

Fundamentos de Aprendizaje Automático y Reconocimiento de Patrones

Actividades en clase: Práctico 5

Graciana Castro, Martín Schmidt, Federico Lecumberry, Guillermo Carbajal

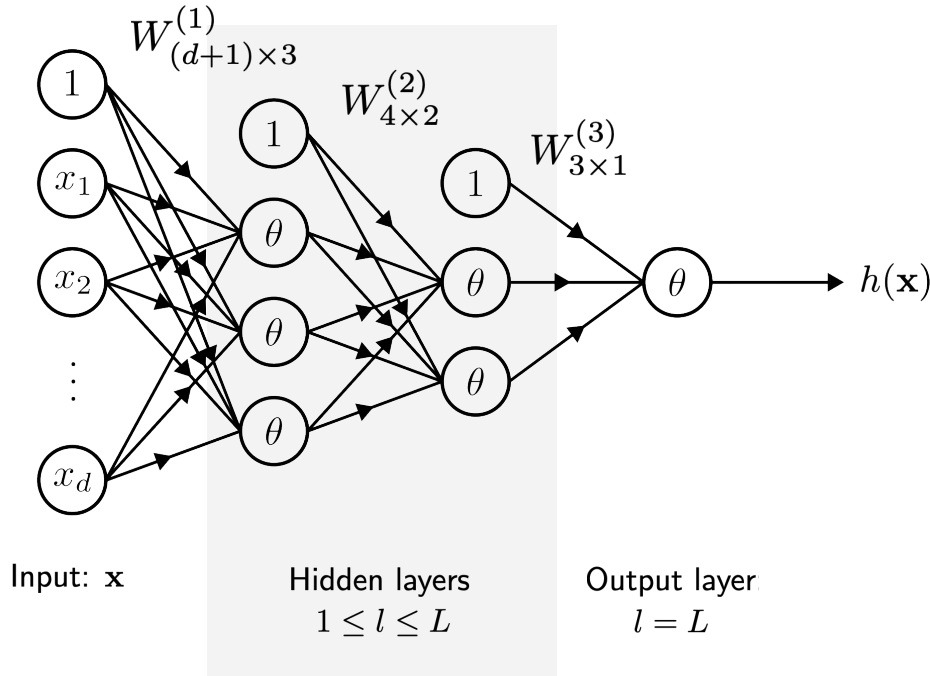
Instituto de Ingeniería Eléctrica

2024

Tabla de contenido

- Ejercicio 1: implementación de una red neuronal de dos capas
- Ejercicio 2: Jugando con Tensorflow playground

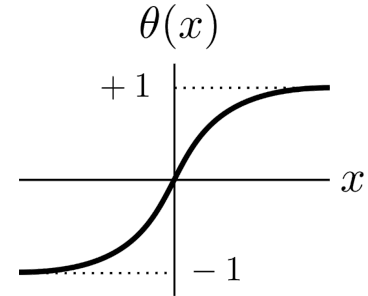
Redes Neuronales



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

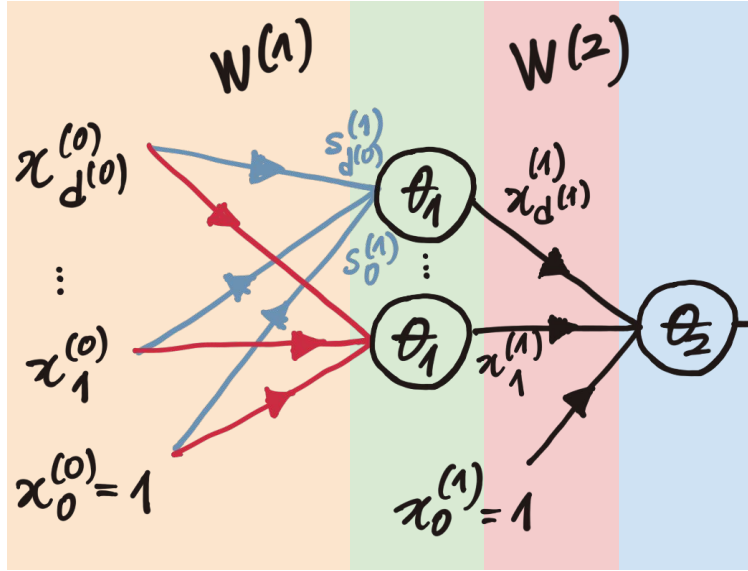
$$\mathbf{d} = [d^{(0)}, d^{(1)}, \dots, d^{(L)}]$$

Activation Function
(differentiable!!!)



