Tecnologías para Inteligencia Artificial (247101009)
Tema 6. Redes Neuronales Artificiales

Javier Vales Alonso
Máster Universitario en Ingeniería Telemática

2020

Universidad Politécnica de Cartagena

Tabla de contenido

¿Cómo estudiar esta unidad?

1. Haga una primera lectura de la unidad. Concéntrese en ver las ideas generales y hacer una primera revisión de las matemáticas.
2. Haga una revisión a fondo de las matemáticas y resuelva en el notebook los ejercicios indicados. Intente comprender todos los desarrollos involucrados. En caso de dudas, lea las referencias sugeridas (ver referencias en Tema 0) o contacte con el profesor.
3. Finalmente, envíe el notebook a través de AV.

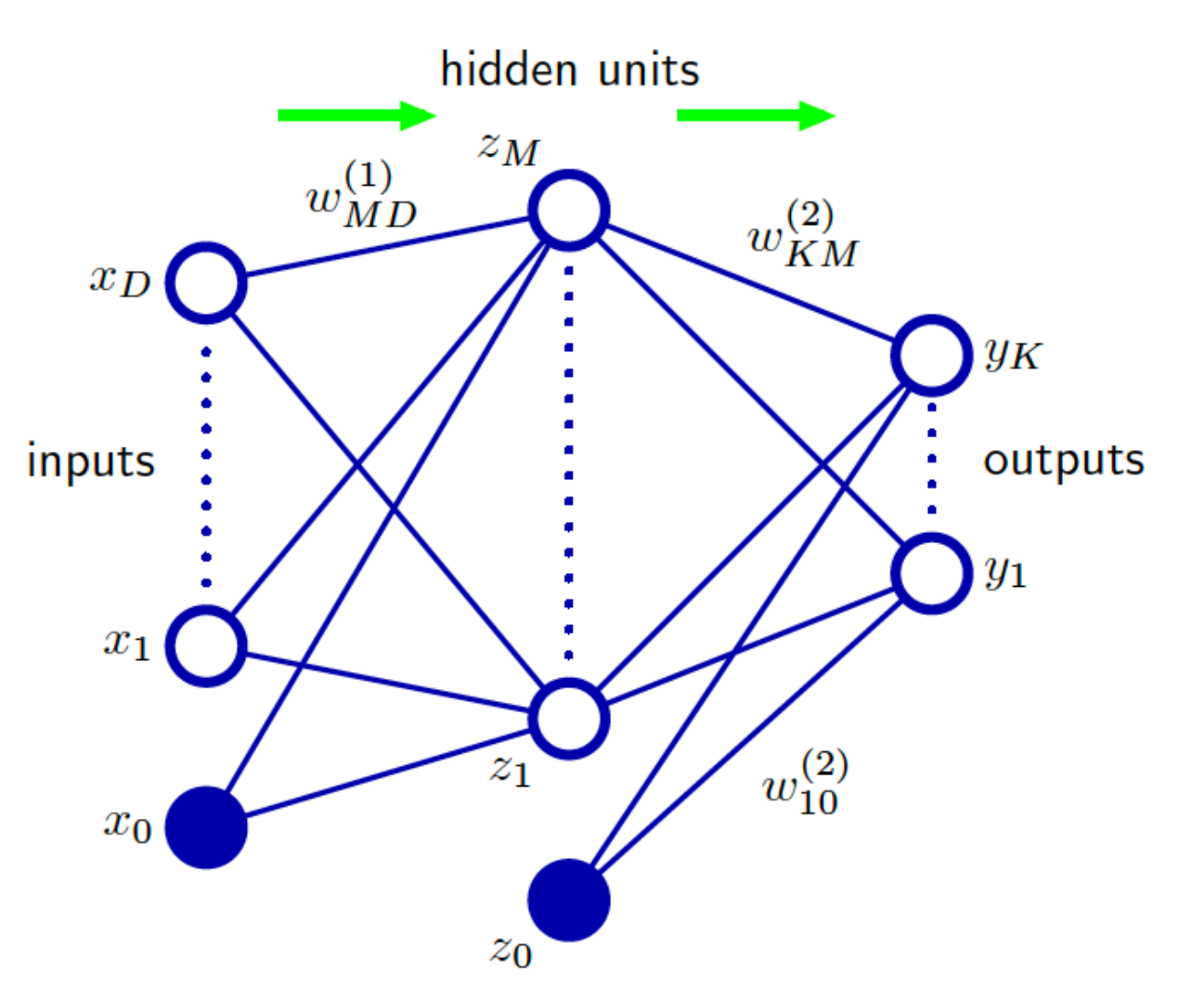
Introducción

Introducción

Una red neuronal artificial (*Artificial Neural Network - ANN*) es una estructura que permite crear modelos de predicción **no lineales**. Se emplean tanto en aprendizaje supervisado como no supervisado. En aprendizaje supervisado permiten implementar sistemas regresores (incluidos multi-target), y clasificadores probabilísticos (binarios o multiclase).

Al crear modelos no lineales permiten usar **bases con un número fijo de características** pero que serán **adaptativas**, i.e., que se ajustarán automáticamente a los datos de entrenamiento. Como penalización, resultan **funciones de coste no convexas**, por lo que hay que desarrollar algoritmos de entrenamiento específicos, como **back-propagation**.

Estructura básica de una ANN



Estructura básica de una ANN (II)

La red posee una **capa de entrada**, con D entradas, una por cada variable de la instancia $\boldsymbol{x}$. A continuación hay L capas, cada una con $M(l)$ nodos. Las $L-1$ primeras capas se llaman **capas ocultas**, tras ella está la **capa de salida**, con tantas salidas K como dimensiones tenga el *target* a predecir.

