P3 - TP2

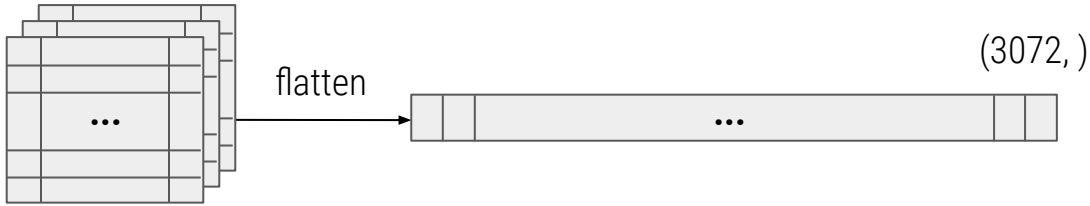
Aprendizaje Profundo y Redes Neuronales Artificiales
Materia Optativa -Instituto Balseiro - 2020

P3 - TP2: Red Neuronal Densa para CIFAR-10

- CIFAR-10

(32, 32, 3)

- Dos capas totalmente conectadas → **Capa 1: 100** Neuronas (Sigmoid) - “Oculta”
  Capa 1: 10 Neuronas (Lineal) - “Salida”
- Función costo → MSE
- Regularización L2
- Entrada → Vector 1D (3072)



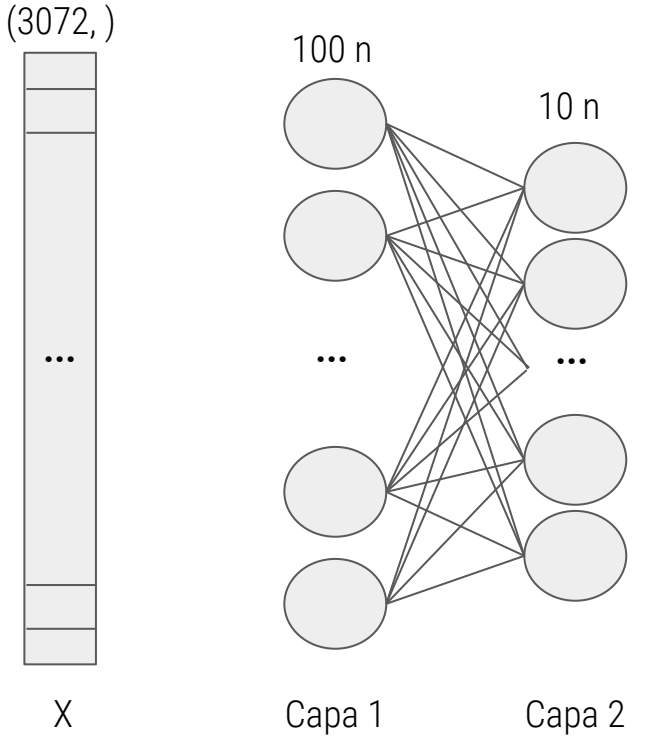
- Salida → Vector 1D (10) → Clase 4:

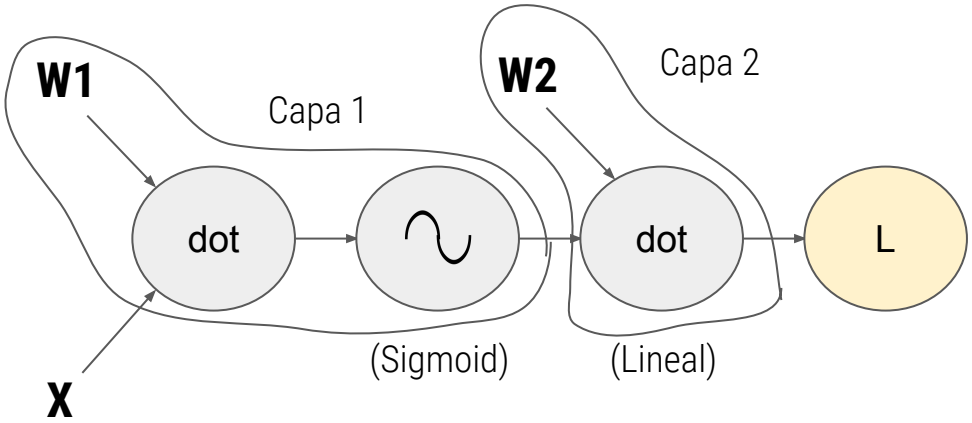
0	0	0	**1**	0	0	0	0	0	0

- Armar grafo computacional

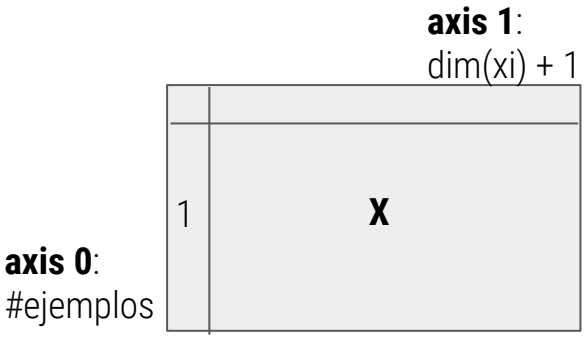
P3 - TP2: Red Neuronal Densa para CIFAR-10

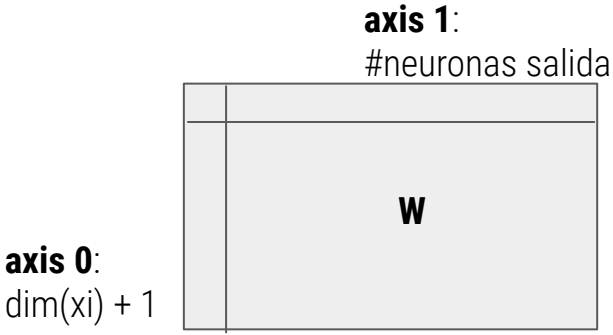
- Dos capas totalmente conectadas
  - **Capa 1: 100** Neuronas (Sigmoid) - “Oculta”
  - **Capa 1: 10** Neuronas (Lineal) - “Salida”

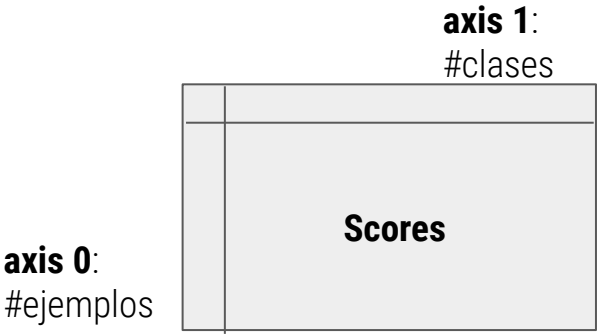




Convención de “axis”





