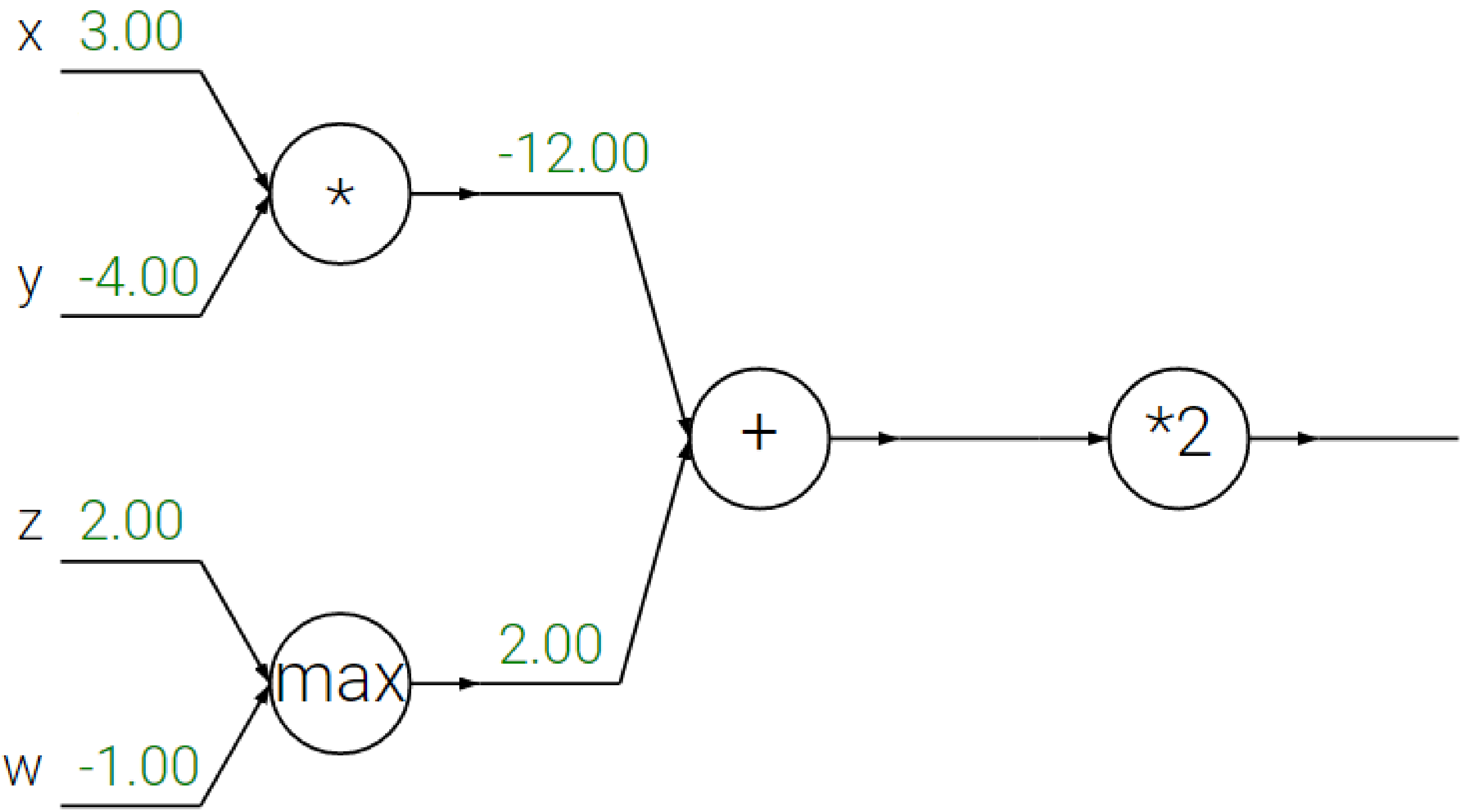
Source: http://cs231n.github.io

Forward and Backward Pass



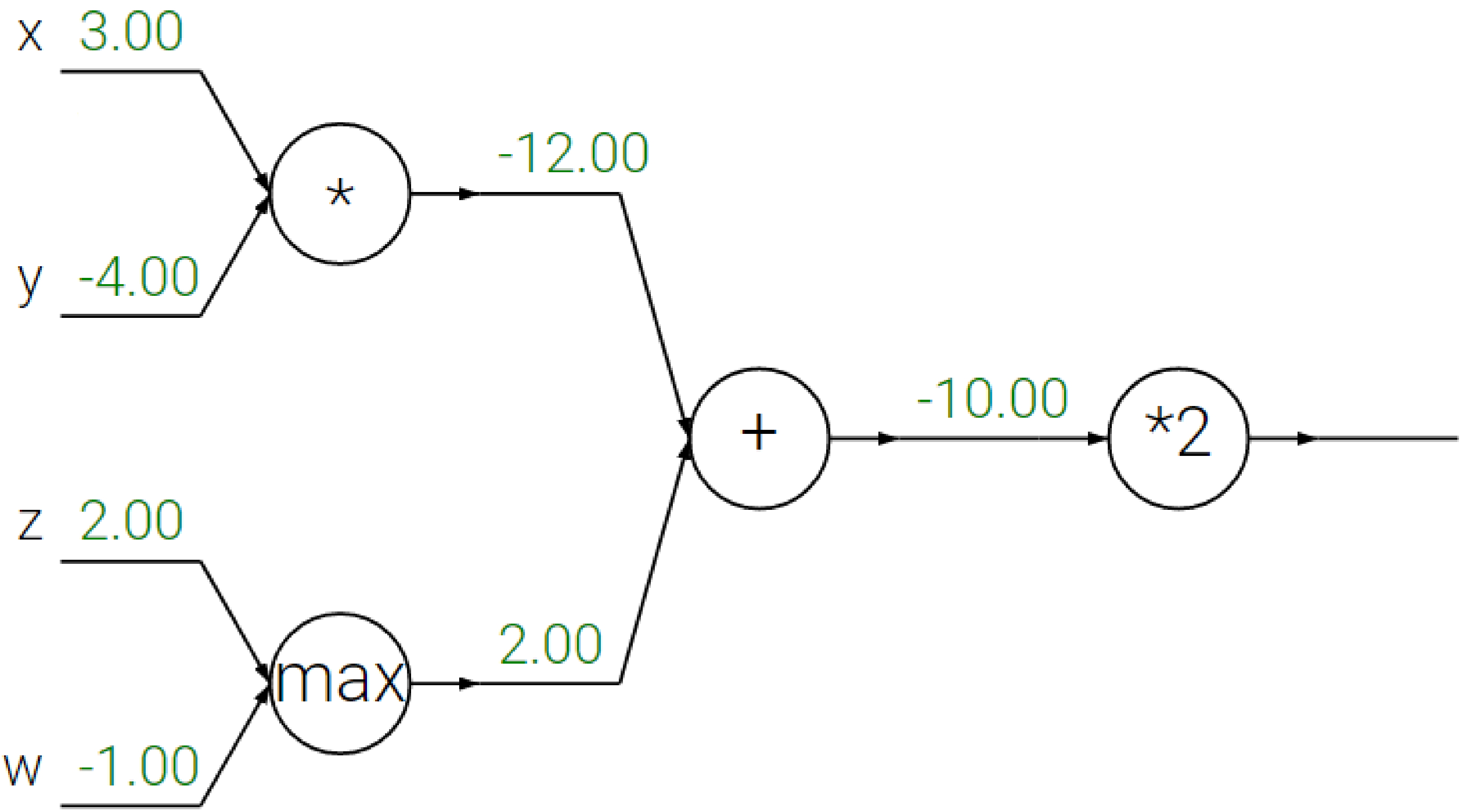
Source: http://cs231n.github.io

Forward and Backward Pass



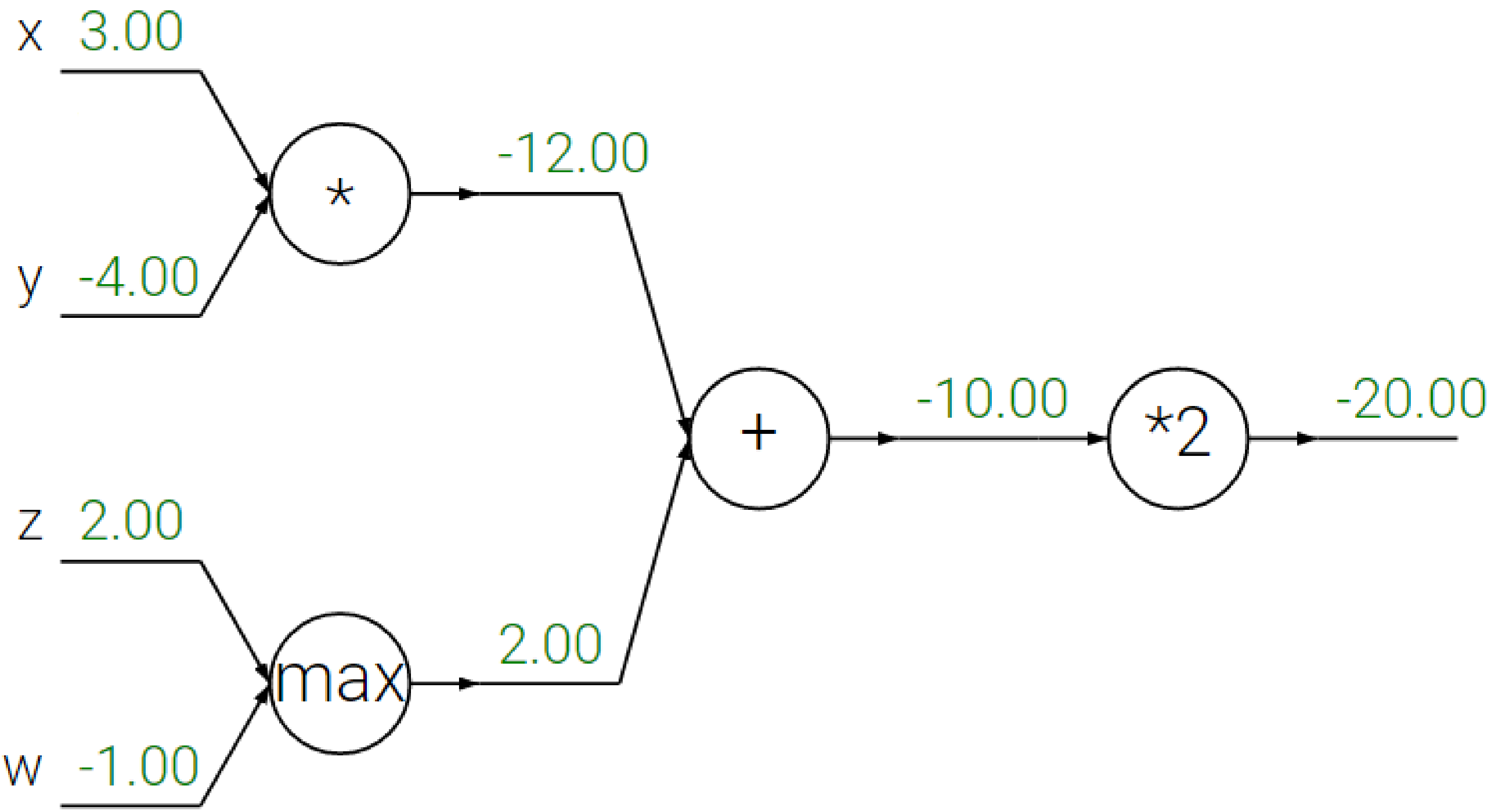
Source: http://cs231n.github.io

Forward and Backward Pass



