Universidad Nacional del Altiplano
Facultad de Ingeniería Mecánica Eléctrica, Electrónica y Sistemas
Escuela Profesional del Ingeniería de Sistemas

SIS 227: Tópicos Avanzados de Inteligencia Computacional

SESIÓN N° 05:

Análisis de sentimiento con Redes Neuronales

Ing. Donia Alizandra Ruelas Acero

Introducción

Anteriormente se realizó análisis de sentimiento con regresión logística y naive Bayes, que no pueden captar el sentimiento de un tweet como:

"I am not happy "

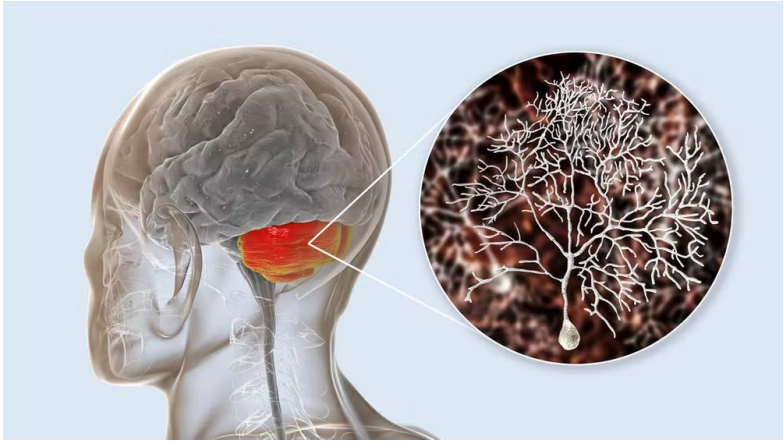
"If only it was a good day".

1.Redes Neuronales para Análisis de Sentimiento

Redes Neuronales
Inspiración biológica

Los pájaros nos inspiraron a volar, la naturaleza ha inspirado muchos otros inventos.

Parece lógico, entonces, mirar la arquitectura del cerebro en busca de inspiración sobre cómo construir una máquina inteligente



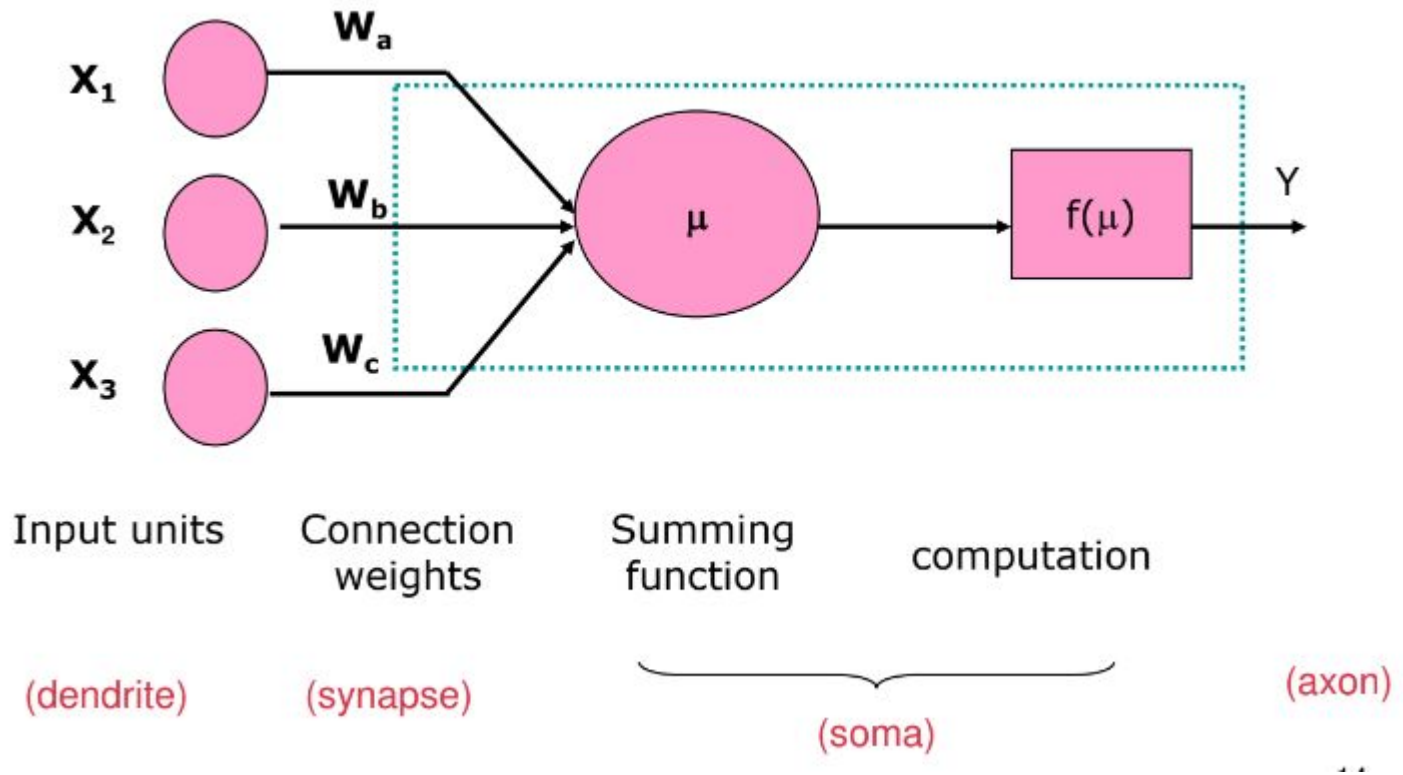
Esta es la idea clave que inspiró las redes neuronales artificiales (ANN). ***Sin embargo, aunque los aviones se inspiran en las aves, no tienen que batir las alas.***

