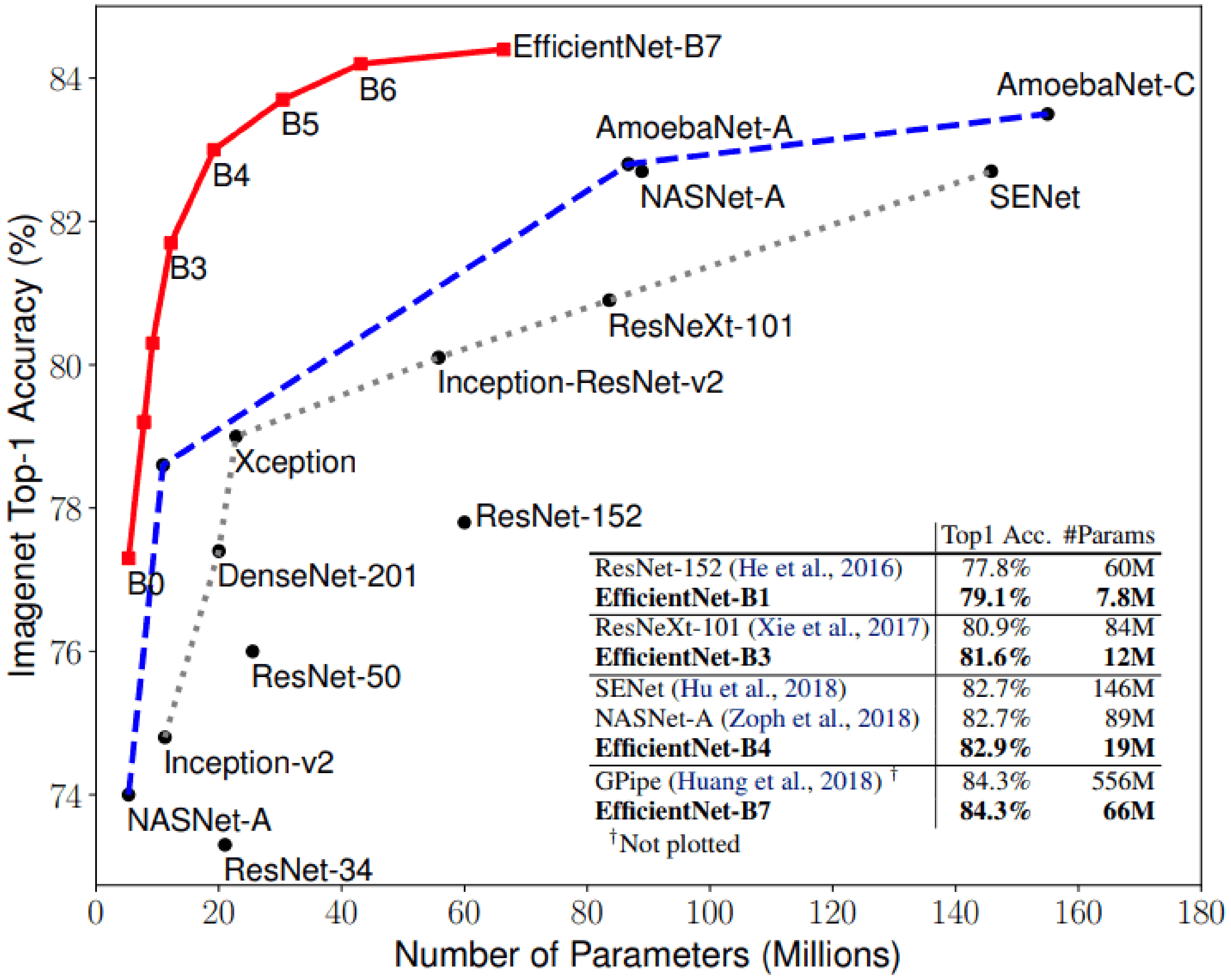
Lecture 06: Simple Perceptron, Adaline, and Logistic Regression

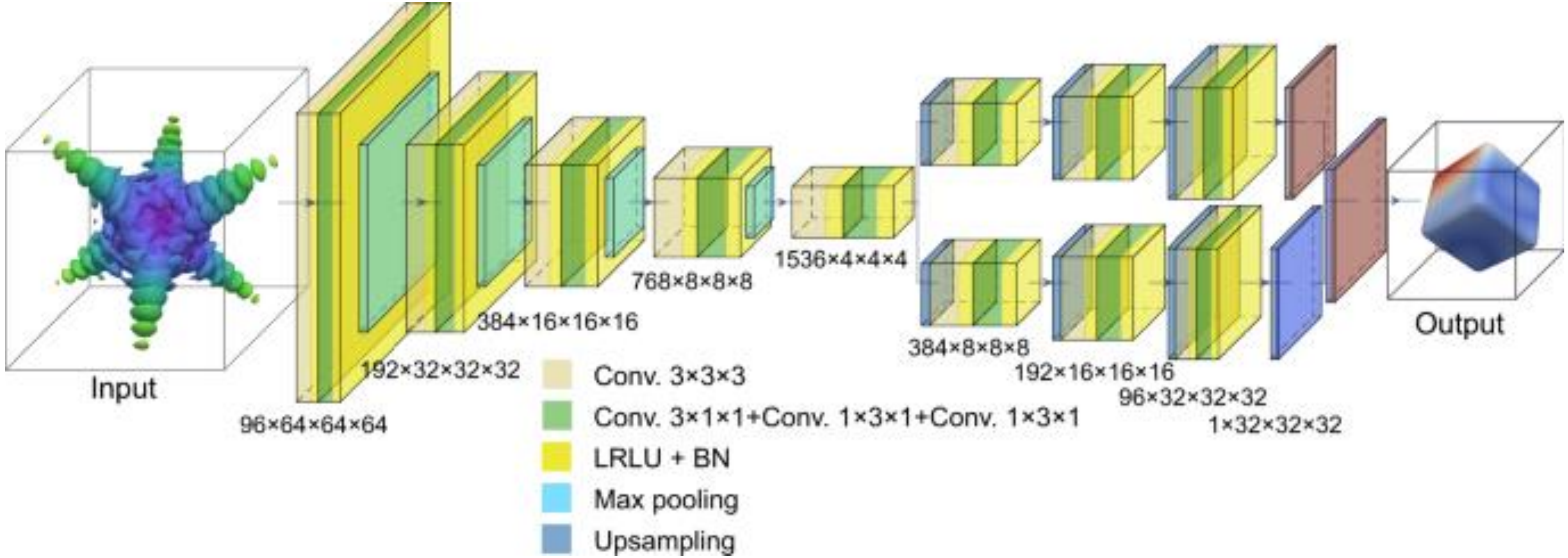
Instructor: Sergei V. Kalinin



	Top1 Acc.	#Params
ResNet-152 (He et al., 2016)	77.8%	60M
EfficientNet-B1	**79.1%**	**7.8M**
ResNeXt-101 (Xie et al., 2017)	80.9%	84M
EfficientNet-B3	**81.6%**	**12M**
SENet (Hu et al., 2018)	82.7%	146M
NASNet-A (Zoph et al., 2018)	82.7%	89M
EfficientNet-B4	**82.9%**	**19M**
GPipe (Huang et al., 2018) †	84.3%	556M
EfficientNet-B7	**84.3%**	**66M**

†Not plotted

https://theaisummer.com/cnn-architectures/

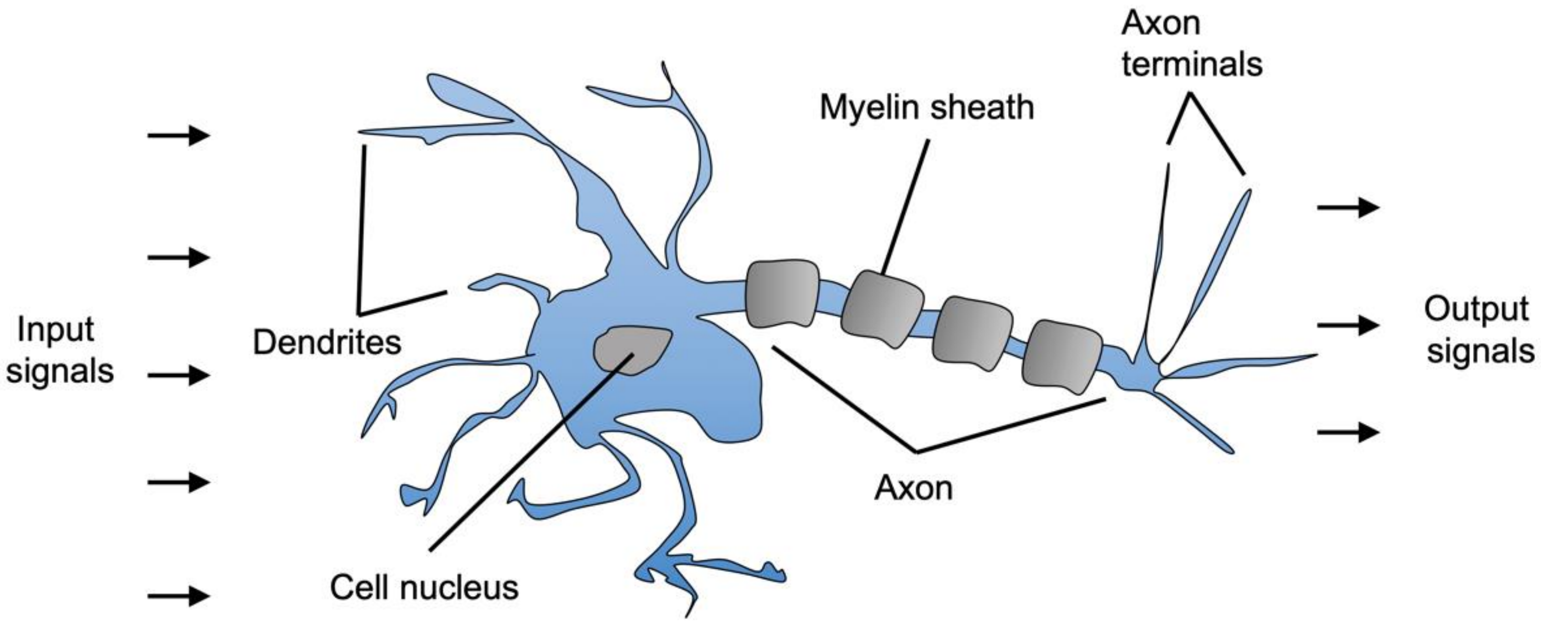


Longlong Wu, Shinjae Yoo, Ana F. Suzana, Tadesse A. Assefa, Jiecheng Diao, Ross J. Harder, Wonsuk Cha & Ian K. Robinson, *Three-dimensional coherent X-ray diffraction imaging via deep convolutional neural networks*