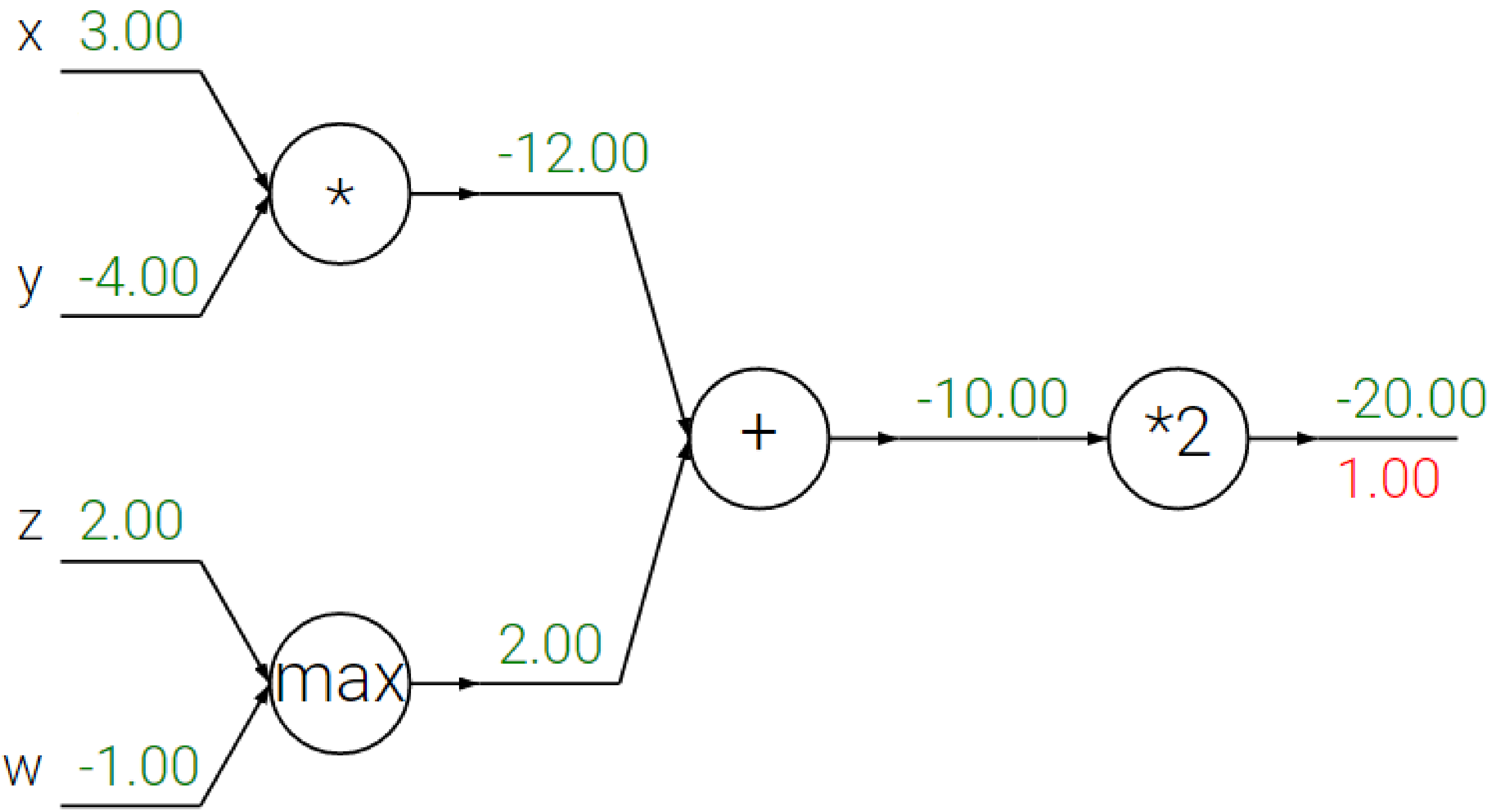
Source: http://cs231n.github.io

Forward and Backward Pass



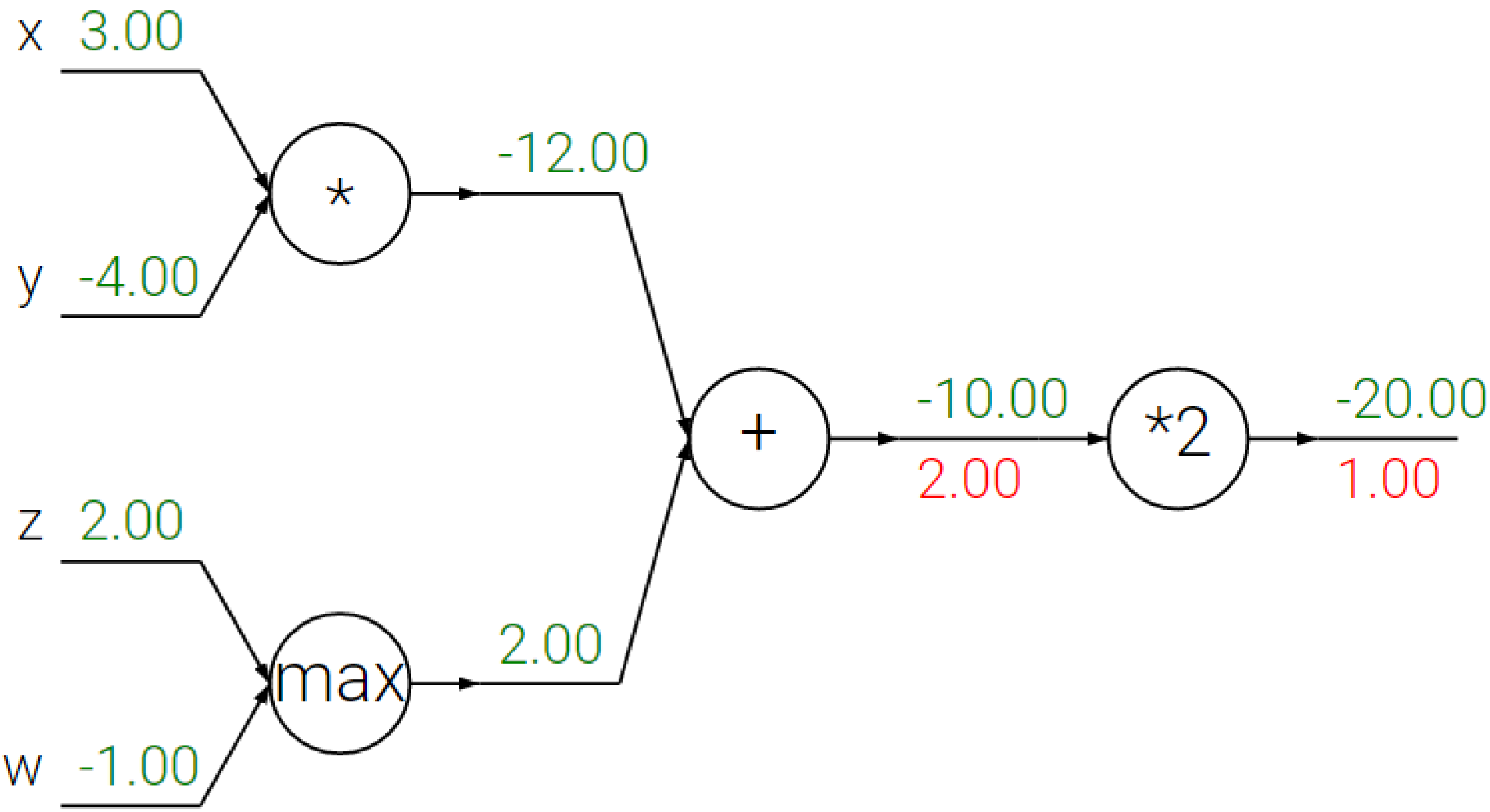
Source: http://cs231n.github.io

Forward and Backward Pass



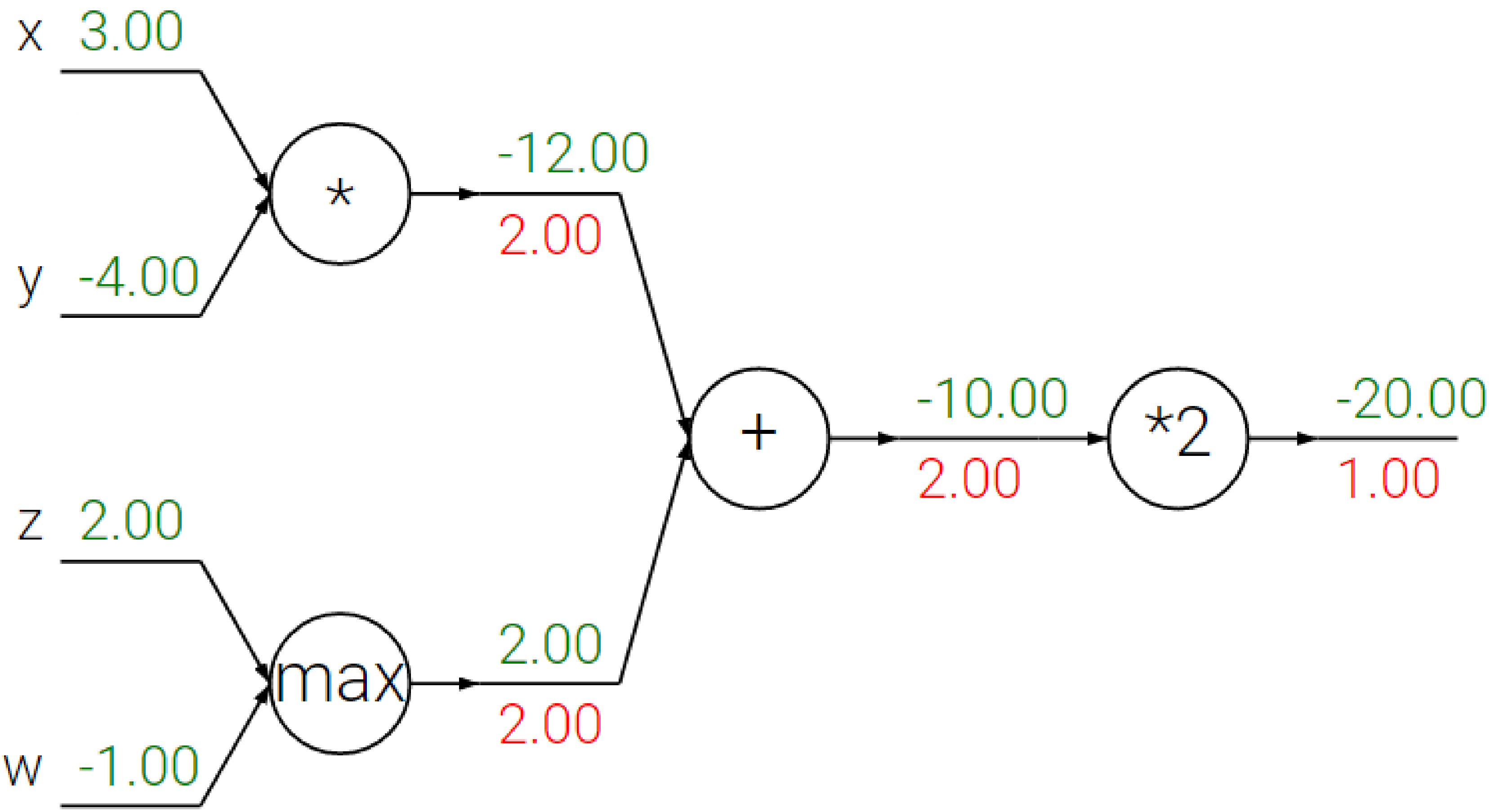
Source: http://cs231n.github.io

Forward and Backward Pass