Redes Neuronales
Red Neuronal Artificial

Los humanos somos increíbles en el reconocimiento de patrones. Nuestros cerebros procesan "entradas" del mundo, las clasifican (eso es una araña; eso es un helado), y luego generan una "salida" (huye de la araña; prueba el helado). Y lo hacemos de forma automática y rápida, con poco o ningún esfuerzo.

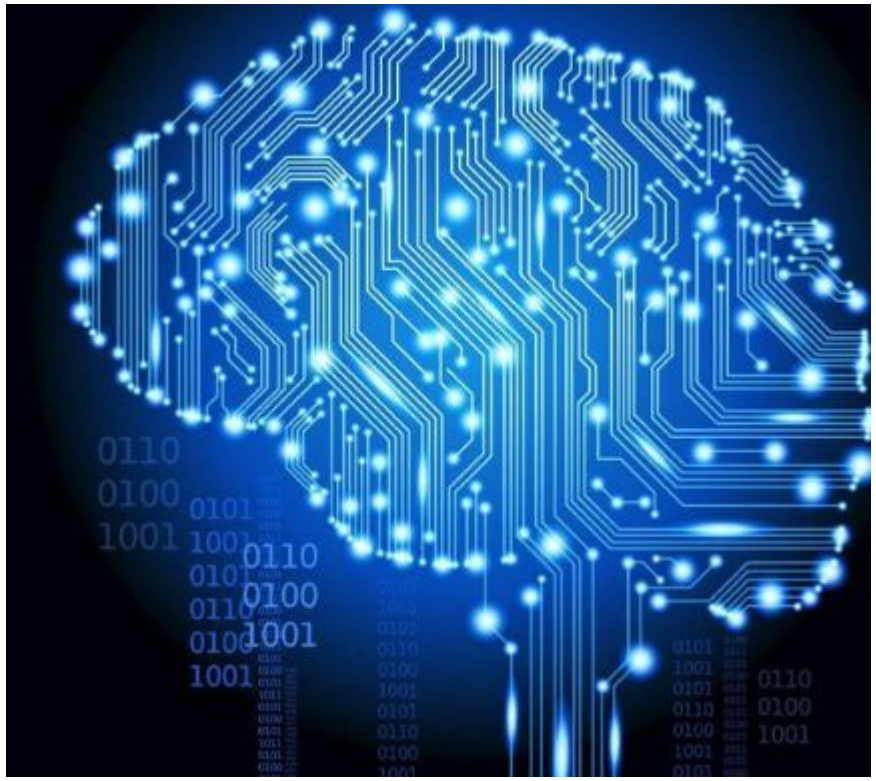
Las redes neuronales imitan en gran medida la forma en que nuestros cerebros resuelven el problema: tomando entradas, procesándolas y generando una salida. Al igual que nosotros, aprenden a reconocer patrones, pero lo hacen entrenándose en conjuntos de datos etiquetados.

