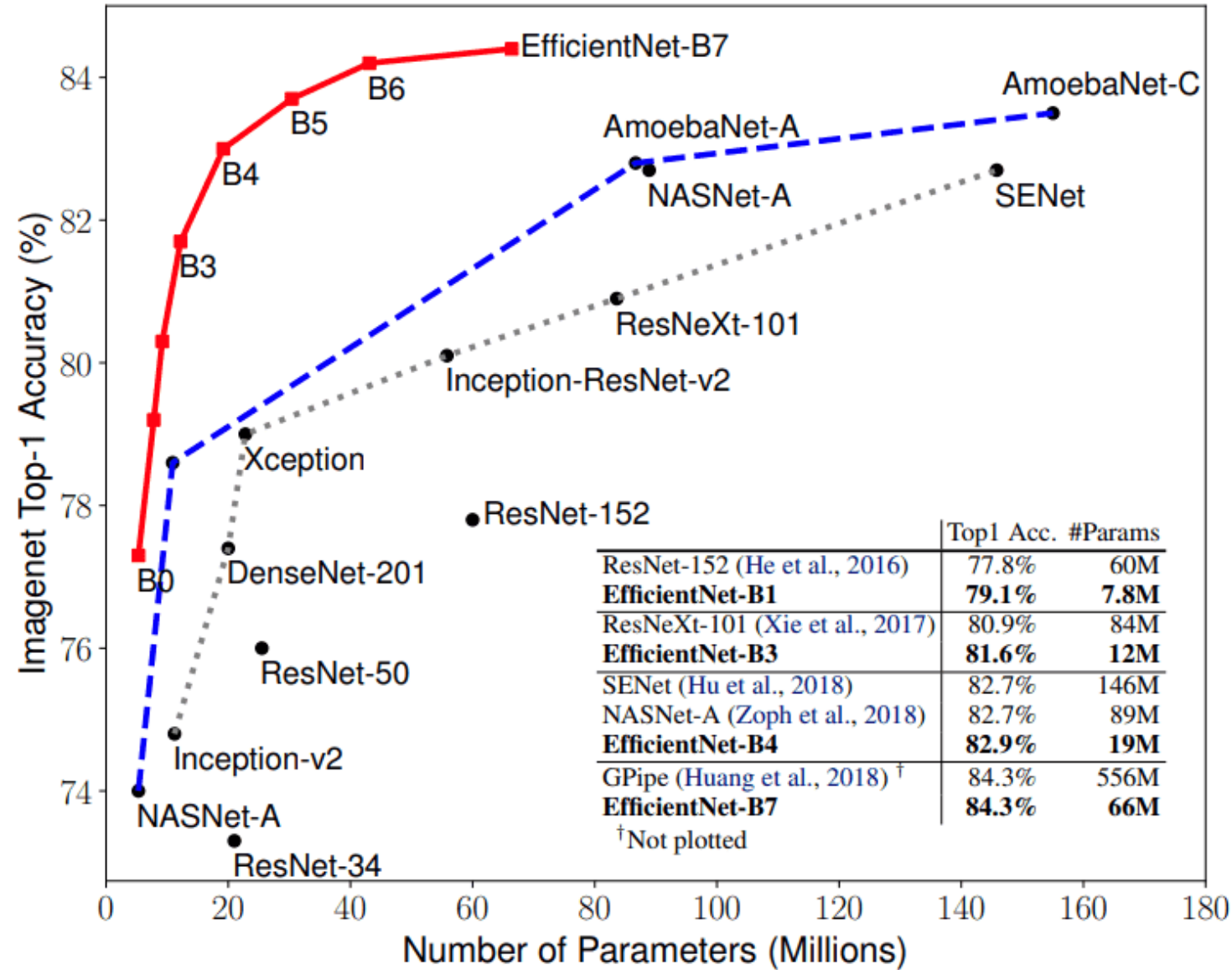
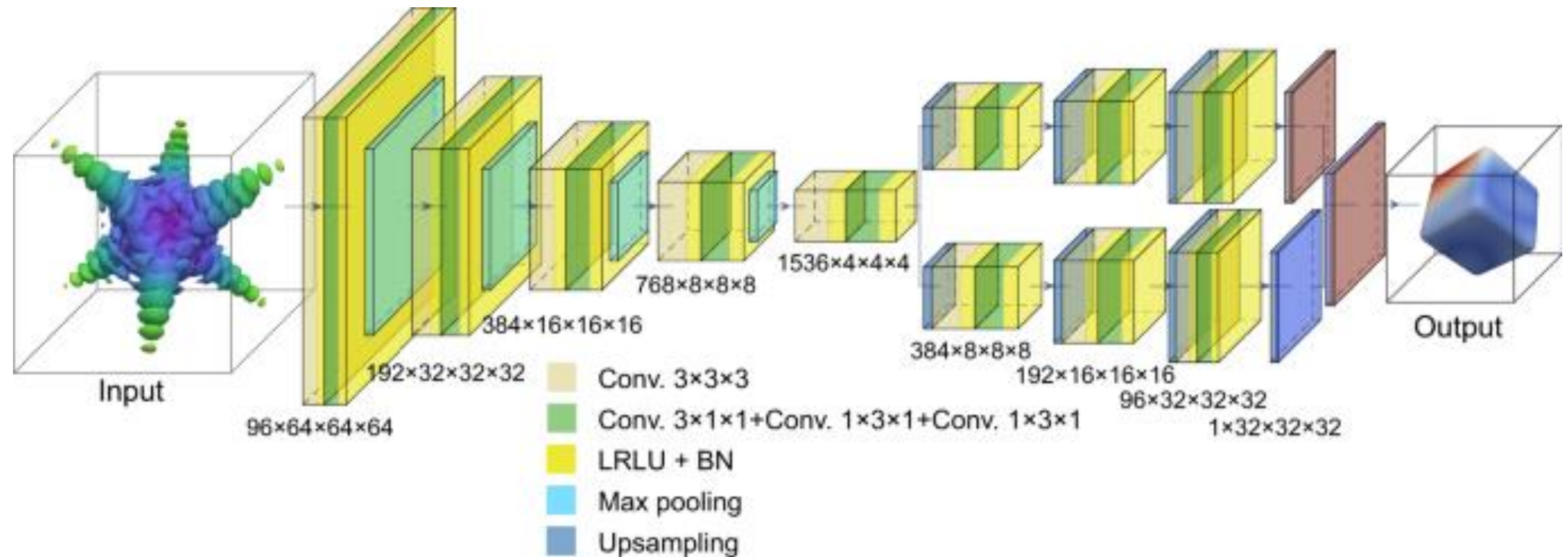