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

Brain structure and McCulloch-Pitts neuron

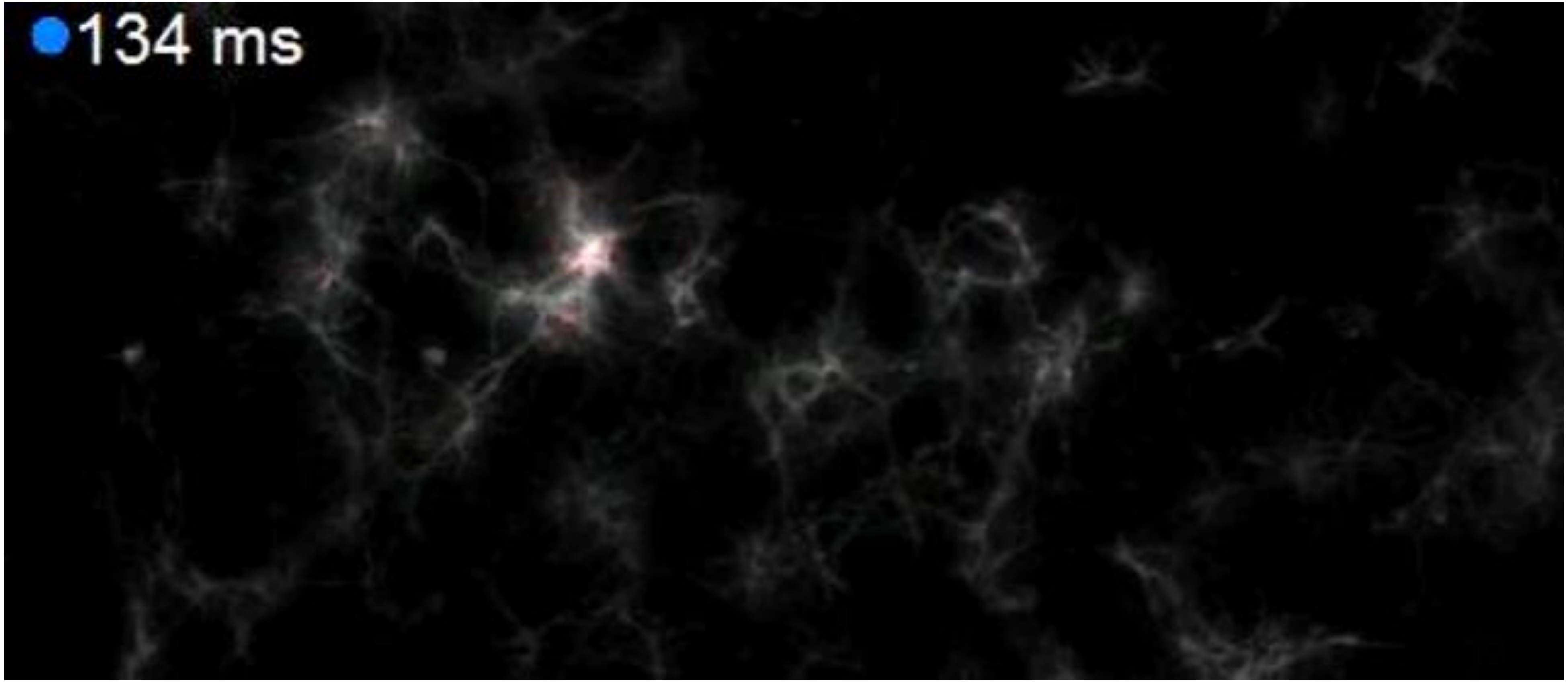
https://www.kenhub.com/en/library/anatomy/histology-of-neurons



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).

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

Neurons in Action



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

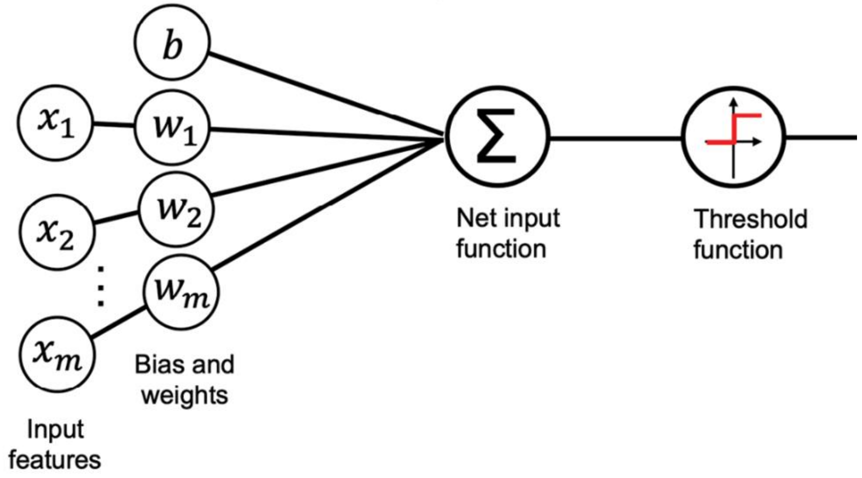
Building Linear Neuron

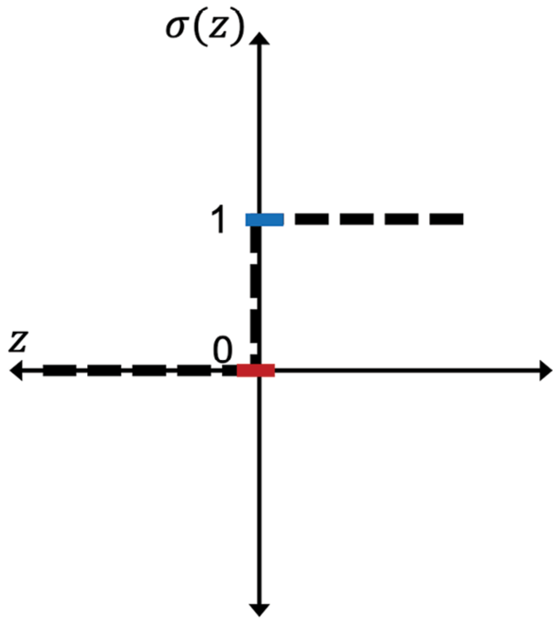
Input: $$\boldsymbol{x} = \begin{bmatrix} x_1 \\ \vdots \\ x_m \end{bmatrix}$$

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

Linear transform: $$z = \boldsymbol{w_1 x_1} + \ldots + \boldsymbol{w_m x_m} + \boldsymbol{b} = \\ = \boldsymbol{w^T x} + \boldsymbol{b}$$

