Aprendizaje supervisado

Maestría en Estadística

Pablo Vena

Universidad de Buenos Aires

26 de noviembre de 2021

Lecturas

ISLR Capítulo 8.

ESL Capítulo 11.

Redes neuronales de una capa

Una red neuronal toma un vector

$$X = (X_1, X_2, \ldots, X_p)$$

y construye una función no lineal $f(X)$ para predecir la respuesta Y.

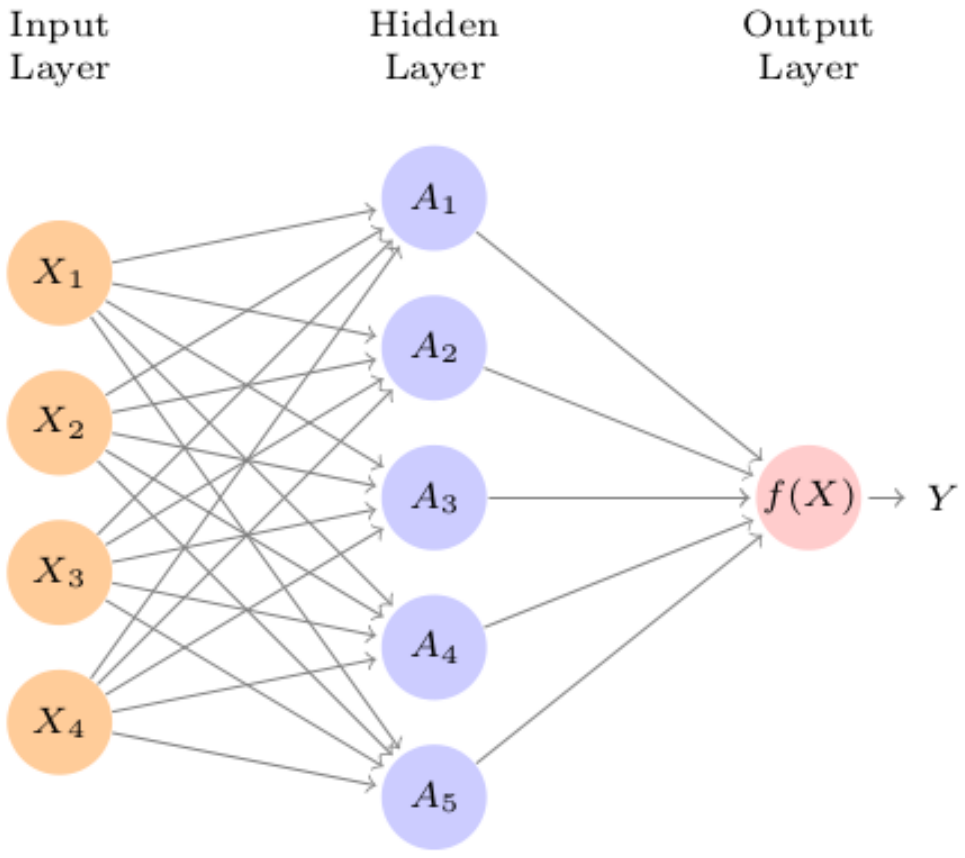
Otros métodos no lineales

- Árboles,
- Boosting,
- GAM (modelos aditivos generalizados)

Las redes se diferencian por la **estructura** particular del modelo.

Redes neuronales de una capa

La figura muestra una **feed-forward neural network**.



- Modela una variable cuantitativa mediantes $p = 4$ predictores.
- Las *features* $X_1, \ldots, X_4$ forman la **input layer**.
- Las flechas indican que cada covariable alimenta a cada una de las $K = 5$ **hidden units**.

Redes neuronales de una capa

El modelo de la red neuronal tiene la forma

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

y se construye en dos pasos:

1. se calculan las **activaciones** a partir de las *features*,
2. las activaciones alimentan la capa de salida (*output layer*).

Redes neuronales de una capa

Las K **activaciones** A_k en la capa oculta se calculan como funciones de las *input features* $X_1, \ldots, X_p$

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

donde $g(z)$ es una **función de activación no lineal** especificada.

Podemos pensar cada A_k como una transformación $h_k(X)$ de las covariables originales.

Redes neuronales de una capa

Los resultados de las K **activaciones** A_k contribuyen a la capa de salida

$$f(X) = \beta_0 + \sum_{k=1}^{K} \beta_k A_k$$

que resulta una regresión lineal en las $K = 5$ activaciones.

