uc3m | Universidad Carlos III de Madrid

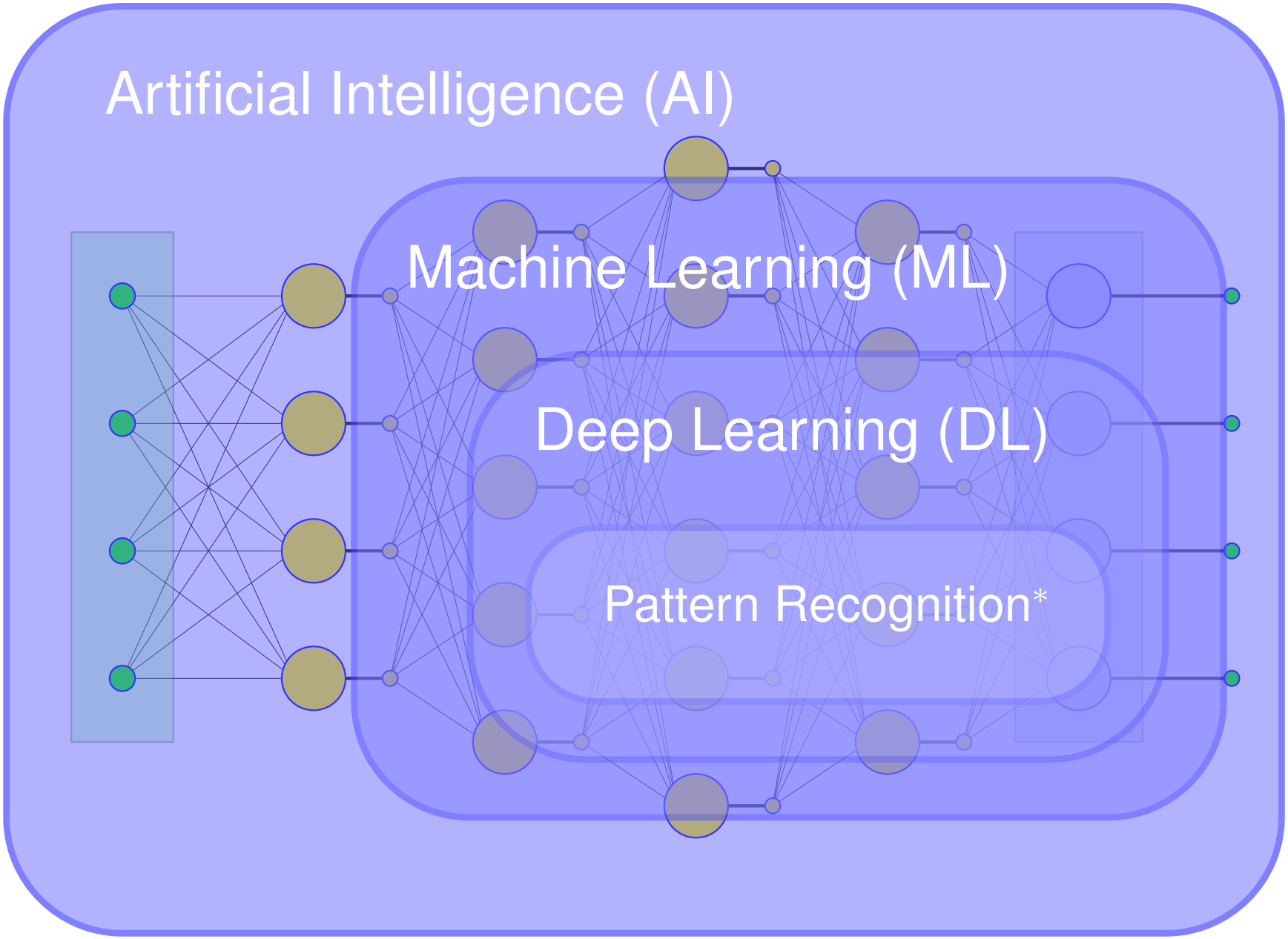
Máquinas Discriminativas Profundas (*Deep Learning*)

http://www.tsc.uc3m.es/~mlazaro/Docencia/DL.html

Marcelino Lázaro

Universidad Carlos III de Madrid

Contexto


Artificial Intelligence (AI)
Machine Learning (ML)
Deep Learning (DL)
Pattern Recognition*

Índice de contenidos

- Aprendizaje profundo (*"deep learning"*) para clasificación
  - El problema de clasificación (o reconocimiento) de patrones
  - Clasificación de patrones con redes neuronales
    - ★ Redes neuronales no profundas
    - ★ Redes neuronales profundas
- Diseño de una red de autocodificadores apilados (*SDA: Stacked Denoising Autoencoders*)
  - Entreno de autocodificadores
  - Uso de autocodificadores para la construcción de una red profunda
  - Similar: Redes de creencia profundas (*DBNs: Deep Belief Networks*)
- Redes profundas con *Drop-Out*
- Redes convolucionales (*CNNs: Convolutional Neural Networks*)

Aprendizaje profundo (para clasificación)

Problemas de clasificación

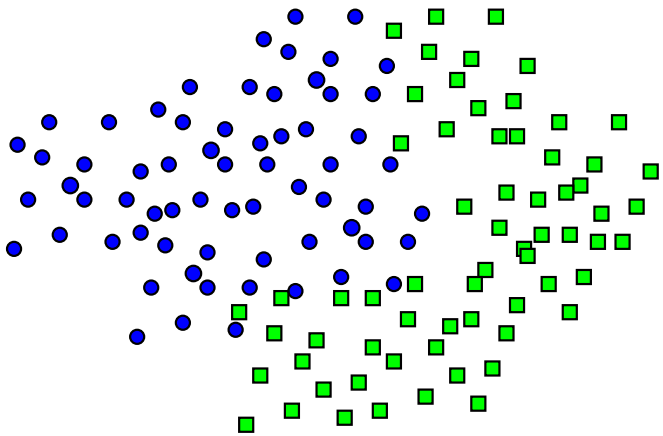
- Problema de clasificación genérico (reconocimiento de patrones)
  - Patrón $\mathbf{x} \in \boldsymbol{R}^D \rightarrow \mathbf{x} = [x_1, x_2, \cdots, x_D]^T$
  - Asignar $\mathbf{x}$ a una clase de entre un conjunto $\mathcal{C}$ (M-ário)

$$\mathcal{C} = \{C_1, C_2, \cdots, C_M\}$$

- Ejemplo: clasificación binaria

$$\mathcal{C} = \{0, 1\} \equiv \{-1, 1\} \equiv \{\text{null, alternative}\}$$

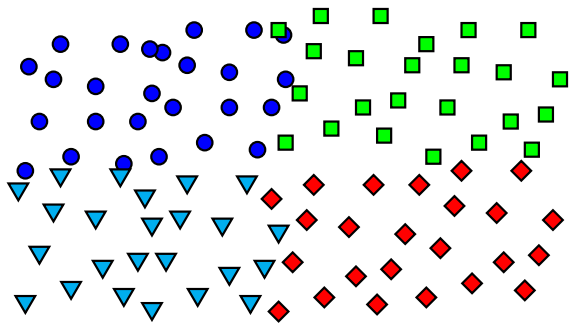
Ejemplo $D = 2$



- Ejemplo: clasificación 4-ária

$$\mathcal{C} = \{1, 2, 3, 4\}$$

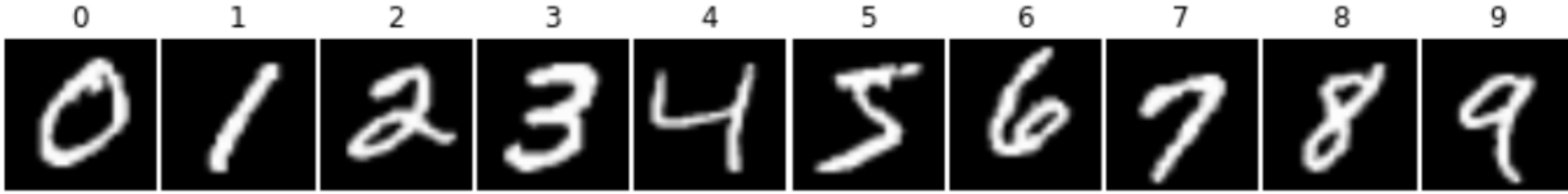
Ejemplo $D = 2$



- Problema M-ário se puede descomponer en varios binarios

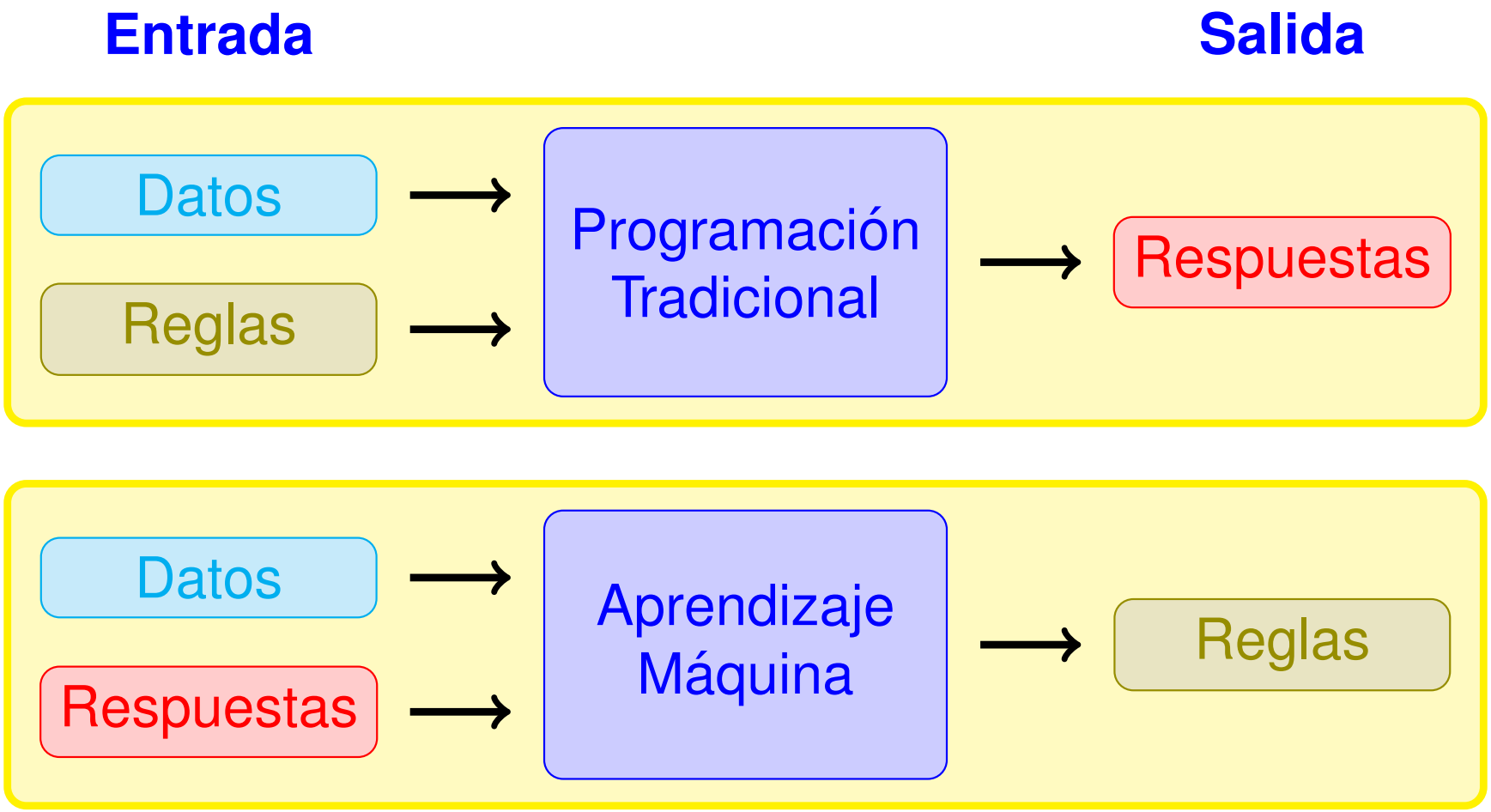
Patrones - Ejemplos

- Números manuscritos en imágenes (MNIST)



  - Patrón: imagen (dimensión D : número de píxeles)
  - Clases: posibles dígitos ($M = 10$)
- Predicción de fallo en el pago de prestamo hipotecario (HMEQ)
  - Patrón
    - ★ Cantidad del préstamo
    - ★ Valor de la propiedad hipotecada
    - ★ Ocupación (variable categórica)
    - ★ Tiempo (años) en el puesto de trabajo actual
    - ★ ...
  - Clases: Crédito devuelto, Crédito impagado ($M = 2$, binario)

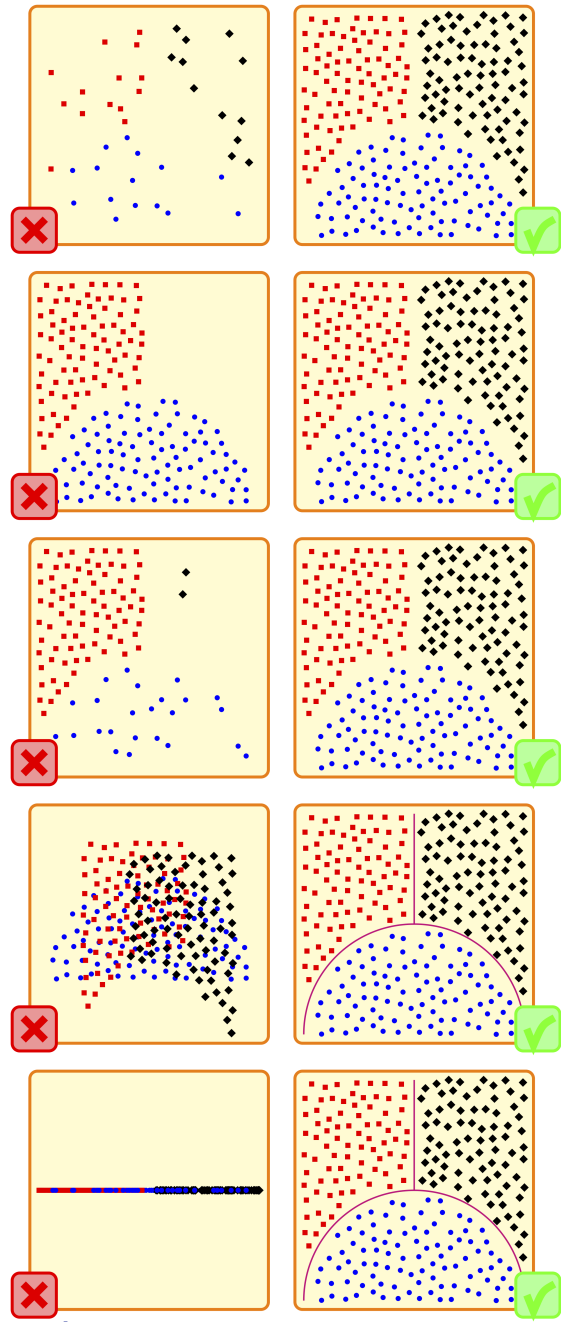
Aprendizaje máquina (supervisado) - Nuevo paradigma



- Respuestas: también denominadas **etiquetas** (*labels*, *target values*)
  - Problemas de Clasificación: indican la clase a la que pertenece cada patrón
- Reglas: también denominadas **modelos**

Calidad de los Datos

- Los datos de calidad son clave para el éxito de un método de aprendizaje máquina
  - Modelos de estado del arte pueden tener resultados mediocres sin buenos datos
  - Modelos sencillos pueden tener buenos resultados con buenos datos
- La calidad no es algo fácil de medir
  - Datos de calidad para una aplicación pueden no serlo para otra
- Características comunes
  - Precisión y consistencia
  - Cantidad suficiente
  - Representatividad
    - ★ Aunque existen técnicas de detección de novedad
  - Equilibrados (tanto como sea posible)
    - ★ Aunque existen técnicas para gestionar el desequilibrio
  - Discriminativos
    - ★ Distribuciones diferentes para las clases a distinguir - Cuidado con el preprocesado
  - Completitud
    - ★ Contienen todo lo necesario para resolver el problema



Clasificación mediante métodos de aprendizaje máquina

- Información disponible: conjunto de patrones (DATOS)
  - Conjunto de entrenamiento (N patrones)

$$\{(\mathbf{x}_k, y_k)\}_{k=1}^{N} \qquad \mathbf{x}_k \in \boldsymbol{R}^D \qquad y_k \in \mathcal{C} = \{C_1, C_2, \cdots, C_M\}$$