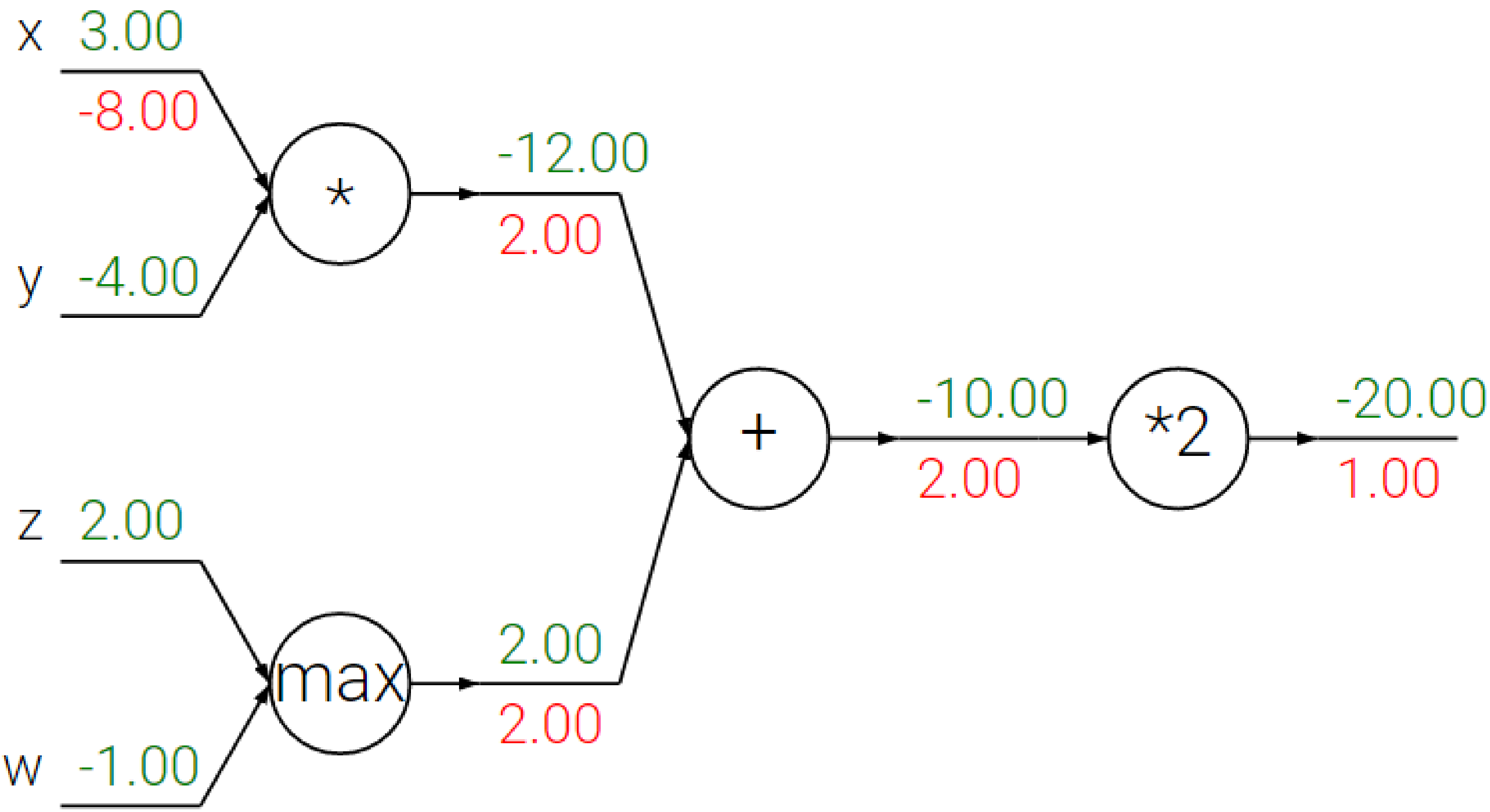
Source: http://cs231n.github.io

Forward and Backward Pass



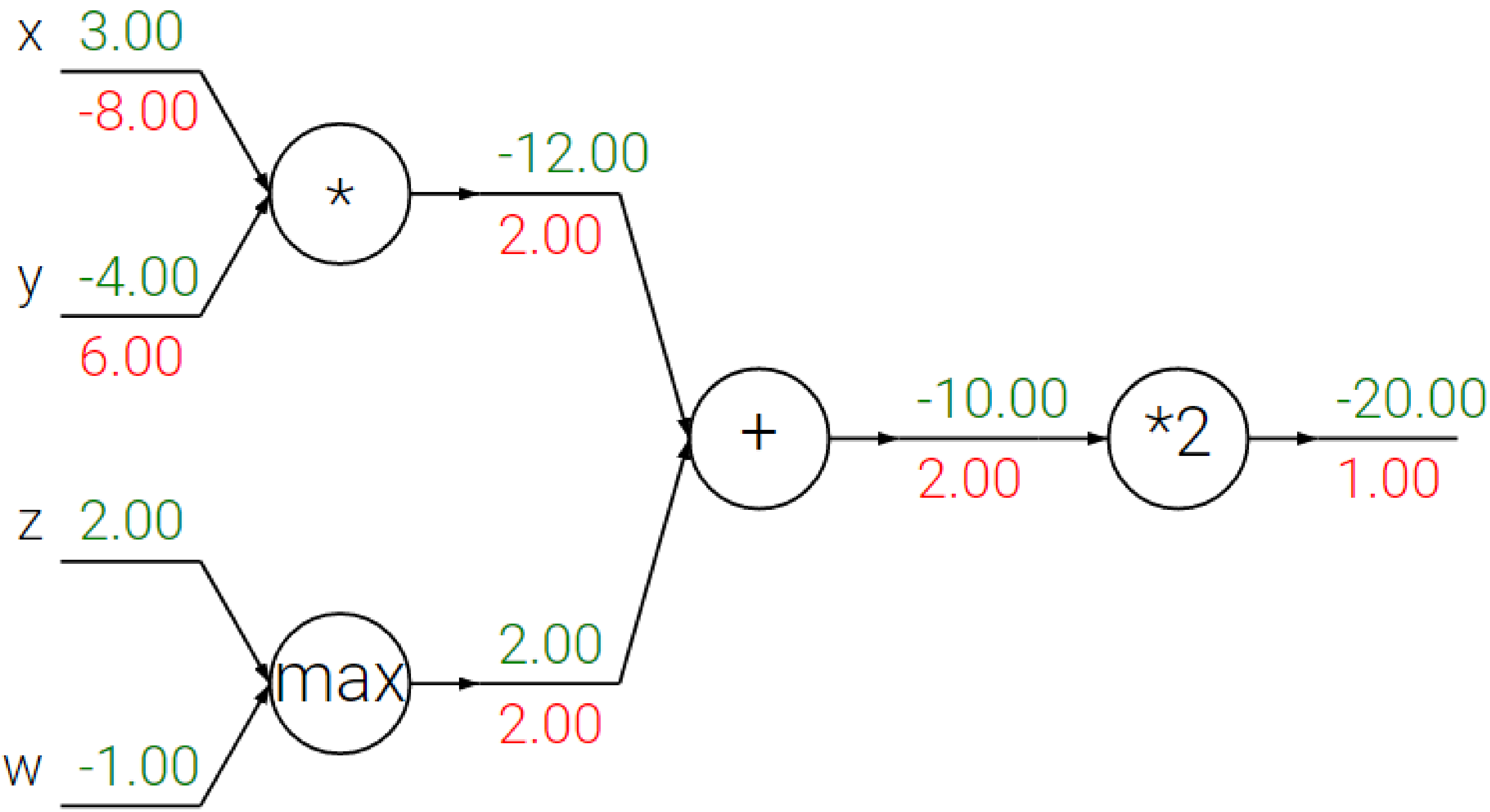
Source: http://cs231n.github.io

Forward and Backward Pass



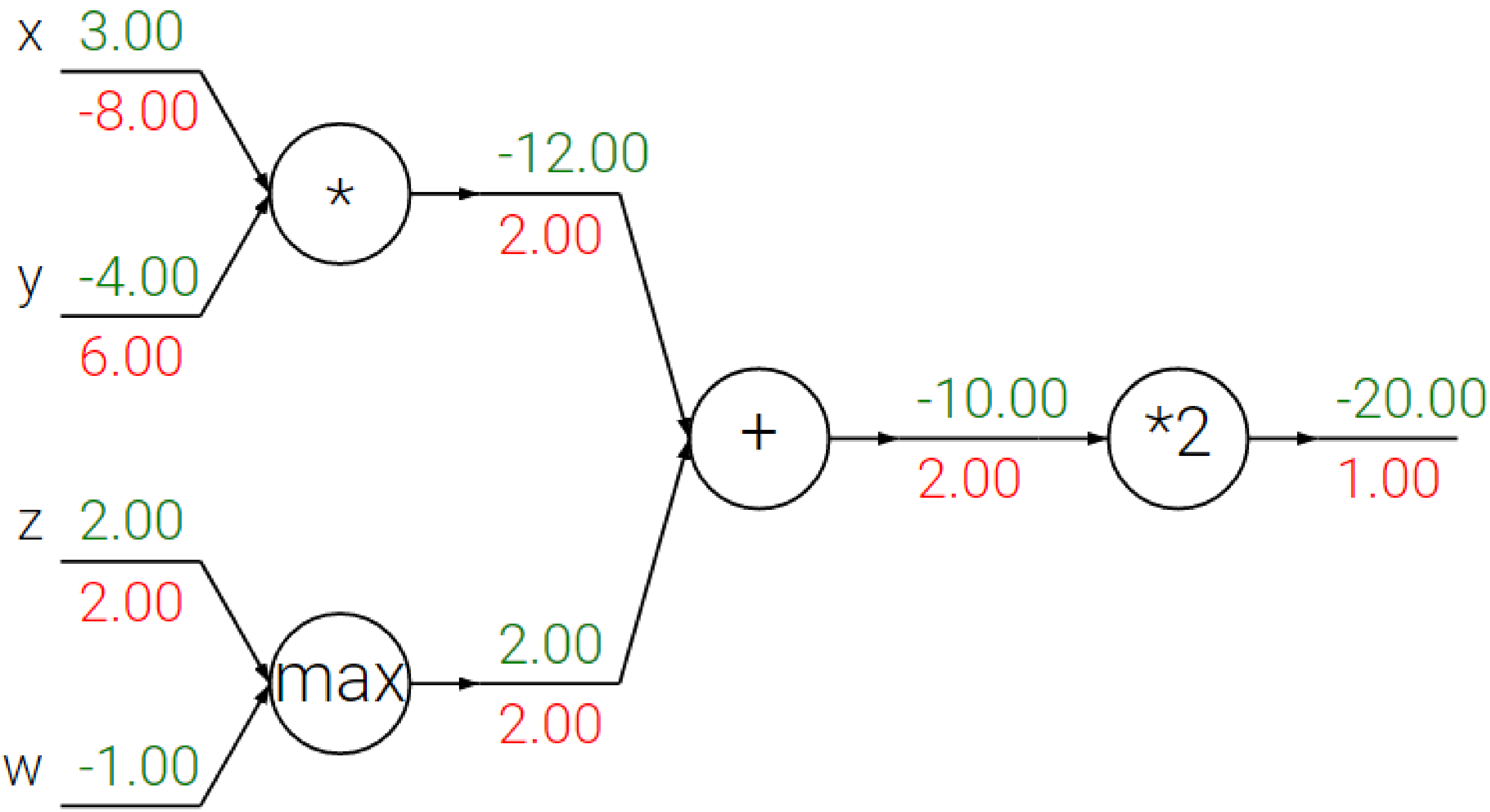
Source: http://cs231n.github.io

Forward and Backward Pass