Red es neuronales de una capa

Los parámetros

$$\beta_0, \ldots, \beta_k$$

y

$$w_{10}, \ldots, w_{1p}, \ldots, w_{K0}, \ldots, w_{Kp}$$

se estiman a partir de los datos.

Buscamos los parámetros (pesos) que minimicen el error cuadrático medio

$$\sum_{i=1}^{n} (y_i - f(x_i))^2$$

Funciones de activación

Función sigmoidea

En los comienzos de las redes neuronales, la función de activación era la función *sigmoidea*

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

que es la misma que usamos para regresión logística para convertir una función lineal en probabilidades entre 0 y 1.

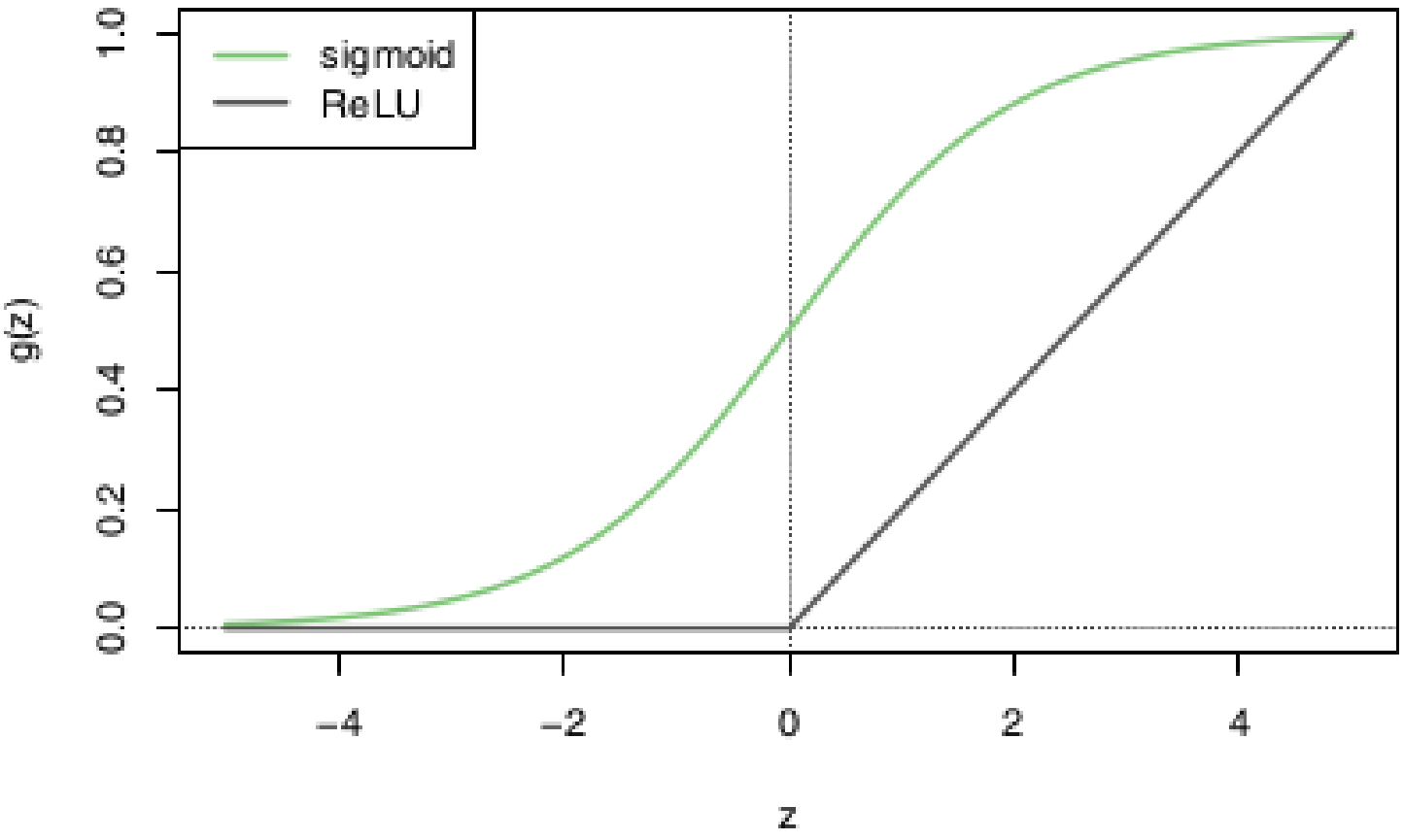
Función ReLU (*rectified linear unit*)

Una alternativa es la función de activación *ReLU*

$$g(z) = (z)_+ = \begin{cases} 0 & z < 0 \\ z & \text{en caso contrario} \end{cases}$$

Es eficiente en términos computacionales.

