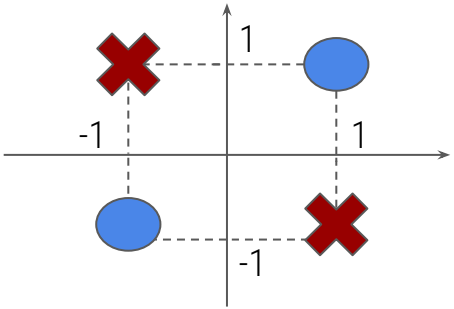
P6 - TP2

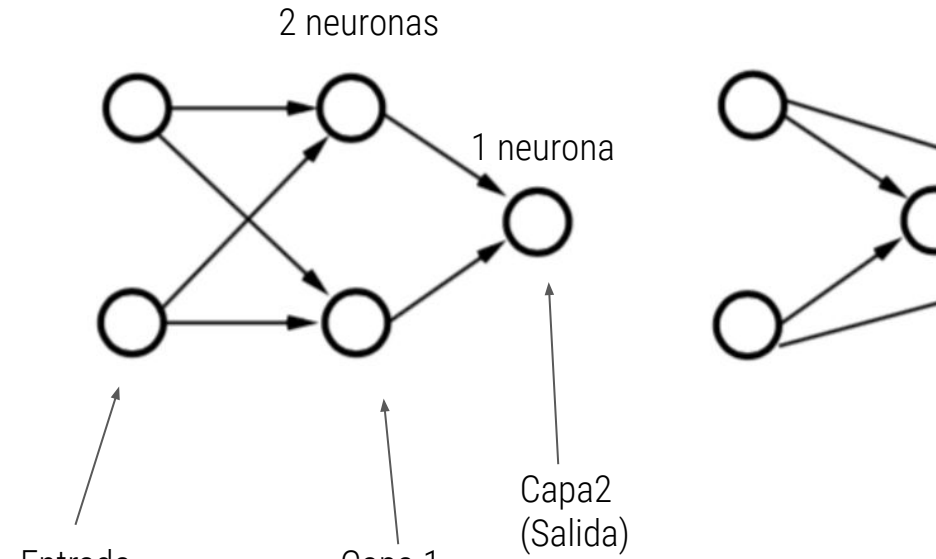
Aprendizaje Profundo y Redes Neuronales Artificiales
Materia Optativa -Instituto Balseiro - 2020

