# Linear regression

We have seen how a linear regression model can be used to fit a dataset of inputs and targets $\mathcal{D} := (\mathbf{x}_i, y_i)_{i=1}^N$

$$f_\theta(\mathbf{x}) = \theta^T \phi(\mathbf{x})$$

The parameters $\theta$ are chosen to minimise the mean squared error loss

$$L_{MSE}(\theta) = \frac{1}{N} \sum_{i=1}^{N} (f_\theta(\mathbf{x}_i) - y_i)^2$$

It can be shown that the solution to this minimisation problem can be found explicitly, using the **normal equation**:

$$\hat{\theta} = (\Phi_{\mathbf{x}}^T \Phi_{\mathbf{x}})^{-1} \Phi_{\mathbf{x}}^T \mathbf{y}$$

# Logistic regression

We can also consider classification problems, where we have a dataset of inputs and targets $\mathcal{D} := (\mathbf{x}_i, y_i)_{i=1}^N$ with $y_i \in \{0, 1\}$

A logistic regression classifier is defined as a model

$$f_\theta(\mathbf{x}) = \sigma(\theta^T \phi(\mathbf{x}))$$

where $\sigma$ is the logistic sigmoid function, given by

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

Now, the output of our model is interpreted as $p(y = 1 \mid x)$

The loss function that we wish to minimise is the **binary cross entropy**:

$$L_{BCE}(\theta) = -\frac{1}{N} \sum_{i=1}^{N} \{y_i \log f_\theta(\mathbf{x}_i) + (1 - y_i) \log(1 - f_\theta(\mathbf{x}_i))\}$$

# Logistic regression

$$L_{BCE}(\theta) = -\frac{1}{N}\sum_{i=1}^{N}\{y_i \log f_\theta(\mathbf{x}_i) + (1 - y_i)\log(1 - f_\theta(\mathbf{x}_i))\}$$

Unlike linear regression, there is no closed-form solution of the logistic regression problem

However, it can be shown that the loss function is convex

To optimise the parameters of the logistic regression model, we need to resort to **gradient-based optimisation**

# Gradient Descent

- Model $f_\theta$
- Loss function $L(\theta;\mathcal{D}) = \frac{1}{N}\sum_{x_i,y_i\in\mathcal{D}} l(y_i, f_\theta(x_i))$
- Initialise $\theta_0$
- Gradient $\nabla_\theta L(\theta_0;\mathcal{D})$ $\nabla_\theta L(\boldsymbol{\theta}_t;\mathcal{D})$
- Update $\theta_1 = \theta_0 - \eta\nabla_\theta L(\theta_0;\mathcal{D})$ $\boldsymbol{\theta}_{t+1} = \theta_t - \eta\nabla_\theta L(\theta_t;\mathcal{D})$

# Stochastic Gradient Descent

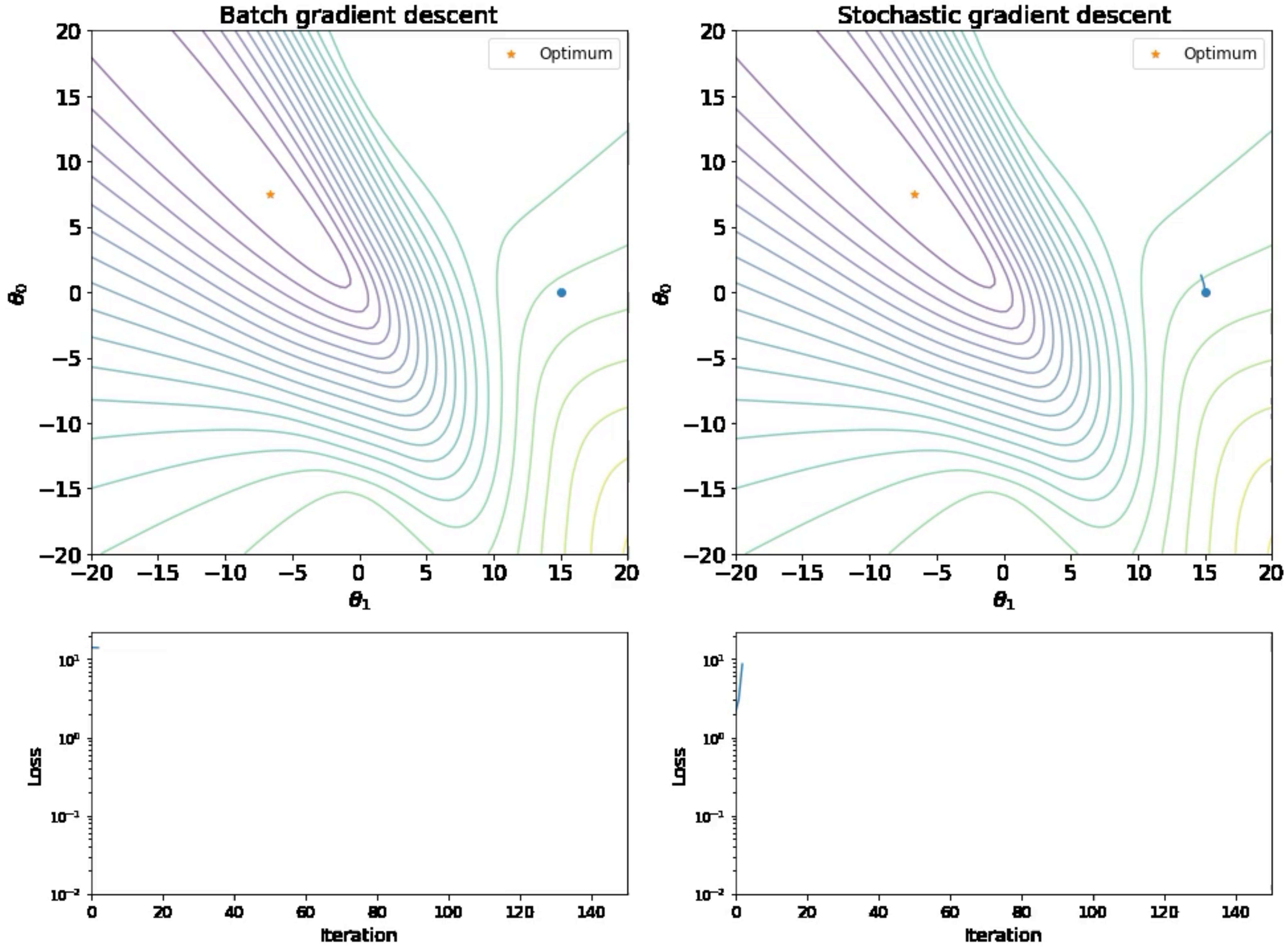
- Model $f_\theta$
- Initialise $\theta_0$

```
for t in range(num_iterations):
```

