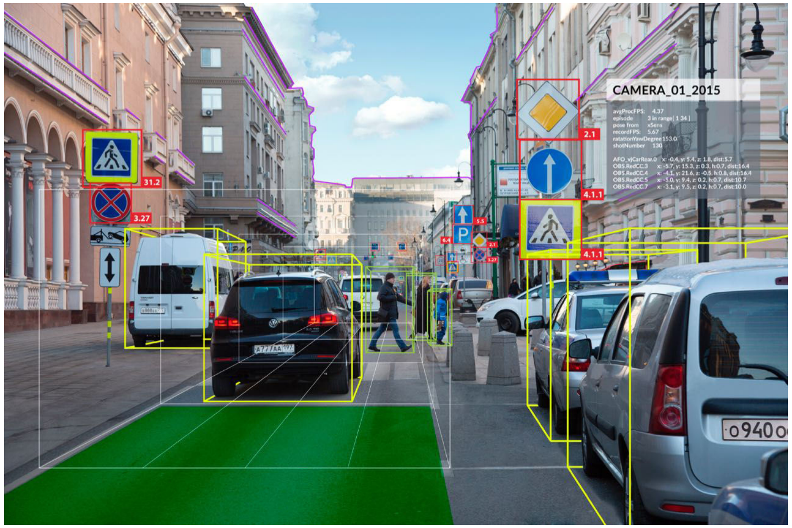
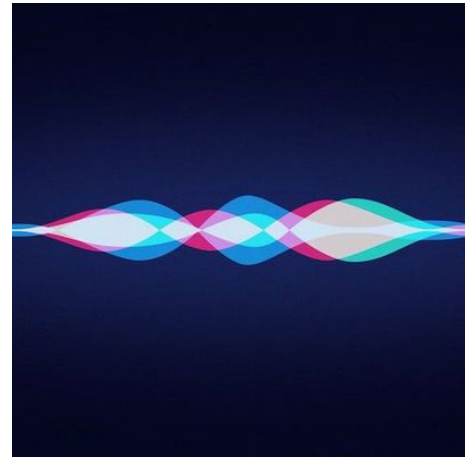
Introduction to Deep Learning

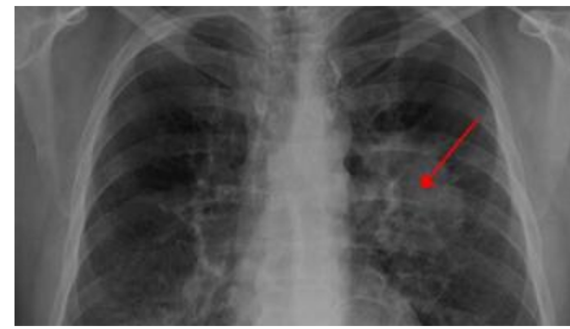
Raghu Ramanujan

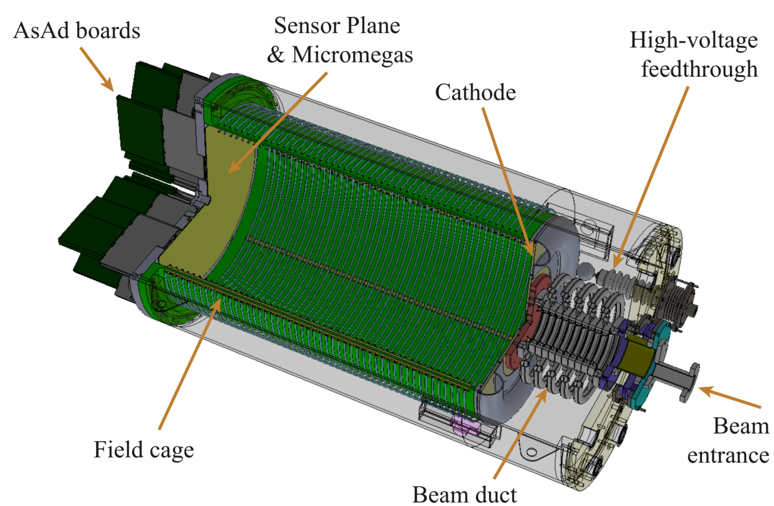
Dept. of Mathematics and Computer Science

Davidson College


CAMERA_01_2015






AsAd boards
Sensor Plane
& Micromegas
Cathode
High-voltage
feedthrough
Field cage
Beam duct
Beam
entrance