Cada una de los nodos de las capas se denomina **neurona**, por analogía con las redes neuronales biológicas. La salida del nodo i de la capa l se denota $z_i^{(l)}$, o simplemente x_i o y_i para las capas de entrada y de salida, respectivamente. Se asume además un nodo extra $x_0{=}1$, y $z_0^{(l)}{=}1$ para permitir un término de bias.

Las redes se identifican según el número de L-capas. Por ejemplo, la red de la transparencia anterior sería una red de 2 capas.

Estructura básica de una ANN (III)

De una capa a la siguiente, la salida del nodo i se pondera por un peso $w_{ji}^{(l)}$, siendo l la capa y j el nodo de la capa l.

La suma ponderada por los pesos de las entradas a un nodo se llama **activación**. Para el nodo j de la capa l viene dado por:

$$a_j^{(l)} = \sum_{i=0}^{M(l-1)} w_{ji}^{(l)} z_i^{(l-1)}$$

Esa entrada se transforma usando una **función de activación** $h^{(l)}(\cdot)$ para generar la salida hacia la siguiente capa:

$$z_j^{(l)} = h^{(l)}(a_j^{(l)}) = h^{(l)}\left(\sum_{i=0}^{M(l-1)} w_{ji}^{(l)} z_i^{(l-1)}\right)$$

La función de activación dependerá de la capa (o nodo) y se escoge **no-lineal y diferenciable**.

Estructura básica de una ANN (IV)

La **función de activación de la capa de salida** debe escogerse de acuerdo al tipo de sistema predictivo:

- Para un regresor cada función de activación en la capa de salida es la identidad, por tanto, $y_k = a_k$
- Para un clasificador binario múltiple (e.g., mail SPAM/NO SPAM, FORMAL/NO FORMAL), la función de activación es alguna cuyo valor pueda ser interpretado como una probabilidad, por ejemplo, la sigmoide. Por tanto, $y_k = \sigma(a_k)$.
- Para un clasificador multi-clase cada salida es la probabilidad de pertenecer a una clase. Una función adecuada será la *softmax*.