P6 - TP2: Problema del XOR - POO

- Problema de XOR

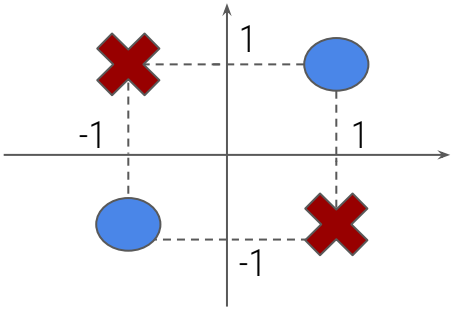


- Función activación ⟶ tanh
- Función costo ⟶ MSE

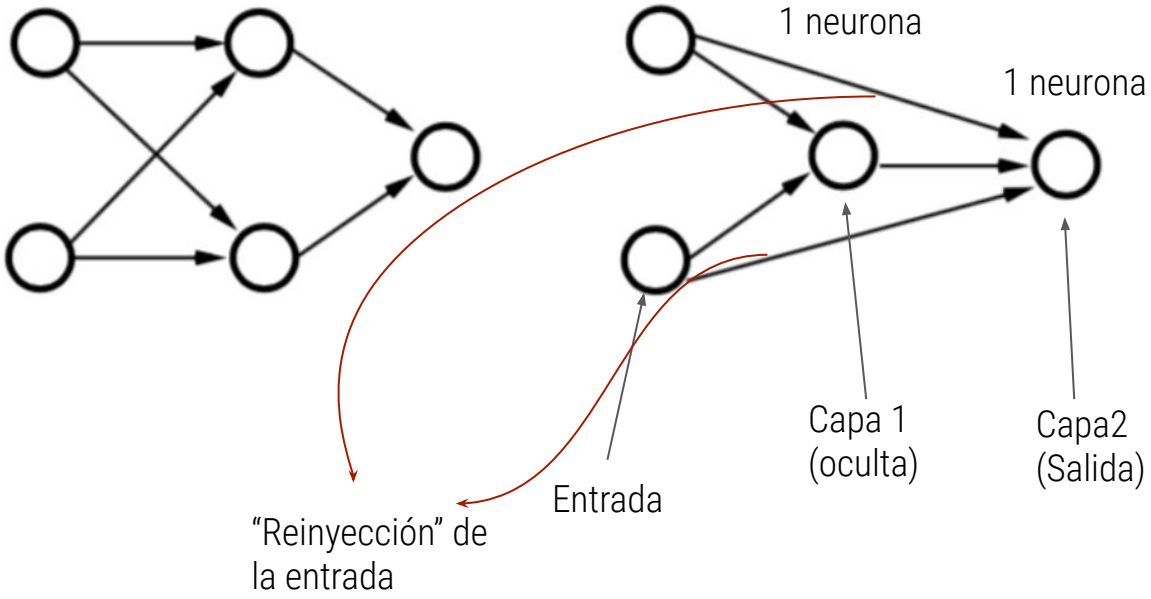


P6 - TP2: Problema del XOR - POO

- Problema de XOR

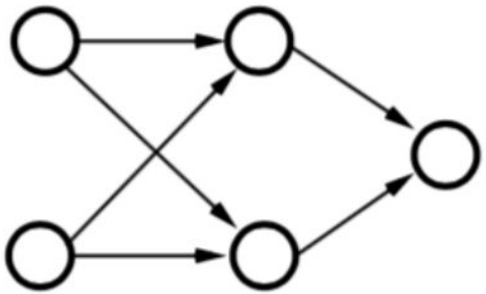


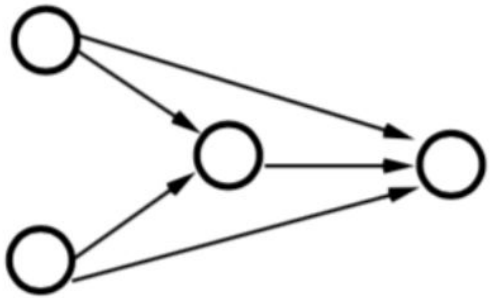
- Función activación ⟶ tanh
- Función costo ⟶ MSE

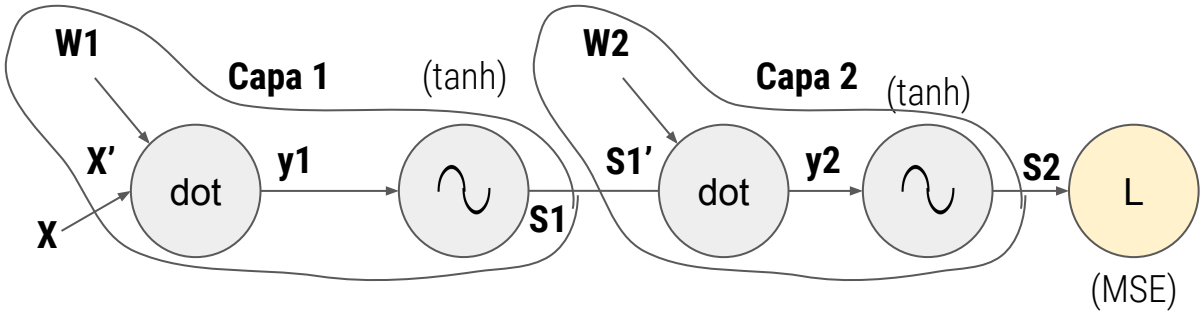


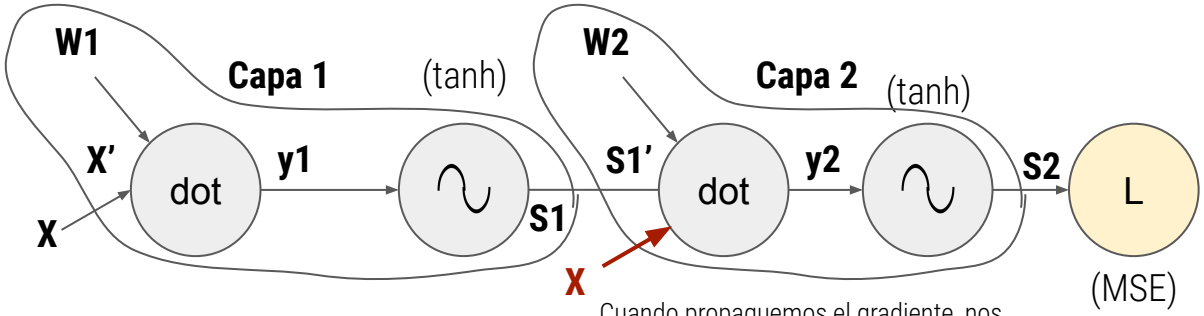
P6 - TP2: Problema del XOR - POO

¿Cómo armaríamos el grafo?



