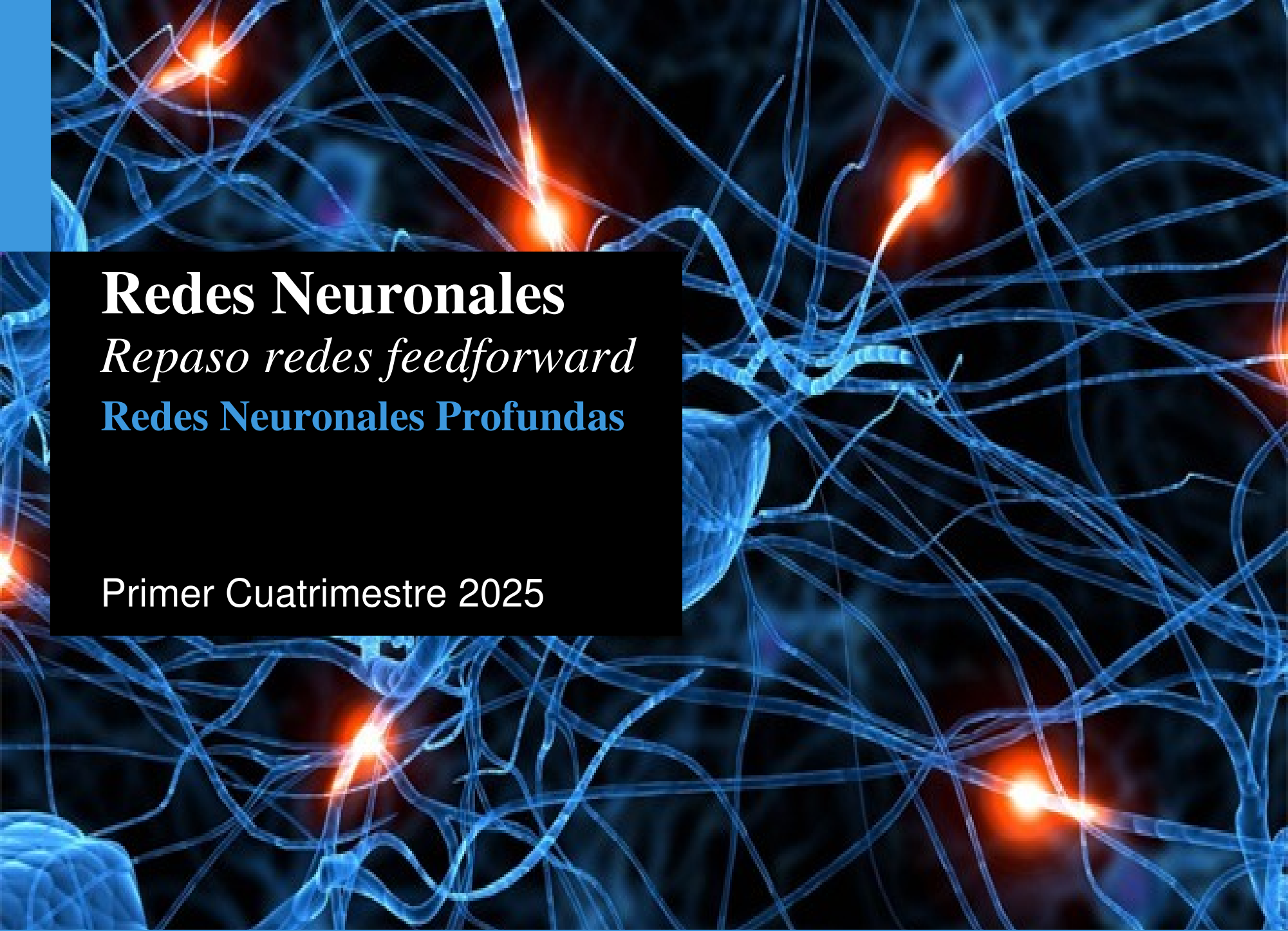


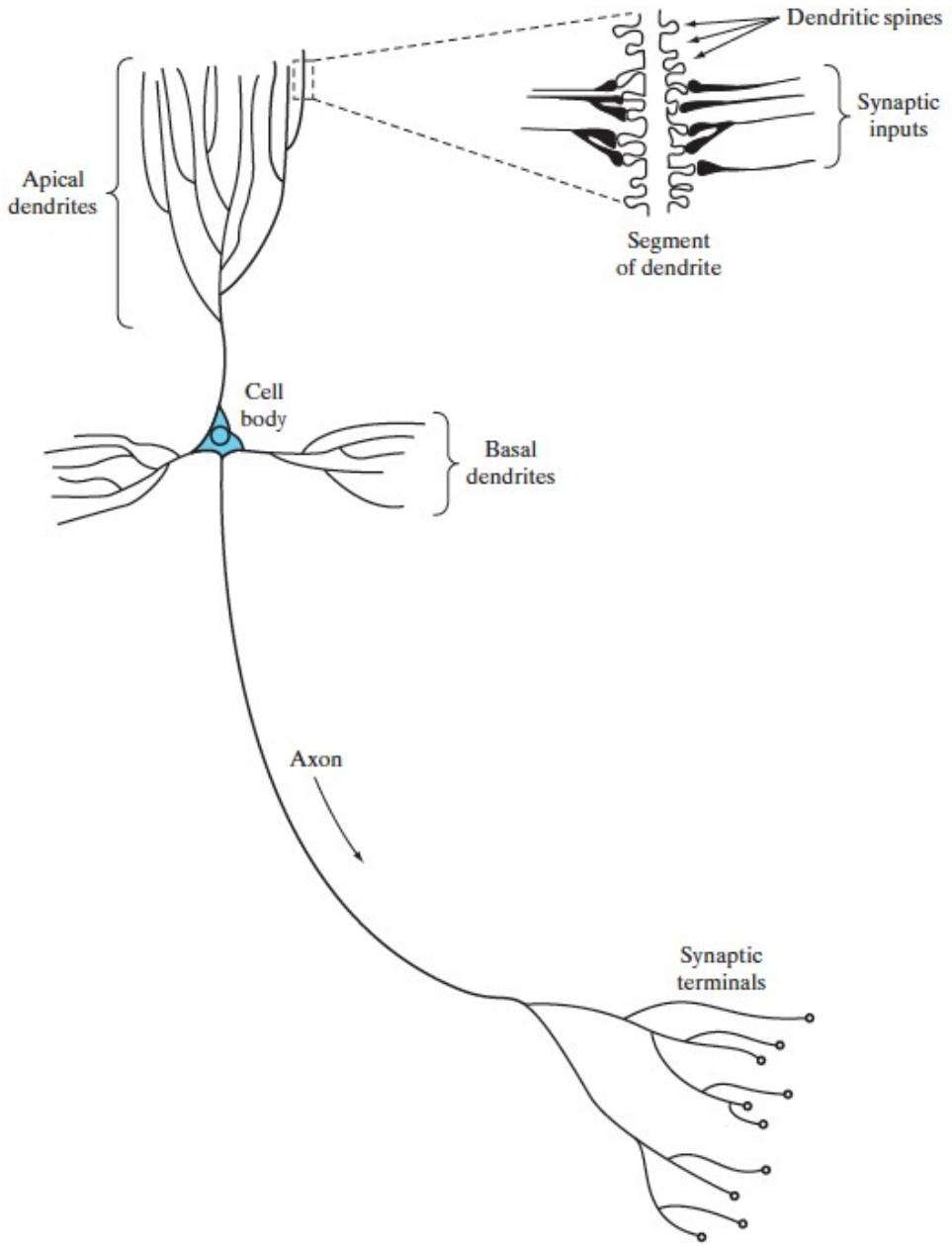
# Redes Neuronales

*Repaso redes feedforward*