Funciones de activación



Funciones de activación

- La denominación de *red neuronal* proviene de pensar a las *hidden units* en analogía a las neuronas, que pueden activarse o no en función de los estímulos recibidos.
- Si la función de activación fuera lineal, toda la red se reduce a una regresión lineal.
- La falta de linealidad permite que el modelo capture relaciones complejas y efectos de interacción.

Redes neuronales multicapa

- Las redes neuronales modernas tienen usualmente
  - más de una capa oculta y
  - varias unidades por capa.
- Teóricamente, una sola capa con suficientes unidades tiene la capacidad de aproximar cualquier función.
- Sin embargo, en la práctica resulta más fácil emplear varias capas de tamaños no tan grandes.

Redes neuronales multicapa

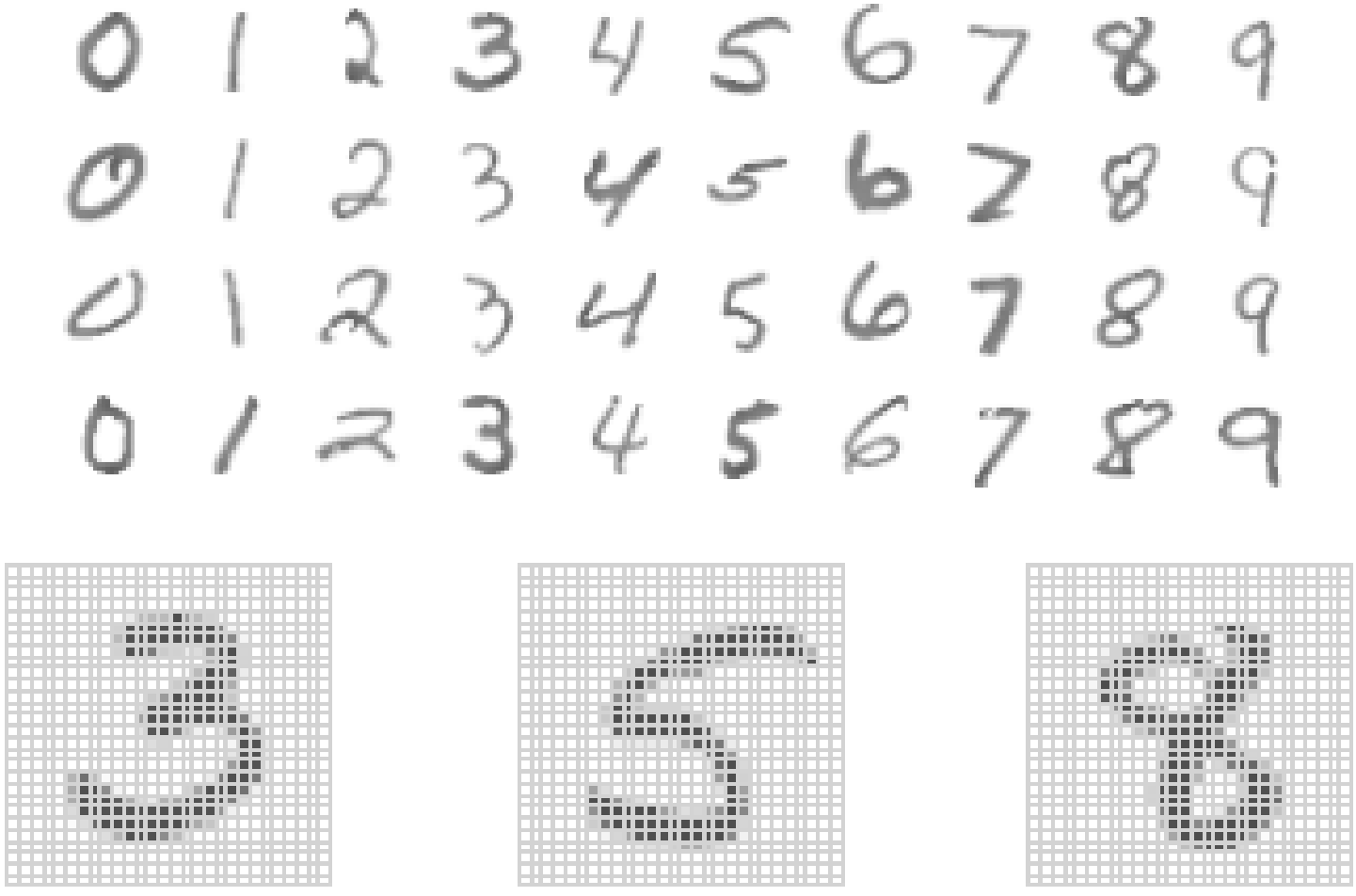


Figura: MNIST.

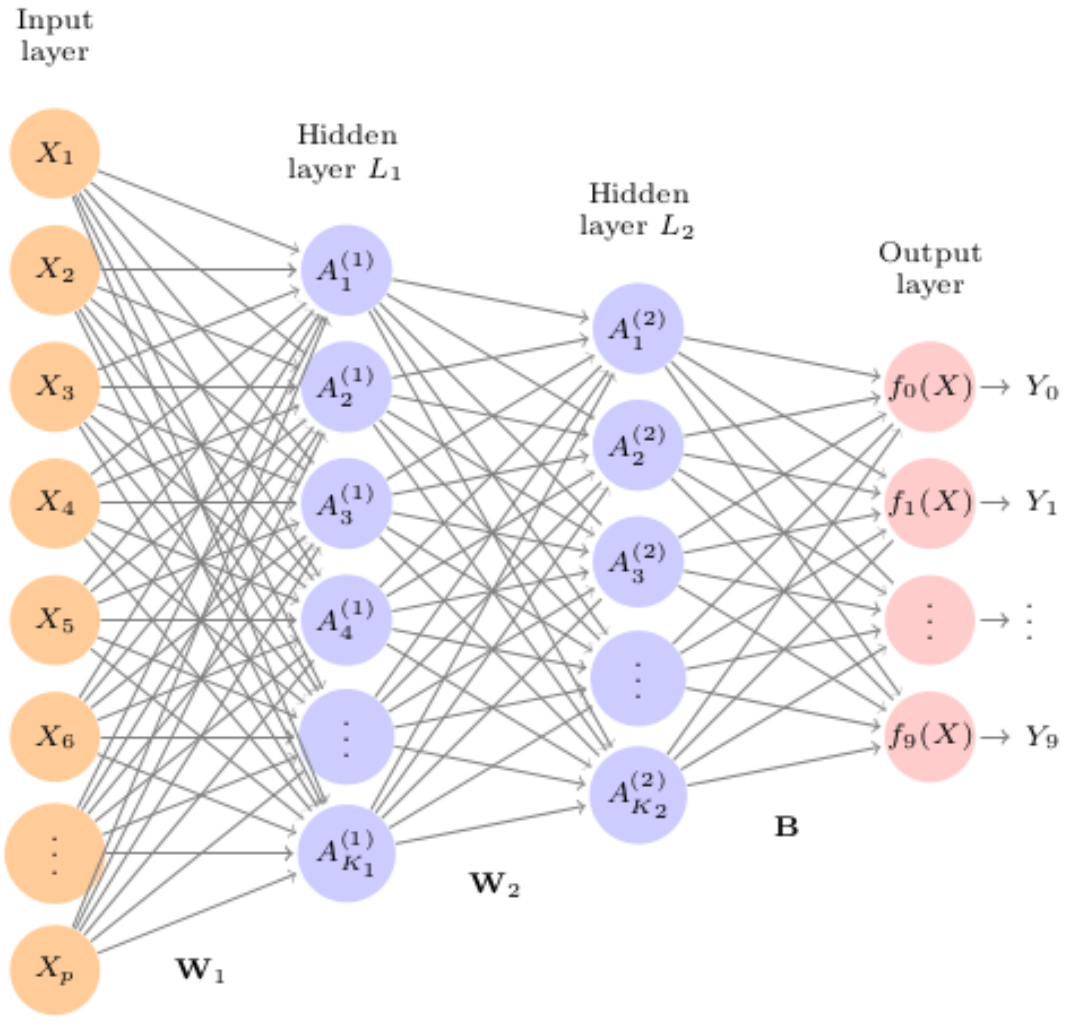
Redes neuronales multicapa

- El objetivo es construir un modelo para clasificar las imágenes a su dígito correspondiente.
- Cada imagen tiene $p = 28 \times 28 = 784$ pixeles en una escala de grises de 0 a 255 valores (8 bits).
- La respuesta es la clase correspondiente, representada por el vector

$$Y = (Y_0, Y_1, \ldots, Y_9)$$

  de 10 variables *dummy* codificadas como **one-hot encoding**.
- El conjunto de datos tiene 60000 imágenes de entrenamiento y 10000 de testeo.