# Lecture 06: Simple Perceptron

Sergei V. Kalinin





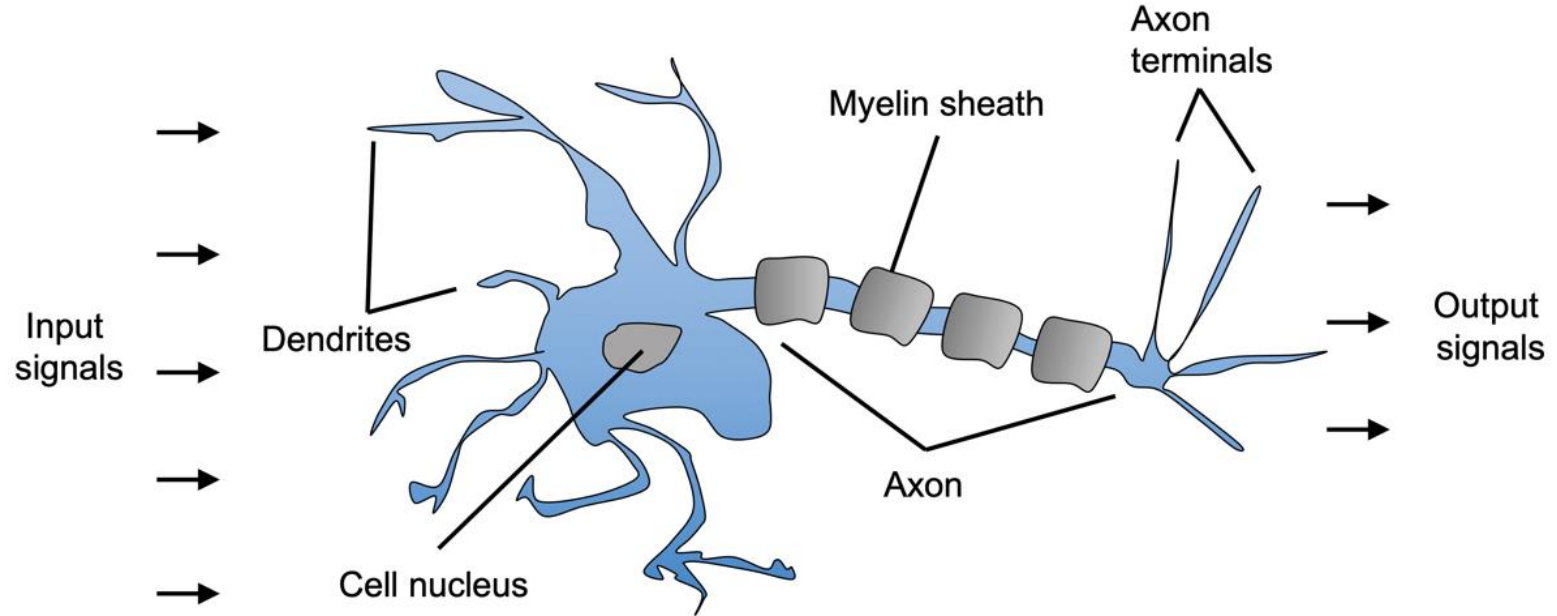
Longlong Wu, Shinjae Yoo, Ana F. Suzana, Tadesse A. Assefa, Jiecheng Diao, Ross J. Harder, Wonsuk Cha & Ian K. Robinson, *Three-dimensional coherent X-ray diffraction imaging via deep convolutional neural networks*