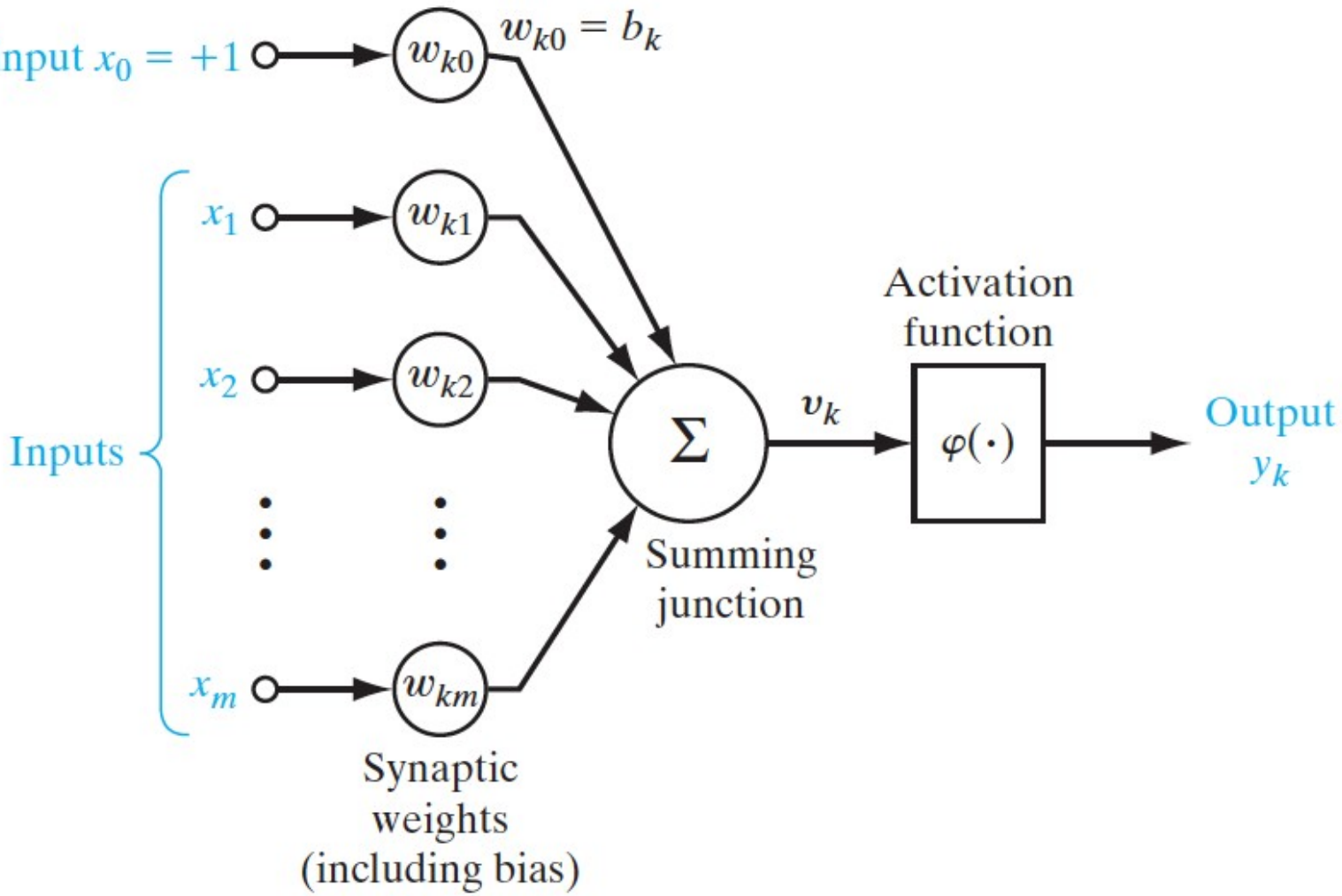
**Redes Neuronales Profundas**

Primer Cuatrimestre 2025

# Neurona biológica