Redes Neuronales
Neurona Biológica VS Neurona Artificial



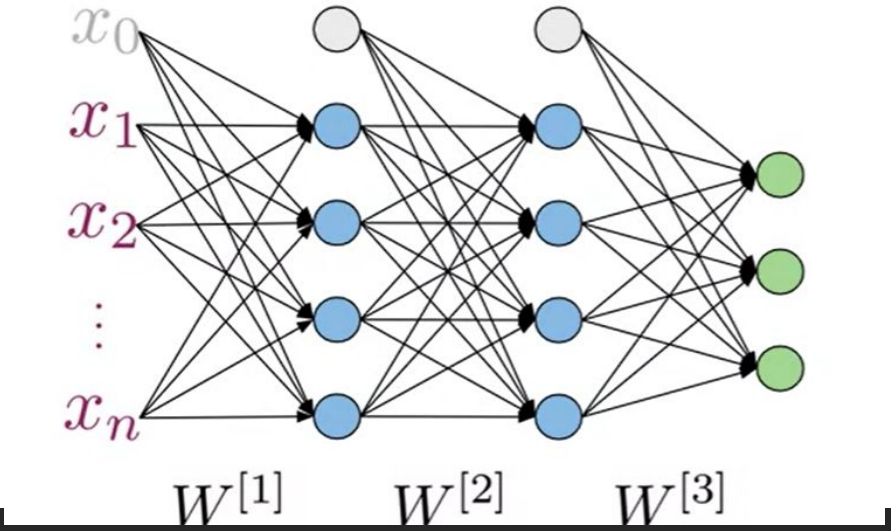
Redes Neuronales
¿Qué son las redes neuronales?

Son estructuras computacionales que de una manera muy simple intentan imitar la forma en que el cerebro humano **reconoce patrones.**



Redes Neuronales

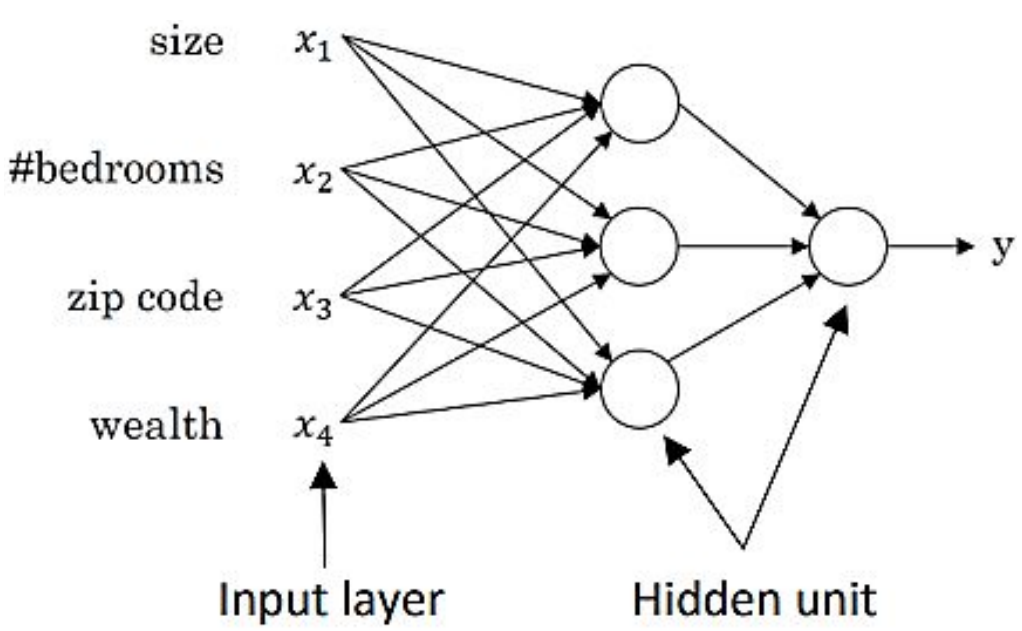
Son estructuras computacionales que de una manera muy simple intentan imitar la forma en que el cerebro humano **reconoce patrones.**