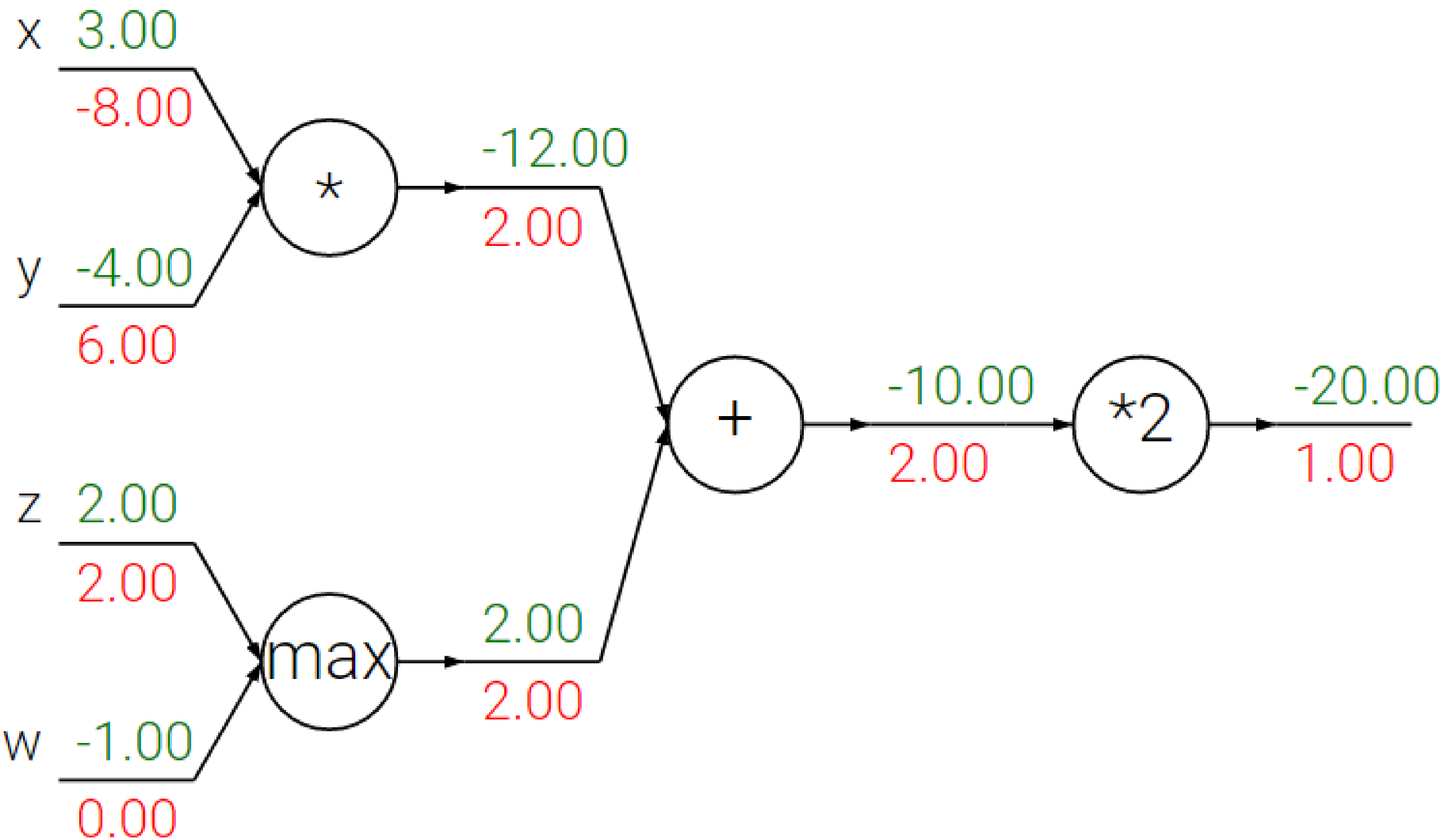
Source: http://cs231n.github.io

Forward and Backward Pass



Source: http://cs231n.github.io

Forward and Backward Pass



Source: http://cs231n.github.io

Sigmoid example

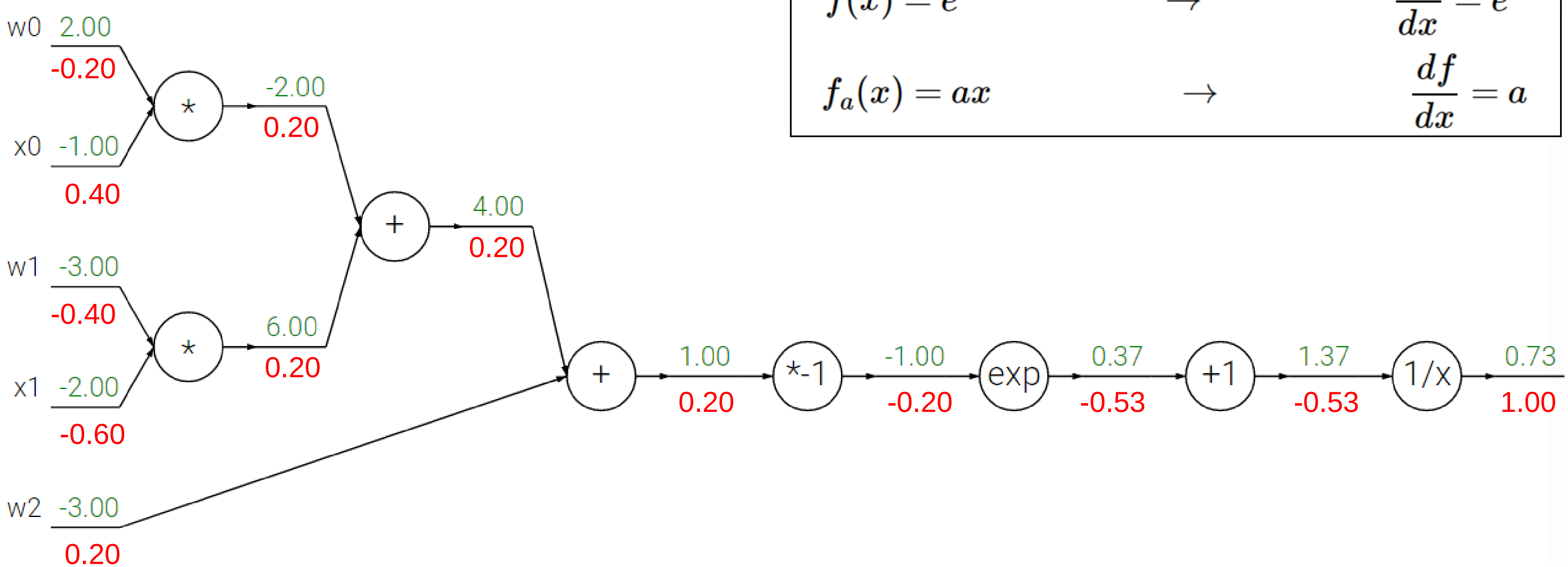
$$f(w,x) = \frac{1}{1+e^{-(w_0x_0+w_1x_1+w_2)}}$$