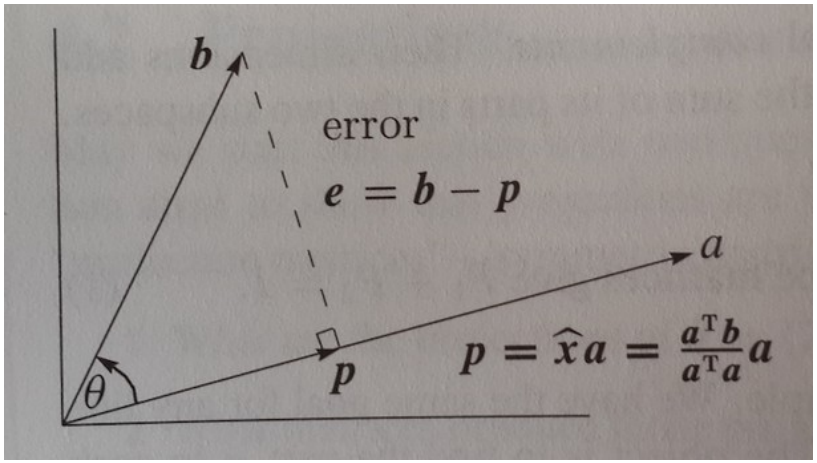


# Red Neuronal Artificial *Perceptrón Simple*




Red neuronal feed-forward de una sola capa.

