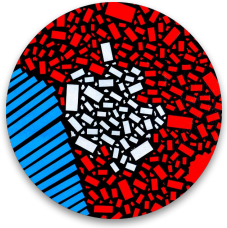


Fundamentos de Aprendizaje Automático y Reconocimiento de Patrones

Actividades en clase: Práctico 5

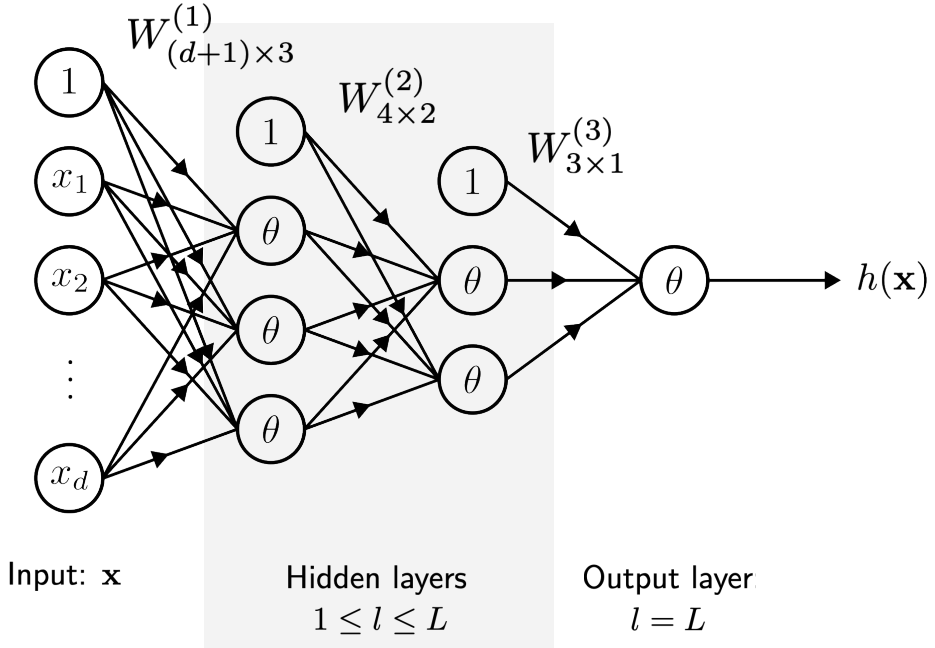
Graciana Castro, Martín Schmidt, Federico Lecumberry, Guillermo Carbajal

Instituto de Ingeniería Eléctrica

2024

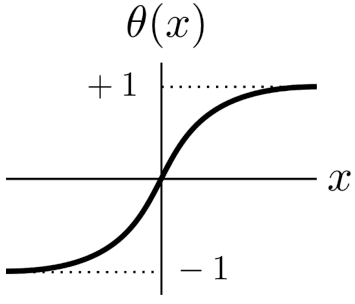
Tabla de contenido

Redes Neuronales



Activation Function

(differentiable!!!)

