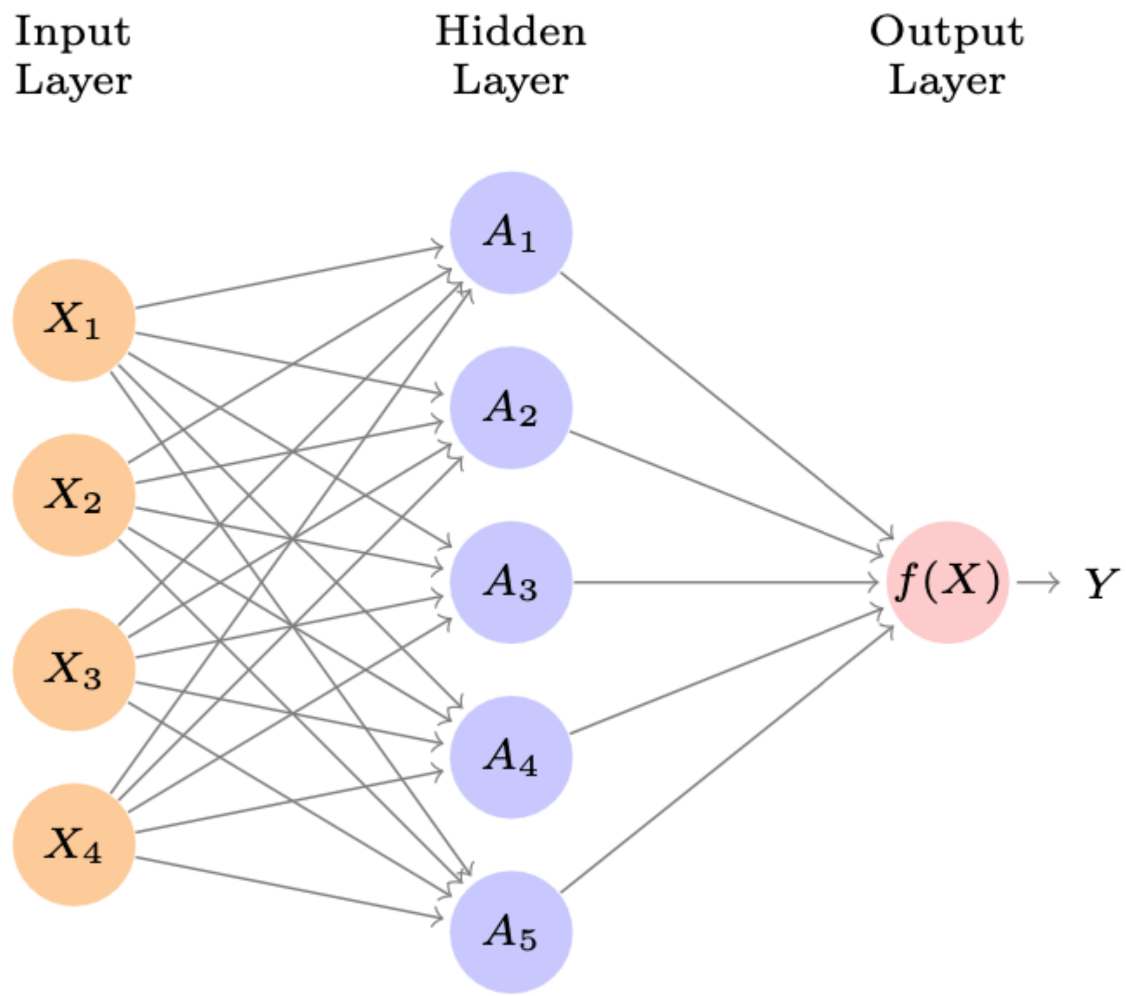
# Lecture 3

- More on neural networks
- Loss functions and fitting
- Overfitting and underfitting