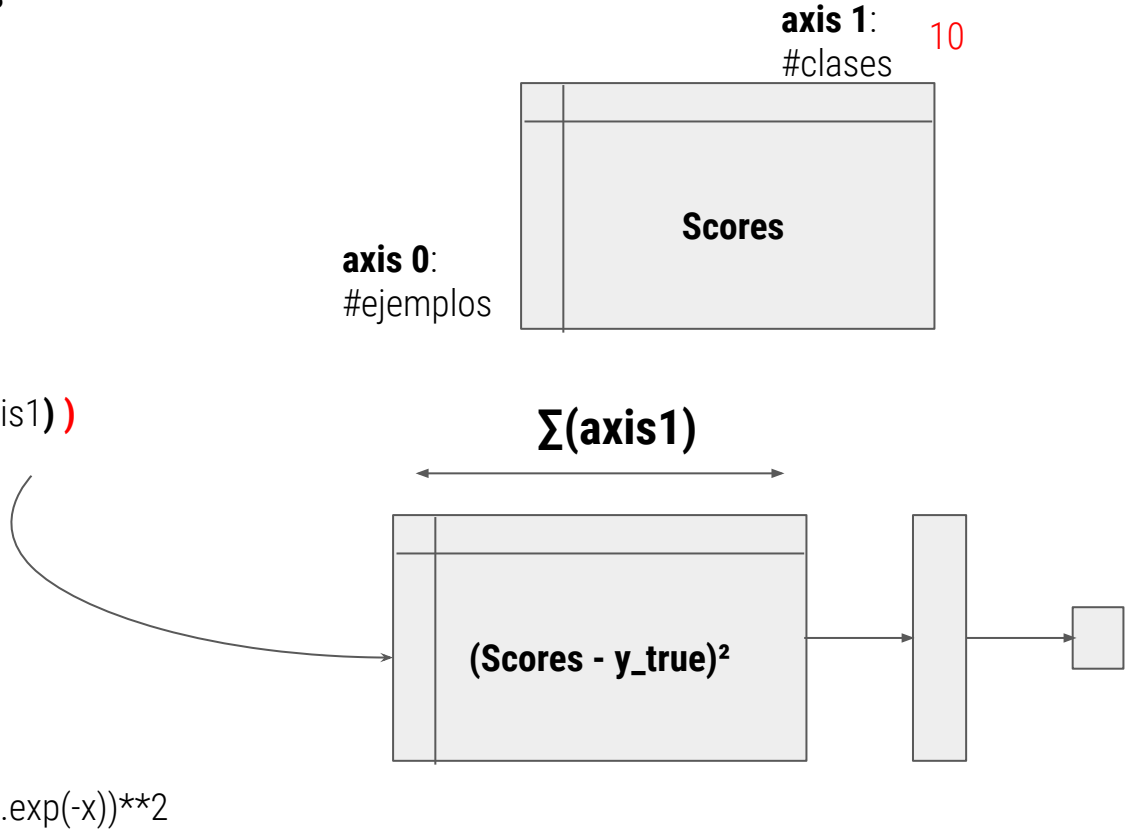


Cada fila de X es un ejemplo “xi”

Cada fila de S tiene los scores de cada clase

Definición de métricas, f. act, y gradientes


```
# métrica acc
acc (scores, y_true):
    y_pred ← argmax(scores, axis1)
    return media(y_pred == y_true)

# métrica MSE
MSE (scores, y_true):
    mse ← media( suma( (scores - y_true)² , axis1) )
    return mse

# Gradiente MSE
grad_mse (scores, scores_true):
    grad_mse ← 2* (scores - y_true)
    return grad_mse

# Sigmoid
1/(1+np.exp(-x))

# grad_sigmoid
np.exp(-x)/(1+np.exp(-x))**2
```

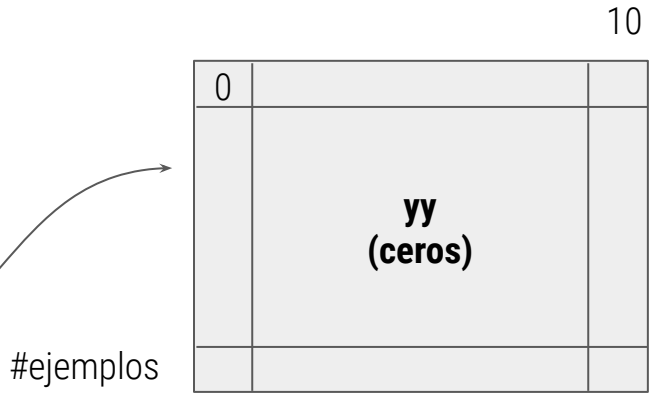


Parámetros

```
n_clases ← 10
n_neuronas ← 100
idx ← arange(#ejemplos)
# epochs, batch_size, lr, reg_factor, dim_x, ...
```

Preprocesado

```
x_train, y_train, x_test, y_test ← importamos datos
…  # reshape de los datos, flatten

# Pasar los datos Y al formato pedido (train, test)
# Para cada imagen, y_train = [0  0 1 … 0]
yy_train ← array_zeros(#ejemplos, #clases)
yy_train[np.arange(#ejemplos), y_train] = 1

# Restamos la media del batch
x_train ←  x_train - media(x_train, axis=0)
x_test ←  x_test - media(x_train, axis=0)
```