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

Redes Neuronales



$$x^{(0)} = (x_{d^{(0)}}^{(0)}, \dots, x_1^{(0)}, 1) \in \mathbb{R}^{d^{(0)}+1}$$

$$s_j^{(l)} = w_{lj} x_j^{(l-1)}$$

$$s^{(l)} = W^{T(l)} x^{(l-1)}$$

$$x_k^{(l)} = \theta_l(s_j^{(l)})$$

$$x^{(l)} = (\theta_l(s^{(l)}), 1)^T$$

$$W = (W^{(1)}, W^{(2)})$$

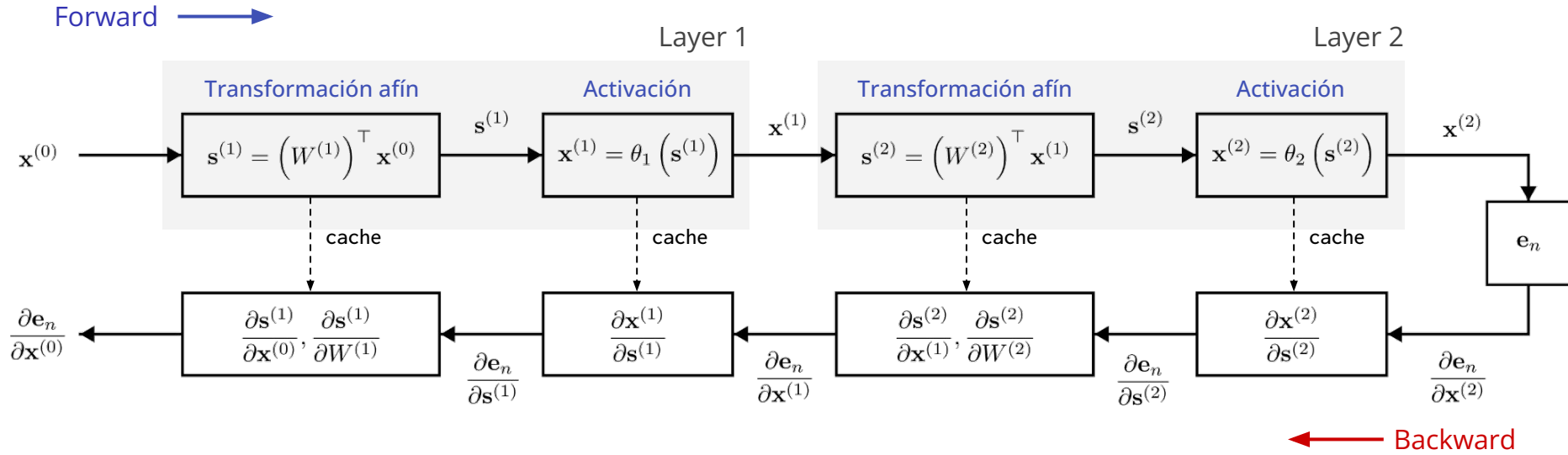
$$W^{(1)} \in \mathbb{R}^{(d^{(0)}+1) \times d^{(1)}}$$