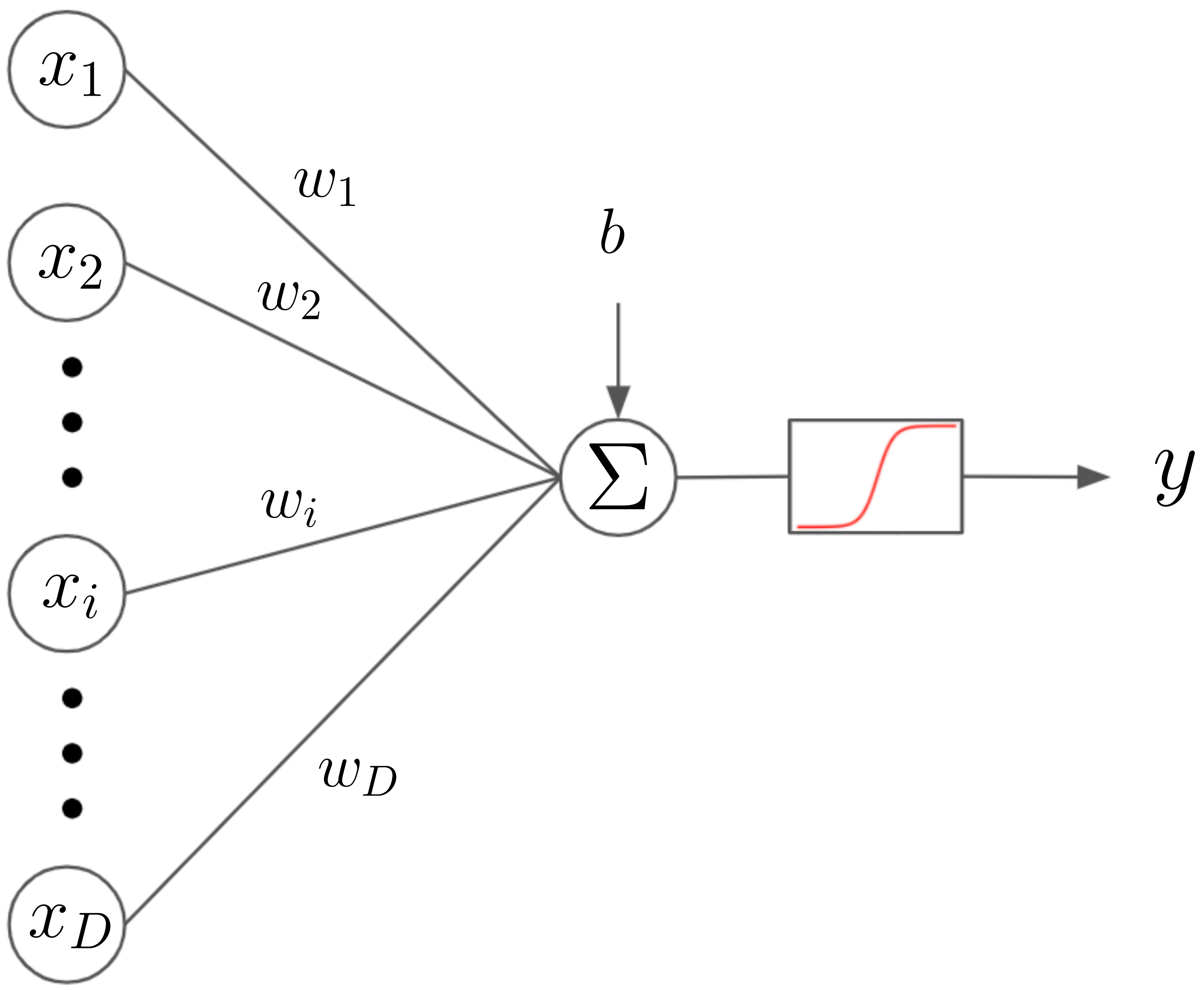
- Sample a minibatch $\mathcal{D}_m$
- Loss function $L(\theta_t; \mathcal{D}_m) = \frac{1}{M} \sum_{x_i, y_i \in \mathcal{D}_m} l(y_i, f_{\theta_t}(x_i))$
- Gradient $\nabla_\theta L(\theta_t; \mathcal{D}_m)$
- Update $\theta_{t+1} = \theta_t - \eta \nabla_\theta L(\theta_t; \mathcal{D}_m)$

Robbins, H. and Monro, S. (1951), "A stochastic approximation method", *The annals of mathematical statistics*, 400–407.