Red Neuronal Profunda

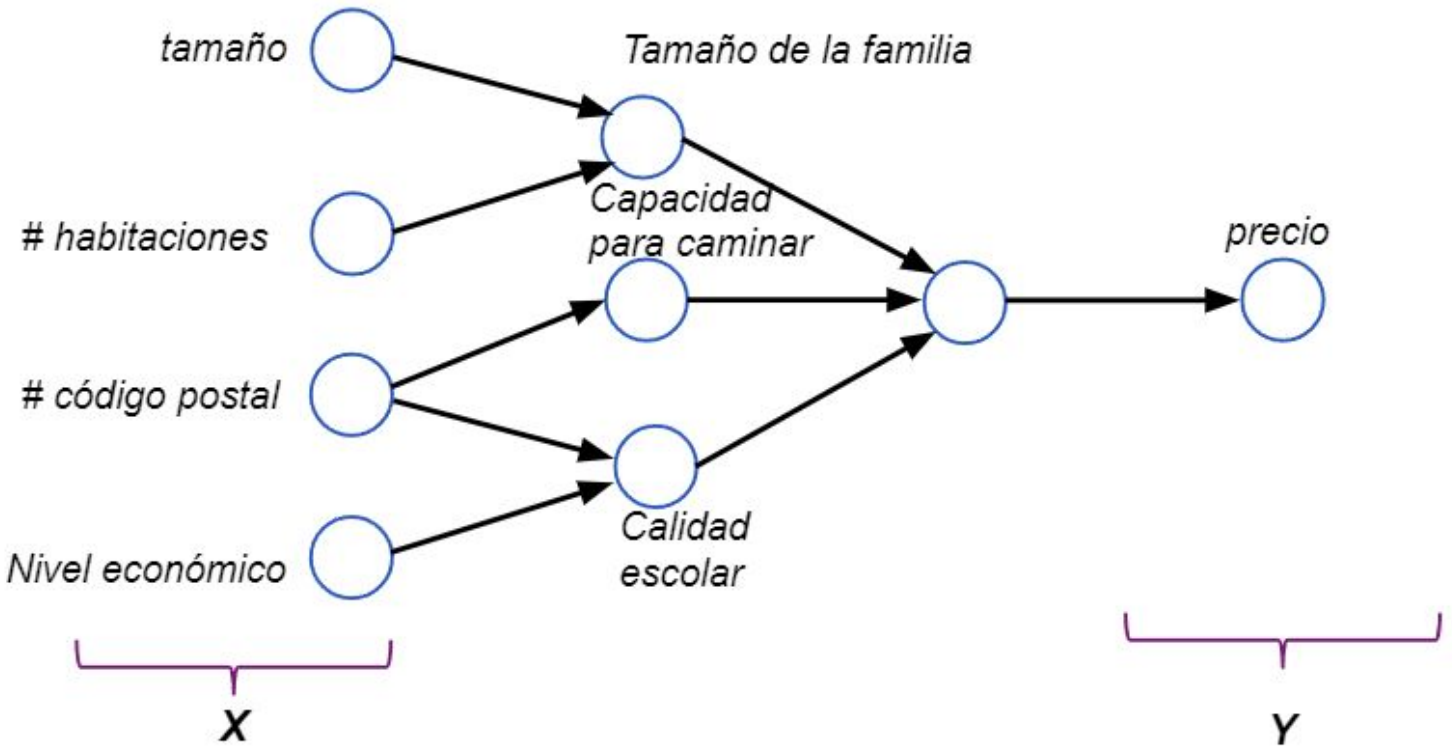
Aparecen las capas ocultas y está compuesta por unidades ocultas en la red neuronal, que cada uno de ellos toma sus entradas de las funciones de entrada.

Es decir, se generan nuevas sub características que son combinación de las características iniciales. Por ejemplo se combinan las características de tamaño de casa y n° de habitaciones que determina la subcaracterística tamaño de familia.