Neural Network Fundamentals

The Neuron

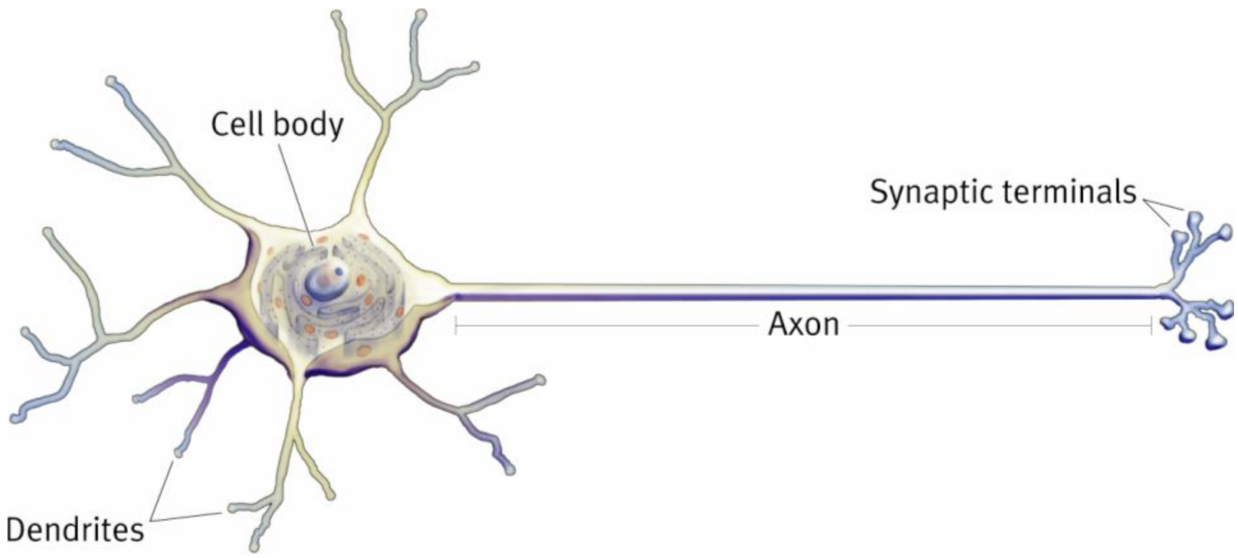


Image courtesy: `http://bio3520.nicerweb.com/Locked/chap/ch03/neuron.html`

The Neuron

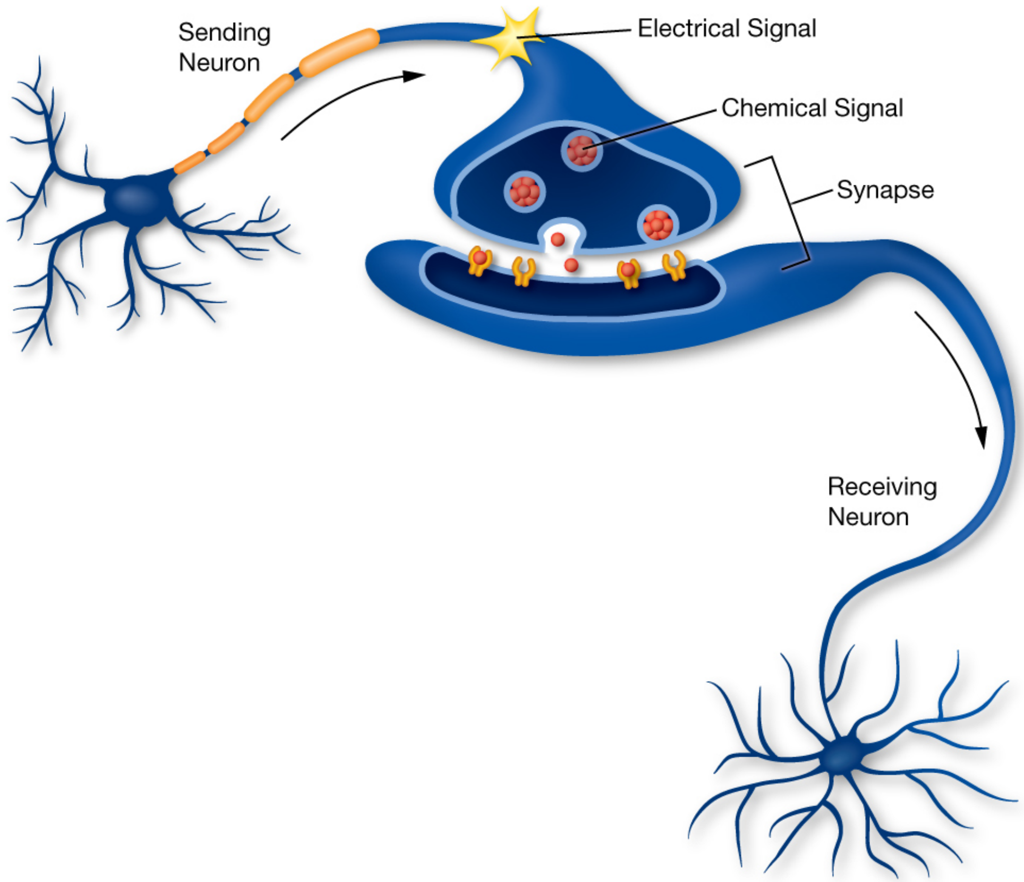


Image courtesy: `http://learn.genetics.utah.edu/content/addiction/neurons/`

“Real” Neuron

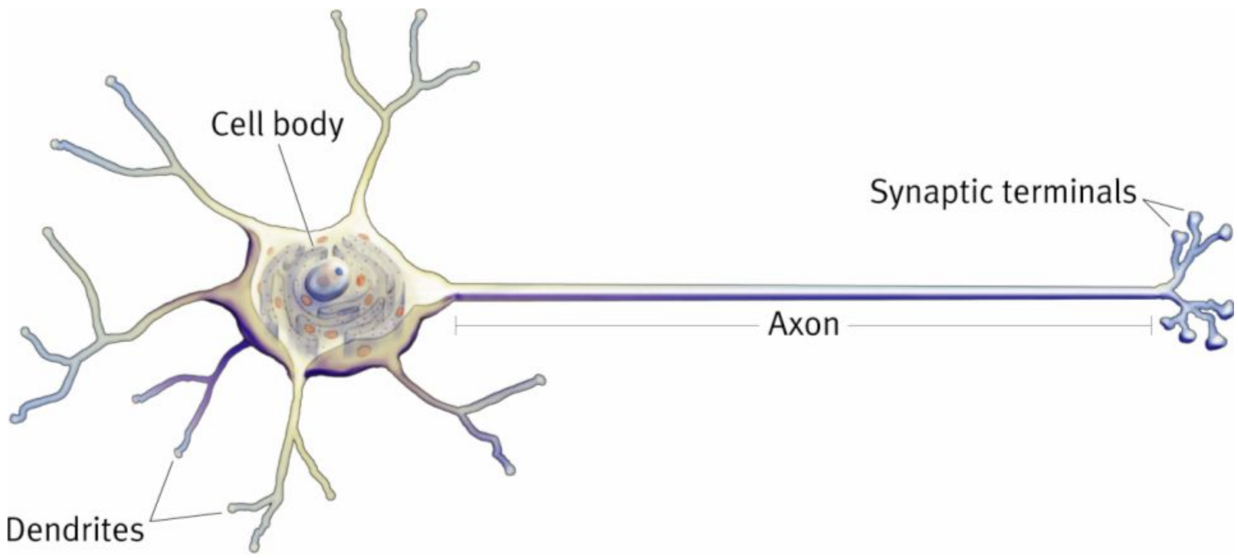
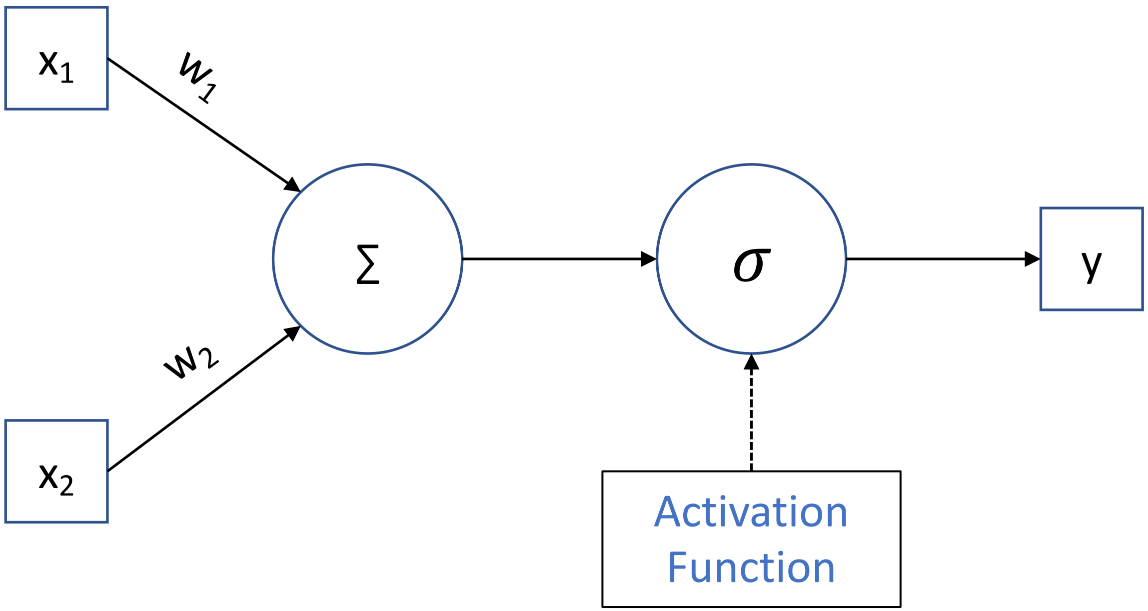
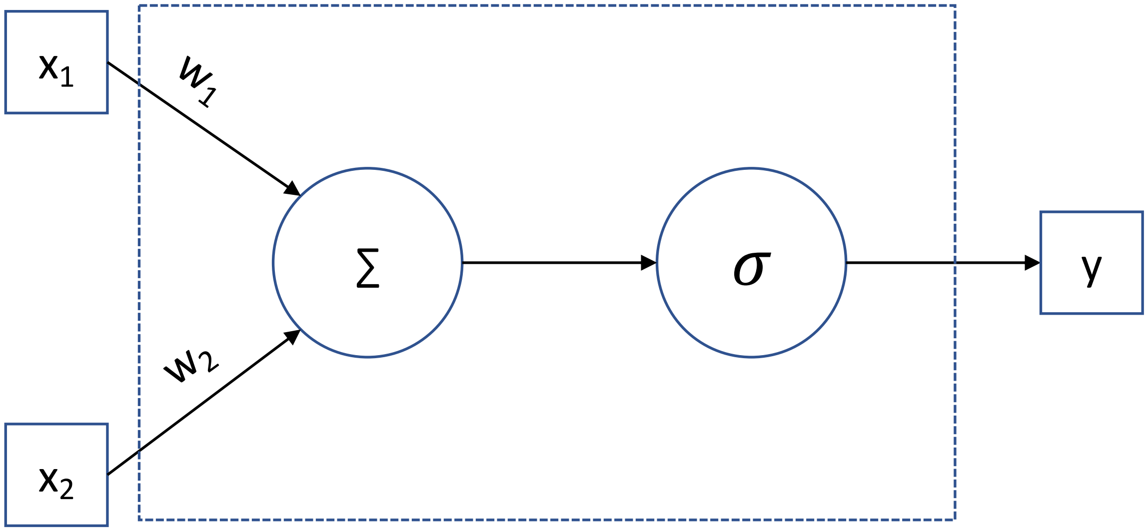


Image courtesy: `http://bio3520.nicerweb.com/Locked/chap/ch03/neuron.html`

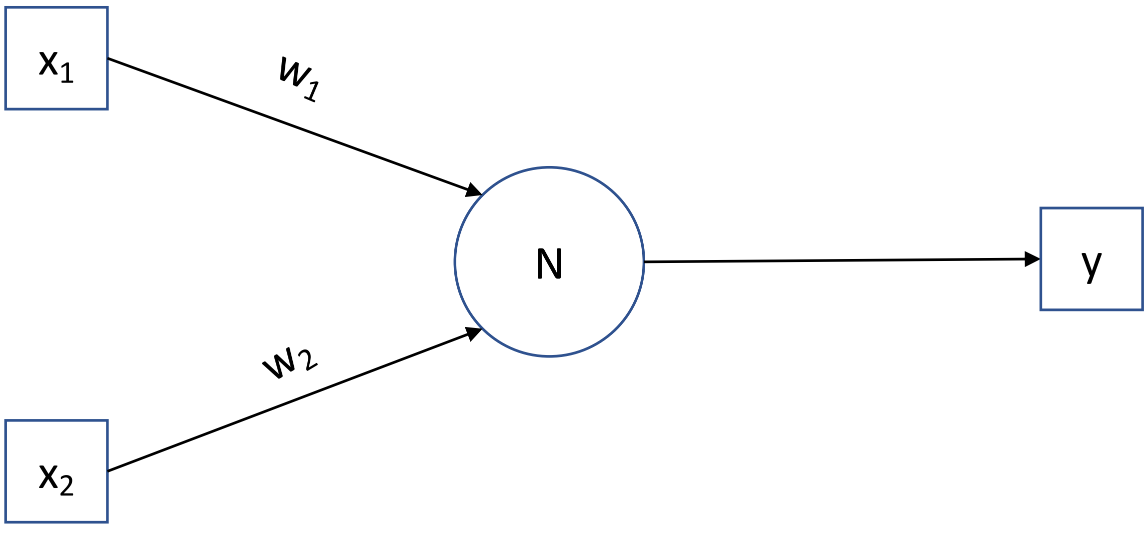
Artificial Neurons



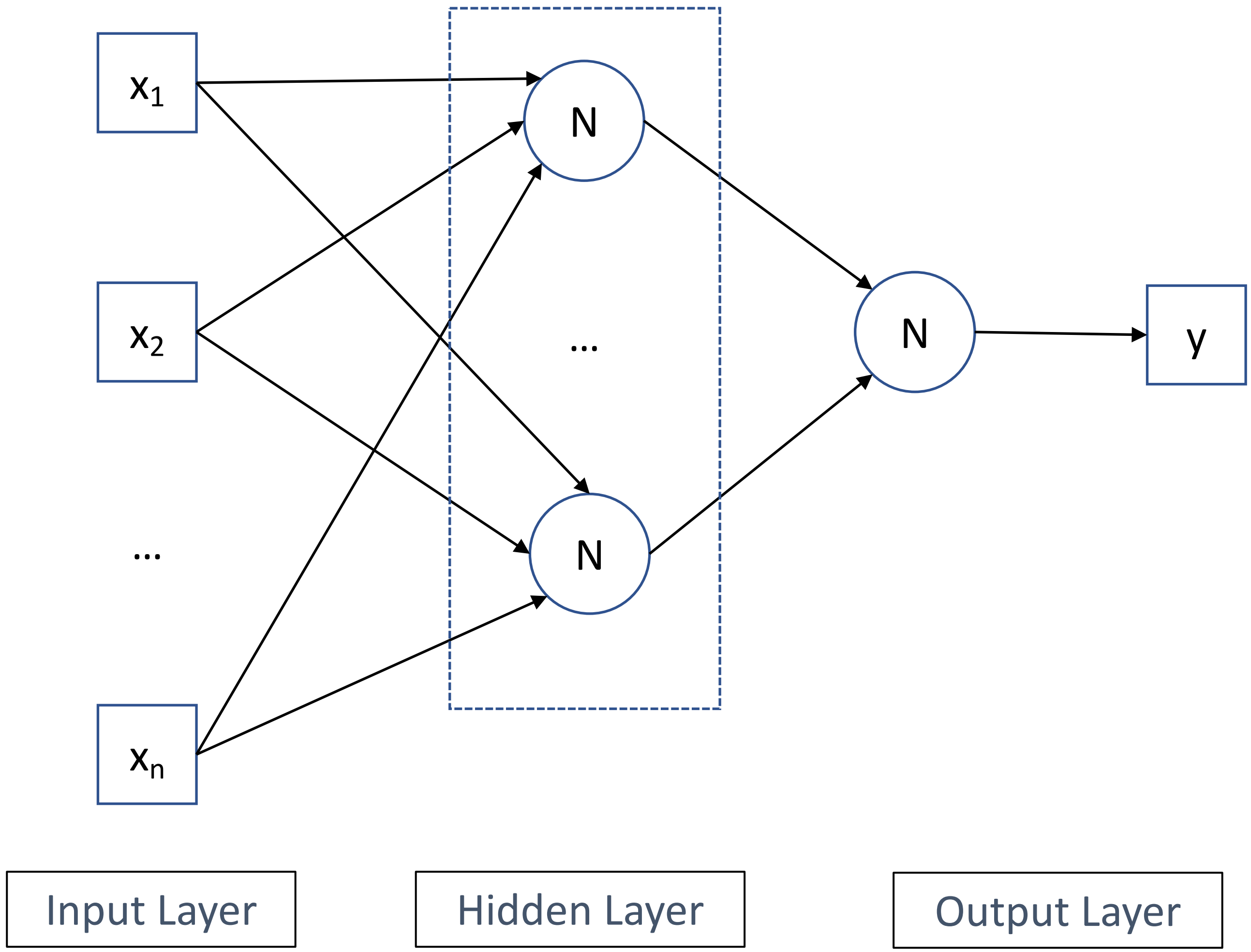
Artificial Neurons



Artificial Neurons



Fully-Connected Neural Network



Backpropagation:

Initialize all weights in the network to small, random numbers.

loop

for each training example $(\mathbf{x}, y)$ **do**

ForwardProp:

For each hidden unit h, $a_h = \sigma(net_h) = \sigma(\sum_i w_{ih} x_i)$

$\hat{y} = a_k = \sigma(net_k) = \sigma(\sum_h w_h a_h)$

BackProp:

$\delta_k = \frac{\partial J}{\partial net_k} = (y - \hat{y})\hat{y}(1 - \hat{y})$