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

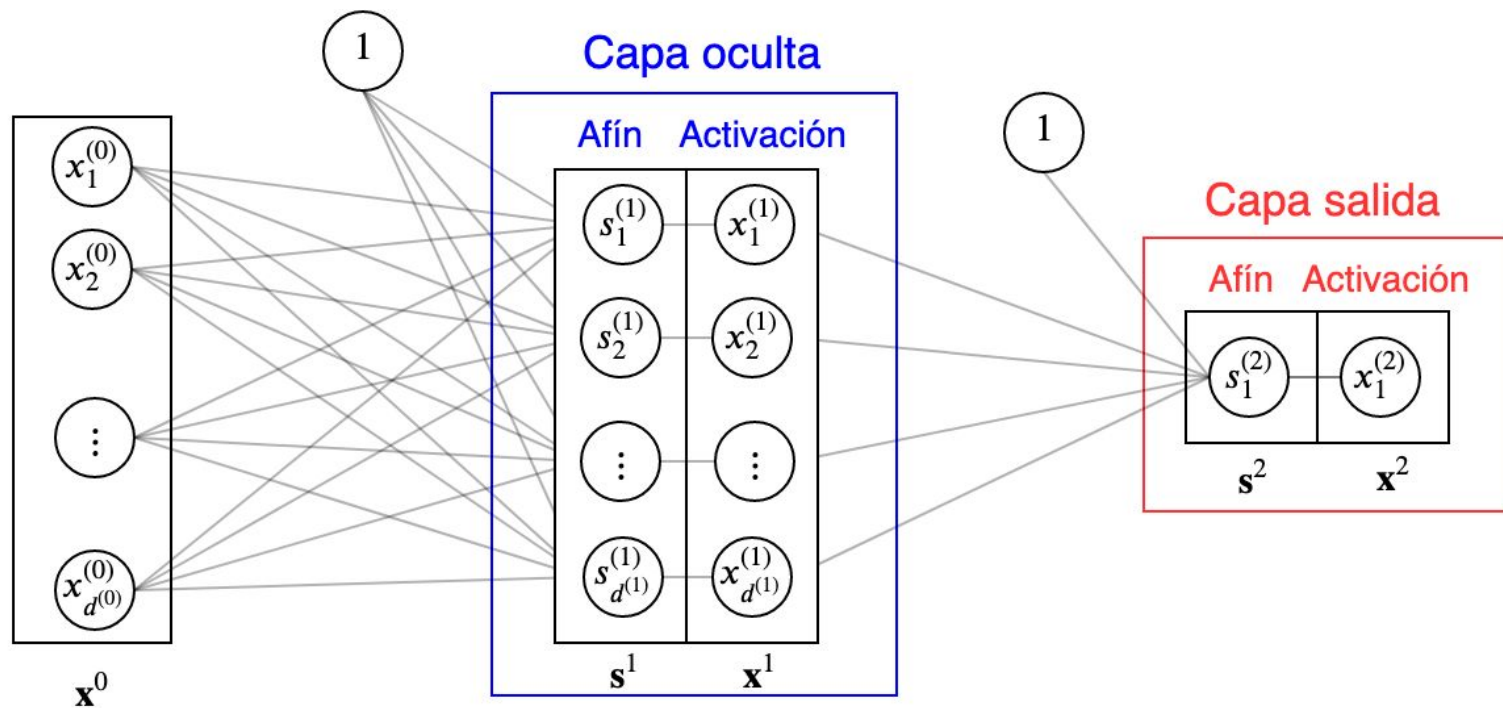
$$x = x^{(0)} \xrightarrow{W^{(1)}} s^{(1)} \xrightarrow{\theta_1} x^{(1)} \xrightarrow{W^{(2)}} s^{(2)} \xrightarrow{\theta_2} x^{(2)} = h(x)$$