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

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

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

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



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):

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

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

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} :

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

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} :

$$\nabla_{w_{y_i}} L_i = - \underbrace{\left(\sum_{j \neq y_i} \mathbb{1}(w_j^T x_i - w_{y_i}^T x_i + \Delta > 0) \right)}_{\text{Count of the number of classes that didn't meet the desired margin}} x_i$$

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} :

$$\nabla_{w_{y_i}} L_i = - \underbrace{\left(\sum_{j \neq y_i} 1(w_j^T x_i - w_{y_i}^T x_i + \Delta > 0) \right)}_{\text{Count of the number of classes that didn't meet the desired margin}} x_i$$

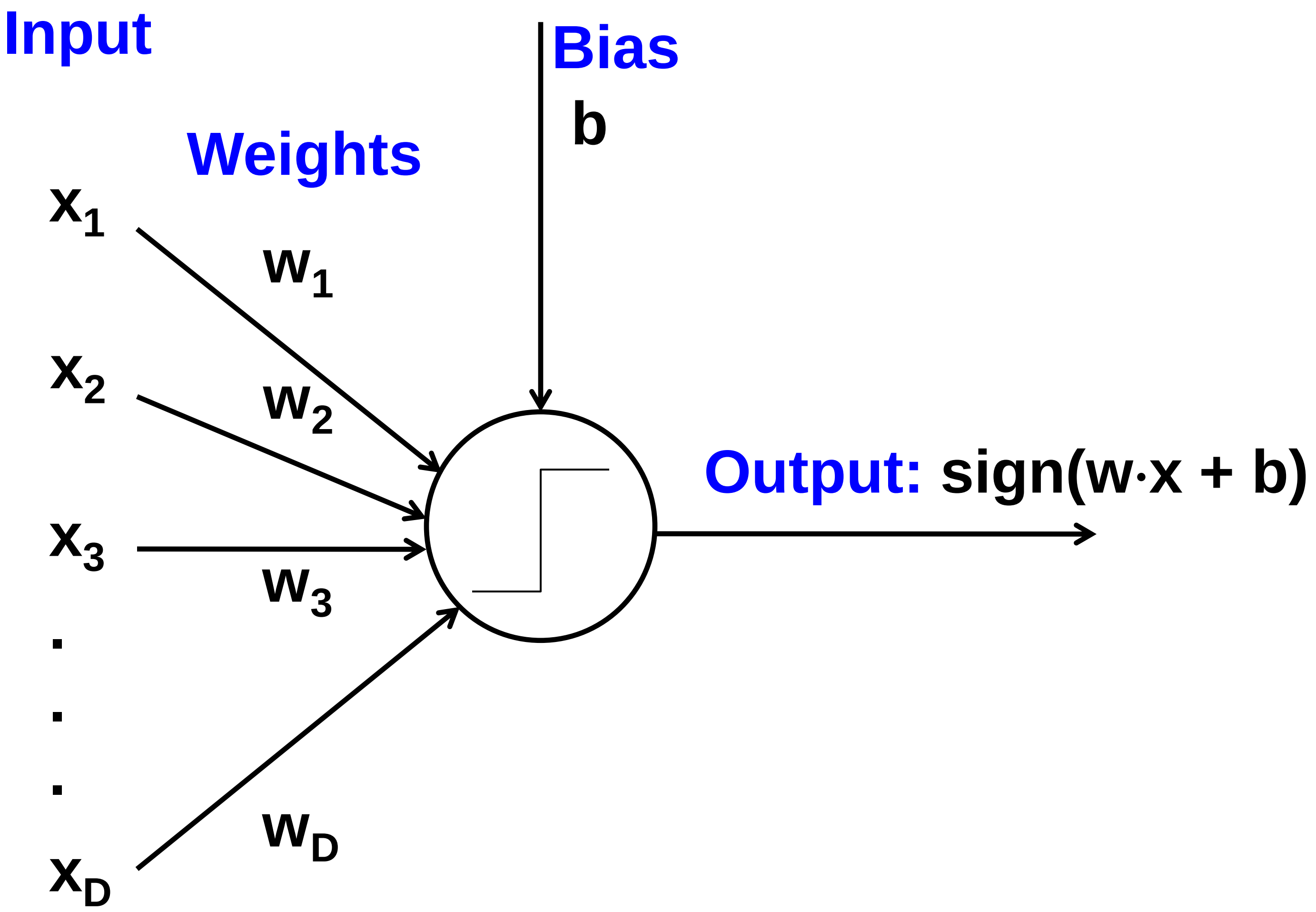
Gradient for the other rows where $j \neq y_i$:

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

Source: http://cs231n.github.io

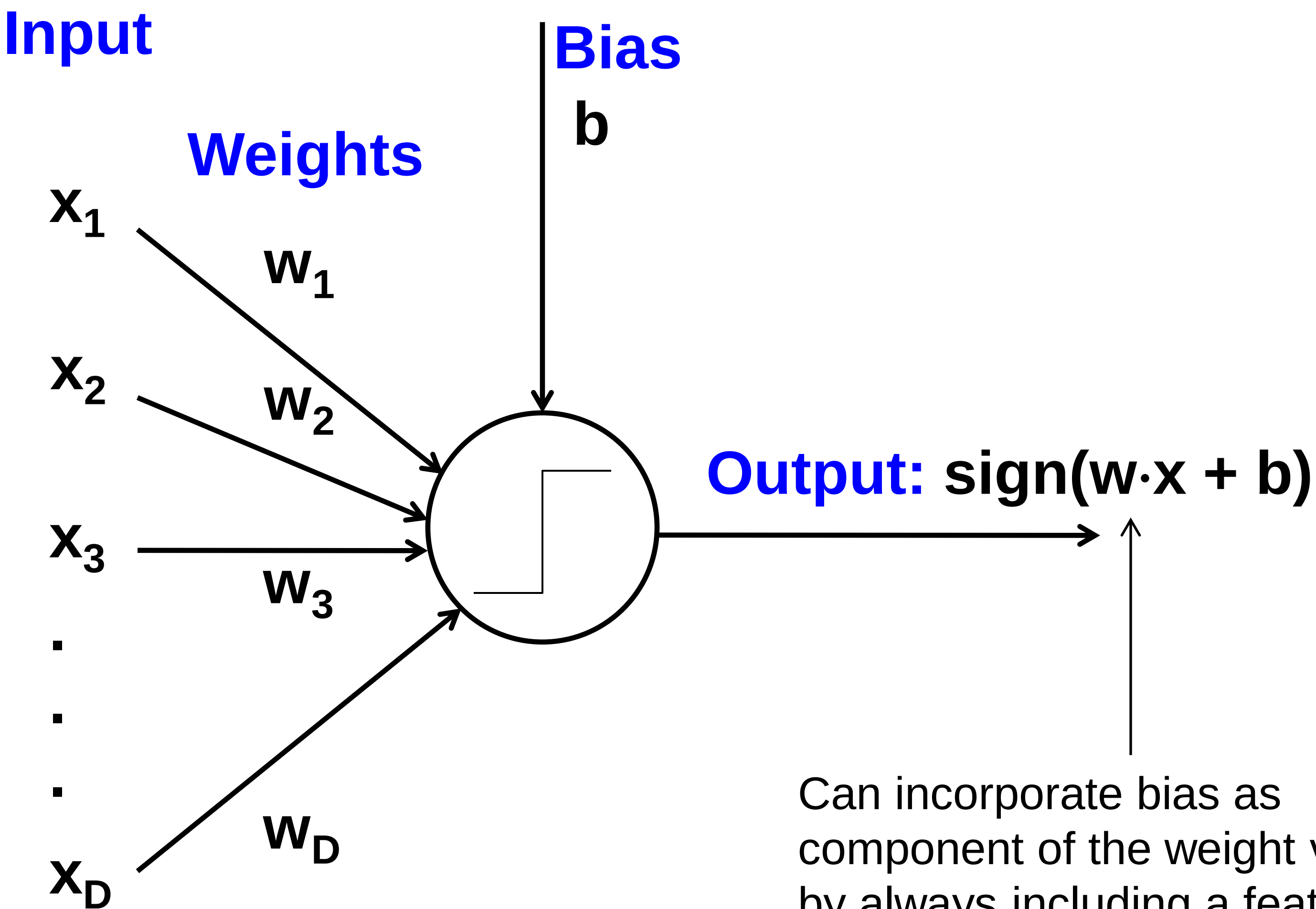
Perceptron

- Supervised learning of binary classifier



Perceptron

- Supervised learning of binary classifier



Can incorporate bias as component of the weight vector by always including a feature with value set to 1

Perceptron update rule

- Initialize weights randomly
- Cycle through training examples in multiple passes (*epochs*)

Perceptron update rule

- Initialize weights randomly
- Cycle through training examples in multiple passes (*epochs*)
- For each training instance **x** with label y:
  - Classify with current weights: $y' = \text{sign}(\mathbf{w}.\mathbf{x})$

Perceptron update rule

- Initialize weights randomly
- Cycle through training examples in multiple passes (*epochs*)
- For each training instance **x** with label y:
  - Classify with current weights: $y' = \text{sign}(\mathbf{w} \cdot \mathbf{x})$
  - Update weights: $\mathbf{w} \leftarrow \mathbf{w} + \alpha(y - y')\mathbf{x}$

Perceptron update rule

- Initialize weights randomly
- Cycle through training examples in multiple passes (*epochs*)
- For each training instance **x** with label y:
  - Classify with current weights: $y' = \text{sign}(\mathbf{w}.\mathbf{x})$
  - Update weights: $\mathbf{w} \leftarrow \mathbf{w} + \alpha(y-y')\mathbf{x}$
  - **What happens if y' is correct?**

Perceptron update rule

- Initialize weights randomly
- Cycle through training examples in multiple passes (*epochs*)
- For each training instance **x** with label y:
  - Classify with current weights: $y' = \text{sign}(\mathbf{w}.\mathbf{x})$
  - Update weights: $\mathbf{w} \leftarrow \mathbf{w} + \alpha(y-y')\mathbf{x}$
  - **What happens if y' is correct?**
  - **Otherwise, if y' is wrong -**