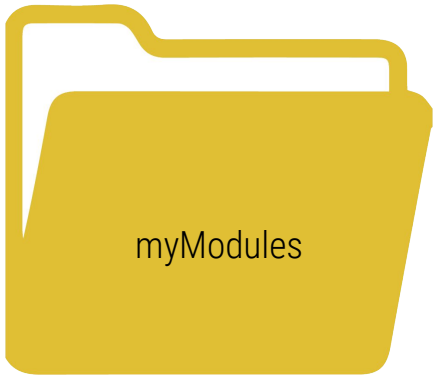


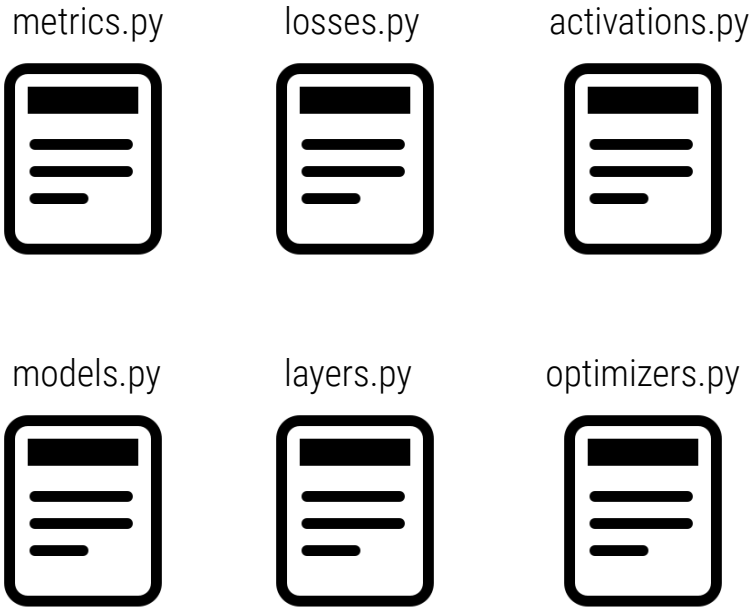




Cuando propaguemos el gradiente, nos quedamos con la parte de S1, descartamos lo de X (Similar a como hacíamos con el "bias", quitando columnas)

P6 - TP2: Problema del XOR - POO

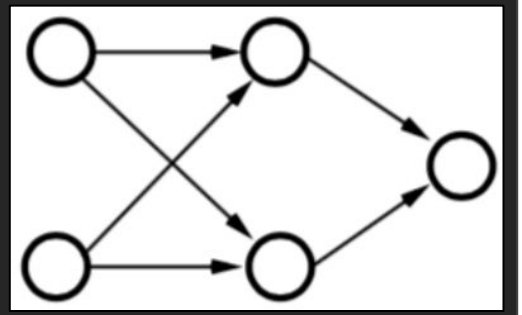




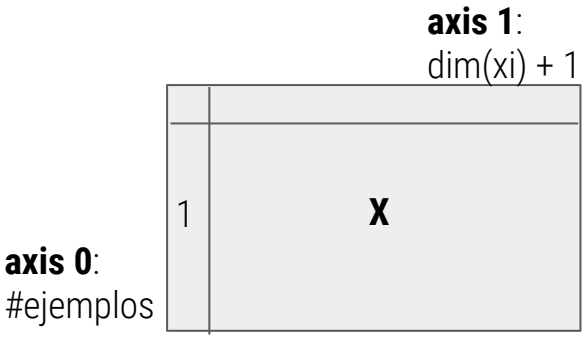
¿Cómo queremos usar los objetos?

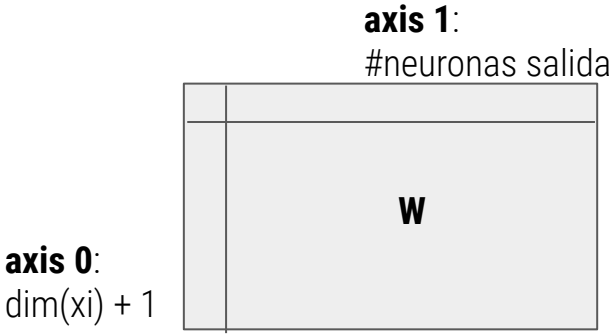
P6 - TP2: Problema del XOR - POO

```
# Identify the problem: 'XOR' or 'Image'
problem_flag = 'XOR' #P6

# Dataset
x_train = np.array([[-1,-1],[-1,1],[1,-1],[1,1]])
y_train = np.array([[1],[-1],[-1],[1]])

# Regularizer
reg1 = regularizers.L2(0.1)
reg2 = regularizers.L1(0.2)

# Create model
model = models.Network()
model.add(layers.Dense(units=2,activations.Tanh(),input_dim=x_train.shape[1], regularizer=reg1))
model.add(layers.Dense(units=1,activations.Tanh(), regularizer=reg2))

# Train network
model.fit(x_train=x_train,y_train=y_train,x_test=x_test,y_test=y_test,
        batch_size=x_train.shape[0], epochs=200, opt=optimizers.SGD(lr=0.05),
        problem_flag=problem_flag)
```



Convención de “axis”





metrics.py

métrica MSE

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

métrica acc

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

scores #clases

1.8		0.5
	...	
-2.2		-1.7

#ejemplos

métrica acc_xor

```
acc (scores, y_true):
    scores[scores > 0.5] ← 1
    scores[scores < -0.5] ← -1
    return media(scores == y_true)
```