# Las neuronas actúan como filtros




La activación de una neurona involucra el producto escalar del vector input y el vector de pesos. Si los vectores están normalizados, el producto escalar es la proyección del vector input sobre el vector de pesos. Entonces, la activación mide la similaridad entre el input y el vector de pesos. Esto nos permite ver a las neuronas como filtros lineales en el sentido de que la neurona es capaz de discriminar entre inputs que son similares a su vector de pesos (que tienden a generar activaciones positivas grandes) e inputs que son disímiles a su vector de pesos (que generan activaciones positivas pequeñas, nulas o negativas)

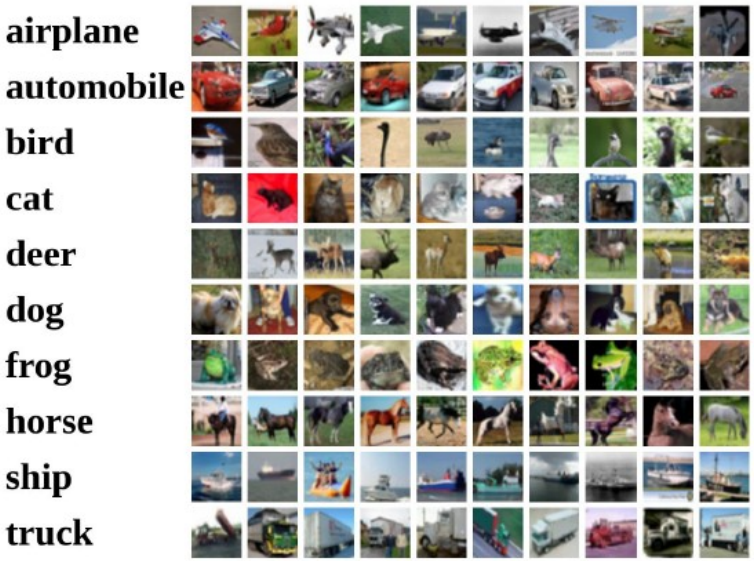
# El perceptrón simple es un clasificador lineal

En aprendizaje automático, el objetivo del aprendizaje supervisado es usar las características de un objeto para identificar a qué clase (o grupo) pertenece. Un clasificador lineal logra esto tomando una decisión de clasificación basada en el valor de una combinación lineal de sus características. Las características de un objeto son típicamente presentadas en un vector llamado vector de características. Si la entrada del clasificador es un vector de características $x$ , entonces el resultado de salida es

$$y = f(x^T \cdot w) = f(\sum x_j w_j)$$

donde $w$ es un vector real de pesos y $f$ es una función que convierte el producto escalar de los dos vectores en la salida deseada. El vector de pesos $w$ se aprende de un conjunto de muestras de entrenamiento. Entonces, el perceptrón simple es un tipo de clasificador lineal (hay otros tipos, como SVM, etc).