    - ★ Supervisado/no supervisado (¿etiquetas y_k disponibles?)
- Máquina entrenable (con unos parámetros $\mathbf{w}$)
  - Salida de la máquina

$$\mathbf{o} = g(\mathbf{x}, \mathbf{w})$$

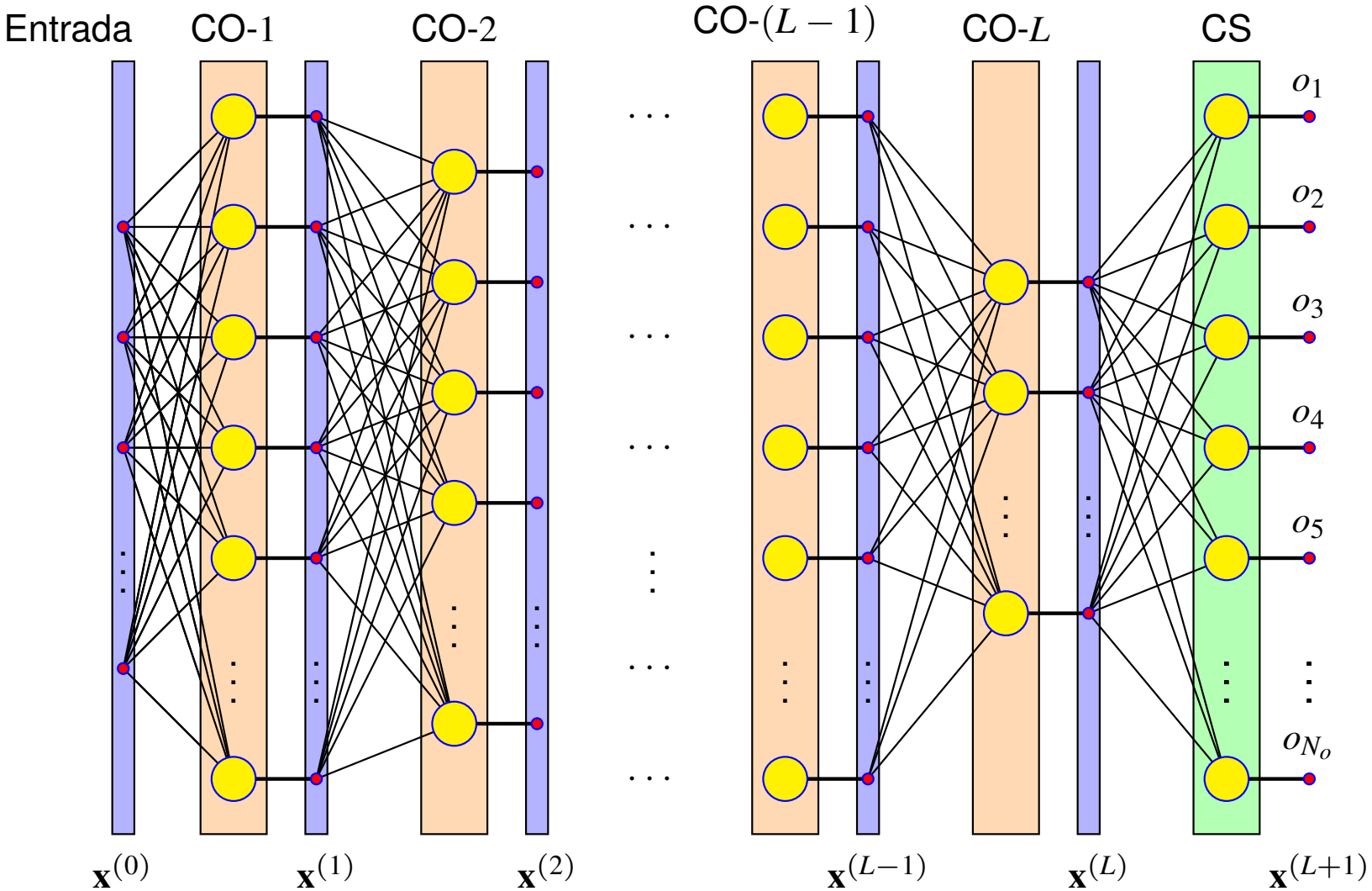
  - Regla de decisión

$$\hat{\mathbf{y}} = \text{decision}(\mathbf{o})$$

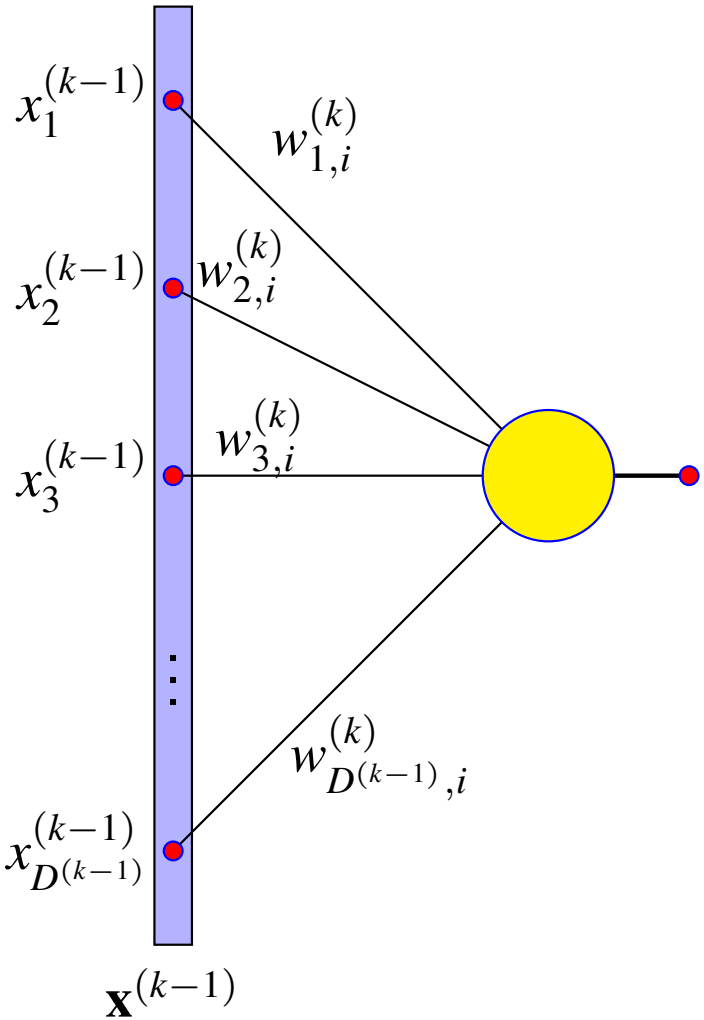
  - Aprendizaje a partir de los ejemplos (entrenamiento)
    - ★ Ajuste de los parámetros de la máquina $\mathbf{w}$
    - ★ Se ajustan para clasificar los ejemplos

Red Neuronal (red "feed-forward" con L capas ocultas)

- Una capa de entrada (capa 0) de la dimensión de los patrones $D^{(0)} = D$
- L capas ocultas con $D^{(k)}$ neuronas en la capa k (con $k \in \{1, 2, \cdots, L\}$)
- Una capa de salida (capa $L+1$) con $D^{(L+1)} = N_o$ neuronas (salidas)



Redes neuronales : Neurona i - Capa k



Entradas: $x_1^{(k-1)}, x_2^{(k-1)}, \cdots, x_{D^{(k-1)}}^{(k-1)}$

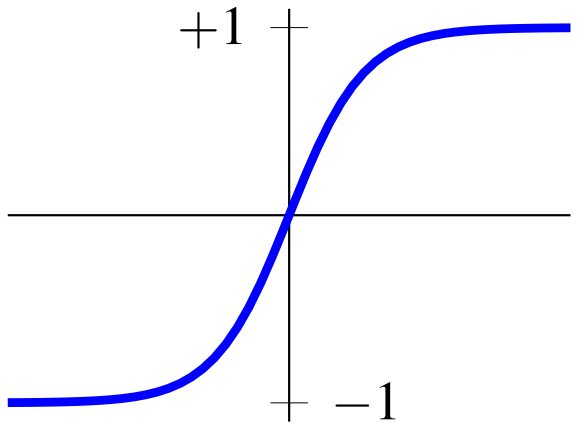
Salidas de las neuronas de la capa $k-1$

Término de sesgo (bias): $x_{D^{(k-1)}+1}^{(k-1)} \equiv 1$

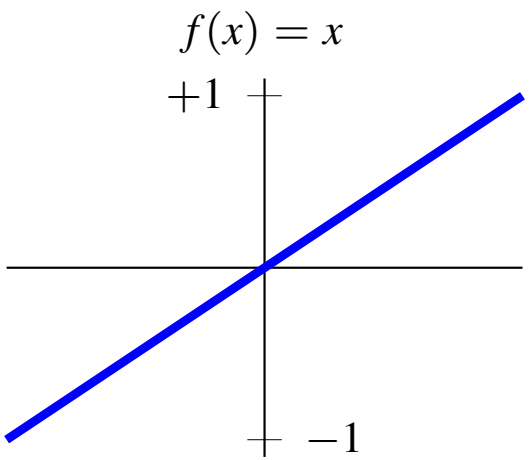
$$x_i^{(k)} = f\left(\sum_{j=1}^{D^{(k-1)}} w_{j,i}^{(k)}\, x_j^{(k-1)} + b_i^{(k)}\right), \quad b_i^{(k)} \equiv w_{D^{(k-1)}+1,i}^{(k)}$$

$f(\cdot) \equiv$ función de activación

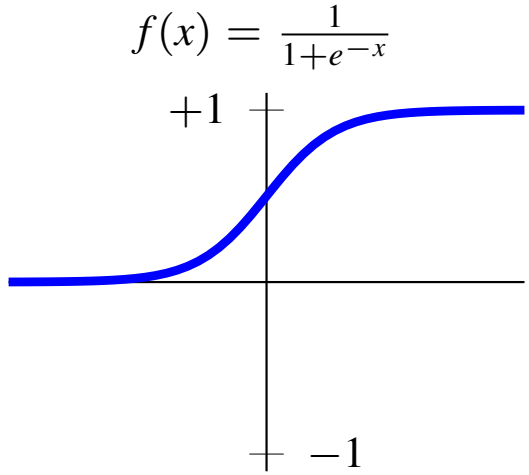
$$f(x) = \tanh(x)$$



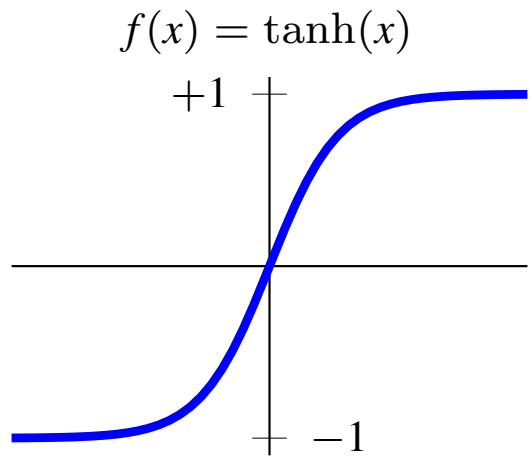
Funciones de activación más frecuentes (tradicionales)



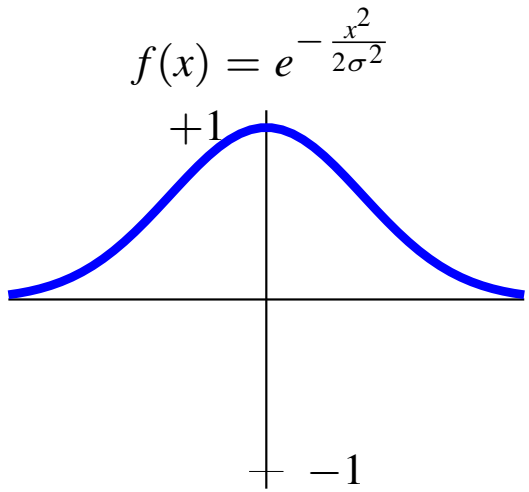
Lineal (Identidad)



Logística (sigmoide)



Tangente Hiperbólica

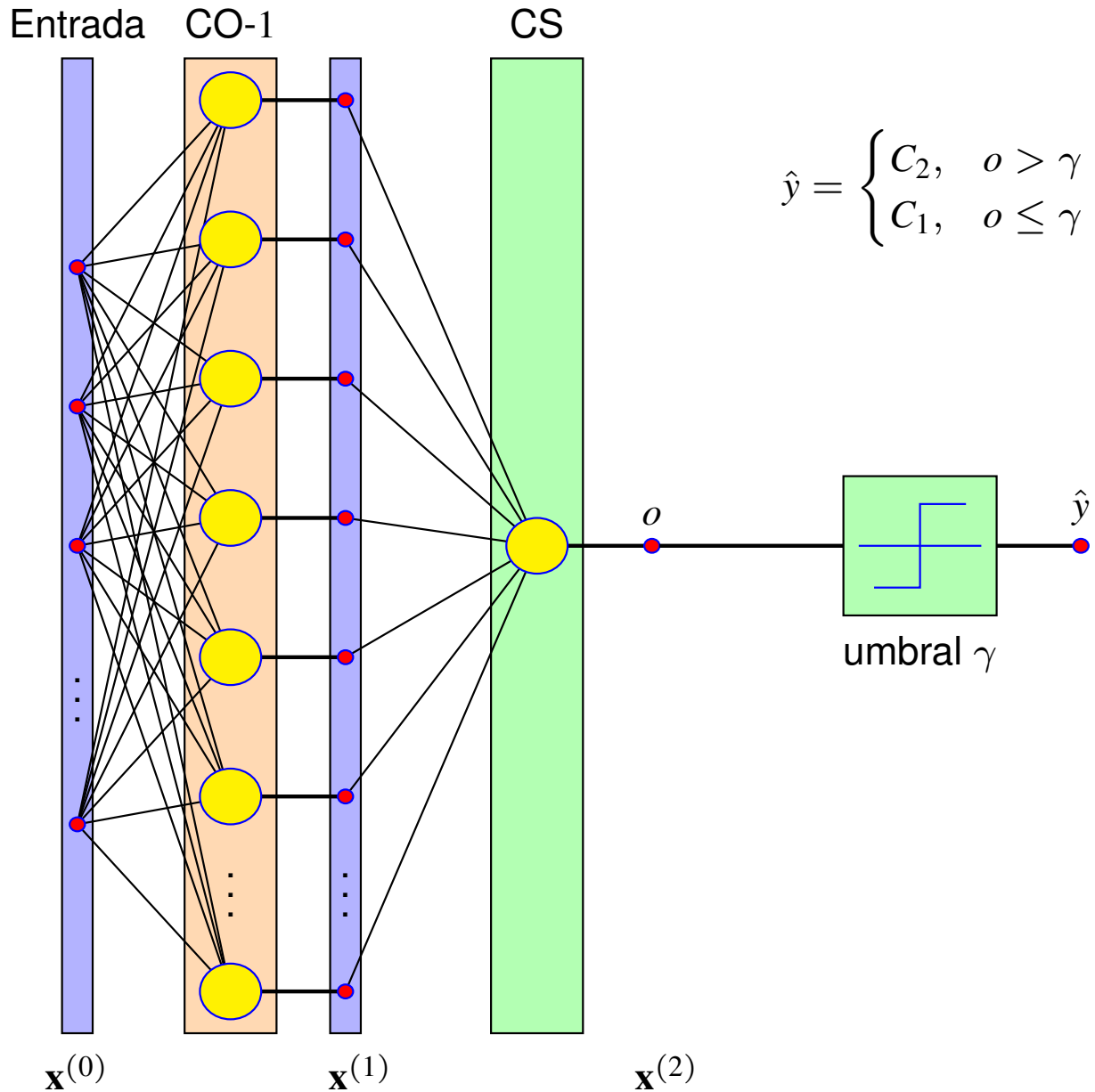


Gausiana

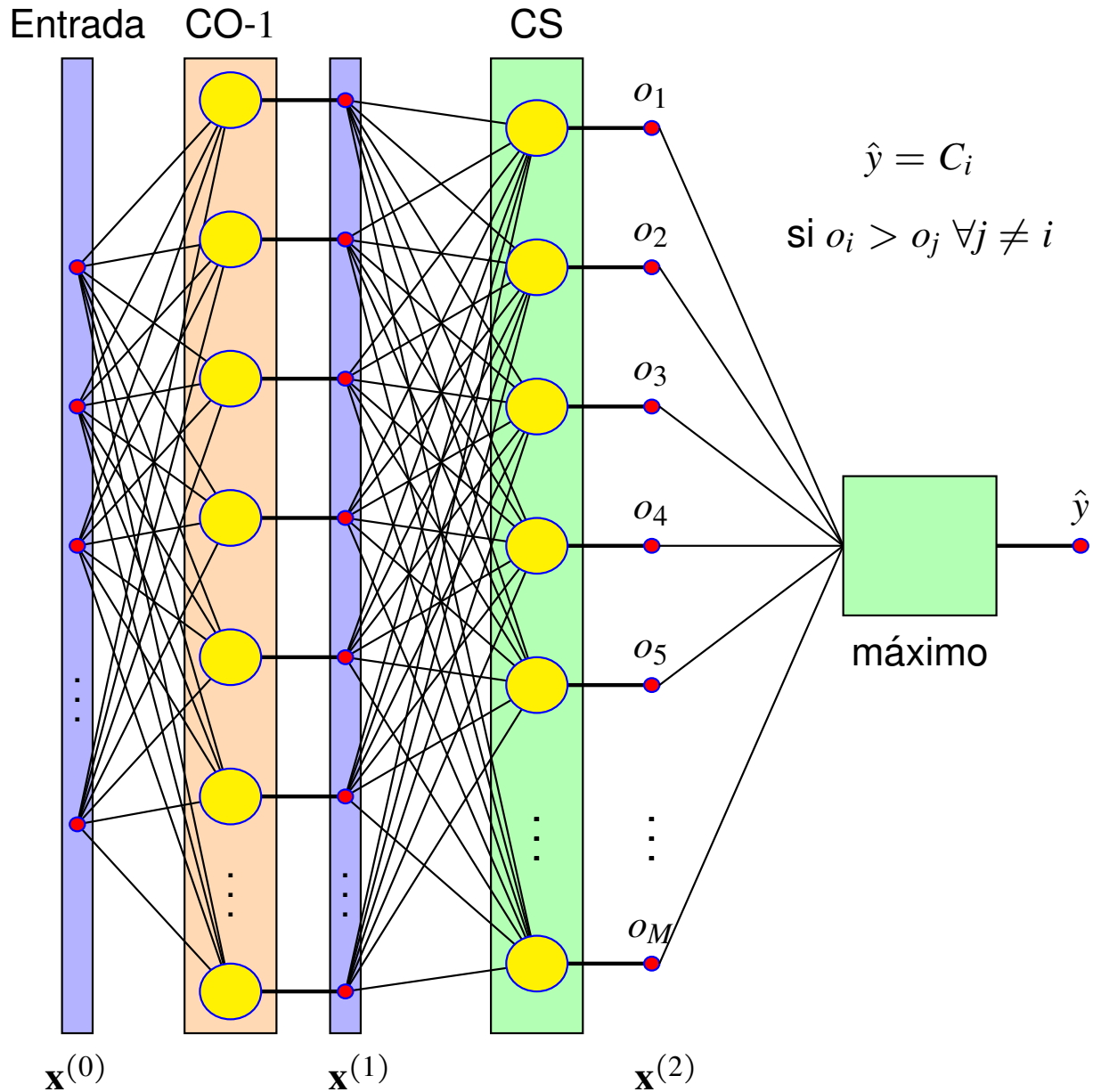
$$o_i = \frac{e^{z_i}}{\sum_{k=1}^{M} e^{z_k}}$$