scores

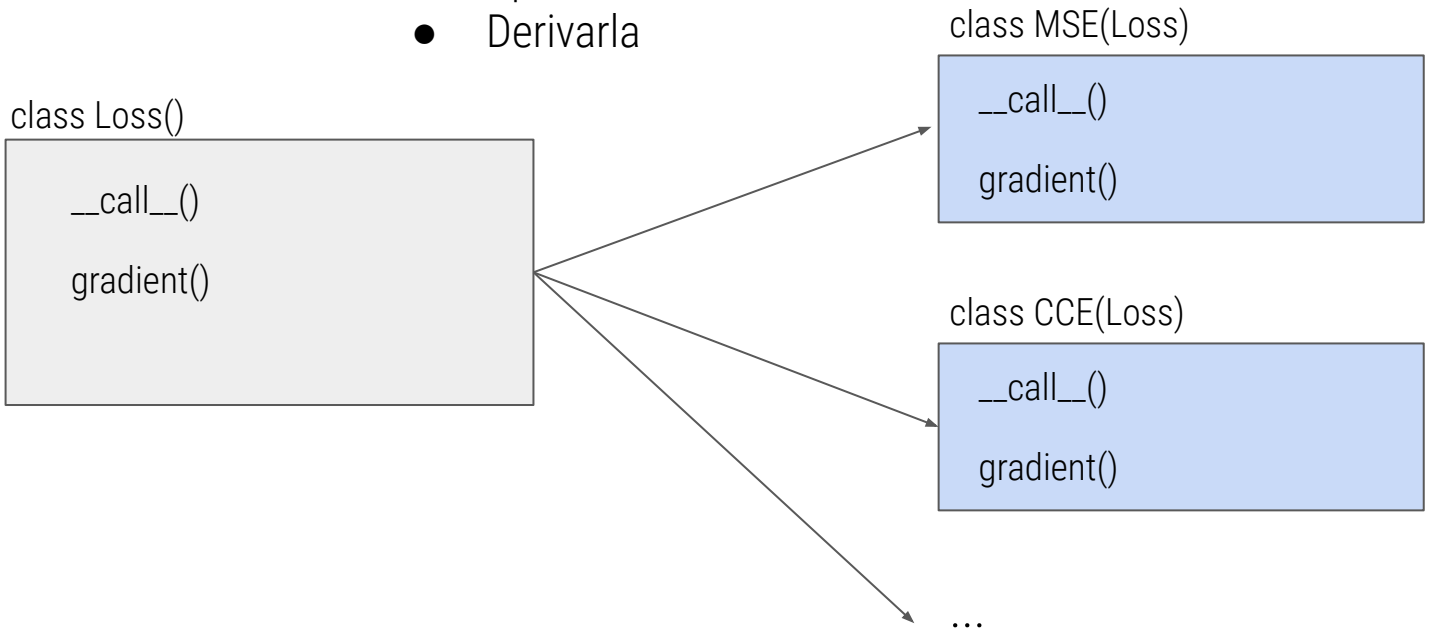
0.8
...
-0.2

#ejemplos

losses.py

¿Qué necesito de una Loss?

- Aplicarla
- Derivarla



activations.py

¿Qué necesito de una activación?

- Aplicarla
- Derivarla

class Activation()

```
def __call__(self):
    pass
def gradient(self):
    pass
```

class ReLu(Loss)

```
def __call__(self, x):
    ...
def gradient(self, x):
    ...
```

class Tanh(Loss)

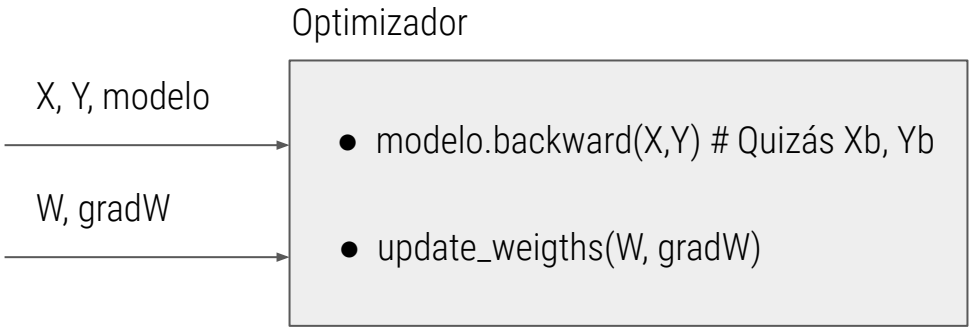
```
def __call__(self, x):
    ...
def gradient(self, x):
    ...
```

class Tanh(Loss)

class Linear(Loss)

optimizers.py

¿Qué debería poder hacer un optimizador?



Aplica alguna **regla particular de actualización** de los pesos
(ADAM, SGD, RMSProp, Adagrad, Adadelta, …)

https://www.tensorflow.org/api_docs/python/tf/keras/optimizers

optimizers.py

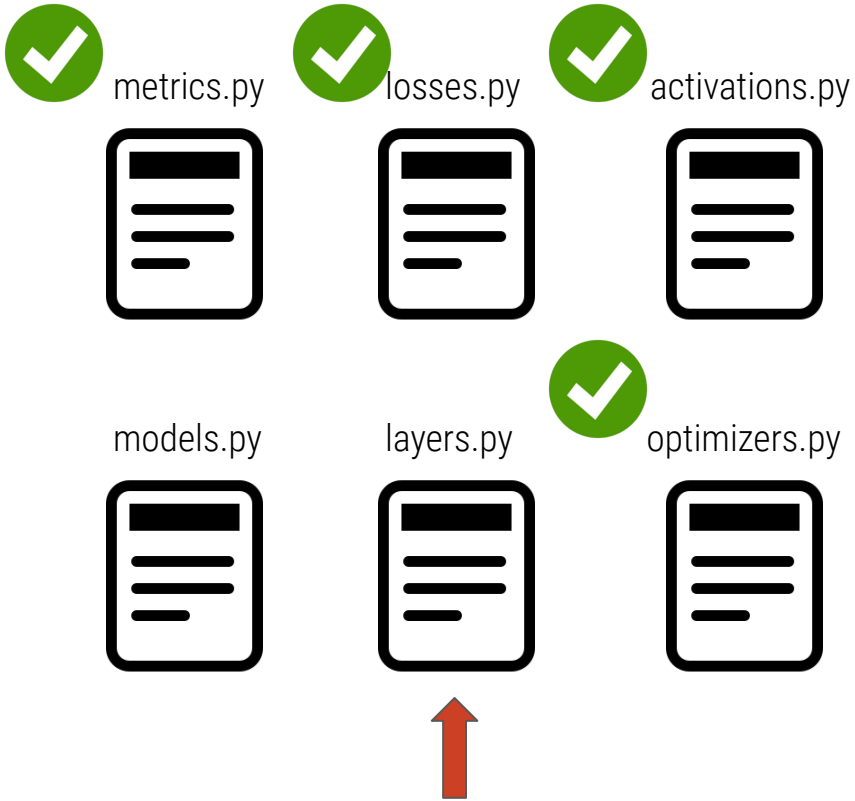
¿Cómo implementar el optimizador?