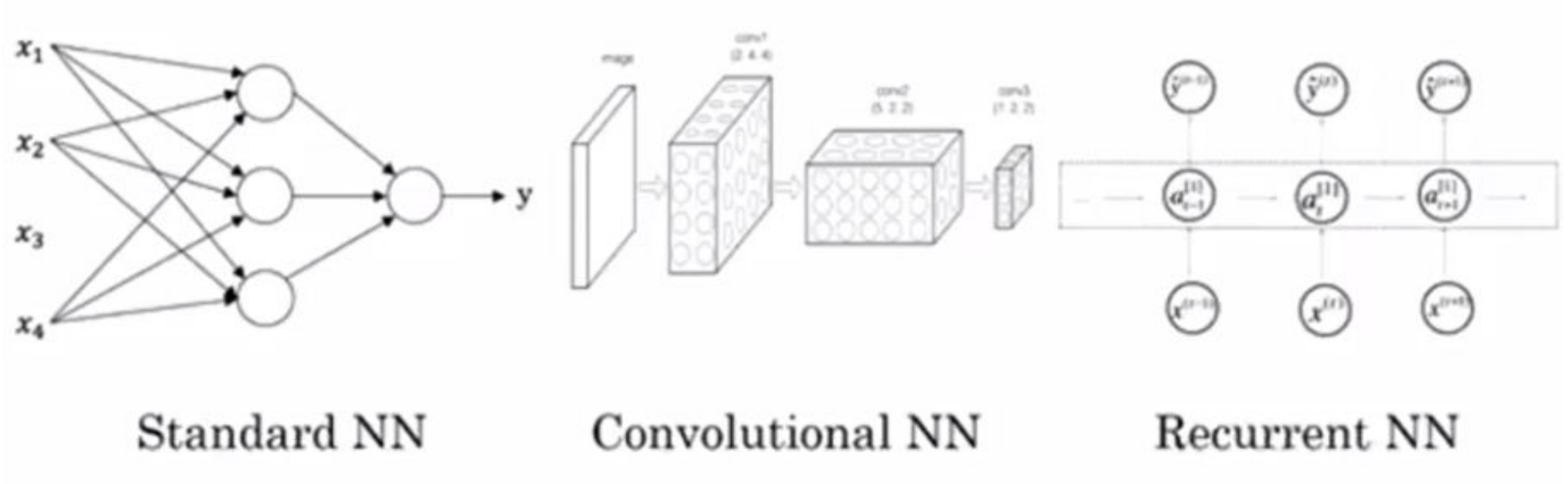


Red Neuronal Profunda

Cada neu



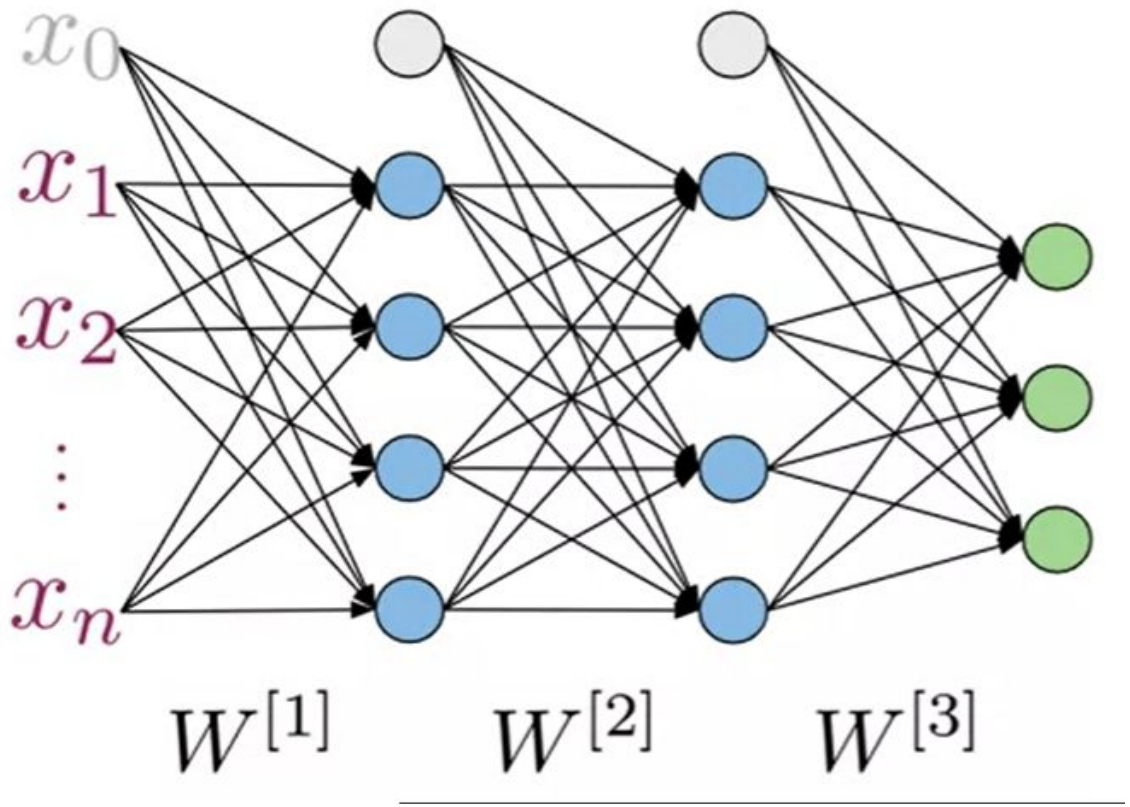
Redes Neuronales Arquitectura



2. Proceso de Aprendizaje

- Forward Propagation
- Backward Propagation