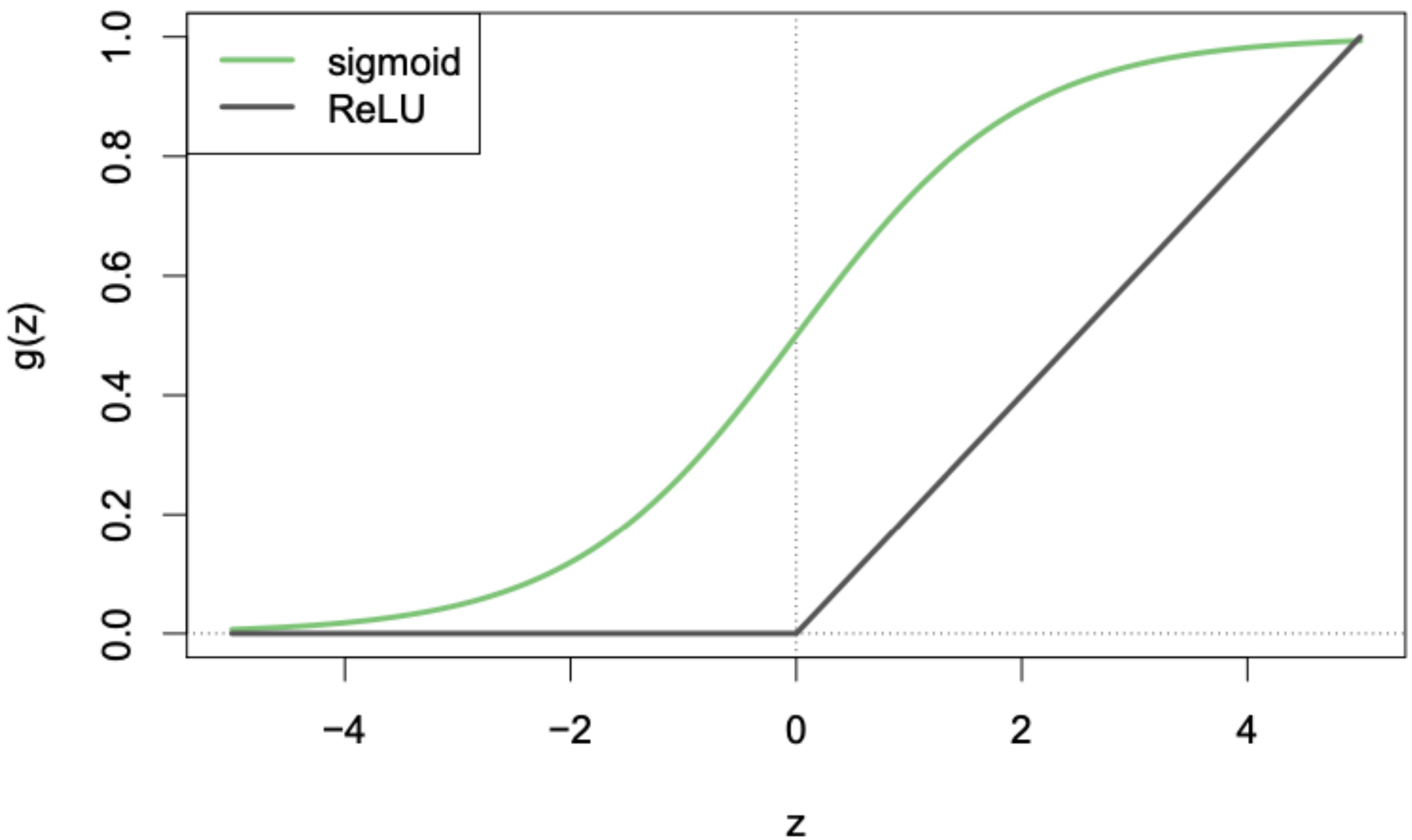
**September 19, 2024**

# Single Layer Neural Network

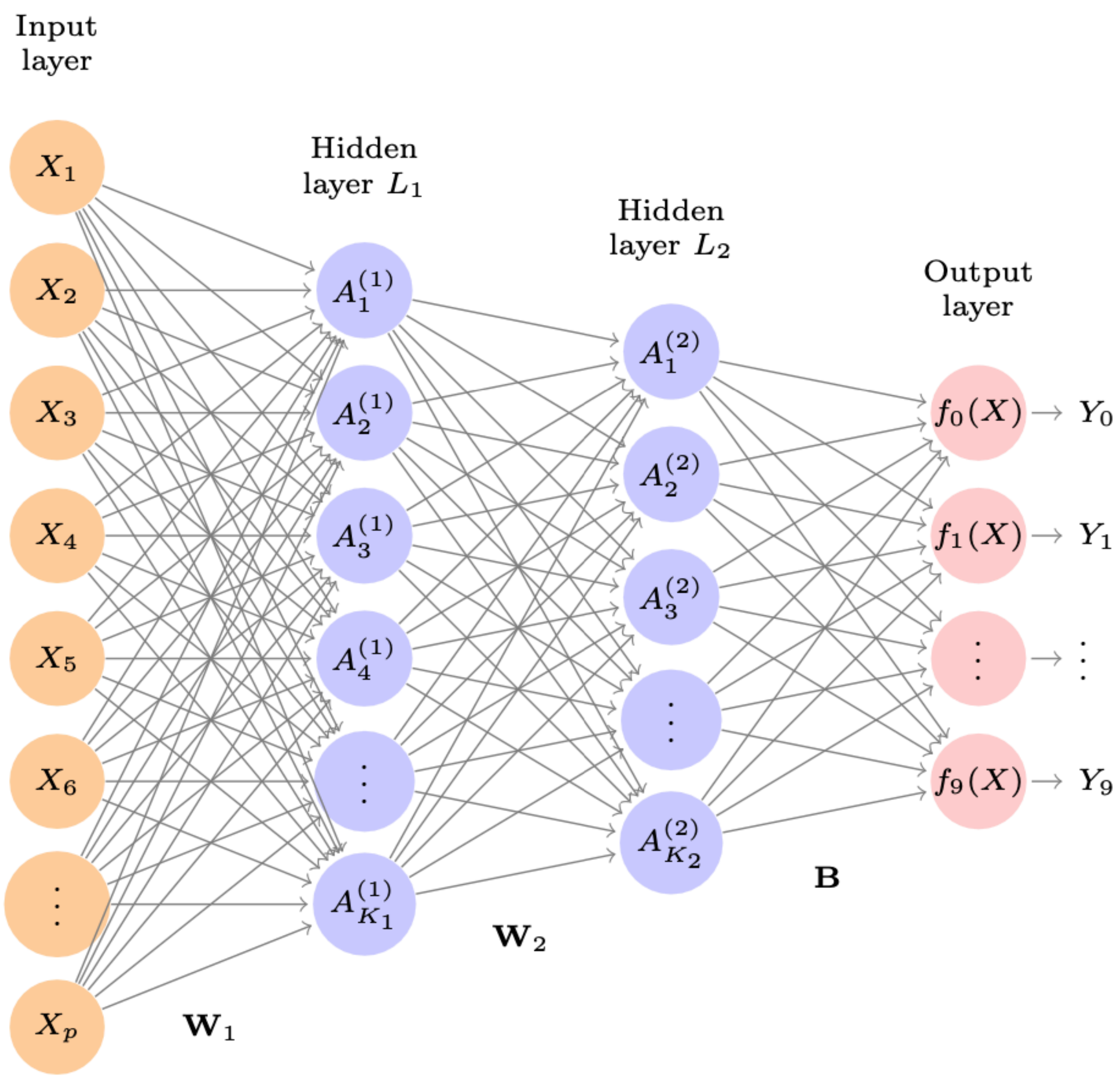



$$\begin{aligned} f(X) &= \beta_0 + \textstyle\sum_{k=1}^{K} \beta_k h_k(X) \\ &= \beta_0 + \textstyle\sum_{k=1}^{K} \beta_k g(w_{k0} + \sum_{j=1}^{p} w_{kj} X_j). \end{aligned}$$

