Aprendizaje Automático Profundo (Deep Learning)

Dr. Facundo Quiroga - Dr. Franco Ronchetti

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
- otros

Error a optimizar

- Error cuadrático
- Entropía
- Otros

Ejemplos nuevos

Estudio	Edad	Promedio
10	19	4
11	20	3

Modelo

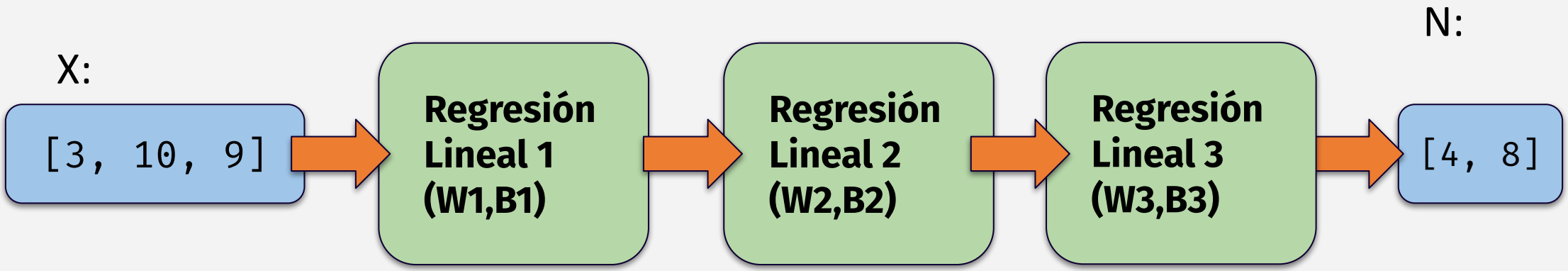
Parámetros (entrenados)

Predicciones

P1	P2
7	7.2
4.5	5.2

Redes Neuronales

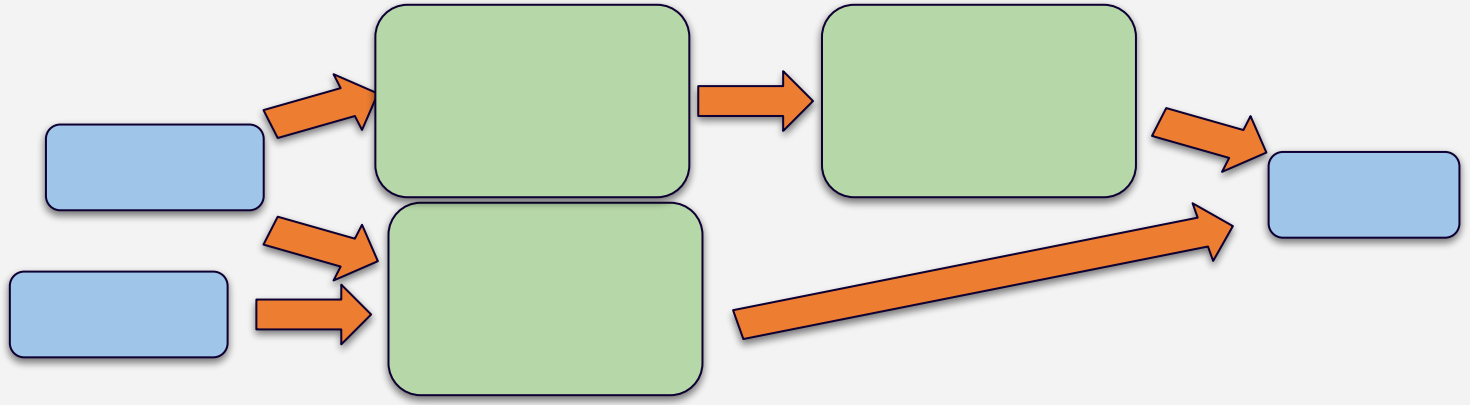
- Red neuronal para predecir notas
- Idea básica
  - Apilar transformaciones
    - Regresiones lineales
    - Otras
- Transformación = Capa



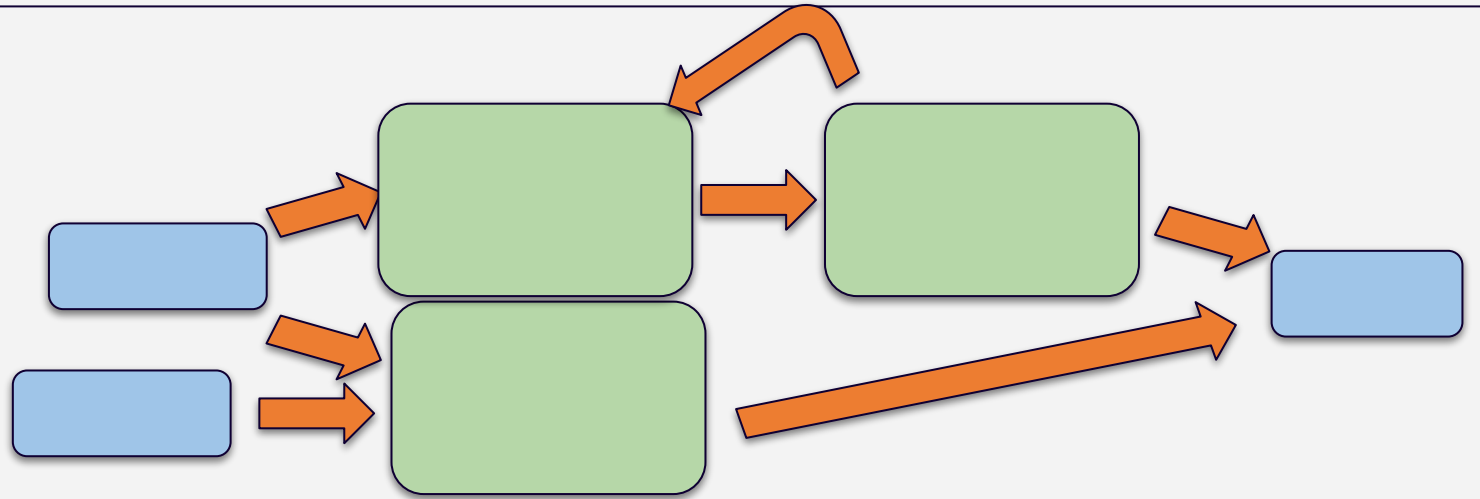
Topologías posibles de red

Lineal (Secuencial)