For each weight w_h, $w_h \leftarrow w_h - \eta \delta_k a_h$

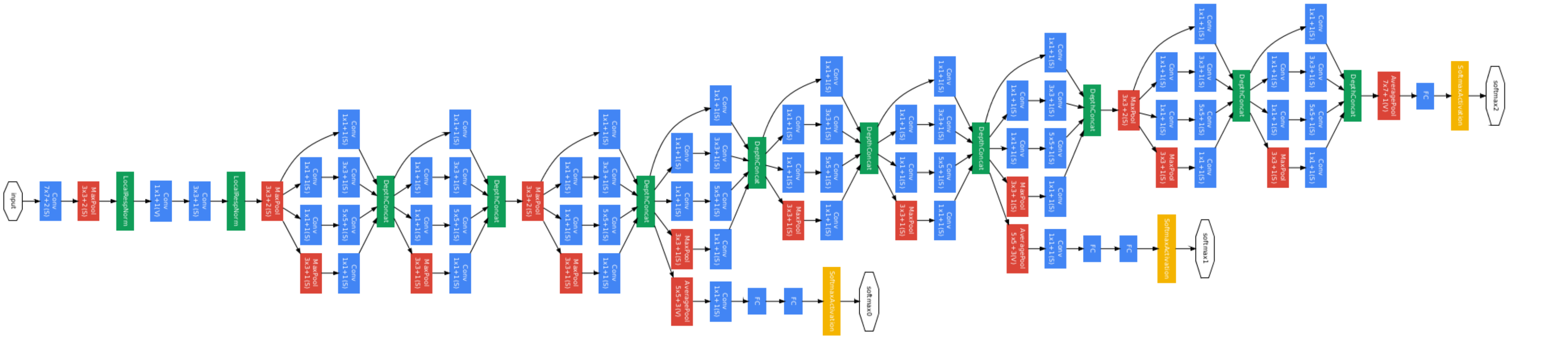
For each hidden unit h, $\delta_h = \delta_k w_h a_h (1 - a_h)$

For each weight w_{ih}, $w_{ih} \leftarrow w_{ih} - \eta \delta_h x_i$

end for

end loop

Modern Neural Networks



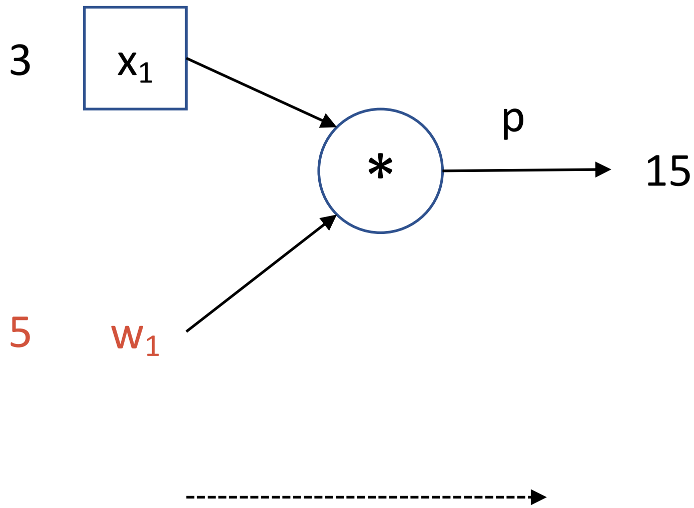
Automatic Differentiation

Automatic Differentiation

- Use the abstraction of a computational graph

- Define your computation and let engine worry about optimization

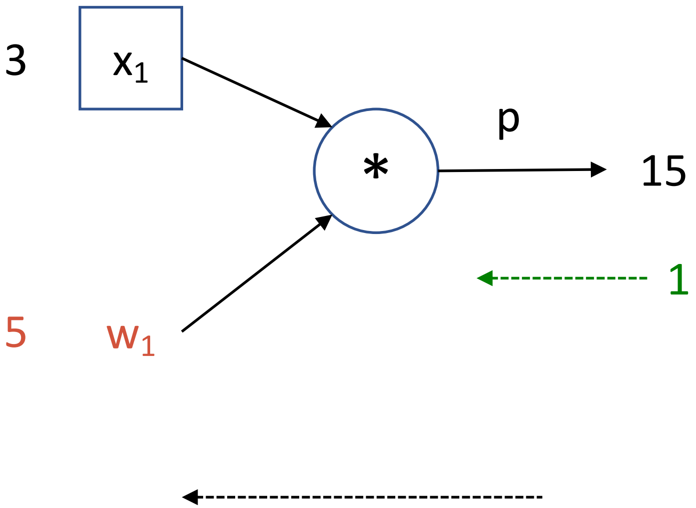
Computational Graph



Forward Pass

- Apply the operator

Computational Graph

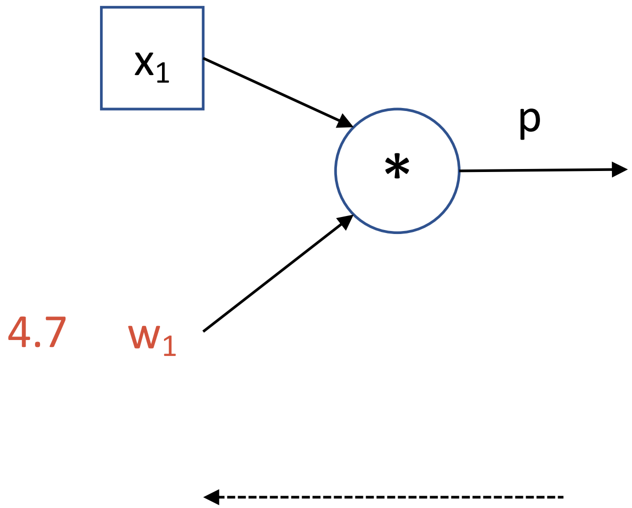


$$\frac{\partial p}{\partial w_1} = x_1$$

Backward Pass

- Adjust parameter using local gradient 3 (scaled by a learning rate)

Computational Graph

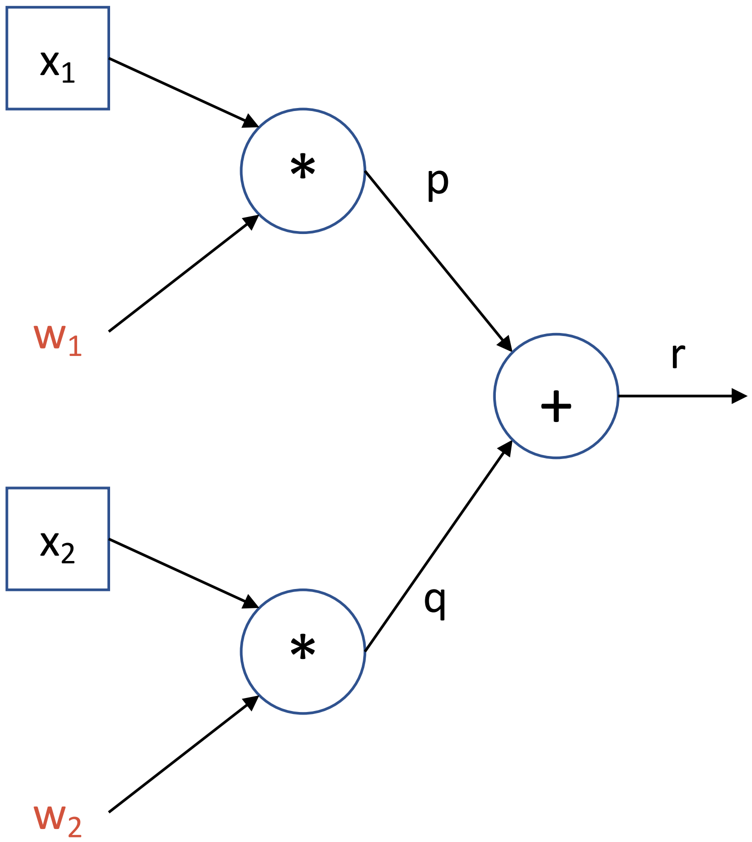


$$\frac{\partial p}{\partial w_1} = x_1$$

Backward Pass

- Adjust parameter using local gradient 3 (scaled by a learning rate)