class Optimizer()

```
def __init__(self, lr, ..):
  self lr = lr
   ...

def __call__(self, X, Y, model):
   pass

def update_weights(self, W, gradW):
   pass
```

class SGD(Loss)

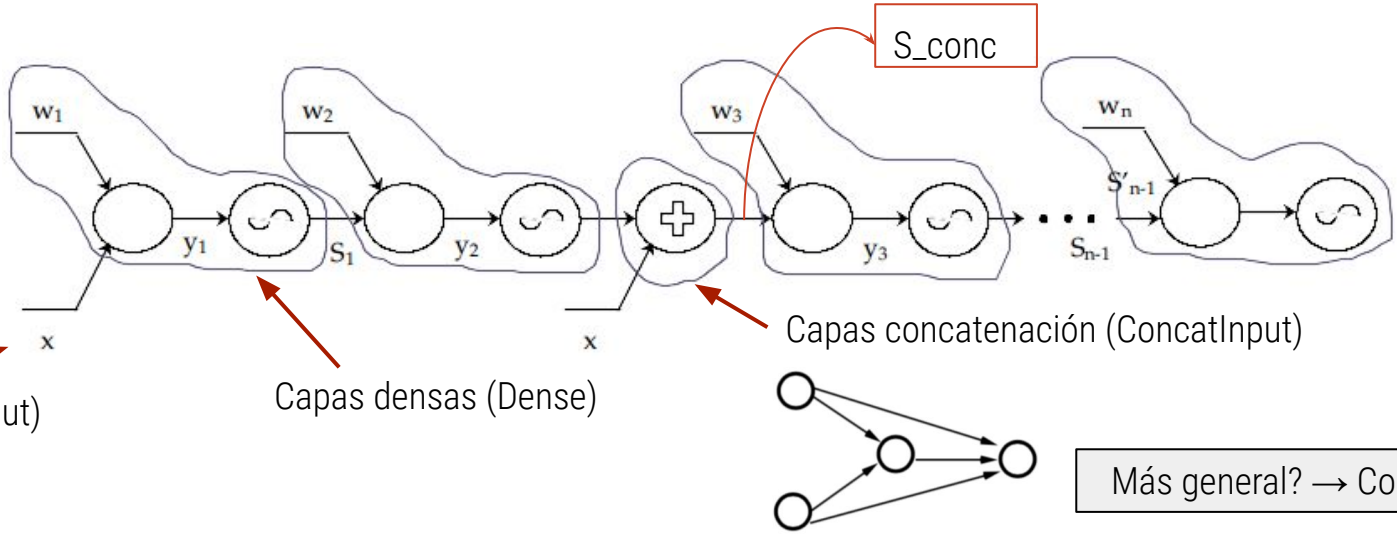
```
def __init__(self, lr, ..., bs):
  super().__init__(lr, ...)
  self.bs = bs
   ...

def __call__(self, X, Y, model):
   # Decidir si armar o no batchs
   model.backward(X, Y)

def update_weights(self, W, gradW):
   W -= self.lr * gradW # SGD
```



Hay muchas muchas formas de implementación...

