Redes Neuronales

Aprendizaje Automático - Los Elementos

Ejemplos (etiquetados)

Estudio	Edad	Promedio	N1	N2
2	24	4	7	7.2
5	22	3	4.5	5.2
7	25	4	6.3	6

Modelo

Parámetros (aleatorios)

Algoritmo de Aprendizaje

- Descenso de Gradiente

Error a optimizar

- Error cuadrático
- Entropía
- Otros

Ejemplos nuevos

Estudio	Edad	Promedio
10	19	4
11	20	3

Modelo

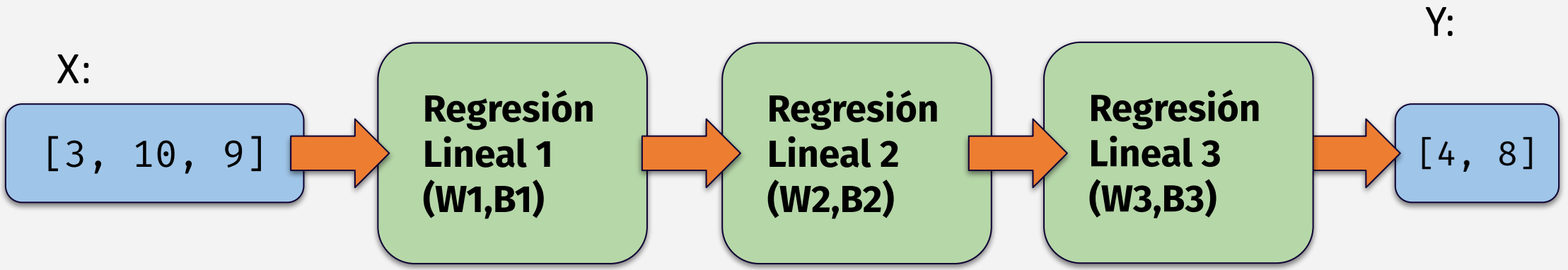
Parámetros (entrenados)

Predicciones

P1	P2
7	7.2
4.5	5.2

Redes Neuronales

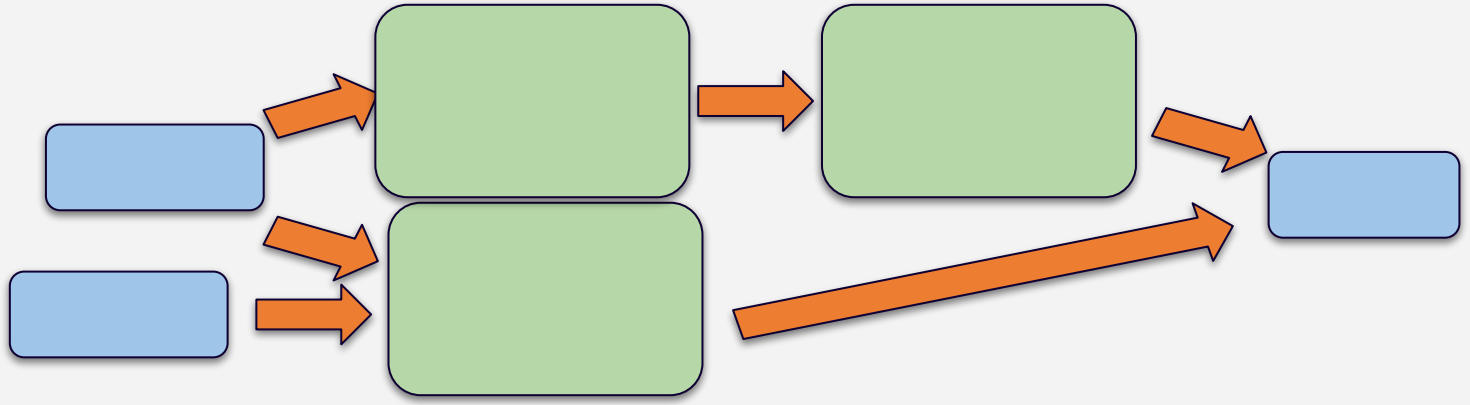
- Red neuronal para predecir notas
- Idea básica
  - Apilar transformaciones
    - Regresiones lineales
    - Otras
- Transformación = Capa
- Red de ejemplo:



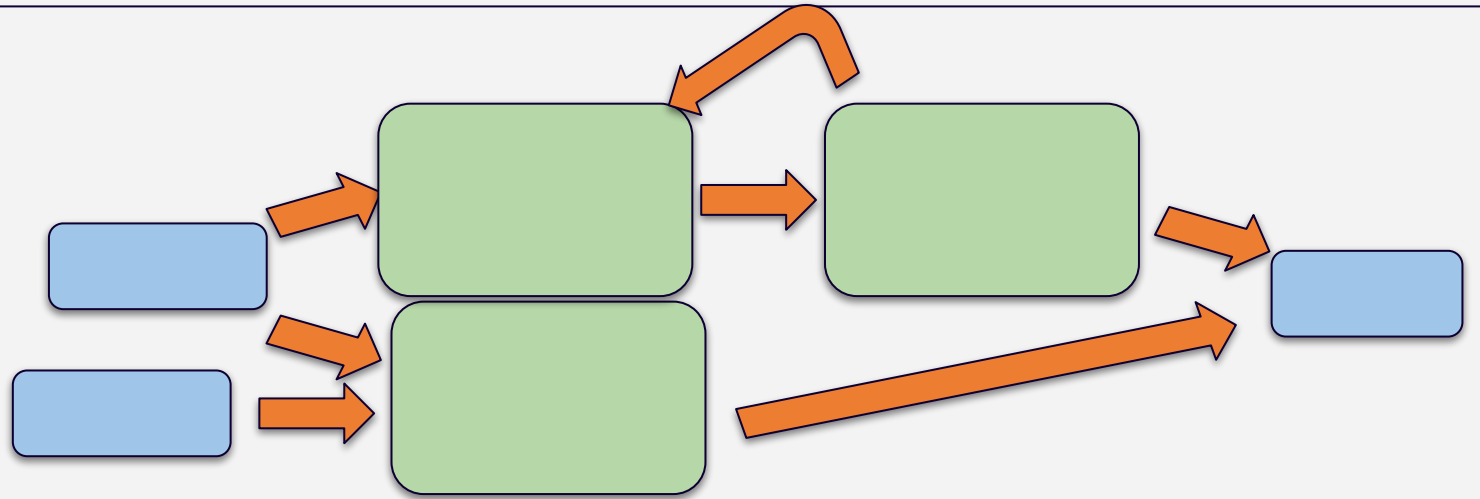
Topologías posibles de red

Lineal (Secuencial)

