Multilayer NN – DL Fitting

Aprendizaje Bioestadístico

Edwin Santiago Alférez

EEBE- UPC

May 2025

Outline

✓Single (Hidden) Layer Neural Network

✓Multilayer Neural Networks

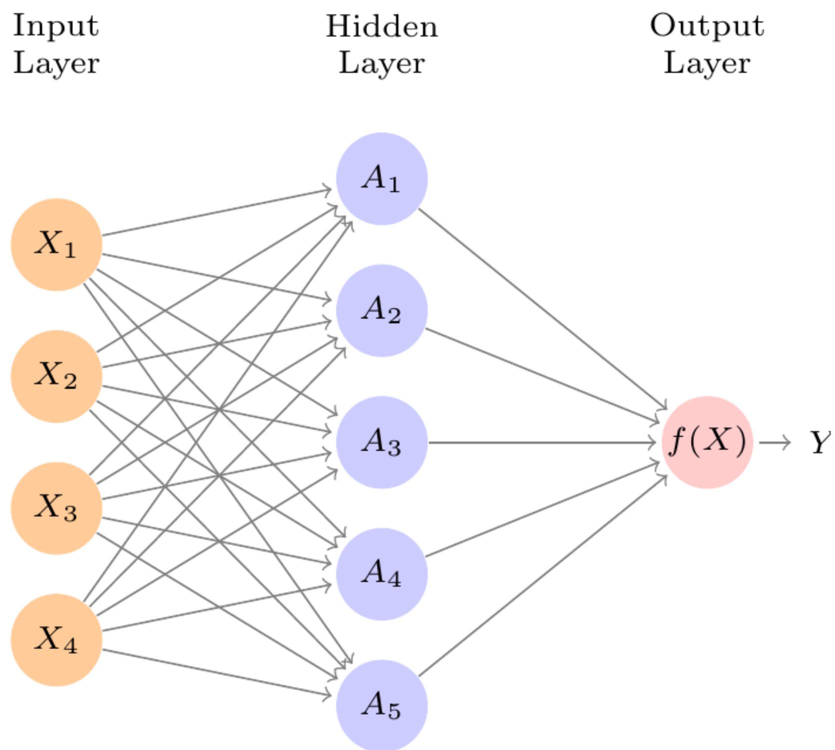
✓MNIST Example and Results

✓Fitting Neural Networks

✓Practice

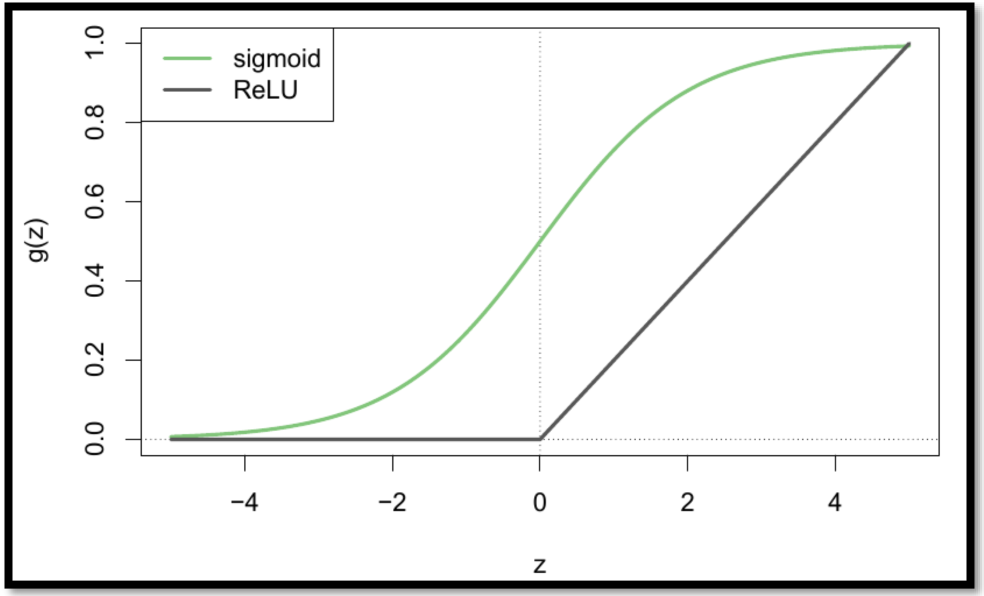
Single (Hidden) Neural Network

Single (Hidden) Layer Neural Network



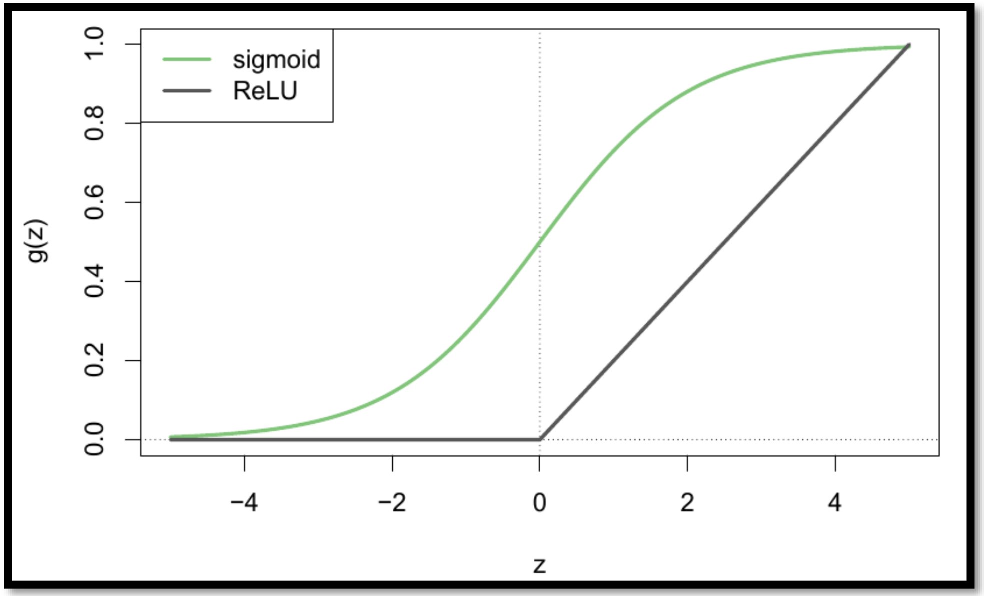
$$f(X) = \beta_0 + \sum_{k=1}^{K} \beta_k h_k(X) = \beta_0 + \sum_{k=1}^{K} \beta_k g\left(w_{k0} + \sum_{j=1}^{p} w_{kj} X_j\right)$$

Some Details



