Aprendizaje Profundo

(Primera sesión de demostraciones)

Actuales avances en Máquinas de Aprendizaje

(Seminario FOM)

.

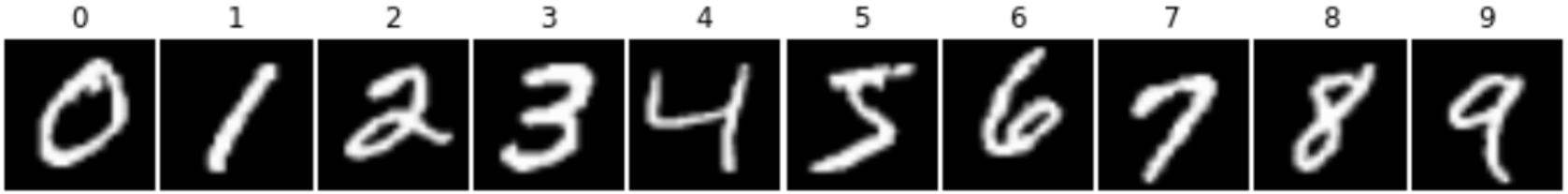
http://www.tsc.uc3m.es/~mlazaro/Docencia/FOM.html

Marcelino Lázaro

Universidad Carlos III de Madrid

Índice de contenidos

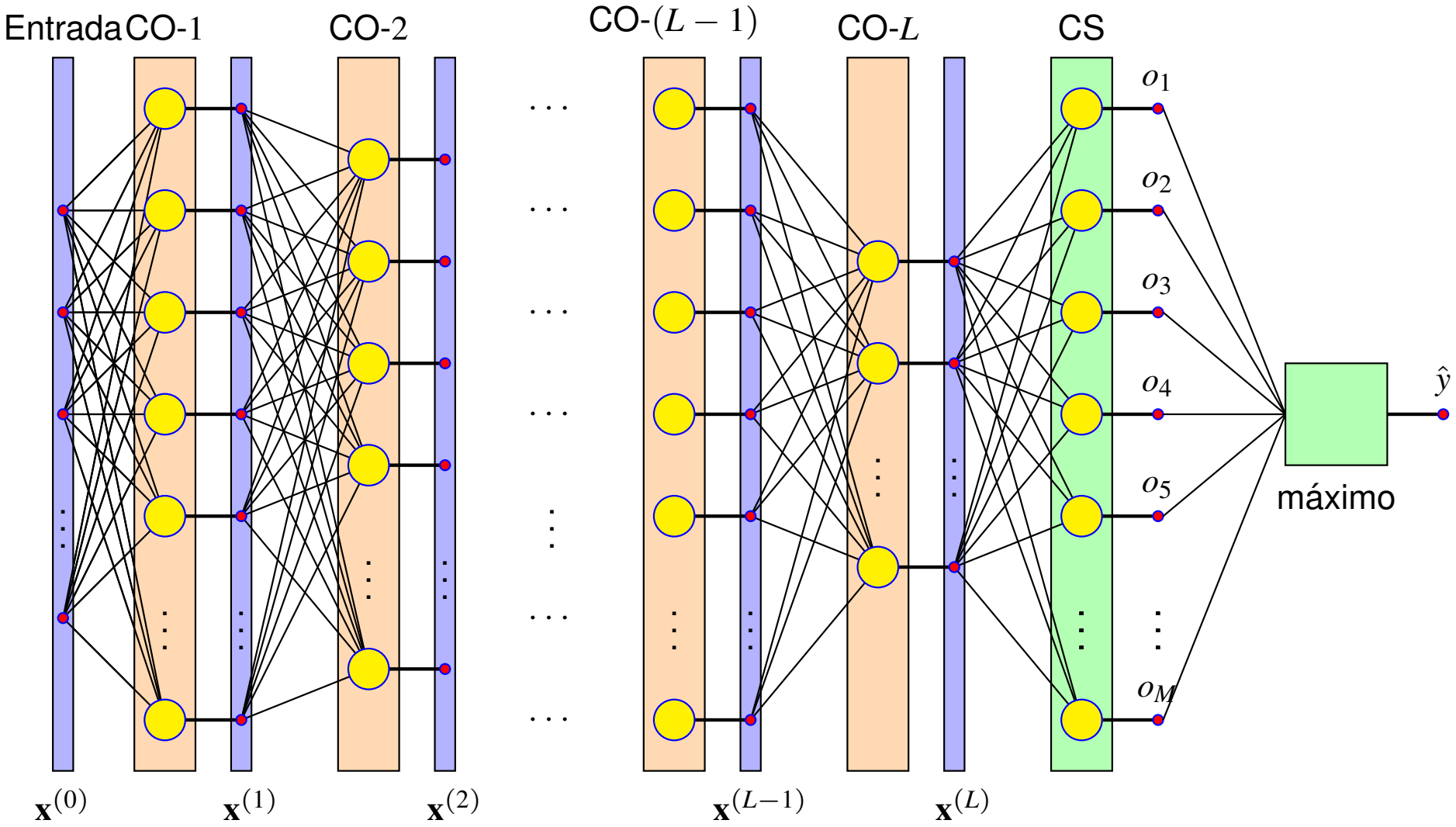
- Aprendizaje profundo (*"deep learning"*) para clasificación



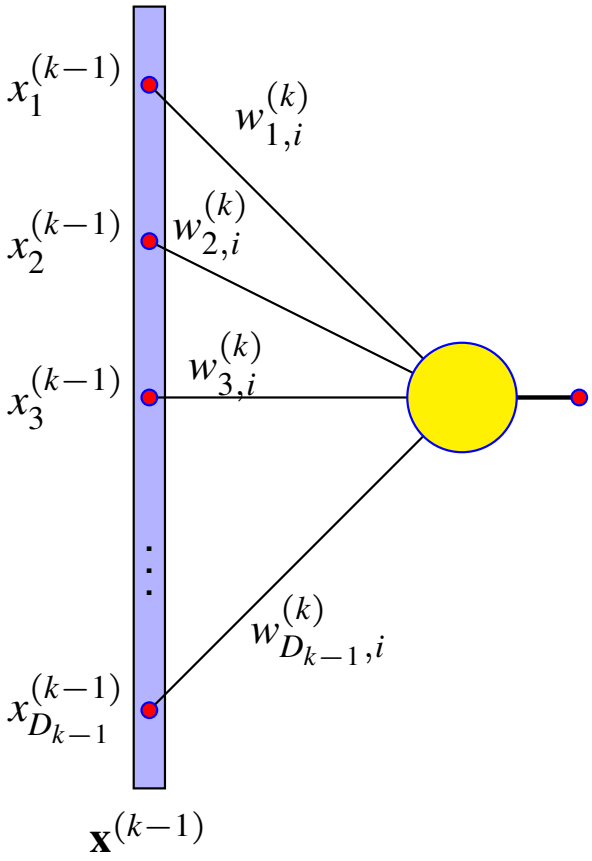
  - Aplicación a una versión reducida de la base de datos MNIST
- Diseño de una red de autocodificadores apilados (*SDA: Stacked Denoising Autoencoders*)
  - Entreno de autocodificadores
  - Uso de autocodificadores para la construcción de una red profunda
- Redes profundas con Drop-out
- Redes convolucionales (*CNN: Convolutional Neural Networks*)
- Breve descripción de las funciones de aprendizaje disponibles