Proceso de Aprendizaje
Forward Propagation



$a^{[i]}$ Activations ith layer

$$a^{[0]} = X$$

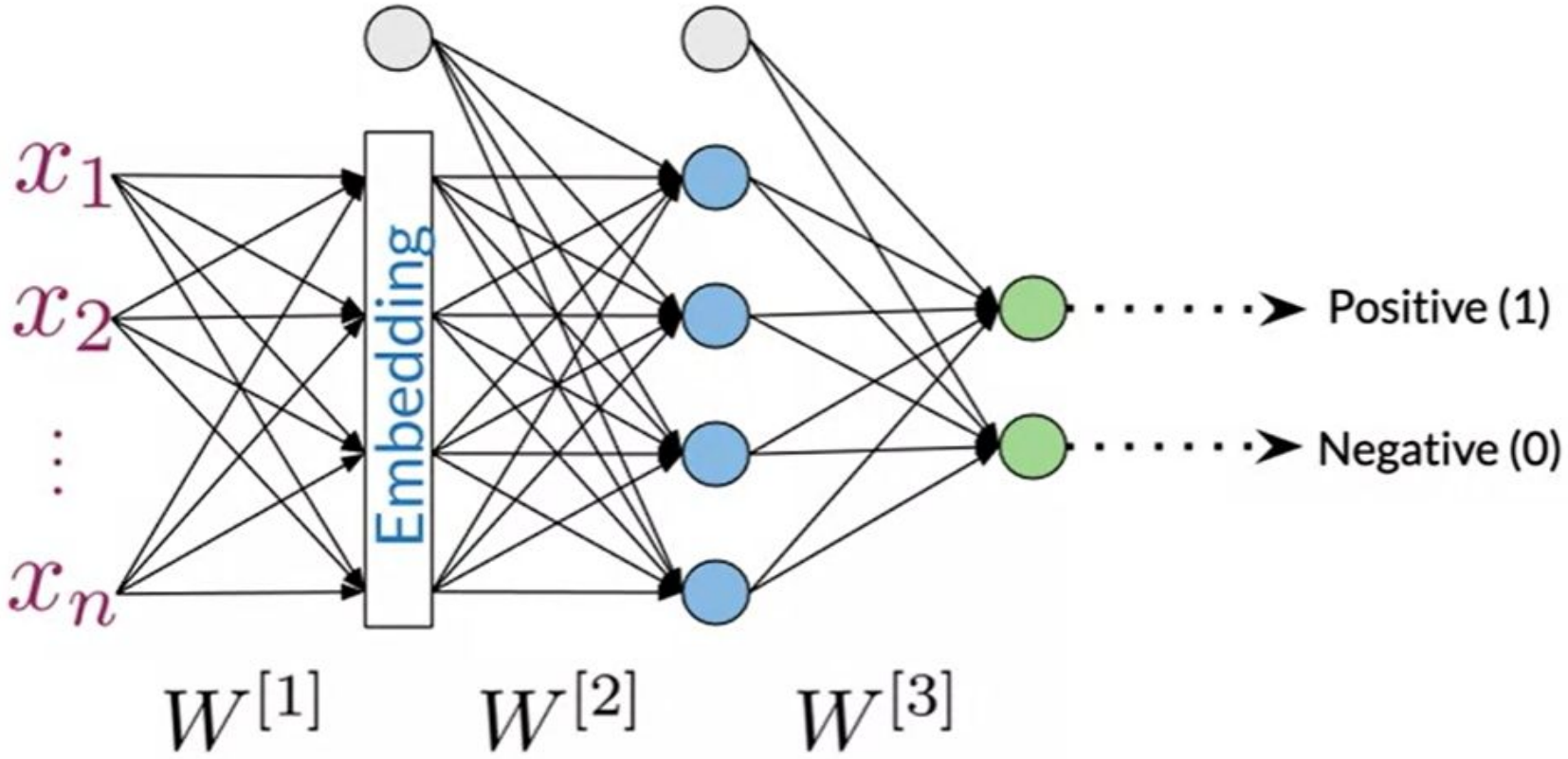
$$z^{[i]} = W^{[i]} a^{[i-1]}$$

$$a^{[i]} = g^{[i]}(z^{[i]})$$