- $A_k = h_k(X) = g(\omega_{k0} + \sum_{j=1}^{p} \omega_{kj} X_j)$are called the activations in the hidden layer.
- $g(z)$ is called the activation function. Popular are the sigmoid and rectified linear, shown in figure.
- Activation functions in hidden layers are typically nonlinear, otherwise the model collapses to a linear model.
- So the activations are like derived features - nonlinear transformations of linear combinations of the features.
- The model is fit by minimizing $\sum_{i=1}^{n} \left(y_i - f(x_i)\right)^2$(e.g. for regression).

Some Details

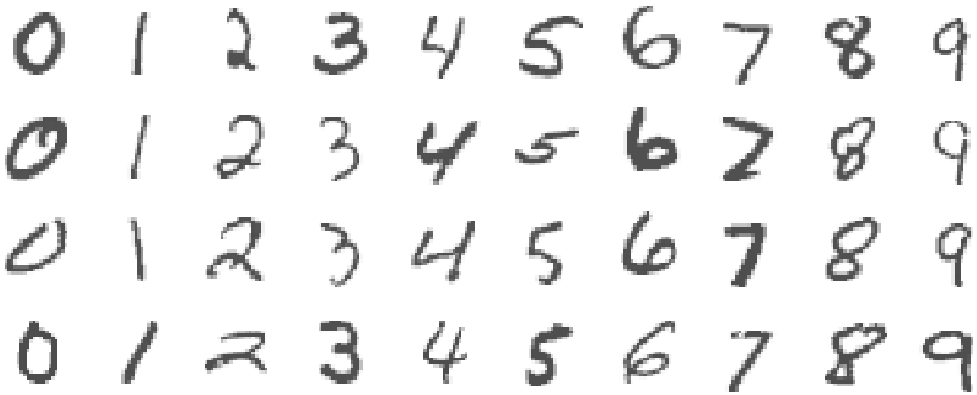


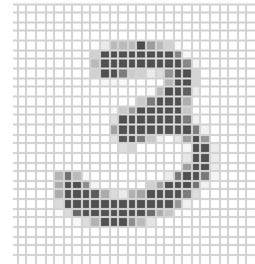
$$g(z) = \frac{e^z}{1+e^z} = \frac{1}{1+e^{-z}}$$