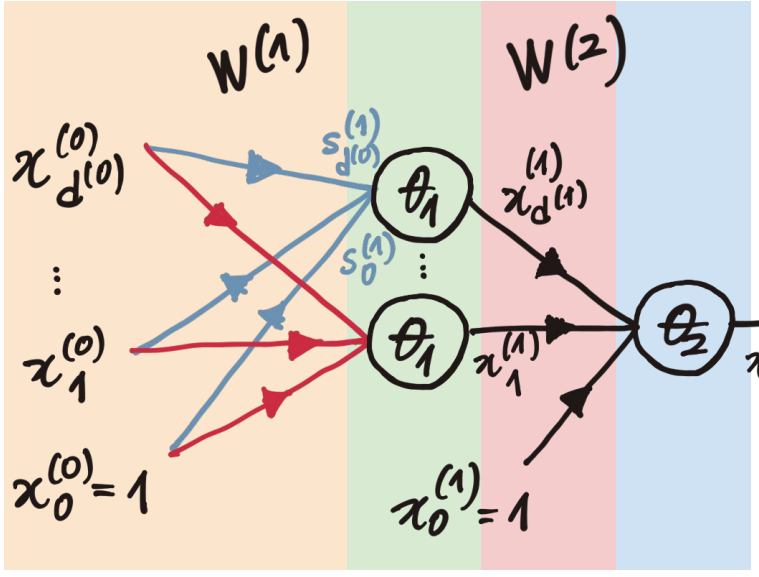


$$\theta(x) = \tanh(x) = \frac{e^x - e^{-x}}{e^x + e^{-x}}$$

Layers dimensions: $\mathbf{d} = [d, 3, 2, 1]$

$$\mathbf{d} = \left[d^{(0)}, d^{(1)}, \cdots, d^{(L)}\right]$$

Redes Neuronales



$$x^{(0)} = \left(x^{(0)}_{d^{(0)}}, \dots, x^{(0)}_1, 1 \right) \in \mathbb{R}^{d^{(0)}+1}$$

$$s^{(\ell)}_j = w_{ij}\, x^{(\ell-1)}_i \qquad x^{(\ell)}_k = \theta_\ell \left(s^{(\ell)}_j \right)$$

$$s^{(\ell)} = W^{T(\ell)} x^{(\ell-1)} \qquad x^{(\ell)} = \left(\theta_\ell \left(s^{(\ell)} \right), 1 \right)^T$$

$$W = \left(W^{(1)}, W^{(2)} \right)$$

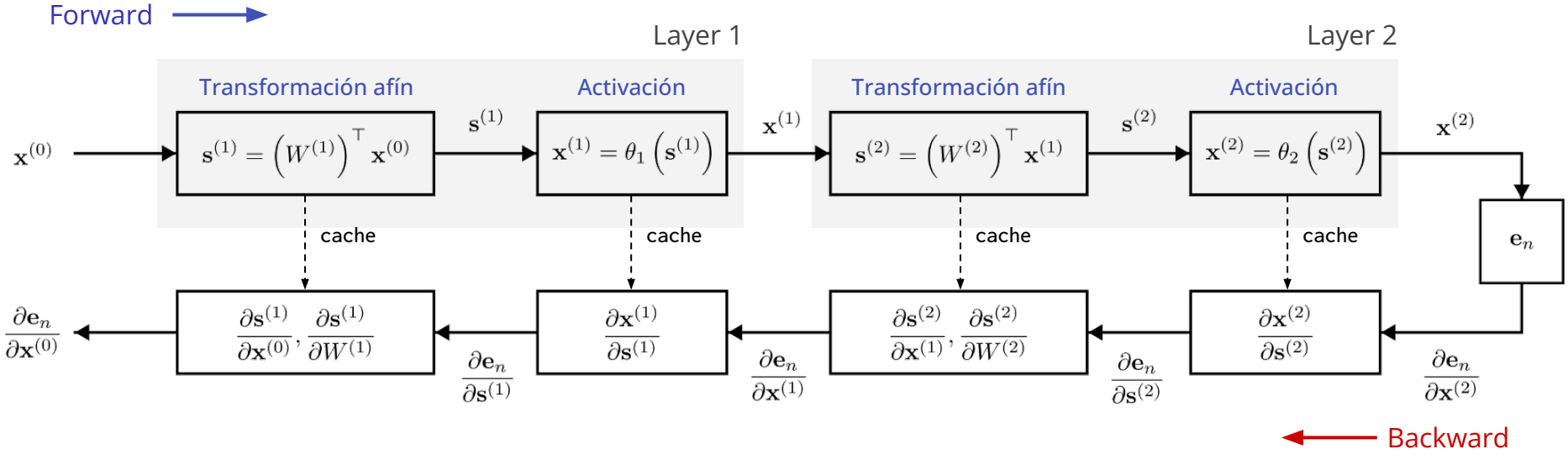
$$W^{(1)} \in \mathbb{R}^{(d^{(0)}+1) \times d^{(1)}}$$