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

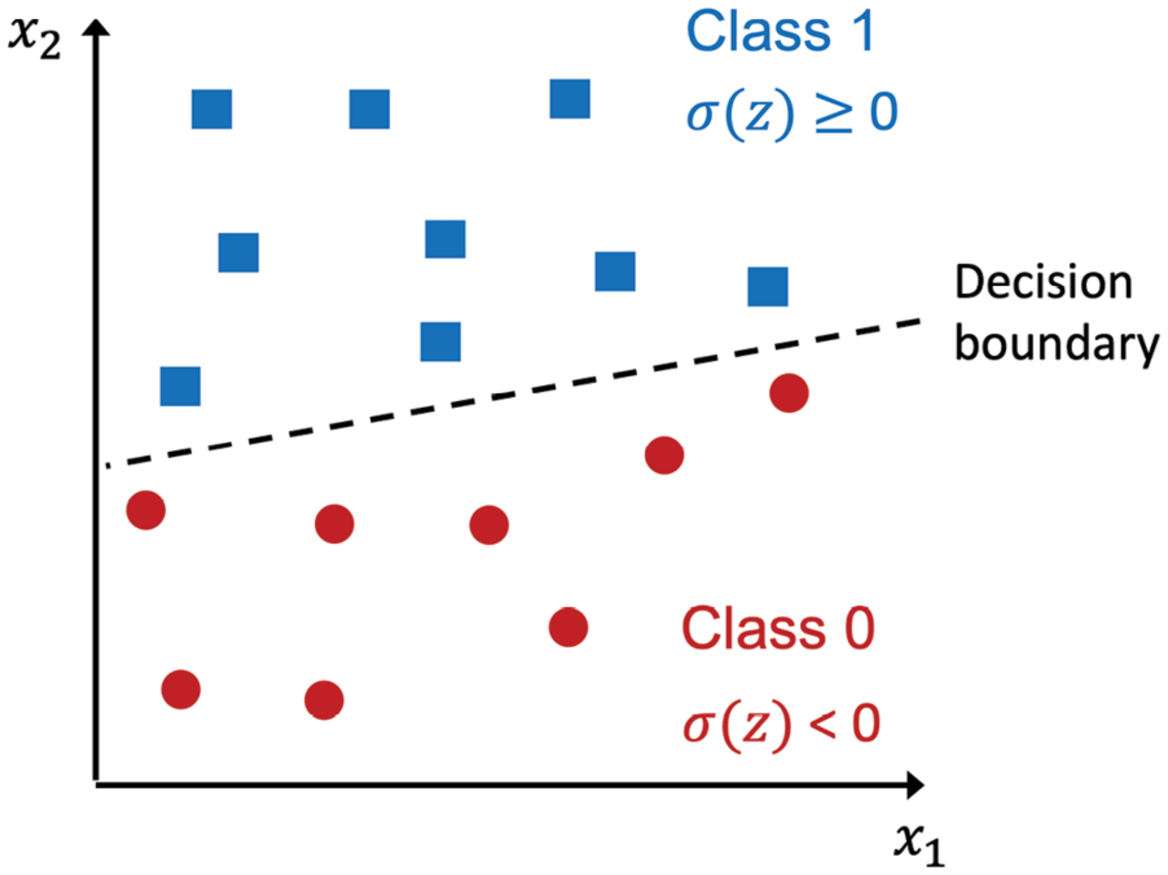




where $z = \boldsymbol{w}^T\boldsymbol{x} + b$

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

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, ***x(i)***:
- Compute the output value, $\boldsymbol{y(i) = w^T x(i) + b}$
- Update the weights and bias unit: $w_j := w_j + \Delta w_j$ and $b := b + \Delta b$
- Where $\Delta w_j = \eta\left(y^{(i)} - \hat{y}^{(i)}\right)x_j^{(i)}$ and $\Delta b = \eta\left(y^{(i)} - \hat{y}^{(i)}\right)$

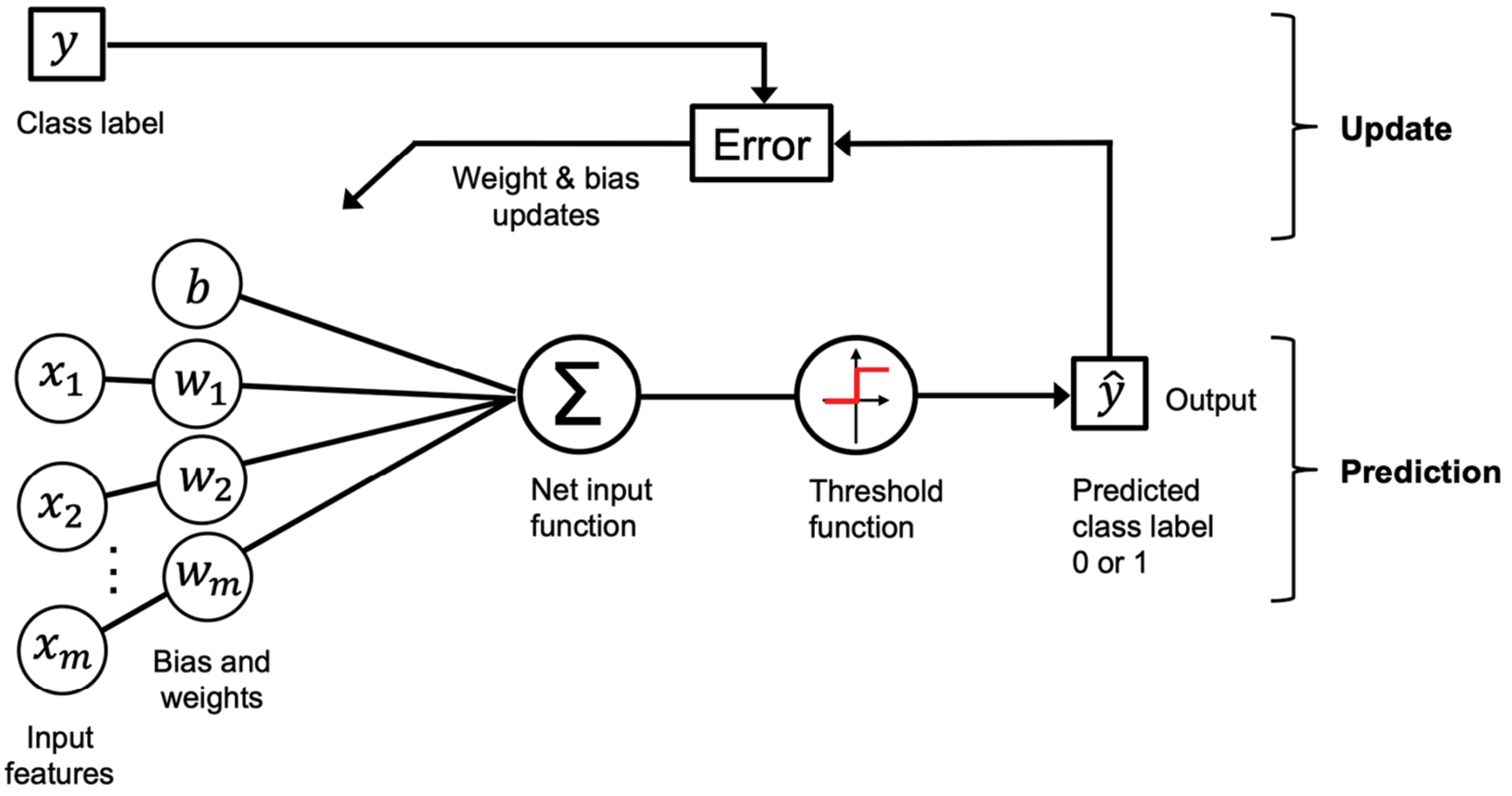
Each weight, w_j, corresponds to a feature, x_j, in the dataset,

η 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,

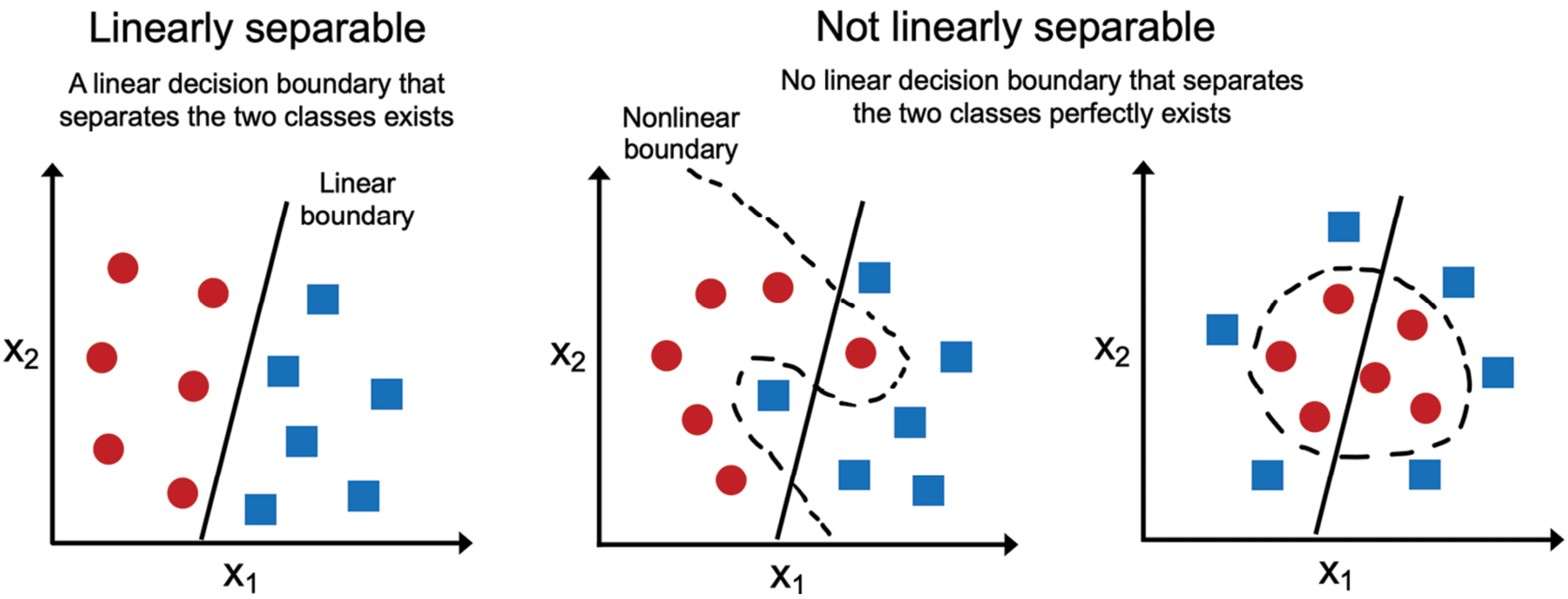
$\hat{y}^{(i)}$ is the **predicted class label**