# Interpretando un clasificador lineal




$$f(x,W) = Wx + b$$

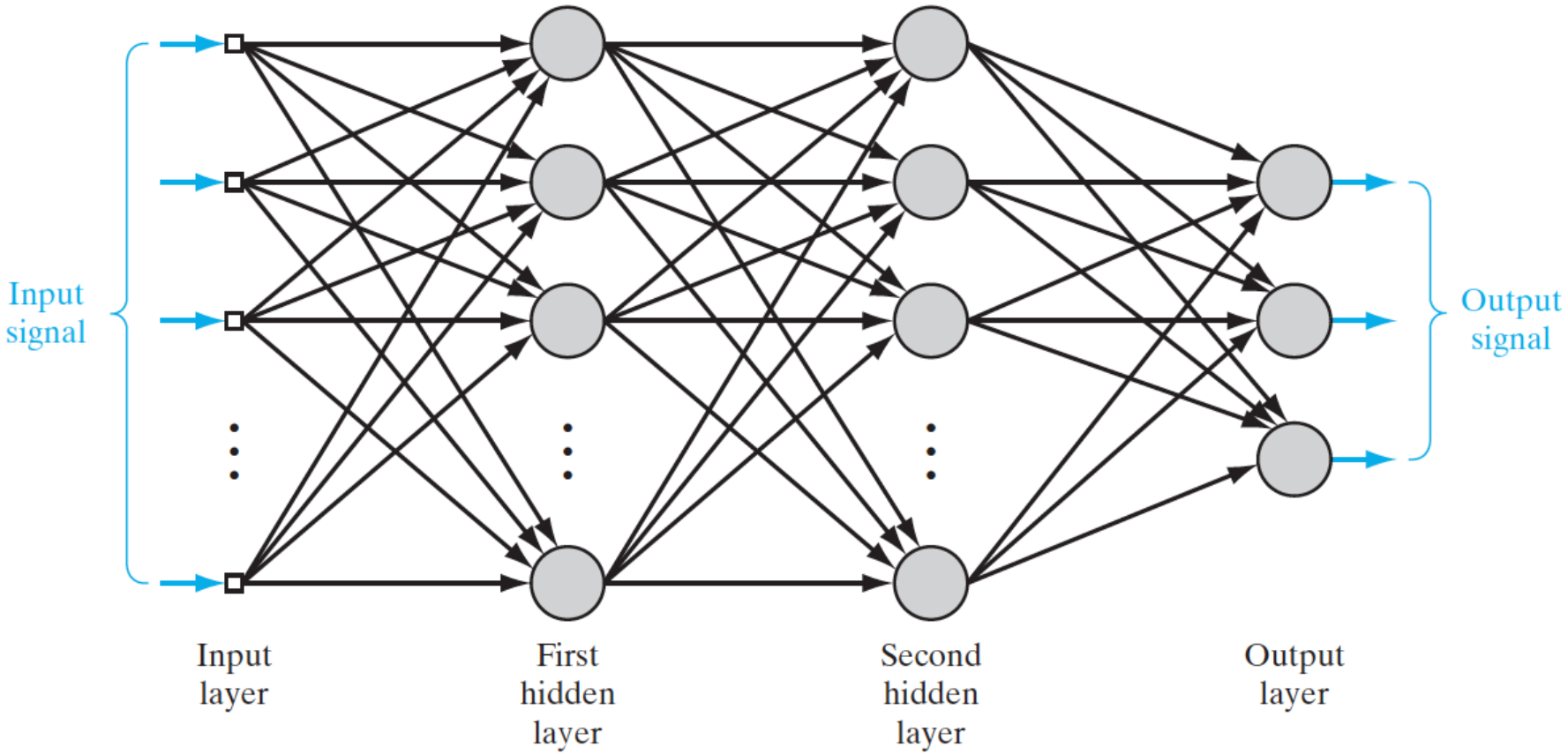
Example trained weights of a linear classifier trained on CIFAR-10:


CIFAR-10 es una base de datos de 60000 imágenes de 10 clases, con 6000 imágenes por clase. Abajo se observan los pesos finales de un perceptrón simple de 10 unidades de salida, tras el entrenamiento.