Acíclico

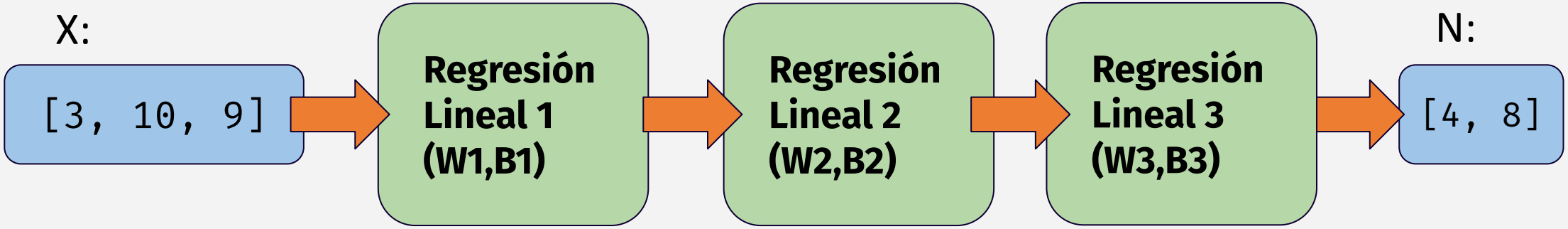


Cíclico



forward : cálculo de la salida de la red

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



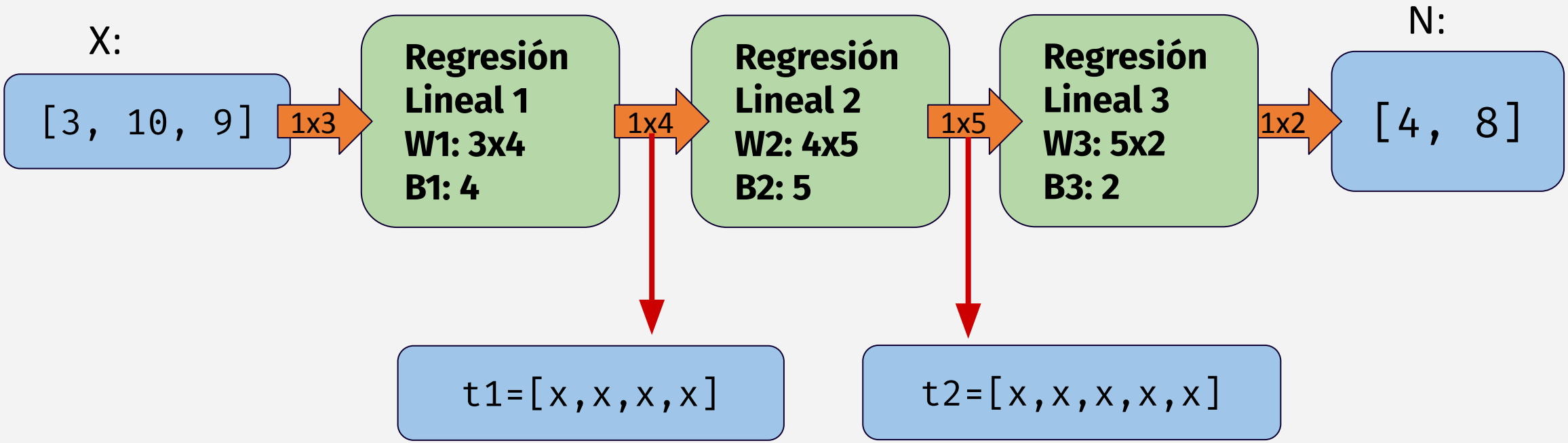
RL 1 recibe tamaño 3
=> W1 tiene tamaño 3x?

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

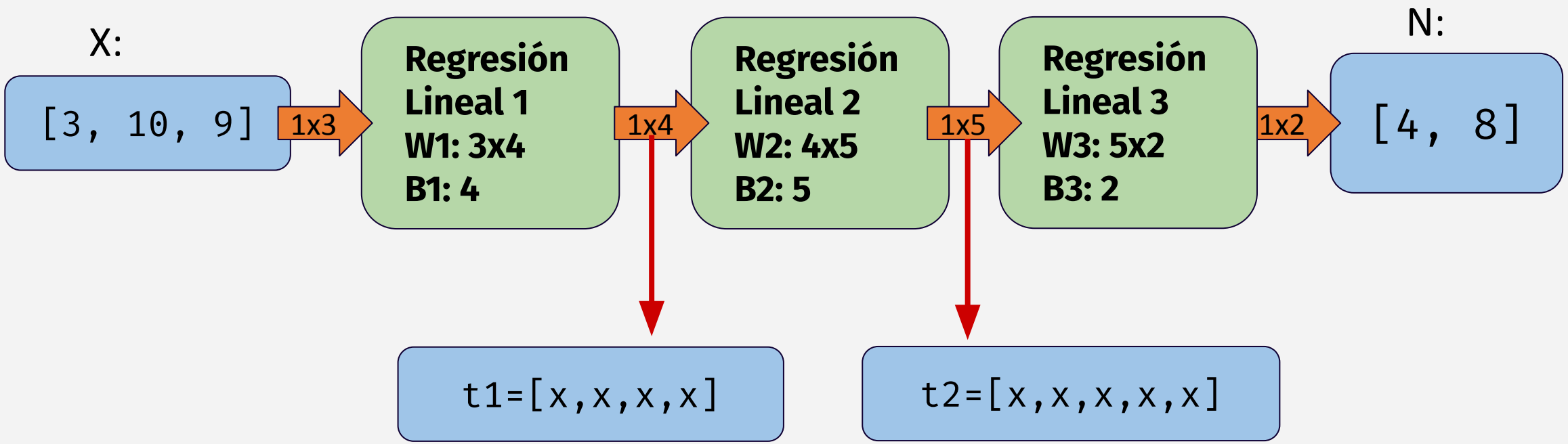
- Usamos 3 regresiones
  - Podemos usar N
  - Depende del problema

forward y tamaño de parámetros y salidas



- t1 y t2
  - Valores intermedios
  - Se descartan

forward en detalle



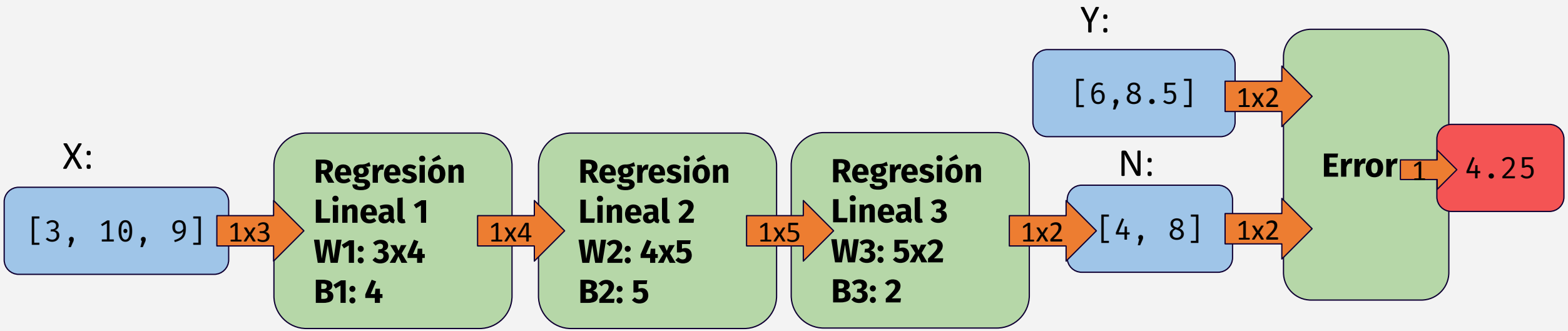

```
N  = RL3( RL2( RL1(X) ) )
t1 = RL1(X)
t2 = RL2(t1)
N  = RL3(t2
```