Aprendizaje profundo (para clasificación)

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



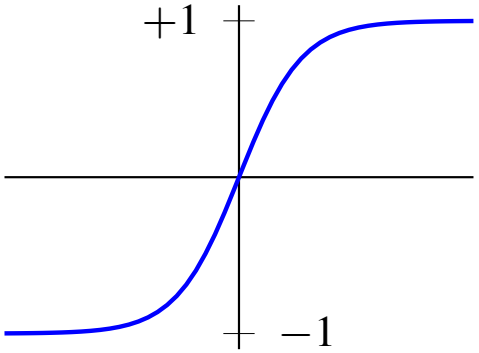
Neurona - Capa k



$$x_i^{(k)} = f\left(\sum_{j=1}^{D_{k-1}} w_{j,i}^{(k)}\, x_j^{(k-1)} + w_b^{(k)}\right)$$

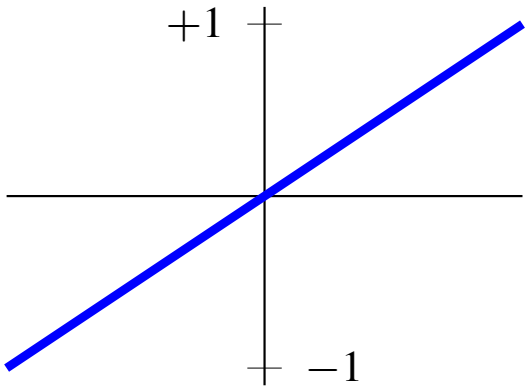
$f(\cdot) \equiv$ función de activación

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



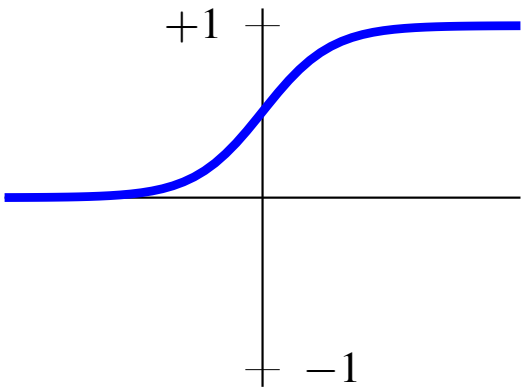
Funciones de activación más frecuentes

$f(x) = x$



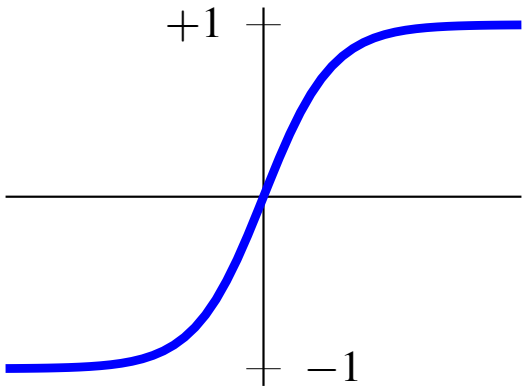
Lineal

$f(x) = \frac{1}{1+e^{-x}}$



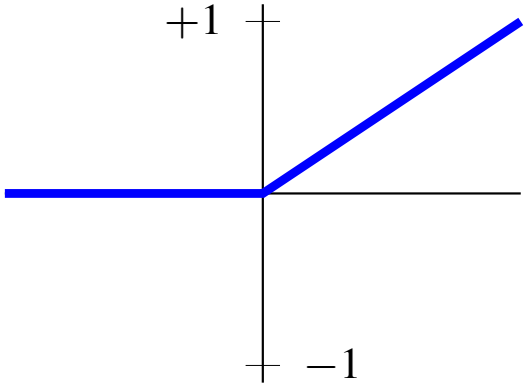
Logística

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



Tangente Hiperbólica

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

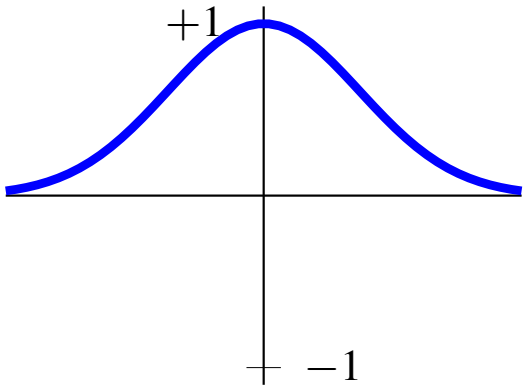


Lineal rectificada (ReLU)