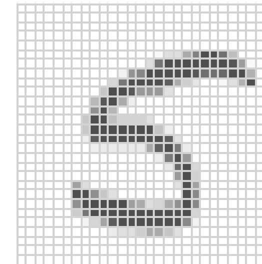
$$g(z) = (z)_+ = \begin{cases} 0 & \text{if } z < 0 \\ z & \text{otherwise} \end{cases}$$

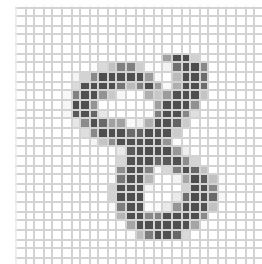
Multilayer Neural Networks

Multilayer Neural Networks





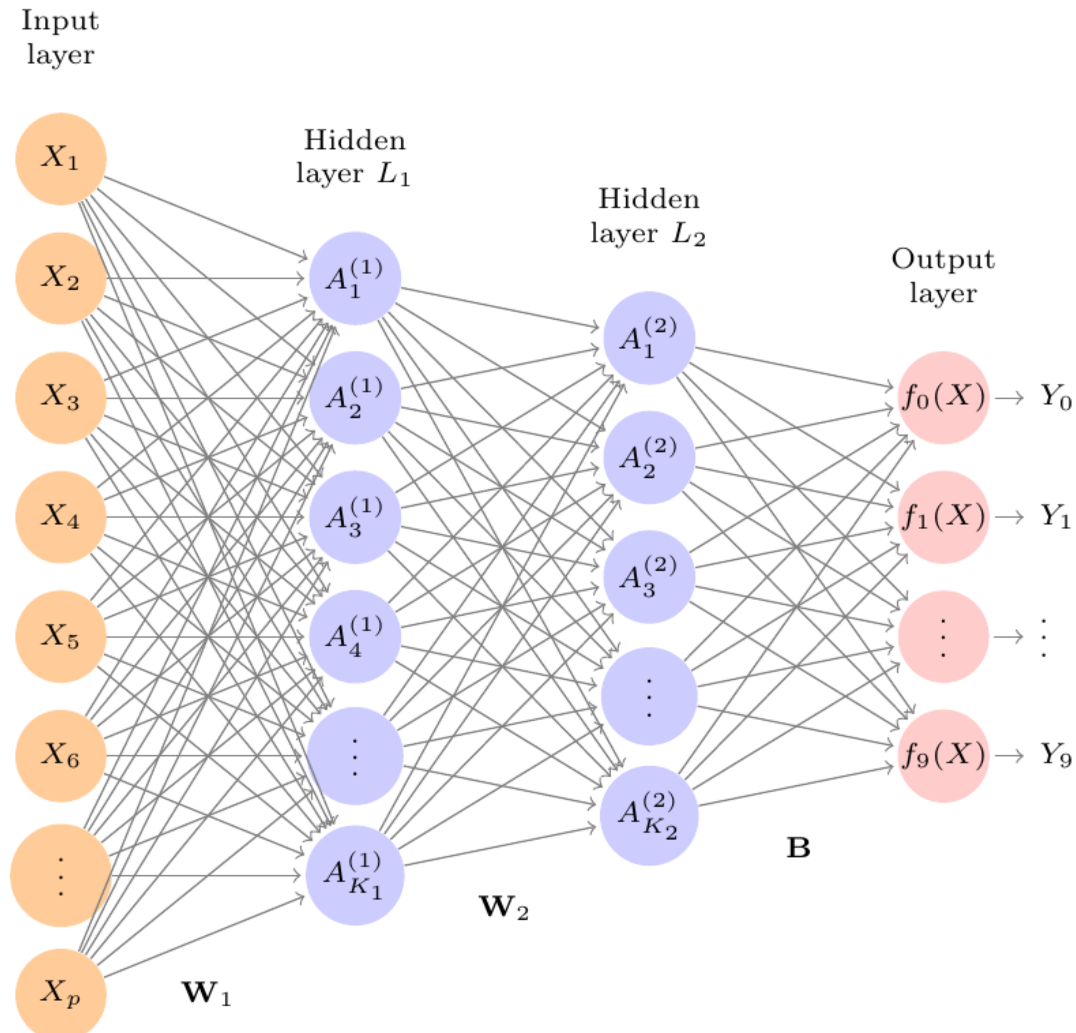




- Handwritten digits 28 × 28 grayscale images
- 60K train, 10K test images Features are the 784 pixel grayscale values ∈ (0, 255)
- Labels are the digit class 0–9