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

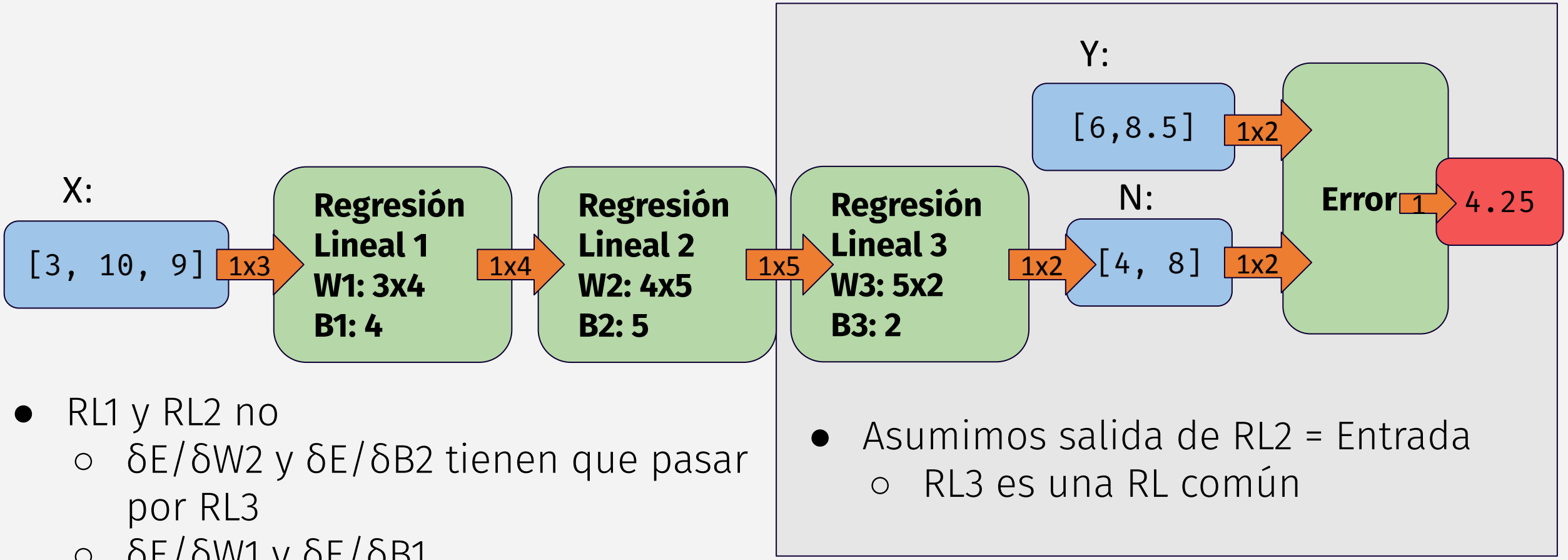
```
def forward(x,RL1,RL2,RL3)
    t1=RL1.forward(x)
    t2=RL2.forward(t1)
    N =RL3.forward(t2)
    return N
```

Redes Neuronales - Función de error

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



Visión por capas

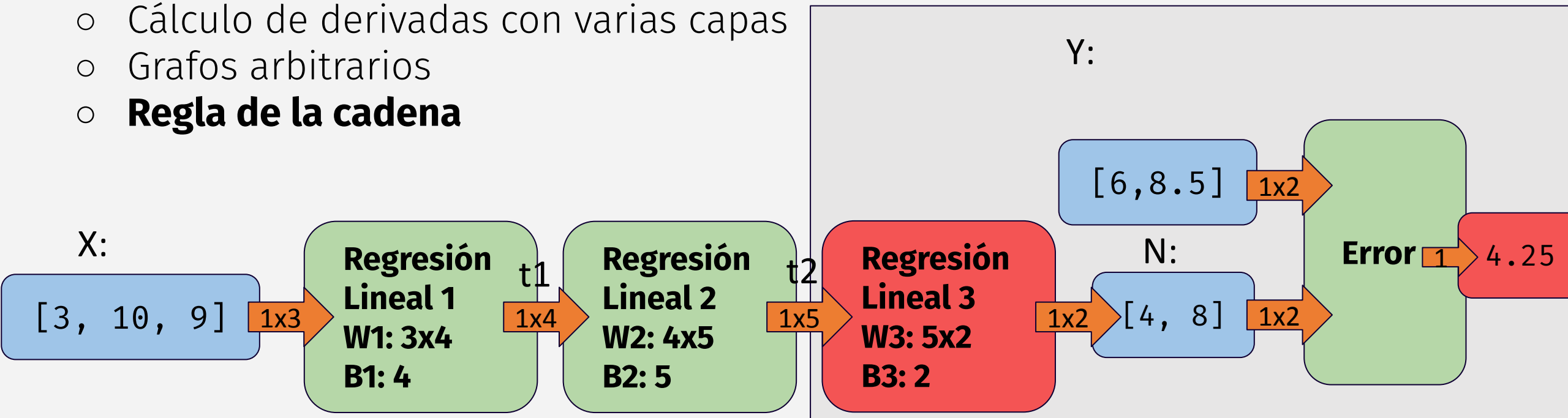


- RL1 y RL2 no
  - δE/δW2 y δE/δB2 tienen que pasar por RL3
  - δE/δW1 y δE/δB1
    - tienen que pasar por RL2 y RL3