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

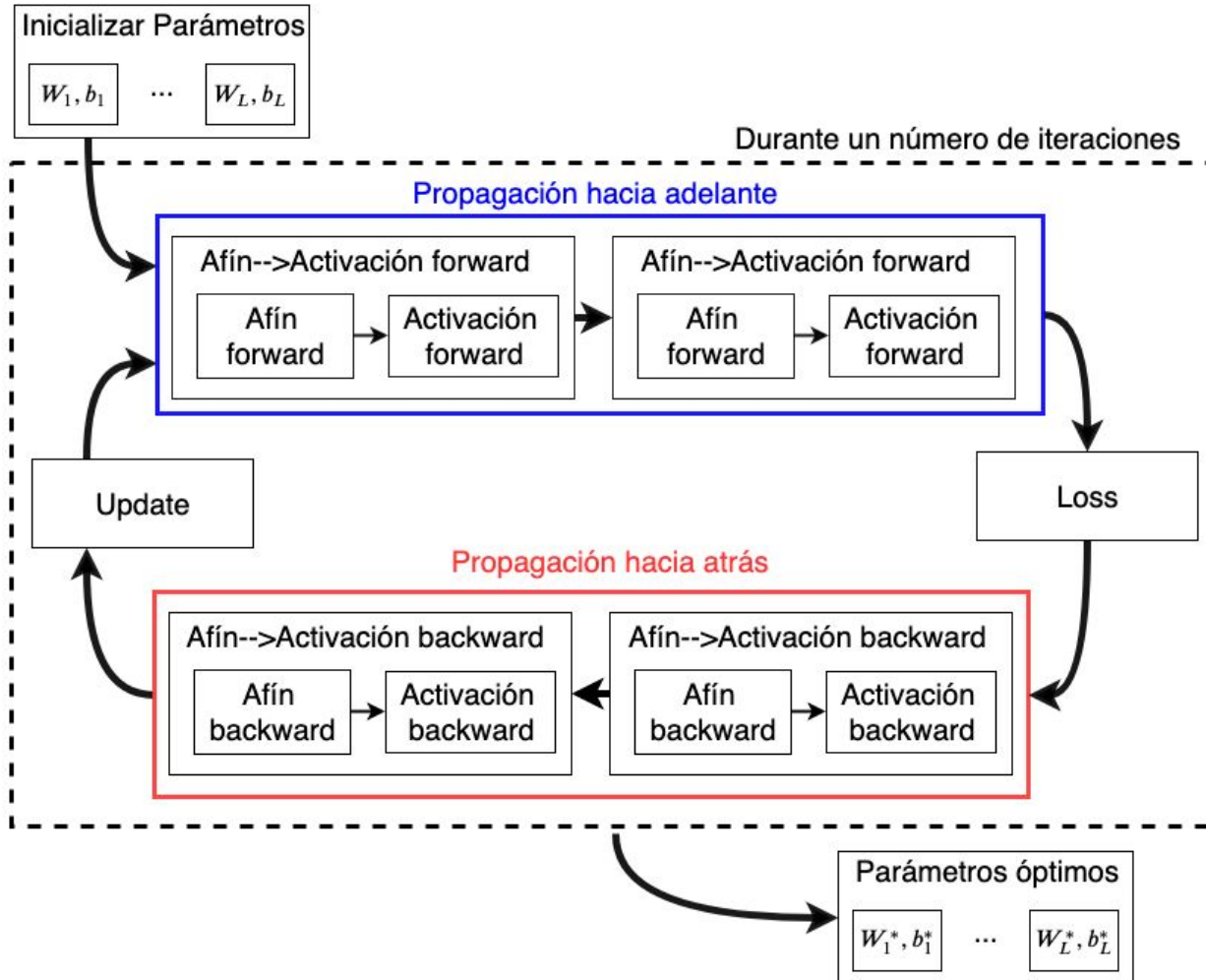
Ejercicio 1: implementación de una red neuronal de dos capas