Acíclico

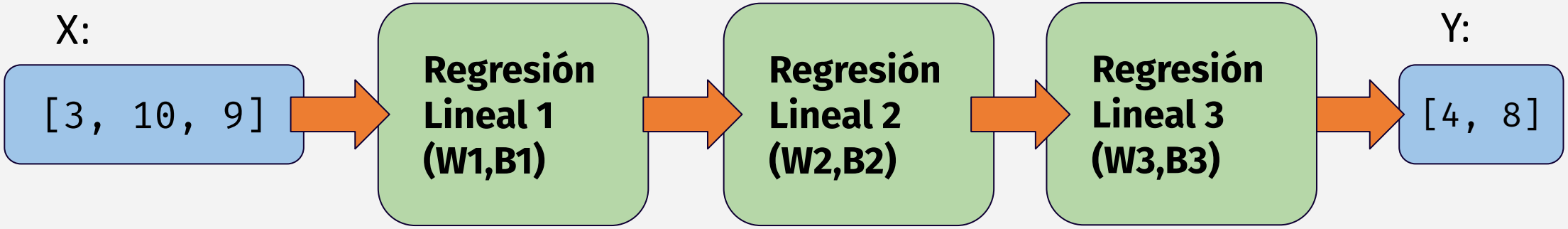


Cíclico



forward : cálculo de la salida de la red

- Salida de la red
  - Cálculo **forward**



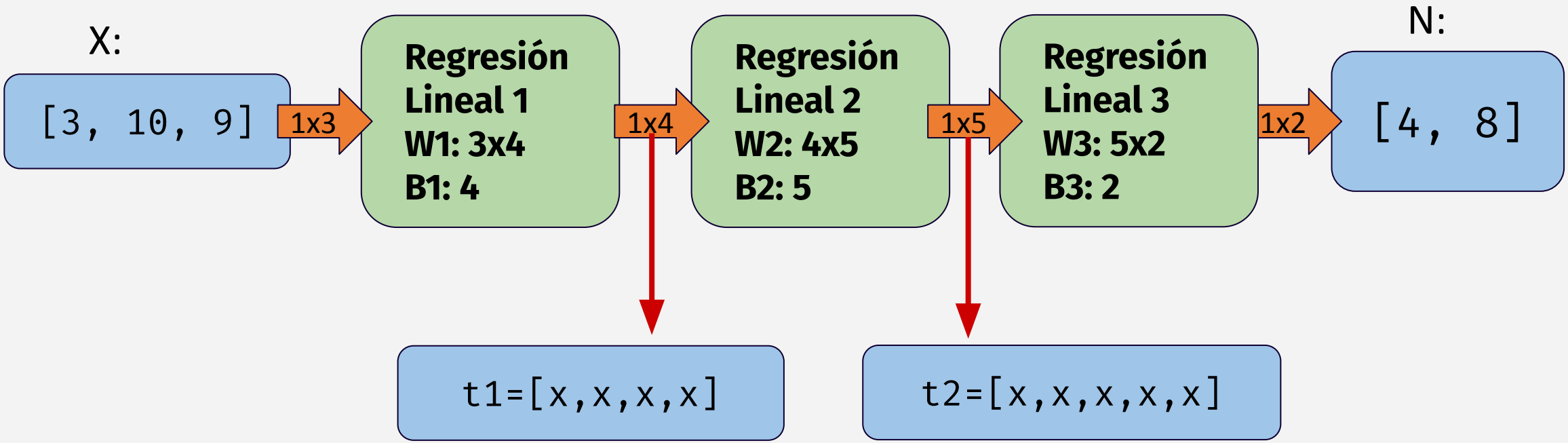
RL 1 recibe tamaño 3
=> W1 tiene tamaño 3x?

RL 2: libre de decidir tamaños de entrada/salida

RL 3 tiene que generar algo de tamaño 2
=> W3 tiene tamaño ?x2
B3 tiene tamaño 2

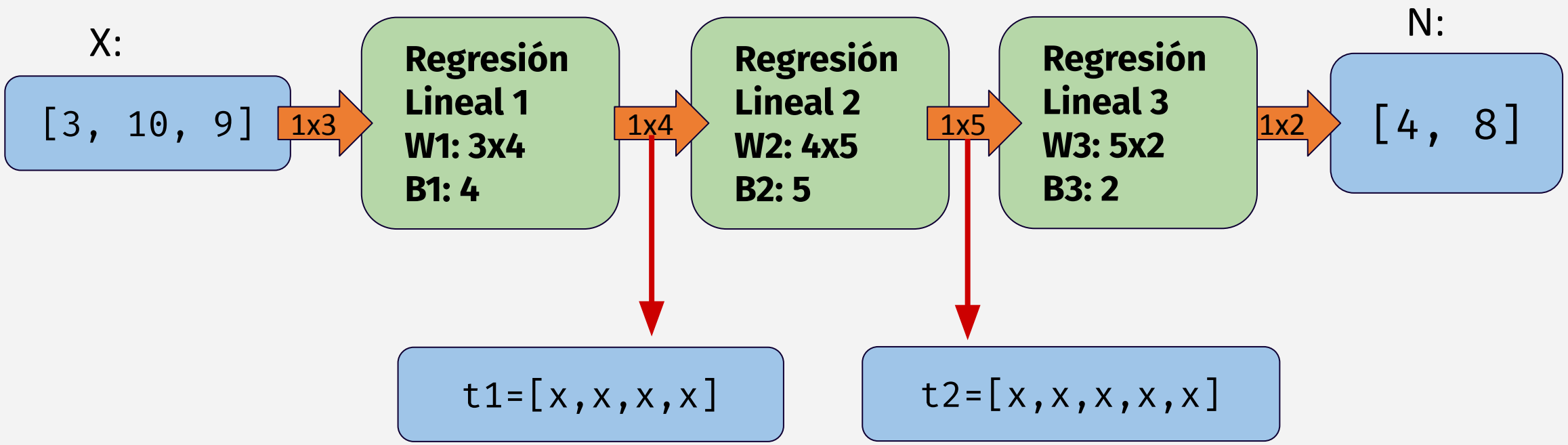
- Usamos 3 regresiones
  - Podemos usar Y
  - Depende del problema