Computational Graph

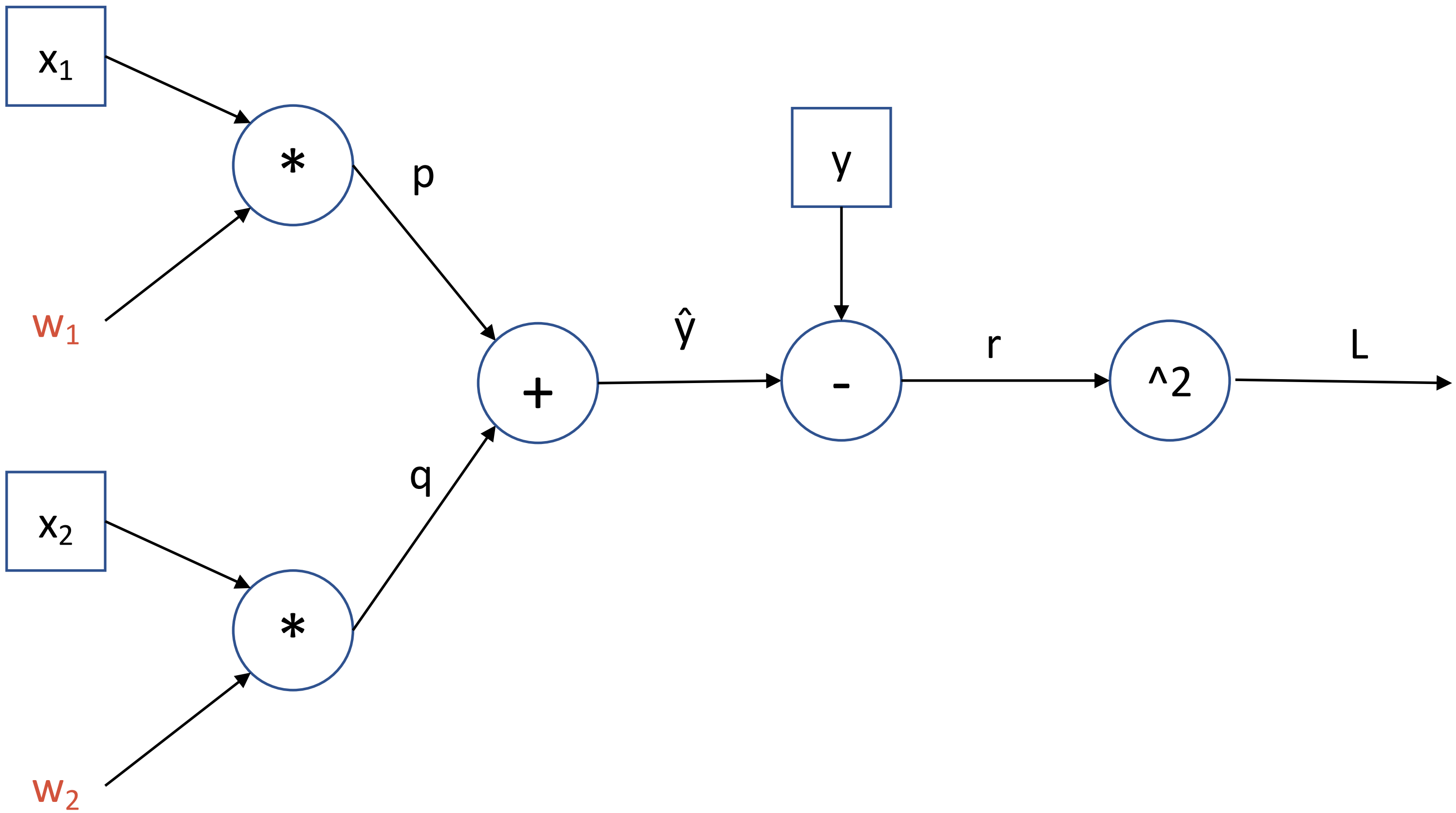


$$\frac{\partial r}{\partial p} = 1$$

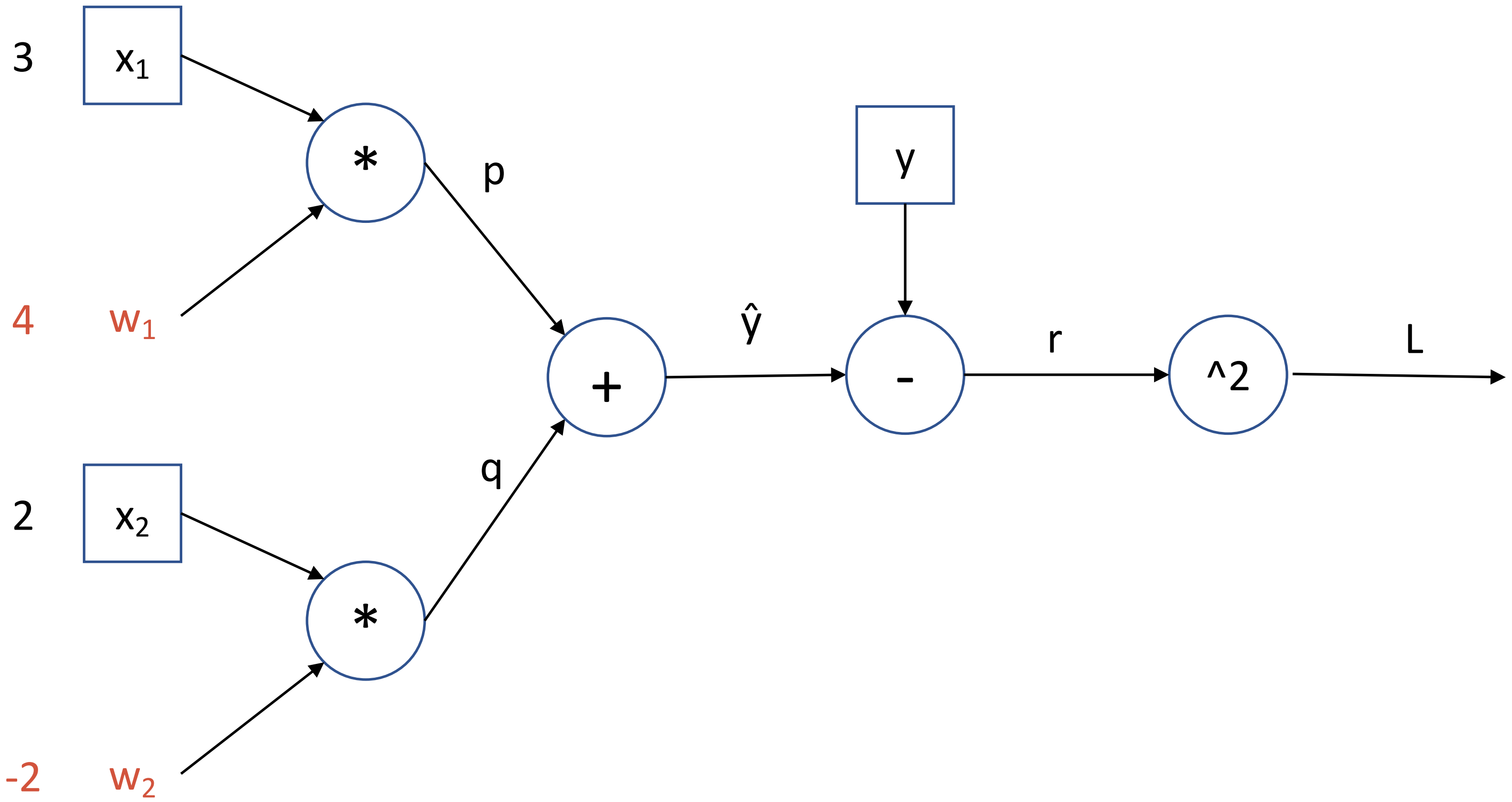
$$\frac{\partial p}{\partial w_1} = x_1$$

$$\frac{\partial r}{\partial w_1} = \frac{\partial r}{\partial p}\frac{\partial p}{\partial w_1}$$