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

Softmax

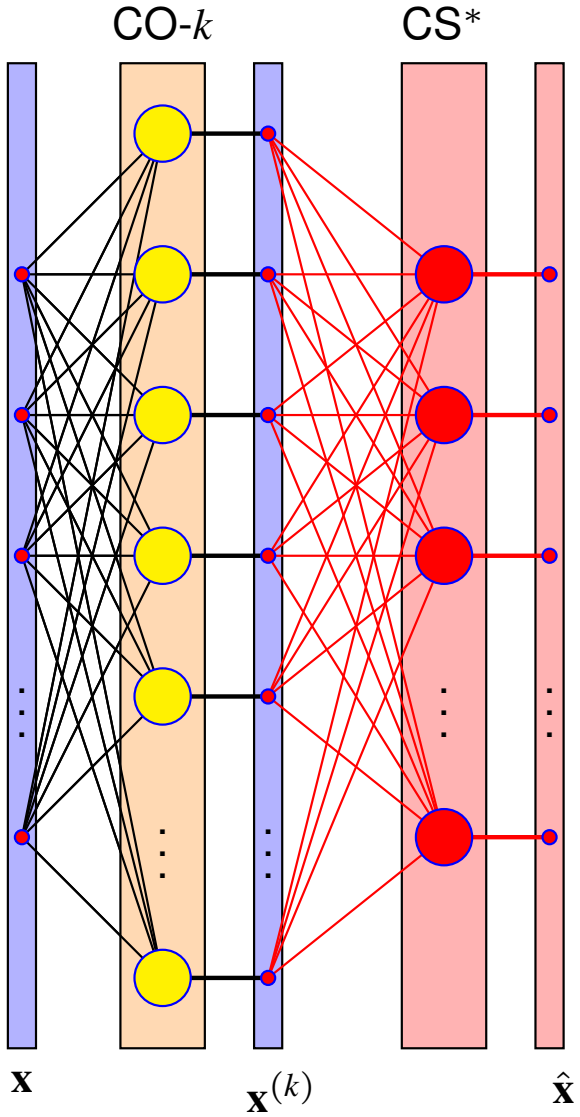
$f(x) = e^{-\frac{x^2}{2\sigma^2}}$



Gausiana