forward y tamaño de parámetros y salidas



- t1 y t2
  - Valores intermedios
  - Se descartan

forward en detalle



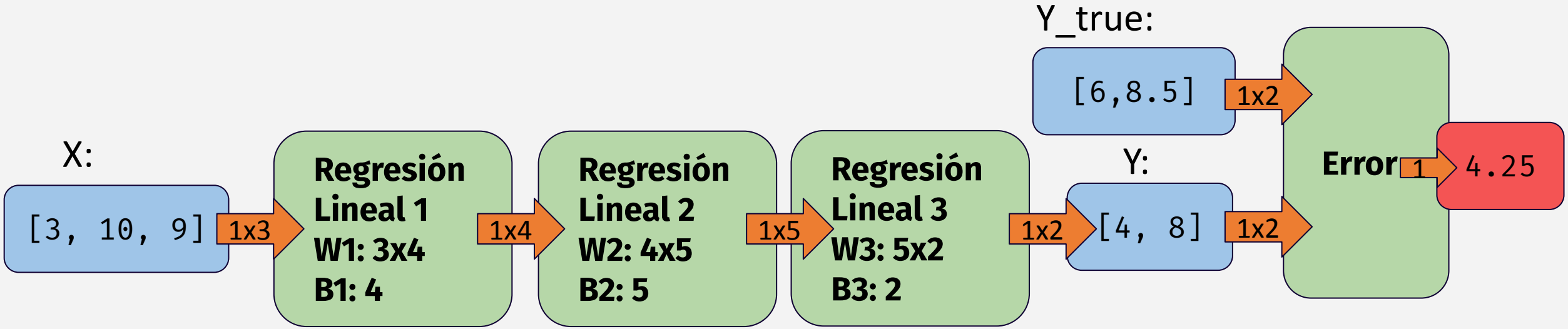
N = RL3(RL2(RL1(X)))
t1 = RL1(X)
t2 = RL2(t1)
Y = RL3(t2)

N =((X W1+B1) W2+B2)W3+B3
t1 =X W1+B1
t2 = t1 W2 + B2
Y = t2 W3 + B3

```
def forward(x,RL1,RL2,RL3)
    t1=RL1.forward(x)
    t2=RL2.forward(t1)
    Y =RL3.forward(t2)
    return Y
```

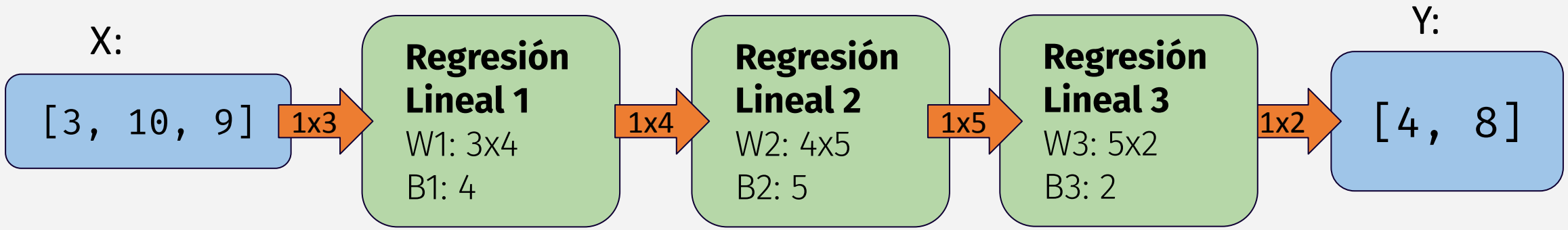
Redes Neuronales - Función de error

- Función de error
  - Considerarla como una **capa** más
    - Capa especial
    - No se usa para predecir
    - Solo para entrenar



Redes Neuronales - Funciones de activación

- Pero esto no va a funcionar..
  - 3 Regresiones = 1 Regresión
  - N Regresiones = 1 Regresión

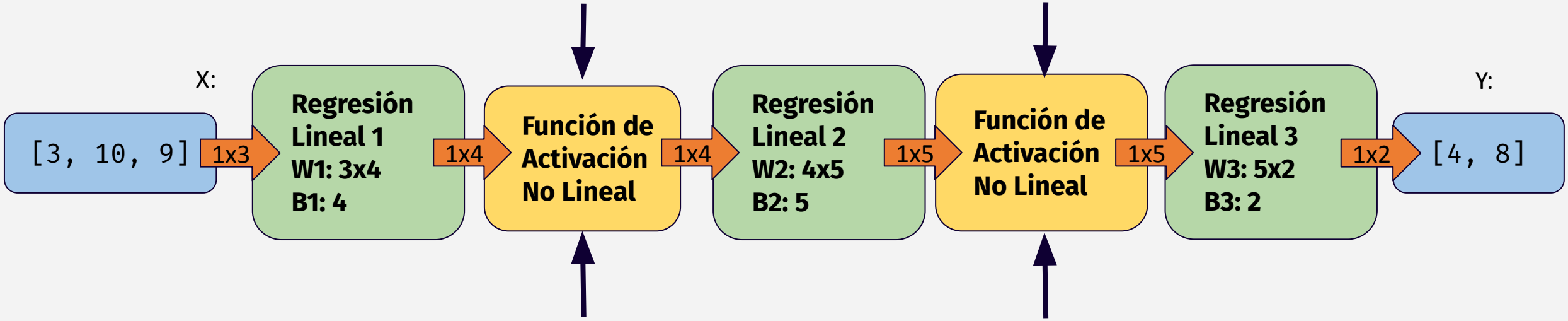


W y V son matrices constantes
=> W.V es otra matriz.
W es una matriz y B es un vector => W.B es otro vector

$$Y = ((X\ W1+B1\ )\ W2\ +\ B2\ )\ W3\ +\ B3$$
$$Y = ((X\ W1\ W2\ +\ B1\ W2)\ +\ B2)\ W3\ +\ B3$$
$$Y = (X\ W4\ +\ B4)\ \ W3\ +\ B3$$
$$Y = (X\ W4\ W3\ +\ B4\ W3)\ \ +\ B3$$
$$Y = \ X\ W5\ +\ B5$$

Redes Neuronales - Funciones de activación

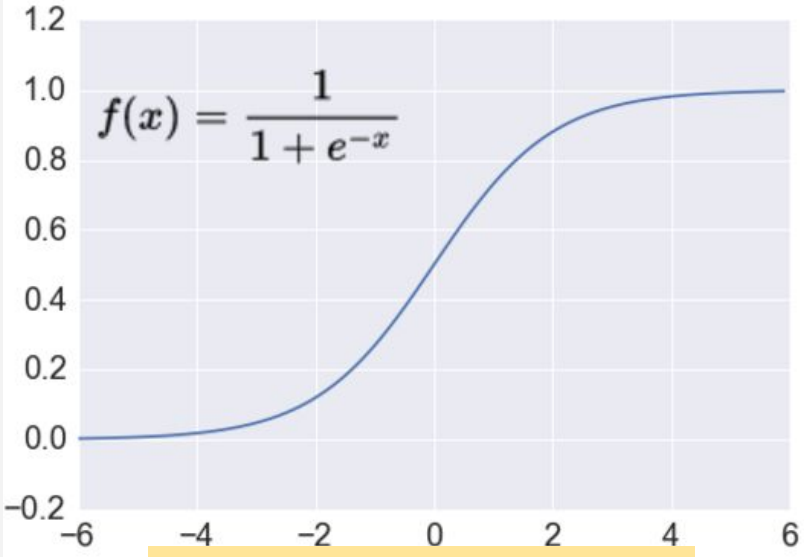
- Permiten estimar con funciones **no lineales**
- Cualquier función, siempre que sea derivable
- sin(x), cos(x), tan(x)
- f(x)=x^2, g(x)=x^25*5-584, etc



- Funciones populares: ReLU(x), Tanh(x) o Sig(x)
- Capas SIN parámetros
  - No se entrenan!