Redes neuronales multicapa



La figura muestra una arquitectura multicapa con

- 2 capas ocultas

  L_1 (256 unidades) y
  L_2 (128 unidades)

- 10 variables de salida (dependientes entre sí)

Redes neuronales multicapa

La primera capa tiene K_1 activaciones

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

que son las entradas de la siguiente capa para calcular las próximas activaciones

$$A_j^{(2)} = h_j^{(2)}(X) = g\left(w_{j0}^{(2)} + \sum_{k=1}^{K_1} w_{jk}^{(2)} A_k^{(1)}\right)$$

Los pesos de cada capa pueden juntarse en una matriz W_i.

- Para L_1 la matriz tiene $(784 + 1) \times 256 = 200960$ elementos.
- La matriz correspondiente W_2 tiene $(256 + 1) \times 128 = 32896$.

Redes neuronales multicapa

En la capa de salida hay 10 respuestas:

$$Z_m = \beta_{m0} + \sum_{\ell=1}^{K_2} \beta_{m\ell} h_\ell^{(2)}(X) = \beta_{m0} + \sum_{\ell=1}^{K_2} \beta_{m\ell} A_\ell^{(2)}$$

Son 10 modelos lineales. La matriz B tiene dimensión $129 \times 10 = 1290$.

Queremos que $f_m(X) = P(Y = m|X)$, usamos la función de activación *softmax*

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

- Esto asegura que las 10 salidas se comporten como probabilidades (no negativas y suman uno).
- El modelo (como en regresión logística) estima las probabilidades y luego clasifica a la clase de mayor probabilidad.

Entrenamiento

Cross-entropy

Para entrenar la red, como la respuesta es cualitativa, buscamos aquellos coeficientes que minimizan la **negative multinomial log-likelihood**

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

Cantida de parámetros

- Red multicapa: 235146.
- regresión multinomial logística: $785 \times 9 = 7065$.

Overfitting

Para evitar el sobreajuste se emplean técnicas de **regularización**.