Perceptron update rule

- Initialize weights randomly
- Cycle through training examples in multiple passes (*epochs*)
- For each training instance **x** with label y:
  - Classify with current weights: $y' = \text{sign}(\mathbf{w}.\mathbf{x})$
  - Update weights: $\mathbf{w} \leftarrow \mathbf{w} + \alpha(y-y')\mathbf{x}$
  - **What happens if y' is correct?**
  - **Otherwise, if y' is wrong -**

  $w_i \leftarrow w_i + \alpha(y-y')x_i$

    – If y = 1 and y' = -1, w_i will be increased if x_i is positive or decreased if x_i is negative → $\mathbf{w} \cdot \mathbf{x}$ will get bigger

Perceptron update rule

- Initialize weights randomly
- Cycle through training examples in multiple passes (*epochs*)
- For each training instance **x** with label y:
  - Classify with current weights: $y' = \text{sign}(\mathbf{w.x})$
  - Update weights: $\mathbf{w} \leftarrow \mathbf{w} + \alpha(y-y')\mathbf{x}$
  - **What happens if y' is correct?**
  - **Otherwise, if y' is wrong -**

    $w_i \leftarrow w_i + \alpha(y-y')x_i$
    - If $y = 1$ and $y' = -1$, w_i will be increased if x_i is positive or decreased if x_i is negative → $\mathbf{w}\cdot\mathbf{x}$ will get bigger
    - If $y = -1$ and $y' = 1$, w_i will be decreased if x_i is positive or increased if x_i is negative → $\mathbf{w}\cdot\mathbf{x}$ will get smaller

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*)

$$\sigma(\sum_i w_i x_i + b)$$

$$\downarrow$$

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*)

$$\sigma(\sum_i w_i x_i + b)$$

$$\downarrow$$

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

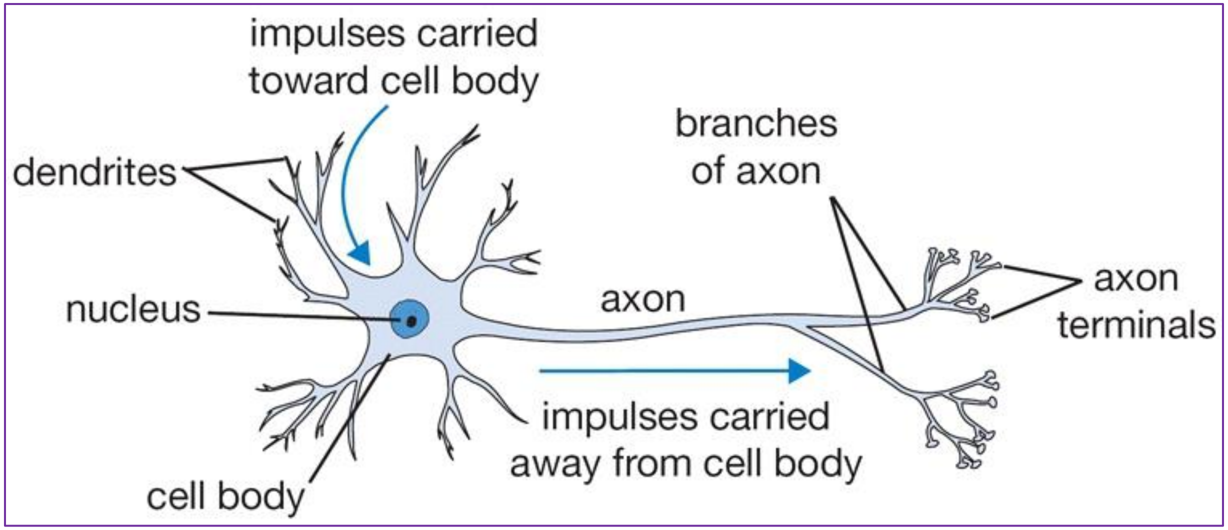
Probability of the other class would be:

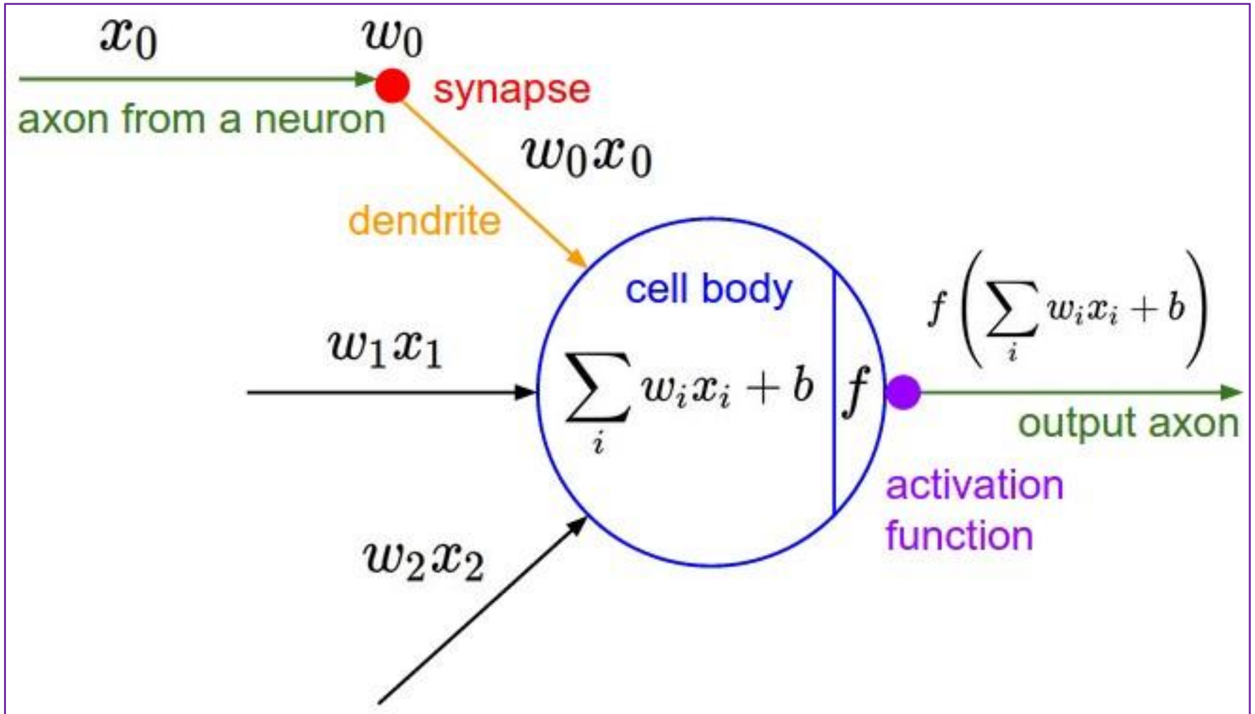
$$P(y_i = 0 \mid x_i; w) = 1 - P(y_i = 1 \mid x_i; w)$$