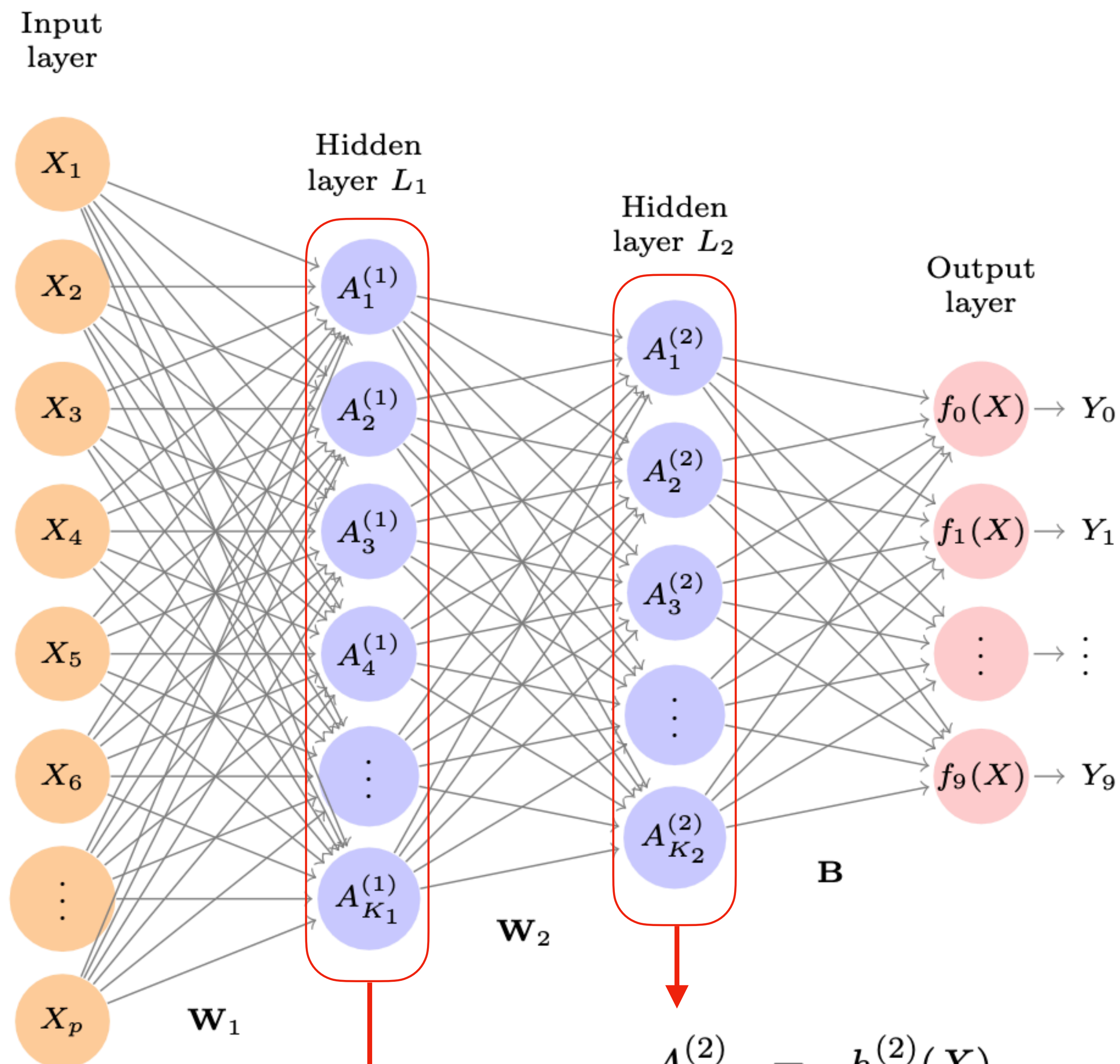
# Activation Functions



# Multilayer Neural Network



# Multilayer Neural Network



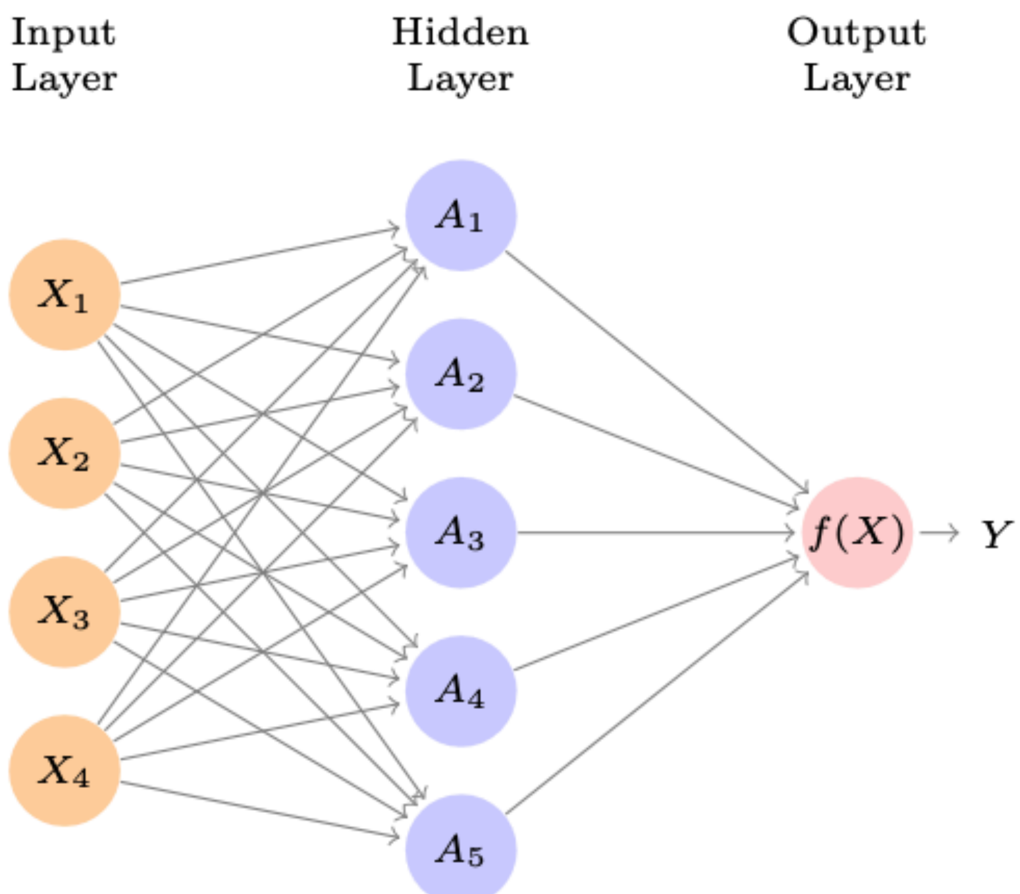

$$
\begin{aligned}
A_\ell^{(2)} &= h_\ell^{(2)}(X) \\
&= g(w_{\ell 0}^{(2)} + \textstyle\sum_{k=1}^{K_1} w_{\ell k}^{(2)} A_k^{(1)})
\end{aligned}
$$

$$
\begin{aligned}
A_k^{(1)} &= h_k^{(1)}(X) \\
&= g(w_{k0}^{(1)} + \textstyle\sum_{j=1}^{p} w_{kj}^{(1)} X_j)
\end{aligned}
$$

# Number of Parameters

- Suppose we have
  - 784 inputs
  - 256 activations in the first hidden layer
  - 128 activations in the second hidden layer
- The first matrix of weights $\mathbf{W_1}$ contains $785 \cdot 256 \approx 201{,}000$ entries.
- The second matrix of weights $\mathbf{W_2}$ contains $257 \cdot 128 \approx 33{,}000$ entries.