$$W^{(2)} \in \mathbb{R}^{(d^{(1)}+1) \times 1}$$

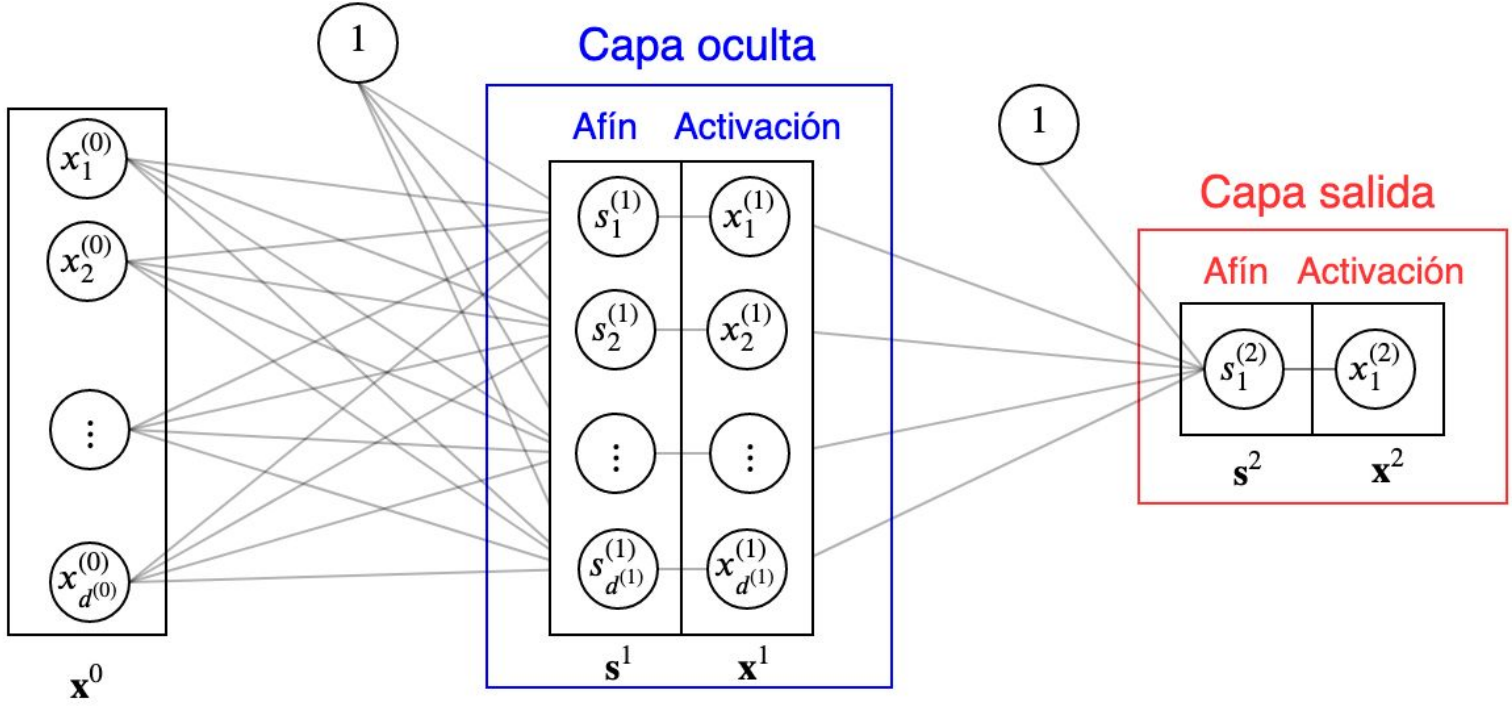
$$x = x^{(0)} \xrightarrow{W^{(1)}} s^{(1)} \xrightarrow{\theta_1} x^{(1)} \xrightarrow{W^{(2)}} s^{(2)} \xrightarrow{\theta_2} x^{(2)} = h(x)$$

$$f(x^{(0)}, W) \equiv h(x) = \theta_2 \left(W^{T(2)} x^{(1)} \right) = \theta_2 \left(W^{T(2)} \theta_1 \left(W^{T(1)} x^{(0)} \right) \right)$$