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

```
34 # Se inicializan los parámetros del diccionario invocando a una de las
35 # funciones previamente implementadas.
36 parametros = inicializar_pesos(d_0, d_1, d_2, semilla=semilla)
37
38 # Se obtienen W1, b1, W2 y b2 del diccionario de parámetros.
39 W1 = parametros["W1"]
40 b1 = parametros["b1"]
41 W2 = parametros["W2"]
42 b2 = parametros["b2"]
43
44 # Loop (descenso por gradiente)
45 for i in range(0, num_iter):
46
47     #####
48     ##### EMPIEZA ESPACIO PARA COMPLETAR CODIGO #####
49     #####
50
51     # Propagación hacia adelante: Afin --> Tanh --> Afin --> Sigmoide.
52     # Entradas: "X, W1, b1". Salidas: "X1, cache1, X2, cache2".
53     # X1, cache1 =
54     # X2, cache2 =
55
56     # Se calcula el costo y se inicia la propagación hacia atrás
57     # costo, dX2 =
58
59     # Propagación hacia atrás.
60     # Entradas: "dX2, cache2, cache1". Salidas: "dX1, dW2, db2, dW1, db1,
61     # dX0 (no utilizado)".
62     # dX1, dW2, db2 =
63     # dX0, dW1, db1 =
64
65     # Se almacenan los gradientes recientemente calculados en el diccionario
66     # gradientes["W1"] =
67     # gradientes["b1"] =
68     # gradientes["W2"] =
69     # gradientes["b2"] =
70
71     # Se actualizan los parámetros
72     # parametros =
73
74     # Se obtienen los nuevos W1, b1, W2 y b2 del diccionario de parámetros.
75     # W1 =
76     # b1 =
77     # W2 =
78     # b2 =
79
80     #####
81     ##### TERMINA ESPACIO PARA COMPLETAR CODIGO #####
82     #####
```


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.