Softmax

Red neuronal no profunda: 1 capa oculta (clasif. binaria)



Red neuronal no profunda: 1 capa oculta (clasif. M-ária)



Entrenamiento supervisado para clasificación

- Entrenamiento supervisado para clasificación
  - Aprendizaje a partir de ejemplos etiquetados (conjunto de entrenamiento)

$$\{(\mathbf{x}_k, y_k)\}_{k=1}^{N} \qquad \mathbf{x}_k \in \boldsymbol{R}^D \qquad y_k \in \mathcal{C} = \{C_1, C_2, \cdots, C_M\}$$

- Entrenamiento de la red
  - Ajuste de los parámetros de la red: $\mathbf{w}$
    - ★ Parámetros adecuados para resolver el problema
  - Minimizar una función de coste relacionada con el objetivo del problema: $J(\mathbf{w})$
- Ejemplos de funciones de coste

$\mathbf{o}_k$: salida de la red para el patrón $\mathbf{x}_k$

$\mathbf{o}_k^r$: valor de referencia en la salida de la red para el patrón $\mathbf{x}_k$

  - MMSE: error cuadrático medio entre la salida de la red y un valor de referencia asociado a cada patrón del conjunto de entrenamiento

$$J^{MMSE}(\mathbf{w}) = \frac{1}{N}\sum_{k=1}^{N} ||\mathbf{o}_k - \mathbf{o}_k^r||^2$$

  - Cross Entropy: entropía cruzada entre la salida de la red y el valor de referencia

$$J^{CE}(\mathbf{w}) = \frac{1}{N}\sum_{k=1}^{N} -\sum_{j=1}^{M} o_{k,j}^r \log(o_{k,j})$$

Valores de referencia para las salidas

- Clasificador binario

$$o^r|c_1 = -1, \qquad o^r|c_2 = +1$$

Decisión: comparar la salida de la red con un umbral (cero)

- Clasificador M-ário: "*One-Hot Encoding*" (OHE)

$$\mathbf{o} = \begin{bmatrix} o_1 \\ o_2 \\ o_3 \\ o_4 \\ \vdots \\ o_M \end{bmatrix}, \mathbf{o}^r|c_1 = \begin{bmatrix} 1 \\ 0 \\ 0 \\ 0 \\ \vdots \\ 0 \end{bmatrix}, \mathbf{o}^r|c_2 = \begin{bmatrix} 0 \\ 1 \\ 0 \\ 0 \\ \vdots \\ 0 \end{bmatrix}, \mathbf{o}^r|c_3 = \begin{bmatrix} 0 \\ 0 \\ 1 \\ 0 \\ \vdots \\ 0 \end{bmatrix}, \cdots, \mathbf{o}^r|c_M = \begin{bmatrix} 0 \\ 0 \\ 0 \\ 0 \\ \vdots \\ 1 \end{bmatrix}$$

Decisión: salida con valor más alto (máximo)

Regularización

- Suavizar la solución de la red

$$J(\mathbf{w}) = J^{COSTE}(\mathbf{w}) + \lambda\, R(\mathbf{w})$$

  - Regularizador $R(\mathbf{w})$
    - ★ Penaliza ciertos valores de $\mathbf{w}$
  - Parámetro de regularización λ (hiper-parámetro)
    - ★ Compromiso entre ajuste a los datos y suavidad de la solución
- Ejemplos de regularizadores
  - Norma $\mathcal{L}_2$

$$R(\mathbf{w}) = \sum_k \sum_i \sum_j \left(w_{i,j}^{(k)}\right)^2$$

    - ★ Asume una distribución a priori Gaussiana sobre los pesos
  - Norma $\mathcal{L}_1$

$$R(\mathbf{w}) = \sum_k \sum_i \sum_j \left|w_{i,j}^{(k)}\right|$$

    - ★ Asume una distribución a priori Laplaciana sobre los pesos
    - ★ Tiende a anular algunos pesos más que la norma $\mathcal{L}_2$

Aprendizaje - Entrenamiento de la red neuronal

- El aprendizaje se plantea como un problema de optimización
  - Minimización de una función de coste $J(\mathbf{w})$ no convexa
- Ajuste de los pesos $\mathbf{w}$ para minimizar $J(\mathbf{w})$
  - Inicialización + ajuste iterativo por descenso de gradiente

$$\mathbf{w}(n+1) = \mathbf{w}(n) + \Delta_{\mathbf{w}}(n)$$

$$\Delta_{\mathbf{w}}(n) = -\mu \nabla_{\mathbf{w}}(n) \begin{cases} \mu: \text{ paso de adaptación} \\ \nabla_{\mathbf{w}}(n): \text{ dirección (opuesta) de adaptación} \end{cases}$$

$$\text{Habitual}: \nabla_{\mathbf{w}}(n) = \left.\frac{\partial J(\mathbf{w})}{\partial \mathbf{w}}\right|_{\mathbf{w}=\mathbf{w}(n)} \text{ (estima del gradiente)}$$

- Número de patrones utilizados para la estima en cada iteración
  - Ajuste estocástico (*on-line*): 1 patrón $(\mathbf{x}_k, y_k \rightarrow o_k^r)$

$$\left.\frac{\partial J(\mathbf{w})}{\partial \mathbf{w}}\right|_{\mathbf{x}=\mathbf{x}_k} = \sum_{j=1}^{M} \frac{\partial J(\mathbf{w})}{\partial o_{k,j}} \times \frac{\partial o_{k,j}}{\partial \mathbf{w}}$$

$$\begin{cases} \frac{\partial J(\mathbf{w})}{\partial o_{k,j}}: \text{ depende de } J(\mathbf{w}) \\ \frac{\partial o_{k,j}}{\partial \mathbf{w}}: \text{ depende de la red} \end{cases}$$

  - Ajuste en bloque (*batch*): N patrones (todos)
  - Ajuste en mini-bloque (*mini-batch*): $1 < N_{batch} < N$ patrones

NOTA: Época (*epoch*): iteraciones en las que se usa todo el conjunto de entrenamiento

Inicialización de los parámetros (w)

- Inicialización aleatoria
  - NO es útil $w_{i,j}^{(k)} = 0,\ \forall k,i,j$
    - ★ Todos los gradientes son nulos (“saddle point”)
  - NO es útil $w_{i,j}^{(k)} = a,\ \forall k,i,j$
    - ★ Todas las neuronas de una capa se comportarían igual
    - ★ Es necesario “romper la simetría”
- Varias posibilidades de inicialización aleatoria
  - En general: pesos pequeños con media nula
  - Ejemplos:

$$\mathbf{w}^{(k)} \sim \mathcal{N}(0,\sigma^2), \quad \sigma^2 = \frac{1}{D^{(k-1)}}$$

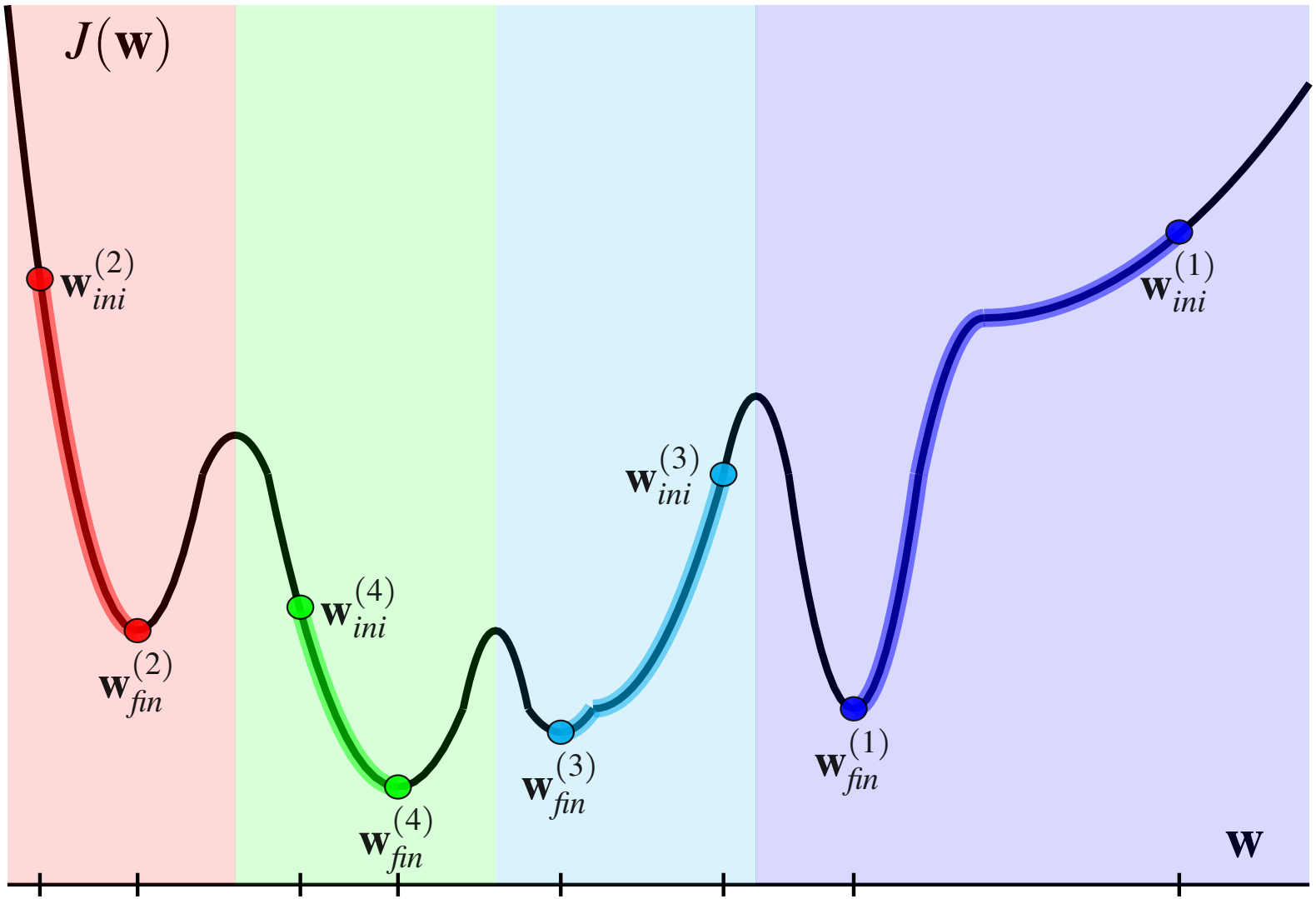
$$\mathbf{w}^{(k)} \sim \mathcal{U}(-a,a), \quad a = \frac{1}{\sqrt{D^{(k-1)}}}$$

$$\mathbf{w}^{(k)} \sim \mathcal{U}(-a,a), \quad a = \frac{\sqrt{6}}{\sqrt{D^{(k)} + D^{(k-1)}}}$$

  - En algunos casos $b_i^{(k)} = 0,\ \forall k,i$

Función de coste no convexa - Mínimos locales

- Solución dependiente de la inicialización



Entrenamiento supervisado: evaluación de soluciones

- Problema real
  - Se diseña la solución neuronal a partir de los datos disponibles en el conjunto de entrenamiento
  - La solución se aplicará después sobre otros datos

    (que en general no están en el conjunto de entrenamiento)
    - ★ Prestaciones relevantes: sobre los datos que no están en el conjunto de entrenamiento
    - ★ Capacidad de generalización
- Aplicación en un problema real (o simulación académica)
  - Conjunto de entrenamiento
    - ★ Conjunto de entrenamiento

      A partir del que se realiza el entrenamiento supervisado de la red (ajuste de los parámetros)
    - ★ Conjunto de validación

      Sirve para validar los distintos diseños (hiper-parámetros como número de capas ocultas, número de neuronas de cada capa, número de épocas para el entrenamiento; distintas inicializaciones (mínimos locales); etc.) y elegir la solución que potencialmente tendrá unas mejores prestaciones
  - Conjunto de test (evaluación)

    Datos a los que se aplicará el modelo diseñado, y que no se han utilizado en su construcción (ni para ajustar los parámetros, ni para validar la mejor configuración): permite medir las prestaciones de la solución

Algunas mejoras para el aprendizaje

- Incluir un término de "Momento"

$$\mathbf{w}(n+1) = \mathbf{w}(n) + \Delta_{\mathbf{w}}(n)$$

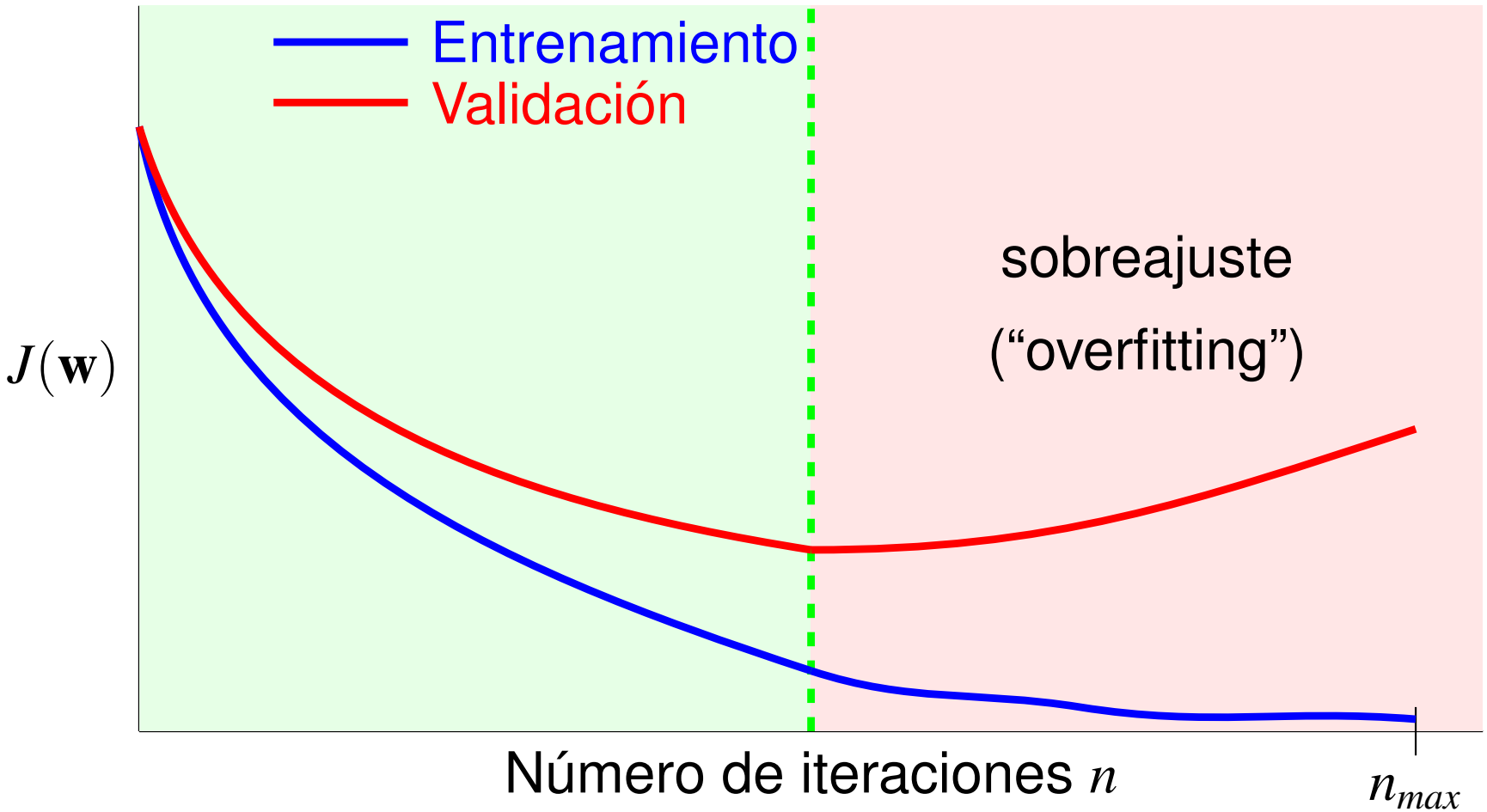
$$\Delta_{\mathbf{w}}(n) = \beta\ \Delta_{\mathbf{w}}(n-1) - \mu\ \left.\frac{\partial J(\mathbf{w})}{\partial \mathbf{w}}\right|_{\mathbf{w}=\mathbf{w}(n)}$$