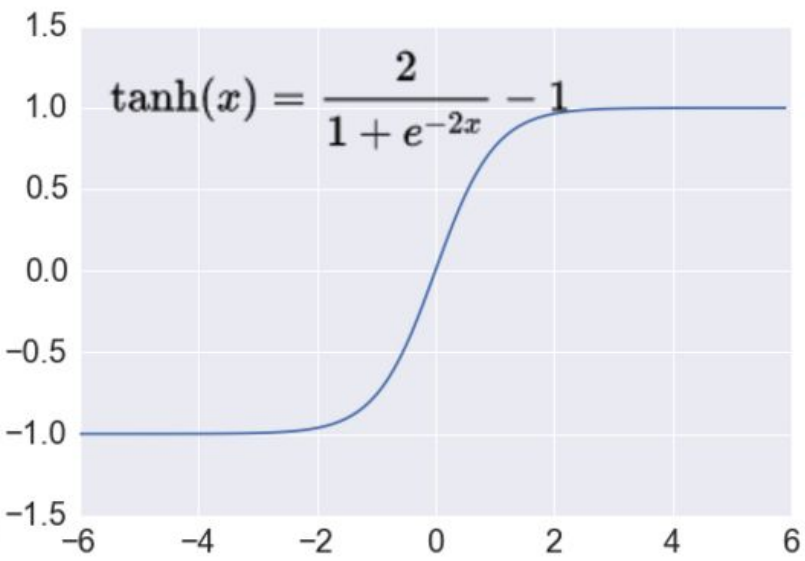
Redes Neuronales - Funciones de Activación

Sigmoid



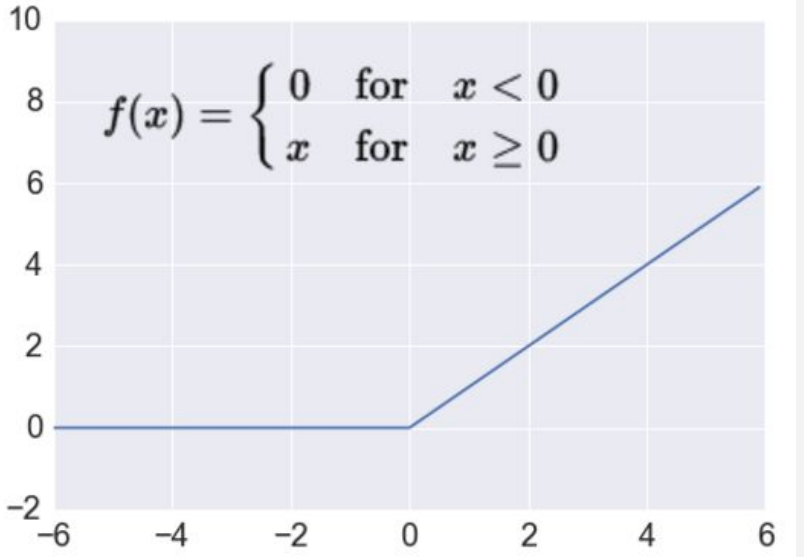

```
def sigmoid(x):
    d=1+exp(-x)
    return 1/d
```

TanH




```
def TanH(x)
    s=sigmoid(x)
    return (x*2)-1
```

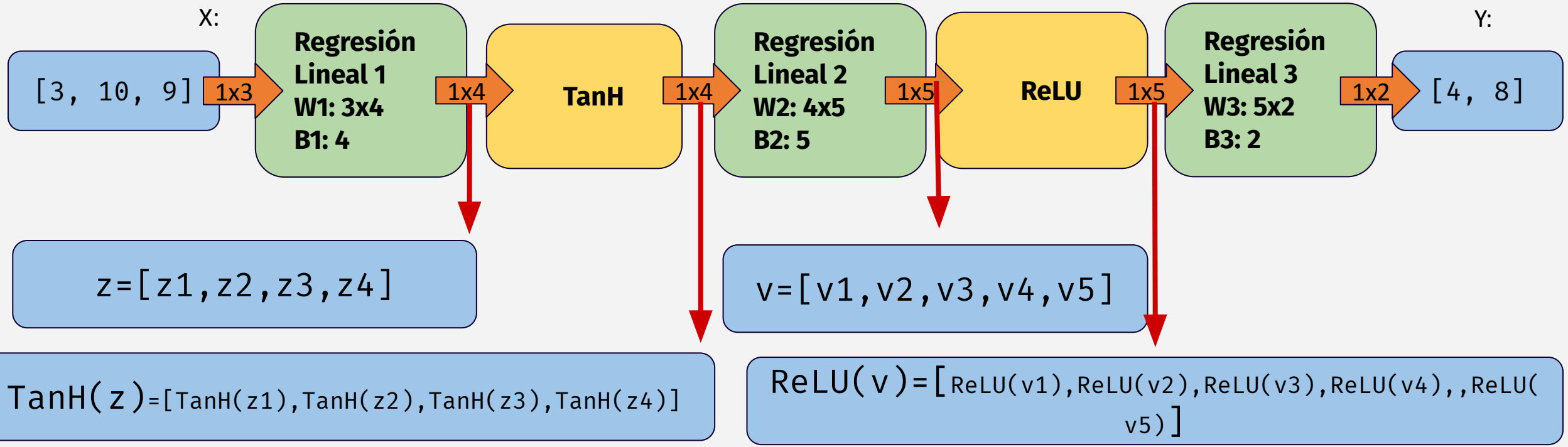
ReLU




```
def ReLU(x):
    if x<=0:
        return 0
    else:
        return x
```

- Sigmoid = la misma que regresión logística
- TanH = (Sigmoide*2)-1
- ReLU = 0 en los negativos, lineal en los positivos
- Generalmente se usa ReLU o variantes, más fácil de entrenar