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

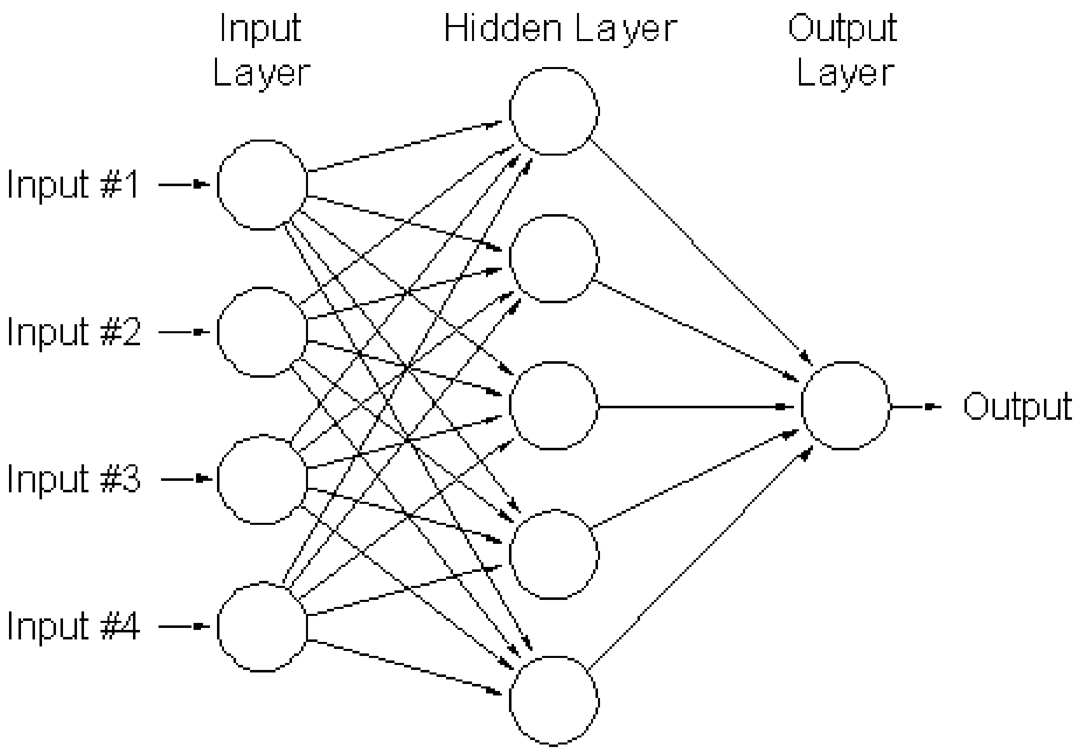




Source: http://cs231n.github.io

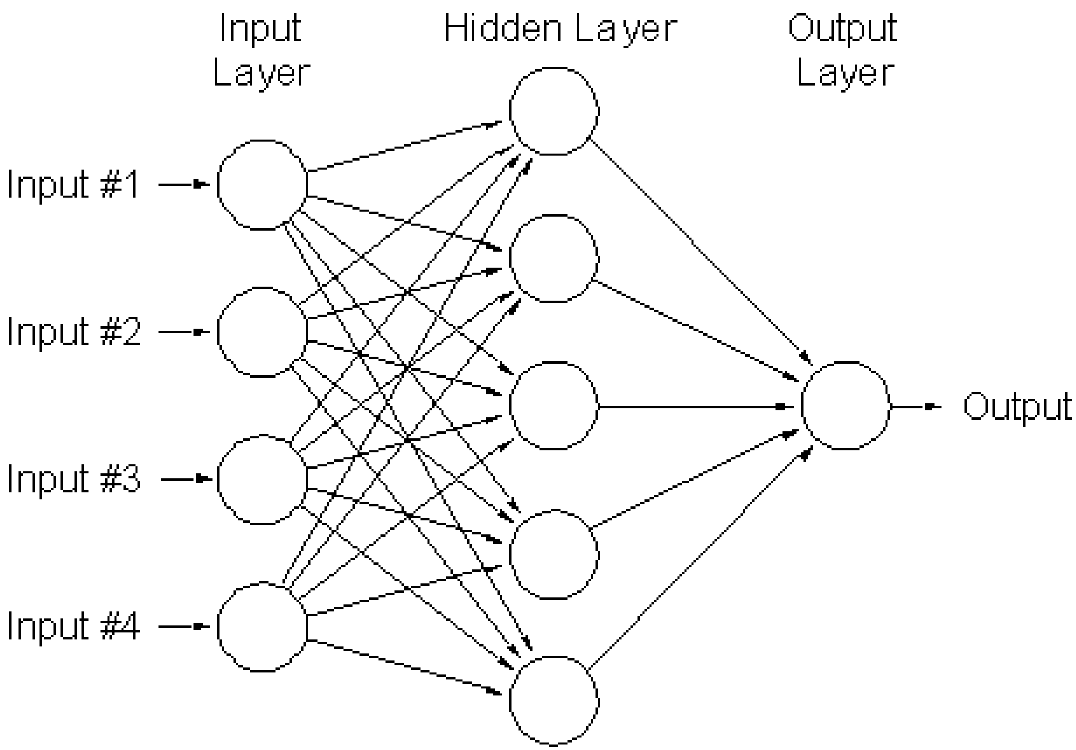
Multi-Layer Neural Networks

- Network with a hidden layer:



Multi-Layer Neural Networks

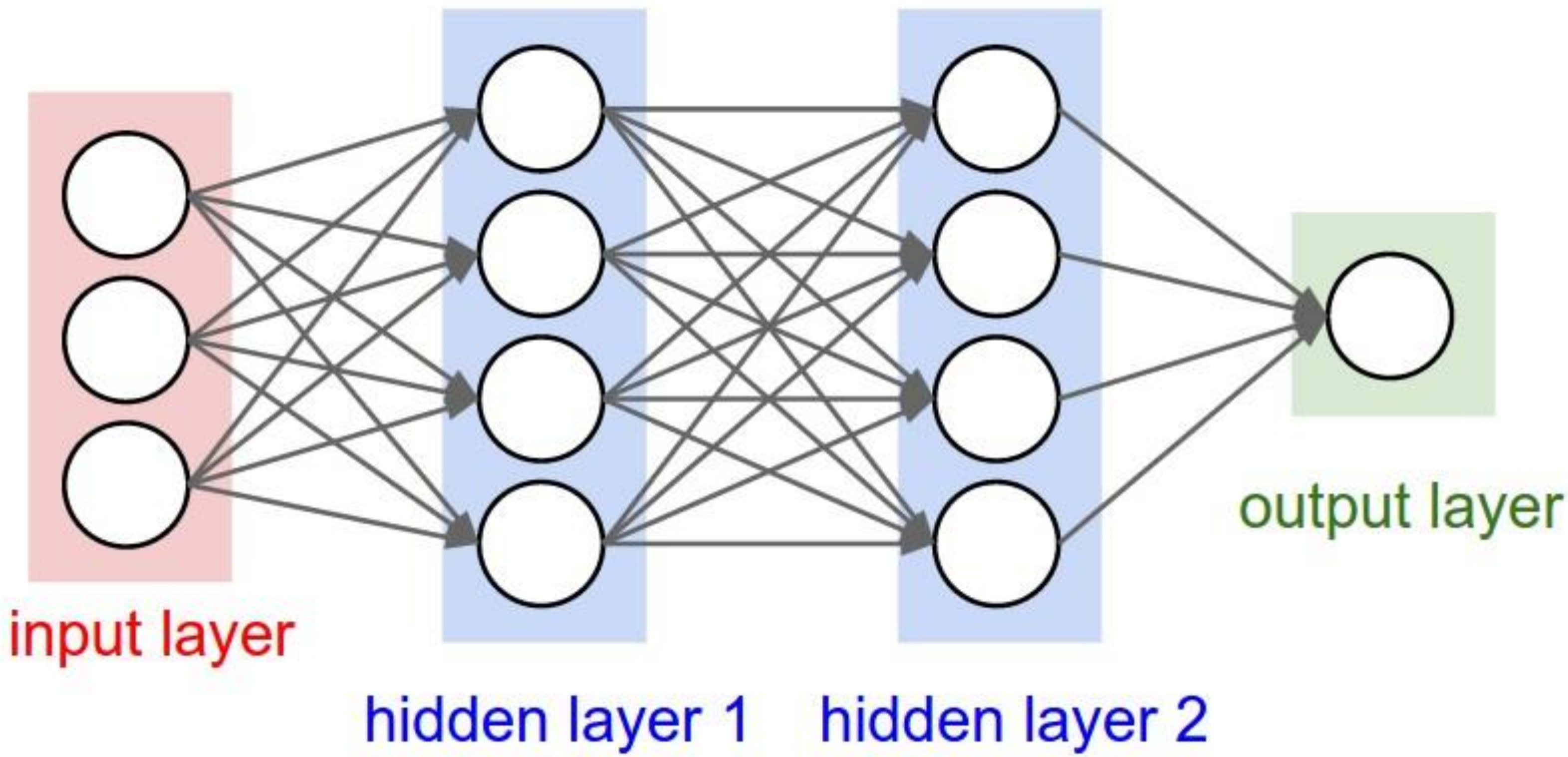
- Network with a hidden layer:



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

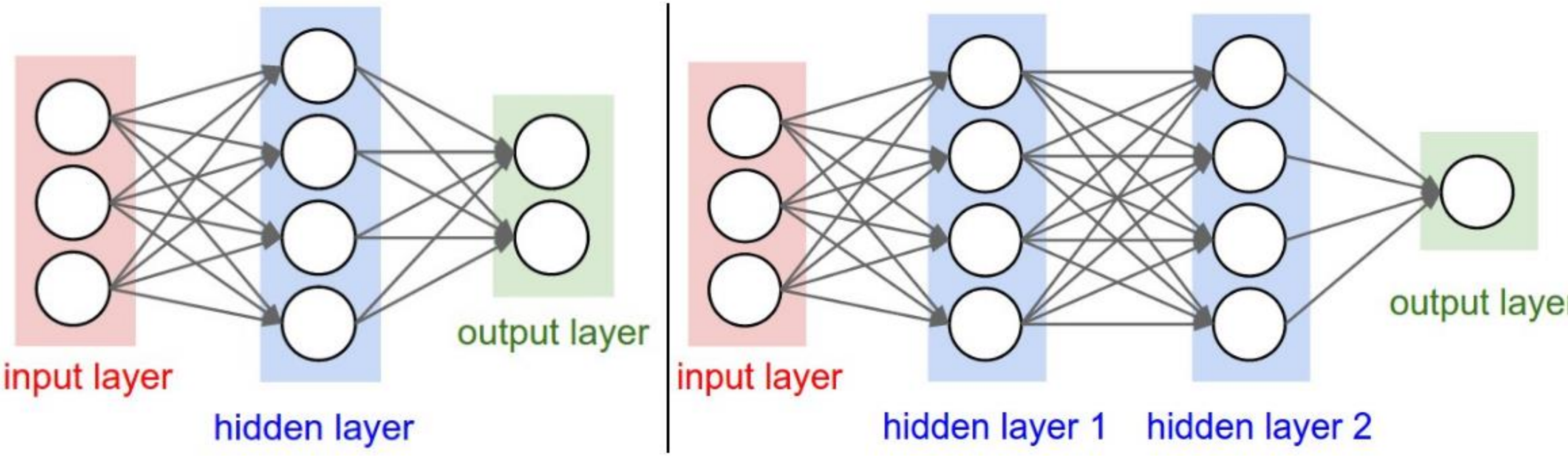
Multi-Layer Neural Networks

- Beyond a single hidden layer:



Source: http://cs231n.github.io

Sizing neural networks



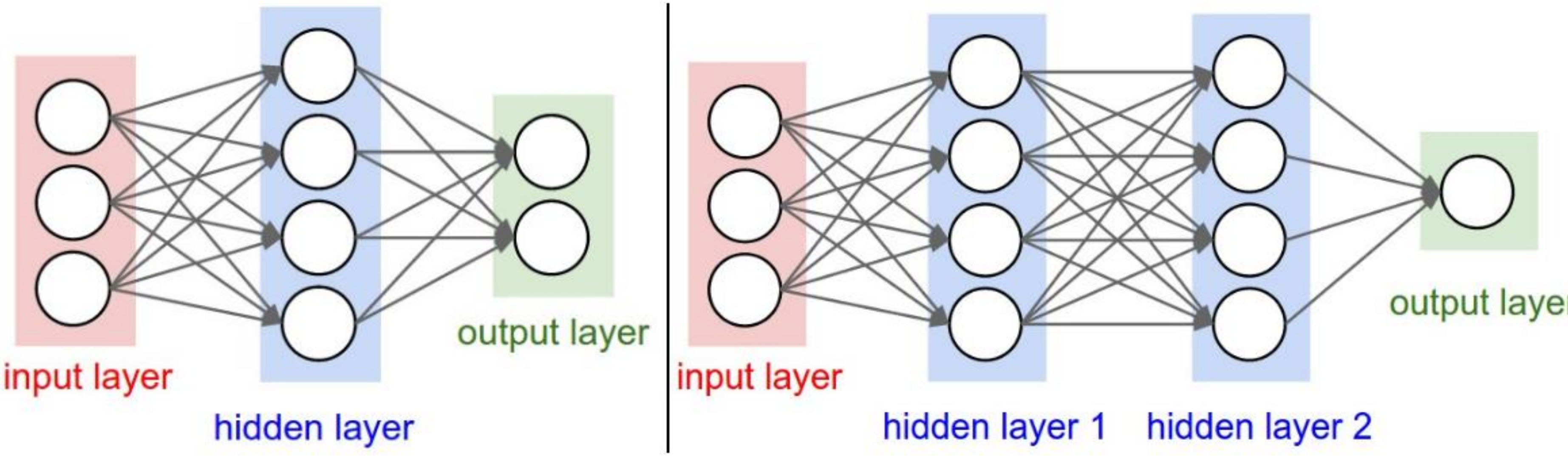
First network (left):