Estructura básica de una ANN (V)

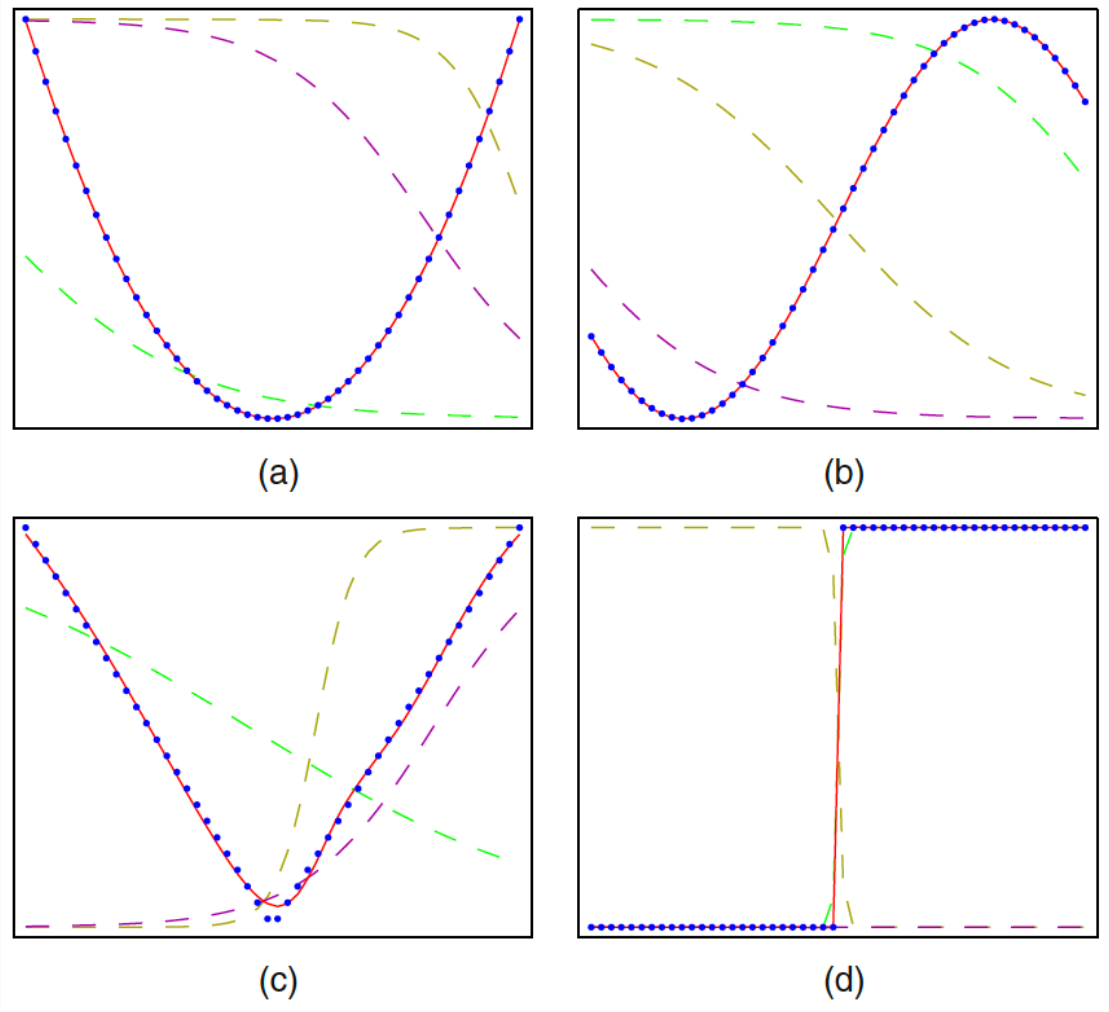
Para el ejemplo de la página 5, asumiendo una salida dada por activaciones logísticas, tendríamos:

$$y_k(\boldsymbol{x}, \boldsymbol{w}) = \sigma\left(\sum_{j=0}^{M} w_{kj}^{(2)} h\left(\sum_{i=0}^{D} w_{ji}^{(1)} x_i\right)\right)$$

Nótese que la salida es no lineal al no serlo $h(\cdot)$. En caso de usar una $h(\cdot)$ lineal el modelo será equivalente a un regresor logístico, al ser lineal la composición de funciones lineales.

Por otra parte, la complejidad del modelo vendrá expresada en función de M, el número de nodos de la capa oculta, y deberá ser controlada por algún método como **regularización** o **early stopping**.