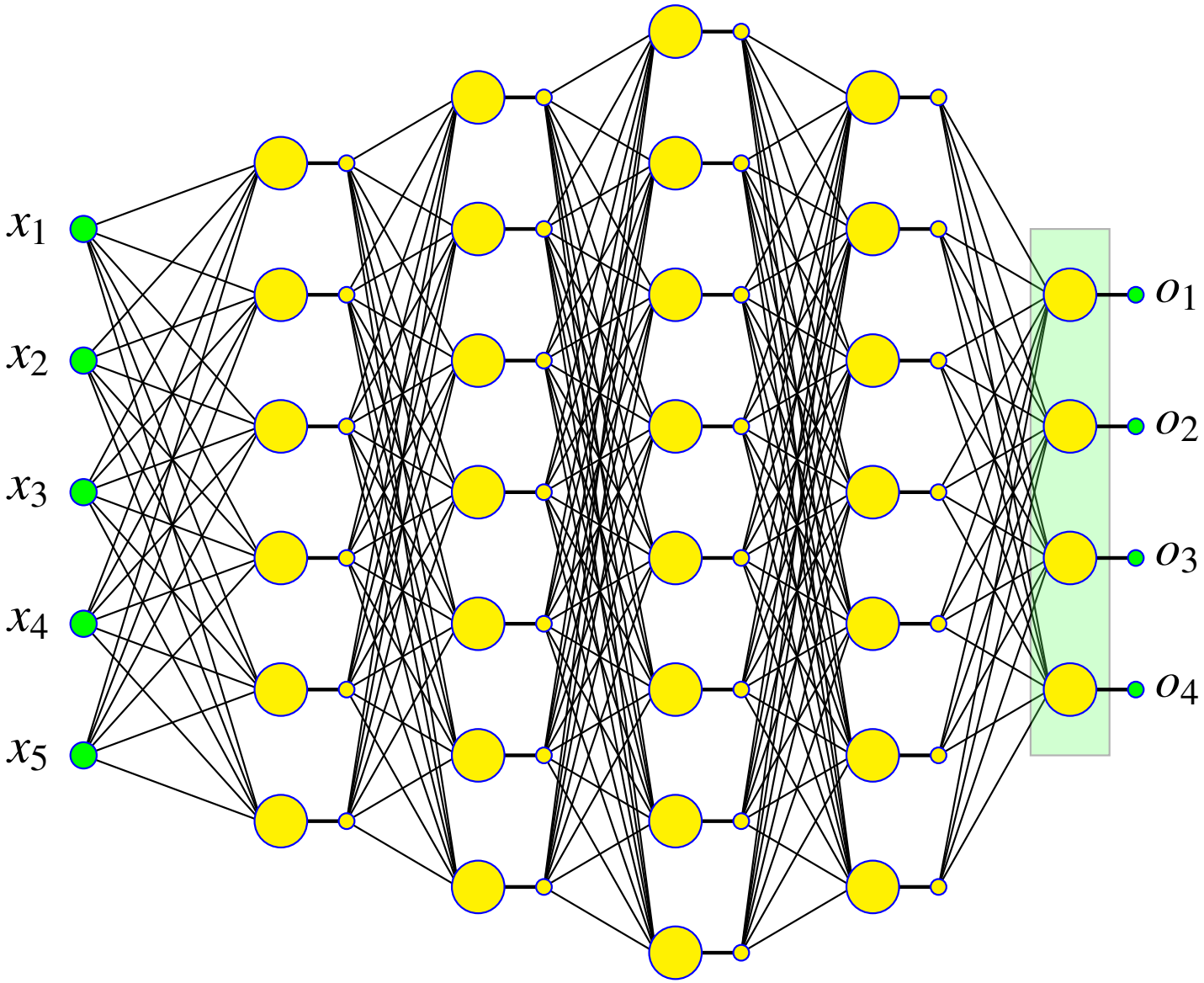
Stacked Denoising Autoencoders

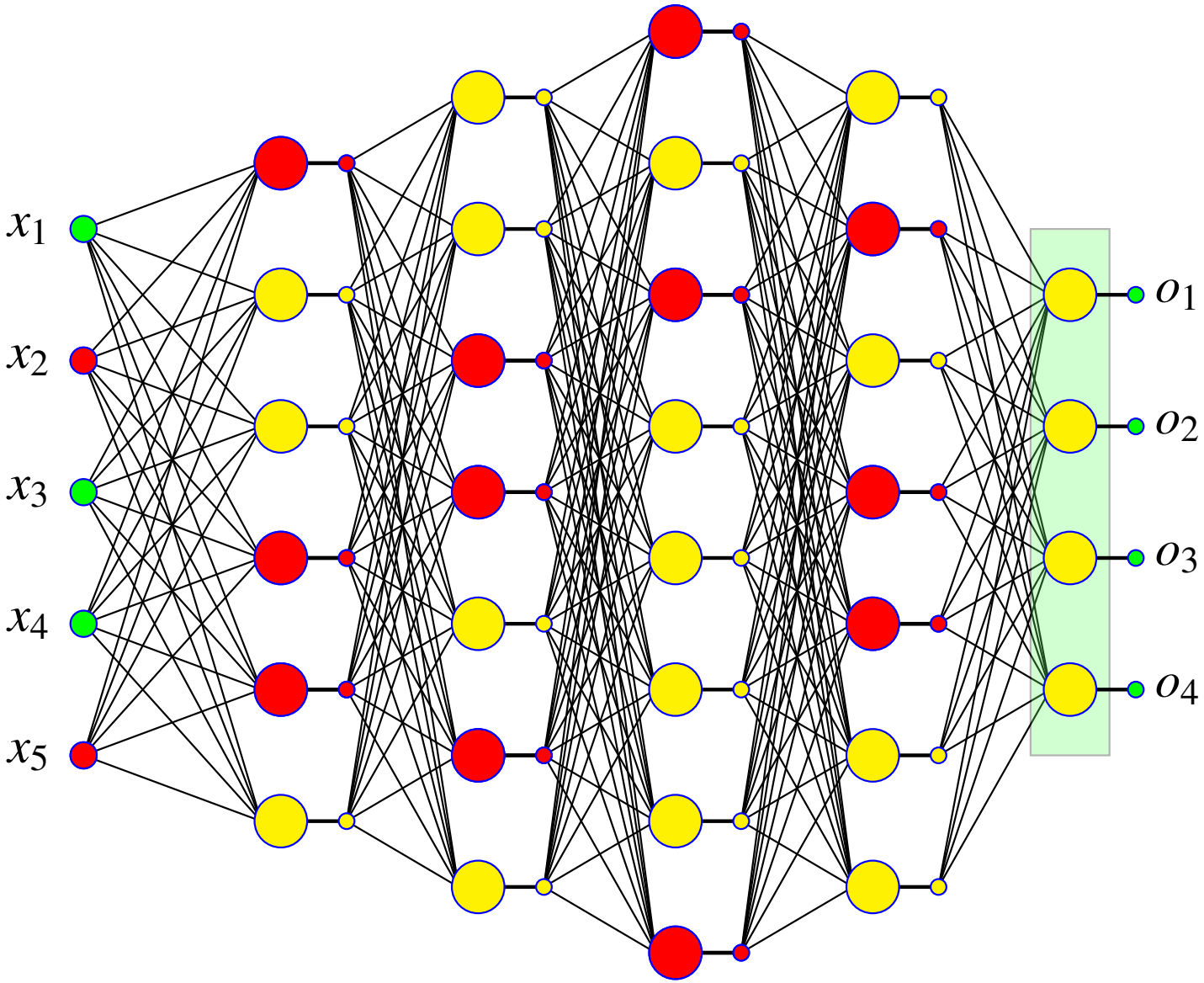
Autocodificadores (*"autoencoders"*)