- Asumimos salida de RL2 = Entrada
  - RL3 es una RL común

backward: Cálculo de derivadas con Backpropagation

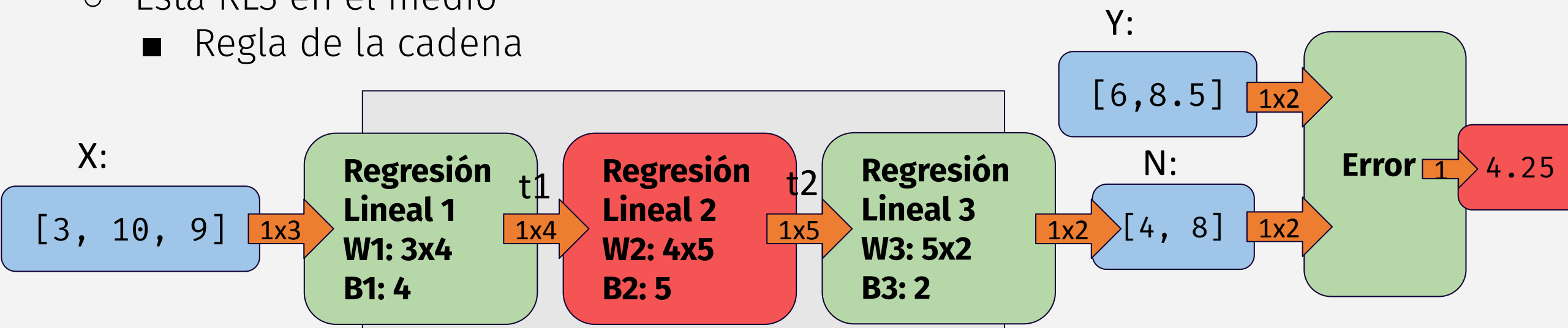
- Función o pasada backward()
- Algoritmo back-propagation
  - Cálculo de derivadas con varias capas
  - Grafos arbitrarios
  - **Regla de la cadena**



- Calculo derivadas de W3 y B3
  - δE/δW3 y δE/δW3
  - También para capa anterior
    - δE/δt2

backward: Cálculo de derivadas con Backpropagation

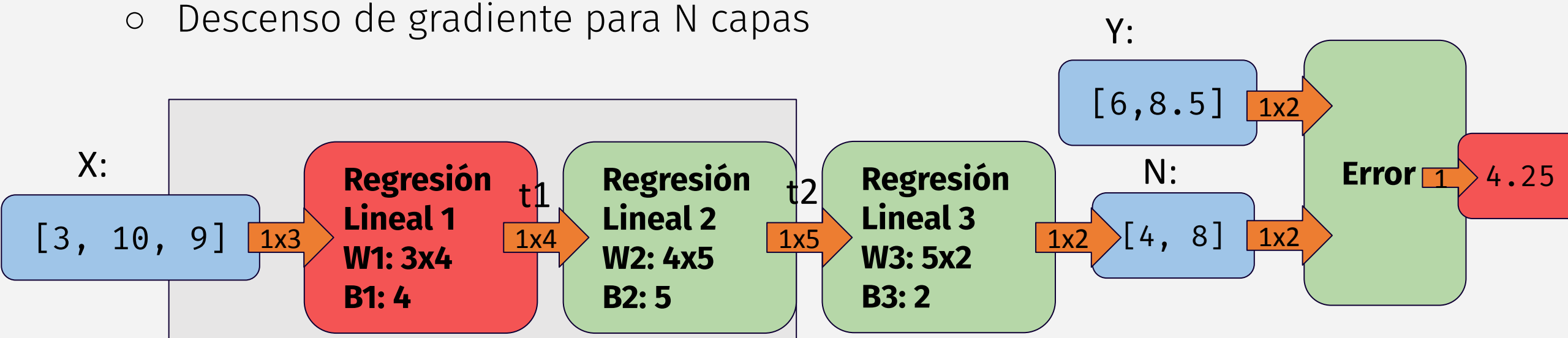
- δE/δW2 y δE/δB2
  - No se pueden calcular directamente
  - Esta RL3 en el medio
    - Regla de la cadena



- δE/δW2 = δE/δt2 × δt2/δW2
- δE/δB2 = δE/δt2 × δt2/δB2
  - δt2/δW2 y δt2/δB2 fáciles
- También calculo δE/δt1
  - para la capa anterior

backward: Cálculo de derivadas con Backpropagation

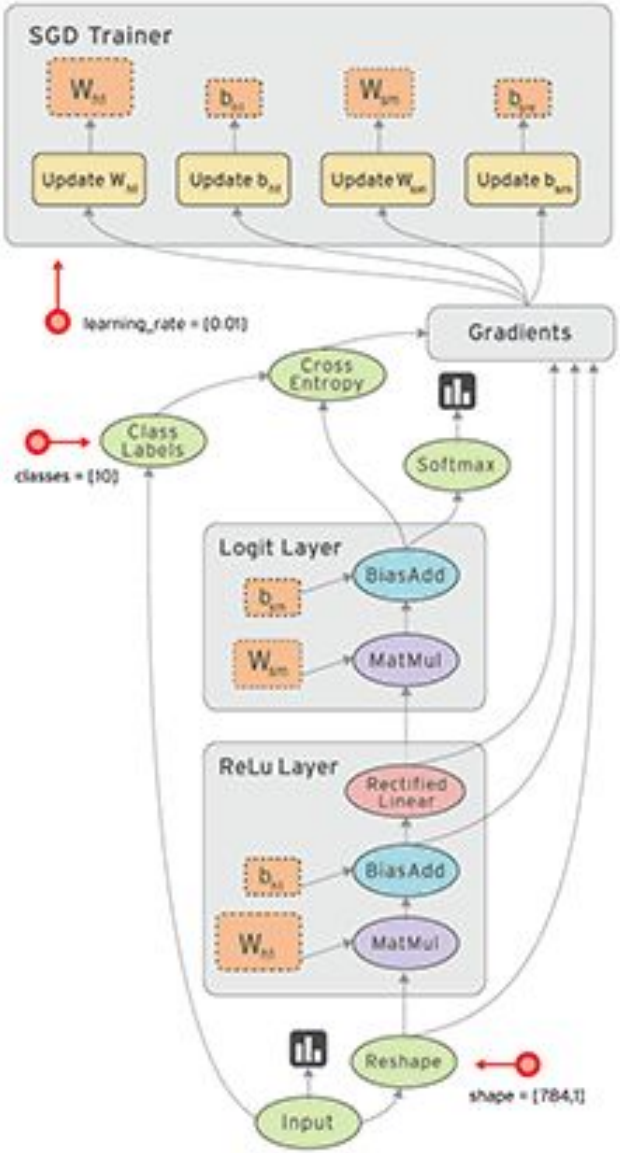
- Función o pasada backward()
- Algoritmo back-propagation
  - Descenso de gradiente para N capas



- δE/δW1 = δE/δt1 × δt2/δW1
- δE/δB1 = δE/δt1 × δt2/δB1
  - δt1/δW2 y δt1/δB2 fáciles
- No necesito δE/δx
  - No hay capa anterior