Redes Neuronales - Funciones de activación




```
def TanH_vector(z):
    res=np.zeros_like(z)
    for i in range(len(z)):
        res[i]=TanH(z[i])
    return res
```

```
def ReLU_vector(v):
    res=np.zeros_like(v)
    for i in range(len(v)):
        res[i]=ReLU(v[i])
    return res
```

Regresión Con Redes + Keras

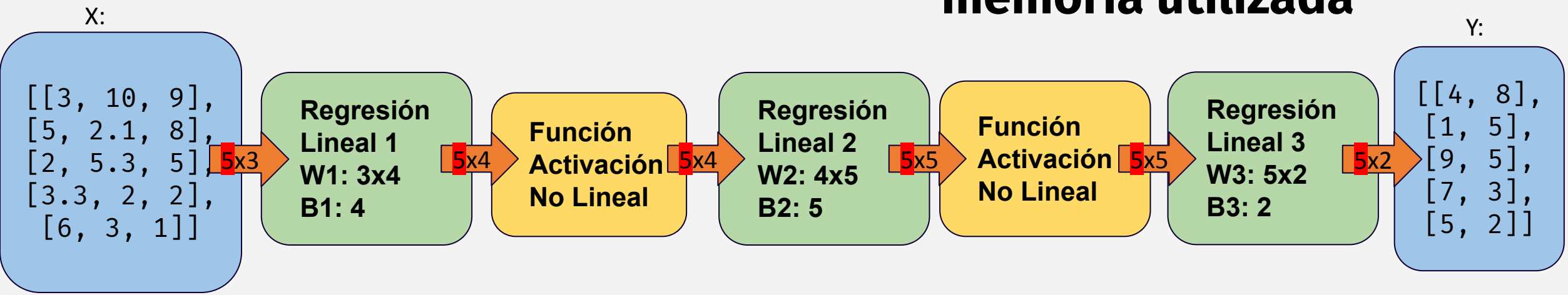
```
x,y=cargar_dataset()
nx,d_in  = x.shape # x tiene tamaño n x d_in
ny,d_out = y.shape # y tiene tamaño n x d_out
import keras

model=keras.Sequential()
model.add(keras.Dense(4,input_shape=[d_in],activation='tanh'))
model.add(keras.Dense(5,activation='relu'))
model.add(keras.Dense(d_out,activation='none'))
model.compile(loss='mse', # error cuadratico medio
              optimizer='sgd') # descenso de gradiente
history = model.fit(x,y,epochs=100,batch_size=32)
y_predicted=model.predict(x)
```

- Los valores 4 y 5 son arbitrarios
- La cantidad de capas también
- Son hiperparámetros

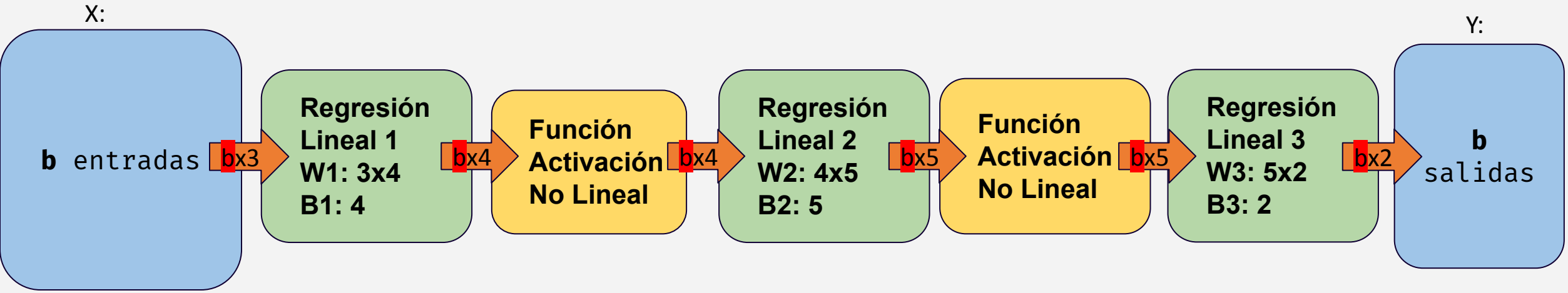
Redes Neuronales - Ejecución por lotes

Ejemplo con batch_size=5

Importante para conocer memoria utilizada

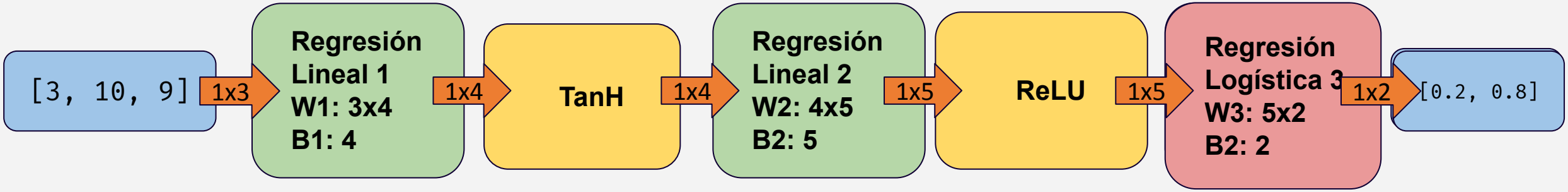


En general, b x n (b = batch_size)