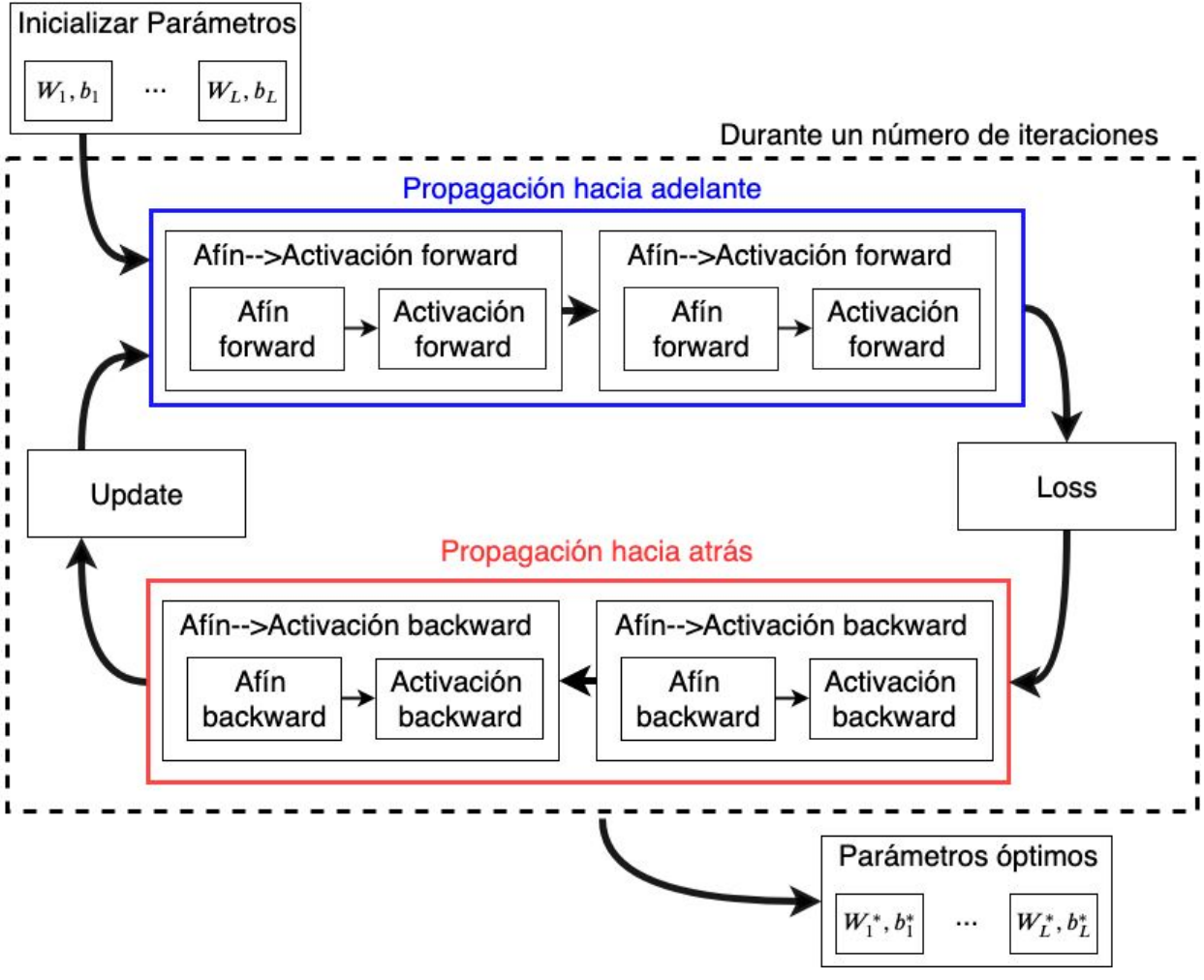
Ejercicio 1: implementación de una red neuronal de dos capas



Ejercicio 1: implementación de una red neuronal de dos capas



Ejercicio 1: implementación de una red neuronal de dos capas



Ejercicio 1: implementación de una red neuronal de dos capas

```
    # Se inicializan los parámetros del diccionario invocando a una de las
    # funciones previamente implementadas.
    parametros = inicializar_pesos(d_0, d_1, d_2, semilla=semilla)

    # Se obtienen W1, b1, W2 y b2 del diccionario de parámetros.
    W1 = parametros["W1"]
    b1 = parametros["b1"]
    W2 = parametros["W2"]
    b2 = parametros["b2"]

    # Loop (descenso por gradiente)
    for i in range(0, num_iter):

        ########################################################
        ####### EMPIEZA ESPACIO PARA COMPLETAR CODIGO #######
        ########################################################

        # Propagación hacia adelante: Afin --> Tanh --> Afin --> Sigmoide.
        # Entradas: "X, W1, b1". Salidas: "X1, cache1, X2, cache2".
        # X1, cache1 =
        # X2, cache2 =

        # Se calcula el costo y se inicia la propagación hacia atrás
        # costo, dX2 =

        # Propagación hacia atrás.
        # Entradas: "dX2, cache2, cache1". Salidas: "dX1, dW2, db2, dW1, db1,
        # dX0 (no utilizado)".
        # dX1, dW2, db2 =
        # dX0, dW1, db1 =

        # Se almacenan los gradientes recientemente calculados en el diccionario
        # gradientes["W1"] =
        # gradientes["b1"] =
        # gradientes["W2"] =
        # gradientes["b2"] =

        # Se actualizan los parámetros
        # parametros =

        # Se obtienen los nuevos W1, b1, W2 y b2 del diccionario de parámetros.
        # W1 =
        # b1 =
        # W2 =
        # b2 =

        ########################################################
        ####### TERMINA ESPACIO PARA COMPLETAR CODIGO #######
        ########################################################
```