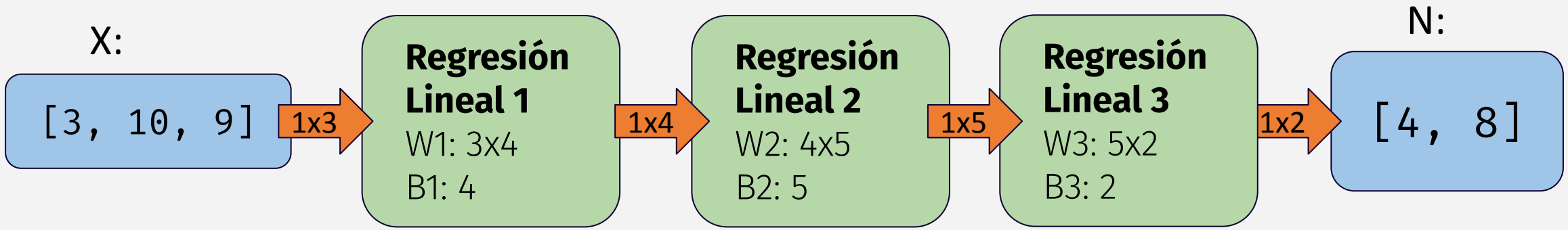
Descenso de gradiente y derivadas

- Descenso de gradiente
  - Igual que antes
  - Más parámetros
  - Misma idea
    - $w = w - \alpha \, \delta E / \delta w$
- Cálculo de derivadas
  - No repetiremos
  - Frameworks actuales calculan automáticamente
    - Especificás el **forward**
      - Capas/arquitectura de la red
    - El framework determina solo el backward



Redes Neuronales - Funciones de activación

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

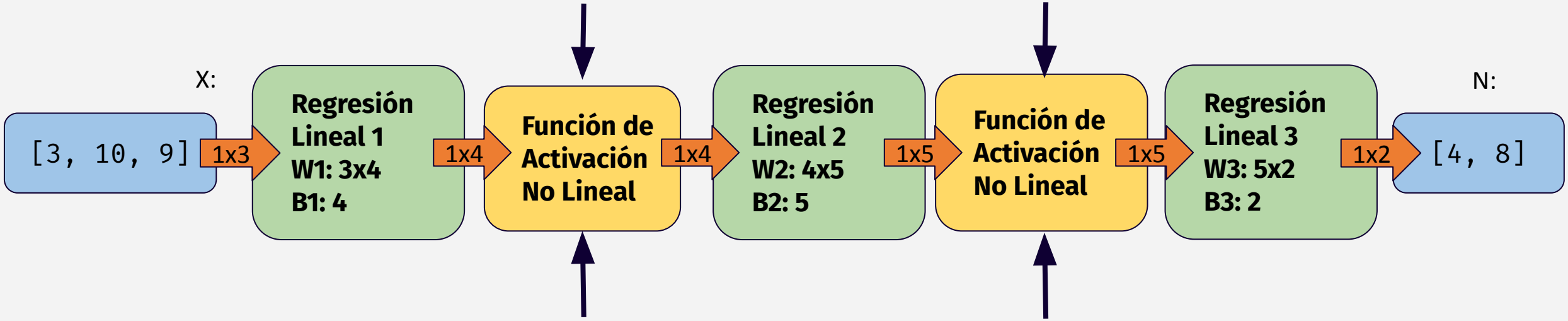


W y V son matrices constantes
=> W.V es otra matriz.
W es una matriz y B es un vector => W.B es otro vector

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

Redes Neuronales - Funciones de activación

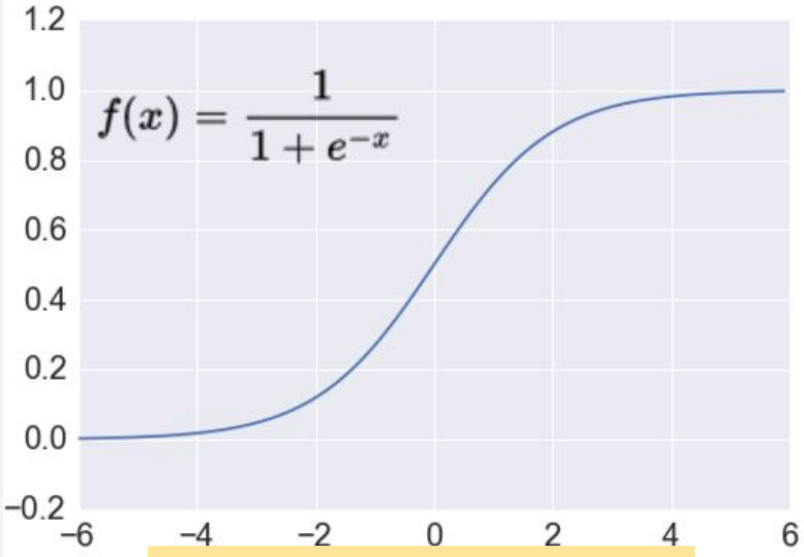
- Permiten estimar con funciones no lineales
- Cualquier función, siempre que sea derivable
- sin(x), cos(x), tan(x)
- f(x)=x^2, g(x)=x^25*5-584, etc



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

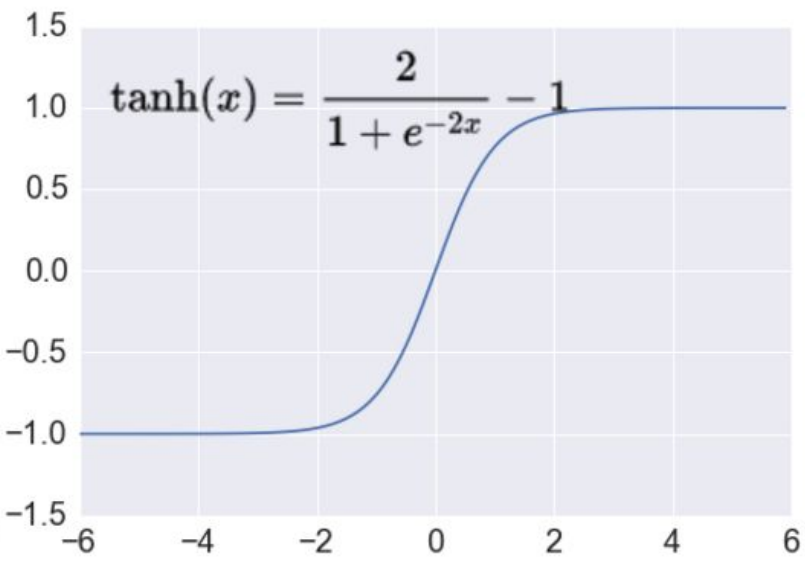
Redes Neuronales - Funciones de Activación