# Interpretando un clasificador lineal

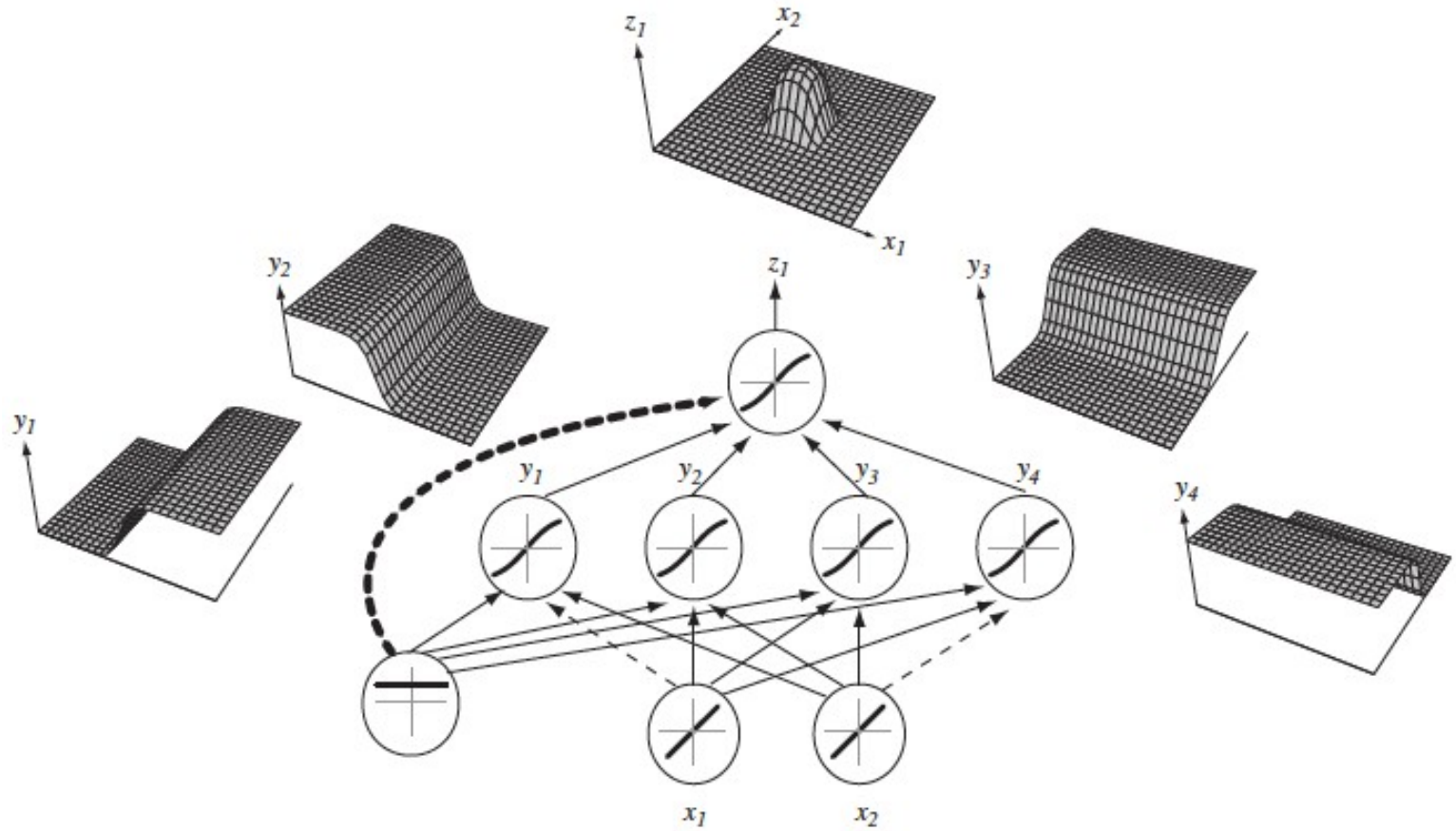
Observar que los pesos finales del perceptrón simple son como templates de las clases. La neurona con mayor activación es la que determina la clasificación y es la más similar a la imagen. Entonces, cuando pongo la imagen de un caballo, la neurona con mayor activación es la octava, que tiene los pesos parecidos a un caballo de dos cabezas. Notar que aunque aunque los caballos tienen una sola cabeza, el peso final de la octava neurona es parecido a un caballo de 2 cabezas, para poder clasificar correctamente a la mayor cantidad los caballos (ya que en un perceptrón simple se tiene un único vector de pesos para identificar a cada clase).

# Red neuronal feedforward multicapa




Red neuronal multicapa de 2 capas ocultas, y una capa de salida con 3 neuronas.