Python: diccionarios y eval()

Diccionarios

A diferencia de las secuencias, que se indexan mediante un rango numérico, los diccionarios se indexan con claves, que pueden ser cualquier tipo inmutable; las cadenas y números siempre pueden ser claves.

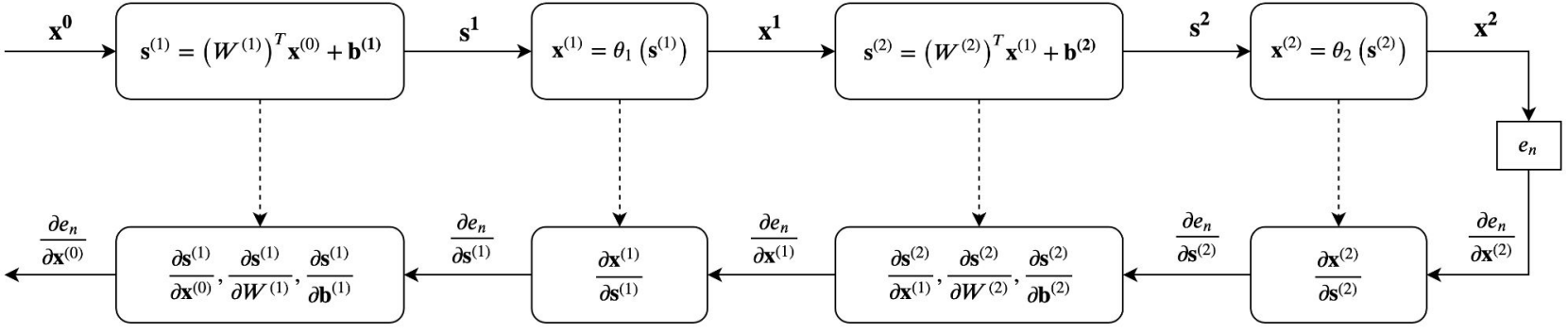
```
>>> tel = {'ana': 123, 'bruno': 456}
>>> tel['cecilia'] = 789
>>> tel
{'ana': 123, 'bruno': 456, 'cecilia': 789}
>>> tel['ana']
123
>>> tel.get('ana')
123
>>> del tel['bruno']
>>> tel['diego'] = 789
>>> tel
{'ana': 123, 'cecilia': 789, 'diego': 789}
>>> 'ana' not in tel
False
>>> tel.keys()
dict_keys(['ana', 'cecilia', 'diego'])
```

eval(expression, globals=None, locals=None)