- Goal: build a classifier to predict the image class.
- We build a two-layer network with 256 units at first layer, 128 units at second layer, and 10 units at output layer.
- Along with intercepts (called biases) there are 235,146 parameters (referred to as weights)

Multilayer Neural Networks



Details of the First and Second Layer

✓The first hidden layer is represented by

$$A_k^{(1)} = h_k^{(1)}(X) = g\left(w_{k0}^{(1)} + \sum_{j=1}^{p} w_{kj}^{(1)} X_j\right)$$

for $k = 1, \ldots, K_1$.

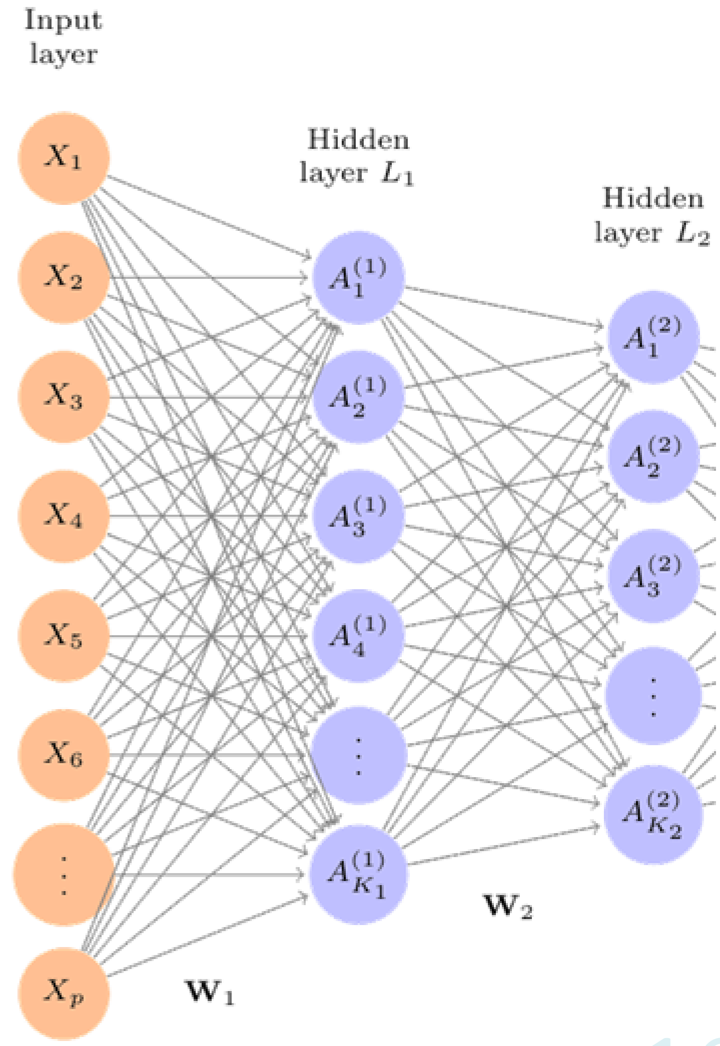
✓The second hidden layer treats the activations $A_k^{(1)}$ of the first hidden layer as inputs and computes new activations

$$A_\ell^{(2)} = h_\ell^{(2)}(X) = g\left(w_{\ell 0}^{(2)} + \sum_{k=1}^{K_1} w_{\ell k}^{(2)} A_k^{(1)}\right)$$

for $\ell = 1, \ldots, K_2$.

✓Notice that each of the activations in the second layer $A_\ell^{(2)} = h_\ell^{(2)}(X)$ is a function of the input vector X.

✓This is the case because while they are explicitly a function of the activations $A_k^{(1)}$ from layer L_1, these in turn are functions of X.