Sigmoid



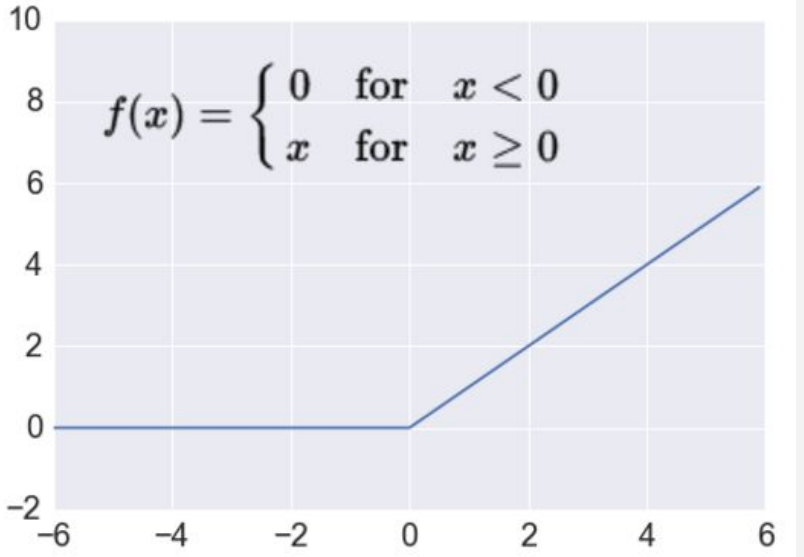

```
def sigmoid(x):
    d=1+exp(-x)
    return 1/d
```

TanH




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

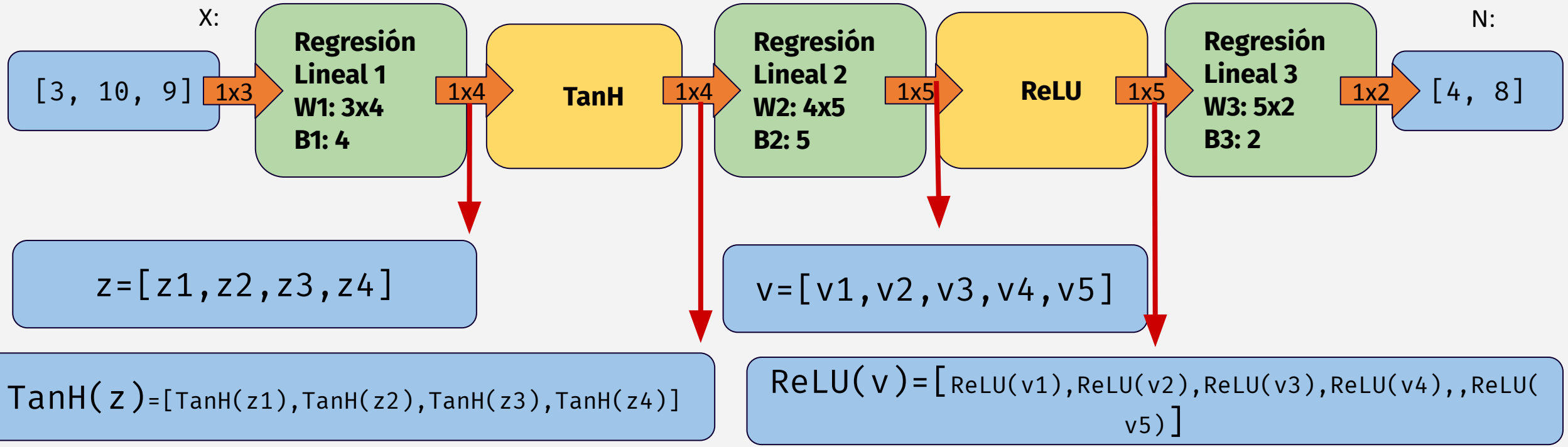
ReLU




```
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

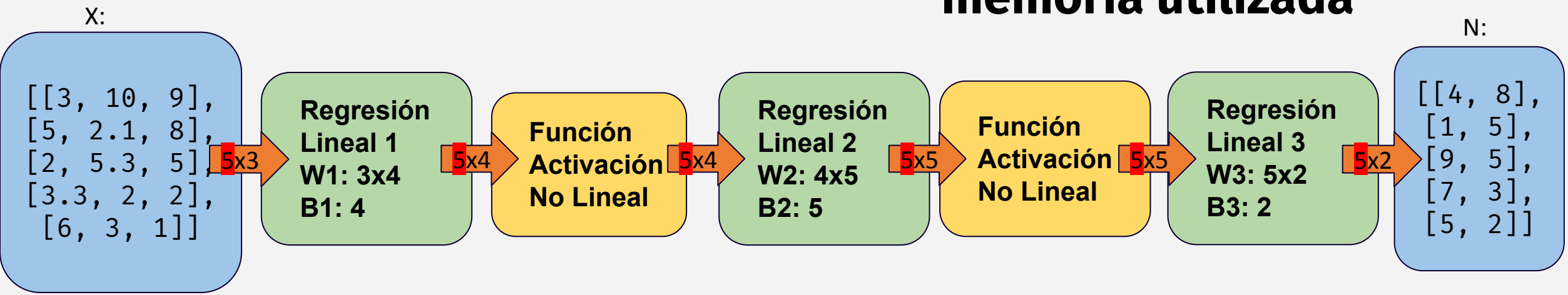
```
x,y=cargar_dataset()
nx,d_in  = x.shape # x tiene tamaño n x d_in
ny,d_out = y.shape # y tiene tamaño n x d_out
import keras