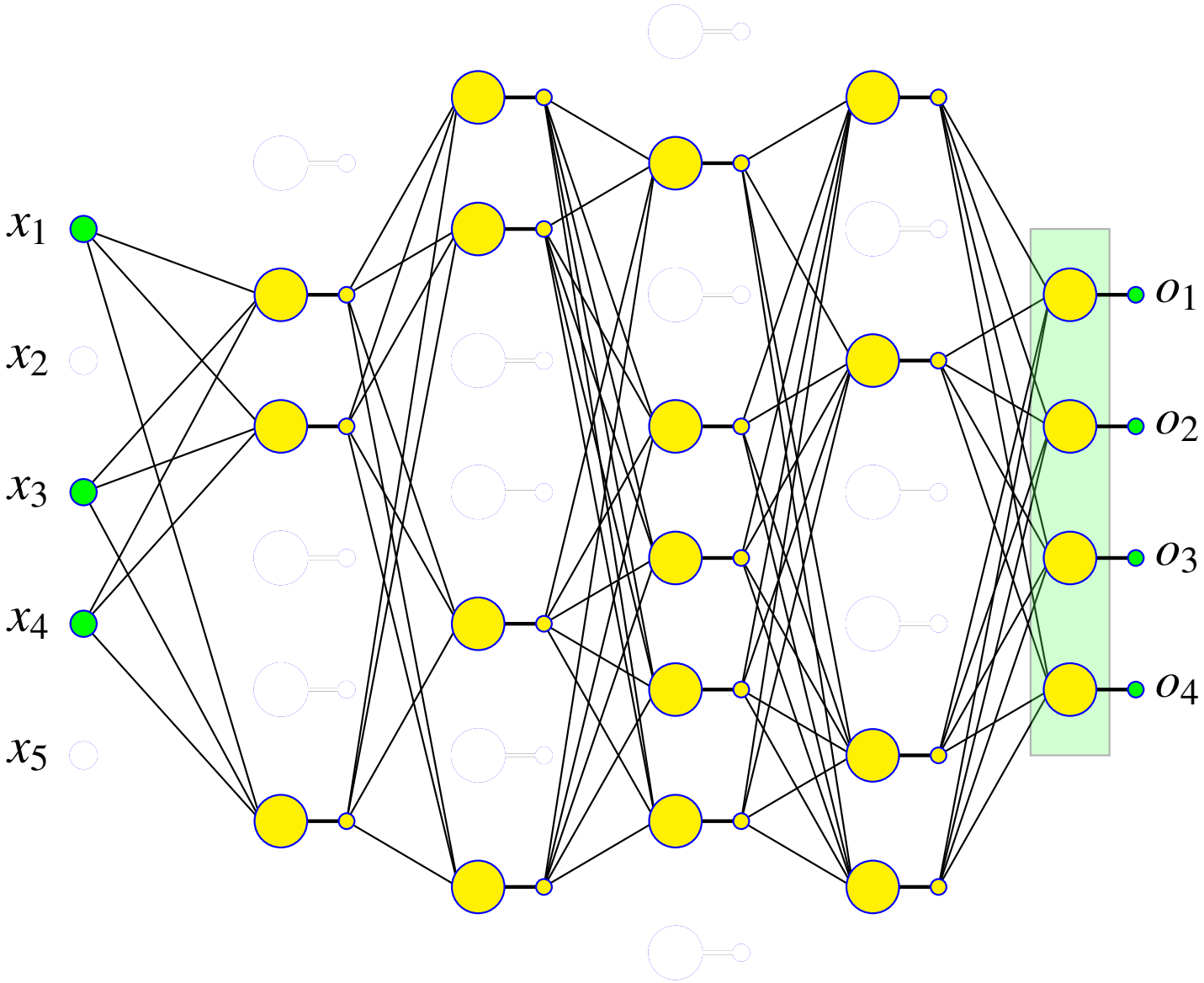


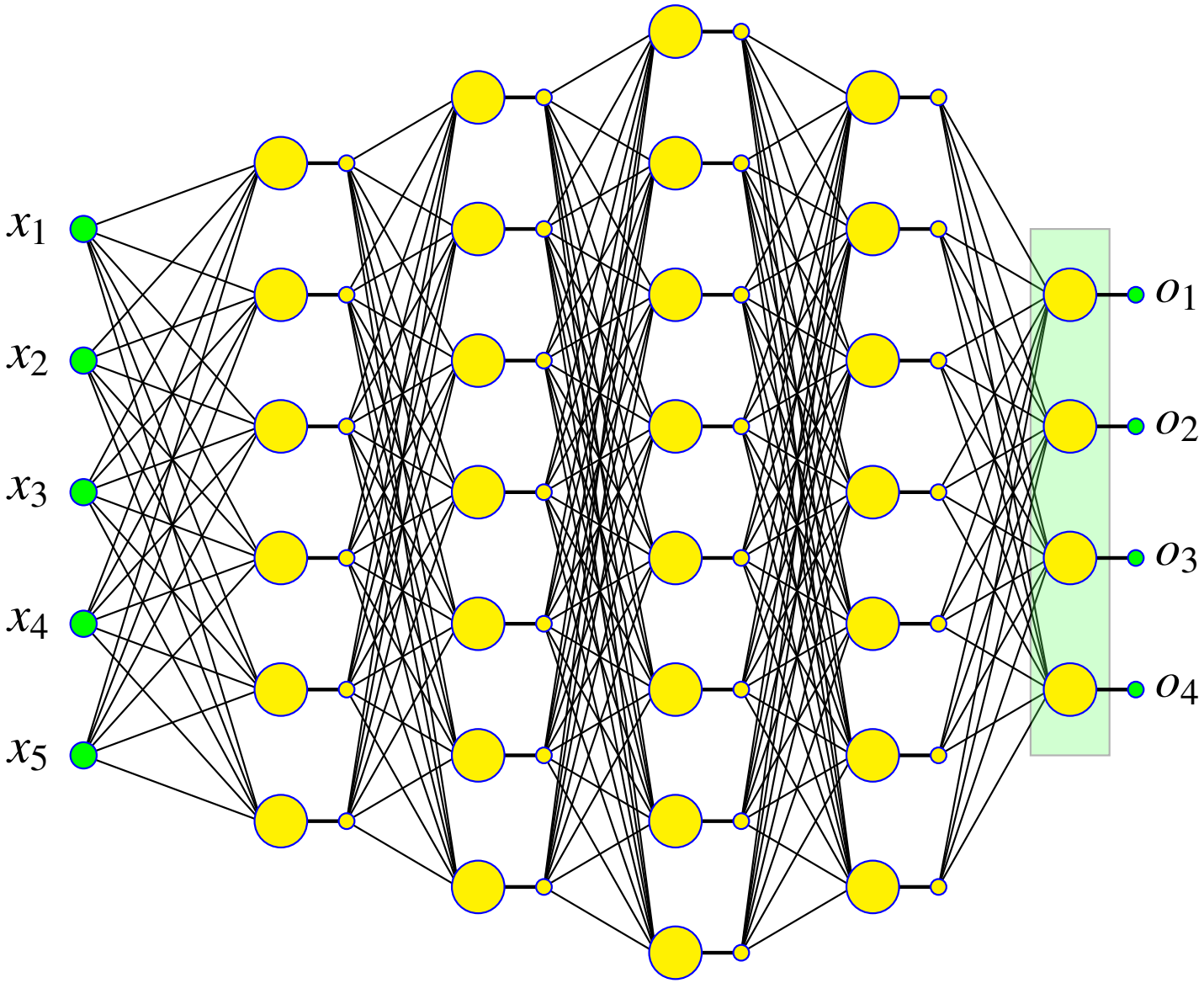
- El entrenamiento de una capa oculta se realiza para obtener una proyección de su entrada sin pérdida de información
  - Entrada: $\mathbf{x} = \mathbf{x}^{(k-1)}$
  - Salida deseada: $\hat{\mathbf{x}} = \mathbf{x}^{(k-1)}$
- La capa de salida se descarta
  - Se usa como mecanísmo para evitar la pérdida 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
    - ★ Entrada: $\mathbf{x} = \mathbf{x}^{(k-1)} + \mathbf{n}$
    - ★ Salida deseada: $\hat{\mathbf{x}} = \mathbf{x}^{(k-1)}$