Training Linear Neuron



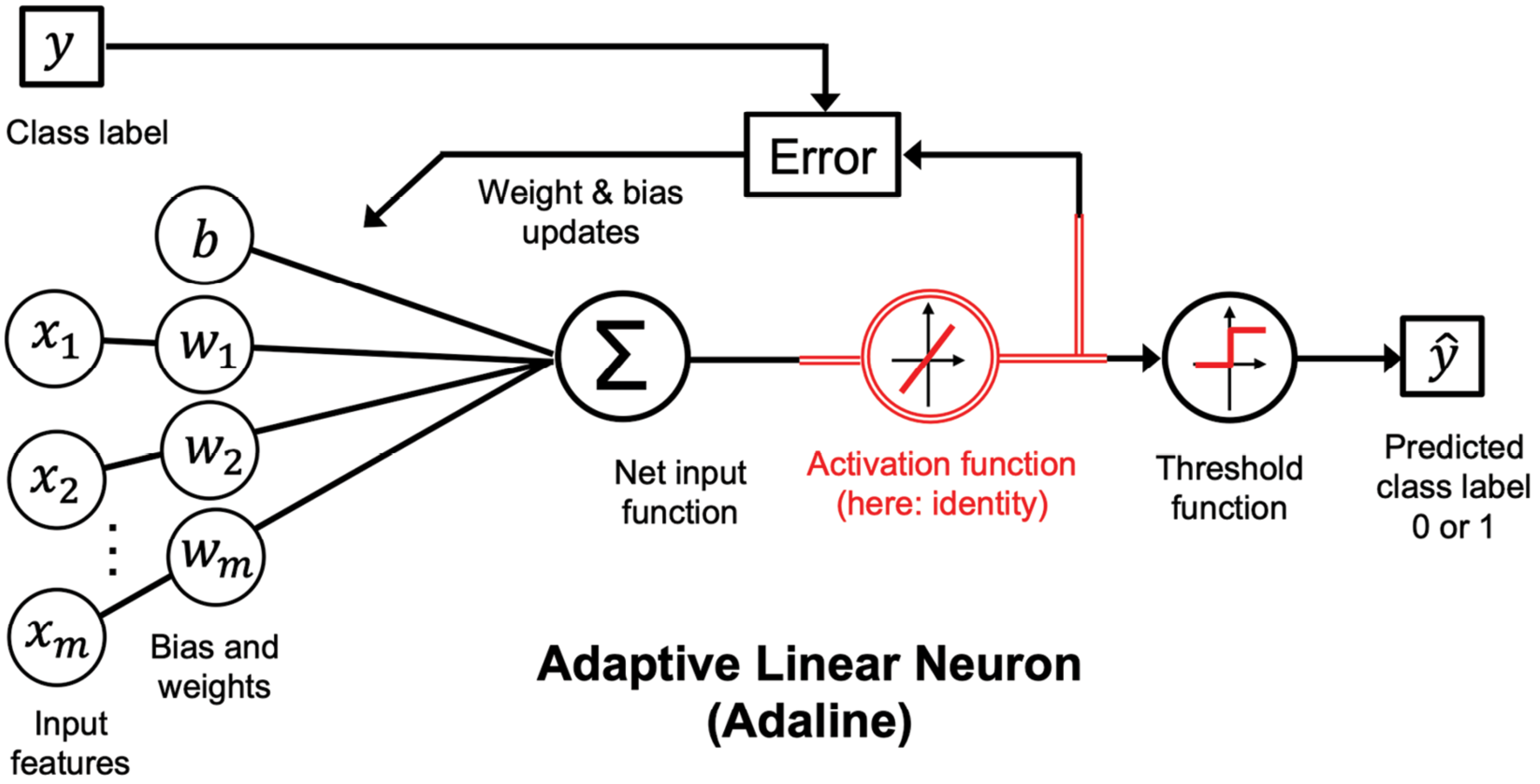
From S. Raschka, Machine Learning with PyTorch and Scikit-Learn

What problems can perceptron solve?



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

Adaline

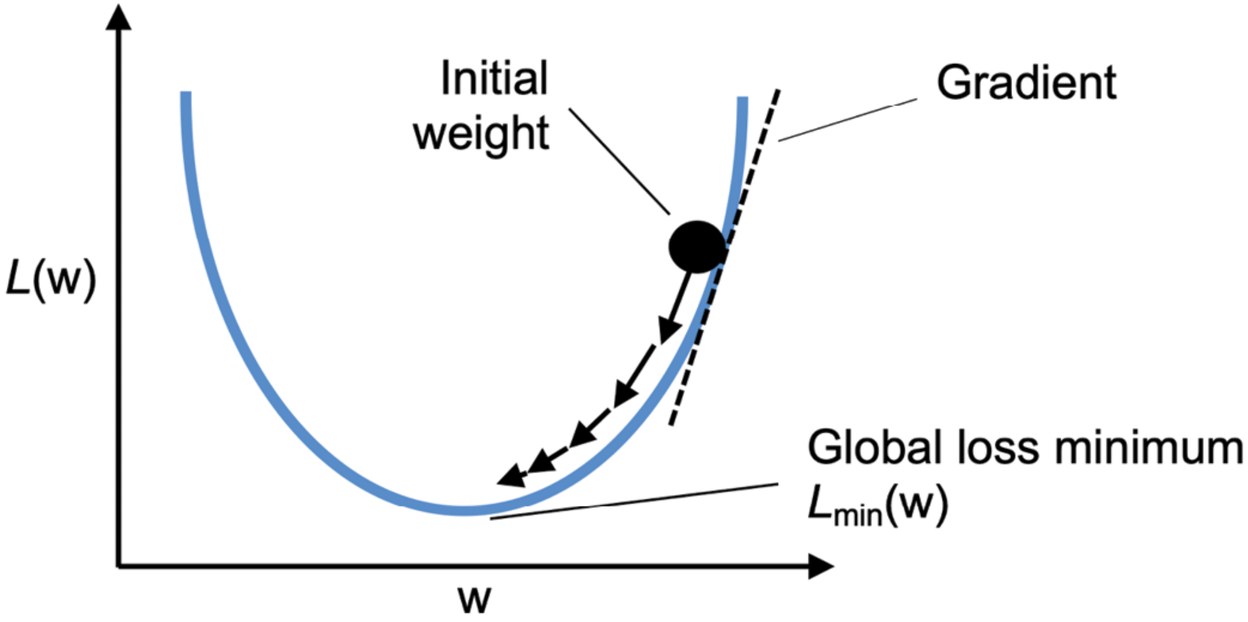


From S. Raschka, Machine Learning with PyTorch and Scikit-Learn

Adaline training

Loss function:

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



Weights update:

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

Learning rule:

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

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

How does it look like?

$$\frac{\partial L}{\partial w_j} = \frac{\partial}{\partial w_j}\frac{1}{n}\sum_i \left(y^{(i)} - \sigma\left(z^{(i)}\right)\right)^2 = \frac{1}{n}\frac{\partial}{\partial w_j}\sum_i \left(y^{(i)} - \sigma\left(z^{(i)}\right)\right)^2$$

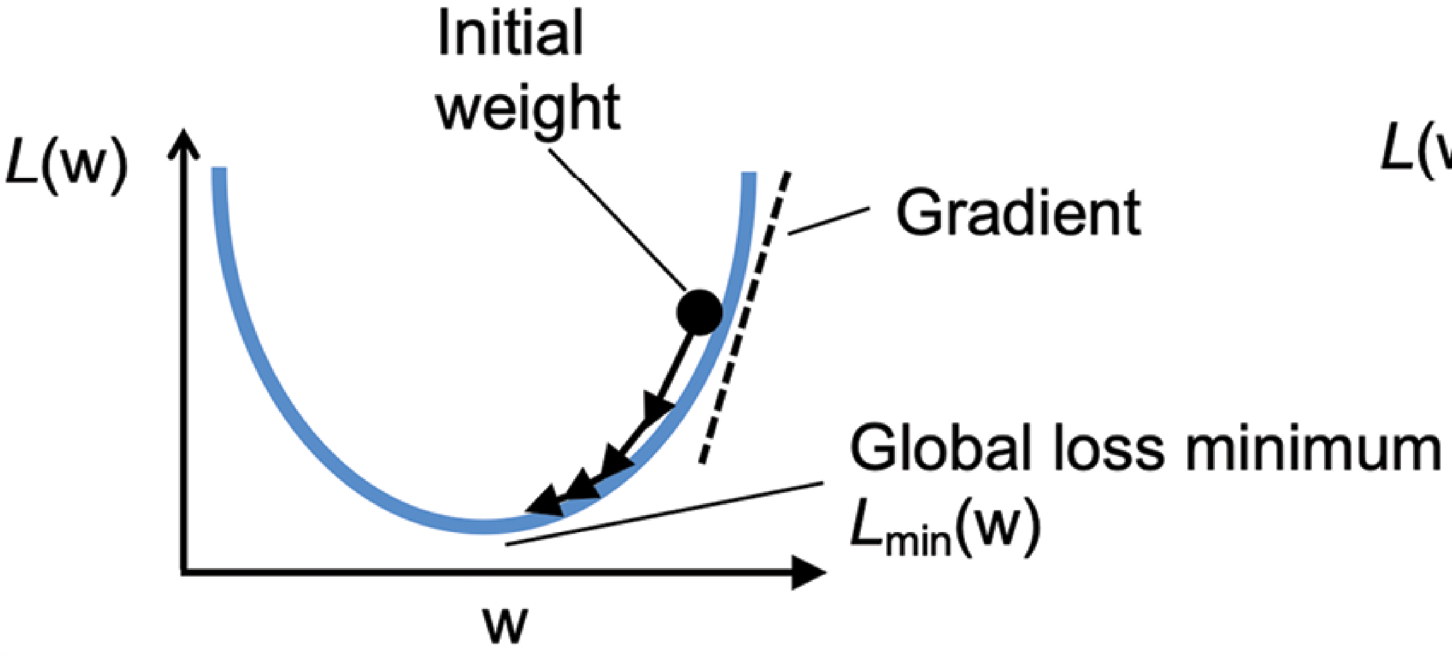
$$= \frac{2}{n}\sum_i \left(y^{(i)} - \sigma\left(z^{(i)}\right)\right)\frac{\partial}{\partial w_j}\left(y^{(i)} - \sigma\left(z^{(i)}\right)\right)$$