  - Puede acelerar la convergencia
  - Algunas variantes
    - ★ Nesterov's Accelerated Gradient (NAG): $\left.\frac{\partial J(\mathbf{w})}{\partial \mathbf{w}}\right|_{\mathbf{w}=\tilde{\mathbf{w}}},\ \tilde{\mathbf{w}} = \mathbf{w}(n) + \beta\ \Delta_{\mathbf{w}}(n-1)$

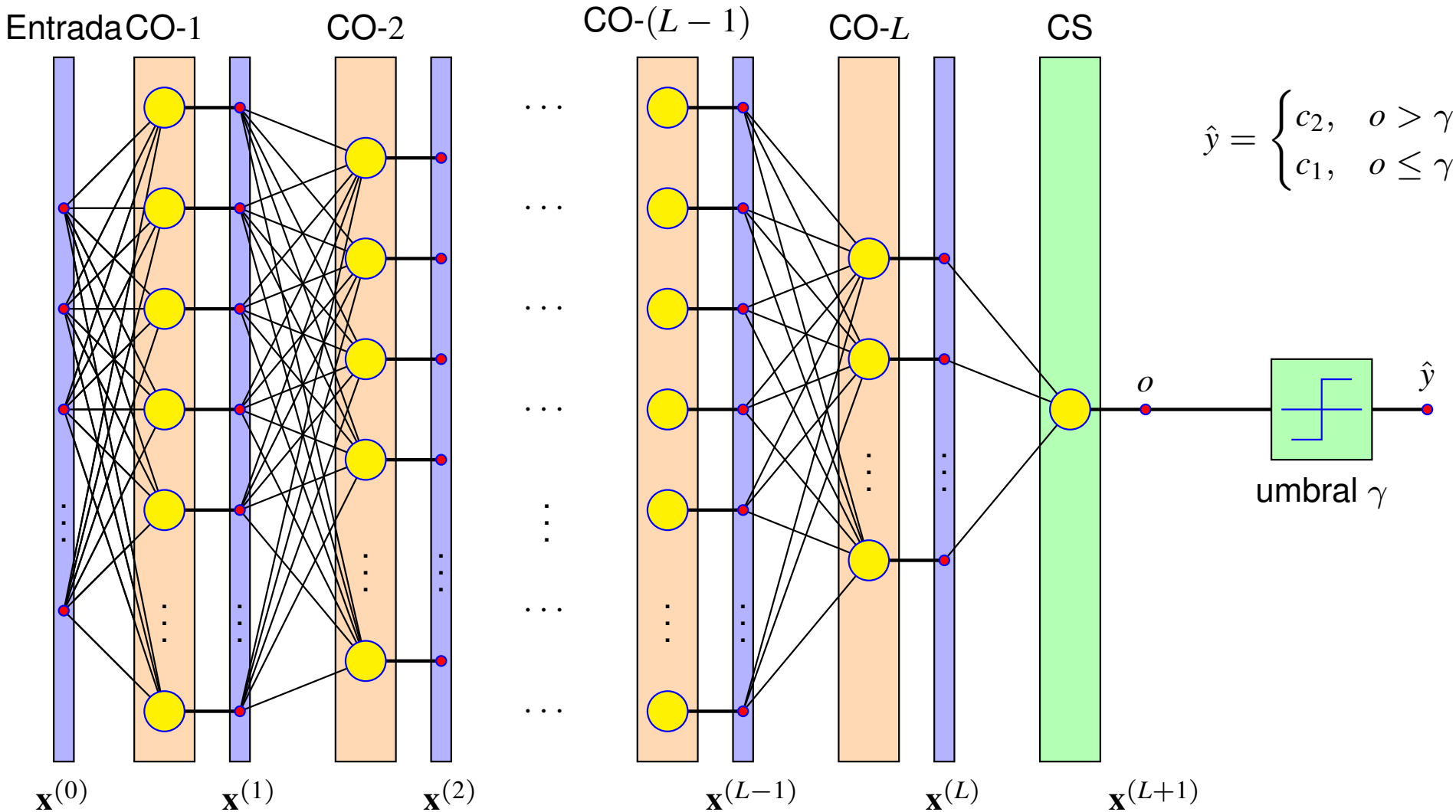
- Decrecimiento del paso de adaptación: $\mu \rightarrow \mu(n)$
  - Compromiso paso grande/pequeño
    - ★ Paso μ grande: rapidez en la fase inicial
    - ★ Paso μ pequeño: mejor ajuste en la fase final
  - Varias estrategias de decrecimiento progresivo
    - ★ Decrecimiento basado en el tiempo ("time-based decay") $\mu(n) = \frac{\mu(0)}{1+\gamma n}$
    - ★ Decrecimiento escalonado ("step-decay") $\mu(n) = \mu(0)\ \gamma^{\left\lfloor \frac{n}{N_{dec}} \right\rfloor}$
    - ★ Decrecimiento exponencial $\mu(n) = \mu(0)\ e^{-\gamma n}$
    - ★ Decrecimiento adaptativo (basado en el conjunto de validación)

Criterio de parada

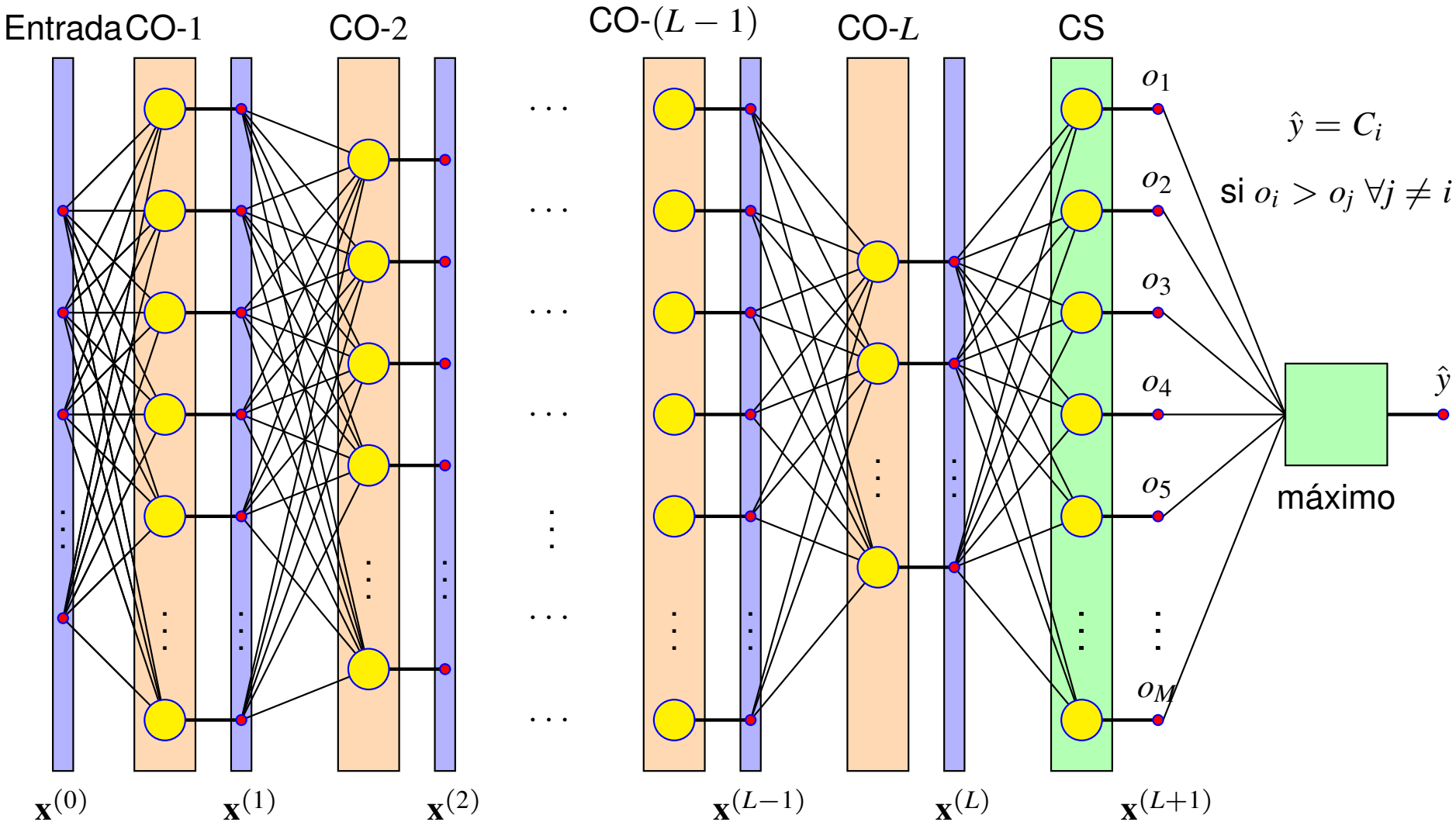
- Fijar un número máximo de iteraciones (o épocas)
- Parada: coste en el conjunto de validación aumenta
- Algunas variaciones
  - Validación cruzada
  - Parada un poco antes (“early stopping”)



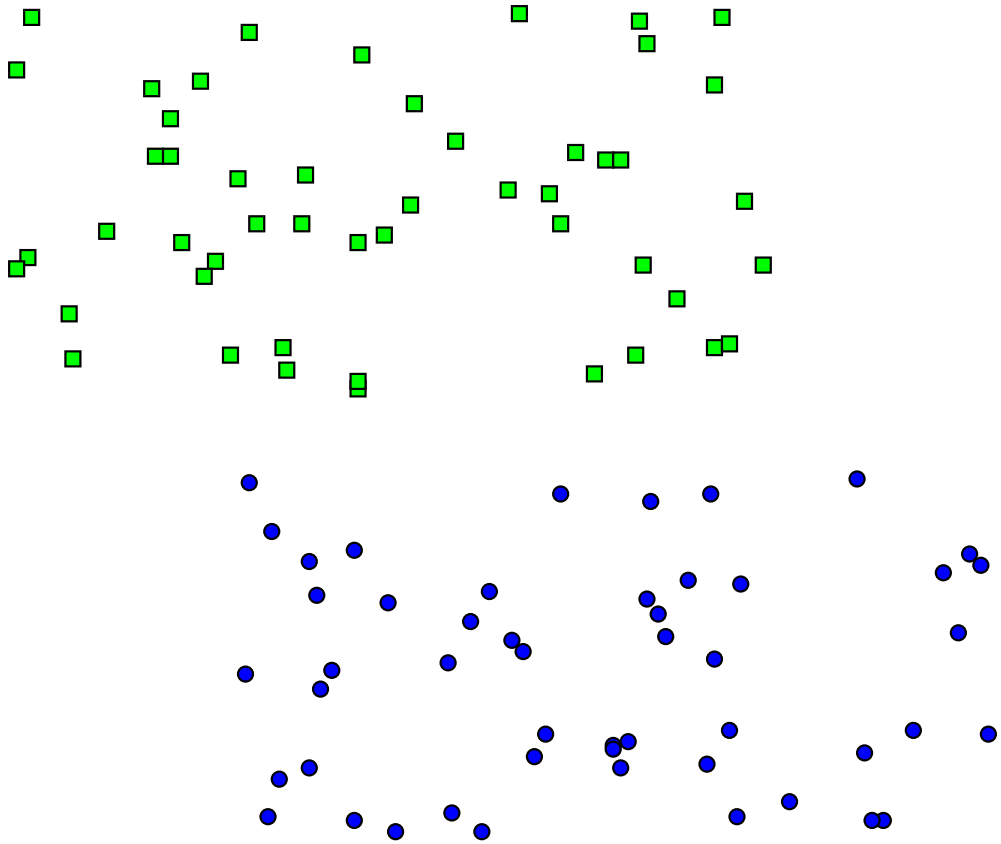
Red profunda: red neuronal con L capas ocultas (binario)



Red profunda: red neuronal con L capas ocultas (M-ário)



Capas ocultas - Proyección de la entrada



Datos proyectados en espacio 2D

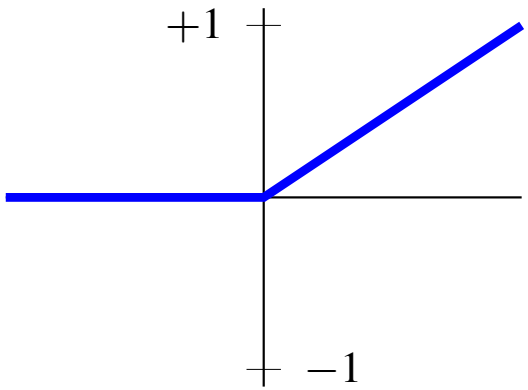
Datos proyectados en espacio 1D

Entrenamiento de redes profundas

- En los primeros años de uso de las redes neuronales las redes profundas no eran frecuentes
  - Carga computacional excesiva para los recursos de la época
  - Dificultades para el entrenamiento mediante el algoritmo de retropropagación de errores (*"backpropagation"*)
    - ★ Problema de dilución del gradiente: aprendizaje muy lento
    - ★ Gran número de parámetros: mínimos locales y sobreajuste
- Gran relevancia en los últimos años (principios de siglo)
  - Avances en hardware de computación (GPUs)
    - ★ Tiempos de entrenamiento reducidos en órdenes de magnitud
  - Métodos de inicialización de las capas ocultas (no supervisados)
    - ★ Autocodificadores
    - ★ Máquinas de Boltzman con restricciones (RBM)
      - Deep Belief Networks (DBNs)
  - Nuevos métodos de optimización (Nesterov momentum, ADAM,...)
  - Alternativas para reducir la dilución del gradiente (batch regularization,...)
  - Nuevos métodos de regularización (drop-out, drop-connect,...)
  - Diseño de redes para aplicaciones específicas (CNNs, LSTMs,...)
  - Nuevas funciones de activación (ReLU, eLU,...)

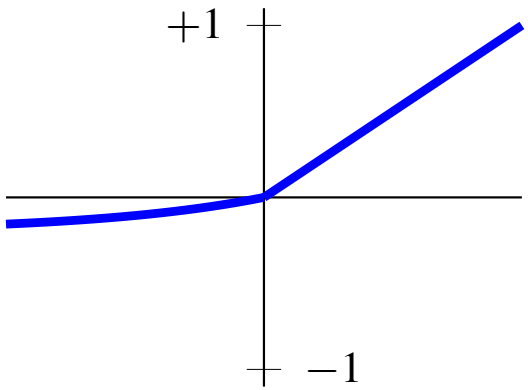
Funciones de activación para redes profundas

$$f(x) = \begin{cases} x, & x \geq 0 \\ 0, & x < 0 \end{cases}$$



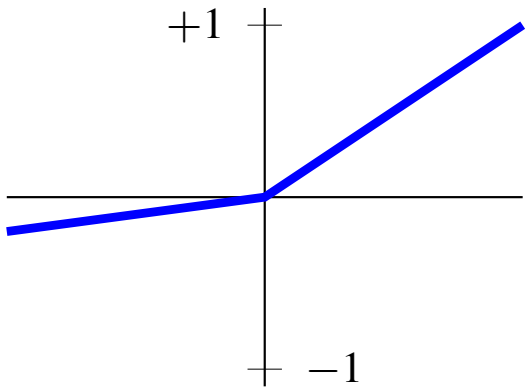
Lineal rectificada (ReLU)

$$f(x) = \begin{cases} x, & x \geq 0 \\ \alpha(e^x - 1), & x < 0 \end{cases}$$



Lineal exponencial (eLU)

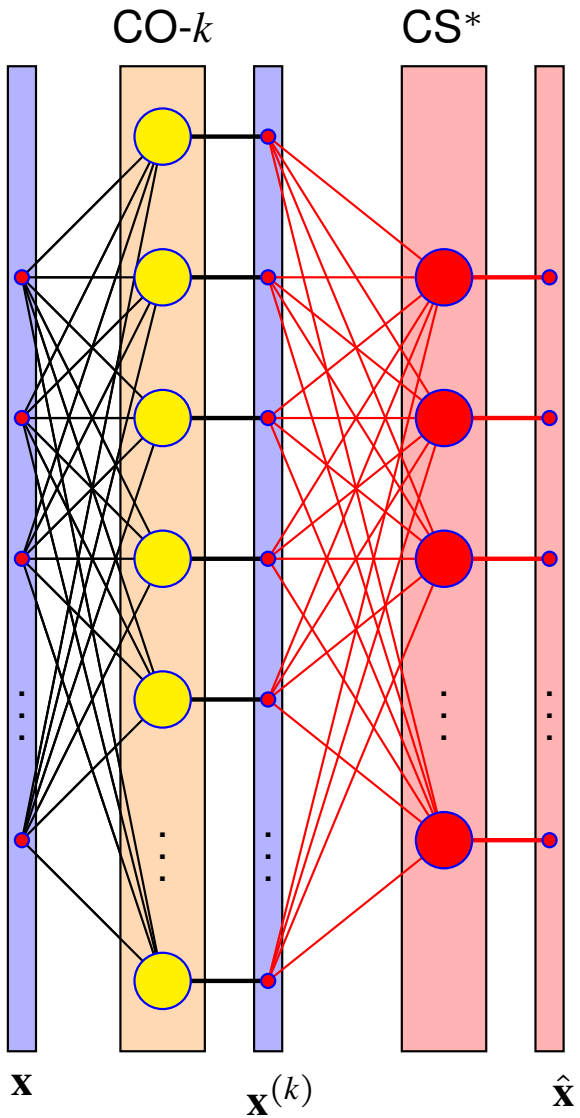
$$f(x) = \begin{cases} x, & x \geq 0 \\ \alpha x, & x < 0 \end{cases}$$