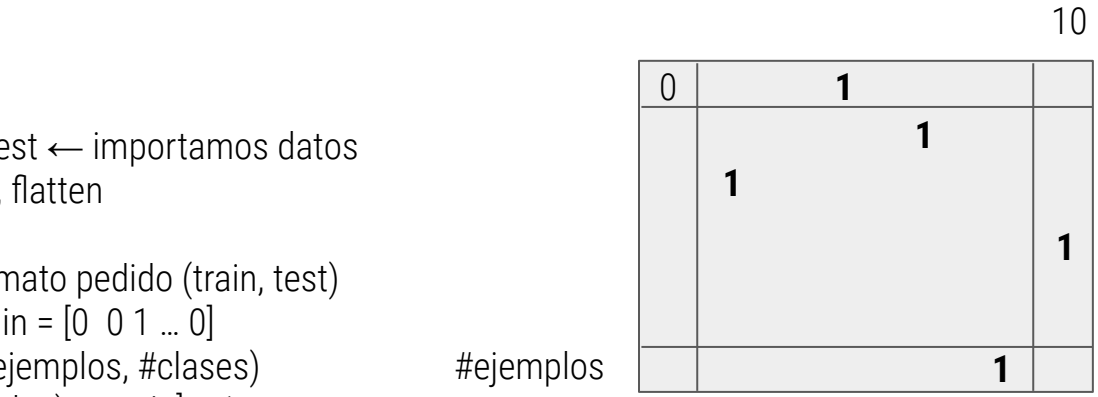


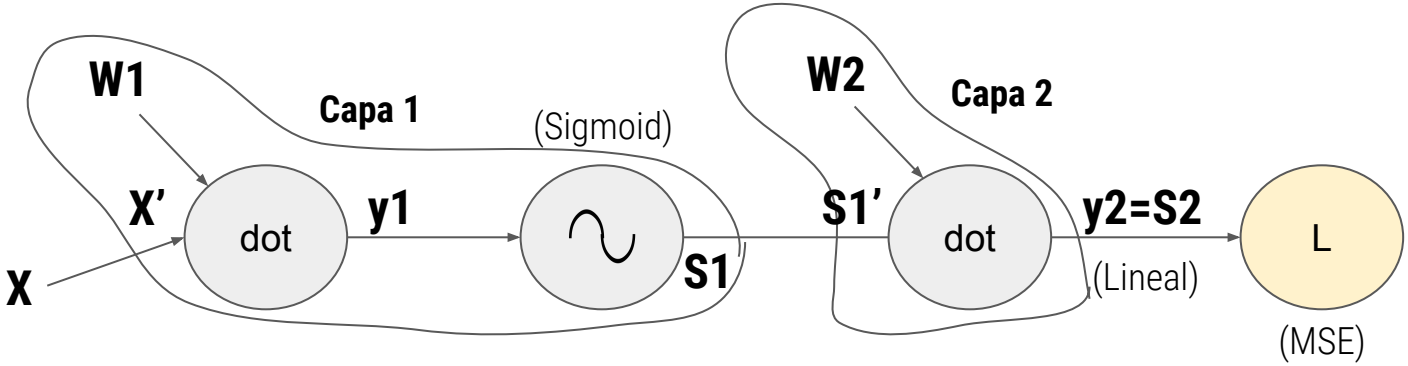
Parámetros

```
n_clases ← 10
n_neuronas ← 100
idx ← arange(#ejemplos)
# epochs, batch_size, lr, reg_factor, dim_x, ...
```

Preprocesado

```
x_train, y_train, x_test, y_test ← importamos datos
…  # reshape de los datos, flatten

# Pasar los datos Y al formato pedido (train, test)
# Para cada imagen, y_train = [0  0 1 … 0]
yy_train ← array_zeros(#ejemplos, #clases)
yy_train[np.arange(#ejemplos), y_train] = 1

# Restamos la media del batch
x_train ←  x_train - media(x_train, axis=0)
x_test ←  x_test - media(x_train, axis=0)
```