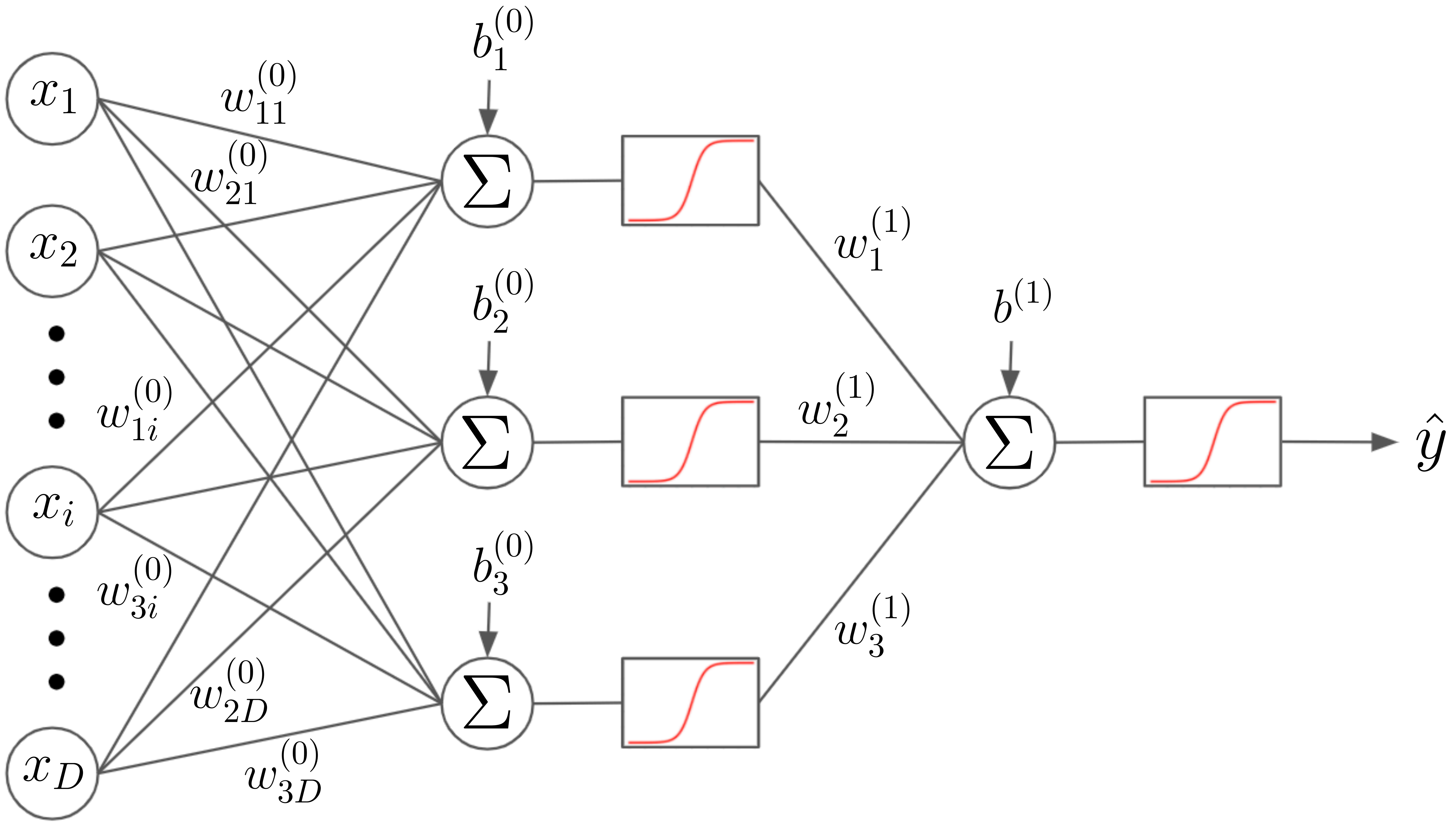
# BGD vs SGD example



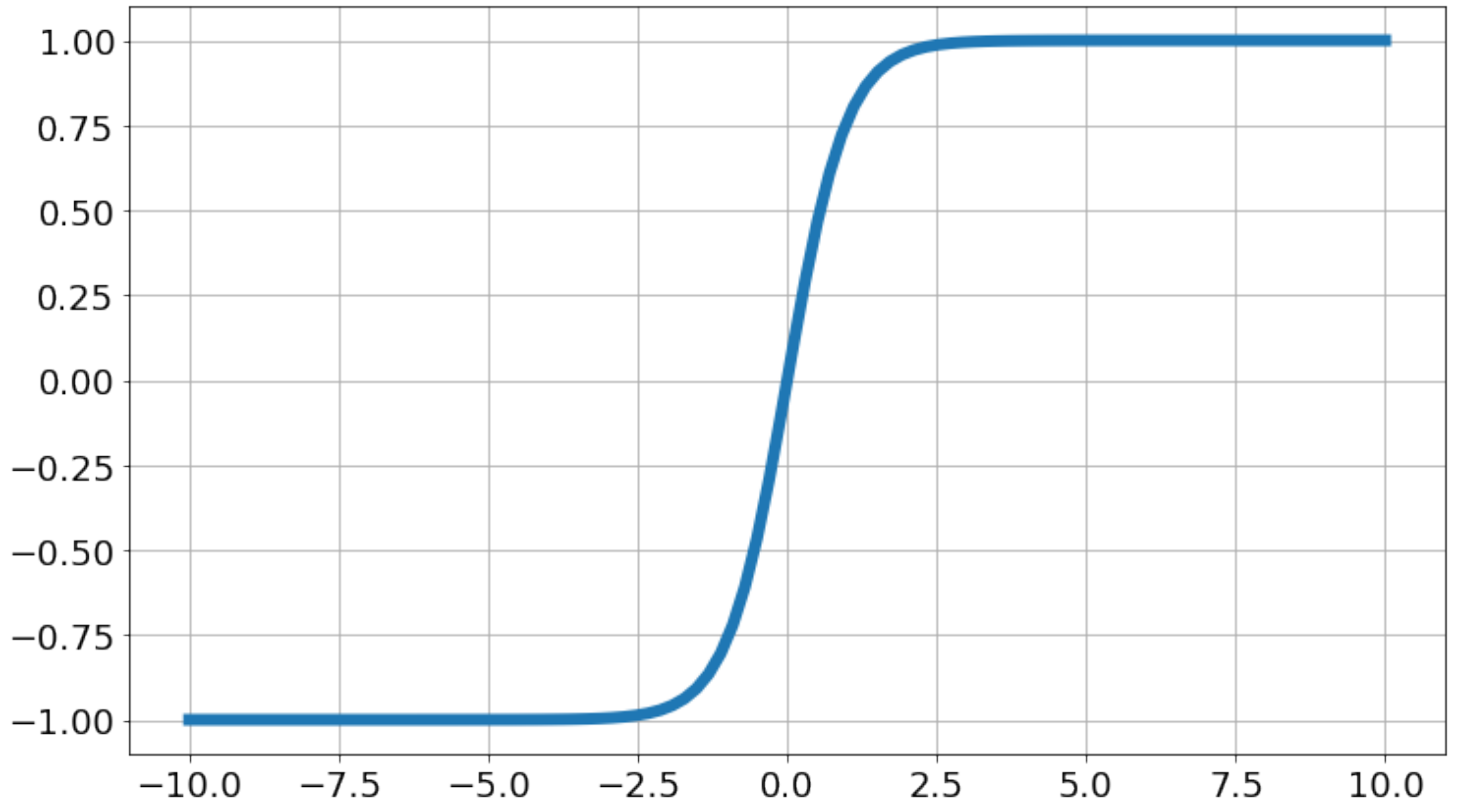