<https://www.nature.com/articles/s41524-021-00644-z>

# Brain structure and McCulloch-Pitts neuron

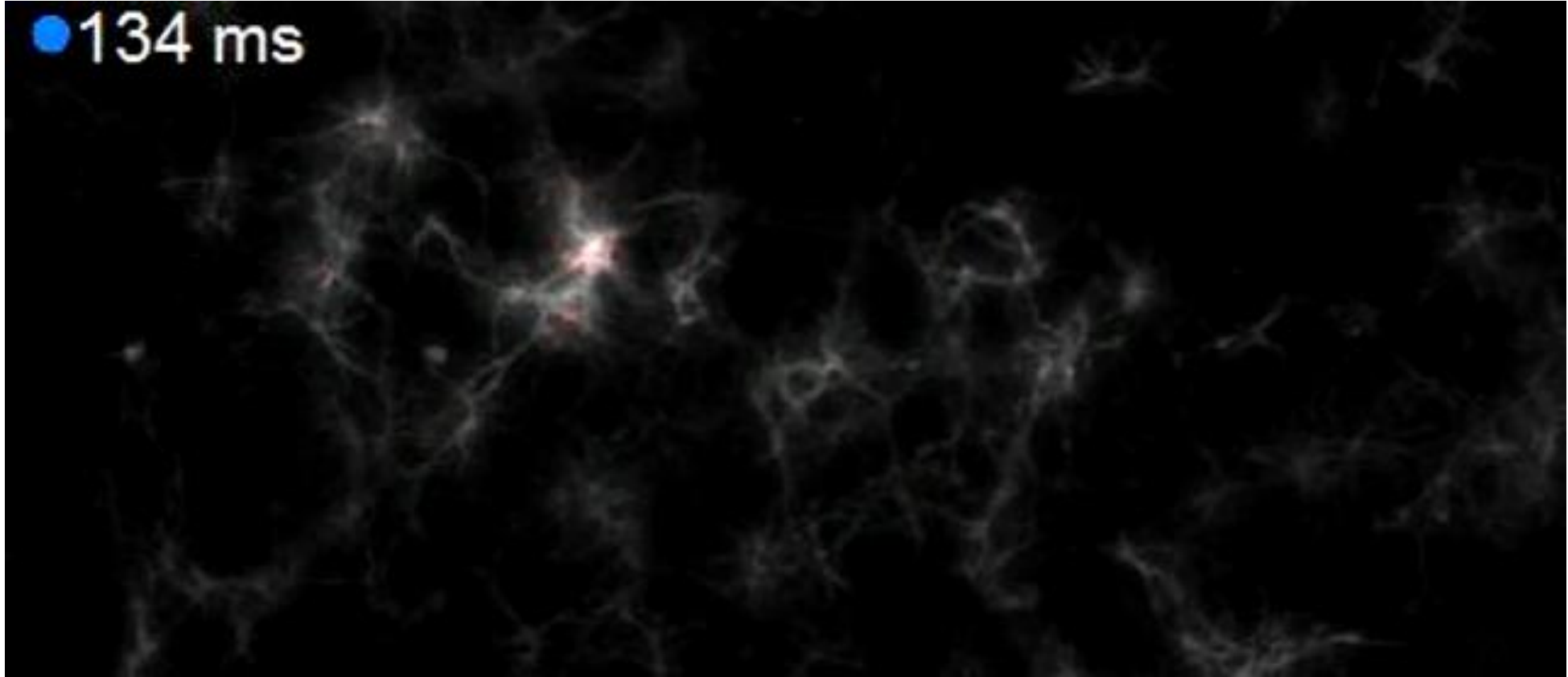


<https://www.kenhub.com/en/library/anatomy/histology-of-neurons>



*A Logical Calculus of the Ideas Immanent in Nervous Activity* by W. S. McCulloch and W. Pitts, *Bulletin of Mathematical Biophysics*, 5(4): 115-133, 1943).

# Neurons in Action



<https://news.harvard.edu/wp-content/uploads/2018/02/imaging-neuronal-activity-video-eurekalert-science-news.mp4>

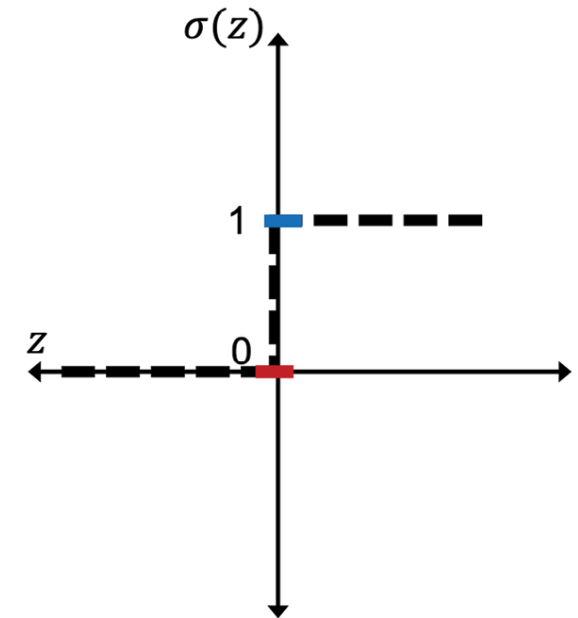
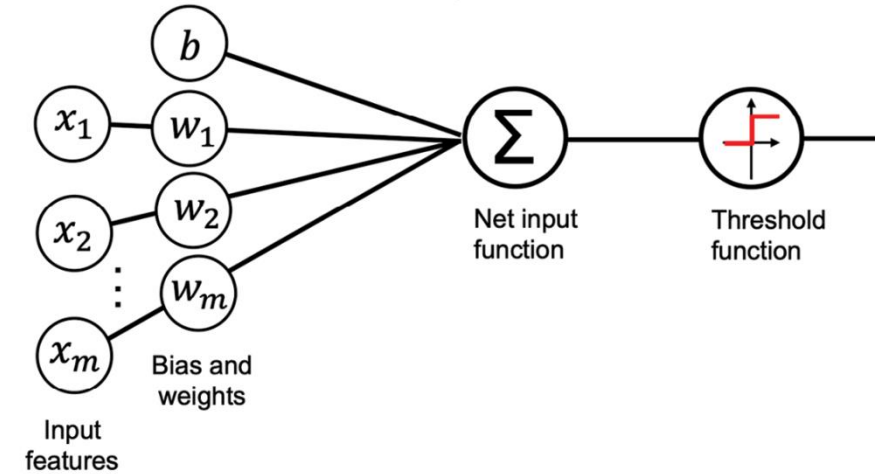
# Building Linear Neuron