Entrenamiento

Dadas n observaciones (x_i, y_i) queremos resolver el siguiente problema

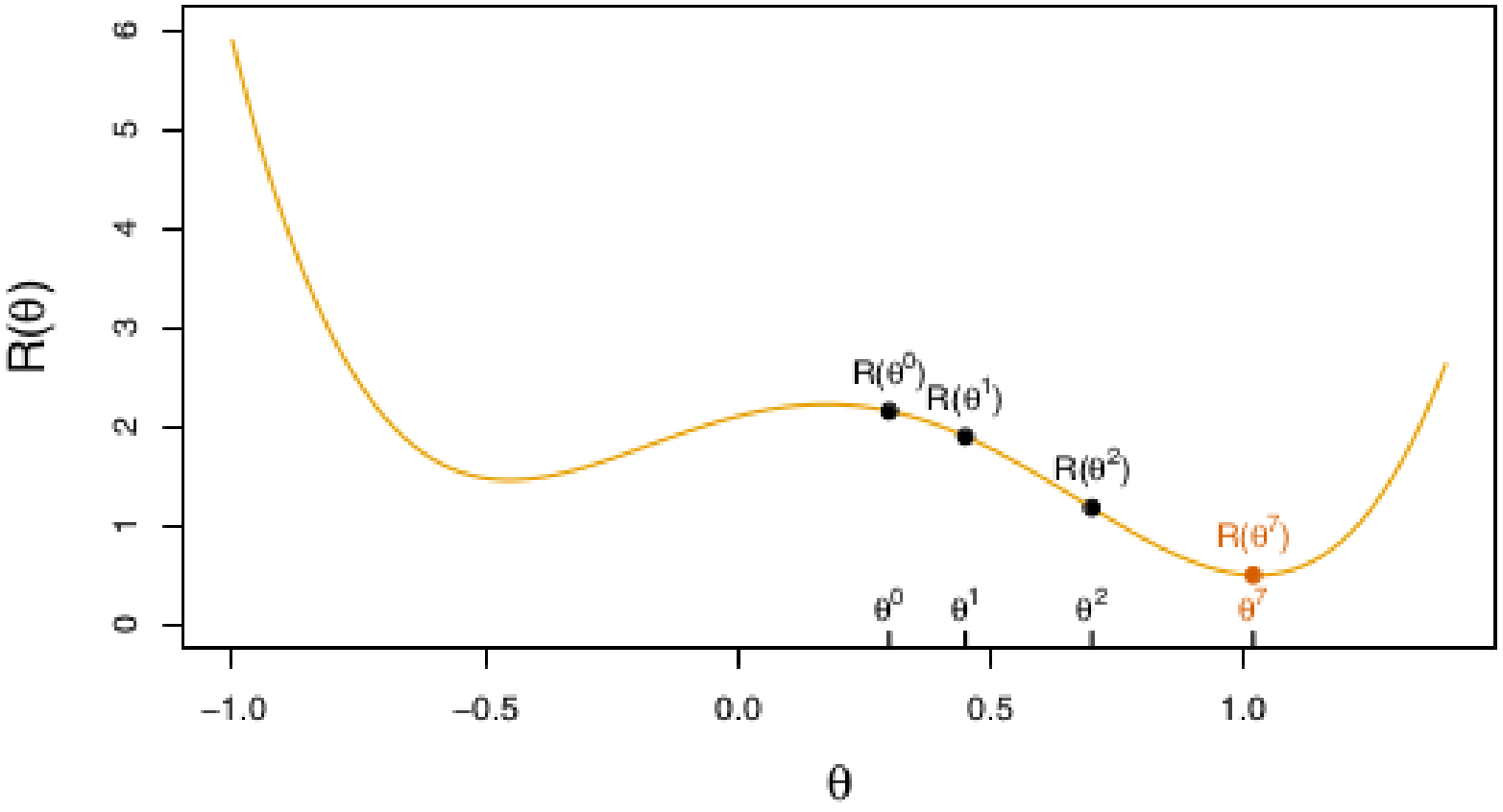
$$\min_{\{w_k\}_1^K, \beta} \frac{1}{2} \sum_{i=1}^{n} (y_i - f(x_i))^2$$

donde

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

- Es un problema de cuadrados mínimos no lineal.
- No es convexo en los parámetros, hay múltiples soluciones.

Entrenamiento



Entrenamiento

Si llamamos θ al vector de parámetros, el problema consiste en minimizar:

$$R(\theta) = \frac{1}{2}\sum_{i=1}^{n}(y_i - f_\theta(x_i))^2$$

La idea del **descenso por gradiente** es

1. Inicializamos los parámetros θ^0, $t = 0$.
2. Iteramos hasta que la función objetivo no decrezca:
    1. Buscamos un vector δ tal que

    $$\theta^{t+1} = \theta^t + \delta$$

    achica la función objetivo:

    $$R(\theta^{t+1}) \leq R(\theta^t)$$

    2. $t \leftarrow t + 1$

Backpropagation

Para hallar los pasos δ, las direcciones de decreciemiento, miramos el gradiente de $R(\theta)$ evaluado en el valor actual de $\theta = \theta^m$:

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

Es la dirección de máximo crecimiento en el espacio de θ:

$$\theta^{m+1} \leftarrow \theta^m - \rho \nabla R(\theta^m) .$$

Para un valor suficientemente chico de la **tasa de aprendizaje** ρ, este paso reduce la función objetivo

$$R(\theta^{m+1}) \leq R(\theta^m) .$$

Por la estructura de composiciones de funciones usamos la regla de la cadena para actualizar los pesos.

Backpropagation

Para simplificar la notación llamamos

$$z_{ik} = w_{k0} + \sum_{j=1}^{p} w_{kj} x_{ij}$$

Derivamos con respecto a β_k:

$$\frac{\partial R_i(\theta)}{\partial \beta_k} = \frac{\partial R_i(\theta)}{\partial f_\theta(x_i)} \cdot \frac{\partial f_\theta(x_i)}{\partial \beta_k} = -(y_i - f_\theta(x_i)) \cdot g(z_{ik})$$

Derivamos con respecto a w_{kj}:

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

En cada caso, se asigna una fracción del residuo a cada parámetro.