# Artificial Neuron



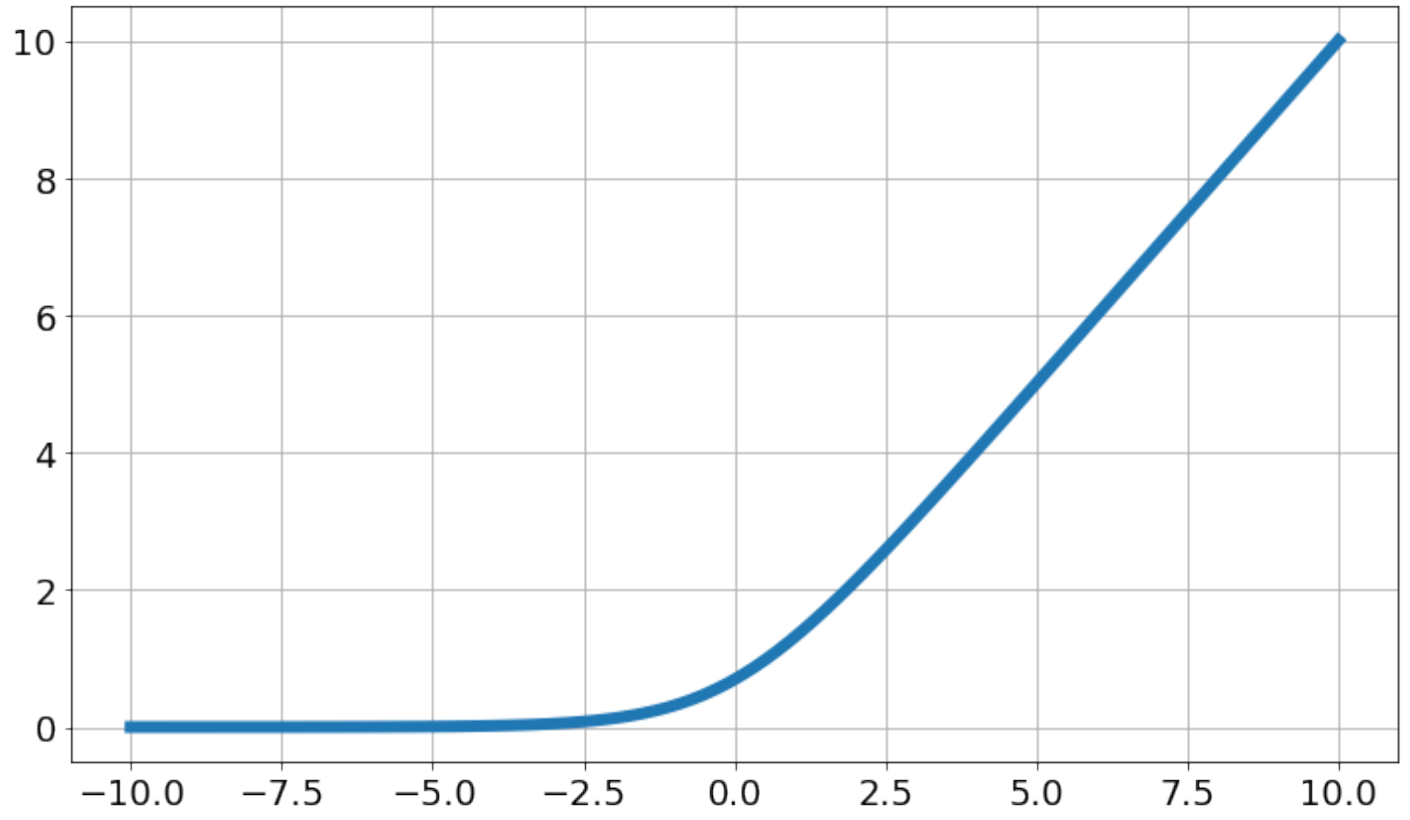
# Multilayer Perceptron with a Single Hidden Layer