Leaky ReLU (LReLU)

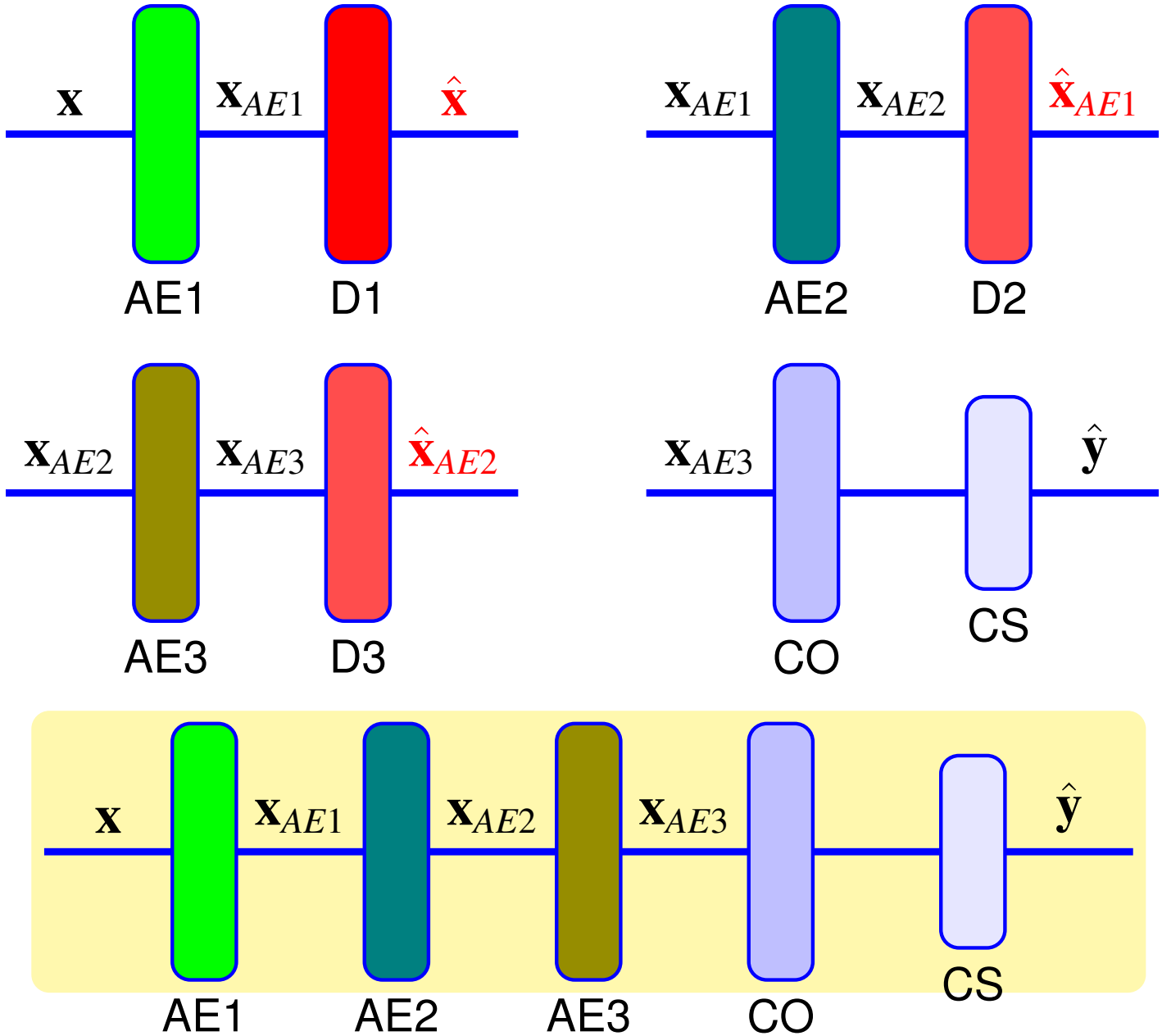
Stacked Denoising Autoencoders

Autocodificadores (*"autoencoders"*)



- El entrenamiento de una capa oculta se realiza para obtener una proyección de su entrada sin pérdida de información
  - Entrada: $\mathbf{x} = \mathbf{x}^{(k-1)}$
  - Salida deseada: $\hat{\mathbf{x}} = \mathbf{x}^{(k-1)}$
- La capa de salida se descarta
  - Se usa como mecanísmo para evitar la pérdida de información, al garantizar que la proyección obtenida es reversible
- Denoising autoencoders
  - Se incluye ruido en la entrada para hacer la proyección robusta a perturbaciones sobre la entrada (y para evitar soluciones triviales)
    - ★ Entrada: $\mathbf{x} = \mathbf{x}^{(k-1)} + \mathbf{n}$
    - ★ Salida deseada: $\hat{\mathbf{x}} = \mathbf{x}^{(k-1)}$

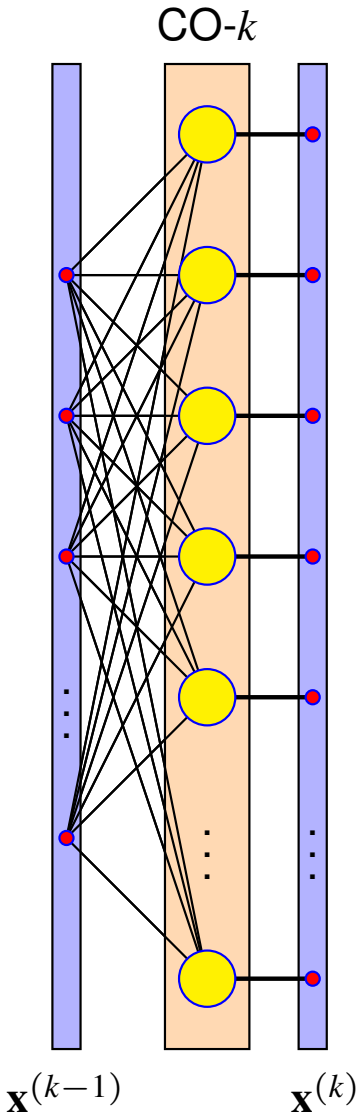
Construcción de red profunda con autocodificadores



Construcción de red profunda con autocodificadores

- Entrenamiento secuencial de $L-1$ autocodificadores ("denoising")
  - Se autocodifica la codificación del autocodificador anterior
  - Entrenamiento no supervisado
- Entrenamiento de la etapa final
  - Capa oculta (opcional)
  - Capa de salida: decisión de la clase
  - Entrenamiento supervisado
- Ajuste fino final
  - Entrenamiento supervisado de toda la red
- Autocodificadores (aprendizaje no supervisado)
  - Extracción de una representación de los datos
    - ★ Independiente de la aplicación supervisada (clasificación, regresión,...)
  - Aplicaciones reales
    - ★ En muchos casos, sólo una pequeña porción de los datos disponibles están etiquetados

Deep Belief Networks (DBNs)



- Agrupamiento de RBMs (*"Restricted Boltzman Machines"*)
  - Modelo gráfico generativo no supervisado
  - Extracción de una representación jerárquica de los datos
  - Modelo probabilístico entre la entrada y capa oculta
  - No es un modelo "feed-forward"
- DBN : Entrenamiento progresivo de cada capa
  - Representación de los datos de una capa
    - ★ Entrada: $\mathbf{x}^{(k-1)}$
    - ★ Salida: capa k, tal que activaciones $\mathbf{x}^{(k)}$ permitan reconstruir $\mathbf{x}^{(k-1)}$
    - ★ Activaciones medias o muestras de $p(\mathbf{x}^{(k-1)}|\mathbf{x}^{(k)})$
  - Entrenamiento de la siguiente capa como una RBM ($\mathbf{x}^{(k+1)}$)
    - ★ Reconstrucción de $\mathbf{x}^{(k)}$
  - Modelo probabilístico de la DBN

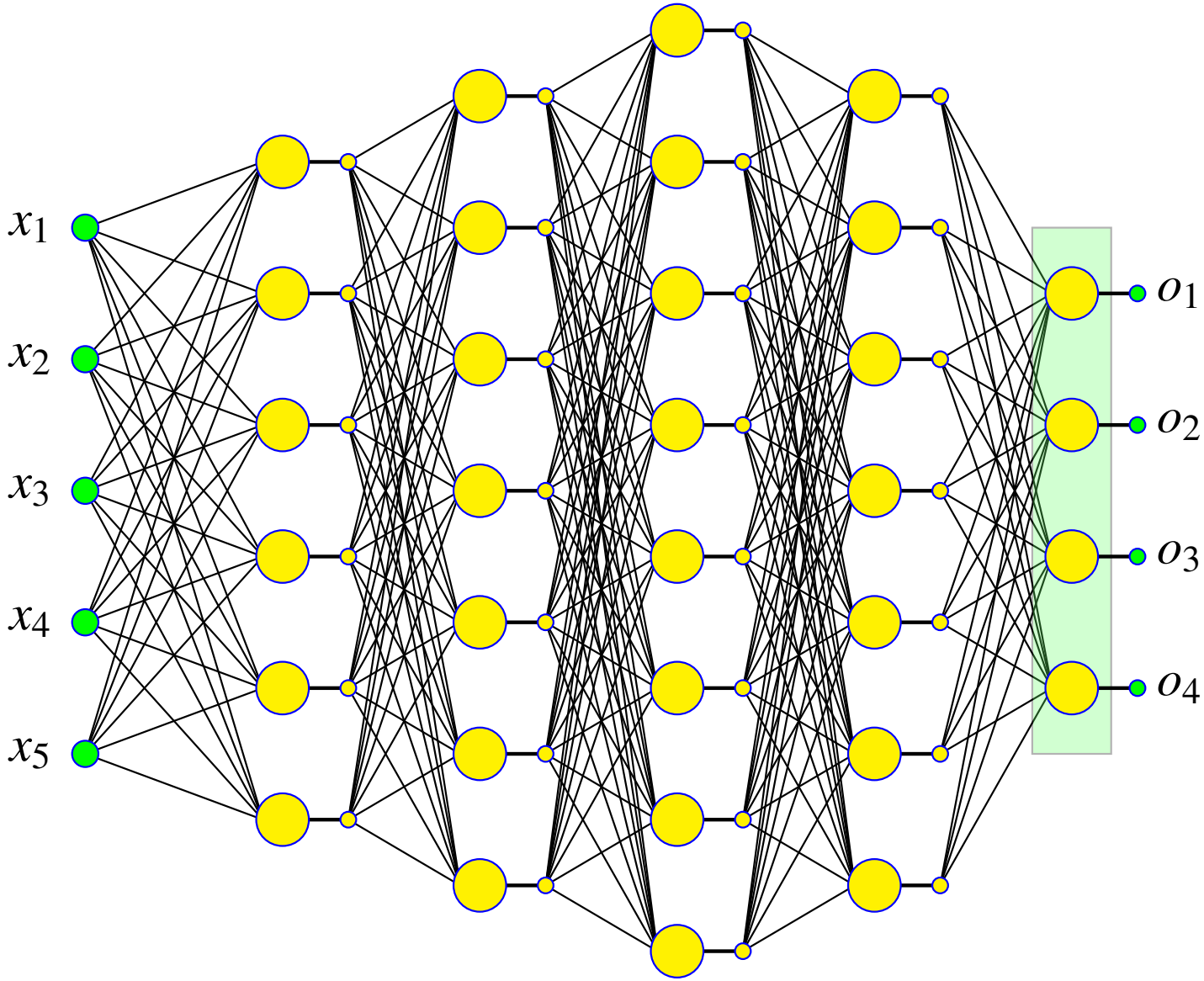
$$p(\mathbf{x}^{(0)}, \mathbf{x}^{(1)}, \cdots, \mathbf{x}^{(L)}) = p(\mathbf{x}^{(0)}|\mathbf{x}^{(1)} \left(\prod_{k=1}^{L-1} p(\mathbf{x}^{(k)}|\mathbf{x}^{(k+1)}) \right)$$

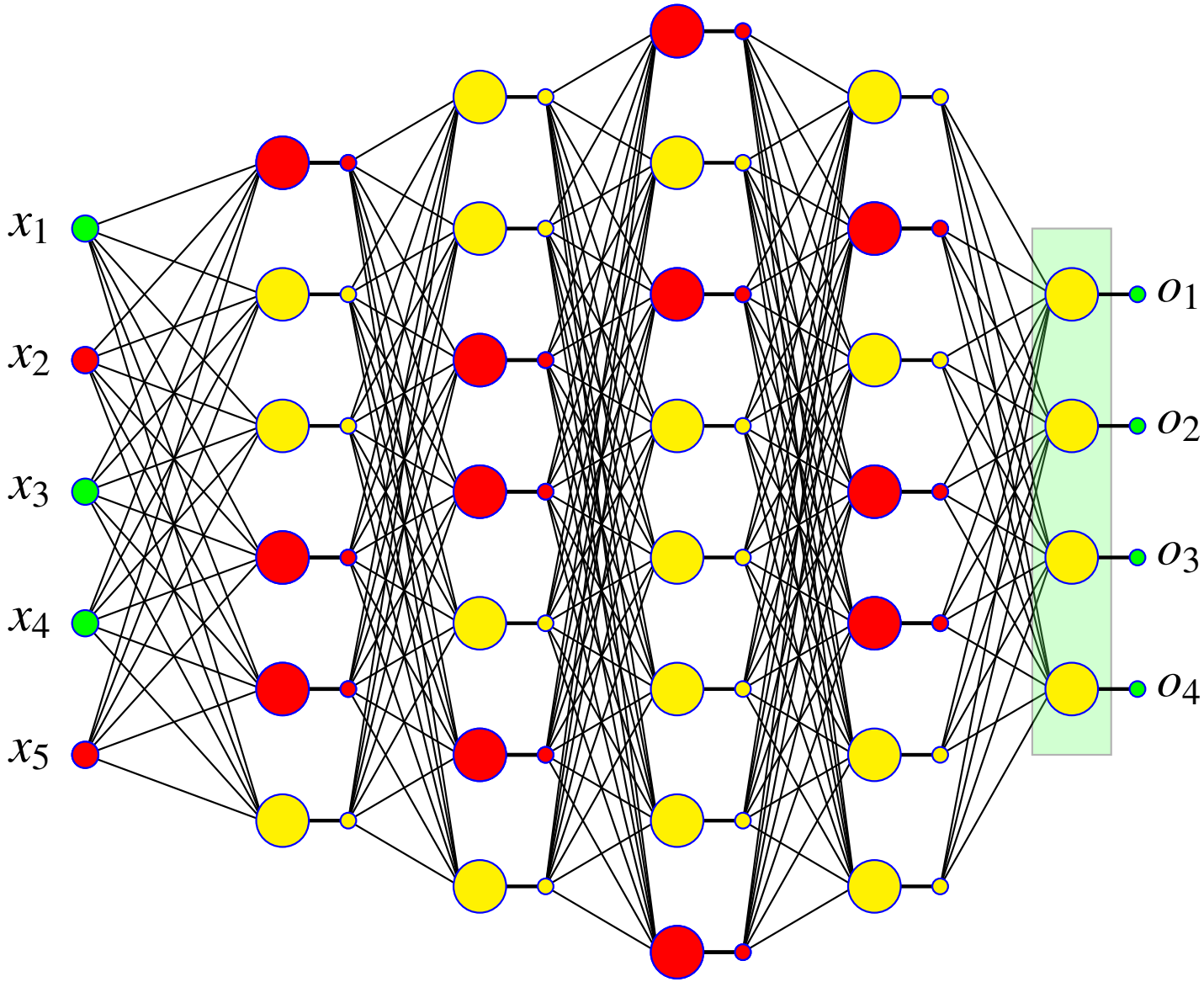
- Opción: inicialización de los pesos de una red profunda
  - Se añadirá una capa de salida supervisada