# Fitting Neural Networks



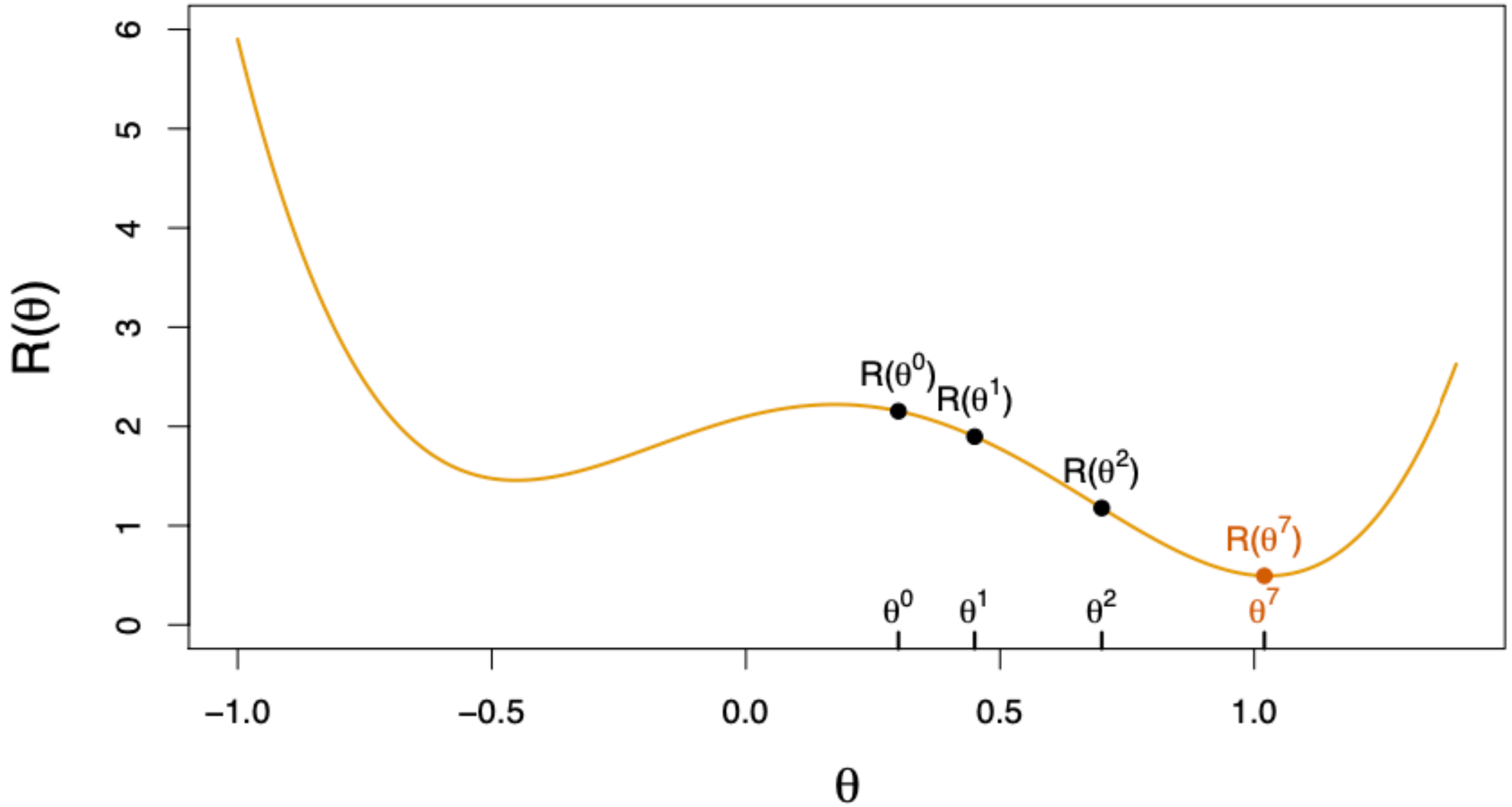

$$\underset{\{w_k\}_1^K,\ \beta}{\text{minimize}} \frac{1}{2}\sum_{i=1}^{n}(y_i - f(x_i))^2,$$

where

$$f(x_i) = \beta_0 + \sum_{k=1}^{K} \beta_k g\Big(w_{k0} + \sum_{j=1}^{p} w_{kj} x_{ij}\Big).$$

# Non-Convex Functions and Gradient Descent

Let $R(\theta) = \frac{1}{2}\sum_{i=1}^{n}(y_i - f_\theta(x_i))^2$ with $\theta = (\{w_k\}_1^K,\ \beta)$.




1. Start with a guess $\theta^0$ for all the parameters in $\theta$, and set $t = 0$.
2. Iterate until the objective $R(\theta)$ fails to decrease:
   (a) Find a vector $\delta$ that reflects a small change in $\theta$, such that $\theta^{t+1} = \theta^t + \delta$ *reduces* the objective; i.e. $R(\theta^{t+1}) < R(\theta^t)$.
   (b) Set $t \leftarrow t + 1$.

# Gradient Descent

- In this simple example we reached the *global minimum*.
- If we had started a little to the left of $\theta^0$ we would have gone in the other direction, and ended up in a *local minimum*.
- Although $\theta$ is multi-dimensional, we have depicted the process as one-dimensional. It is much harder to identify whether one is in a local minimum in high dimensions.

How to find a direction $\delta$ that points downhill? We compute the *gradient vector*

$$\nabla R(\theta^t) = \frac{\partial R(\theta)}{\partial \theta}\Big|_{\theta=\theta^t}$$

i.e. the vector of *partial derivatives* at the current guess $\theta^t$. The gradient points uphill, so our update is $\delta = -\rho \nabla R(\theta^t)$ or

$$\theta^{t+1} \leftarrow \theta^t - \rho \nabla R(\theta^t),$$

where $\rho$ is the *learning rate* (typically small, e.g. $\rho = 0.001$.

# Gradients and Backprobagation

$R(\theta) = \sum_{i=1}^n R_i(\theta)$ is a sum, so gradient is sum of gradients.