Input:  $\mathbf{x} = \begin{bmatrix} x_1 \\ \vdots \\ x_m \end{bmatrix}$

Weights:  $\mathbf{w} = \begin{bmatrix} w_1 \\ \vdots \\ w_m \end{bmatrix}$

Linear transform: 
$$z = w_1x_1 + \dots + w_mx_m + b = \mathbf{w}^T\mathbf{x} + b$$

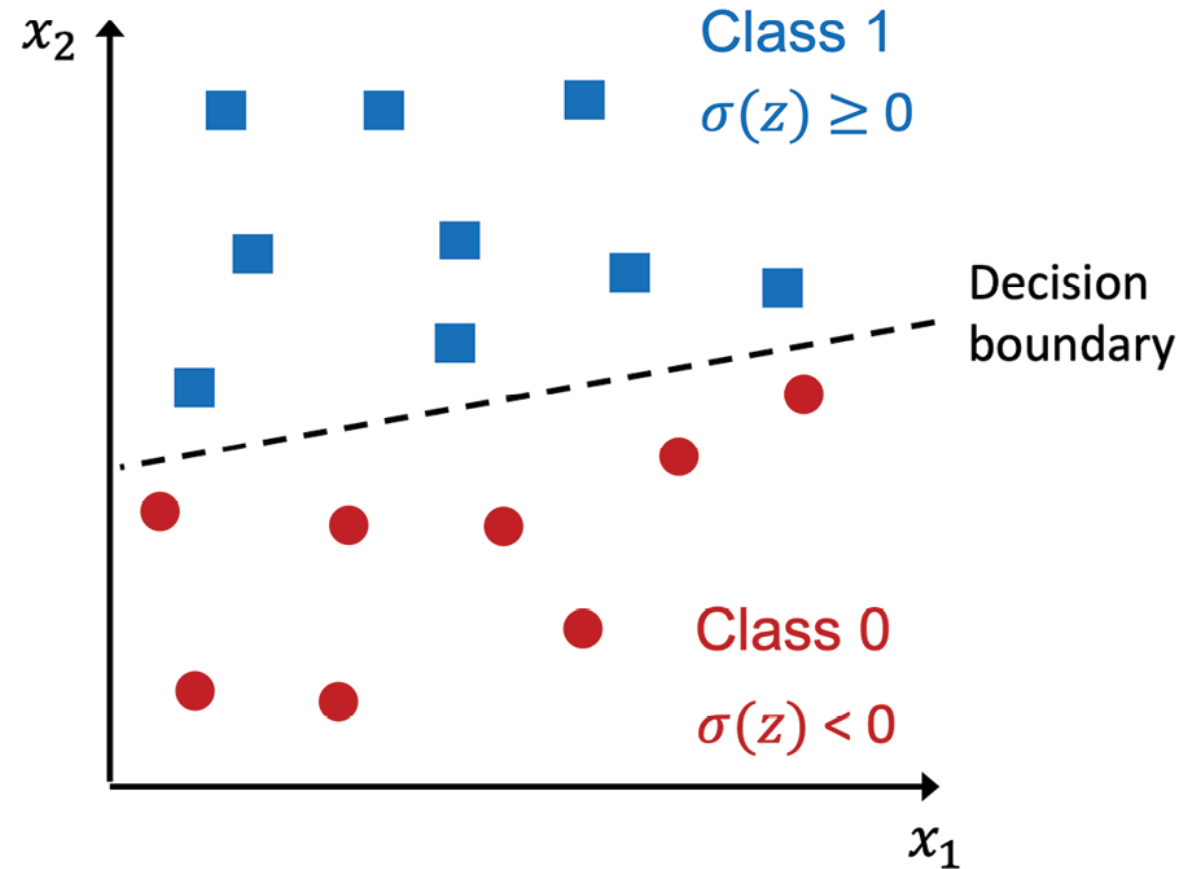
Output: 
$$\sigma(z) = \begin{cases} 1 & \text{if } z \geq 0 \\ 0 & \text{otherwise} \end{cases}$$



where  $z = \mathbf{w}^T\mathbf{x} + b$



# Linear Neuron in 2D



Linear transform:

$$z = w_1 x_1 + w_2 x_2 + b$$

Line:

$$x_2 = -w_1/w_2 x_1 - b/w_2$$

From S. Raschka, Machine Learning with PyTorch and Scikit-Learn

# Training Linear Neuron

- Initialize the weights and bias unit to 0 or small random numbers
- For each training example,  $\mathbf{x}(\mathbf{i})$ :
- Compute the output value,  $\mathbf{y}(\mathbf{i}) = \mathbf{w}^T \mathbf{x}(\mathbf{i}) + \mathbf{b}$
- Update the weights and bias unit:  $w_j := w_j + \Delta w_j$  and  $b := b + \Delta b$
- Where  $\Delta w_j = \eta(y^{(i)} - \hat{y}^{(i)})x_j^{(i)}$  and  $\Delta b = \eta(y^{(i)} - \hat{y}^{(i)})$