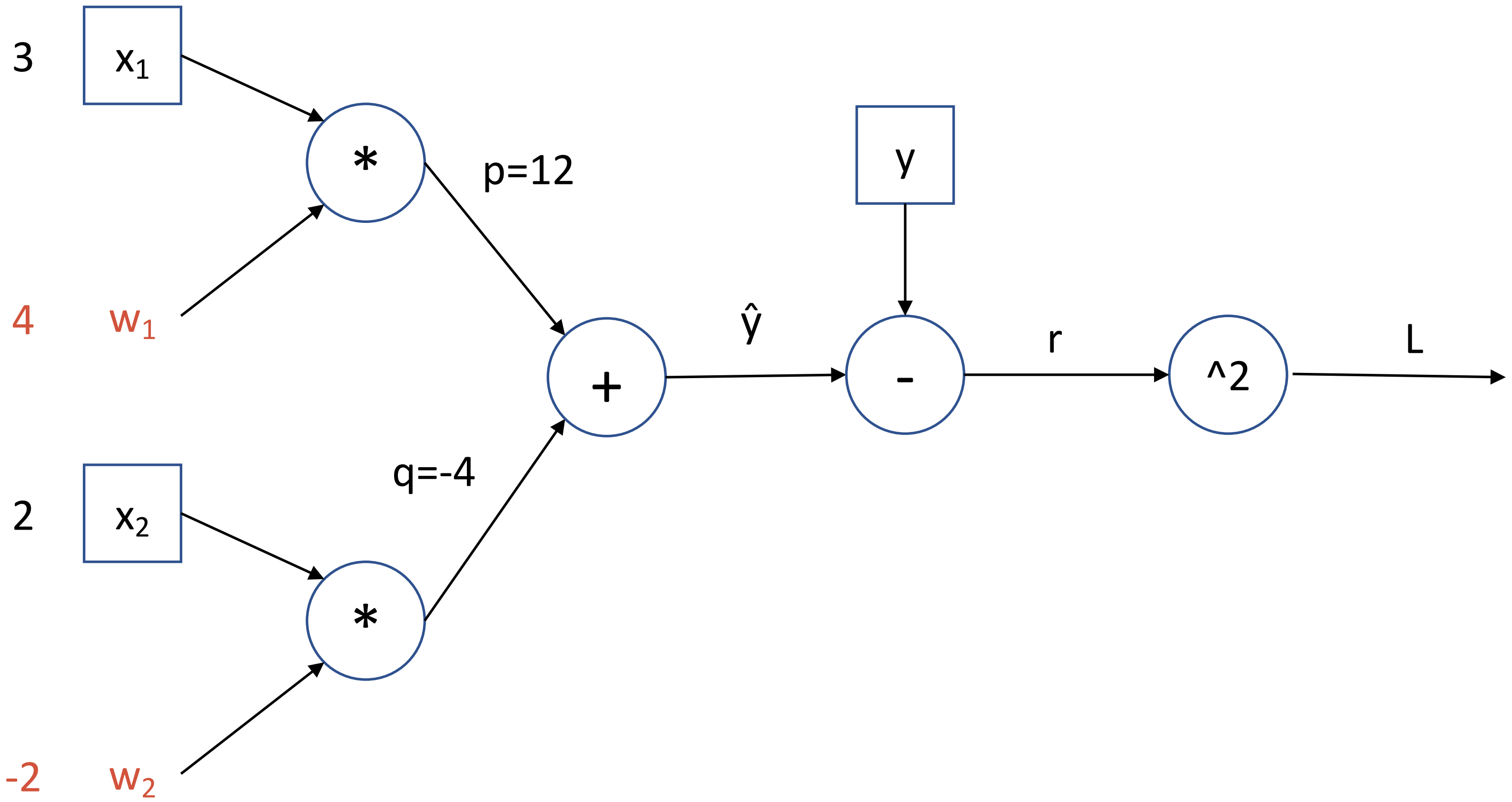
Computational Graph



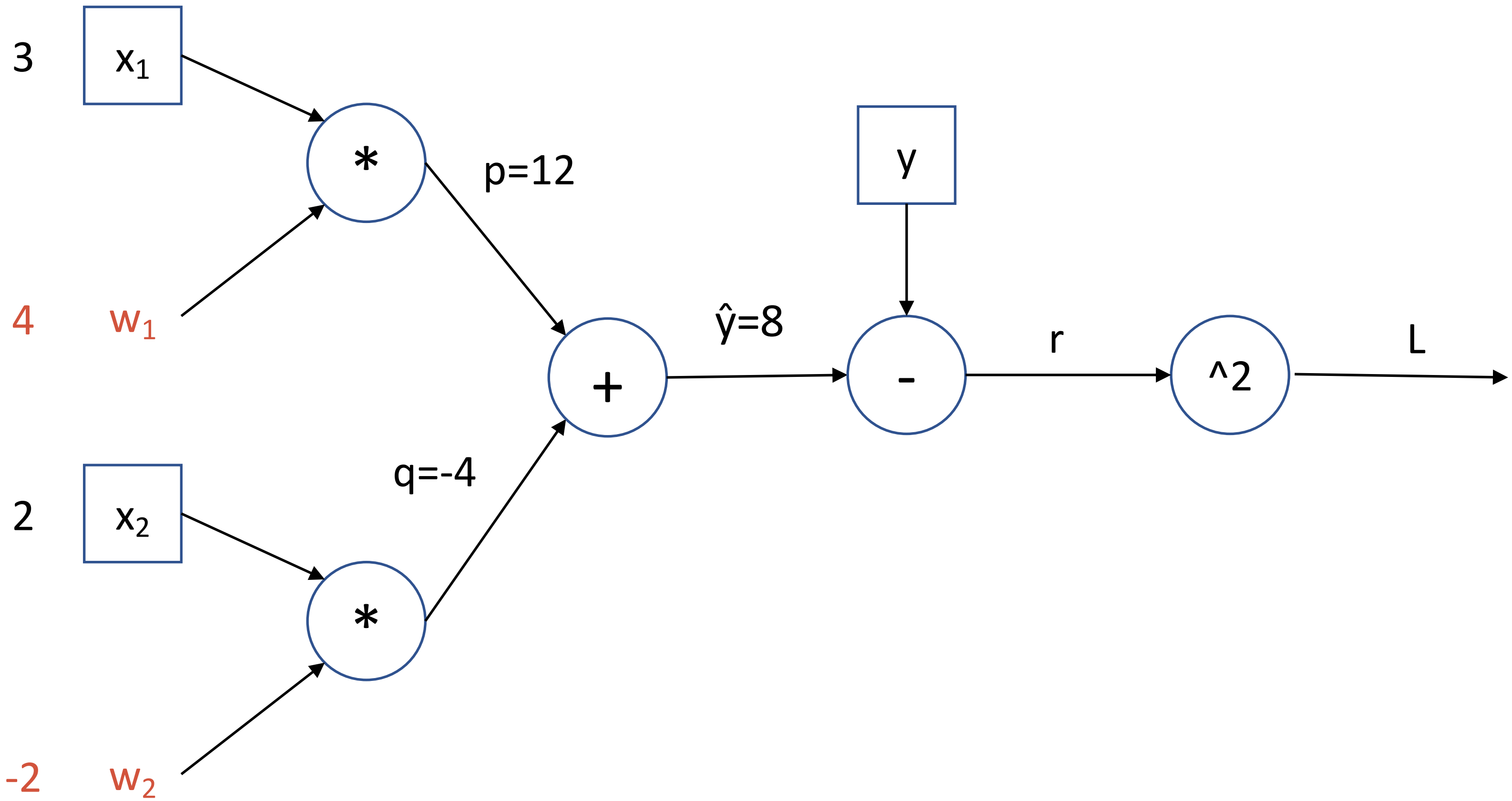
Computational Graph



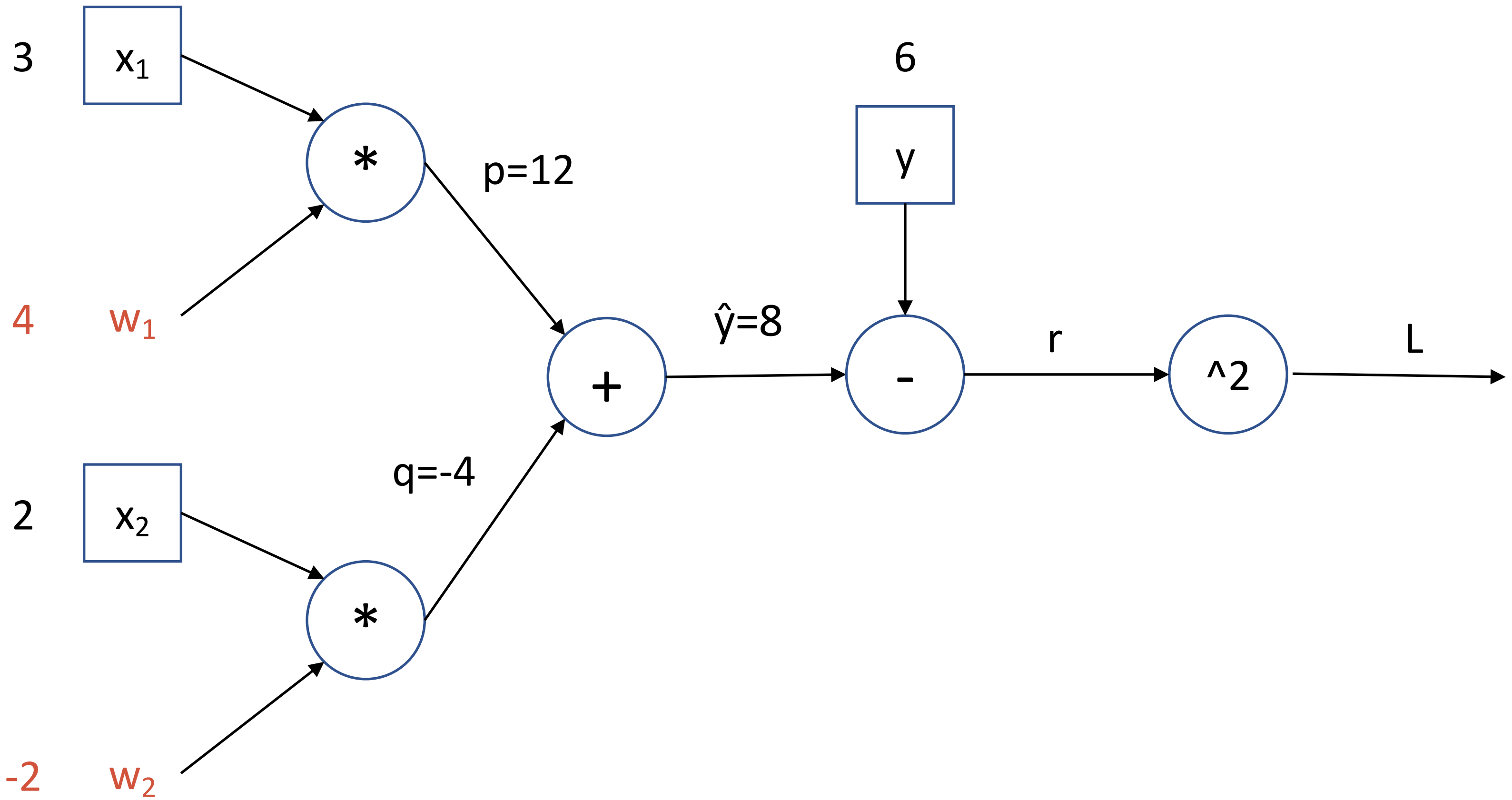
Computational Graph



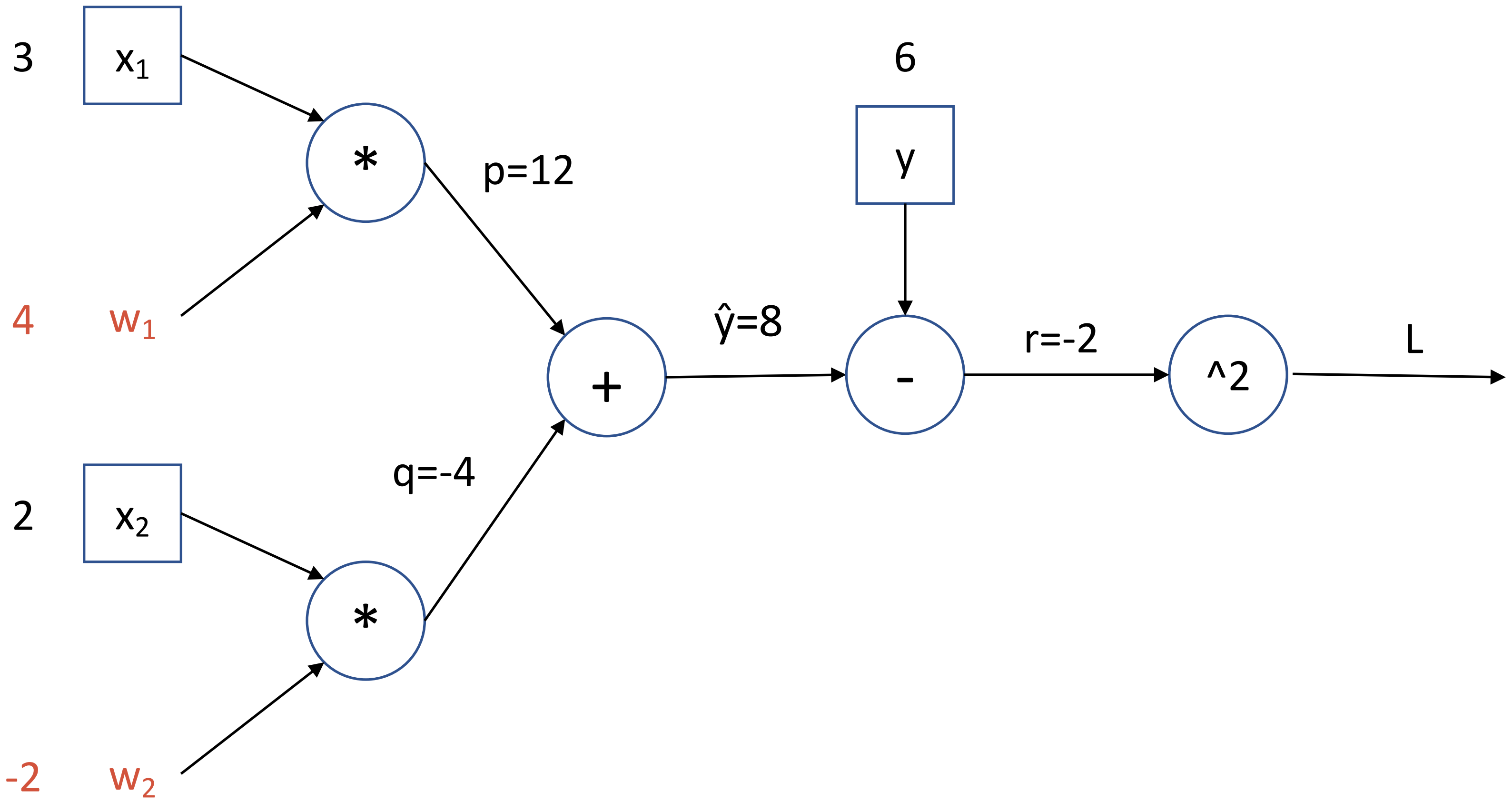
Computational Graph



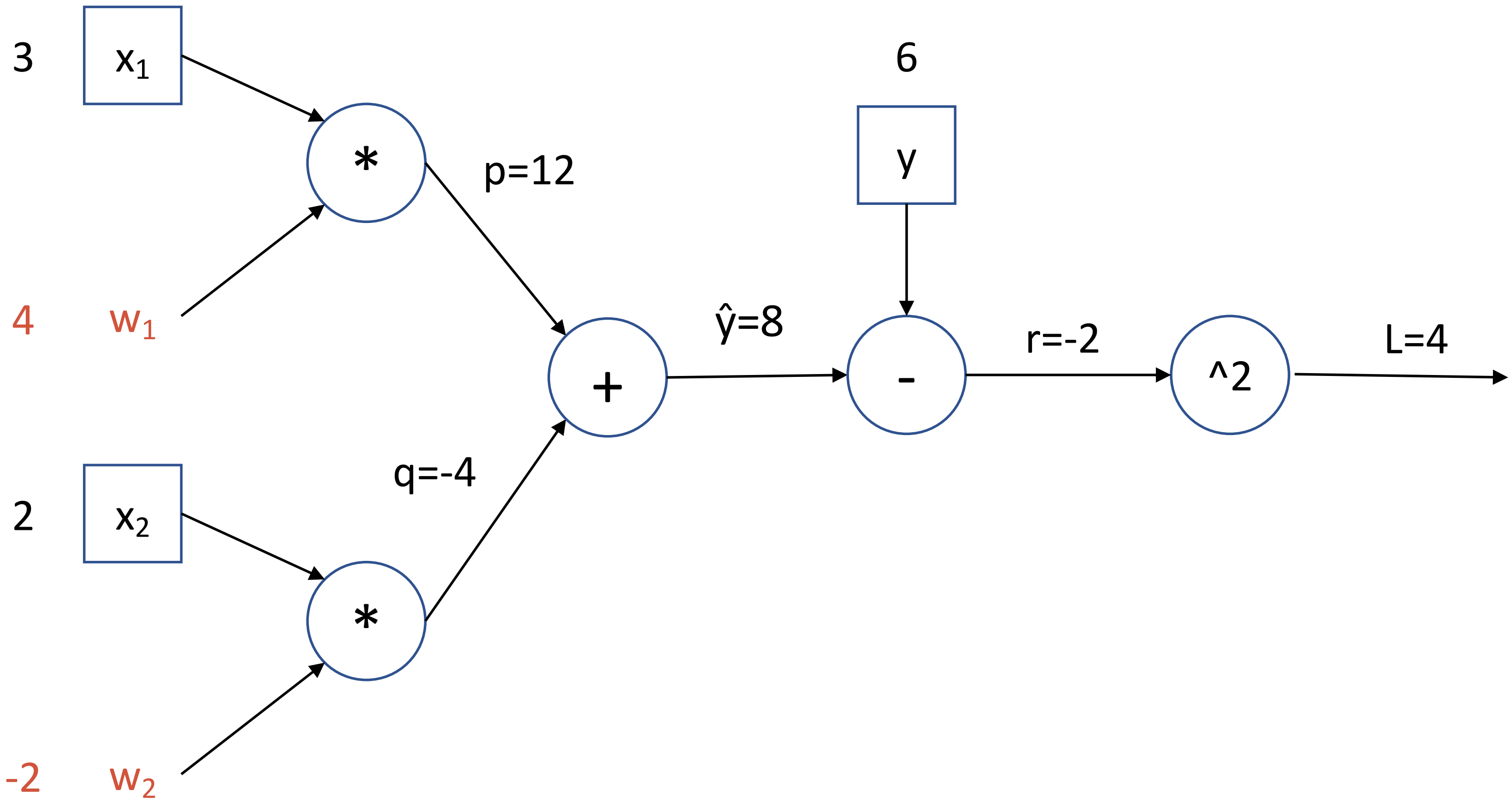
Computational Graph



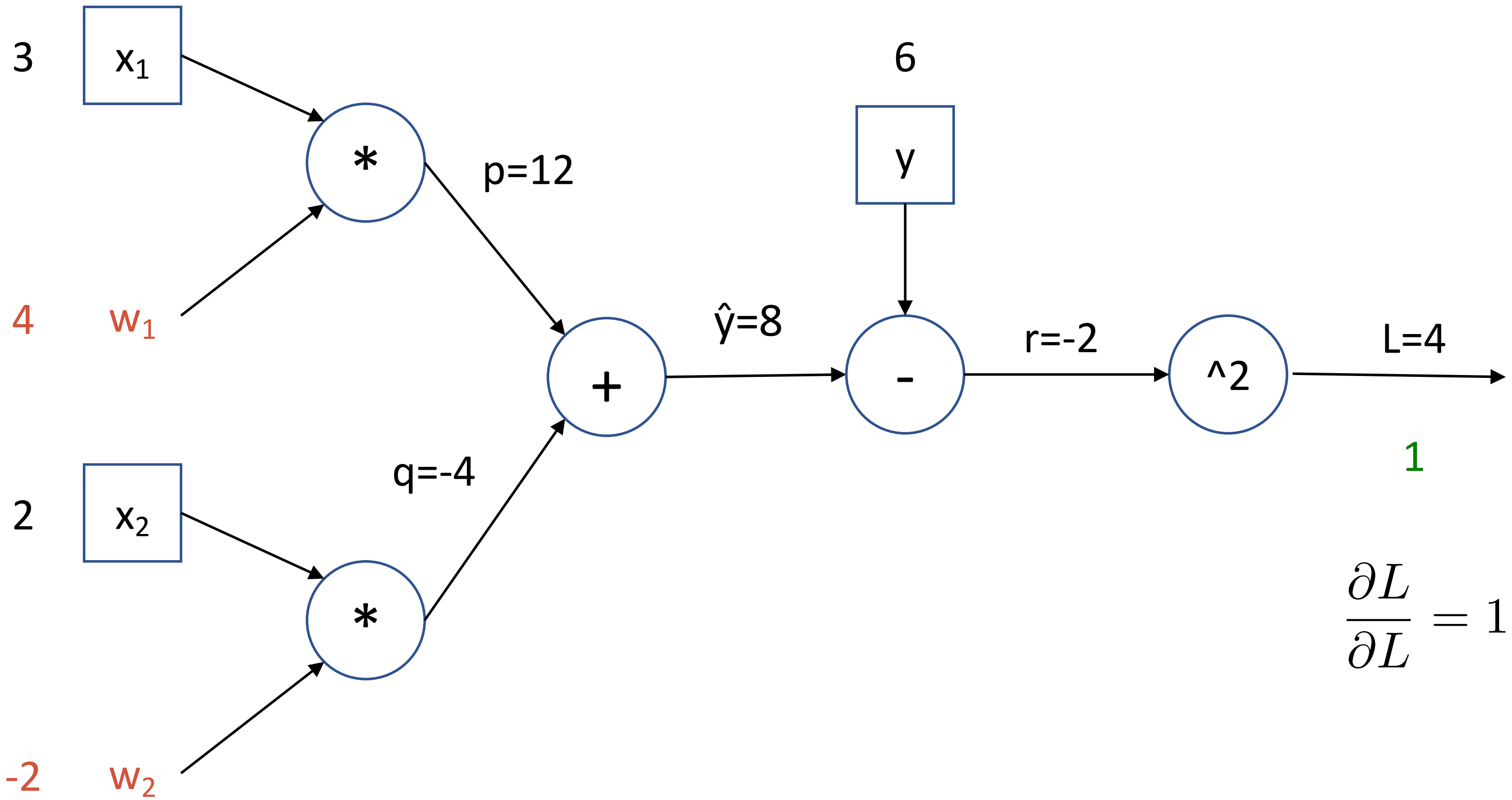