Redes Neuronales - Clasificación

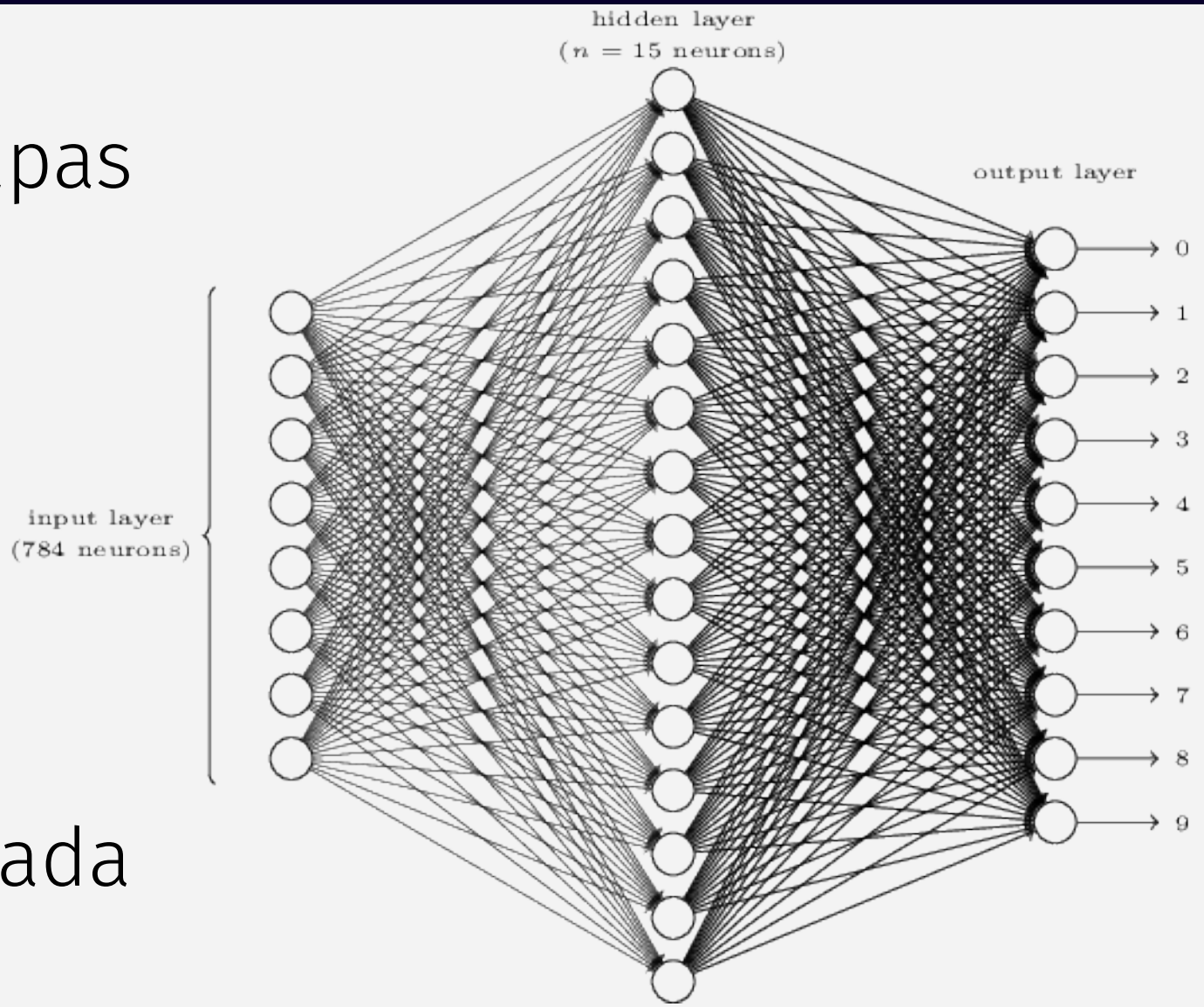
Modificar la red anterior para clasificación con 2 clases:



- Capas intercambiables
  - Pero verificar tamaños
- Cambio última capa:
  - Cambia el tipo de dato de salida
- Última capa se llama cabeza o head.

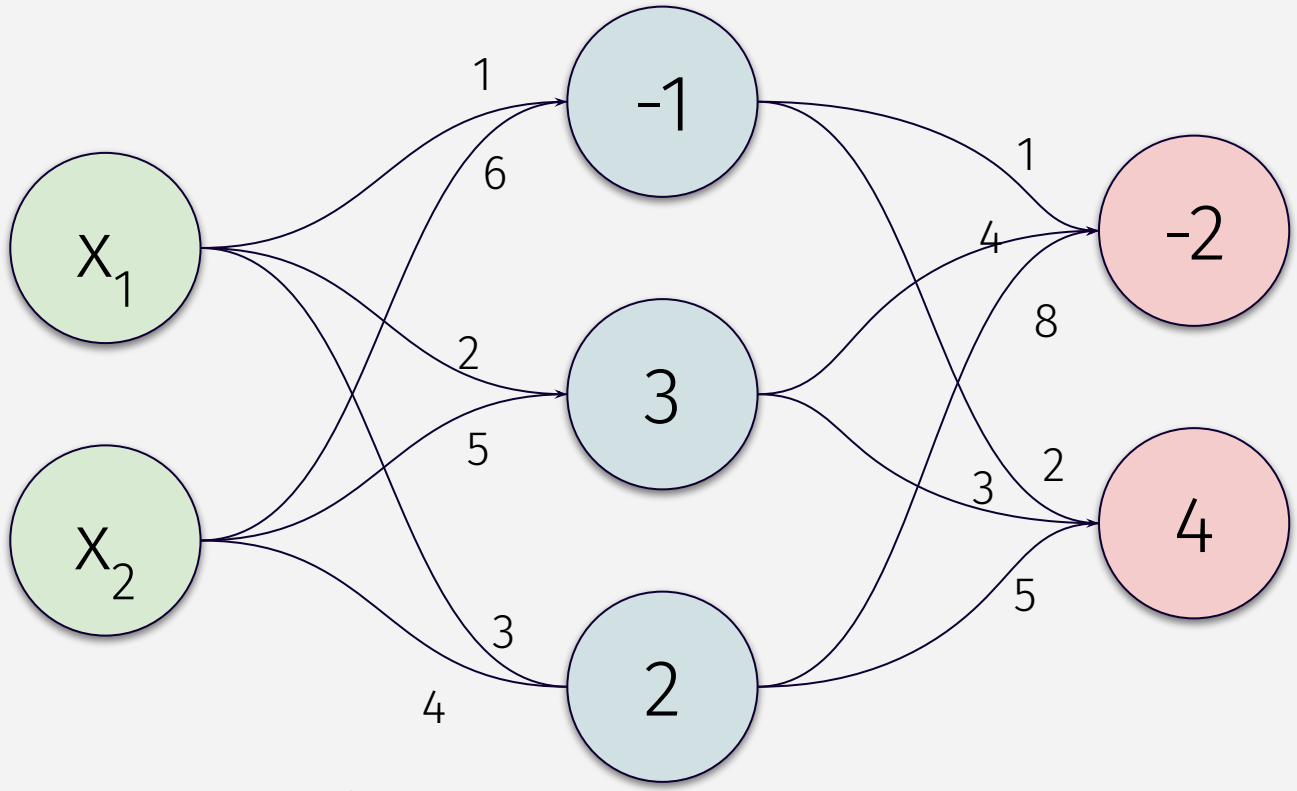
Capa Densa o Fully Connected o Reg. Lineal

- Nomenclatura
  - Viene de red de 2 capas
- Valores = Neuronas
  - De entrada
  - Ocultas
  - Salida
- y = w x + b
  - x: N entradas
  - y: M salidas
  - w conecta cada entrada con cada salida
    - tamaño NxM