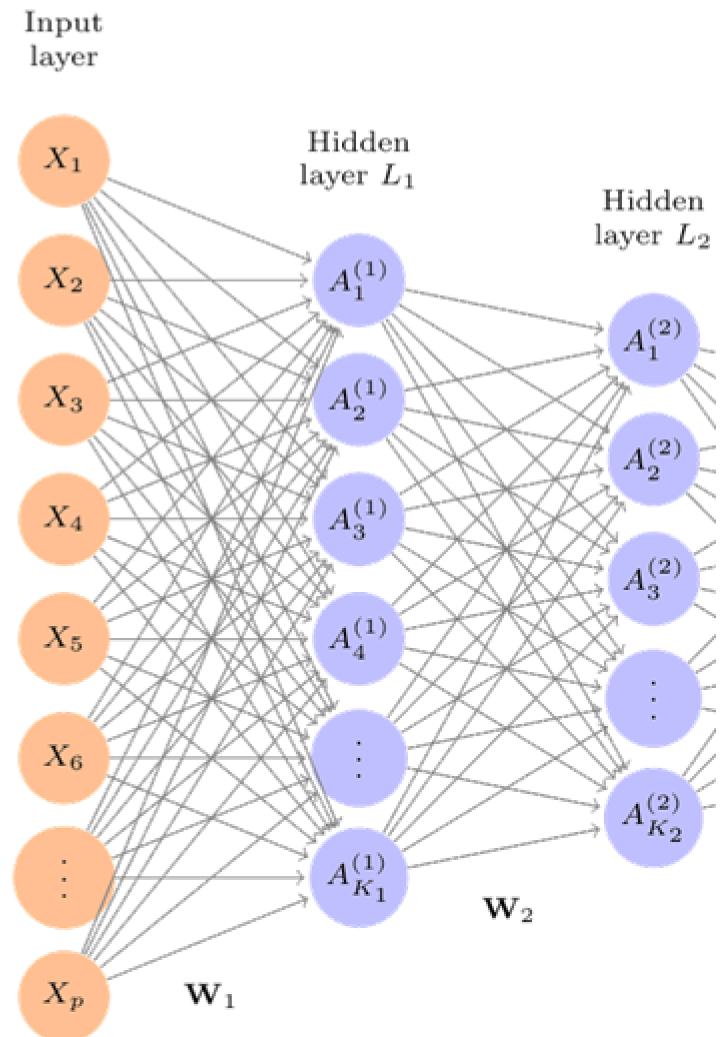


Details of the First and Second Layer

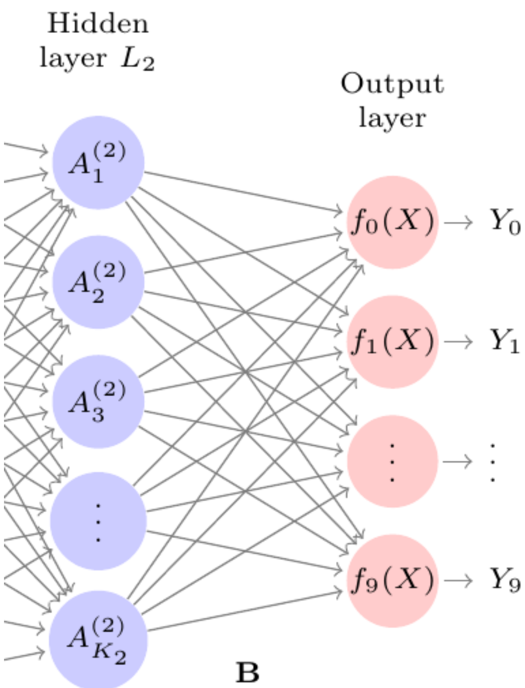
✓The notation $\mathbf{W}_1$ represents the entire matrix of weights that feed from the input layer to the first hidden layer L_1.

✓This matrix will have $785 \times 256 = 200{,}960$ elements; there are 785 rather than 784 because we must account for the intercept or bias term.

✓Each element $A_k^{(1)}$ feeds to the second hidden layer L_2 via the matrix of weights $\mathbf{W}_2$ of dimension $257 \times 128 = 32{,}896$.



Details of the Output Layer



✓Let $Z_m = \beta_{m0} + \sum_{\ell=1}^{K_2} \beta_{m\ell} A_\ell^{(2)}, m = 0,1,\ldots,9$ be 10 linear combinations of activations at second layer.

✓Output activation function encodes the softmax function:

$$f_m(X) = \Pr(Y = m \mid X) = \frac{e^{Z_m}}{\sum_{\ell=0}^{9} e^{Z_\ell}}.$$

✓We fit the model by minimizing the negative multinomial log-likelihood (or cross-entropy):

$$-\sum_{i=1}^{n}\sum_{m=0}^{9} y_{im}\log(f_m(x_i)).$$

✓y_{im} is 1 if true class for observation i is m, else 0 - i.e. one-hot encoded.