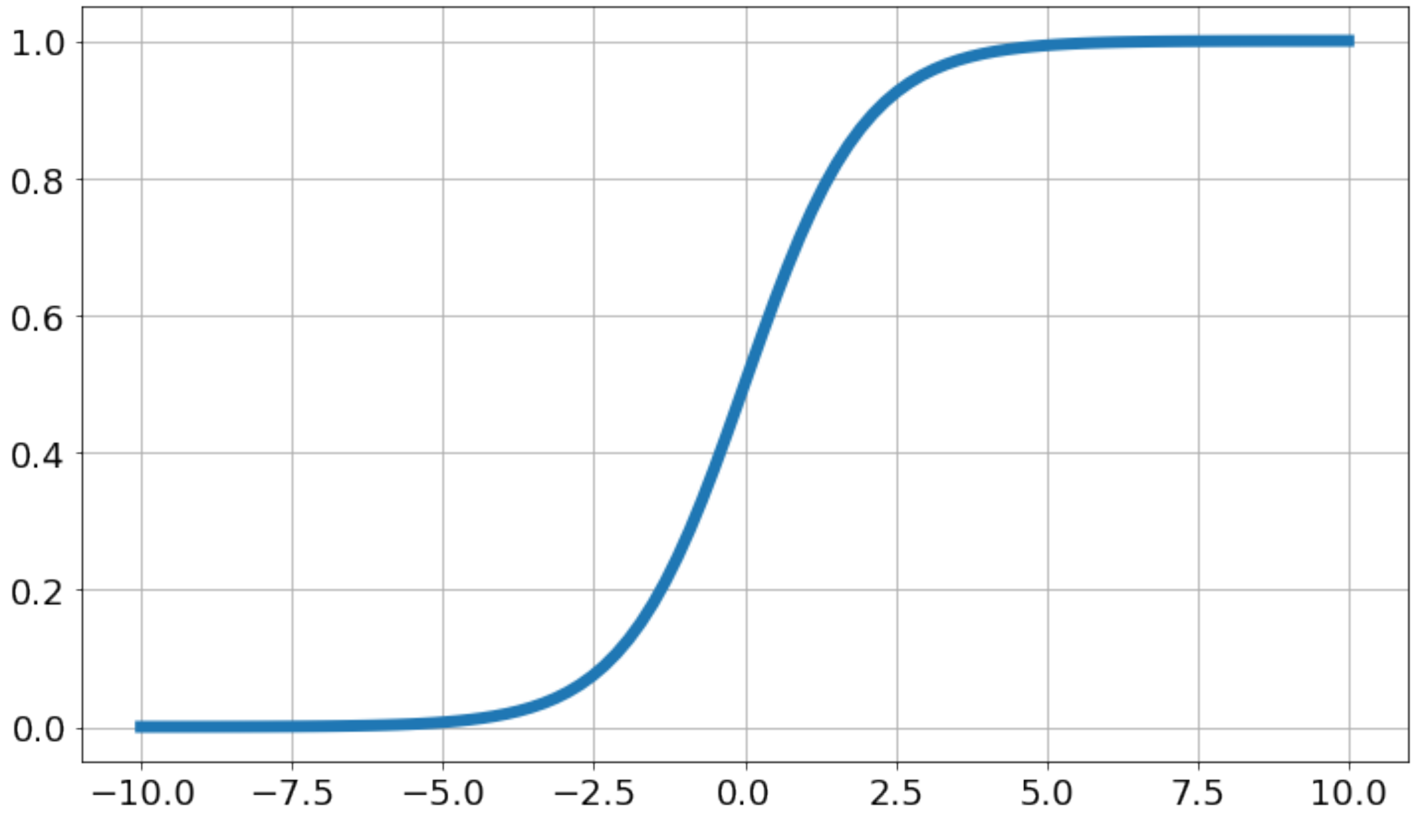
# Activation Functions



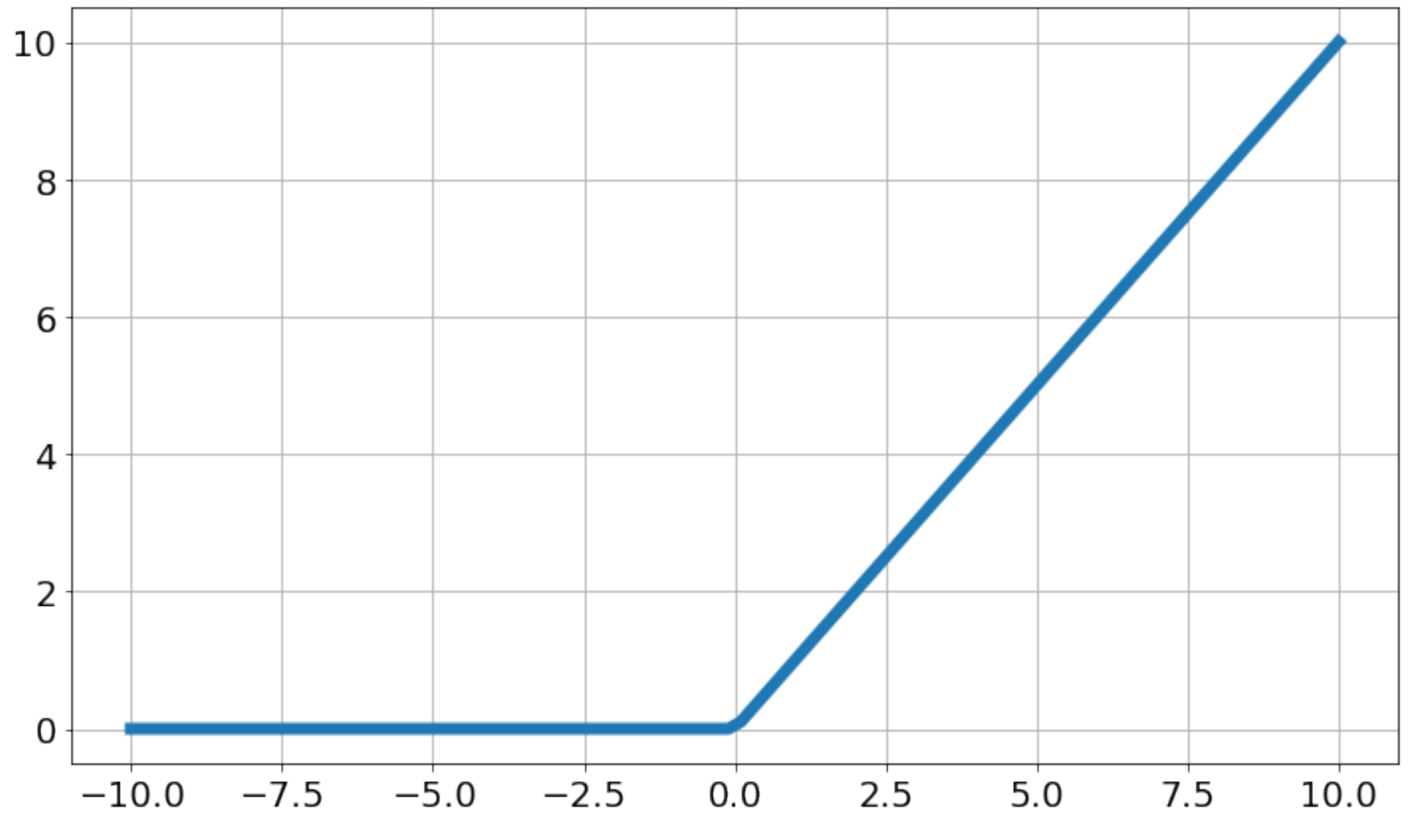

Tanh activation function




Softplus activation function