Descenso estocástico

- En lugar de utilizar las n observaciones, podemos elegir al azar un subconjunto menor, (**minibatch**), para calcular el gradiente.
- Este procedimiento se conoce como **stochastic gradient descent** (SGD).

Early stopping

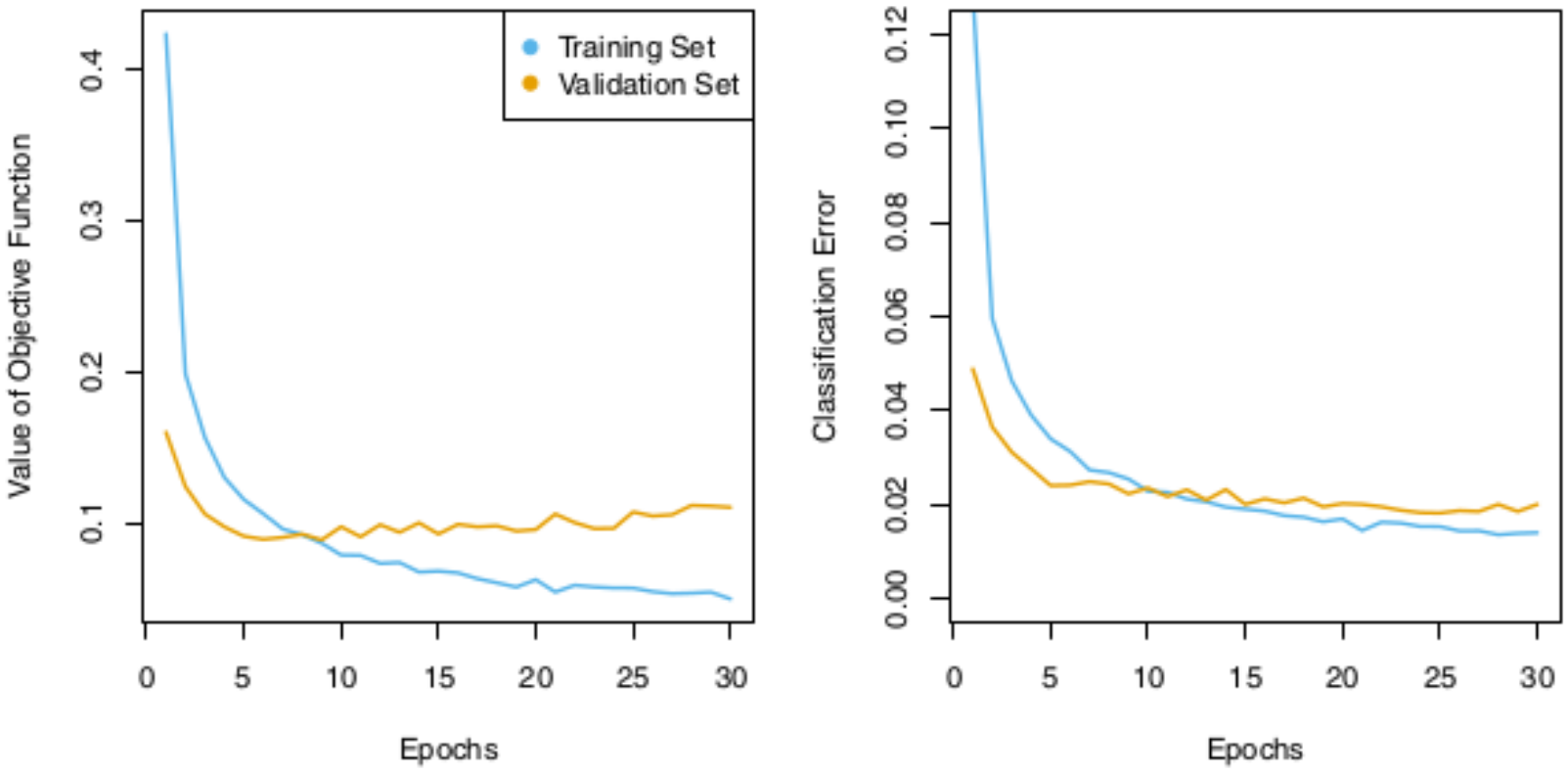


Figura: Early stopping.