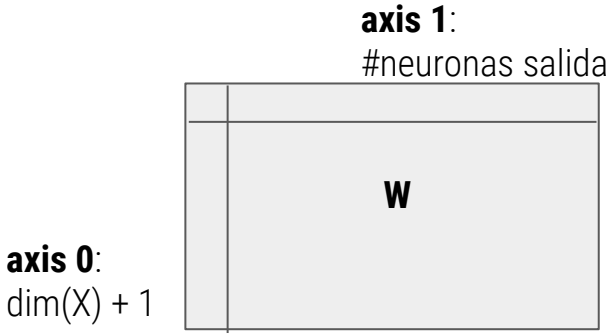




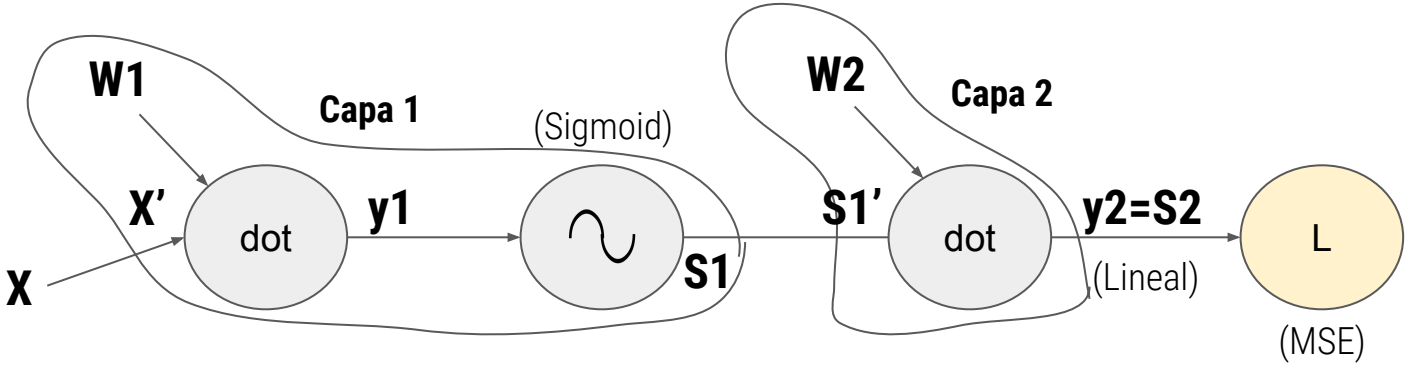
Nomenclatura:
dim(X): dimensión de una fila de X. Equivalente al #columnas de X

Inicialización de los pesos

w1 ← Pesos aleatorios (dim(X) + 1, 100) = (dim (X') , 100) **(3073, 100)**
w2 ← Pesos aleatorios (dim(S1) + 1, 10) = (dim(S1'), 10) **(101, 10)**



Con esta convención:
La matriz de pesos siempre tiene dimensiones: (#número de columnas de la entrada + 1, número neuronas salida)