Each weight,  $w_j$ , corresponds to a feature,  $x_j$ , in the dataset,

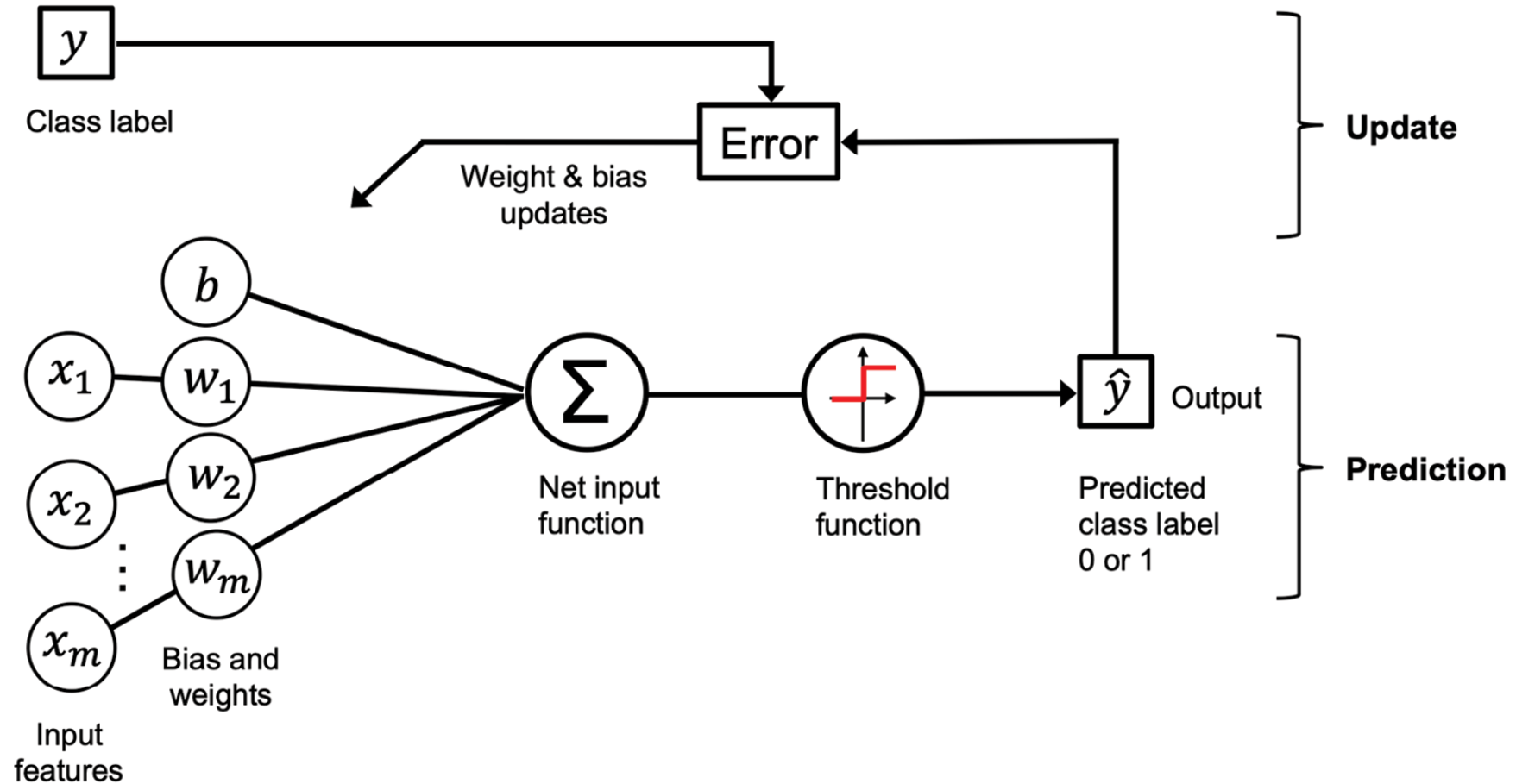
$\eta$  is the **learning rate** (typically a constant between 0.0 and 1.0),