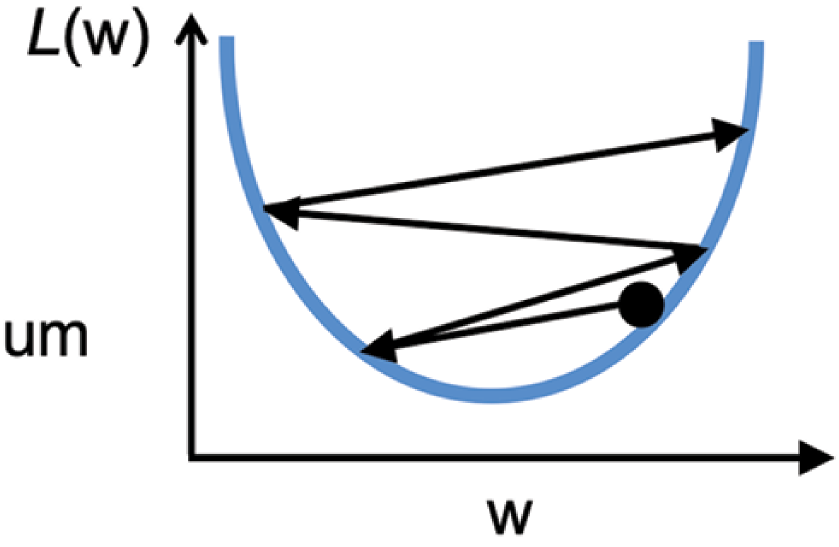
$$= \frac{2}{n}\sum_i \left(y^{(i)} - \sigma\left(z^{(i)}\right)\right)\frac{\partial}{\partial w_j}\left(y^{(i)} - \sum_j\left(w_j x_j^{(i)} + b\right)\right)$$

$$= \frac{2}{n}\sum_i \left(y^{(i)} - \sigma\left(z^{(i)}\right)\right)\left(-x_j^{(i)}\right) = -\frac{2}{n}\sum_i \left(y^{(i)} - \sigma\left(z^{(i)}\right)\right)x_j^{(i)}$$

- Adaline learning rule looks identical to the perceptron rule
- However, $\sigma\left(z^{(i)}\right)$ where $z^{(i)} = \boldsymbol{w}^T\boldsymbol{x}^{(i)} + b$ 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**.

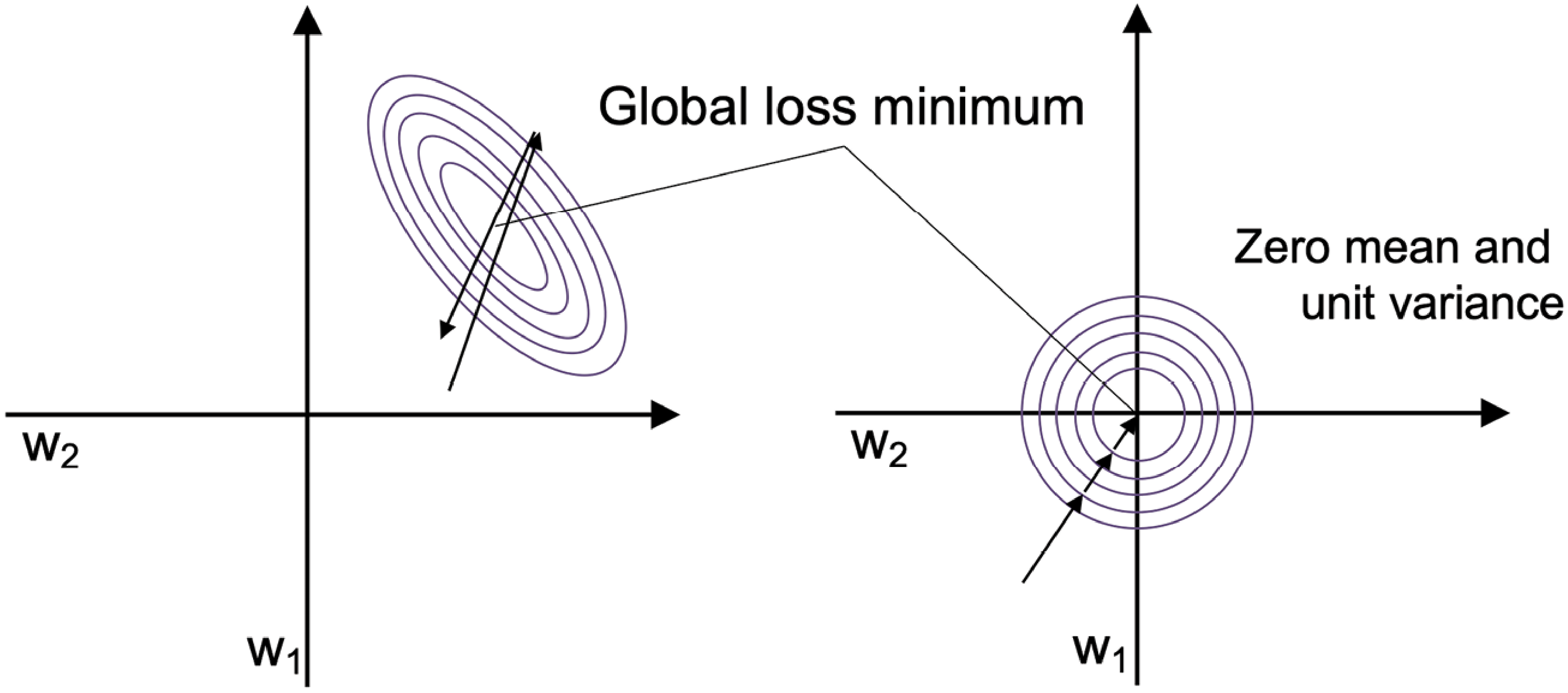
Role of learning rate





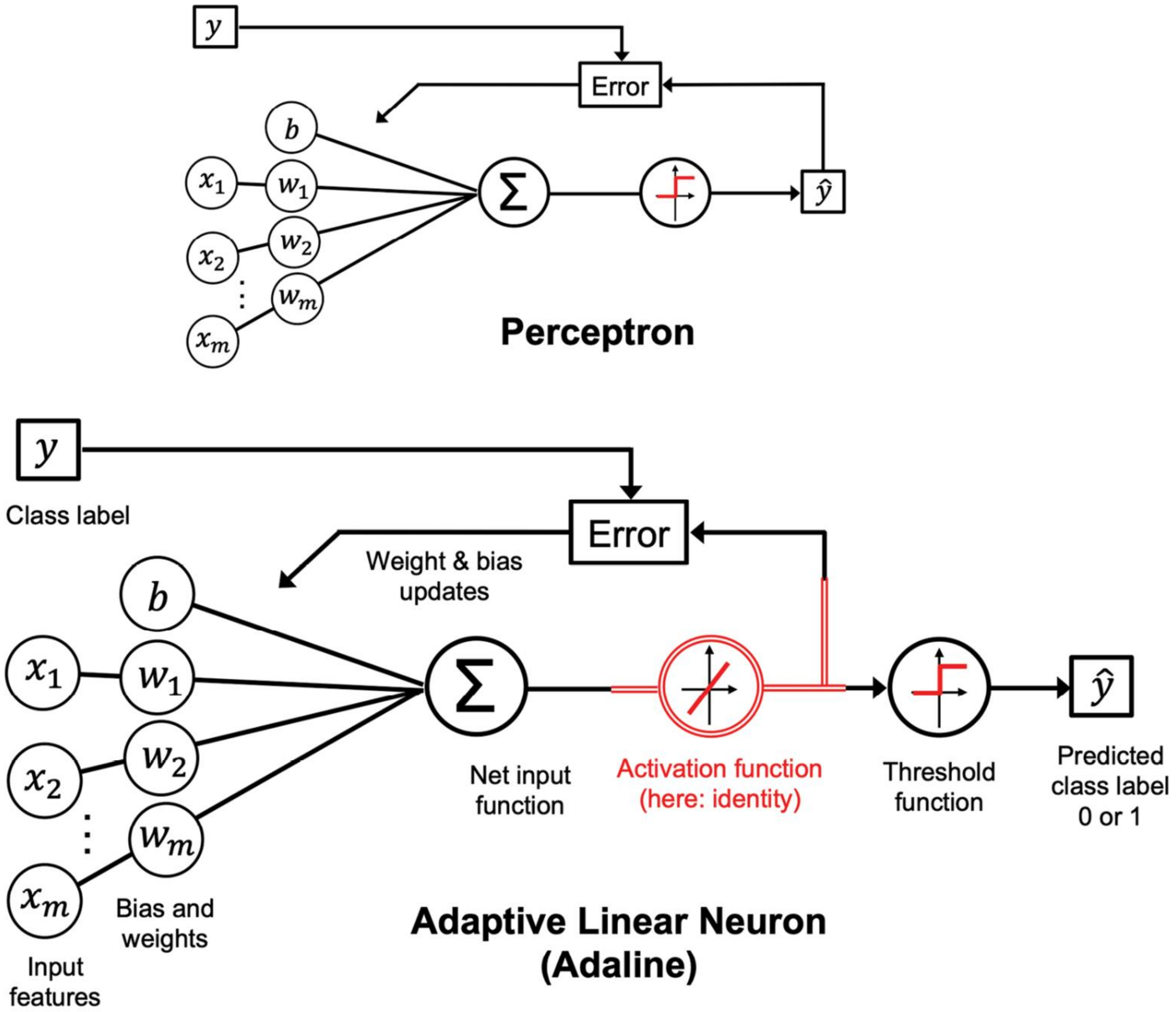
From S. Raschka, Machine Learning with PyTorch and Scikit-Learn

Why do we standardize input features?