# Capacidad de representación de una red multicapa

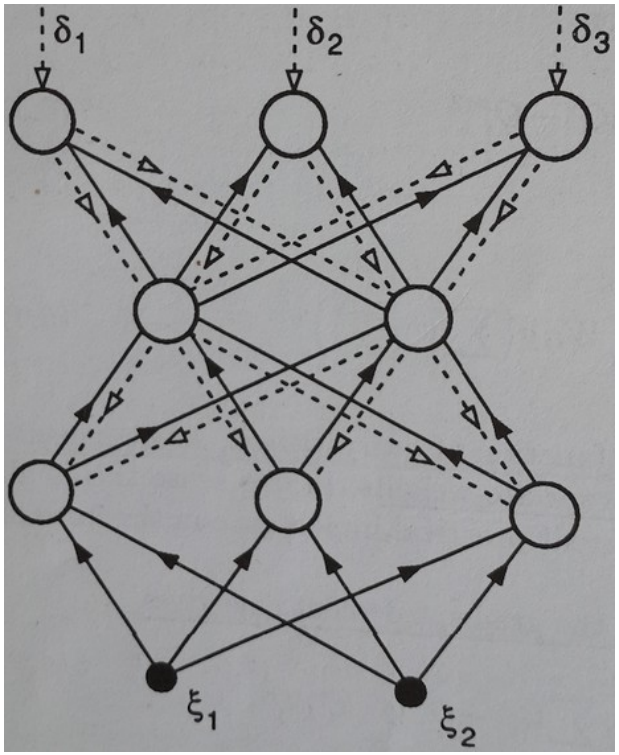



En una red neuronal multicapa, en cada capa podemos modelar un nivel de abstracción. Las capas superiores usan a las abstracciones de las capas inferiores, para poder representar abstracciones más complejas.

# Algoritmo de Backpropagation

El método de aprendizaje de las redes multicapa es el algoritmo de backpropagation, que está basado en minimizar la función de costo $E(w)$, siguiendo la dirección del gradiente descendiente.

1. *Etapa forward:* Calcula el output de la red neuronal.
2. *Etapa backward:* Calcula el gradiente descendiente de $E(w)$ con respecto a los pesos de la red, y actualiza los pesos.



# Pseudocódigo super-simplificado de la red neuronal feedforward multicapa (entrenamiento estocástico)

1) *Inicialización.* Inicializar pesos de la red en forma random.

2) *Entrenamiento.* Mientras MSE > umbral hacer:

2.1) *Epoca:* Mientras haya patrones en conjunto de training:

2.1a) *Activación.* Elegir un patrón del conjunto de training.

2.1b) *Cómputo de la respuesta actual.* Calcular en forma forward el output de la red neuronal.

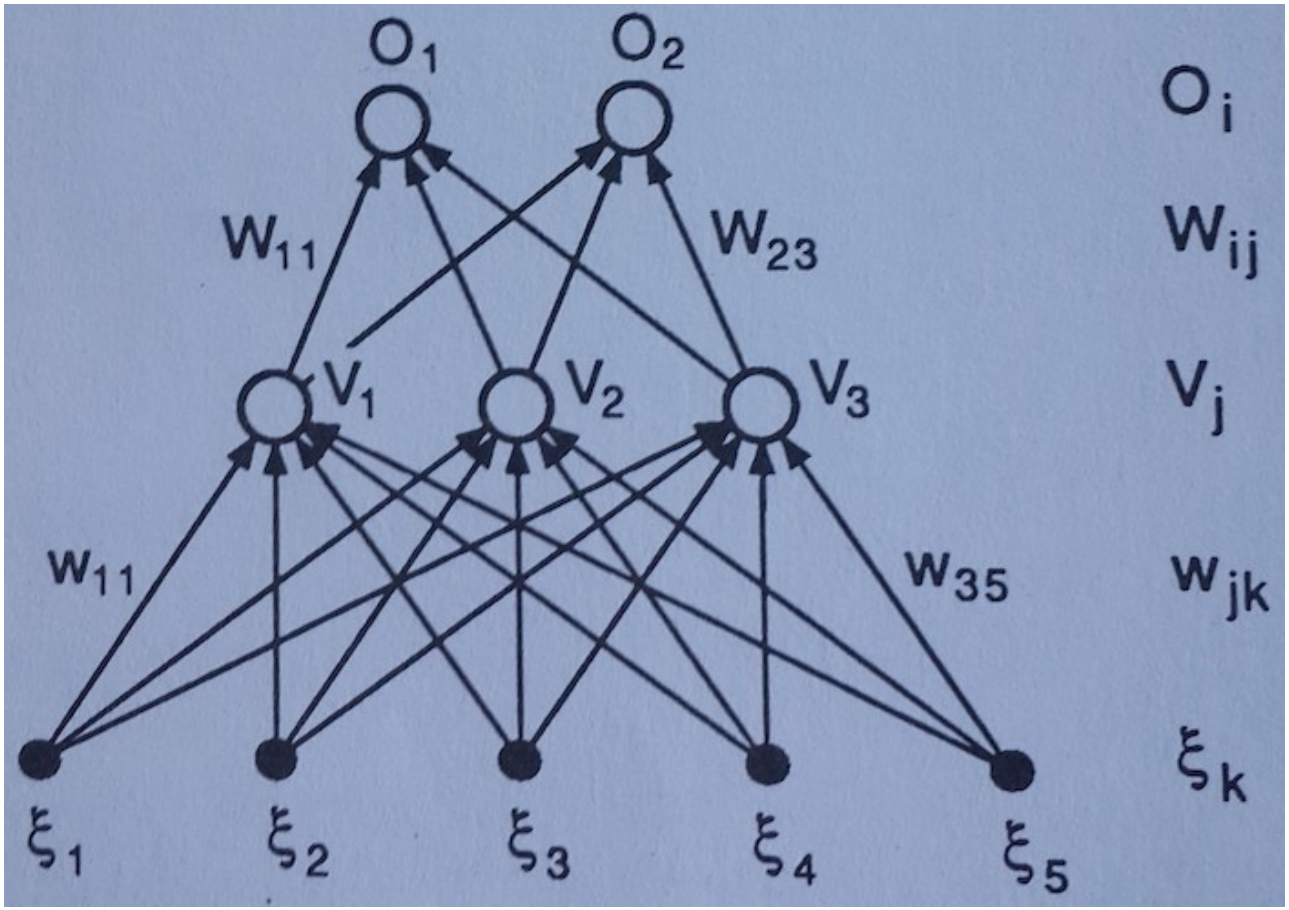
2.1c) *Cómputo del error.* Calcular el error de la red con respecto a la salida esperada.