Nomenclatura:
dim(X): dimensión de una fila de X. Equivalente al #columnas de X

---

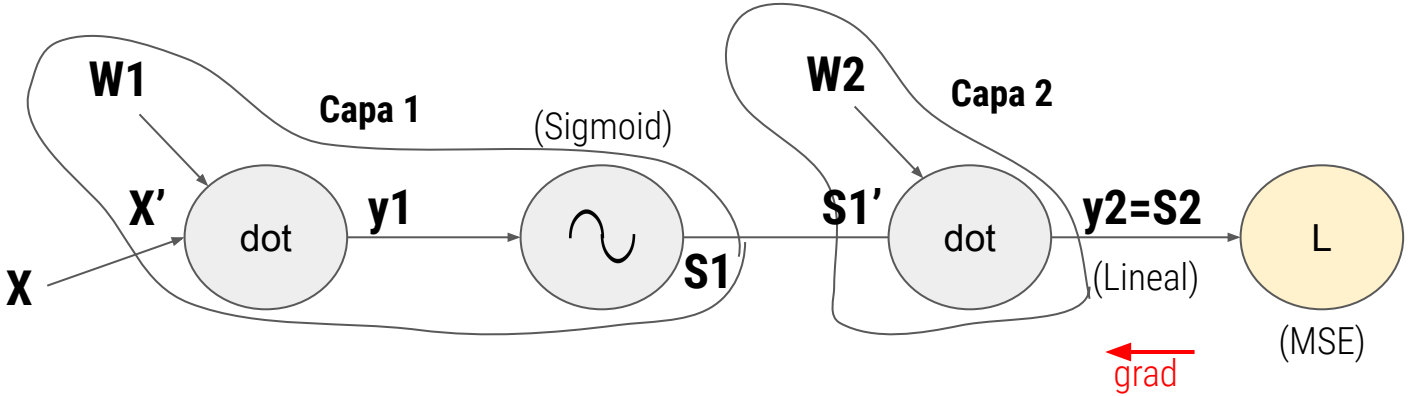
loop en épocas

```
# Mezclamos índices (shuffle, por ej)
loss, acc ← 0, 0
```

loop en iteraciones (#batchs)

```
# Nos quedamos con batch "it"
id_batch ← idx[ it*batch_size : (it+1)*batch_size ]
Xb, Yb ← X, Y evaluados en los índices
# Armamos Xb' (agregamos columna de 1s)
```

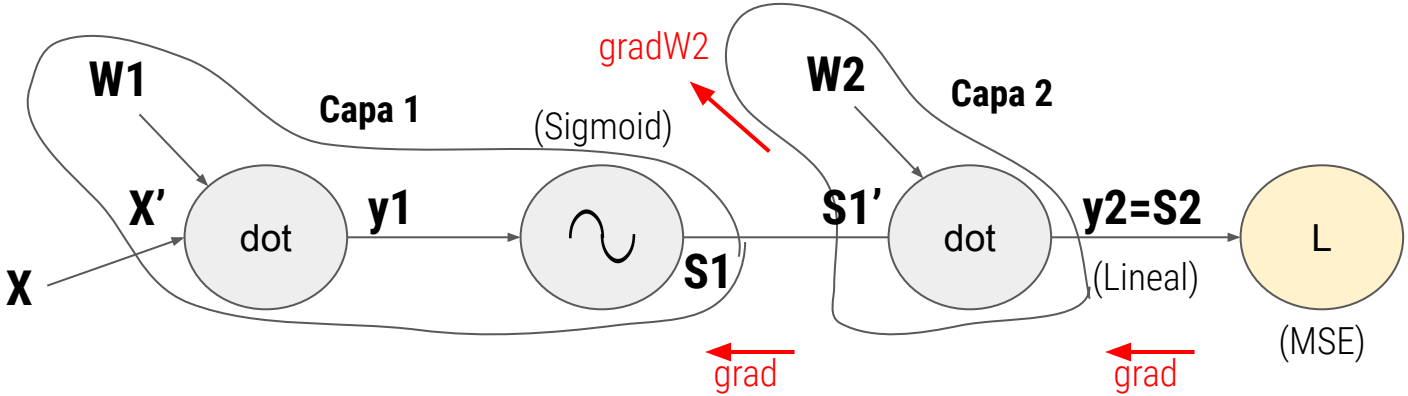
```
# Forward: Calculamos S1 y S2
S1← sigmoid( dot(Xb',W1) )
S1' ← Agregamos columna de 1s
S2← dot( S1', W2 )

# Calculamos la regularización
reg1 ← suma( W1**2 )
reg2 ← suma( W2**2 )
reg ← reg1 + reg2

# Calculamos Loss
loss ← loss + MSE(S2, Yb) + 0.5(reg)
```



Nomenclatura:
dim(X): dimensión de una fila de X. Equivalente al #columnas de X

Calculamos accuracy acumulado
acc ← acc + metric_acc(S2, Yb)

Backward: Gradiente global
grad ← grad_MSE(S2, Yb) # No olvidar que falta el de reg