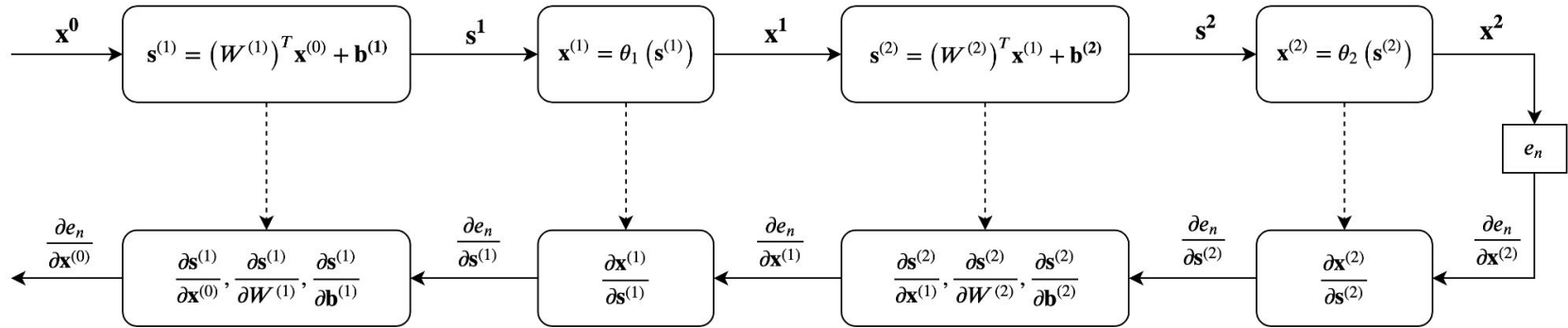
```
>>> tel = {'ana': 123, 'bruno': 456}
>>> tel['cecilia'] = 789
>>> tel
{'ana': 123, 'bruno': 456, 'cecilia': 789}
>>> tel['ana']
123
>>> tel.get('ana')
123
>>> del tel['bruno']
>>> tel['diego'] = 789
>>> tel
{'ana': 123, 'cecilia': 789, 'diego': 789}
>>> 'ana' not in tel
False
>>> tel.keys()
dict_keys(['ana',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

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

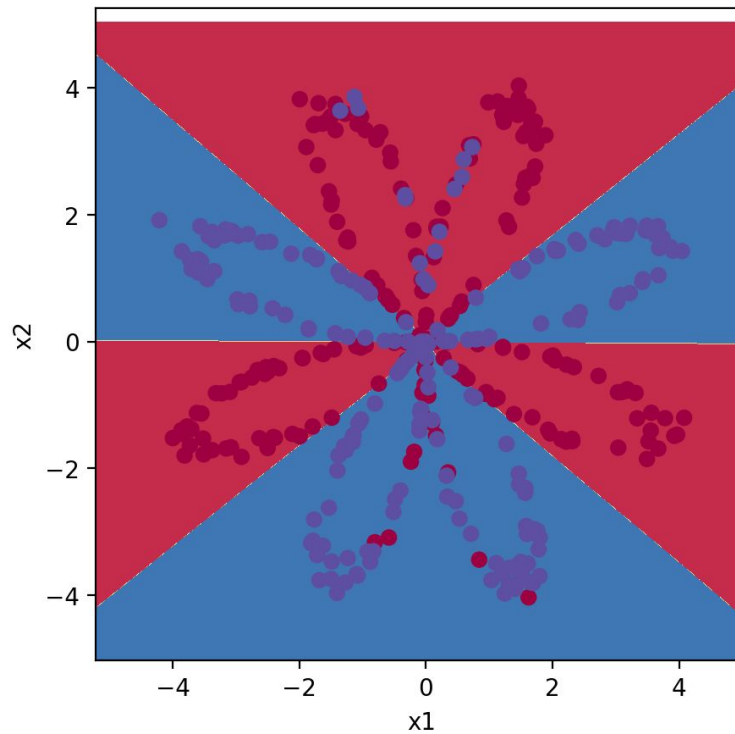
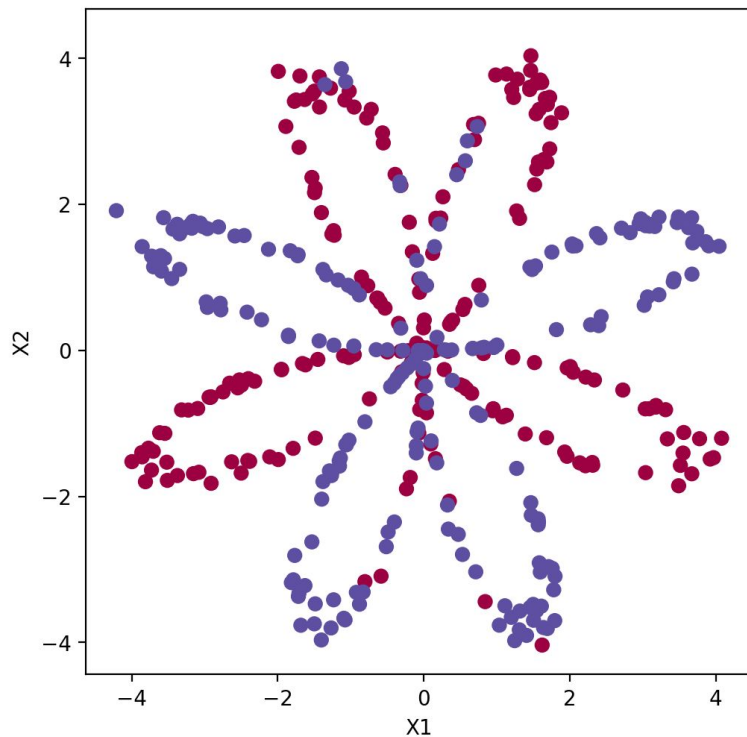
El argumento *expression* se analiza y evalúa como una expresión de Python (técnicamente hablando, una lista de condiciones), usando los diccionarios *globals* y *locals* como espacios de nombres globales y locales.