Computational Graph



Computational Graph

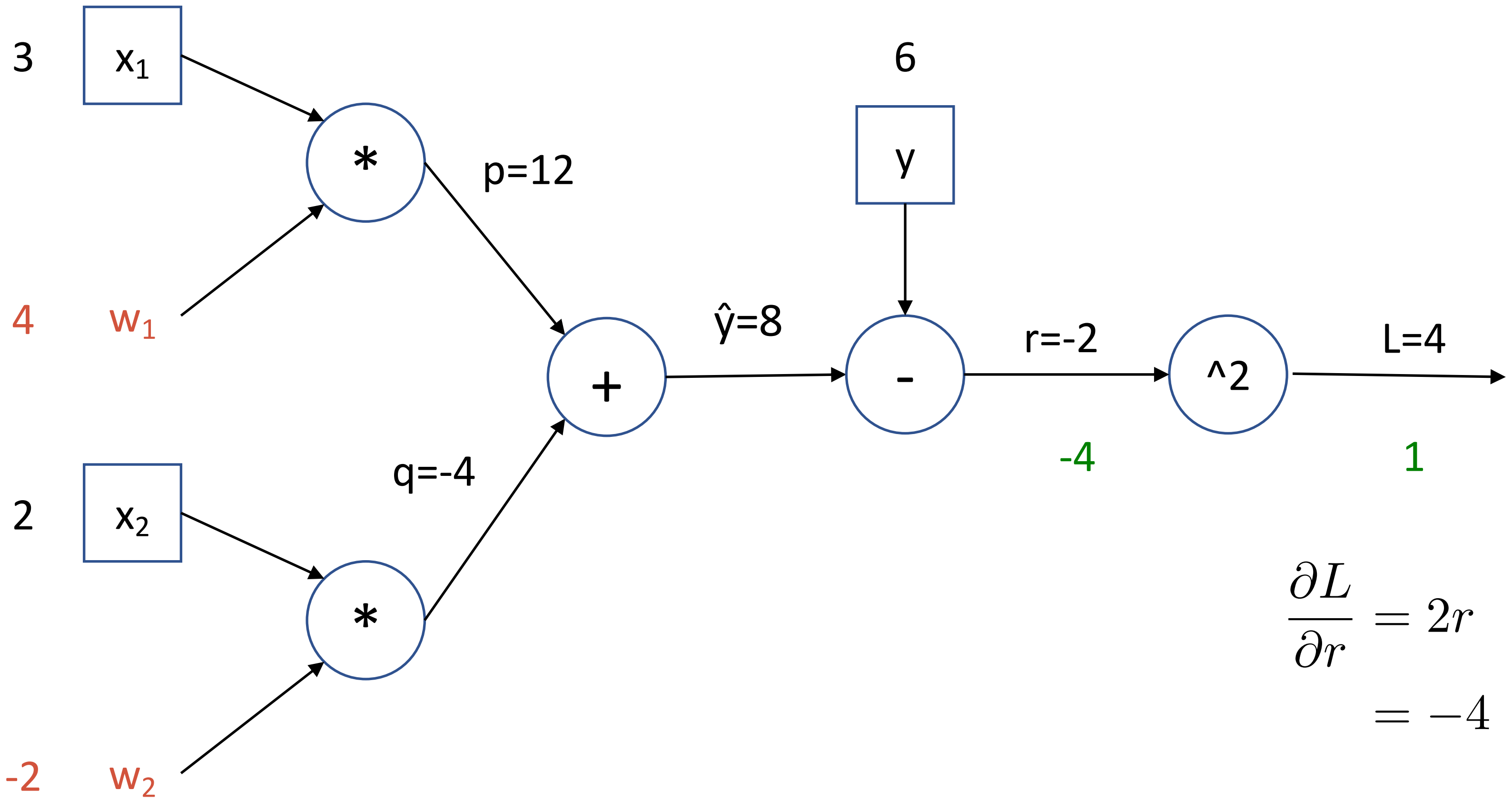


Computational Graph



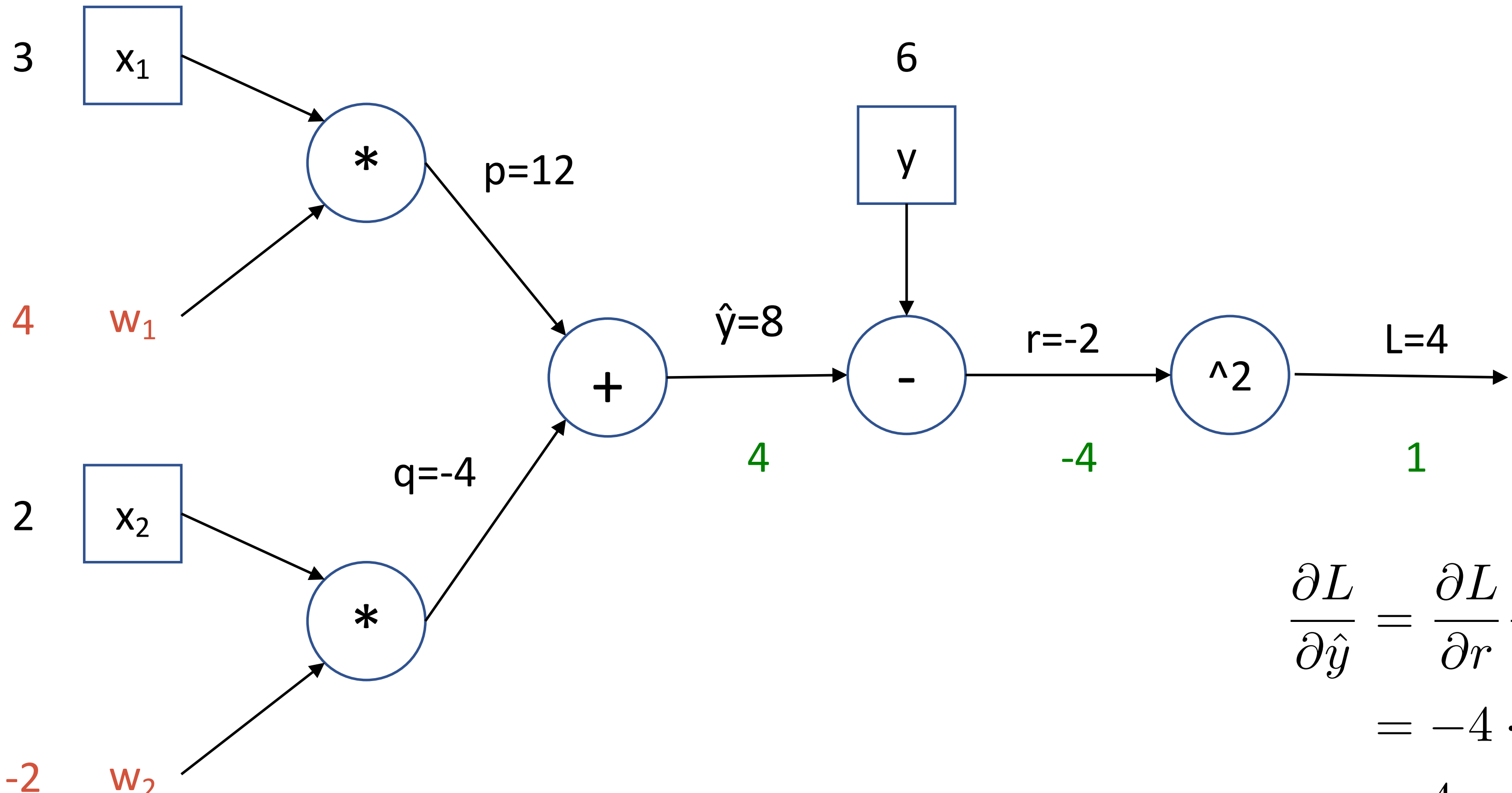
$$\frac{\partial L}{\partial L} = 1$$

Computational Graph



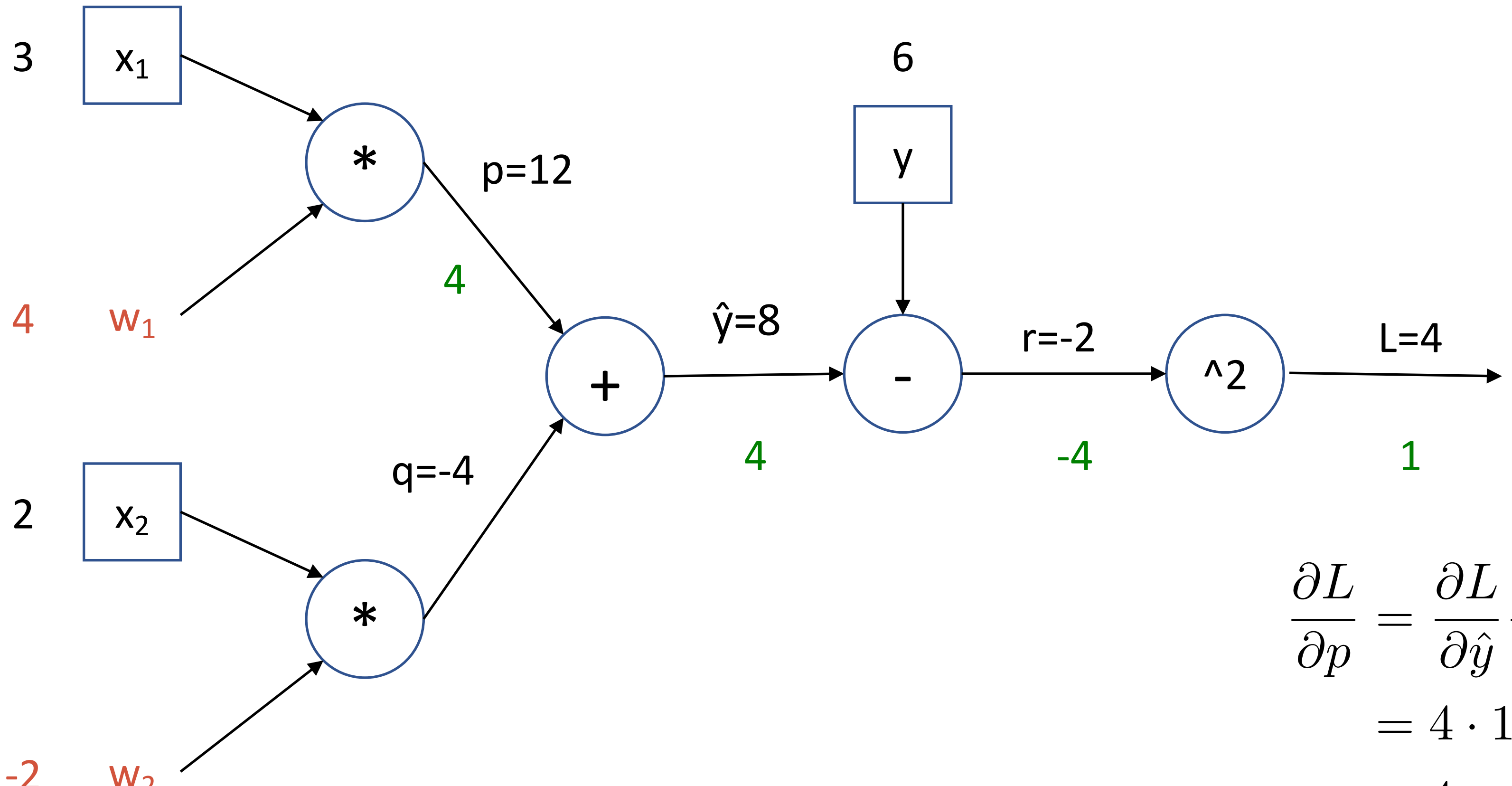
$$\frac{\partial L}{\partial r} = 2r$$
$$= -4$$

Computational Graph



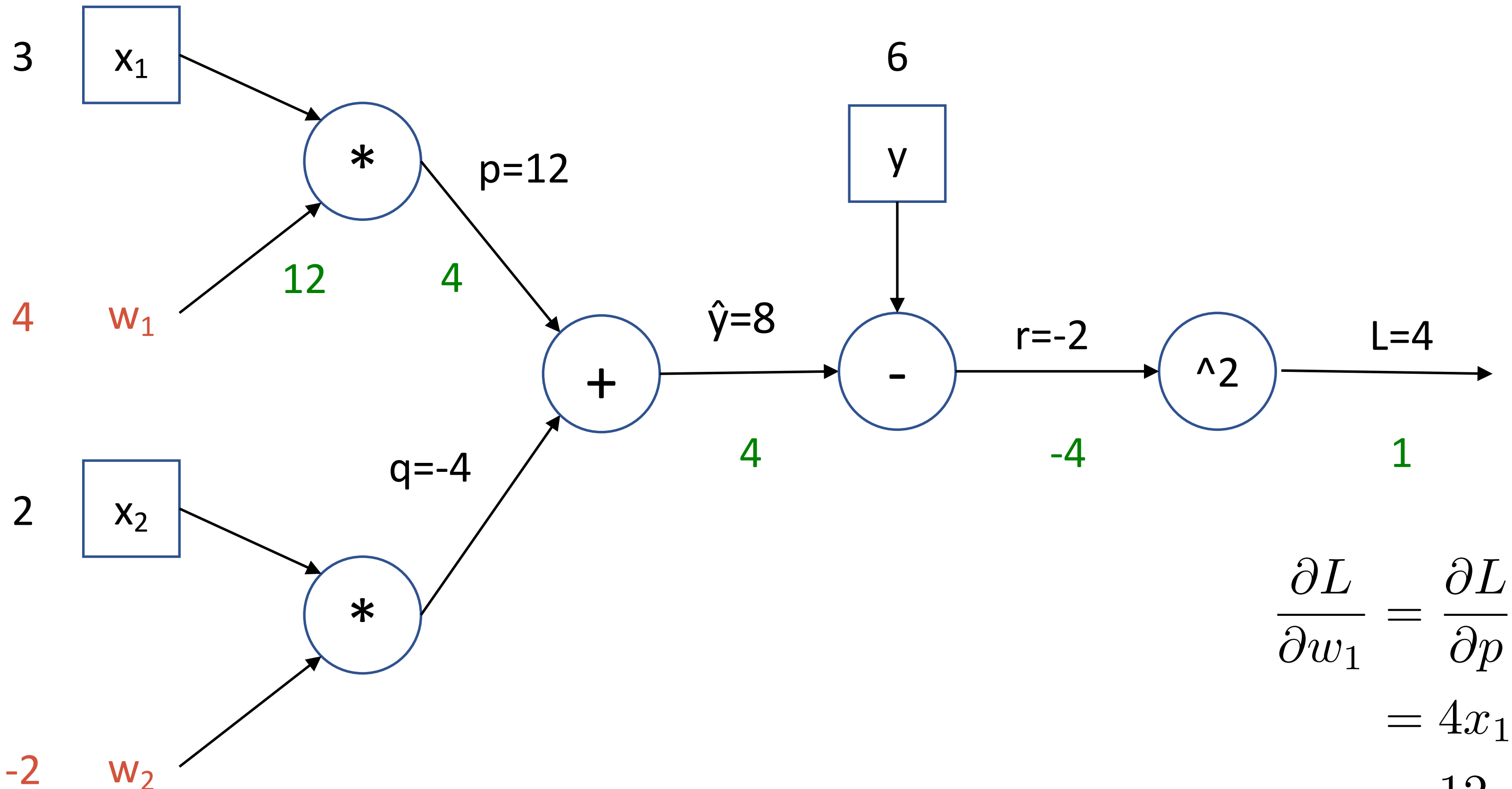