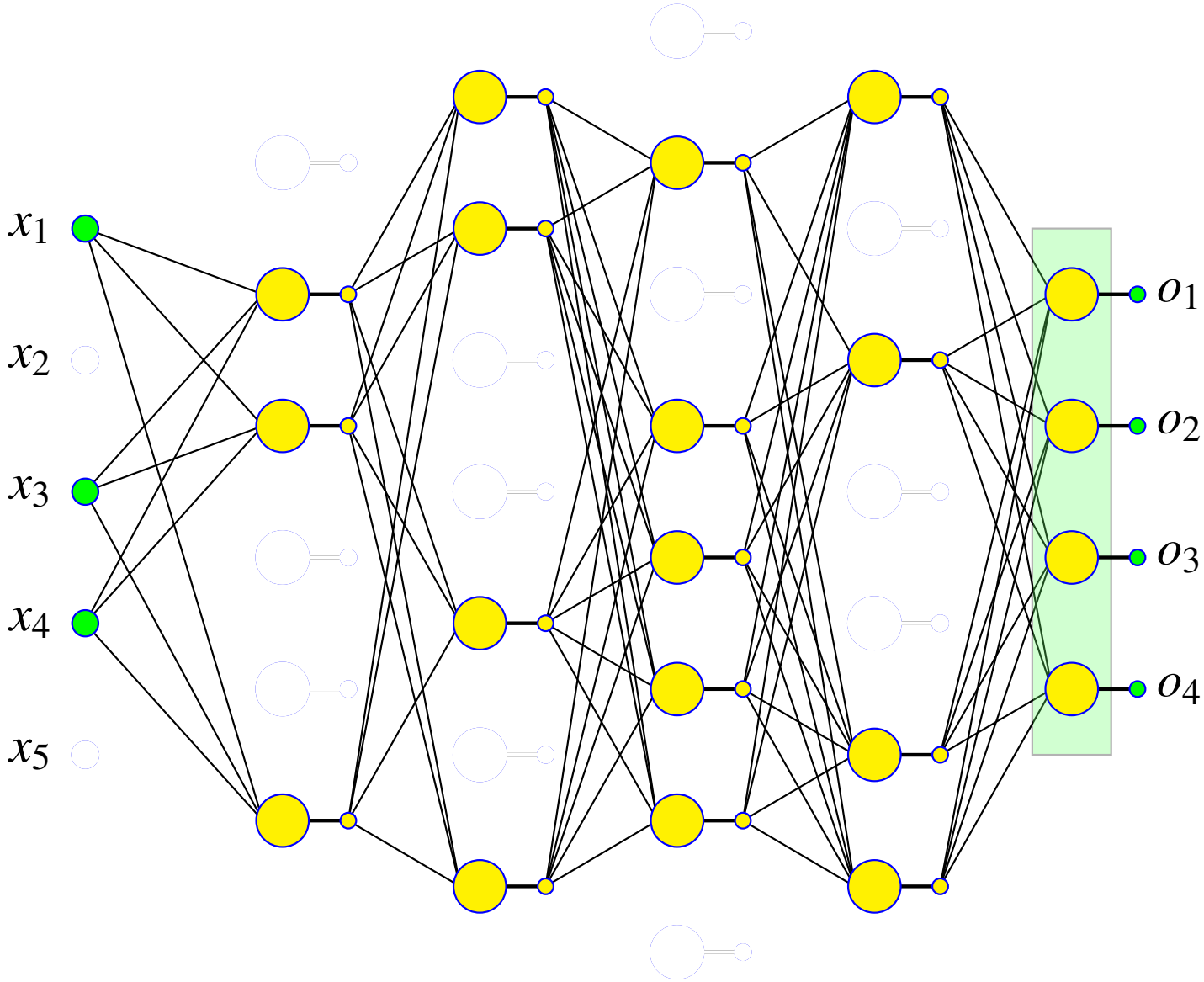
Redes profundas con Drop-Out

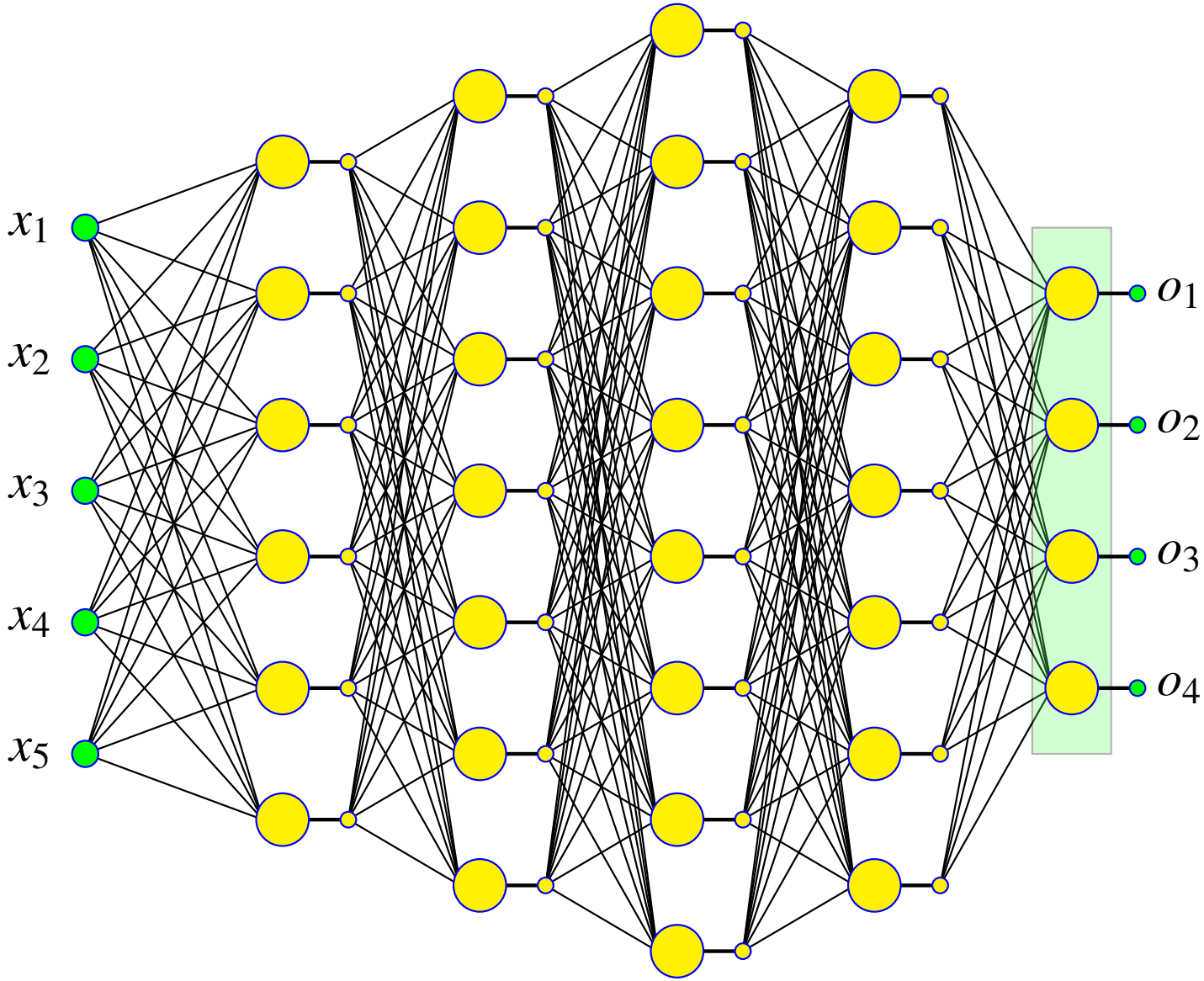
Regularización mediante *drop-out*

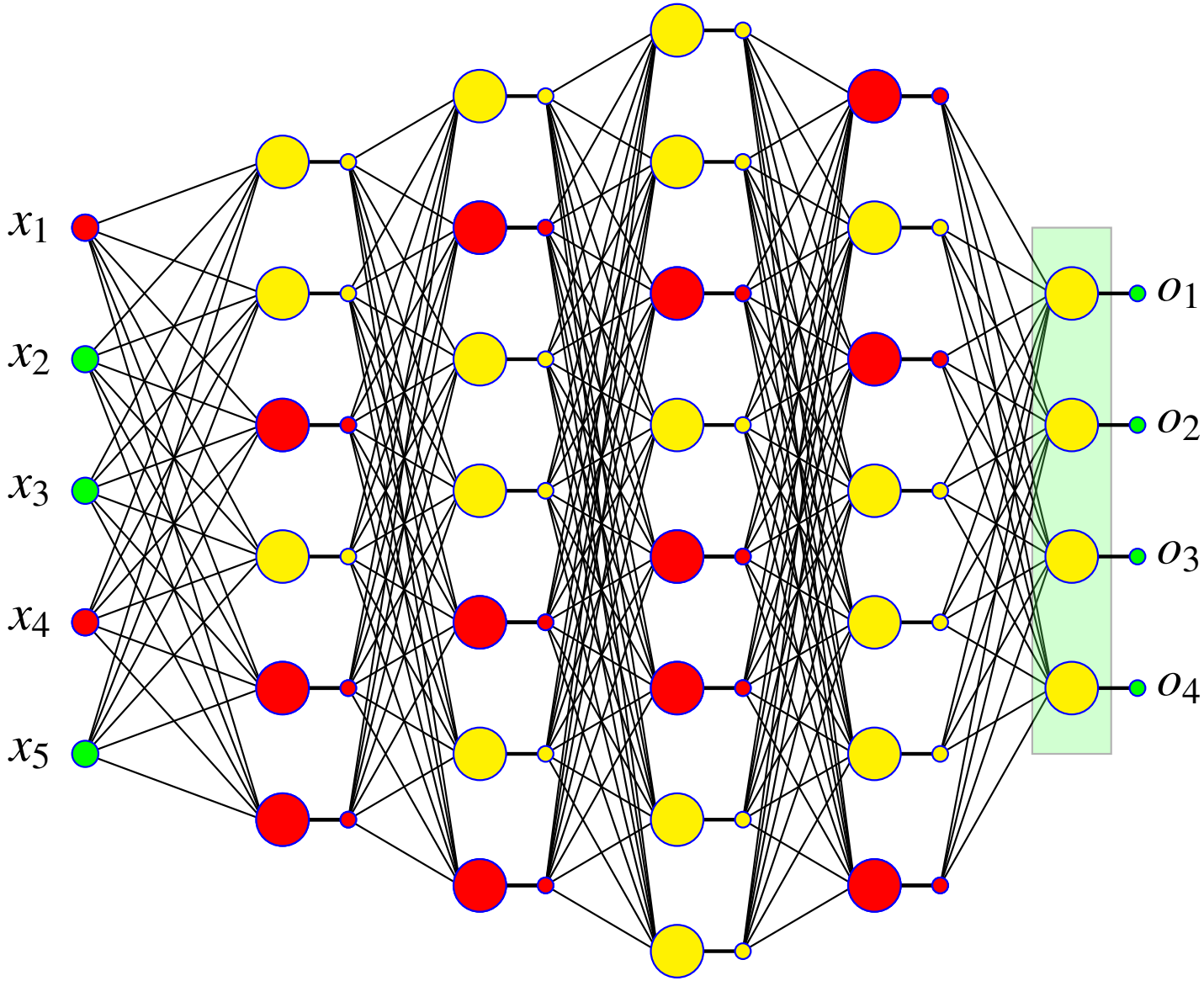
- Método diseñado para evitar el sobreajuste de la red
  - Frecuente en redes grandes (muchos parámetros en relación a los datos disponibles)
- Se define una probabilidad de *drop-out* para:
  - Capa de entrada
  - Capas ocultas
    - Pueden ser iguales en todas las capas o diferentes
- En cada iteración del algoritmo de entrenamiento
  - Se seleccionan para cada capa las neuronas en *drop-out*
    - Selección aleatoria con la probabilidad especificada
  - Se entrena la red como si esas neuronas no existieran
    - Los parámetros de las neuronas en *drop-out* no se modifican en esa iteración
    - Para la estima de los valores de salida de la red y para el cálculo del gradiente en esa iteración, se asume que esas neuronas no existen
- Existen variantes similares
  - *Drop-Connect*: se eliminan de forma aleatoria conexiones, en lugar de neuronas