Estructura básica de una ANN (VI)



Ejemplo de capacidad predictiva de una ANN de 2 capas con 3 neuronas en la capa intermedia con función de activación $\tanh(\cdot)$ y función de activación lineal en las salidas. El conjunto de entrenamiento tiene N=50 puntos.

Entrenamiento de la red

Entrenamiento de la red

Dada un conjunto de datos de entrenamiento $\{\boldsymbol{x}_n\}$, $\{\boldsymbol{t}_n\}$, el objetivo del entrenamiento es encontrar un conjunto de pesos[1] $\boldsymbol{w}$ que minimicen el error asociado.

La función de error dependerá del tipo de sistema predictivo:

- Para un regresor de salida única (**joint-squared-error**):

$$J(\boldsymbol{w}) = \frac{1}{2}\sum_{n=1}^{N}[y(\boldsymbol{x}_n, \boldsymbol{w}) - t_n]^2$$

[1] En ANNs el vector de pesos óptimo no es único, sino que existirán varios, e.g., variando el orden de los nodos.

Entrenamiento de la red (II)

- Para un regresor multi-target (**joint-squared-error**):

$$J(\boldsymbol{w}) = \frac{1}{2}\sum_{n=1}^{N}\sum_{k=1}^{K}[y_k(\boldsymbol{x}_n, \boldsymbol{w}) - (\boldsymbol{t}_n)_k]^2 = \frac{1}{2}\sum_{n=1}^{N}\|\boldsymbol{y}(\boldsymbol{x}_n, \boldsymbol{w}) - \boldsymbol{t}_n\|^2$$

- Para un clasificador binario (**cross-entropy**):

$$J(\boldsymbol{w}) = -\sum_{n=1}^{N}[t_n \ln y_n + (1 - t_n)\ln(1 - y_n)]$$

Entrenamiento de la red (III)

- Para un clasificador binario múltiple (**cross-entropy**):

$$J(\boldsymbol{w}) = -\sum_{n=1}^{N}\sum_{k=1}^{K}[(\boldsymbol{t}_n)_k \ln y_k(\boldsymbol{x}_n, \boldsymbol{w}) + (1-(\boldsymbol{t}_n)_k)\ln(1-y_k(\boldsymbol{x}_n, \boldsymbol{w}))]$$

- Para un clasificador multiclase (**multi-class cross-entropy**):

$$J(\boldsymbol{w}) = -\sum_{n=1}^{N}\sum_{k=1}^{K}(\boldsymbol{t}_n)_k \ln y_k(\boldsymbol{x}_n, \boldsymbol{w})$$

Backpropagation

Backpropagation

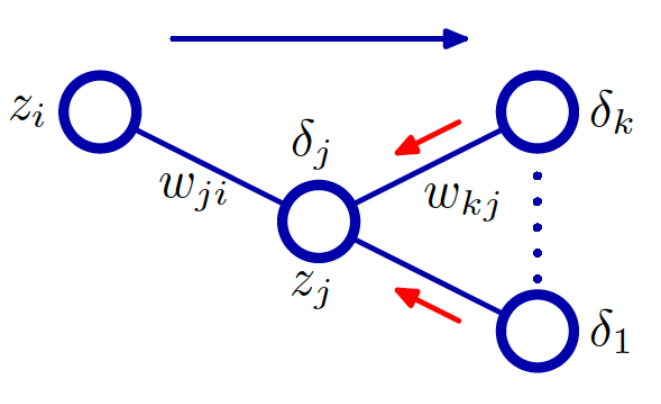
Las funciones de error $J(\boldsymbol{w})$ son **no-convexas** al ser las salidas funciones no lineales de los pesos.

Para su solución se recurre a métodos heurísticos, basados en el descenso del gradiente (hayan mínimos locales). La técnica base para calcular los gradientes se llama algoritmo de **backpropagation**.