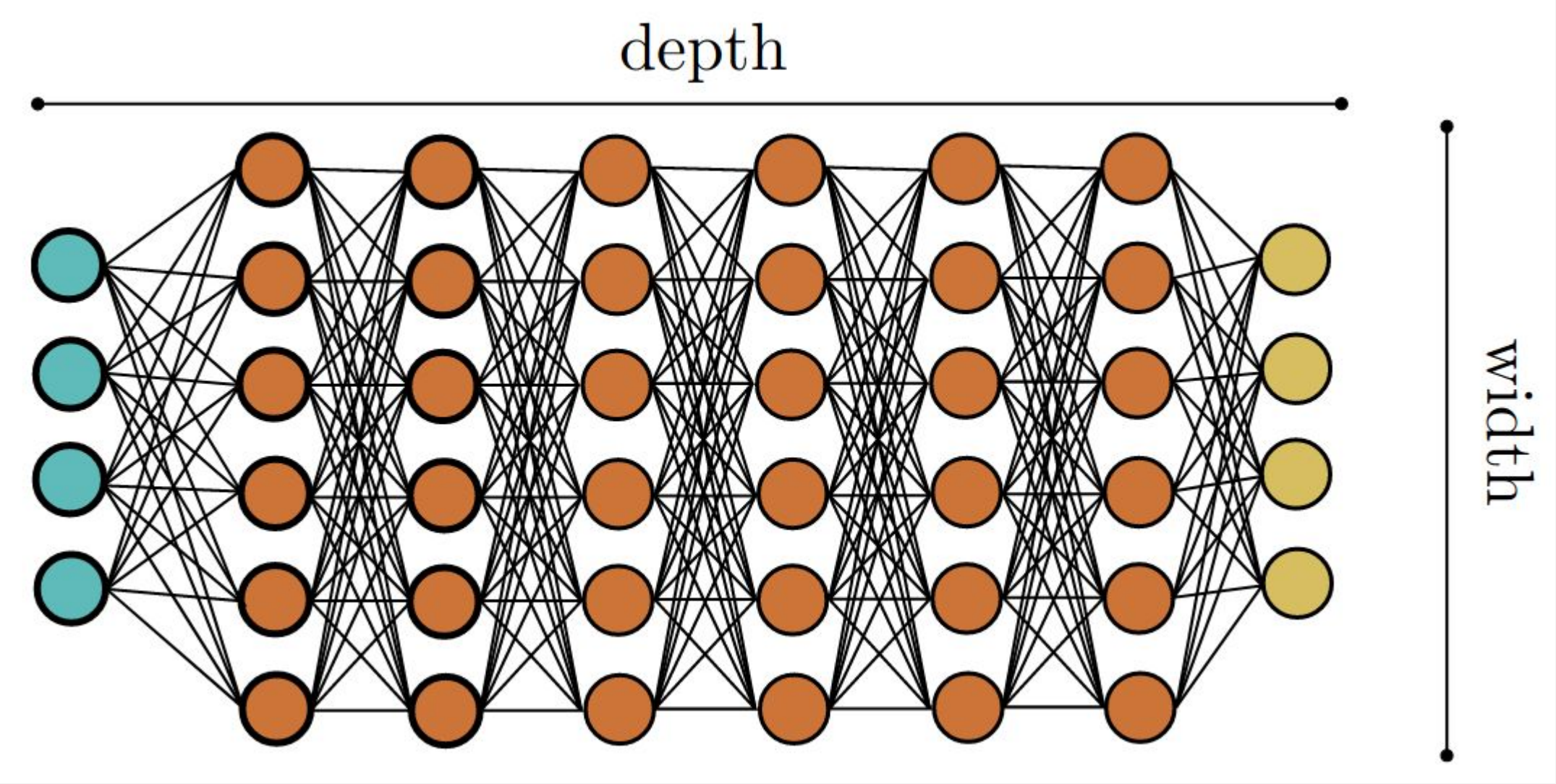
Capa Densa o Fully Connected o Reg. Lineal

- Ejemplo red 2 capas
  - 2 entradas
  - 3 ocultas
  - 2 salidas
- $f(x) = (xw_1+b_1)w_2+b_2$
  - w_1 = [[1,2,3]
    [6,5,4]]
  - b_1= [-1,3,2]
  - w_2 = [[1,2]
    [4,3]
    [8,5]]
  - b_2= [-2,4]



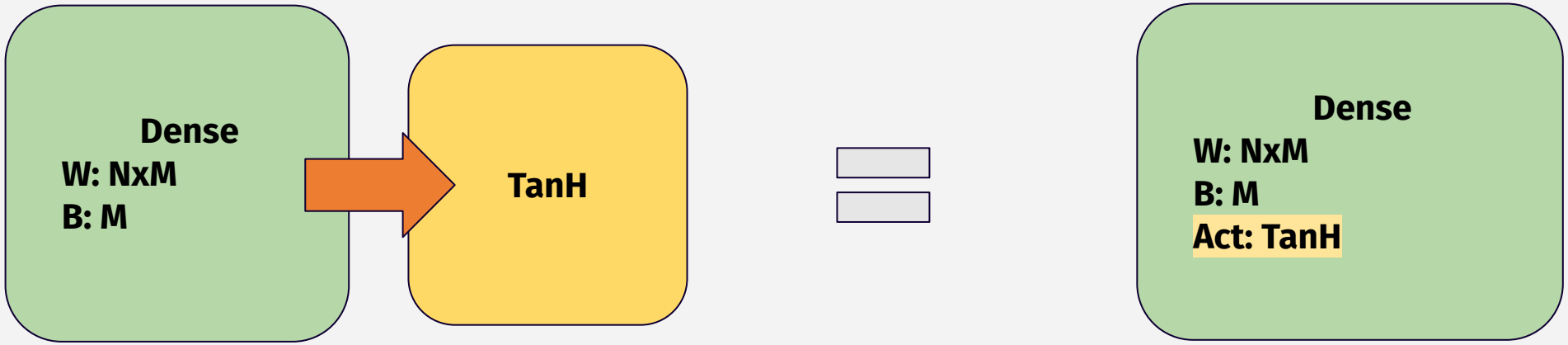
- Aristas
  - Valores de w_1 y w_2
- Nodos
  - Valores de b_1 y b_2