```
>>> 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[%s]')))
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

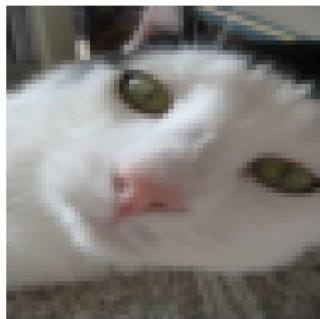


Imagen con gato

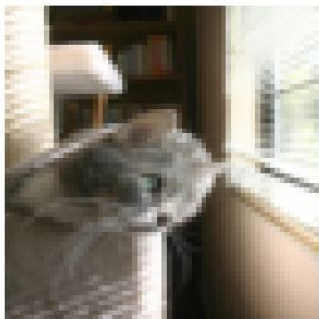


Imagen con gato

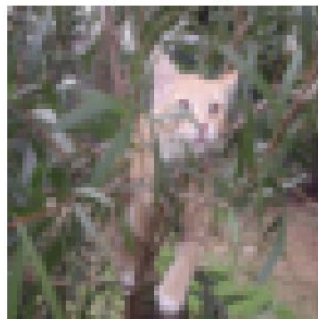


Imagen con gato



Imagen con gato

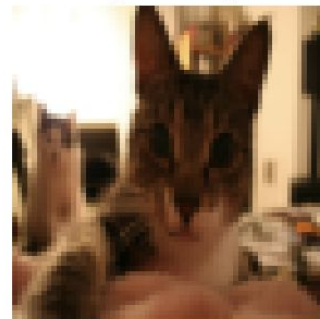


Imagen sin gato

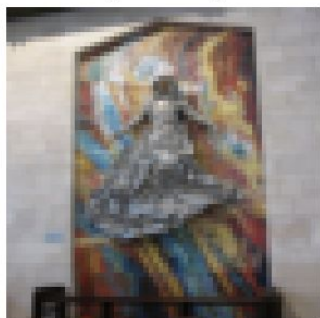


Imagen sin gato



Imagen sin gato



Imagen sin gato



Imagen sin gato



Ejercicio 1: implementación de una red neuronal de dos capas

Bien clasificada



Mal clasificada



Bien clasificada



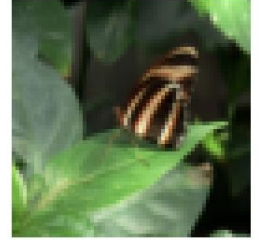
Bien clasificada



Bien clasificada



Mal clasificada



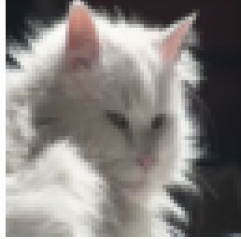
Mal clasificada



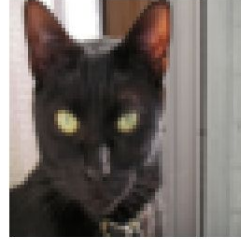
Bien clasificada



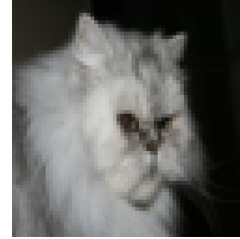
Bien clasificada



Bien clasificada



Bien clasificada



Bien clasificada



Bien clasificada



Mal clasificada



Bien clasificada



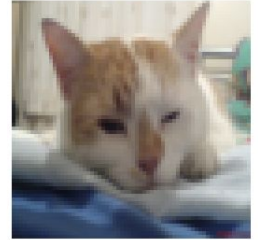
Bien clasificada



Bien clasificada



Bien clasificada



Ejercicio 2: Jugando con Tensorflow playground

Tinker With a **Neural Network** Right Here in Your Browser.
Don't Worry, You Can't Break It. We Promise.

Epoch: 000,000 | Learning rate: 0.01 | Activation: Linear | Regularization: L2 | Regularization rate: 0 | Problem type: Classification

DATA
Which dataset do you want to use?
Ratio of training to test data: 50%
Noise: 35
Batch size: 10
REGENERATE

FEATURES
Which properties do you want to feed in?
X1, X2, X1X2, sin(X1), sin(X2)

HIDDEN LAYER
1 HIDDEN LAYER
1 neuron
This is the output from one neuron. Hover to see it larger.

OUTPUT
Test loss 0.538
Training loss 0.505

Colors shows data, neuron and weight values.

Show test data Discretize output

Cuestionario práctico

passwd: 9872