Para explicarlo, observe en primer lugar que *todas* las funciones de error planteadas pueden descomponerse como:

$$J(\boldsymbol{w}) = \sum_{n=1}^{N} J_n(\boldsymbol{w})$$

Backpropagation (II)



$J_n(\boldsymbol{w})$ es el error asociado a la predicción de la muestra n. Consideraremos la obtención del gradiente de J_n (derivada respecto a cada peso w_{ji}). Aplicando la regla de la cadena:

$$\frac{\partial J_n}{\partial w_{ji}} = \frac{\partial J_n}{\partial a_j}\frac{\partial a_j}{\partial w_{ji}} = \delta_j z_i$$

Ya que $a_j = \sum_i w_{ji} z_i$, por lo que es $\dfrac{\partial a_j}{\partial w_{ji}} = z_i$. El término δ_j se denomina **error**, y debe calcularse para cada nodo de la red.

Backpropagation (III)

Para los nodos de salida tenemos:

$$\delta_k = \frac{\partial J_n}{\partial a_k} = y_k - t_k$$

Y para las unidades ocultas:

$$\delta_j = \frac{\partial J_n}{\partial a_j} = \sum_k \frac{\partial J_n}{\partial a_k} \frac{\partial a_k}{\partial a_j}$$

siendo k los nodos a los que j envía conexiones.

Backpropagation (III)

Teniendo en cuenta que $a_k = \sum_j w_{kj} z_j$ y $z_j = h(a_j)$, resulta:

$$\delta_j = h'(a_j) \sum_k w_{kj} \delta_k$$

Es decir, **el error para la capa l se calcula a partir de los errores de la capa $l+1$.**

Backpropagation (IV)

El algoritmo resultante para calcular J_n es el siguiente:

1. Se aplican las entradas $\boldsymbol{x}_n$ a la red y se **propagan hacia delante**, calculando las activaciones a_j.
2. Se calculan los errores de la capa de salida, δ_k.
3. Se **propagan hacia atrás** los errores calculando δ_j.
4. Finalmente se calculan las derivadas como $\dfrac{\partial J_n}{\partial w_{ji}} = \delta_j z_i$.

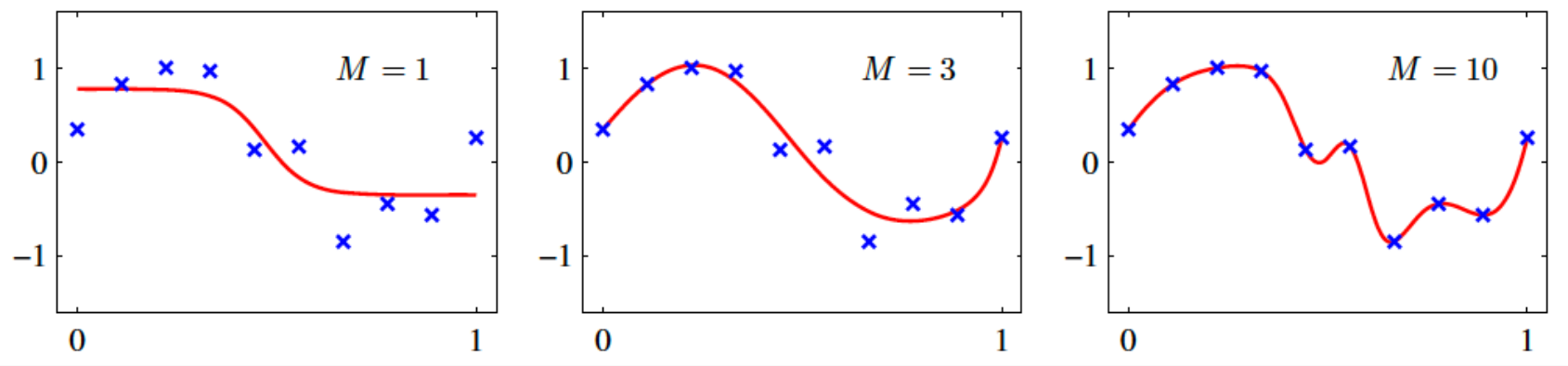
Backpropagation (V)

A partir de las derivadas se aplica descenso del gradiente (o algún derivado), mejorando progresivamente los pesos de la red. Según el modo de aplicación de las derivadas hay dos variantes principales:

- Métodos **batch**. Se usan simultáneamente todo el conjunto de entrenamiento. Para ello se calcula la derivada total, sumando la contribución de cada derivada asociada a cada dato del conjunto de entrenamiento.
- Métodos **online o secuenciales**. En cada iteración se usa sólo un dato del conjunto de entrenamiento. Este método suele ser el más empleado.