Layers

layers.py

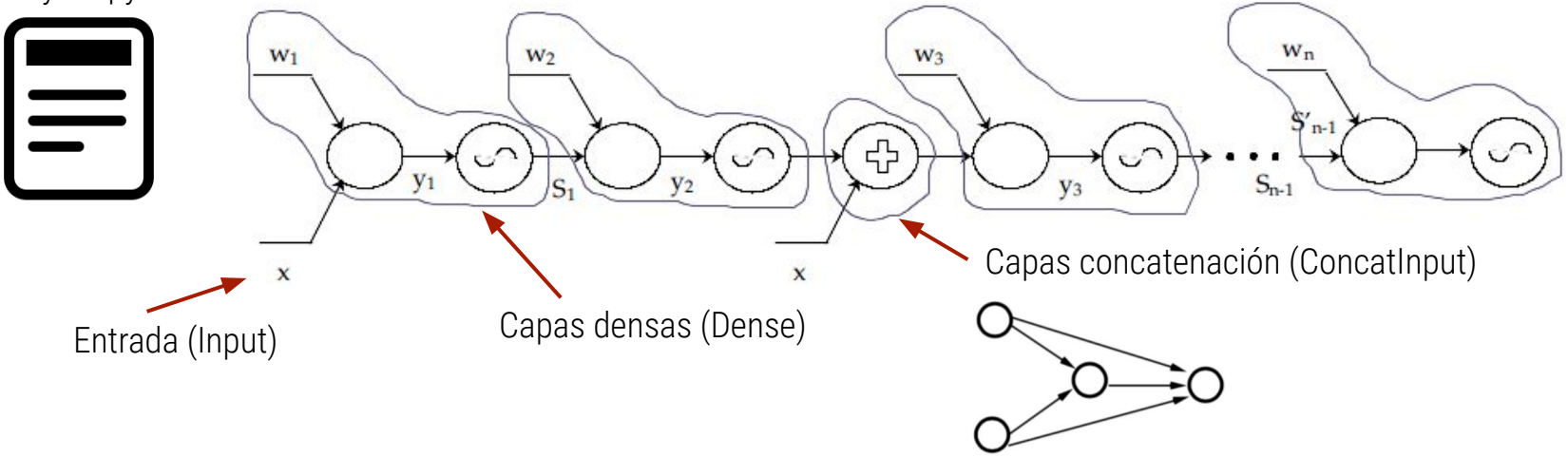


Qué querríamos de un objeto Layer??

```
l1 = Dense(#unidades_1, …) # La creamos
l2 = Dense(#unidades_2, …)

S1 = l1(x)  # La llamamos (__call__)
S2 = l2(S1)
```

```
l0 = Input(X_dim)   # La creamos
l3 = Concat(...)    # La creamos

S_conc = l3(l0, l2)  # La llamamos (__call__)
```

layers.py



Capas especiales: Sin pesos

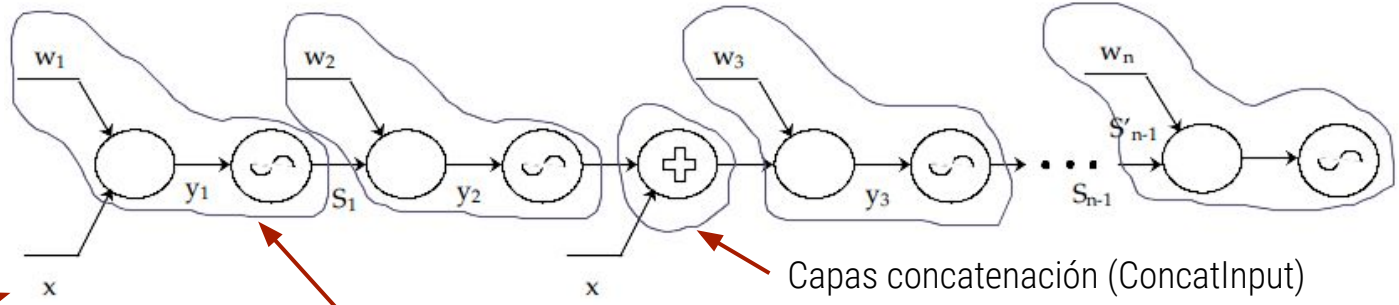
Input

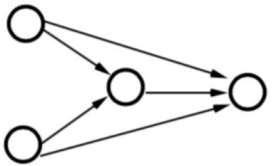
- Necesitamos saber el tamaño X: dim(x_i)
- Necesitamos poder pedirle el tamaño de salida:
  output_dim = dim(x_i)

ConcatInput

- Necesitamos poder llamarla (concatenación)
- Necesitamos conocer los tamaños de cada entrada (X y S_{j-1})
- Necesitamos poder pedirle el tamaño de salida:
  output_dim = dim(x_i) + dim(S_{j-1})

layers.py





Capas densas: Dense → Una clase padre para cualquier capa con pesos

WLayer(BaseLayer)

- Necesitamos saber el tamaño X: dim(x_i)
- Necesitamos saber la activación
- Acceder a sus pesos (get_W)
- Actualizar sus pesos (update_W)
- Setear (o no) regularizador

Dense (WLayer)

- Inicializar pesos
- __call__(X)
- Se me ocurrió método "**dot**" (devuelve y)

Resumen: Una forma posible...

layers.py

BaseLayer()

```
__init__()
  pass
get_output_shape()
  pass
set_output_shape()
  pass
```

Input(BaseLayer)

```
__init__()
  ...
get_output_shape()
  ...
set_output_shape()
  ...
```

ConcatInput(BaseLayer)

```
__init__():
__call__()
set_output_shape()
get_output_shape()
get_input1_shape()
get_input2_shape()
```

WLayer(BaseLayer)

```
__init__(act, xdim):
get_input_shape():
set_input_shape():
get_output_shape():
set_output_shape():
get_weights():
update_weights():
```

→ input_shape = (None, xdim)

No nos importa el nro de ejemplos en la capa

Dense(WLayer)