Results

Method	Test Error
Neural Network + *Ridge Regularization*	2.3%
Neural Network + *Dropout Regularization*	1.8%
Multinomial Logistic Regression	7.2%
Linear Discriminant Analysis	12.7%

- ✓ Early success for neural networks in the 1990s.
- ✓ With so many parameters, regularization is essential.
- ✓ Some details of regularization and fitting will come later.
- ✓ Very overworked problem - best reported rates are $< 0.5\%$!
- ✓ Human error rate is reported to be around 0.2%, or 20 of the $10\ \mathrm{K}$ test images.

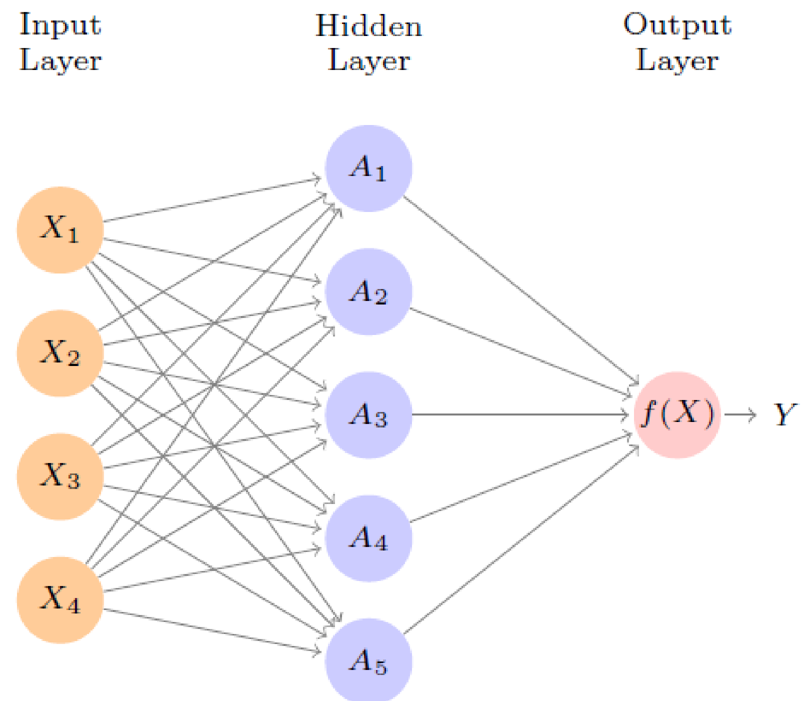
Fitting Neural Networks

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

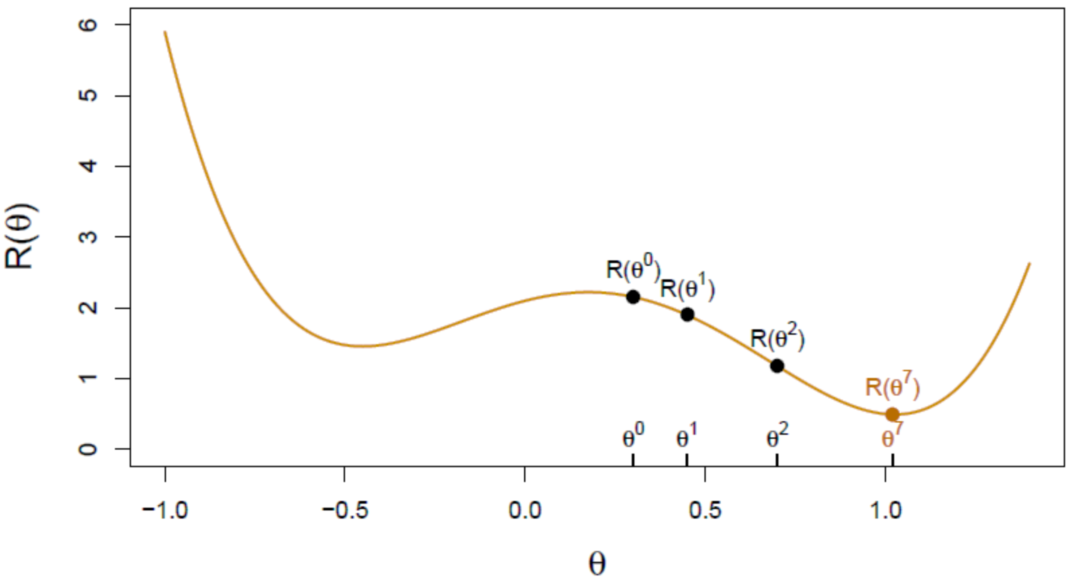
$$f(x_i) = \beta_0 + \sum_{k=1}^{K} \beta_k g\left(w_{k0} + \sum_{j=1}^{p} w_{kj} x_{ij}\right)$$



- This problem is difficult because the objective is non-convex.
- Despite this, effective algorithms have evolved that can optimize complex neural network problems efficiently.