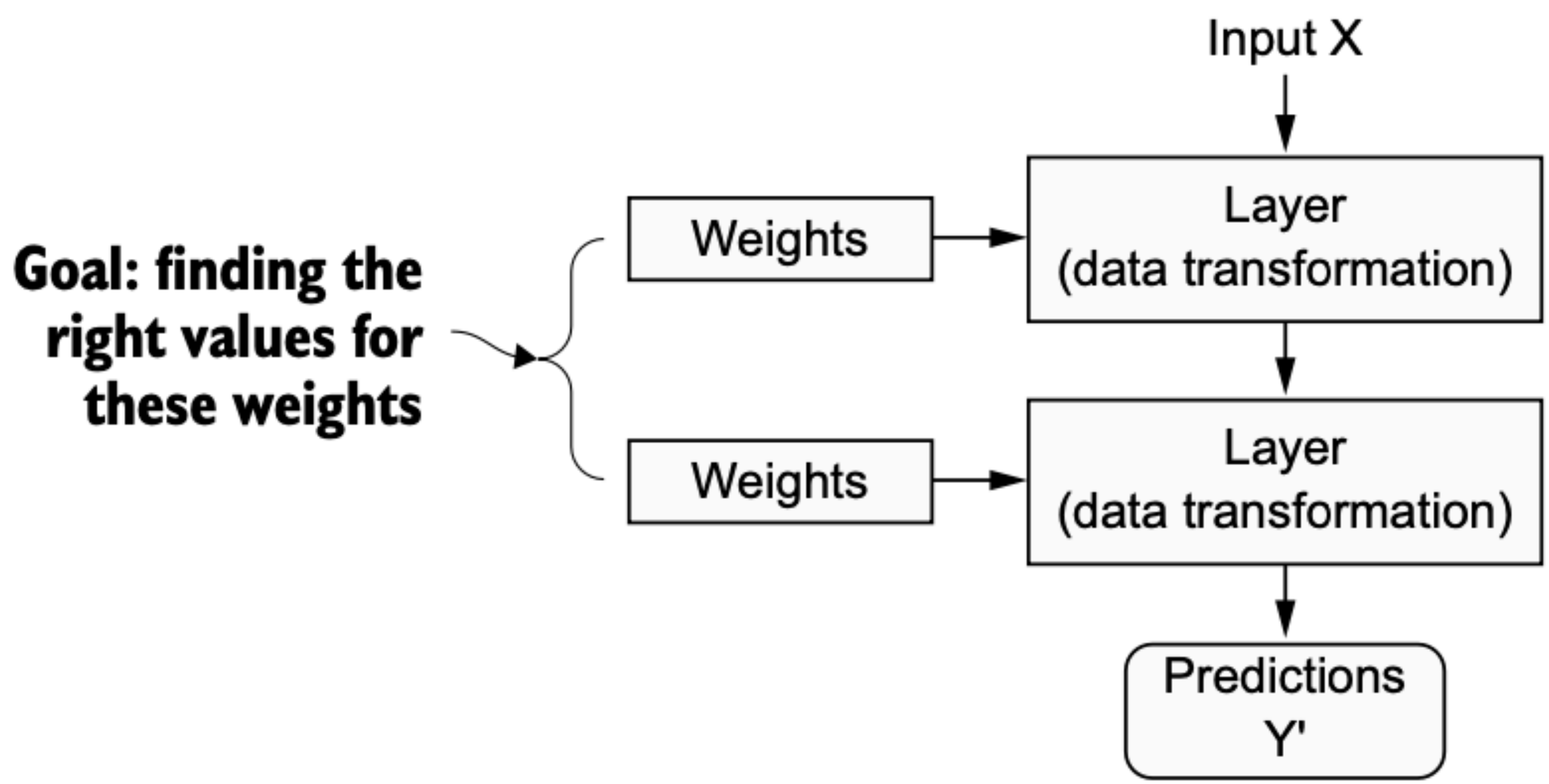
$$R_i(\theta) = \frac{1}{2}(y_i - f_\theta(x_i))^2 = \frac{1}{2}\Big(y_i - \beta_0 - \sum_{k=1}^{K} \beta_k g\big(w_{k0} + \sum_{j=1}^{p} w_{kj}x_{ij}\big)\Big)^2$$

For ease of notation, let $z_{ik} = w_{k0} + \sum_{j=1}^p w_{kj}x_{ij}$.