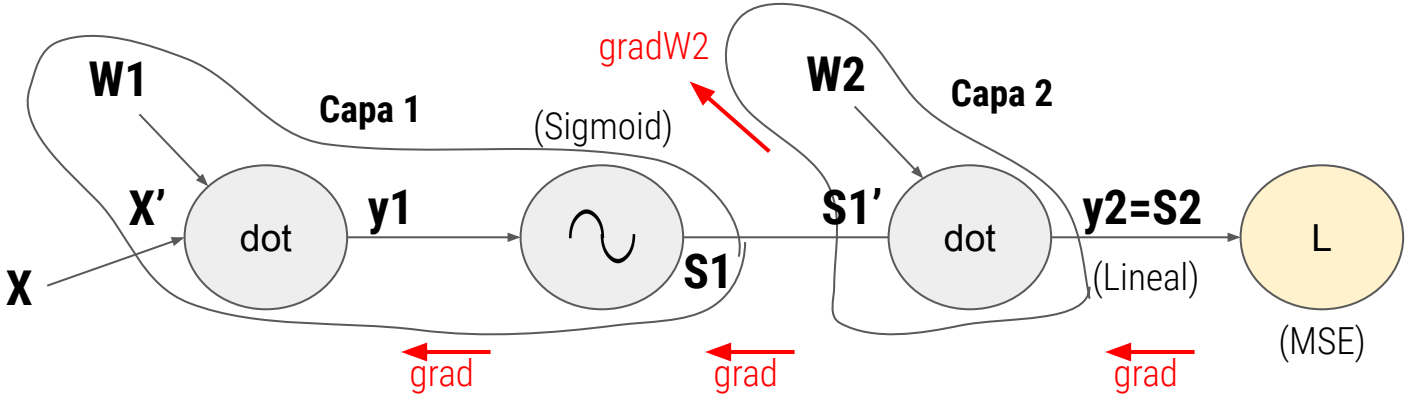


Nomenclatura:
dim(X): dimensión de una fila de X. Equivalente al #columnas de X

Calculamos accuracy acumulado
acc ← acc + metric_acc(S2, Yb)

Backward: Gradiente global
grad ← grad_MSE(S2, Yb) # No olvidar que falta el de reg

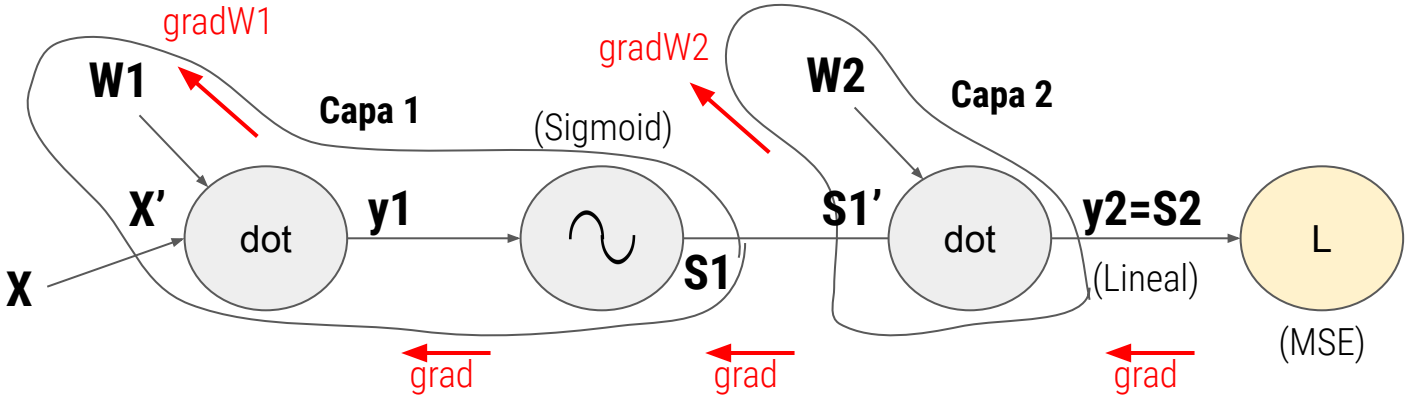
Capa 2
gradW2 ← dot (S1'.T, grad) # grad_local * grad
grad ← dot (grad , W2.T) # grad * grad_local
grad ← grad [: , 1:] # quitamos la primer columna "bias"