No. of neurons (not counting the inputs):

No. of learnable parameters:

Source: http://cs231n.github.io

Sizing neural networks



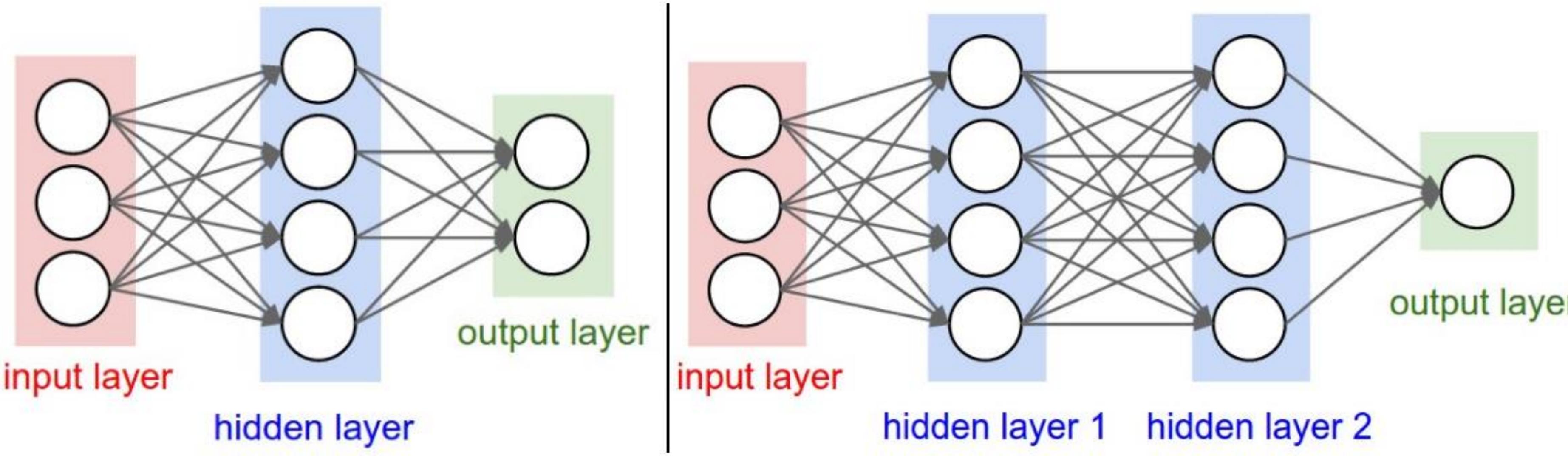
First network (left):

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

No. of learnable parameters:

Source: http://cs231n.github.io

Sizing neural networks



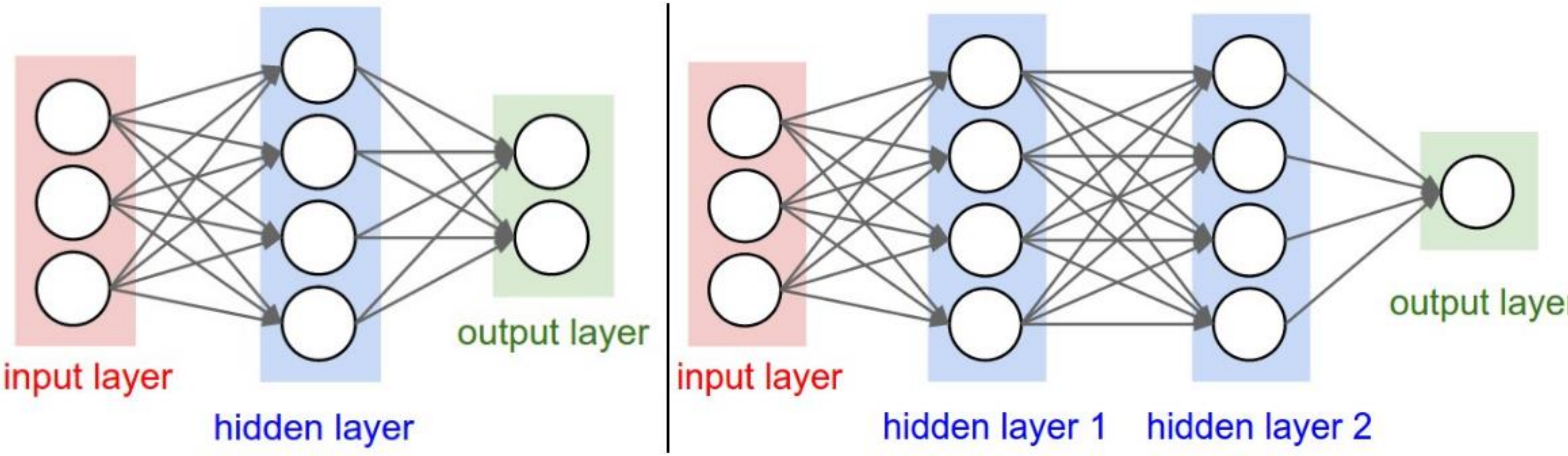
First network (left):

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.

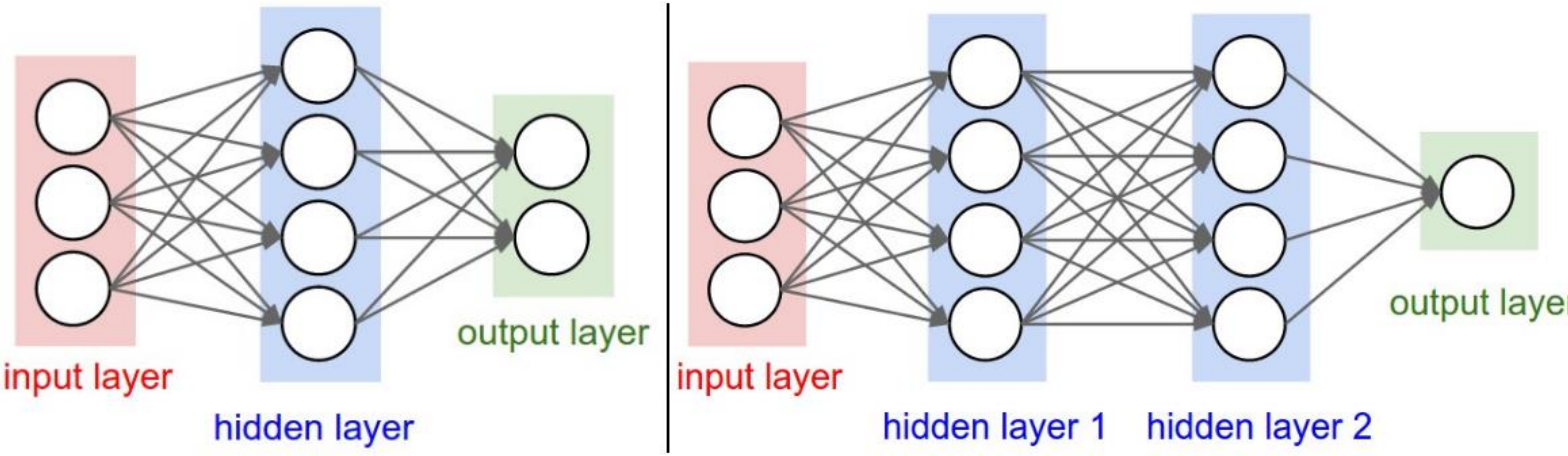
Second network (right):

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.

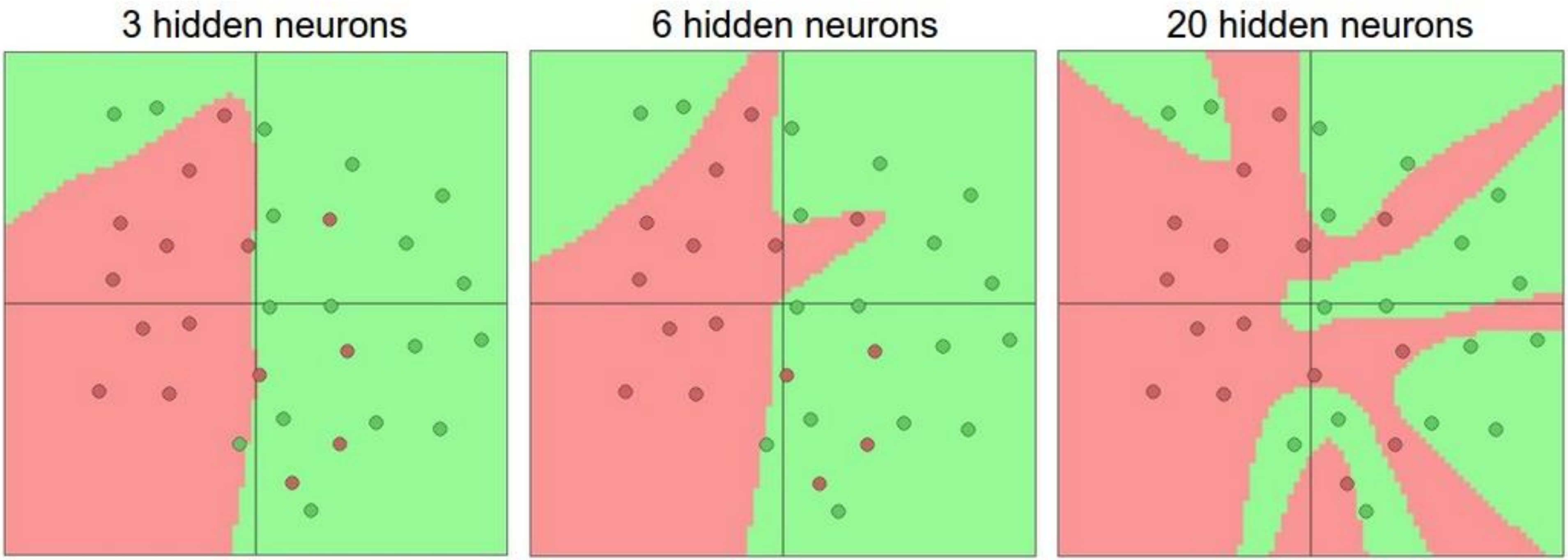
Second network (right):

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

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)

Acknowledgements

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

- Forsyth
- Steve Seitz
- Noah Snavely
- J.B. Huang
- Derek Hoiem
- D. Lowe
- A. Bobick
- S. Lazebnik
- K. Grauman
- R. Zaleski
- Antonio Torralba
- Rob Fergus
- Leibe
- And many more ………..

Next Lecture

Convolutional Neural Networks