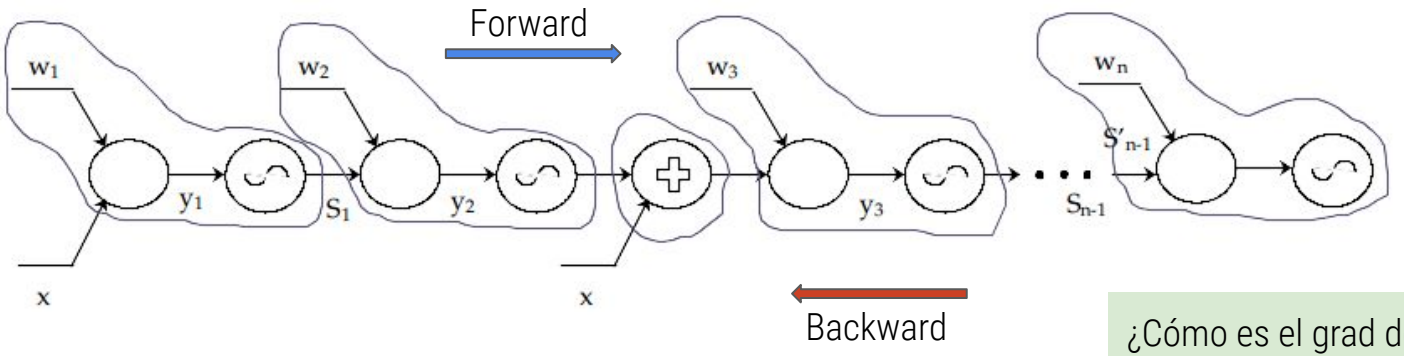
```
init_weights():
__call__():
dot():
```

→ Devuelve act(dot(x,W)

→ Construye X' y devuelve X'.W

Models

models.py



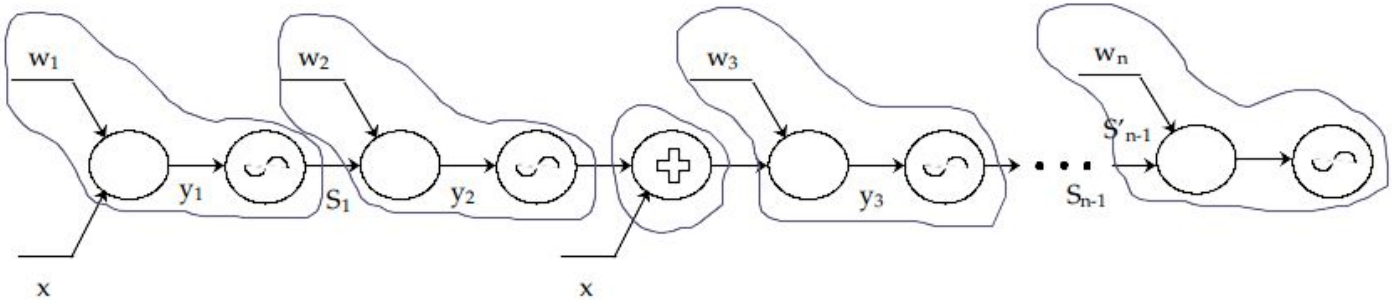
¿Cómo es el grad de la capa ConcatInput?

Descartar los elementos correspondientes a X. Pasar los correspondientes a S2 (ojo bias)

Qué querríamos de un objeto Network??

- Sepa agregar y gestionar capas (llevar su orden, conectarlas, usar los métodos de cada capa, etc)
- Sepa realizar un paso de forward (hasta la capa “j” mejor)
- Sepa realizar un paso de backward (use un optimizador)
- Sepa predecir con un conjunto de datos (forward + devolver predicción)

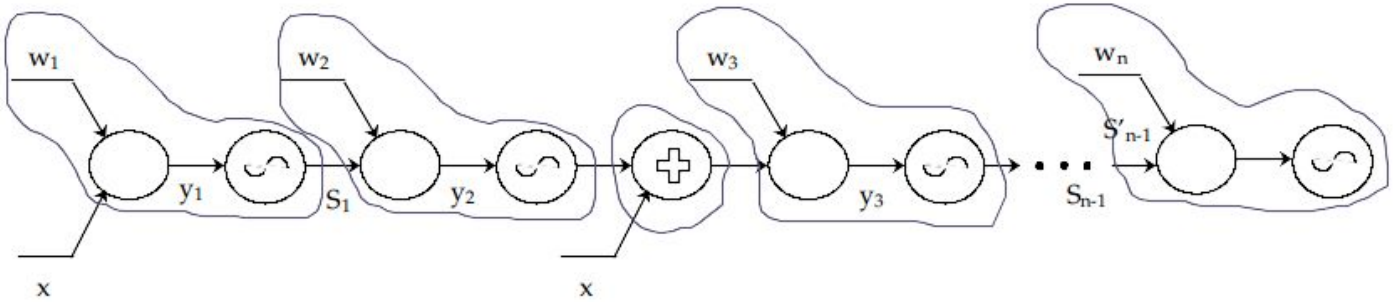
models.py



Clase Network()