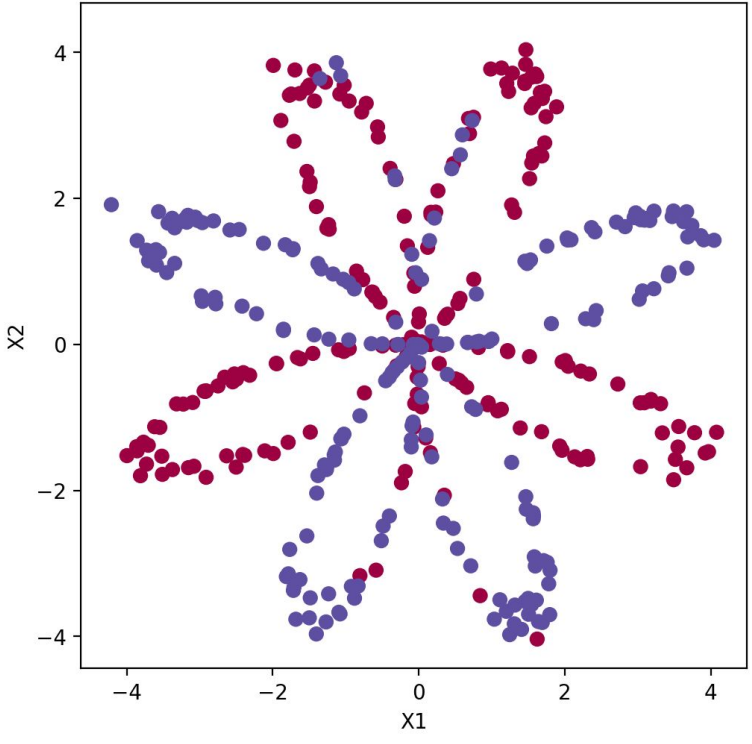
El argumento *expression* se analiza y evalúa como una expresión de Python (técnicamente hablando, una lista de condiciones), usando los diccionarios *globals* y *locals* como espacios de nombres globales y locales.

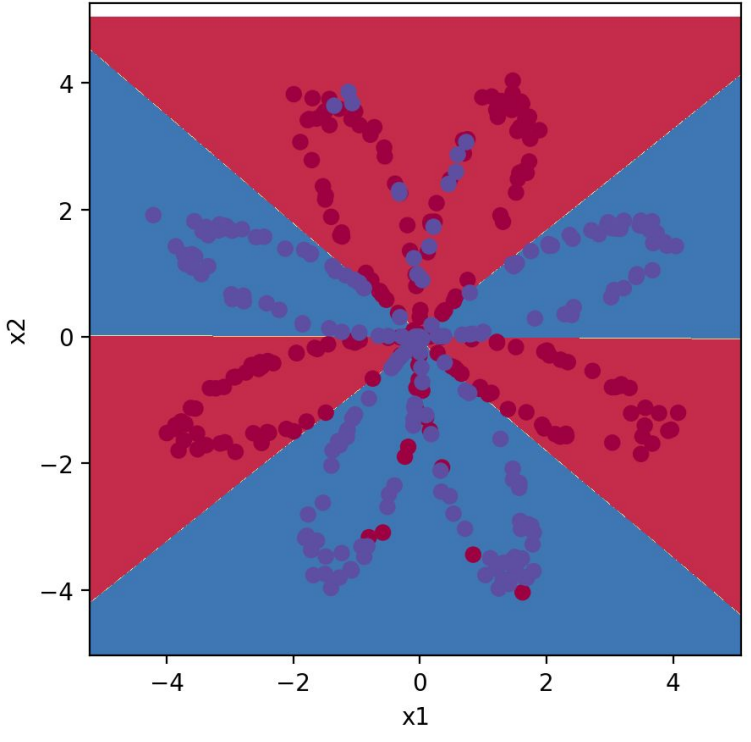
```
>>> x = 'print(42)'
>>> eval(x)
42
>>> tel = {'ana': 123, 'cecilia': 789, 'diego': 789}
>>> tel.keys()
dict_keys(['ana', 'cecilia', 'diego'])
>>> for k in tel.keys():
...:     print("%s: %s" % (k, eval('tel[k]')))
ana: 123
cecilia: 789
diego: 789
```

Ejercicio 1: implementación de una red neuronal de dos capas



Ejercicio 1: implementación de una red neuronal de dos capas





Ejercicio 1: implementación de una red neuronal de dos capas

Imagen con gato

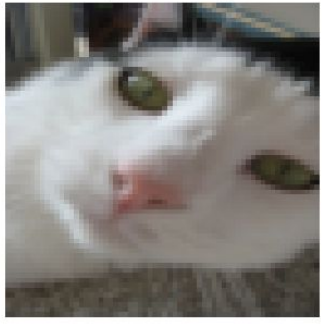


Imagen con gato

Imagen con gato

Imagen con gato

Imagen con gato

Imagen sin gato

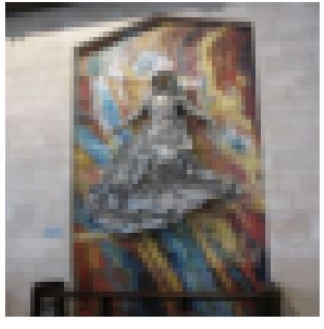
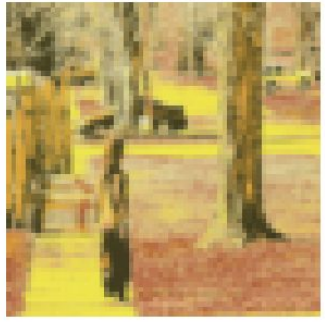