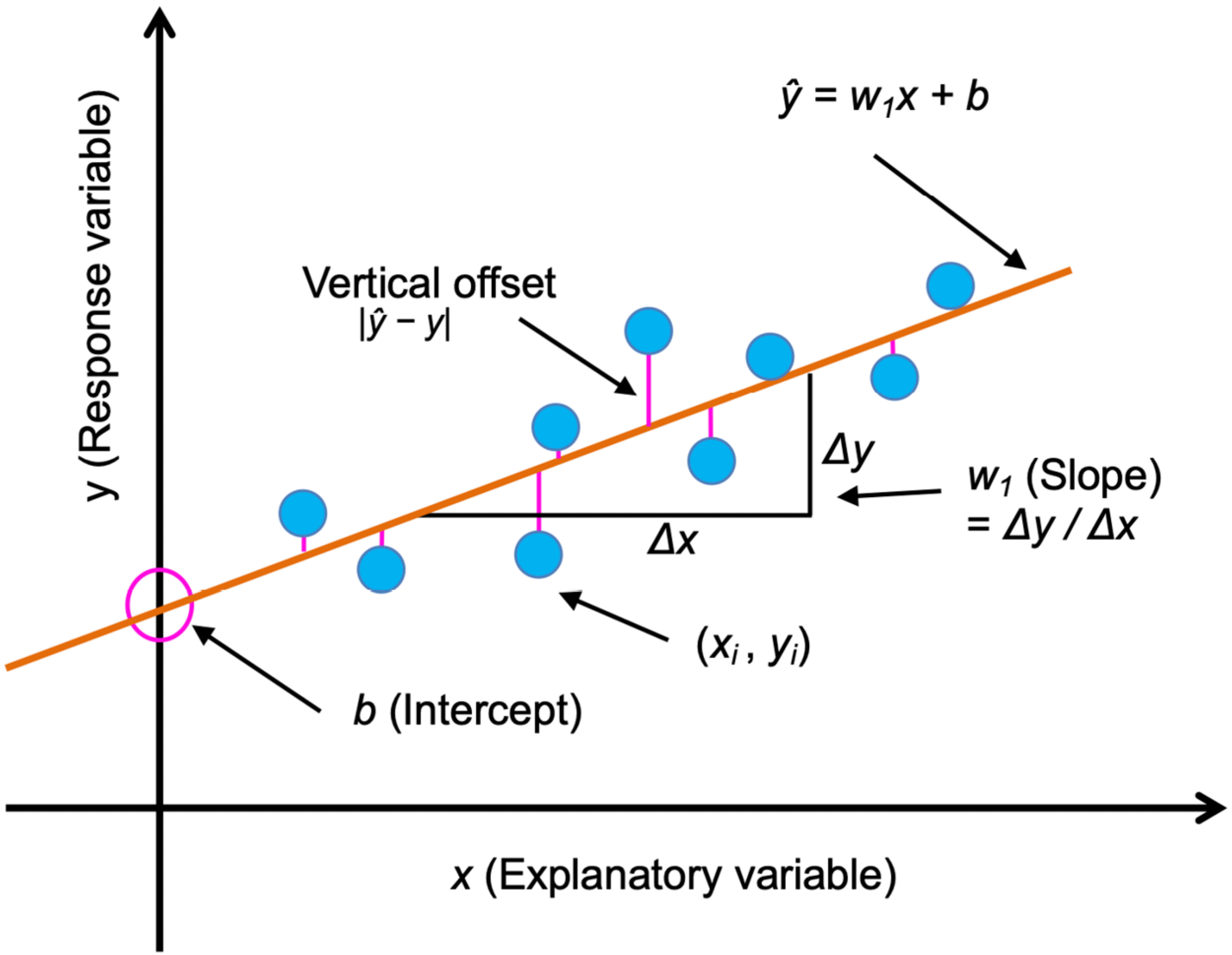
From S. Raschka, Machine Learning with PyTorch and Scikit-Learn

Perceptron and Adaline



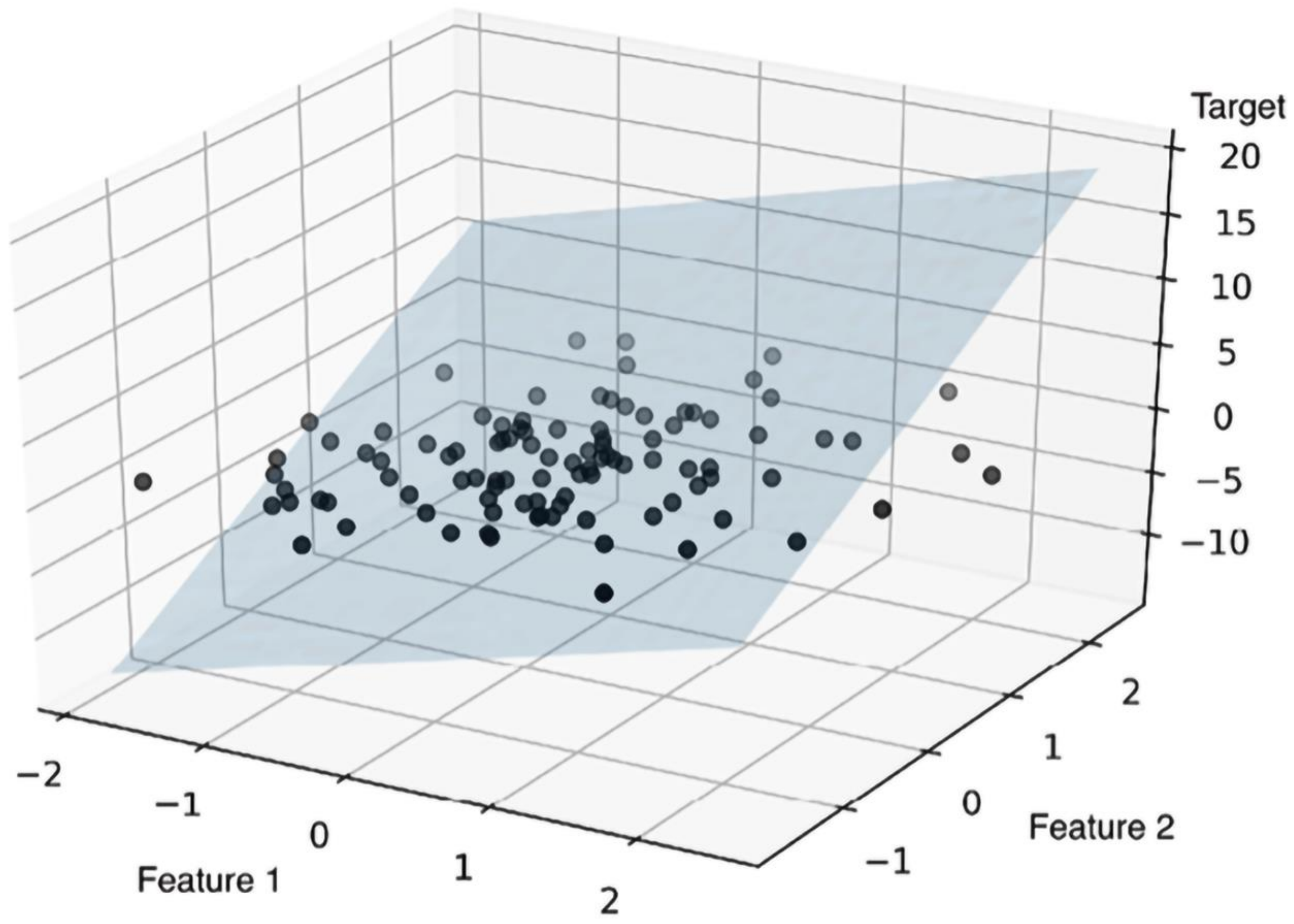
From S. Raschka, Machine Learning with PyTorch and Scikit-Learn

Linear Regression in 1D



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

Linear Regression in 2D



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

RANSAC: Random Sample Consensus

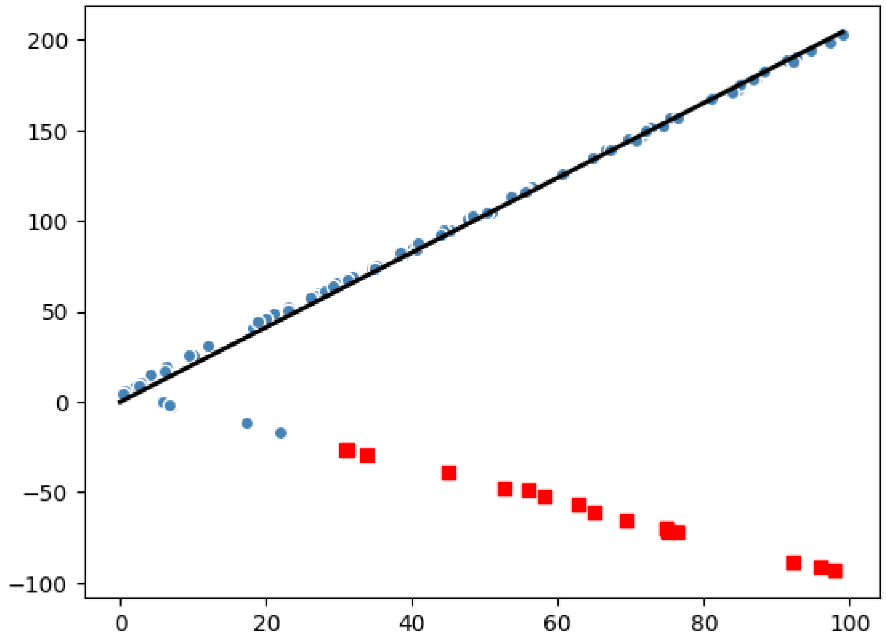
1. Select a random number of examples to be inliers and fit the model.