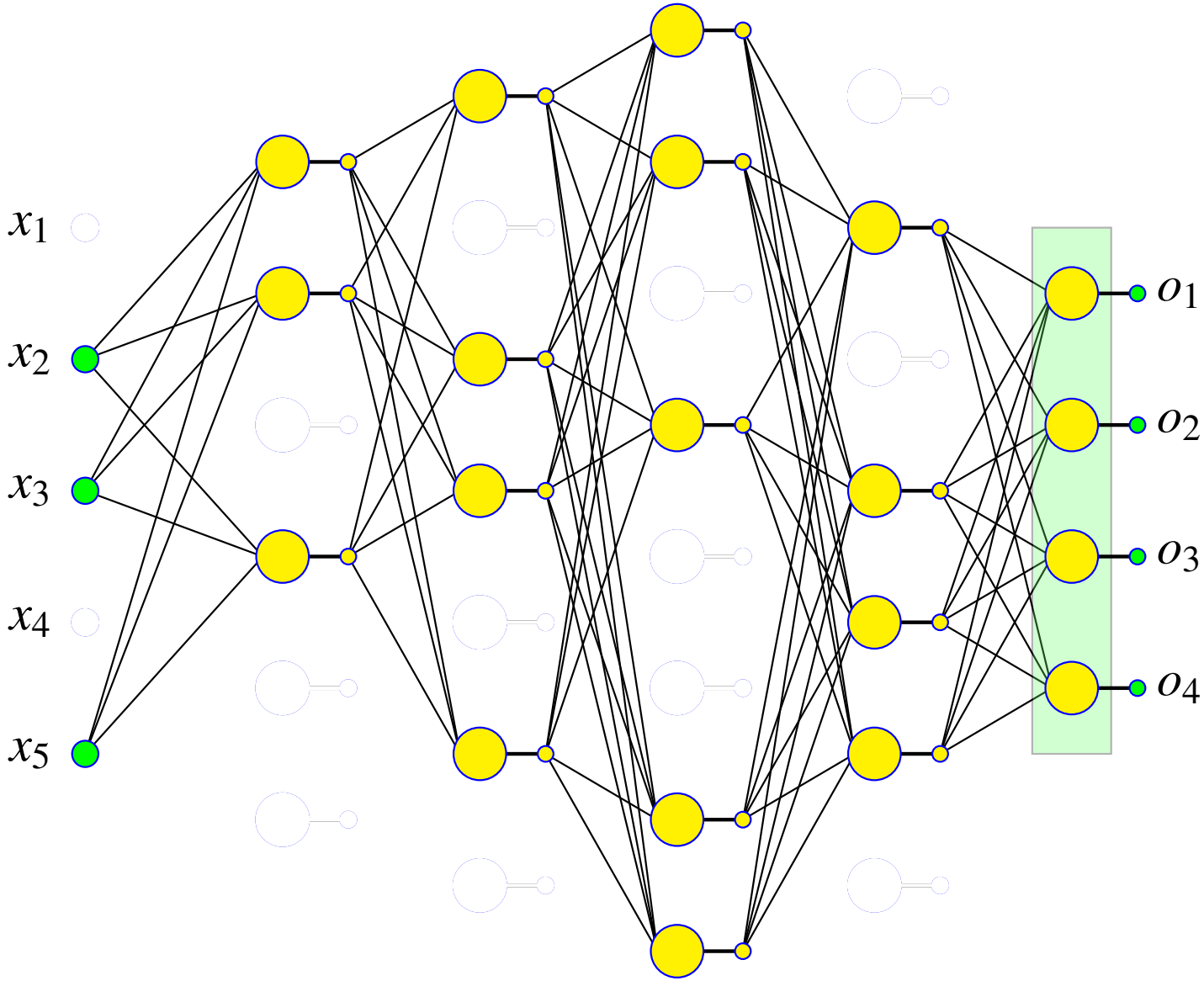


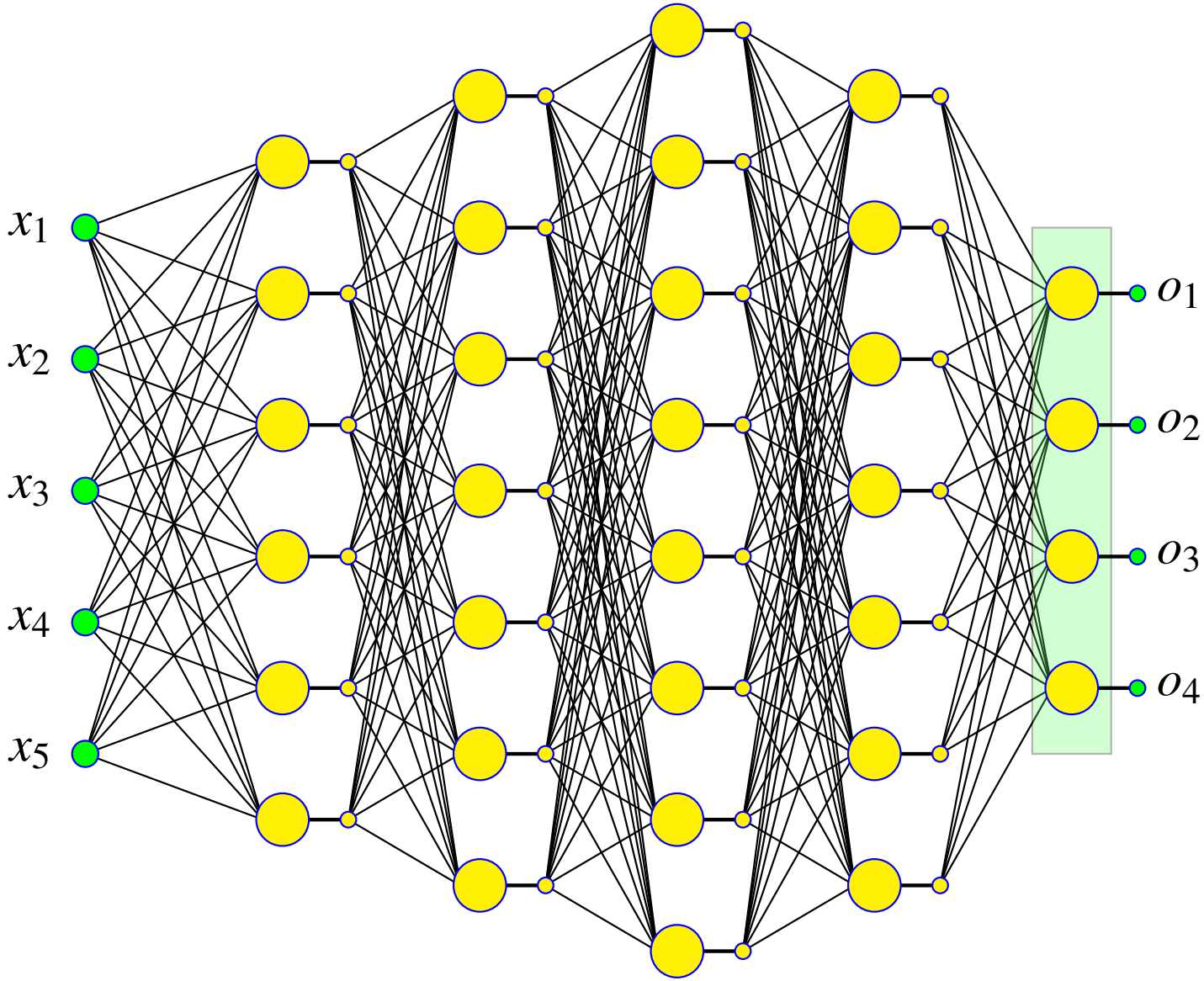









x_1
x_2
x_3
x_4
x_5
o_1
o_2
o_3
o_4

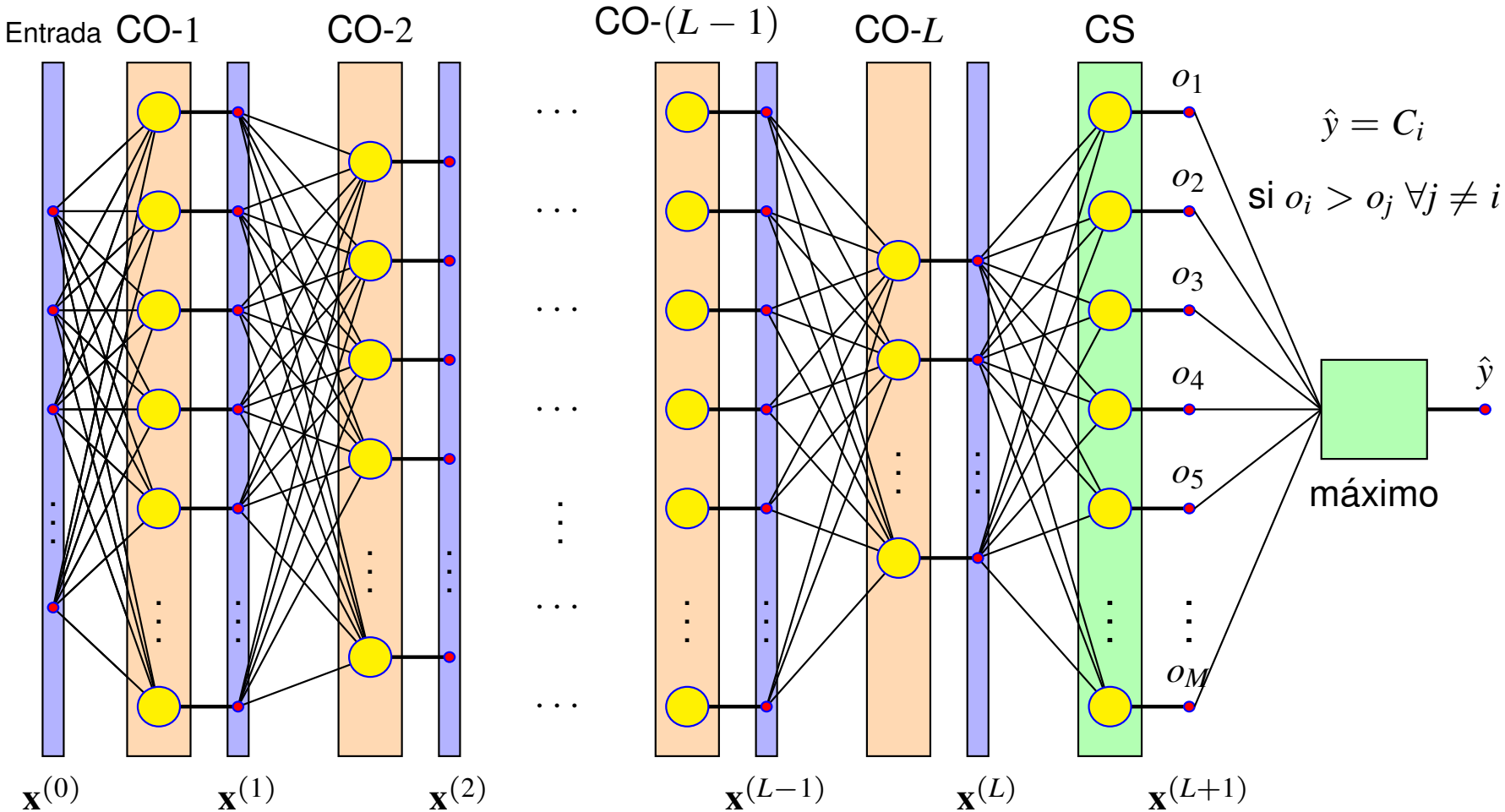


Batch Normalization

Normalización de los datos

- Características de los datos de entrada (sin procesar)
  - Posibles rangos y distribuciones muy diferentes en dimensiones distintas
  - Esta diversidad puede dificultar la clasificación
    - ★ Inicialmente, las dimensiones con valores más grandes pueden ser más influyentes (peso en las salidas, gradientes,...)
- La normalización de los datos de entrada mejora en general las prestaciones obtenidas con redes neuronales
  - Mayor uniformidad en rangos y/o distribuciones
  - No debe distorsionar las diferencias entre valores por dimensión
- Varios tipos de normalización
  - Normalización mínimo/máximo (normalización)
    - ★ Valores mínimo y máximo fijos (típicos $[0, 1]$ o $[-1, +1]$)
  - Normalización en media y varianza (estandarización)
    - ★ Media nula y varianza unidad en todas las dimensiones

Internal Covariance Shift



- Cambios en las distribuciones sobre las entradas en las capas internas de la red
  - Los cambios se propagan y obligan a la red a adaptarse a esos cambios
  - Mayor influencia en redes profundas
  - Puede afectar a las prestaciones y velocidad del entrenamiento

Batch normalization

- Inicialmente diseñado para reducir la *internal covariance shift*
- Tiene otros efectos beneficiosos
  - Acelera en entrenamiento (permite pasos de adaptación mayores)
  - Tiene efectos de regularización (previene sobreajustes)
  - Puede suavizar la función de coste (menor sensibilidad a mínimos locales)
- Normalización en cada mini-batch
  - Normalización en media y varianza de los valores internos de las neuronas

$$o_i^{(k)} = \sum_{j=1}^{D_{k-1}} w_{j,i}^{(k)}\, x_j^{(k-1)} + b_i^{(k)} \underset{\text{normalización}}{\rightarrow} \bar{o}_i^{(k)} = \frac{o_i^{(k)} - \text{media}(o_i^{(k)})}{\text{desv}(o_i^{(k)}) + \epsilon}$$

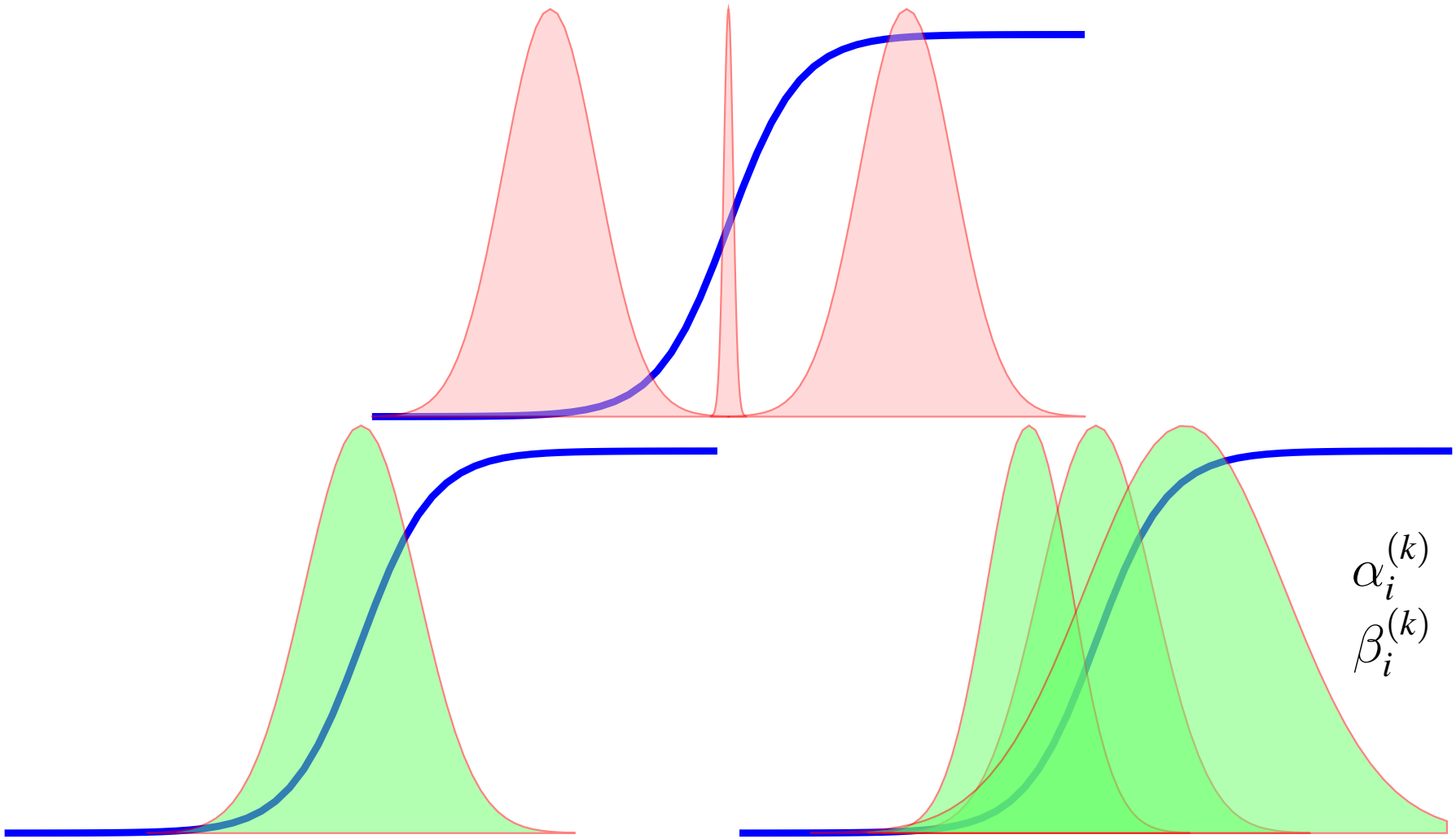
  - La función de activación se aplica a

$$z_i^{(k)} = \alpha_i^{(k)} \bar{o}_i^{(k)} + \beta_i^{(k)} \underset{\text{salida de la neurona}}{\rightarrow} x_i^{(k)} = f\left(z_i^{(k)}\right)$$

- Nuevos parámetros entrenables por neurona

$$\alpha_i^{(k)} \qquad \beta_i^{(k)}$$

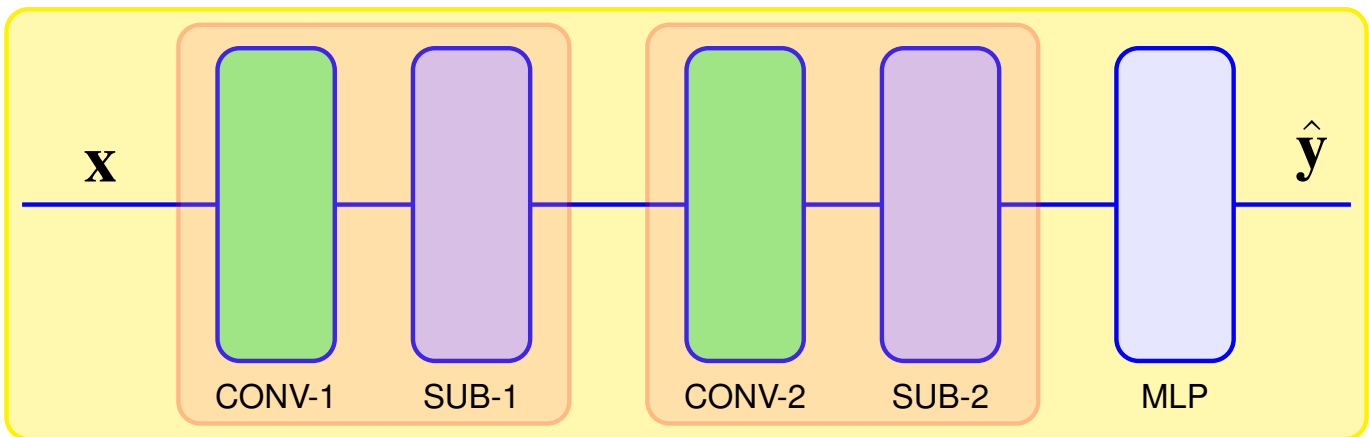
Batch normalization



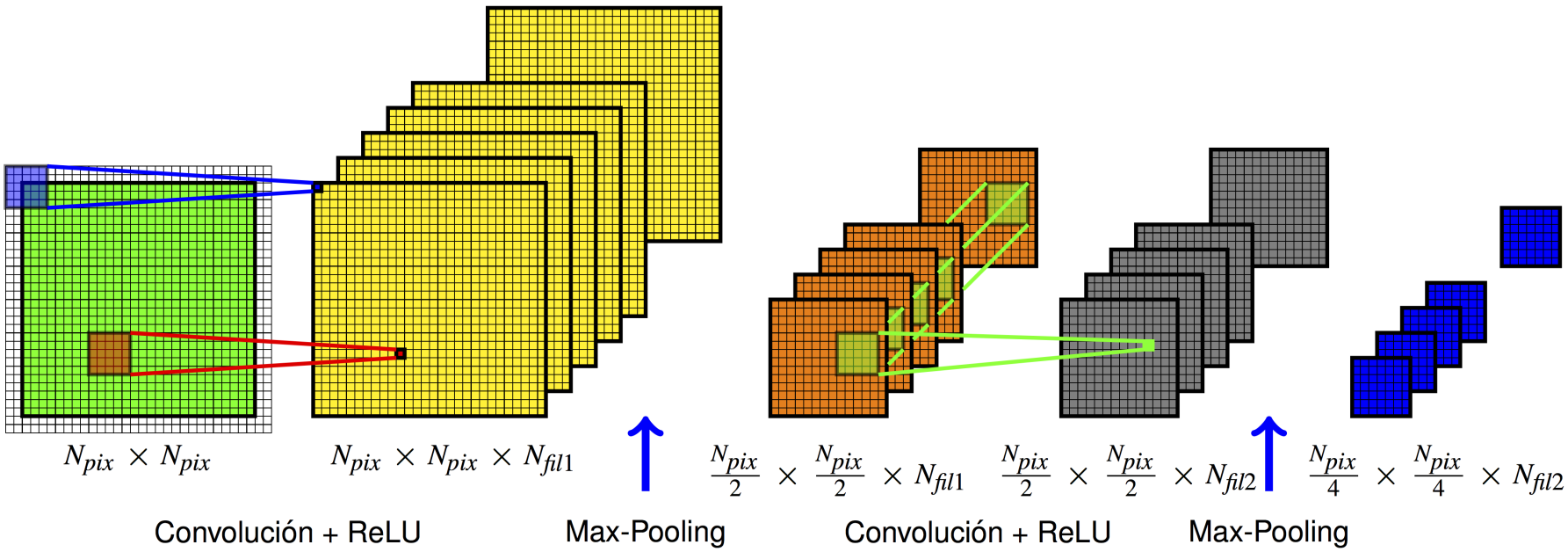
Redes Convolucionales (CNNs)