$$\frac{\partial L}{\partial \hat{y}} = \frac{\partial L}{\partial r}\frac{\partial r}{\partial \hat{y}}$$
$$= -4 \cdot -1$$
$$= 4$$

Computational Graph



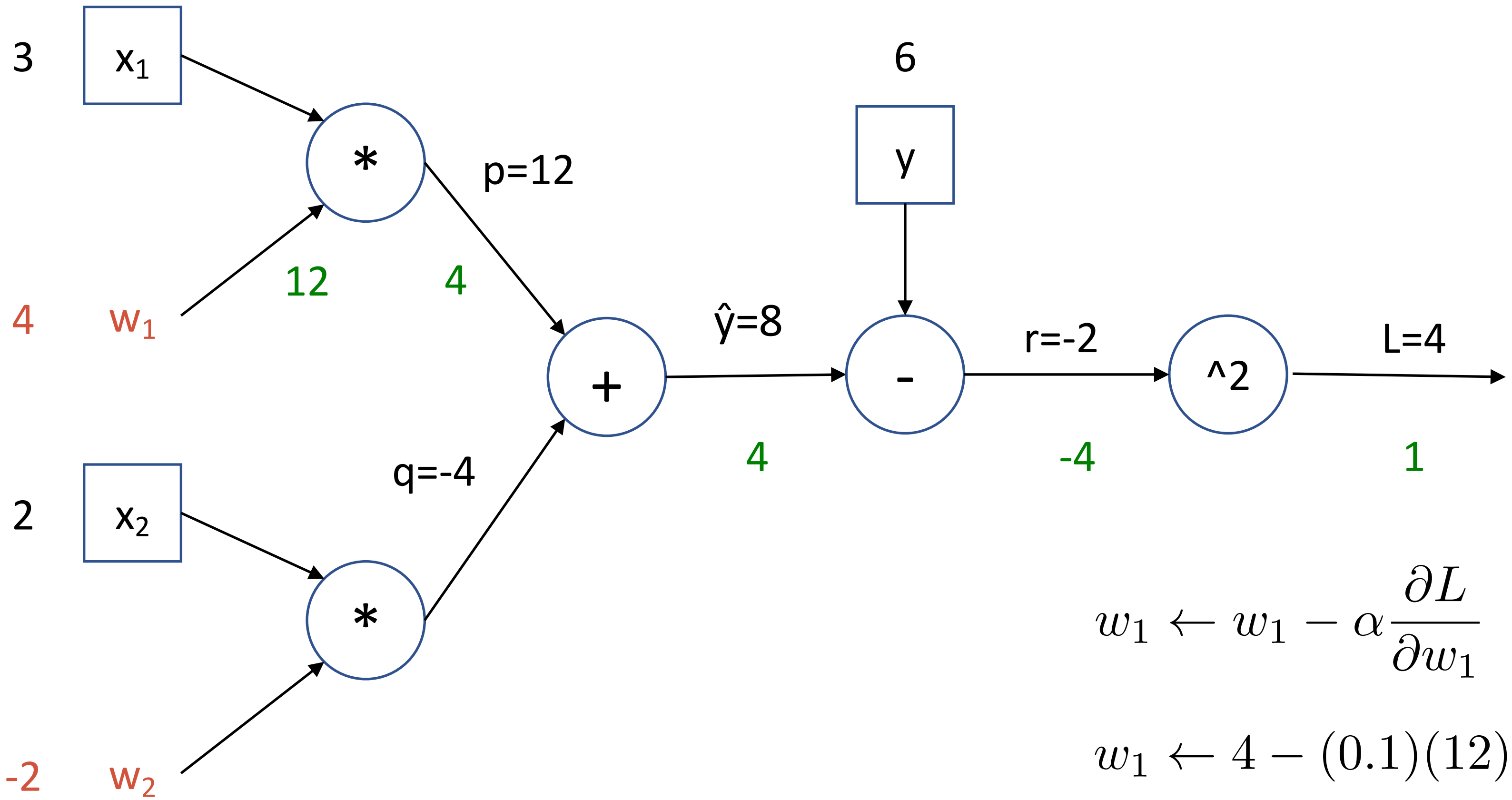
$$
\begin{aligned}
\frac{\partial L}{\partial p} &= \frac{\partial L}{\partial \hat{y}} \frac{\partial \hat{y}}{\partial p} \\
&= 4 \cdot 1 \\
&= 4
\end{aligned}
$$