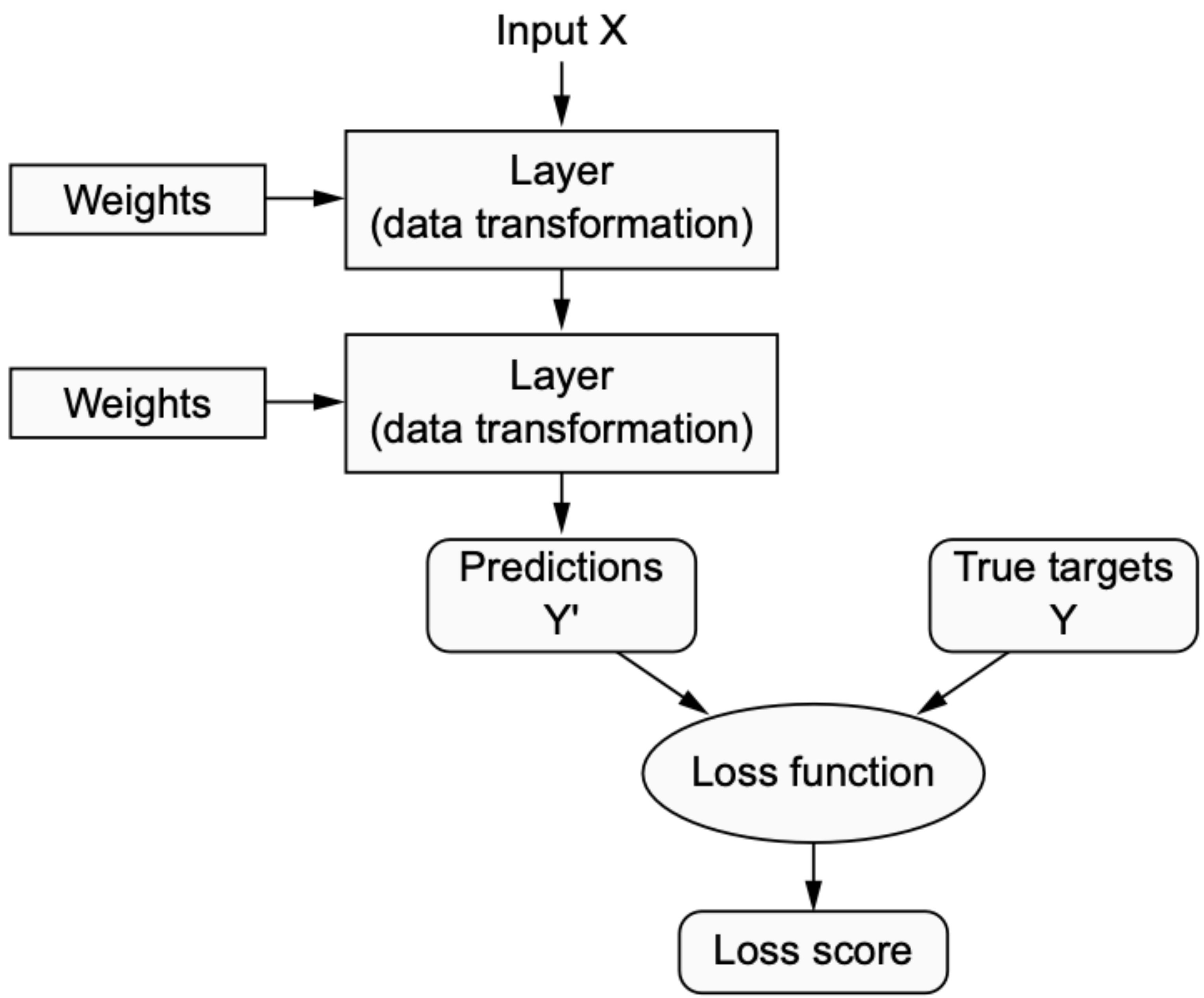
Backpropagation uses the *chain rule for differentiation*:

$$\begin{aligned}
\frac{\partial R_i(\theta)}{\partial \beta_k} &= \frac{\partial R_i(\theta)}{\partial f_\theta(x_i)} \cdot \frac{\partial f_\theta(x_i)}{\partial \beta_k} \\
&= -(y_i - f_\theta(x_i)) \cdot g(z_{ik}). \\
\frac{\partial R_i(\theta)}{\partial w_{kj}} &= \frac{\partial R_i(\theta)}{\partial f_\theta(x_i)} \cdot \frac{\partial f_\theta(x_i)}{\partial g(z_{ik})} \cdot \frac{\partial g(z_{ik})}{\partial z_{ik}} \cdot \frac{\partial z_{ik}}{\partial w_{kj}} \\
&= -(y_i - f_\theta(x_i)) \cdot \beta_k \cdot g'(z_{ik}) \cdot x_{ij}.
\end{aligned}$$

# A neural network is parameterized by its weights



# Loss function measures quality of the network’s output



# Loss score is used as a feedback signal to adjust the weights