Imagen sin gato

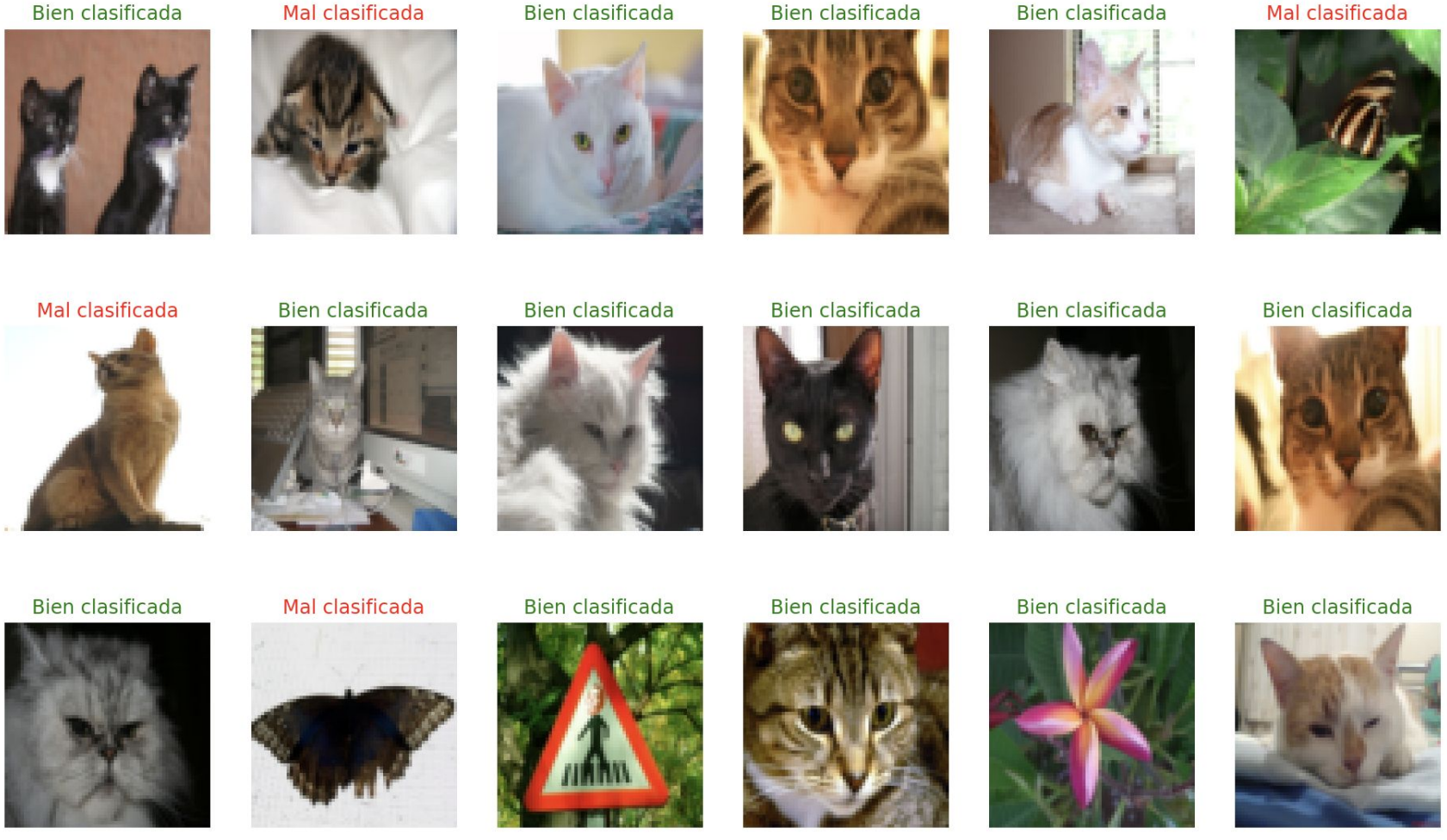
Imagen sin gato

Imagen sin gato

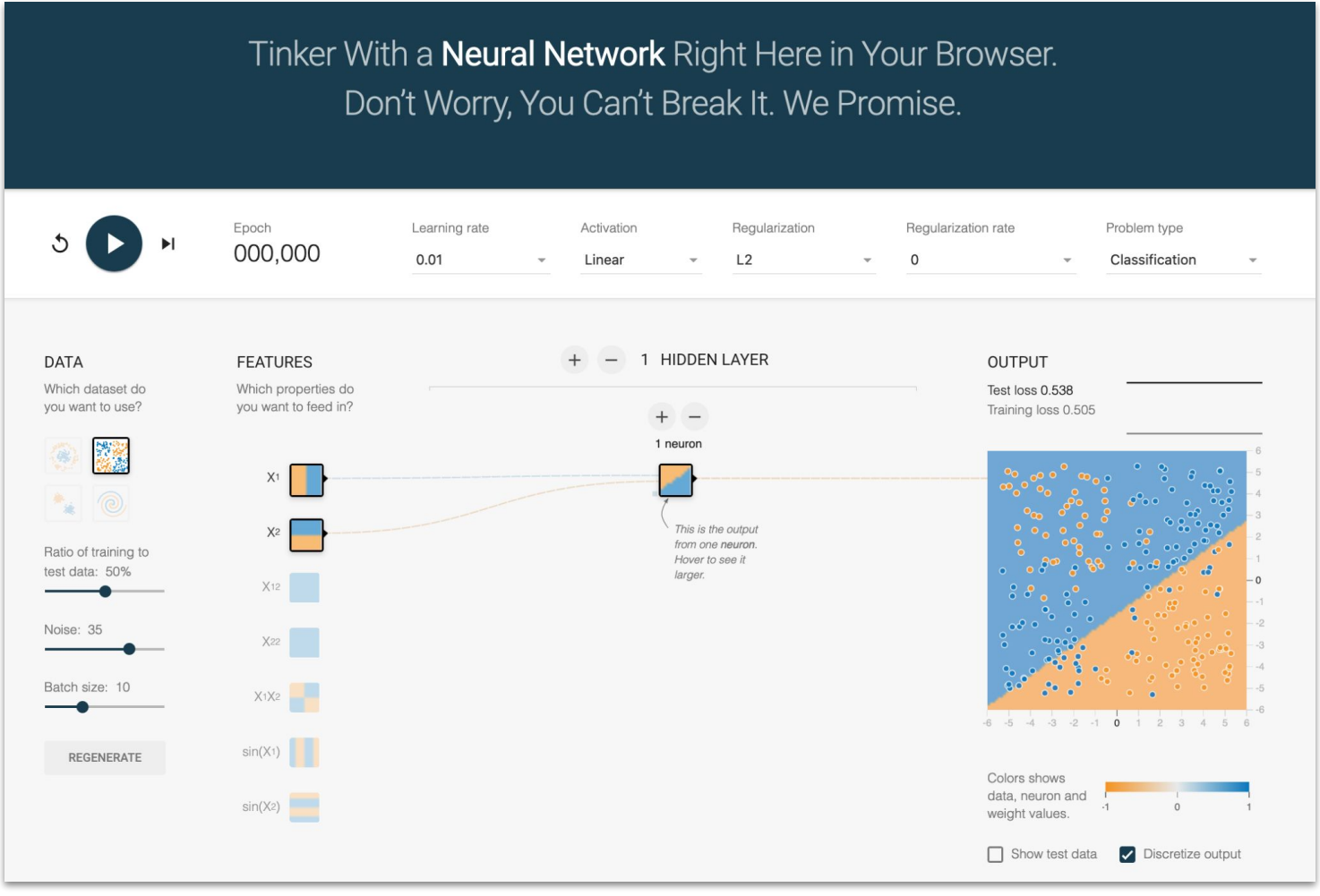
Imagen sin gato



Ejercicio 1: implementación de una red neuronal de dos capas



Ejercicio 2: Jugando con Tensorflow playground



Cuestionario práctico

passwd: 9872