$y^{(i)}$  is the **true class label** of the  $i$ th training example,

$\hat{y}^{(i)}$  is the **predicted class label**



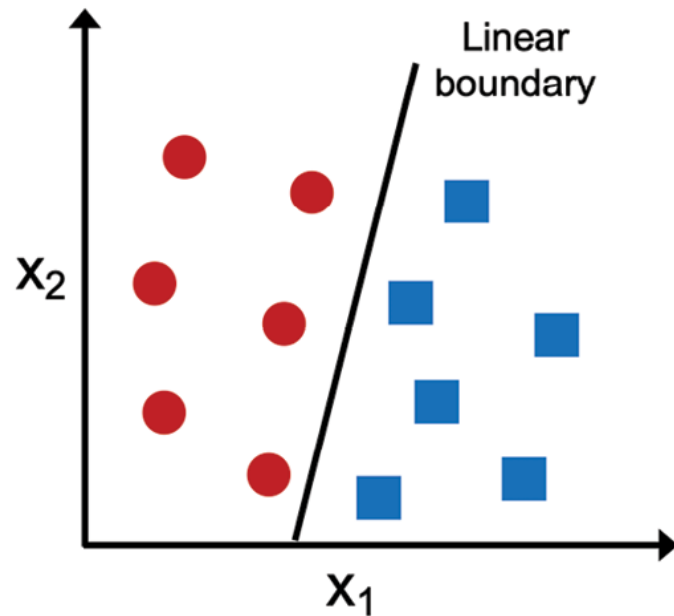
# Training Linear Neuron



# What problems can perceptron solve?

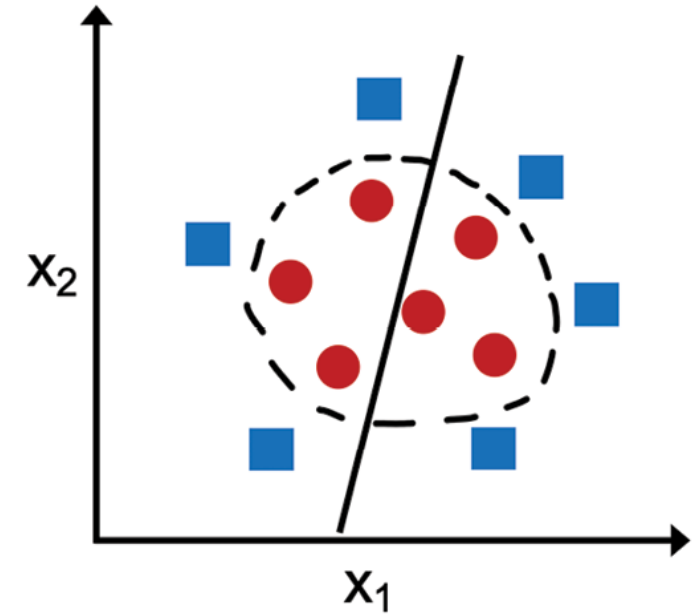
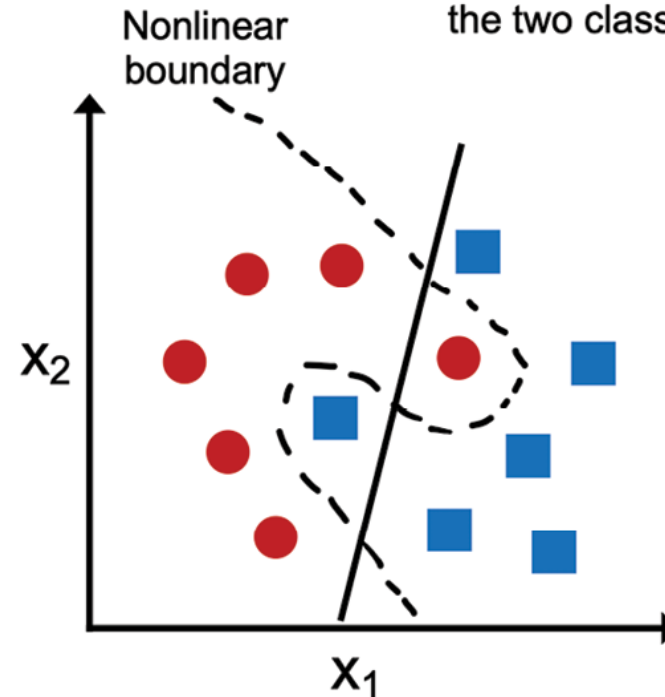
## Linearly separable