- __init__()
  # Definir una lista (vacía) de capas
- add(*layer*)
  # Chequea tipo de *layer* para agregarla
  # Resuelve input_dim de *layer* si no es la primera (output_dim de la anterior)
  # *layer.init_weights()* shapeW → (dim(X) + 1, output_dim)
- get_layer(*number_of_layer*)
  # devuelve la layer pedida del modelo
- fit (*x, y, test_data = [None,None], lr = 1e-3, epochs = 100, bs = 200, loss = losses.MSE(), opt = optimizers.SGD()*)
  # Loop de épocas.
    #Se llama al optimizador (realice backward, decide si usar o no batchs según bs)
    #Se calcula loss (MSE) y métrica (Acc_XOR)
    #Se evalúa loss_test y acc_test si test_data != None
    #Print por pantalla

models.py

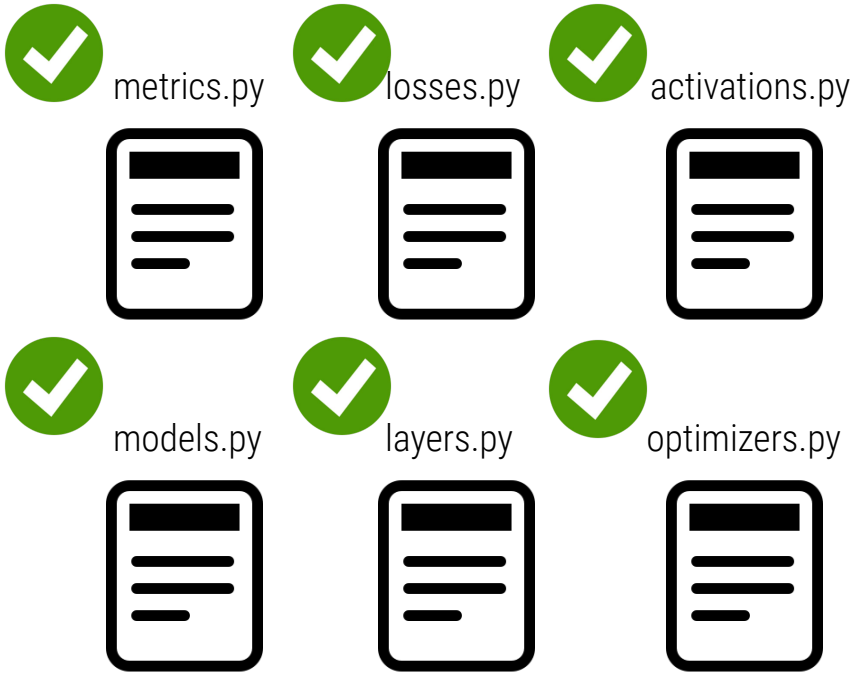


Clase Network()

→ forward_upto(j)
  # Realiza el proceso forward hasta la capa "j"

→ predict(*X*)
  # forward_upto(*last_layer*) y devuelve la predicción (argmax, por ejemplo)

→ backward()
  # Realiza el proceso de backward con el resultado de forward capa por capa: recorre las capas hacia atrás y va calculando los gradientes gradWi, llamando al optimizador para que haga update_Wi

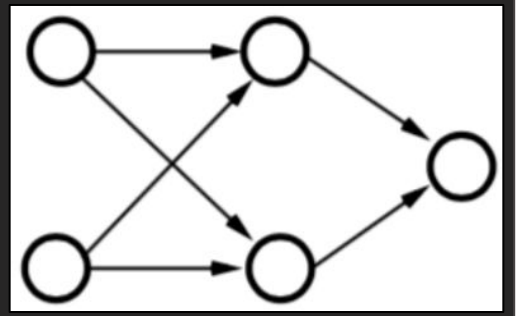


(Extra)

regularizers.py

L2 y gradL2
L1 y gradL1

```
# Identify the problem: 'XOR' or 'Image'
problem_flag = 'XOR' #P6

# Dataset
x_train = np.array([[-1,-1],[-1,1],[1,-1],[1,1]])
y_train = np.array([[1],[-1],[-1],[1]])

# Regularizer
reg1 = regularizers.L2(0.1)
reg2 = regularizers.L1(0.2)

# Create model
model = models.Network()
model.add(layers.Dense(units=2,activations.Tanh(),input_dim=x_train.shape[1], regularizer=reg1))
model.add(layers.Dense(units=1,activations.Tanh(), regularizer=reg2))

# Train network
model.fit(x_train=x_train,y_train=y_train,x_test=x_test,y_test=y_test,
        batch_size=x_train.shape[0], epochs=200, opt=optimizers.SGD(lr=0.05),
        loss=losses.MSE(), problem_flag=problem_flag)
```



```
# Identify the problem: 'XOR' or 'Image'
problem_flag = 'XOR' #P6

# Dataset
x_train = np.array([[-1,-1],[-1,1],[1,-1],[1,1]])
y_train = np.array([[1],[-1],[-1],[1]])

# Regularizer
reg1 = regularizers.L2(0.1)
reg2 = regularizers.L1(0.2)

# Create model
model = models.Network()

input_layer = layers.Input(x_train.shape[1])
model.add(layers.Dense(units=1,activations.Tanh(),input_dim=x_train.shape[1], regularizer=reg1))
model.add(layers.ConcatInput(input_layer))
model.add(layers.Dense(units=1,activations.Tanh(), regularizer=reg2))

# Train network
model.fit(x_train=x_train,y_train=y_train,x_test=x_test,y_test=y_test,
        batch_size=x_train.shape[0], epochs=200, opt=optimizers.SGD(lr=0.05),
        loss=losses.MSE(), problem_flag=problem_flag)
```