Nomenclatura:
dim(X): dimensión de una fila de X. Equivalente al #columnas de X

```
# Calculamos accuracy acumulado
acc ← acc +  metric_acc(S2, Yb)

# Backward: Gradiente global
grad ← grad_MSE(S2, Yb) # No olvidar que falta el de reg


# Capa 2
gradW2 ← dot ( S1'.T, grad ) # grad_local * grad
grad ← dot ( grad , W2.T ) # grad * grad_local
grad ← grad [ : , 1: ] # quitamos la primer columna "bias"
```

```
# Capa 1
grad_sig ← grad_sigmoid(Y1)
grad ← grad * grad_sig
```



Nomenclatura:
dim(X): dimensión de una fila de X. Equivalente al #columnas de X

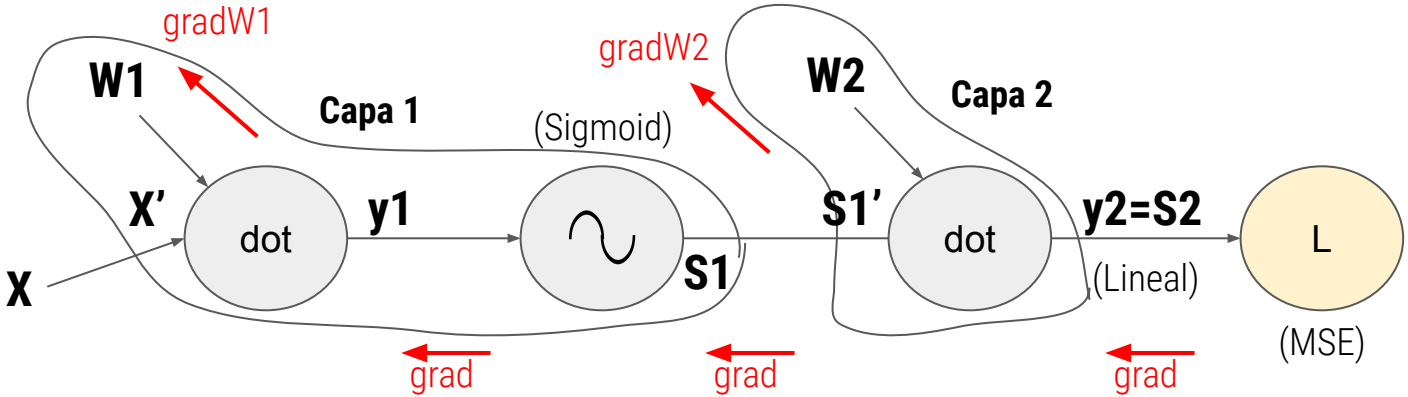
Calculamos accuracy acumulado
acc ← acc + metric_acc(S2, Yb)

Backward: Gradiente global
grad ← grad_MSE(S2, Yb) # No olvidar que falta el de reg

Capa 2
gradW2 ← dot (S1'.T, grad) # grad_local * grad
grad ← dot (grad , W2.T) # grad * grad_local
grad ← grad [: , 1:] # quitamos la primer columna "bias"

Capa 1
grad_sig ← grad_sigmoid(Y1)
grad ← grad * grad_sig

gradW1 ← dot (Xb'.T, grad) # grad_local * grad



Nomenclatura:
dim(X): dimensión de una fila de X. Equivalente al #columnas de X

Actualizamos los pesos
W1 ← W1 - lr * (gradW1 + reg*W1)
W2 ← W2 - lr * (gradW2 + reg*W2)

Promediamos las loss y acc de los batchs
...

Calculamos accuracy con los datos de test
(**Forward** con X_test)

Printeamos por pantalla precisión