A linear decision boundary that separates the two classes exists

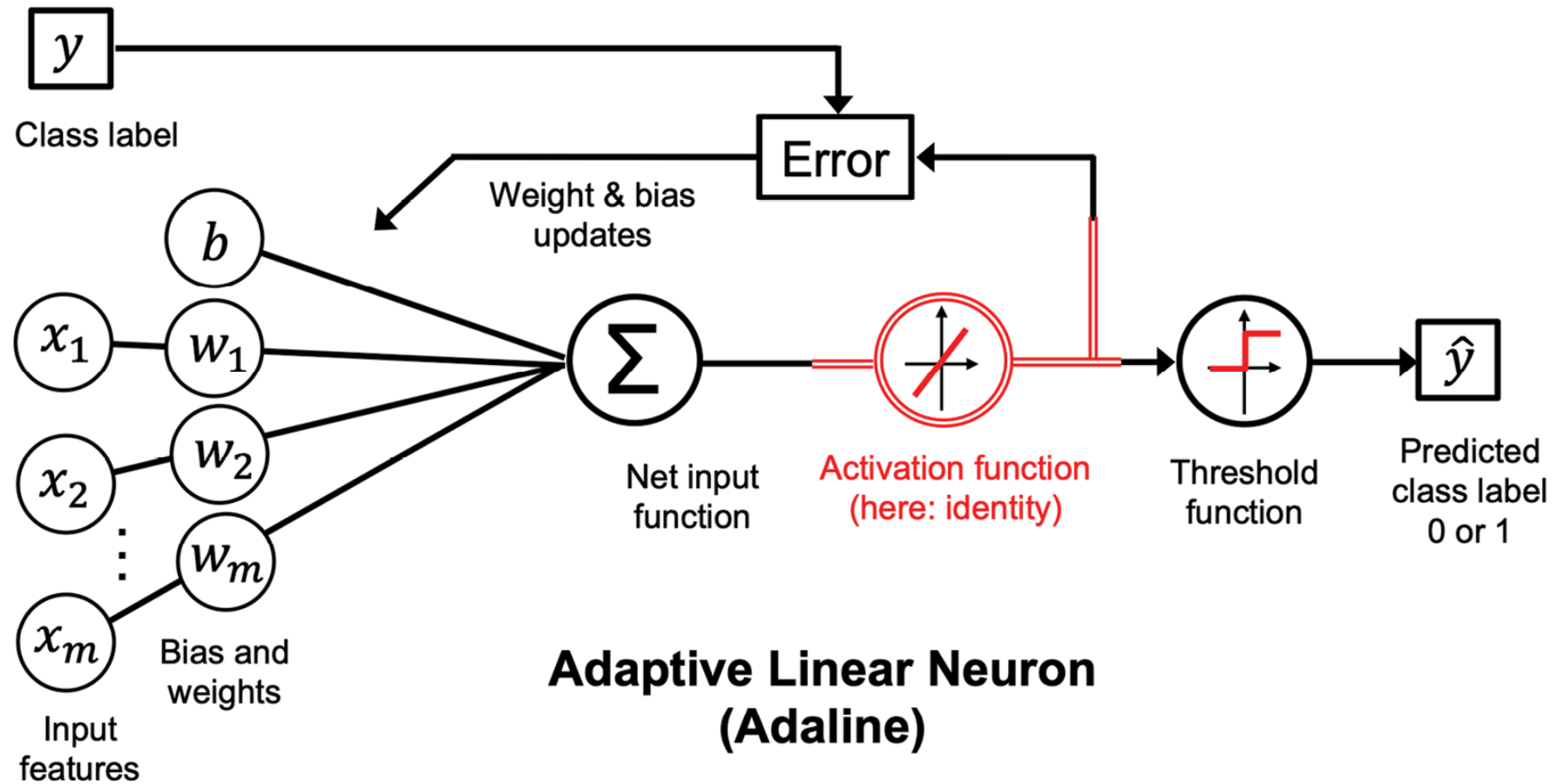


## Not linearly separable

No linear decision boundary that separates the two classes perfectly exists



# Adaline



# Adaline training

**Loss function:**

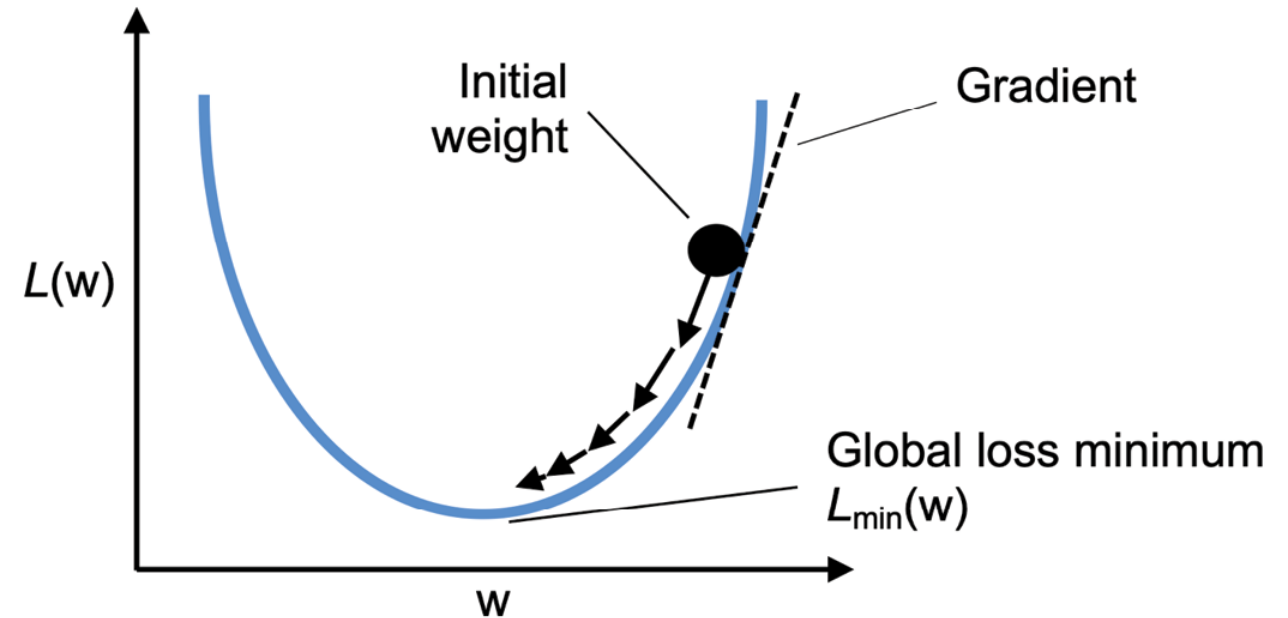
$$L(\mathbf{w}, b) = \frac{1}{n} \sum_{i=1}^n \left( y^{(i)} - \sigma(z^{(i)}) \right)^2$$

**Weights update:**

$$\mathbf{w} := \mathbf{w} + \Delta \mathbf{w}, \quad b := b + \Delta b$$

**Learning rule:**

$$\Delta \mathbf{w} = -\eta \nabla_{\mathbf{w}} L(\mathbf{w}, b), \quad \Delta b = -\eta \nabla_b L(\mathbf{w}, b)$$