Non Convex Functions and Gradient Descent

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



1. Start with a guess θ^0 for all the parameters in θ, and set $t = 0$.
2. Iterate until the objective $R(\theta)$ fails to decrease:
   a) Find a vector δ that reflects a small change in θ, 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 Continued

✓In this simple example we reached the global minimum.

✓If we had started a little to the left of θ^0 we would have gone in the other direction, and ended up in a local minimum.

✓Although θ 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 δ that points downhill? We compute the gradient vector

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

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

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

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

Gradients and Backpropagation

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

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

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

Backpropagation uses the chain rule for differentiation:

$$\frac{\partial R_i(\theta)}{\partial \beta_k} = \frac{\partial R_i(\theta)}{\partial f_\theta(x_i)} \cdot \frac{\partial f_\theta(x_i)}{\partial \beta_k}$$

$$= -\left(y_i - f_\theta(x_i)\right) \cdot g(z_{ik}).$$

$$\frac{\partial R_i(\theta)}{\partial w_{kj}} = \frac{\partial R_i(\theta)}{\partial f_\theta(x_i)} \cdot \frac{\partial f_\theta(x_i)}{\partial g(z_{ik})} \cdot \frac{\partial g(z_{ik})}{\partial z_{ik}} \cdot \frac{\partial z_{ik}}{\partial w_{kj}}$$

$$= -\left(y_i - f_\theta(x_i)\right) \cdot \beta_k \cdot g'(z_{ik}) \cdot x_{ij}.$$

Tricks of the Trade

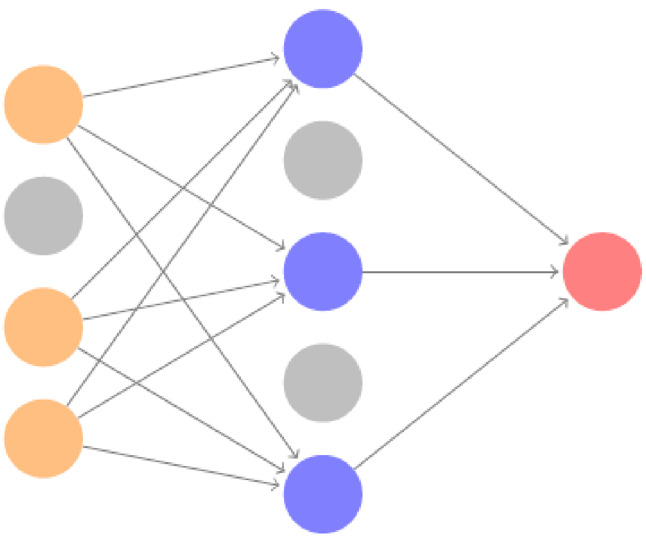
✓Slow learning. Gradient descent is slow, and a small learning rate ρ slows it even further. With early stopping, this is a form of regularization.

✓Stochastic gradient descent. Rather than compute the gradient using all the data, use a small minibatch drawn at random at each step. E.g. for MNIST data, with $n = 60K$, we use minibatches of 128 observations.

✓An epoch is a count of iterations and amounts to the number of minibatch updates such that n samples in total have been processed; i.e. $60K/128 \approx 469$ for MNIST.

✓Regularization. Ridge and lasso regularization can be used to shrink the weights at each layer. Two other popular forms of regularization are dropout and augmentation, discussed next.

Dropout Regularization



✓At each SGD update, randomly remove units with probability ϕ, and scale up the weights of those retained by $1/(1-\phi)$ to compensate.

✓In simple scenarios like linear regression, a version of this process can be shown to be equivalent to ridge regularization.

✓As in ridge, the other units stand in for those temporarily removed, and their weights are drawn closer together.

✓Similar to randomly omitting variables when growing trees in random forests.

Practice