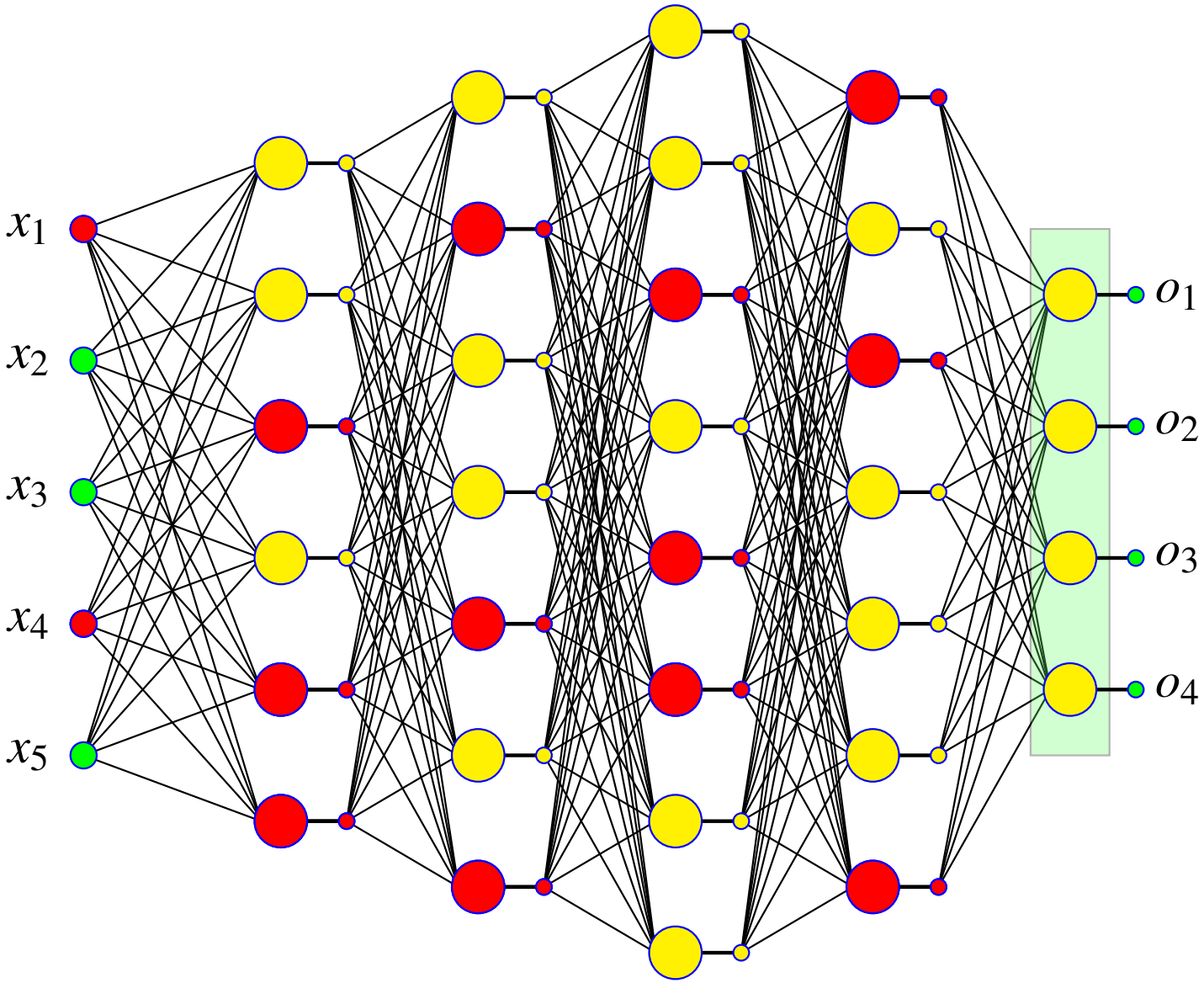
Redes profundas con Drop-Out

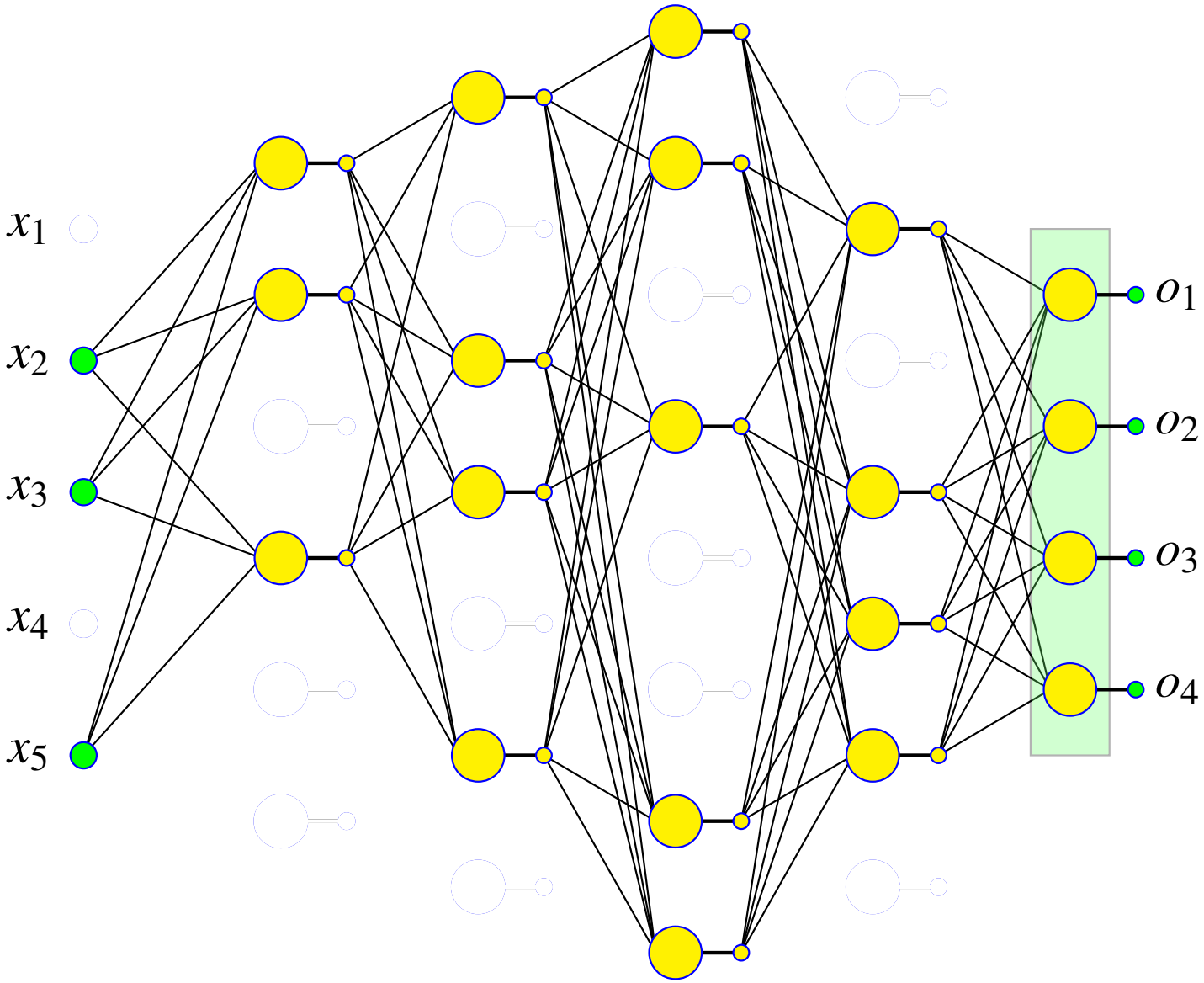