2. Test all other data points against the fitted model and add those points that fall within a user-given tolerance to the inliers.

3. Refit the model using all inliers.

4. Estimate the error of the fitted model versus the inliers.

5. Terminate the algorithm if the performance meets a certain user-defined threshold or if a fixed number of iterations was reached; go back to *step 1* otherwise.



Regularized linear regression

Linear regression can become nontrivial if **x** in y = lin(**x**) has high D

1. Ridge regression:

$$L(\boldsymbol{w})_{Ridge} = \sum_{i=1}^{n} \left(y^{(i)} - \hat{y}^{(i)}\right)^2 + \lambda ||\boldsymbol{w}||_2^2$$

2. Least absolute shrinkage and selection operator (LASSO):

$$L(\boldsymbol{w})_{Lasso} = \sum_{i=1}^{n} \left(y^{(i)} - \hat{y}^{(i)}\right)^2 + \lambda ||\boldsymbol{w}||_1$$

3. Elastic net

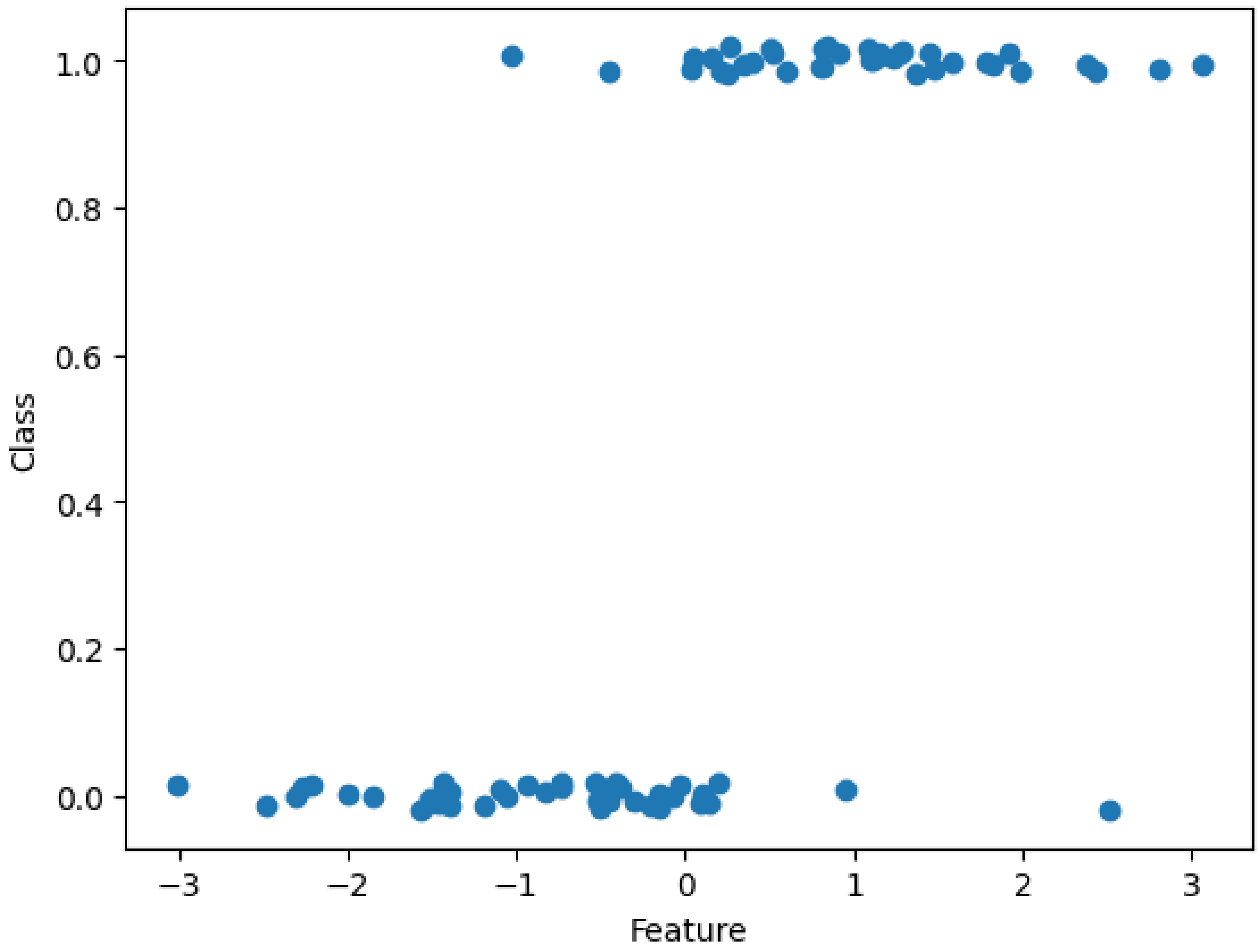
$$L(\boldsymbol{w})_{Elastic\ Net} = \sum_{i=1}^{n} \left(y^{(i)} - \hat{y}^{(i)}\right)^2 + \lambda_2 ||\boldsymbol{w}||_2^2 + \lambda_1 ||\boldsymbol{w}||_1$$

Practically: need careful consideration of what x is. Dependent on physics, better approach can be DCNNs, causal methods, etc.

Logistic regression

https://www.weknowpets.com.au/blogs/news/are-guinea-pigs-the-right-pet-for-your-family

When do we need logistic regression?



Logistic regression

Probability of event: p

Odds: $\frac{p}{(1-p)}$

Logit: $\text{logit}(p) = \log \frac{p}{(1-p)}$

May the odds ever be in your favor!

Logistic model assumes that there is a linear relationship between the weighted inputs and the log-odds

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

Logistic regression

Logistic model:

$$\ln \frac{p}{1-p} = W^T x + b$$

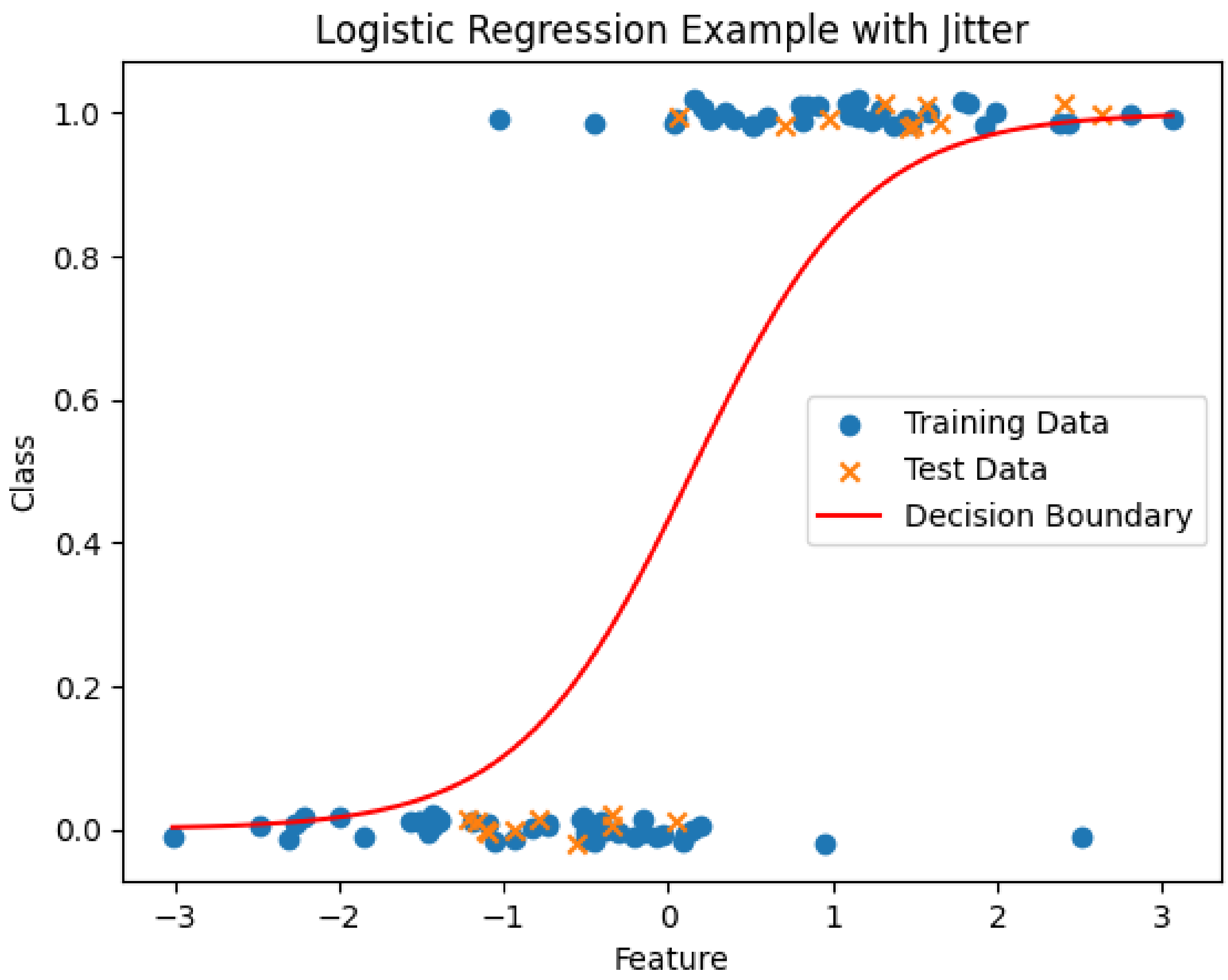
Logistic function:

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

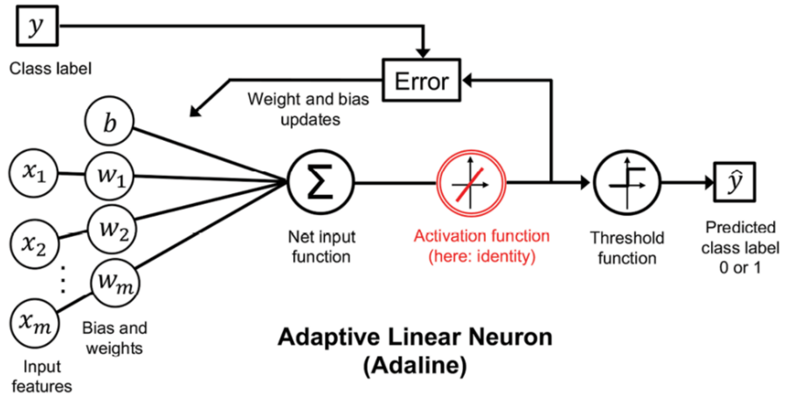
$$z = W^T x + b$$

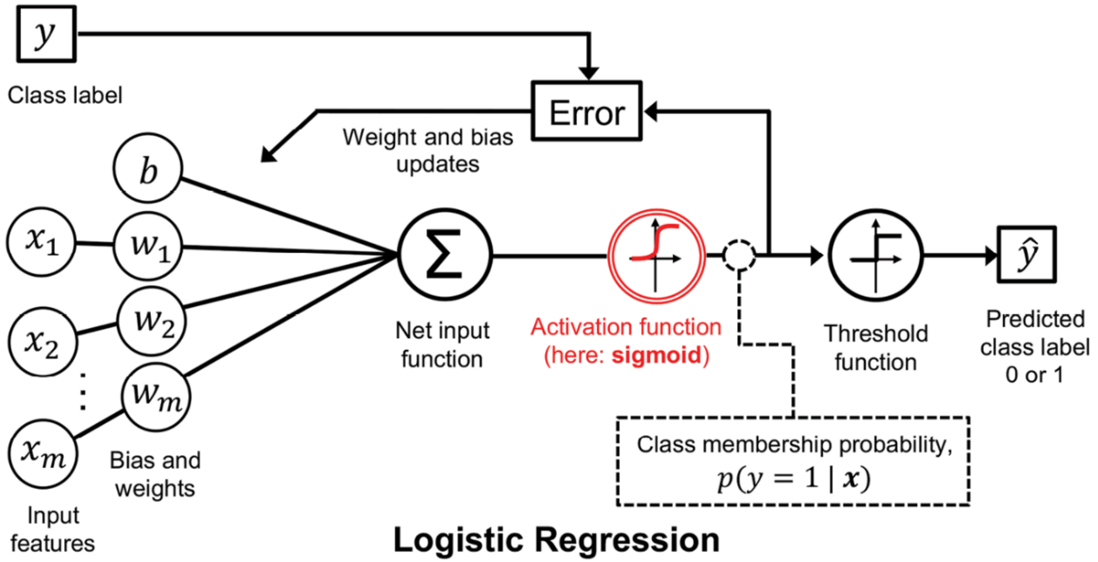
Goal of logistic regression: Predict the "true" proportion of success, p, at any value of the predictor.

Logistic regression prediction



Logistic regression vs. Adaline



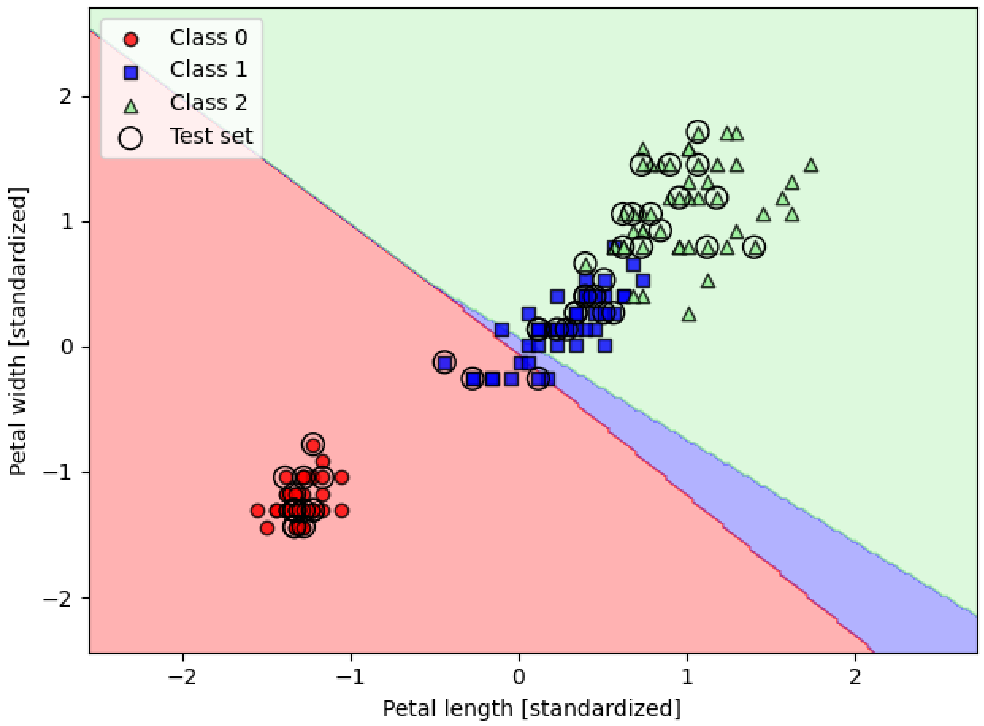


Similar to Adaline, but uses sigmoid as activation function

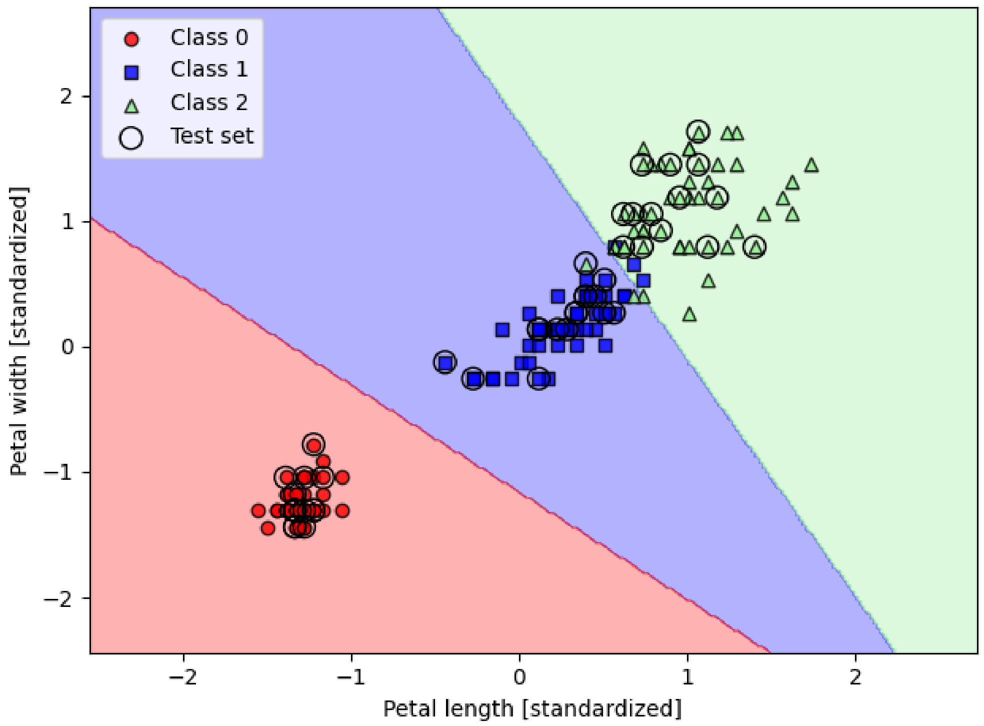
From S. Raschka, Machine Learning with PyTorch and Scikit-Learn

Regularization in logistic regression

C = 0.001



C = 100



1/C

$$L(\boldsymbol{w}, b) = \frac{1}{n} \sum_{i=1}^{n} \left[-y^{(i)} \log\left(\sigma\left(z^{(i)}\right)\right) - \left(1 - y^{(i)}\right) \log\left(1 - \sigma\left(z^{(i)}\right)\right) \right] + \frac{\lambda}{2n} \|\boldsymbol{w}\|^2$$