Computational Graph



$$\frac{\partial L}{\partial w_1} = \frac{\partial L}{\partial p}\frac{\partial p}{\partial w_1}$$
$$= 4x_1$$
$$= 12$$

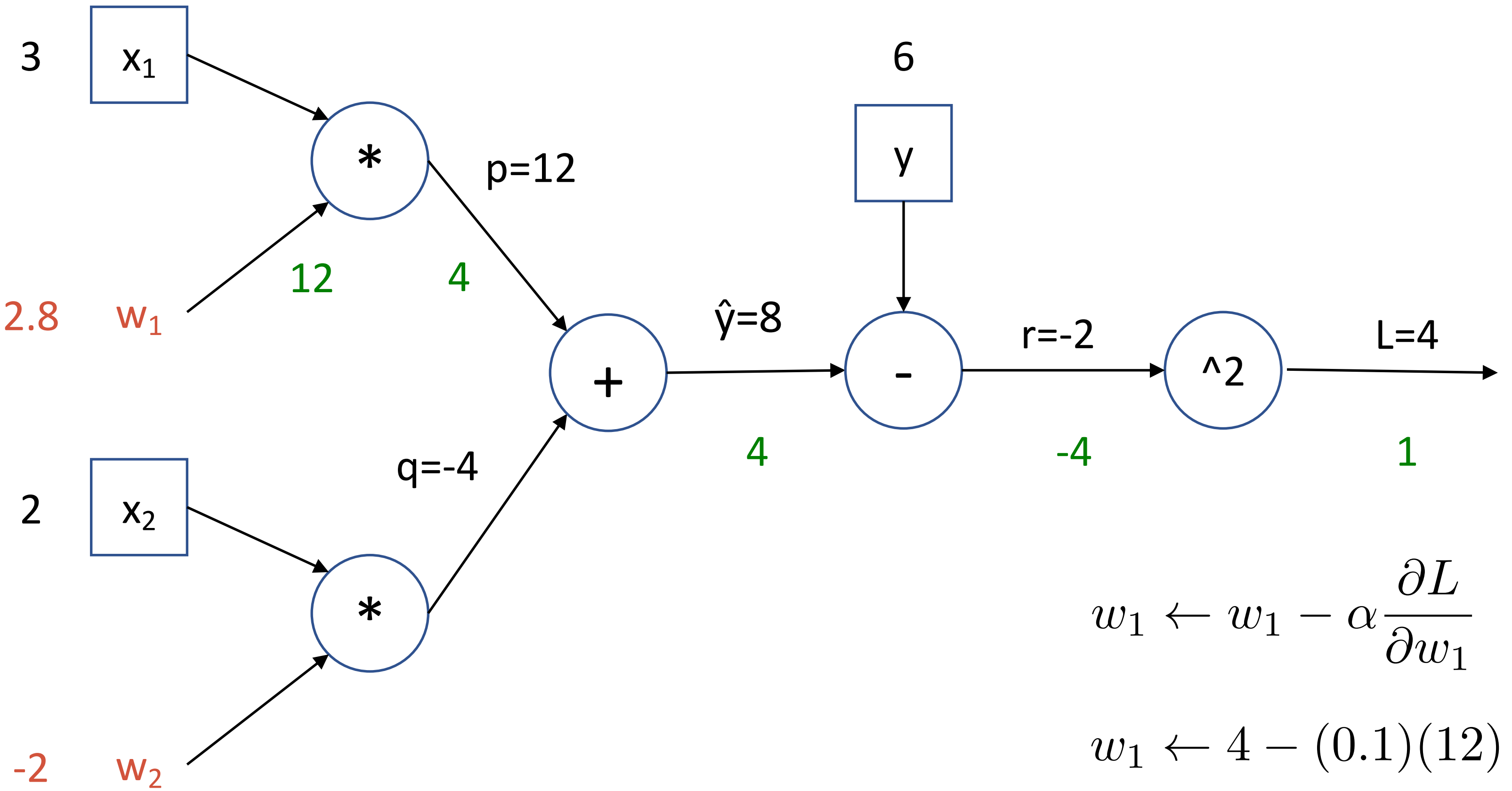
Computational Graph



$$w_1 \leftarrow w_1 - \alpha \frac{\partial L}{\partial w_1}$$

$$w_1 \leftarrow 4 - (0.1)(12) = 2.8$$

Computational Graph

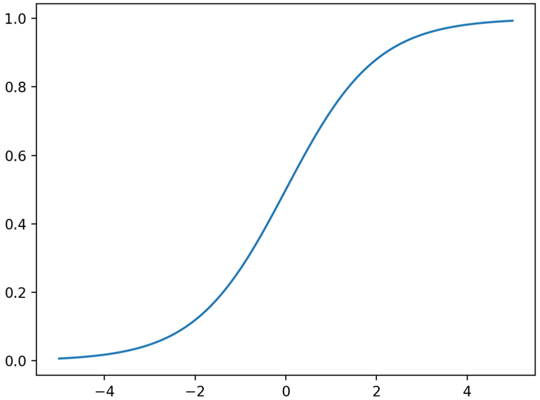


$$w_1 \leftarrow w_1 - \alpha \frac{\partial L}{\partial w_1}$$

$$w_1 \leftarrow 4 - (0.1)(12) = 2.8$$

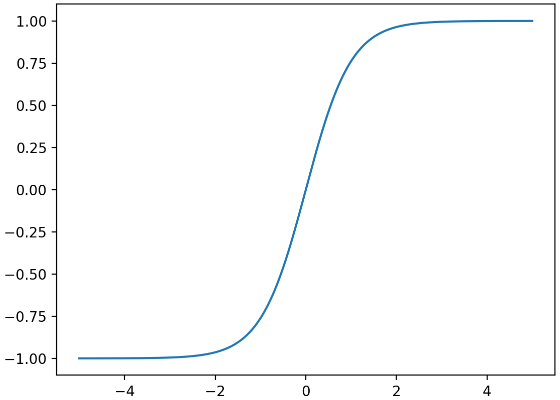
Dynamics of Learning in Deep Networks

Choosing an Activation Function



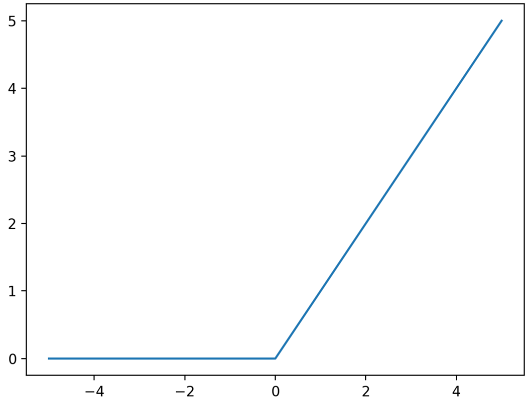
$$f(x) = \frac{1}{1 + e^{-x}}$$

Sigmoid



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

Hyperbolic Tangent
(tanh)



$$f(x) = \max(0, x)$$

Rectified Linear Unit
(ReLU)

Initializing Network Weights

- Set all weights to 0?
  - Bad idea

- Set all weights to random values?
  - For very deep networks, gradients will vanish

- Main insight: want to keep variance of activations roughly same across layers
  - Xavier initialization – for tanh/sigmoid networks
  - He initialization – for ReLu networks
  - Both take into account fan-in/fan-out of each unit

Optimization Methods

Optimization Methods

- (Stochastic) gradient descent

$$\mathbf{w} \leftarrow \mathbf{w} - \alpha \nabla L(\mathbf{w})$$

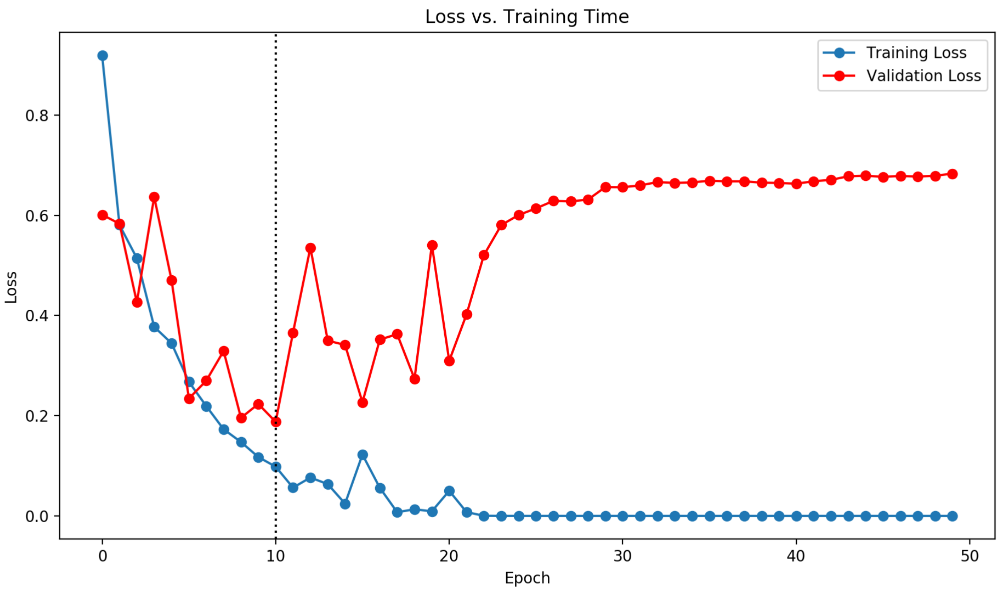
- Stochastic gradient descent + momentum

$$\mathbf{z} \leftarrow \beta \mathbf{z} + \nabla L(\mathbf{w})$$
$$\mathbf{w} \leftarrow \mathbf{w} - \alpha z$$

- Adaptive gradient approaches:
  - RMSProp
  - Adam

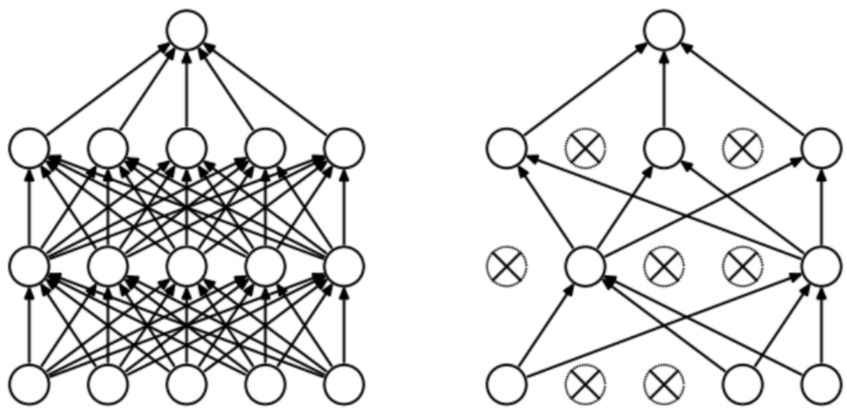
Regularization – Classical Approaches

- Weight decay
  - Add an L2 term to cost function
- Early stopping
  - “Regularization in time”



Regularization – Newer Approaches

- Dropout



(a) Standard Neural Net (b) After applying dropout.

- Batch normalization
  - Motivation: “internal covariate shift”
  - Idea: Normalize activations at every layer

Summary

- Automatic differentiation
- Better hardware + large datasets
- Activation functions with better gradient flow
- Heuristics for weight initialization
- Better optimization algorithms
- Batch normalization