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

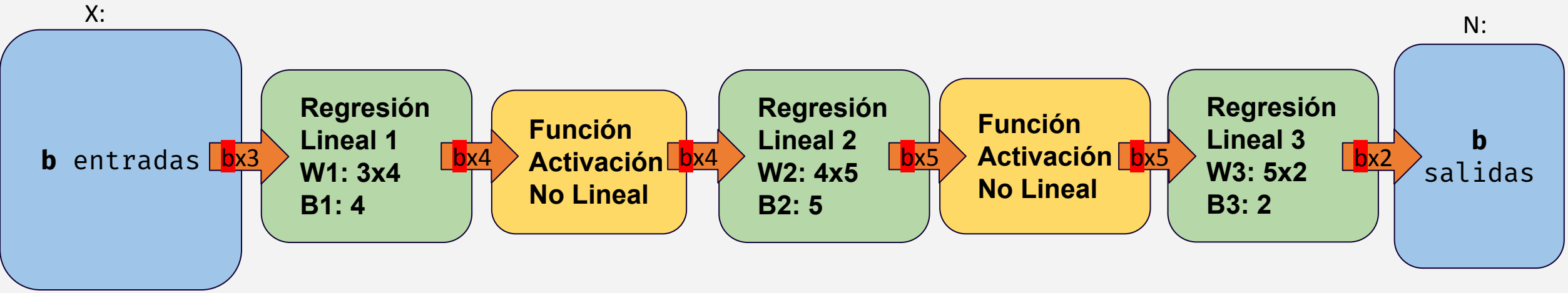
Redes Neuronales - Ejecución por lotes

Ejemplo con batch_size=5

Importante para conocer memoria utilizada

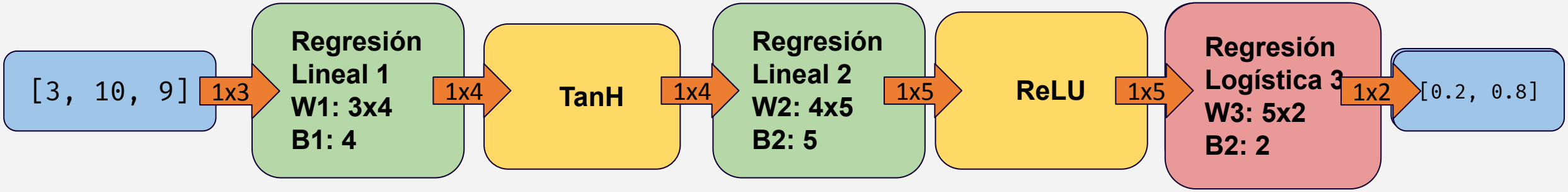


En general, b x n (b = batch_size)



Redes Neuronales - Clasificación

Modificar la red anterior para clasificar:

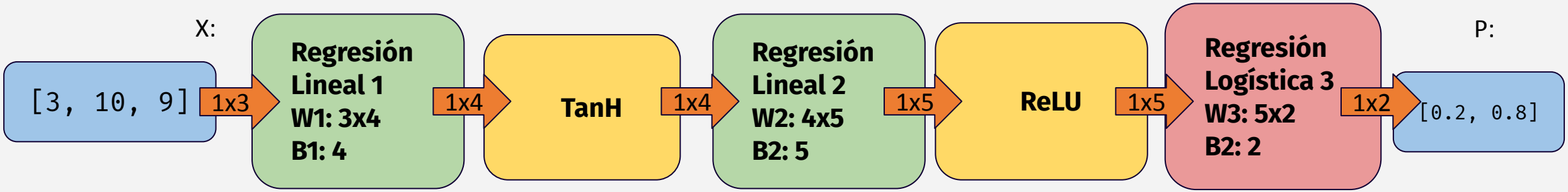


- Capas intercambiables
  - Pero verificar tamaños
- Cambio última capa:
  - Cambia el tipo de dato de salida
- Última capa se llama cabeza o head.

Clasificación Con Redes + Keras

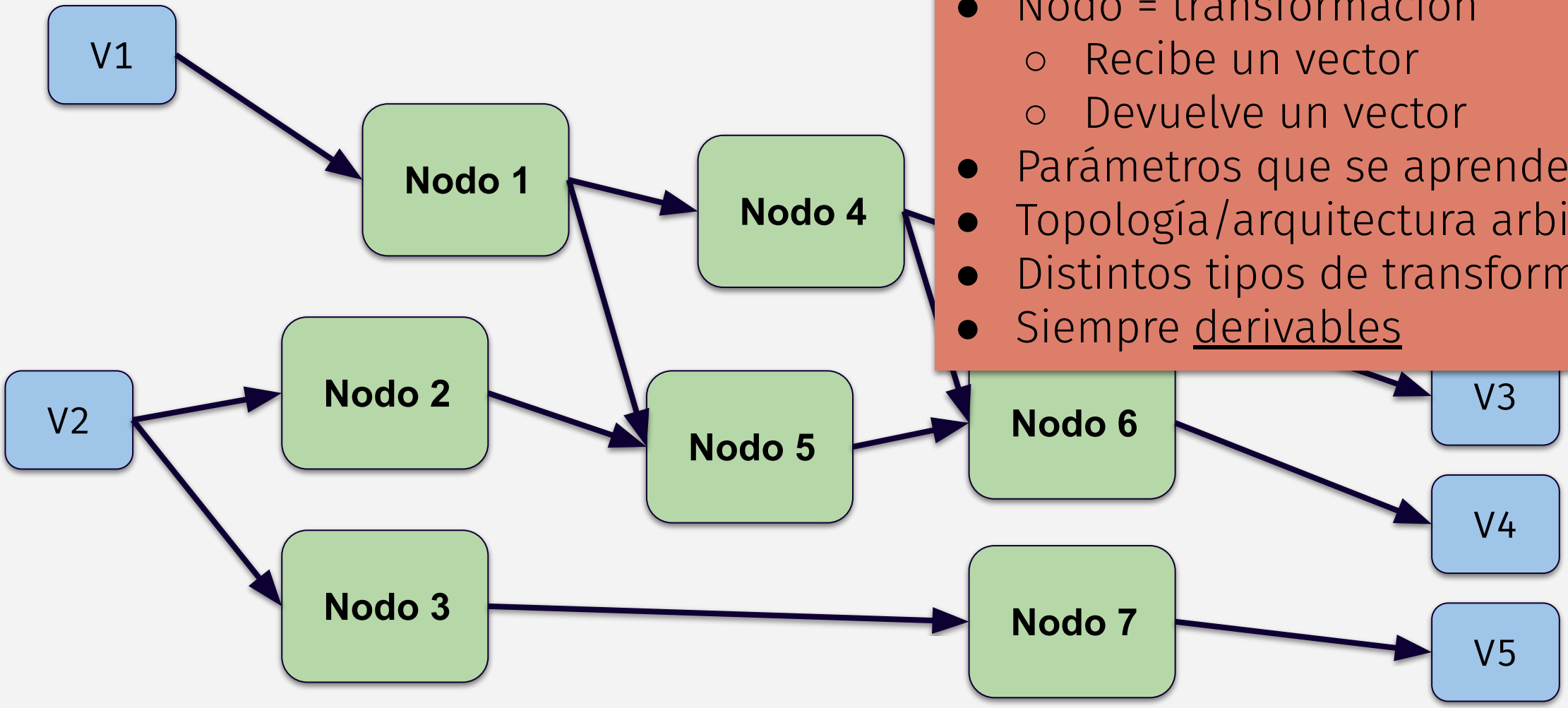
```
x,y=cargar_dataset(one_hot=True)
nx,d_in  = x.shape # x tiene tamaño n x d_in
ny,d_out = y.shape # y tiene tamaño n x d_out
import keras

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)
```

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*
- Algoritmo de entrenamiento para cualquier red
  - Si todas sus capas son derivables