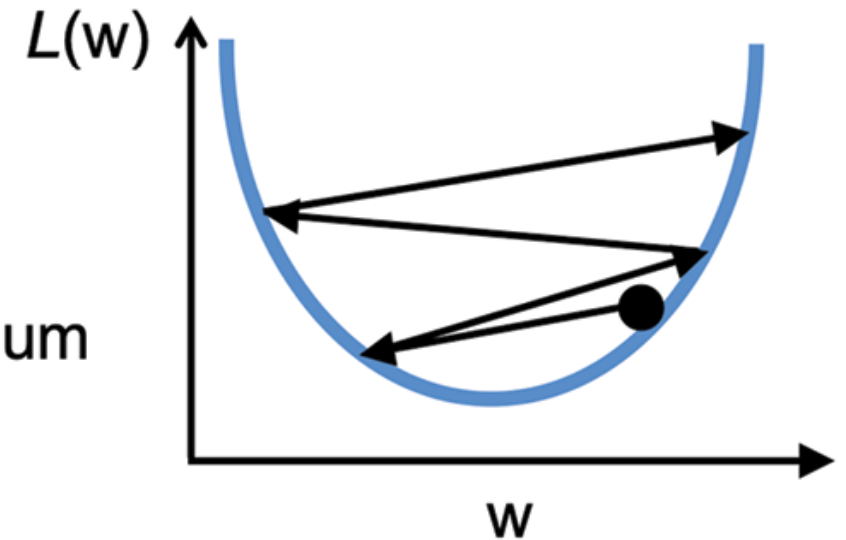
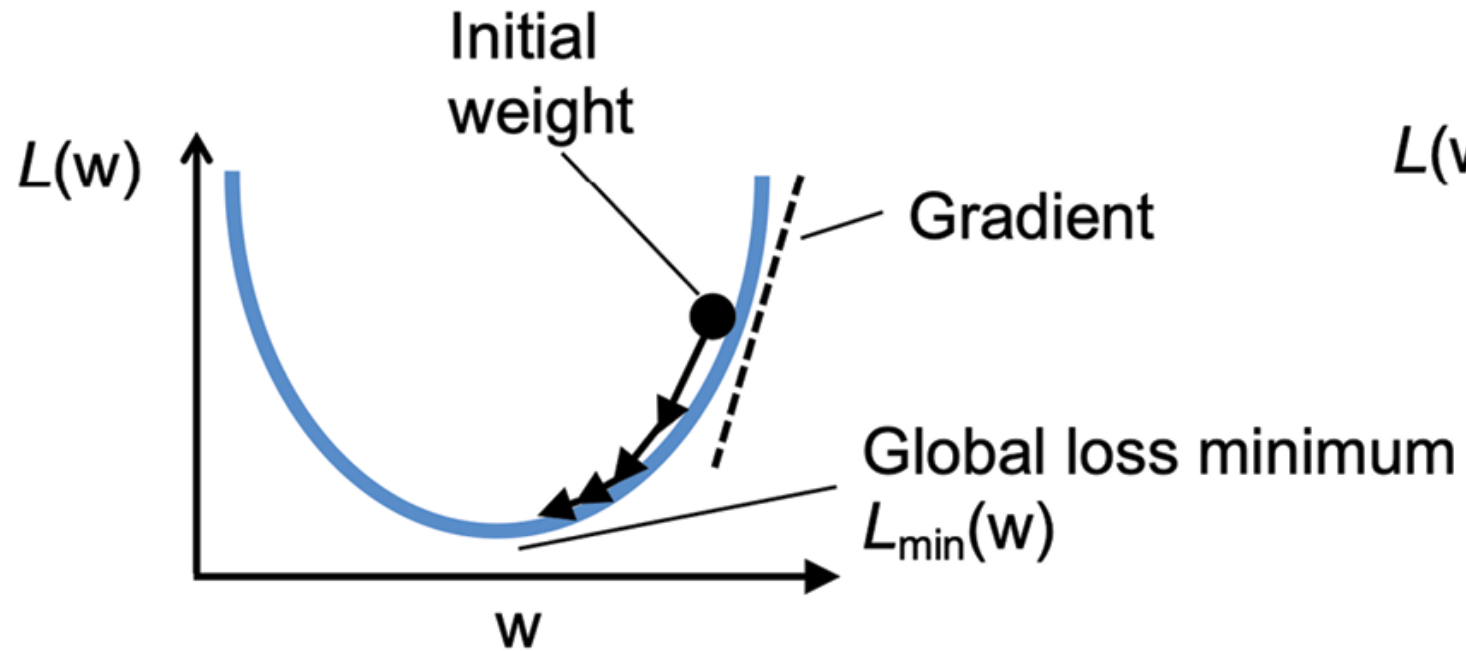


# How does it look like?

$$\begin{aligned}\frac{\partial L}{\partial w_j} &= \frac{\partial}{\partial w_j} \frac{1}{n} \sum_i \left( y^{(i)} - \sigma(z^{(i)}) \right)^2 = \frac{1}{n} \frac{\partial}{\partial w_j} \sum_i \left( y^{(i)} - \sigma(z^{(i)}) \right)^2 \\ &= \frac{2}{n} \sum_i \left( y^{(i)} - \sigma(z^{(i)}) \right) \frac{\partial}{\partial w_j} \left( y^{(i)} - \sigma(z^{(i)}) \right) \\ &= \frac{2}{n} \sum_i \left( y^{(i)} - \sigma(z^{(i)}) \right) \frac{\partial}{\partial w_j} \left( y^{(i)} - \sum_j (w_j x_j^{(i)} + b) \right) \\ &= \frac{2}{n} \sum_i \left( y^{(i)} - \sigma(z^{(i)}) \right) (-x_j^{(i)}) = -\frac{2}{n} \sum_i \left( y^{(i)} - \sigma(z^{(i)}) \right) x_j^{(i)}\end{aligned}$$

- Adaline learning rule looks identical to the perceptron rule
- However,  $\sigma(z^{(i)})$  where  $z^{(i)} = \mathbf{w}^T \mathbf{x}^{(i)} + b$  is a real number and not an integer class label.
- Also, weight update is calculated based on all examples in the training dataset (instead of updating the parameters incrementally after each training example): **batch gradient descent**.

# Role of learning rate



# Why do we standardize input features?