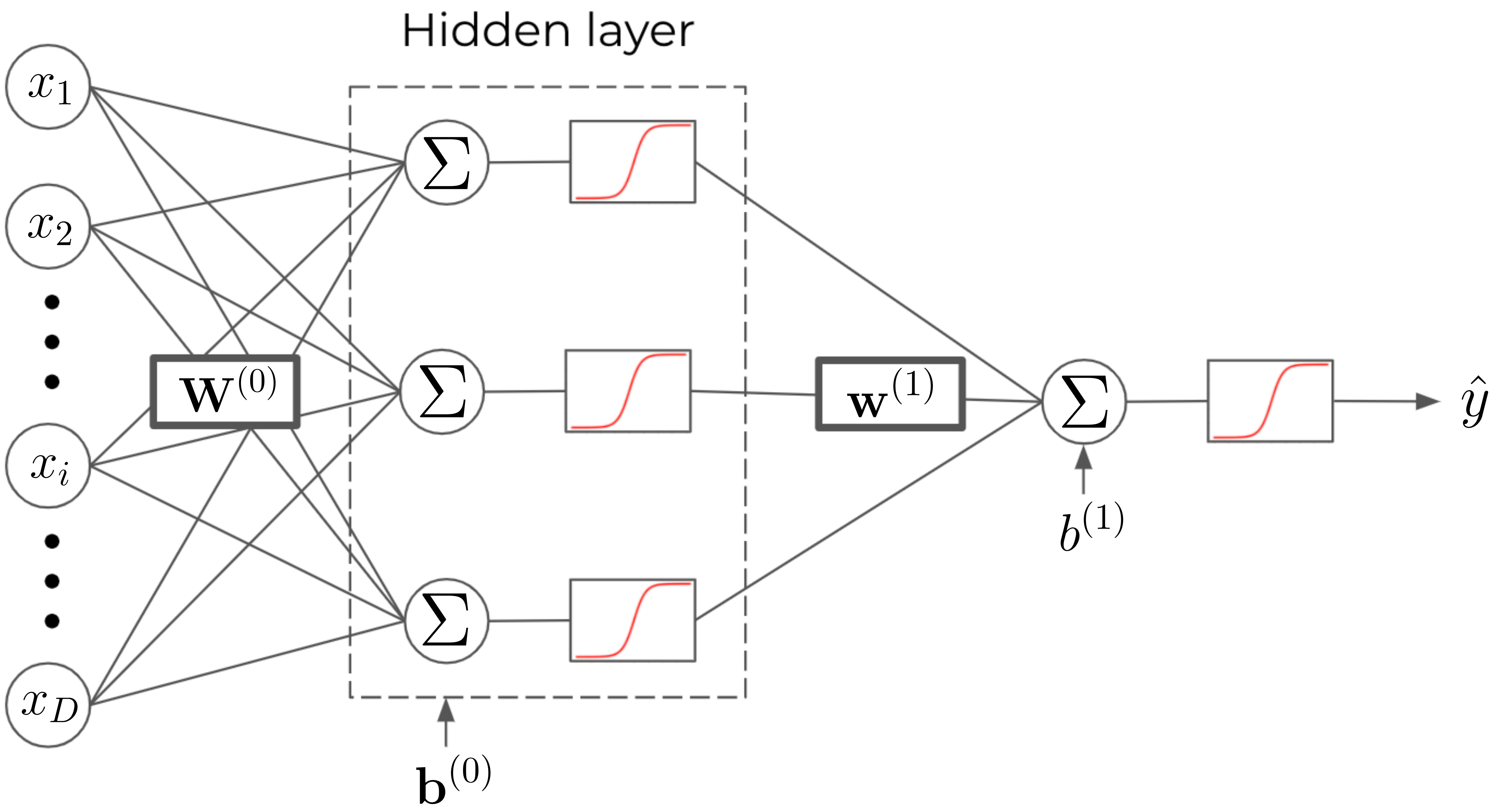



Sigmoid activation function




ReLU activation function

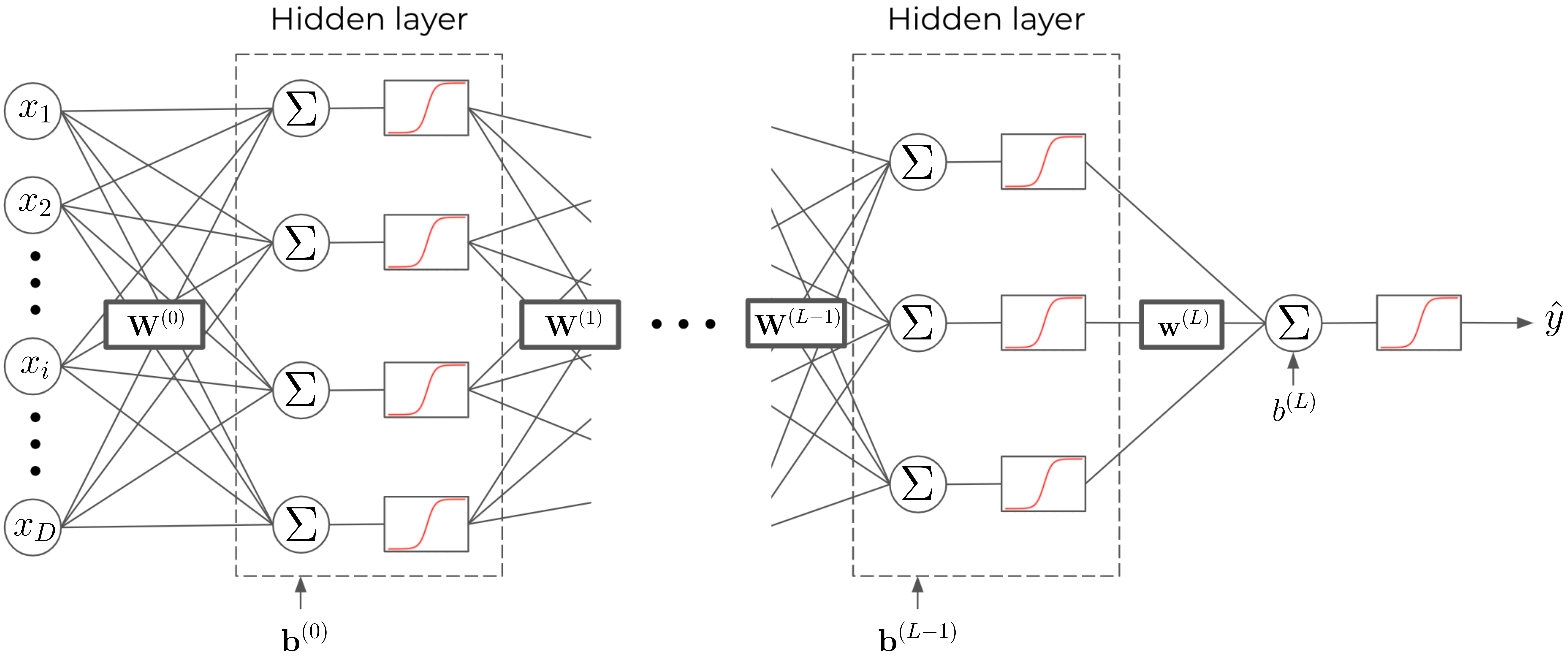
# Multilayer Perceptron with a Single Hidden Layer




$$\mathbf{h}^{(1)} = \sigma\left(\mathbf{W}^{(0)}\mathbf{x} + \mathbf{b}^{(0)}\right)$$

$$\hat{y} = \sigma_{out}\left(\mathbf{w}^{(1)}\mathbf{h}^{(1)} + b^{(1)}\right)$$