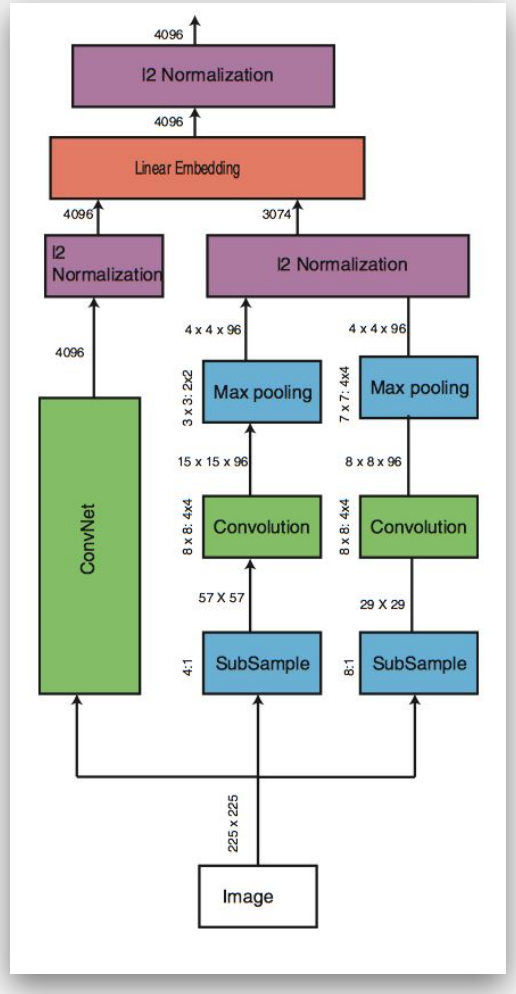
Redes Neuronales Generales



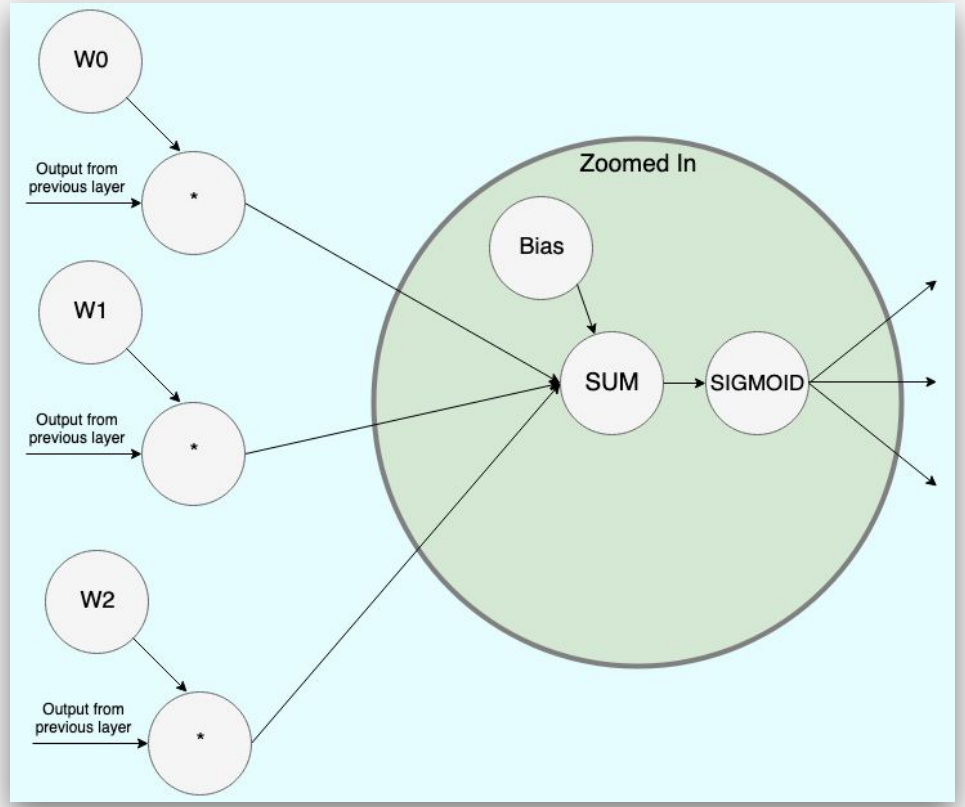
- Grafos computacionales
- Nodo = transformación
  - Recibe un vector
  - Devuelve un vector
- Parámetros que se aprenden
- Topología/arquitectura arbitraria
- Distintos tipos de transformaciones
- Siempre derivables

Redes Neuronales Generales - Escalas

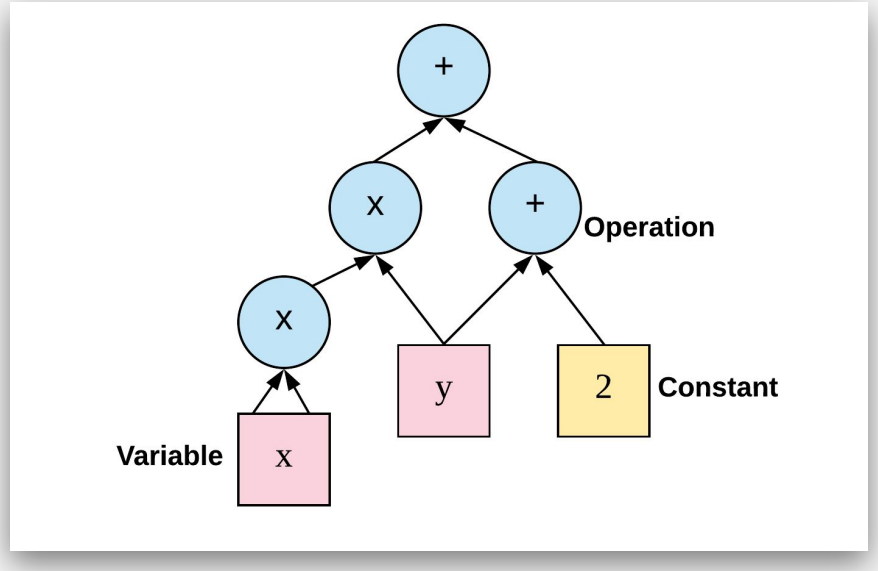
Capas (nivel alto)



Vectores (nivel medio)



Números (nivel bajo)



Redes Neuronales Generales - Nomenclatura

- Nomenclatura inversa
  - Nodos = valores
  - Aristas = funciones/capas
- Interpretación biológica
  - **Nodos** = Neuronas, con valor de *activación*
  - **Aristas** = Sinapsis *transformadoras*
- Se usan ambas nomenclaturas
  - Aprender ambas y a distinguir