Profundidad vs Ancho



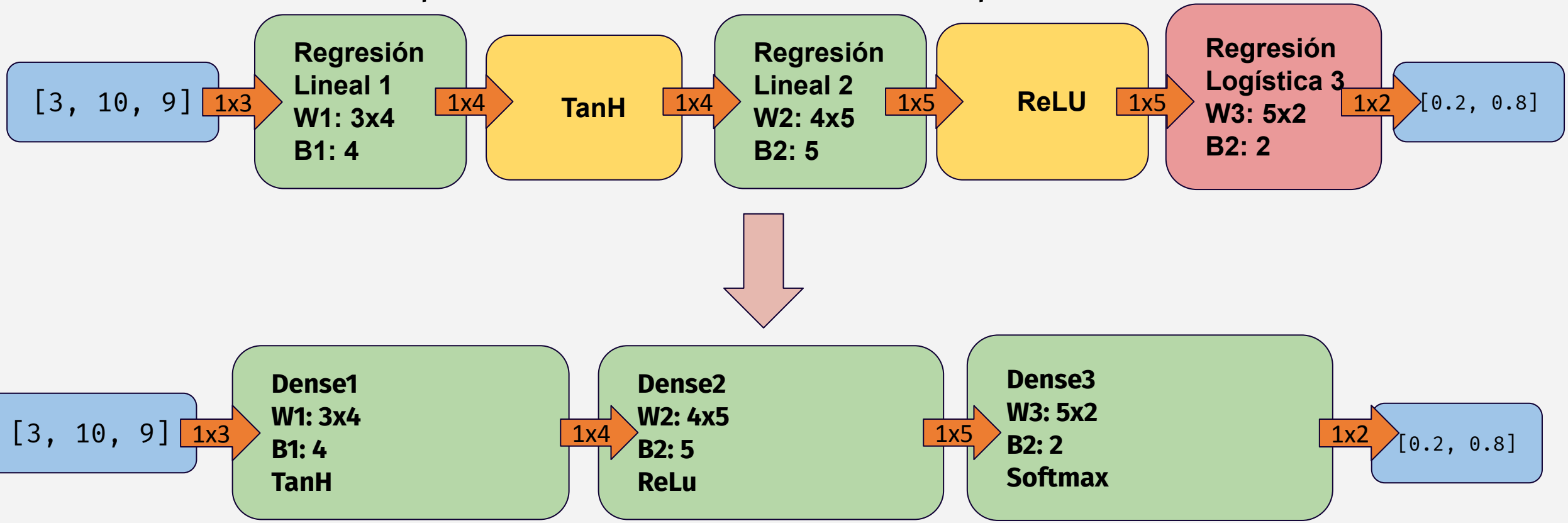
Densas con activación vs Densas y activación

- Activación “dentro” de la capa
  - Equivalente
  - Cómodo



Redes Neuronales - Términos

- Regresión Lineal
- Regresión Logística
  - Reservados para modelos de 1 capa

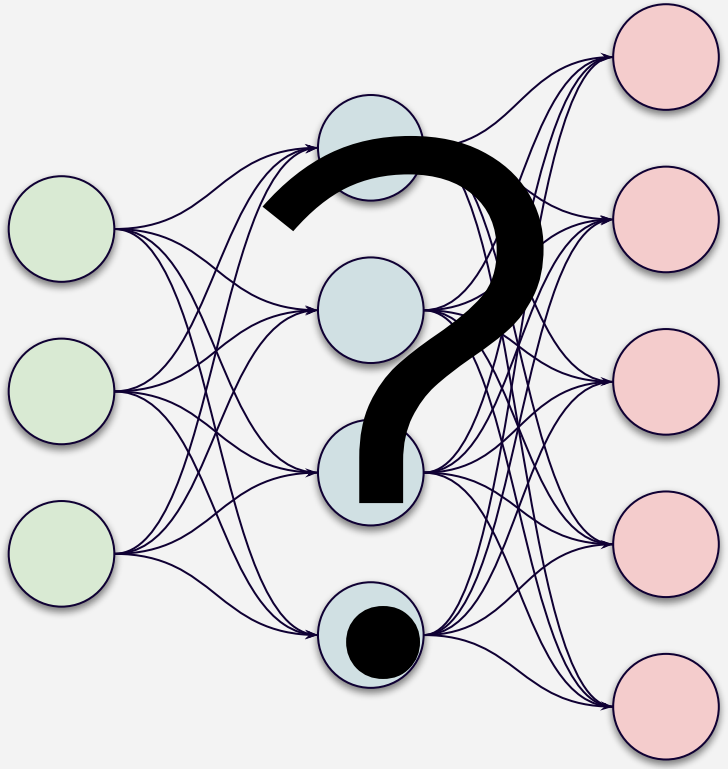


Clasificación Con Redes + Keras

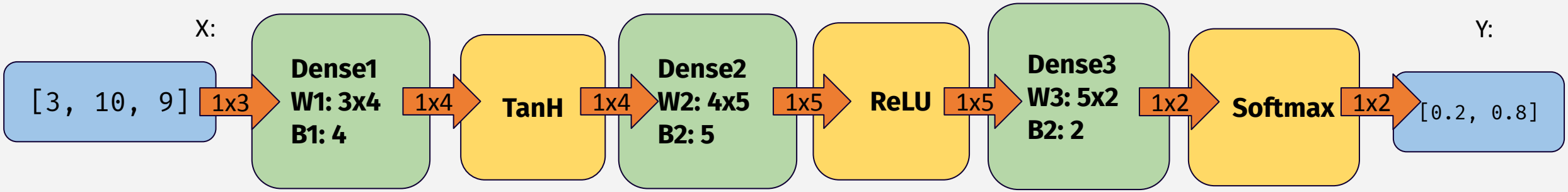
```
x,y=cargar_dataset(one_hot=True)
nx,d_in  = x.shape # x tiene tamaño n x d_in
ny,d_out = y.shape # y tiene tamaño n x d_out
import keras

model=keras.Sequential()
model.add(keras.Dense(4,input_shape=[d_in],activation='tanh'))
model.add(keras.Dense(5,activation='relu'))
model.add(keras.Dense(d_out,activation='softmax'))
model.compile(loss='categorical_crossentropy', # ent cruz
              optimizer='sgd', # descenso de gradiente
              metrics='accuracy')
history = model.fit(x,y,epochs=100,batch_size=32)
y_predicted=model.predict(x)
```

Diseño de redes

- Dado un problema X
  - ¿Cuántas capas?
  - ¿Cuantas neuronas por capa?
  - ¿Qué funciones de activación?
- No hay **un** óptimo
  - Varias combinaciones funcionan
- ¿Cómo diseñar?
  - Experiencia previa y literatura
  - Búsqueda de hiperparámetros
    - #capas, #neuronas, tipos de capa



Redes Neuronales - Resumen



- Transforman vectores de entrada en vectores de salida
- Regresión Lineal/Logística + Funciones de activación
  - Transformaciones no lineales
  - Mayor poder de clasificación/regresión
- Capas modulares
  - Combinar de cualquier forma => *arquitecturas* o *topologías*
- Descenso de gradiente para optimizar