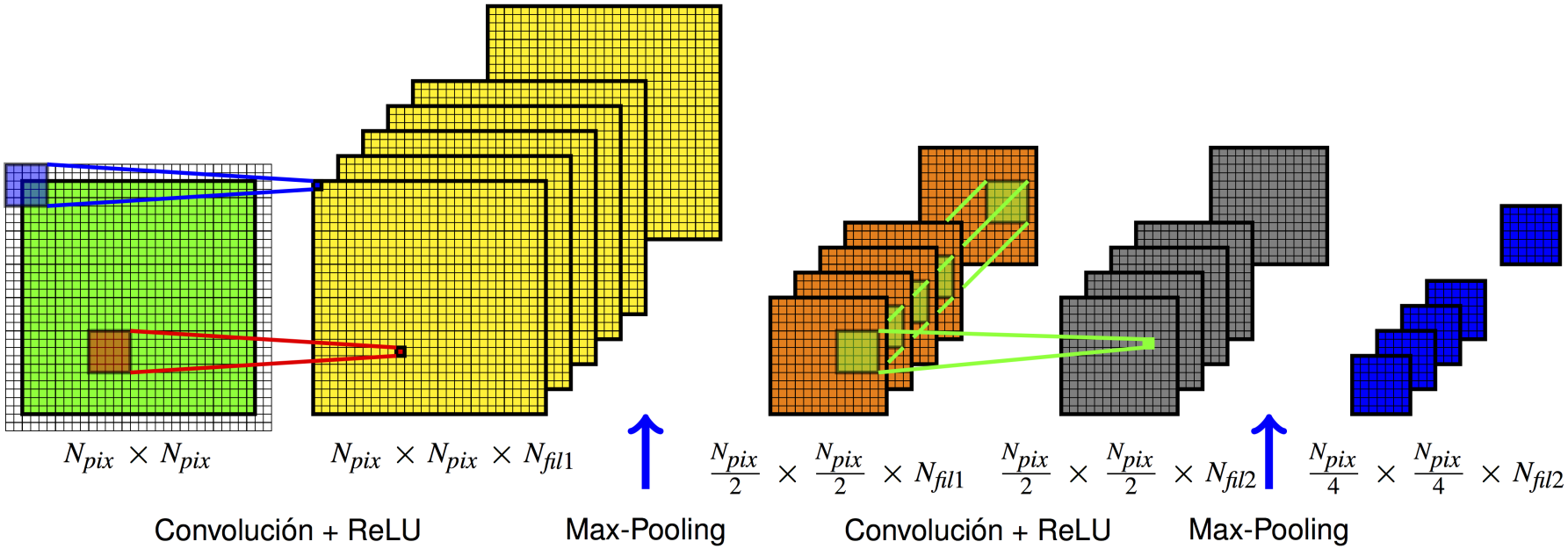




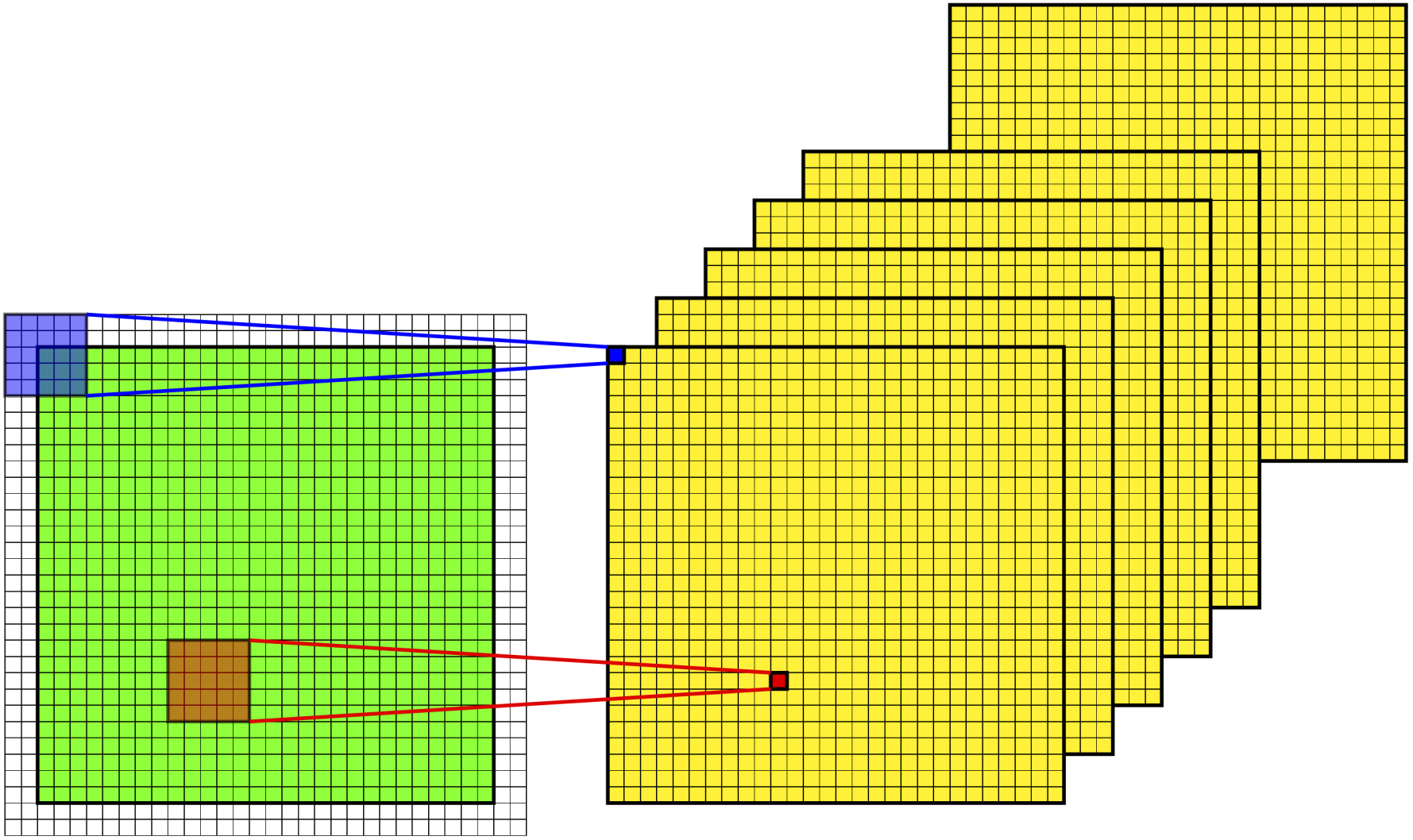


Redes Convolucionales

Convolutional Neural Network (CNN)