Regularización

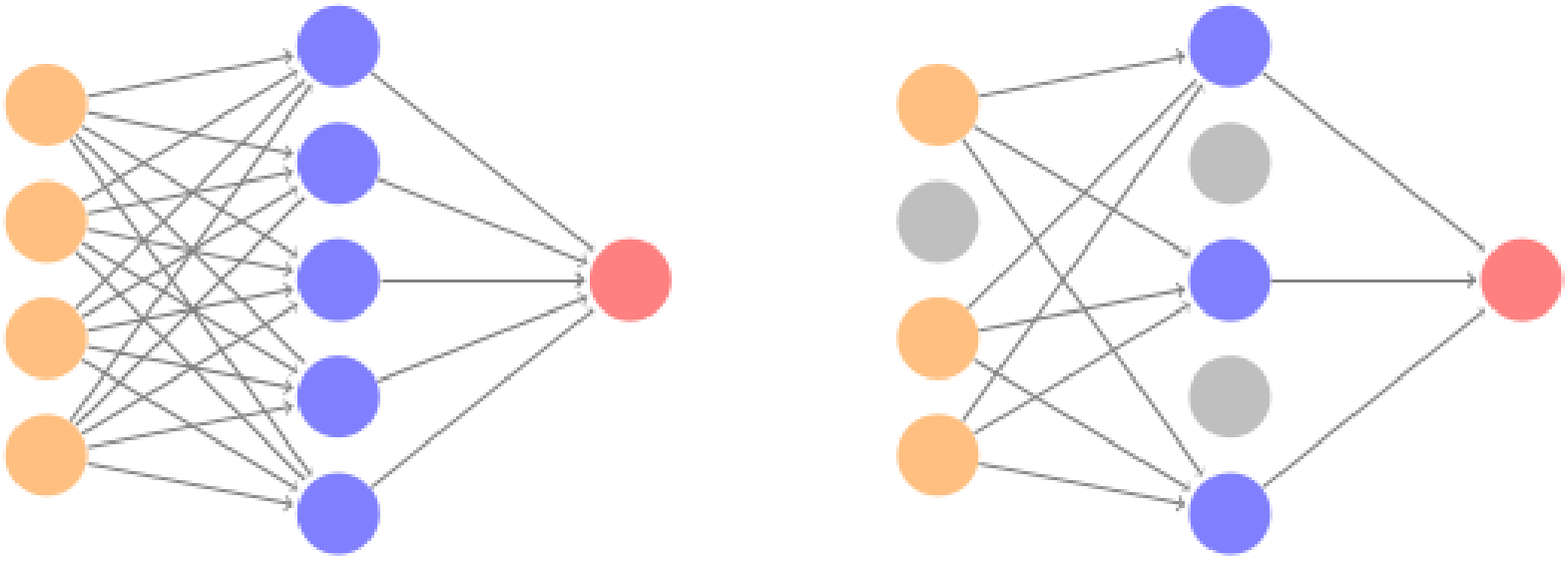
En el ejemplo de la red multicapa para reconocer dígitos, teníamos cuatro veces más parámetros que observaciones. Podemos usar, por ejemplo, regularización *ridge*:

$$R(\theta;\lambda) = -\sum_{i=1}^{n}\sum_{m=0}^{9} y_{im}\log(f_m(x_i)) + \lambda\sum_j \theta_j^2$$

- El parámetro λ generalmente toma un valor chico o puede elegirse con un conjunto de validación.
- Se puede usar un valor distinto de λ para cada capa.
- También se puede usar la penalización ℓ_1.

Dropout

Un mecanismo alternativo de regularización consiste en remover al azar una fracción φ de unidades en cada paso de optimización.



- Los pesos de las unidades que permanecen son escalados por un factor $1/(1-\varphi)$ para compensar.
- Previene que los nodos se sobre-especialicen
- En la práctica se fijan las activaciones de las unidades descartadas como 0.

Hiperparámetros

- Escala de los inputs
- El número de capas ocultas y de unidades por capa.
- Hiperparámetros de ajuste de la regularización y del *dropout*.
- Detalles del descenso de gradiente estocástico. Incluyen el tamaño del *batch* y el número de épocas.

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%

Deep learning

Para el conjunto de datos `Hitters` se ajustaron tres modelos:

- Modelo lineal
- Modelo lineal con regularización lasso (λ elegido por 10-CV)
- Red neuronal con una capa de 64 unidades *ReLU*.

Model	# Parameters	Mean Abs. Error	Test Set R^2
Linear Regression	20	254.7	0.56
Lasso	12	252.3	0.51
Neural Network	1409	257.4	0.54