Control de la complejidad

Control de la complejidad



Mejor aproximación para una red neuronal de una capa oculta con M nodos entrenada con N=10 muestras. Para M=1 hay *under-fitting*, mientras que para M=10 se aprecia *over-fitting*.

Control de la complejidad (II)

Una aproximación para controlar la complejidad es modificar la función de coste añadiendo un regularizador ridge:

$$\widetilde{J}(\boldsymbol{w}) = J(\boldsymbol{w}) + \frac{\alpha}{2}\boldsymbol{w}^T\boldsymbol{w}$$

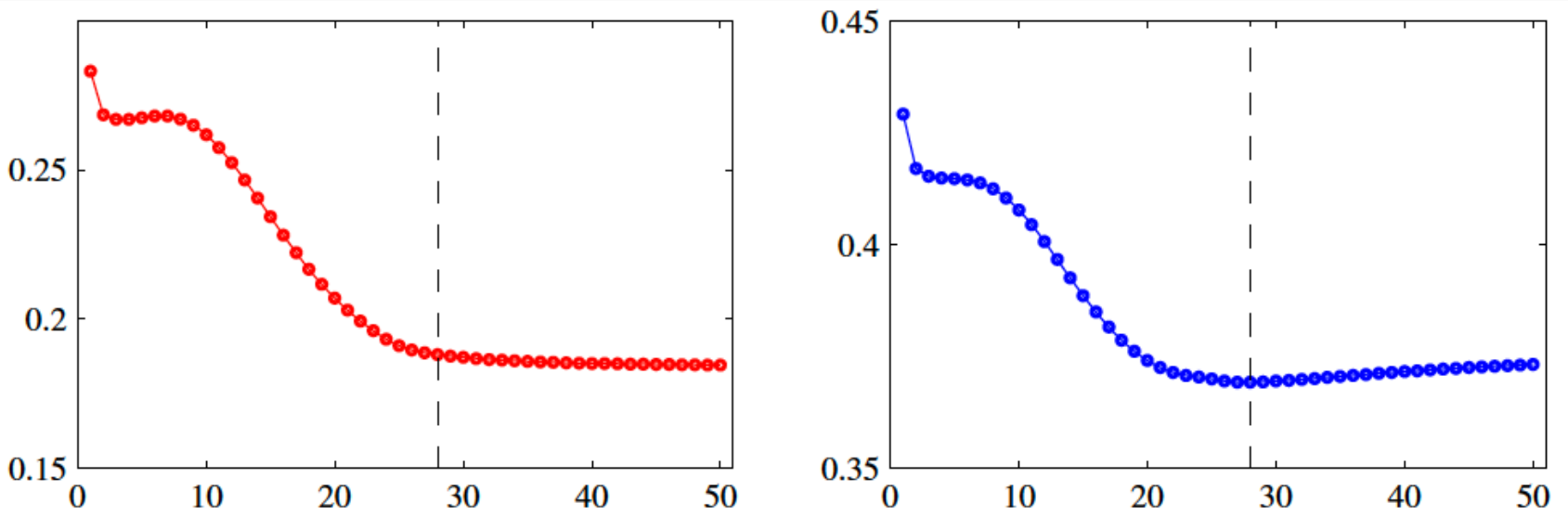
En el contexto de ANNs, se denomina *weight decay* a este tipo de regularización. No obstante, esta técnica suele presentar problemas debido a las propiedades de escalado de algunas funciones de la red.

Control de la complejidad (III)

Otra aproximación es emplear *early stopping*. En general, en los algoritmos de entrenamiento para ANN, el error es una función no-decreciente con la iteración. Sin embargo, **si el error se mide en un conjunto de validación independiente, a menudo disminuye al principio, y después aumenta** a medida que la red comienza a ajustarse demasiado.

Por lo tanto, **el entrenamiento puede detenerse en el punto de menor error con respecto al conjunto de datos de validación**, para obtener una red que tenga un buen rendimiento de generalización.

Control de la complejidad (IV)



Error sobre datos de entrenamiento (rojo), y sobre datos de validación (azul).