# Multilayer Perceptron with Multiple Hidden Layers

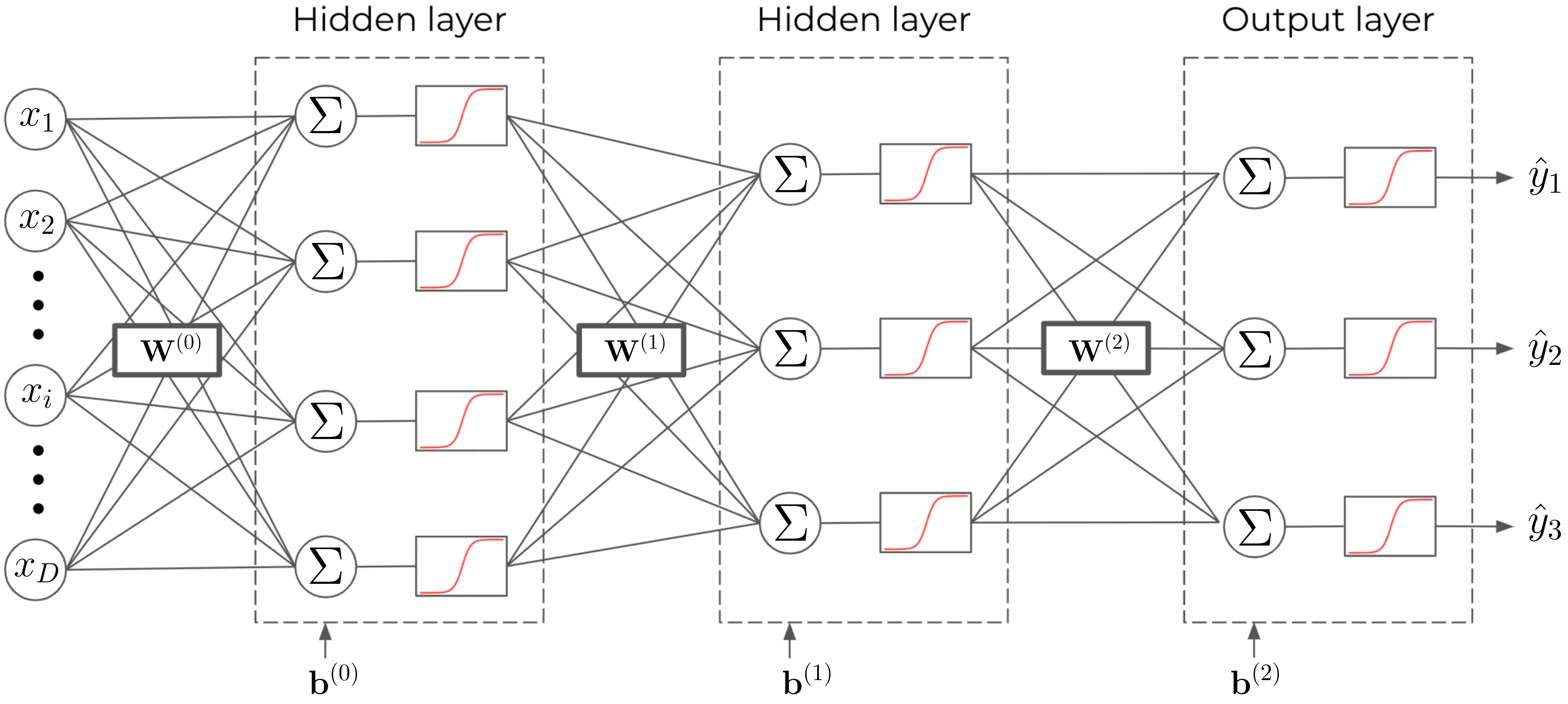



$$\mathbf{h}^{(0)} := \mathbf{x}$$

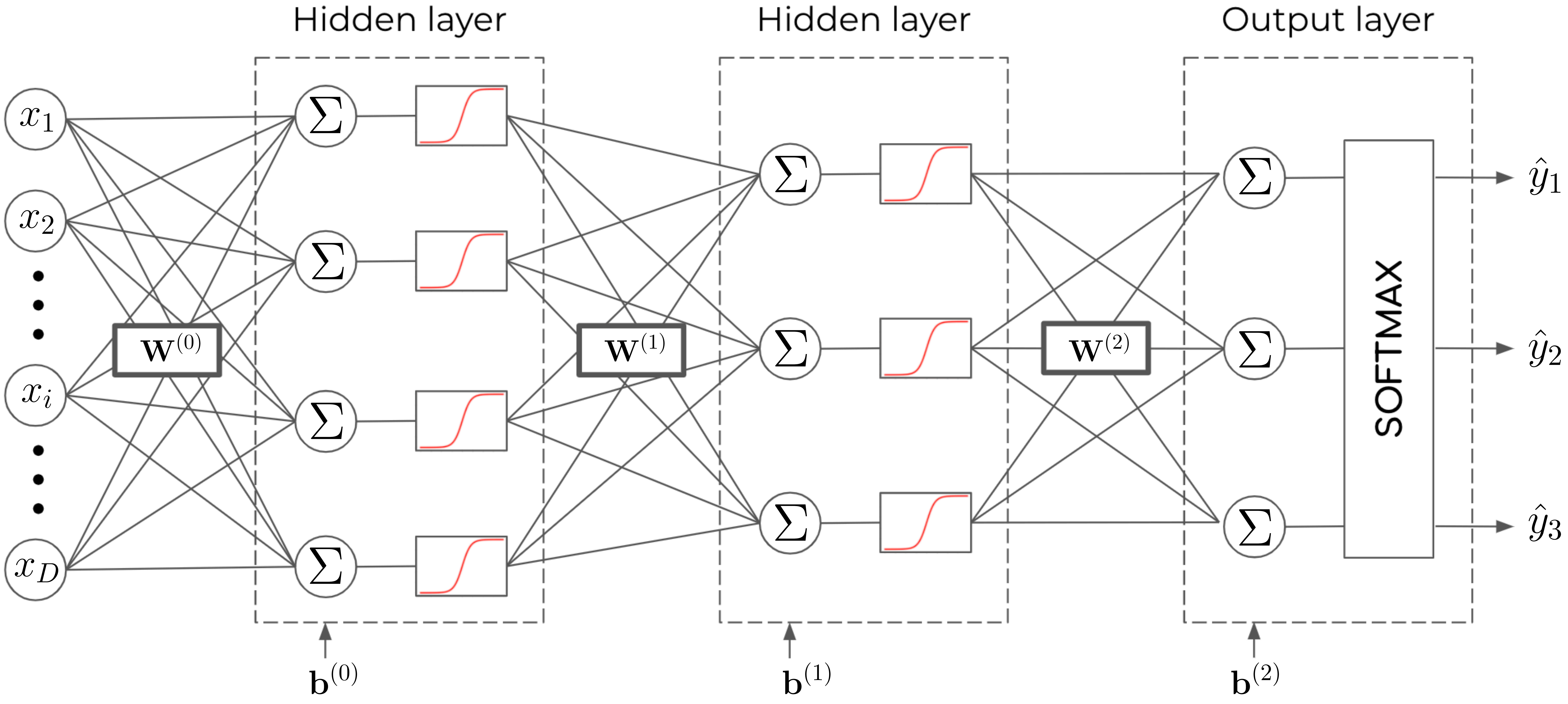
$$\mathbf{h}^{(k)} = \sigma\left(\mathbf{W}^{(k-1)}\mathbf{h}^{(k-1)} + \mathbf{b}^{(k-1)}\right), \quad k = 1, \ldots, L$$

$$\hat{y} = \sigma_{out}\left(\mathbf{w}^{(L)}\mathbf{h}^{(L)} + b^{(L)}\right)$$

# Multilayer Perceptron with Multiple Outputs



# Multilayer Perceptron with Softmax Output




$$\hat{y}_j := \frac{\exp(a_j^{(L+1)})}{\sum_i \exp(a_i^{(L+1)})}$$