2.1d) *Cómputo de los deltas.* Calcular los deltas de la red en forma backward.

2.1e) *Adaptación de los pesos.* Actualizar los pesos de la red.

2.2) *Cálculo MSE:* Calcular el MSE del conjunto de training (opcionalmente, también el MSE del conjunto de validación , si hacen early-stopping).

# Algoritmo de Backpropagation



# Algoritmo de Backpropagation: NOTACION

La capas de la red son notadas como $m$, con $m = 1, 2, .., M$ (sin contar la capa de entrada ya que no tiene pesos). La capa 0 es la capa de entrada

Las unidades son notadas como $V_j$. La unidad $V_j^m$ es la unidad $j$ en la capa $m$.

Los inputs son notados como $\xi_k$. El target como $\zeta$.

$w_{ij}^m$ es el peso entre $V_j^{m-1}$ y $V_i^m$

Los patrones del dataset son notados como $\mu$. El número de patrones es $p$ (o sea, $\mu = 1, 2, .., p$).

$\xi_k^\mu$ es el input $k$ del patrón de entrada correspondiente a la muestra $\mu$.

# Algoritmo de Backpropagation (forward)

1. Initialize the weights to small random values.
2. Choose a pattern $\xi_k^\mu$ and apply it to the input layer ($m = 0$) so that

$$V_k^0 = \xi_k^\mu \qquad \text{for all } k\,.$$

3. Propagate the signal forwards through the network using

$$V_i^m = g(h_i^m) = g\Big(\sum_j w_{ij}^m V_j^{m-1}\Big)$$

for each $i$ and $m$ until the final outputs $V_i^M$ have all been calculated.

# Algoritmo de Backpropagation (backward)

4. Compute the deltas for the output layer

$$\delta_i^M = g'(h_i^M)[\zeta_i^\mu - V_i^M] \tag{6.17}$$

by comparing the actual outputs $V_i^M$ with the desired ones $\zeta_i^\mu$ for the pattern $\mu$ being considered.

5. Compute the deltas for the preceding layers by propagating the errors backwards

$$\delta_i^{m-1} = g'(h_i^{m-1}) \sum_j w_{ji}^m \delta_j^m \tag{6.18}$$

for $m = M, M-1, \ldots, 2$ until a delta has been calculated for every unit.

6. Use

$$\Delta w_{ij}^m = \eta \delta_i^m V_j^{m-1} \tag{6.19}$$

to update all connections according to $w_{ij}^{\text{new}} = w_{ij}^{\text{old}} + \Delta w_{ij}$.

7. Go back to step 2 and repeat for the next pattern.

# Laboratorio

Tutorial de Python Numpy:
http://cs231n.github.io/python-numpy-tutorial

Clasificación de imágenes:
http://cs231n.github.io/classification

Clasificación lineal: http://cs231n.github.io/linear-classify

Optimización: http://cs231n.github.io/optimization-1