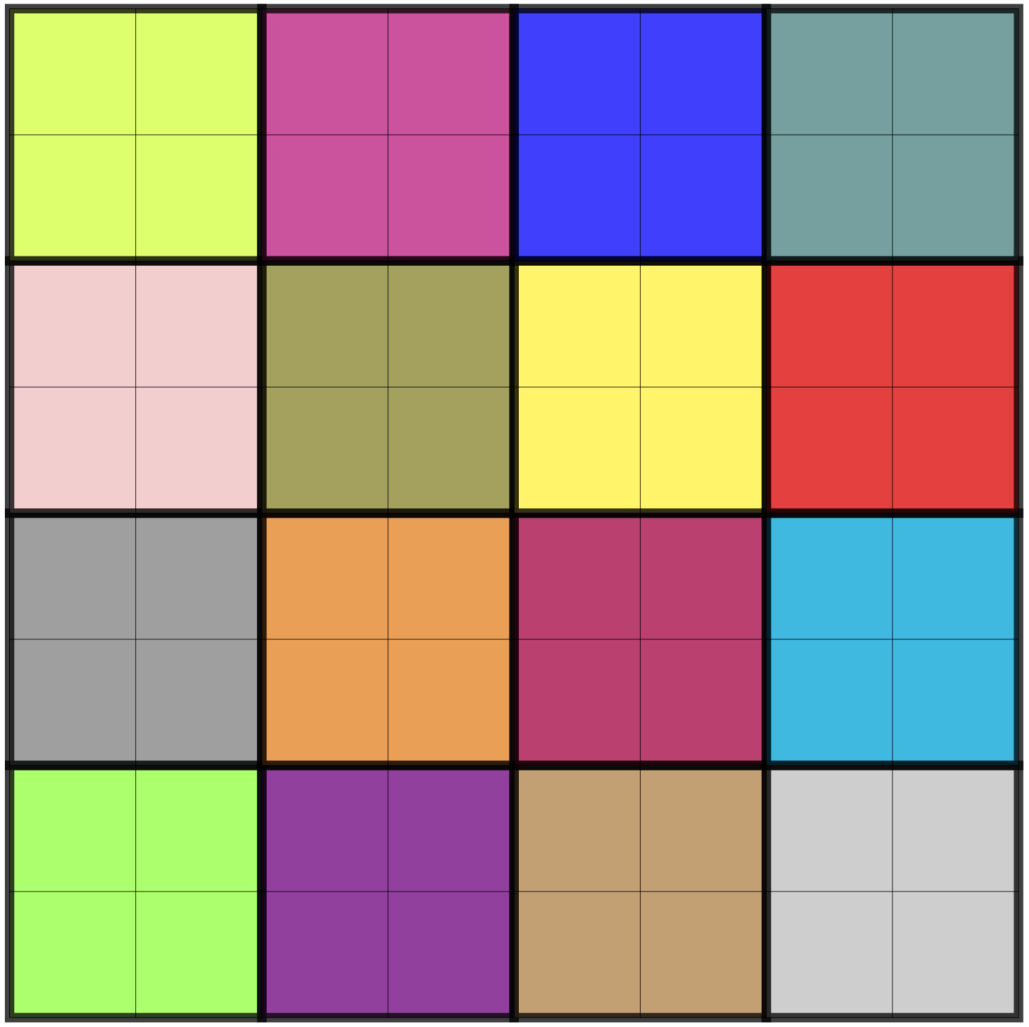


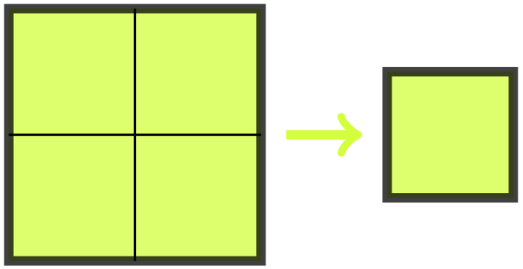
Convolución (con "*padding*")



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

Max-Pooling





Maximum Value

Python

Descripción de las funciones presentes en *metodosDL.py*

Funciones en Python (evaluación MLPs)

- (o,H)=mlp(x,W,tAct)
  - x: patrones de entrada (Ne x Np)
  - W: pesos de la red: lista the No + 1 elementos
    - Capa oculta 1 (Nn1 x (Ne+1)
    - Capa oculta i (Nni x (Nn(i-1)+1)
    - Capa de salida (Ns x Nnu+1)
  - tAct : parámetro que define el tipo de funciones de activación de las diferentes capas (vector de No+1 elementos)
    - 0: lineal
    - 1: tanh
    - 2: logistic
    - 3: rectified linear (ReLU)
    - 4: softmax

  Nni: número de neuronas de la capa oculta i (si i=0, Nni=Ne)

  Nnu: número de neuronas de la última capa oculta

  No: número de capas ocultas
  - o: salida de la red neuronal (Ns x Np)
  - H: lista con las salidas de las capas ocultas (No elementos)

Funciones en Python (entrenamiento MLPs)

Entrenamiento de MLPs con distinto número de capas ocultas

- (ws,paso,coste,dWm)=entrena_mlp(x,y,W,Nepoch)
  - x: patrones de entrada (Ne x Np)
  - W: lista con los pesos de la red (No+1 elementos)
  - Nepoch: lista con el número de iteraciones y el tamaño del mini-batch
    - Nepoch=[Niter,Nbatch]
- Parámetros adicionales
  - fCoste: función de coste a minimizar ('mmse', 'entropia', 'wmmse')
  - optimizador: método de optimización ('gradiente', 'momento')
  - tAct: tipo de funciones de activación de las capas de la red
  - pDO: probabilidad de Drop-Out de la entrada y capas ocultas (lista No+1 el.)
  - paso: parámetros del paso de adaptación
    - Gradiente: paso=[mu,muCrec,muDec]
    - Momento: paso=[mu,momento]
  - flagEvo: True para que se obtenga el coste cada iteración

Funciones en Python (entrenamiento AE)

Funciones para el entrenamiento de autocodificadores (AE)

- (we,wd,coste,mu)=entrena_AE(x,we,wd,Nepoch,tAct,mu)
  Entrenamiento de un autocodificador
- (we,wd,coste,mu)=entrena_AEtied(x,we,wd,Nepoch,tAct,mu)
  Entrenamiento de un autocodificador con pesos “atados”
- (we,wd,coste,mu)=entrena_DAE(x,we,wd,Nepoch,tAct,mu,pRuido)
  Entrenamiento de un autocodificador con ruido
- (we,wd,coste,mu)=entrena_DAEtied(x,we,wd,Nepoch,tAct,mu,pRuido)
  Entrenamiento de un autocodificador con ruido y pesos “atados”
  - we: pesos del codificador (Nn x Ne+1)
  - wd: pesos del decodificador (Ne x Nn+1)
  - pRuido: probabilidad de ruido (de poner una entrada a cero)

Resultados (Aproximados)

Resultados aproximados esperables

- Resolución 10×10 (2.000 patrones de entrenamiento)
  - Red neuronal de 1 capa oculta: $\approx 86{,}5\,\%$
  - Red neuronal de 4 capas ocultas: $\approx 85\,\%$
  - Red neuronal profunda con autocodificadores: $\approx 88{,}5\,\%$
  - Red neuronal profunda con Drop-Out: $\approx 88\,\%$
    - ⋆ Pocas muestras de entrenamiento, y muchas capas ocultas en la configuración de pruebas
  - Red CNN: $\approx 91\,\%$
- Resolución 28×28
  - Red neuronal de 1 capa oculta: $\approx 92\,\%$
  - Red neuronal profunda con autocodificadores: $\approx 98\,\%$
  - Red neuronal profunda con Drop-Out: $\approx 98\,\%$
  - Red CNN: $\approx 99\,\%$