Redes convolucionales

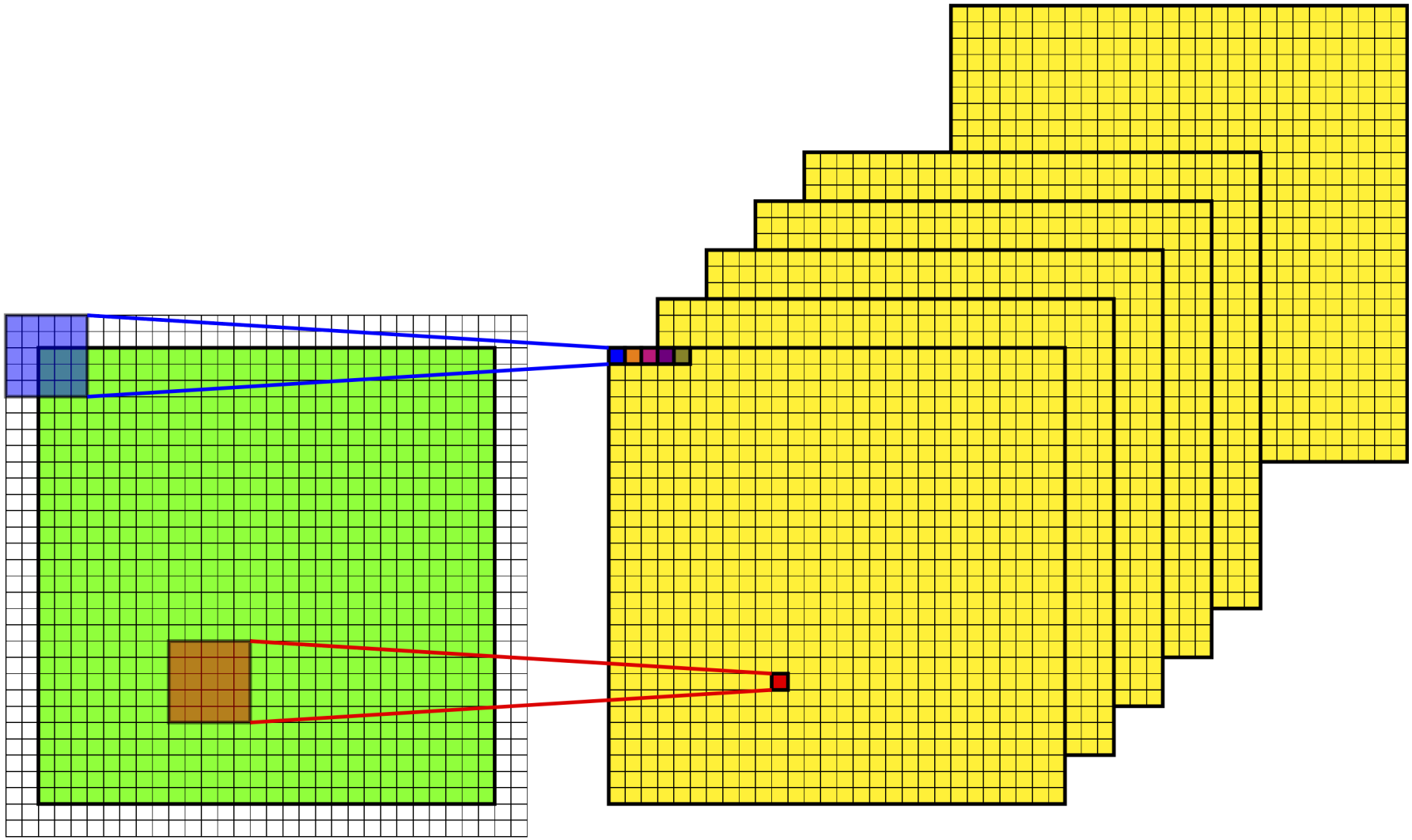
- Diseñadas inicialmente para procesado de imágenes
  - Correlación espacial (a nivel local)
- Redes hacia delante con varios tipos de etapas
  - Capas de convolución
    - ★ Explotan la correlación espacial con filtros convolutivos
    - ★ Habitualmente combinada con activaciones ReLU
  - Capas de submuestreo (tipo *max-pooling*)
    - ★ Reducen la carga computacional
    - ★ Proporcionan cierta invariancia a la traslación
  - Capas convencionales (MLPs)
    - ★ Etapa final para la tarea de clasificación
      - Las etapas previas se pueden interpretar como extractores de características



Convolutional Neural Network (CNN)



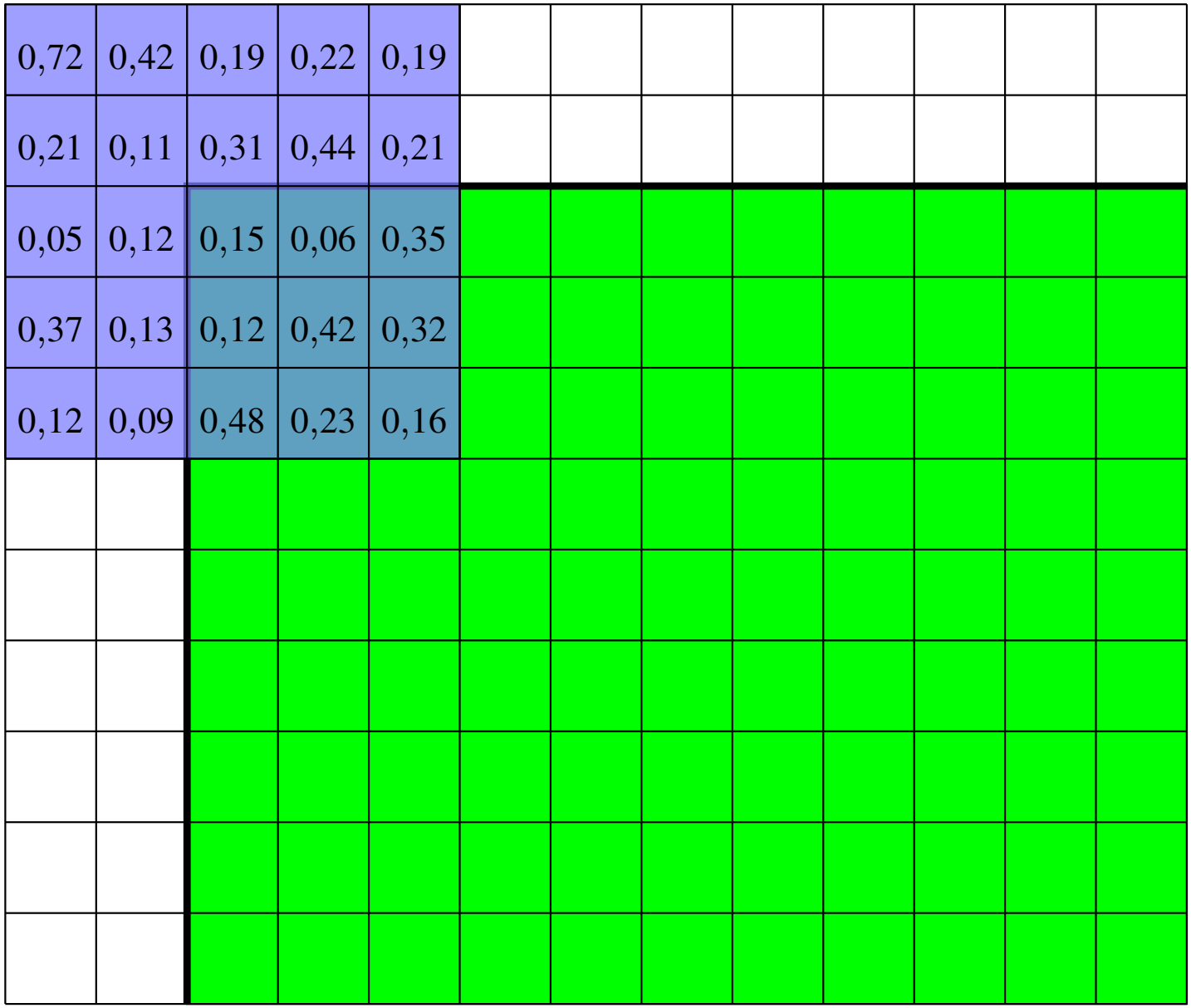
Convolución (con "*padding*")



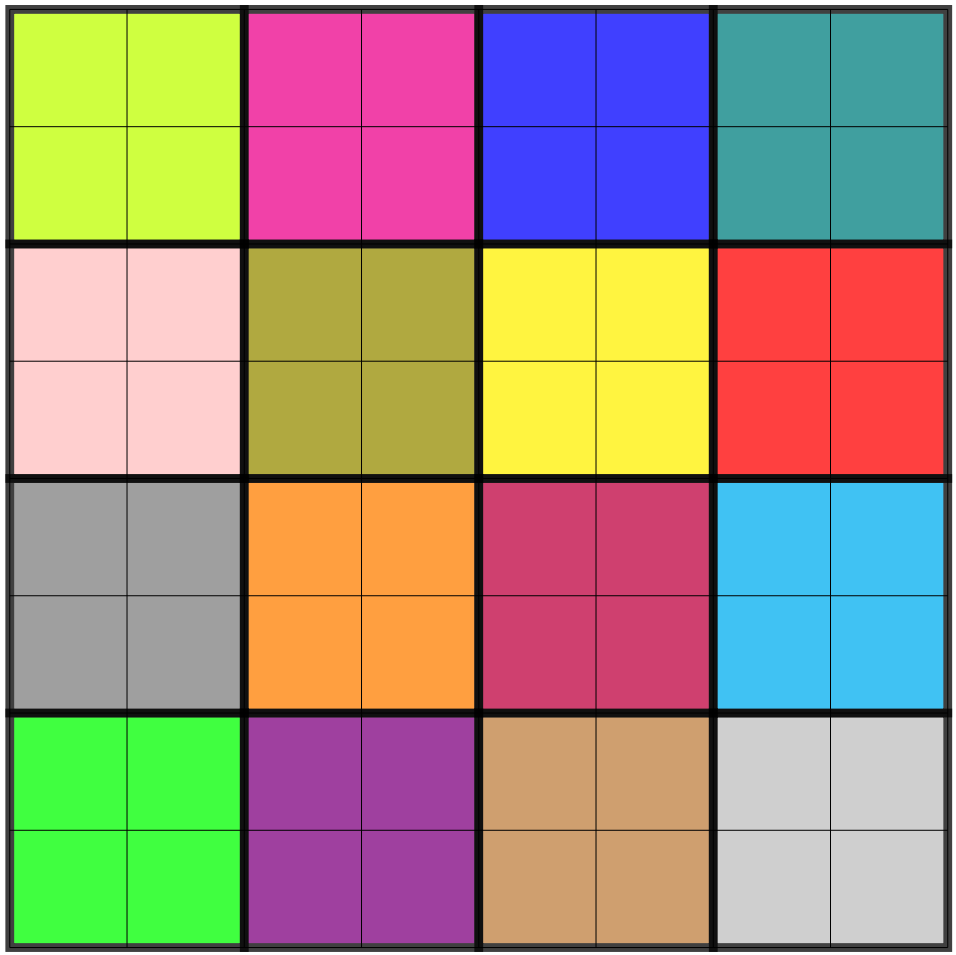
$N_{pix} \times N_{pix}$

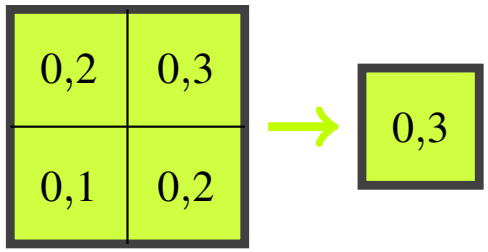
$N_{pix} \times N_{pix} \times N_{fil}$

Filtro de convolución (5×5)



Max-Pooling (Ejemplo: 2×2)





Maximum Value

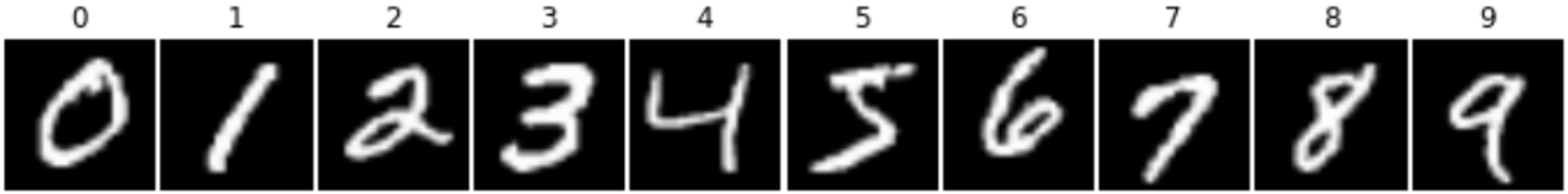
Conclusiones
y
Líneas actuales de investigación

Aprendizaje profundo para clasificación

- Las redes neuronales son una alternativa para la clasificación de patrones
- Las redes profundas están en el estado del arte para este problema
- Gran desarrollo e impacto en los últimos años
  - Técnicas de inicialización avanzadas
    - ★ Autocodificadores, RBMs,...
  - Técnicas de regularización
    - ★ Drop-out, drop-connect, batch-normalization,...
  - Nuevas funciones de activación
    - ★ ReLU, eLU, LReLU,...
  - Diseño de redes específicas para algunas aplicaciones
    - ★ CNNs (imágenes), LSTMs (series temporales),...

Ejemplo de aplicación

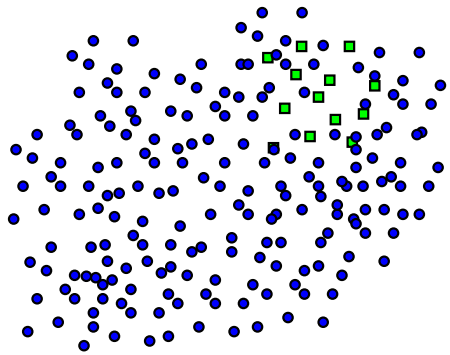
- Las CNNs son la solución de referencia para clasificación en imágenes
- Bibliografía: Multiples resultados sobre la base de datos MNIST



- Clase práctica sobre aprendizaje profundo (Python Notebook)
  - Aplicación de varios métodos a una versión reducida de la base de datos MNIST
  - Diseño de una red neuronal convencional (no profunda)
  - Diseño de una red de autocodificadores apilados (*SDA: Stacked Denoising Autoencoders*)
    - ★ Entreno de autocodificadores
    - ★ Uso de autocodificadores para la construcción de una red profunda
  - Redes profundas con Drop-out
  - Redes convolucionales (*CNNs: Convolutional Neural Networks*)

Problemas actuales de interés en clasificación de patrones

- Problemas desequilibrados (*imbalanced*)
  - Número de muestras muy diferente entre clases
    - ★ Problemas para la detección de las clases minoritarias



- Problemas con costes dependientes
  - Costes dependientes de la clase
  - Costes dependientes del ejemplo
- Problemas de clasificación ordinal
  - Las clases tienen una estructura ordinal
    - ★ La importancia de los errores depende de la distancia entre clases
- Posibles soluciones
  - Formulación Bayesiana (probabilidades de clase, costes de error)
  - Técnicas de diversidad

Referencias

Tutoriales sobre aprendizaje profundo

Bengio, Y. (2009).

Learning deep architectures for AI.
Foundations and Trends in Machine Learning, 2:1–127.

Deng, L. and Yu, D. (2014).

Deep learning: methods and applications.
Foundations and Trends in Signal Processing, 7:197–387.

LeCun, Y., Bengio, Y., and Hinton, G. (2015).

Deep learning.
Nature, 521:436–444.

Autocodificadores y redes de creencia

Vincent, P., Larrochelle, H., Lajoie, I., Bengio, Y., and Manzagol, P.-A. (2010).

Stacked denoising autoencoders: Learning useful representations in a deep network with a local denoising criterion.
Journal of Machine Learning Research, 11:3371–3408.

Hinton, G., Osindero, S., and Teh, Y. (2006).

A fast learning algorithm for deep belief nets.
Neural Computation, 18(7):1527–1554.

Redes convolucionales

Fukushima, K. (1988).

Neocognitron: A hierarchical neural network capable of visual pattern recognition.
Neural Networks, 1(2):119–130.

LeCun, Y., Boser, B., Denker, J., Henderson, D., Howard, R., Hubbard, W., and Jackel, L. D. (1989).

Backpropagation applied to handwritten zip code recognition.
Neural Computation, 1(4):541–551.

Referencias (II)

Drop-out y Batch normalization

Srivastava, N., Hinton, G., Krizhevsky, A., Sutskever, I., and Salakhutdinov, R. (2014).

Dropout: A simple way to prevent neural networks from overfitting.
Journal of Machine Learning Research, 15:1929–1958.

Ioffe, S. and Szegedy, C. (2015).

Batch normalization: Accelerating deep network training by reducing internal covariate shift.
ArXiv e-prints 1502.03167 [cs.LG].

Redes específicas

Hochreiter, S. and Schmidhuber, J. (1997).

Long short-term memory.
Neural Computation, 9:1735–1780.

Líneas de investigación de interés

Japkowicz, N. (2000).

The class imbalance problem: Significance and strategies.
In *Proceedings of the 2000 Intl Conf. on Artificial Intelligence (ICAI)*, pages 111–117, Las Vegas, USA.

He, H. and Garcia, E. A. (2009).

Learning from imbalanced data.
IEEE Transactions on Knowledge and Data Engineering, 21(9):1263–1284.

Elkan, C. (2001).

The foundations of cost-sensitive learning.
In *Proceedings of the 17th International Joint Conference on Artificial Intelligence*, volume 2, pages 973–978.

Gutierrez, P. A., Pérez-Ortiz, M., Sánchez-Monedero, J., Fernández-Navarro, F., and Hervás-Martínez, C. (2016).

Ordinal regression methods: Survey and experimental study.
IEEE Transactions on Knowledge and Data Engineering, 28(1):127–146.

Referencias (III)

Líneas de investigación de interés

González, S., García, S., Lázaro, M., Figueiras-Vidal, A. R., and Herrera, F. (2017).

Class switching according to nearest enemy distance for learning from highly imbalanced data.
Pattern Recognition, 70:12–24.

Lázaro, M., Hayes, M. H., and Figueiras-Vidal, A. R. (2018).

Training neural network classifiers through Bayes risk minimization applying unidimensional Parzen windows.
Pattern Recognition, 77:204–215.

Lázaro, M., Herrera, F., and Figueiras-Vidal, A. R. (2020).

Ensembles of cost-diverse Bayesian neural learners for imbalanced binary classification.
Information Sciences, (520):31–45.

Lázaro, M. and Figueiras-Vidal, A. R. (2021).

A Bayes-risk minimization machine for example-dependent cost classification.
IEEE Transactions on Cybernetics, 51(7):3524–3534.

Lázaro, M. and Figueiras-Vidal, A. R. (2023).

Neural network for ordinal classification of imbalanced data by minimizing a bayesian cost.
Pattern Recognition, 137.

Mediavilla-Relaño, J., Lázaro, M., and Figueiras-Vidal, A. R. (2023).

Imbalance example-dependent cost classification: A bayesian based method.
Expert Systems with Applications, 213, Part B.