Redes Neuronales para Análisis de Sentimiento

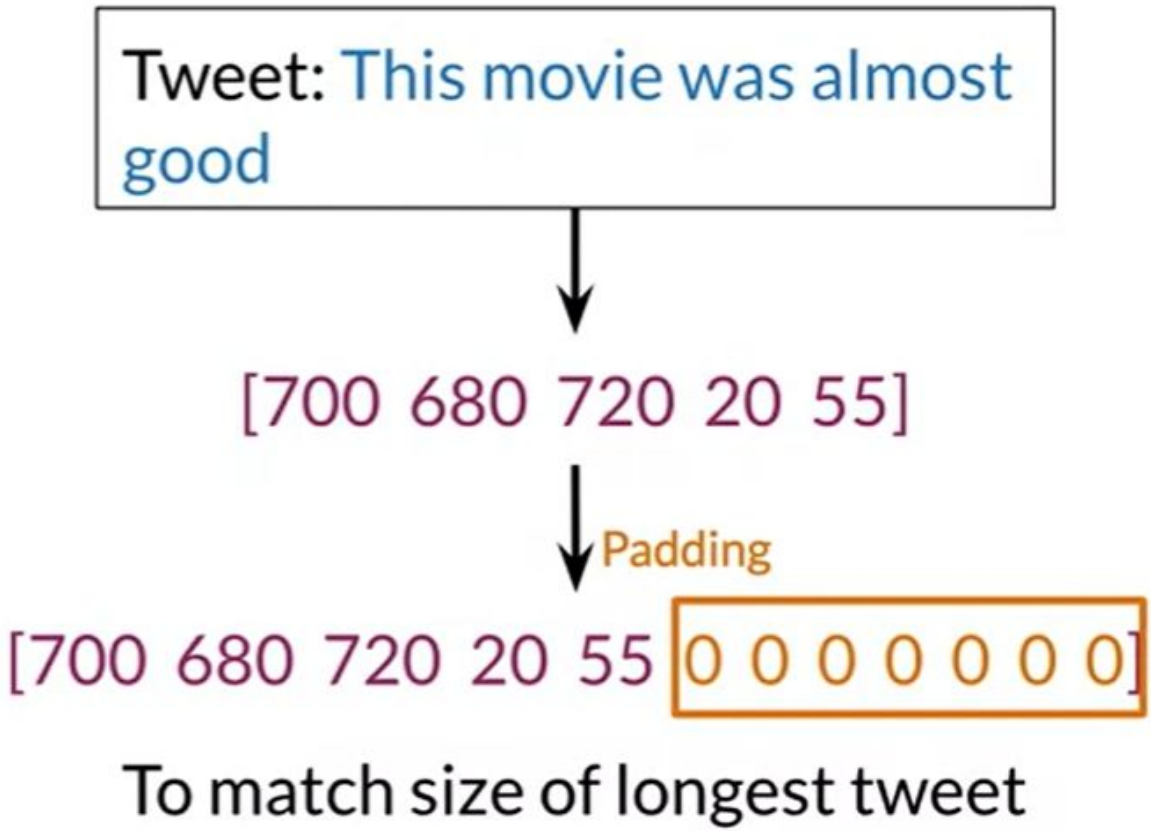
La red neuronal permite predecir sentimientos para tweets complejos.

Tweet: This movie was almost good



Representación inicial

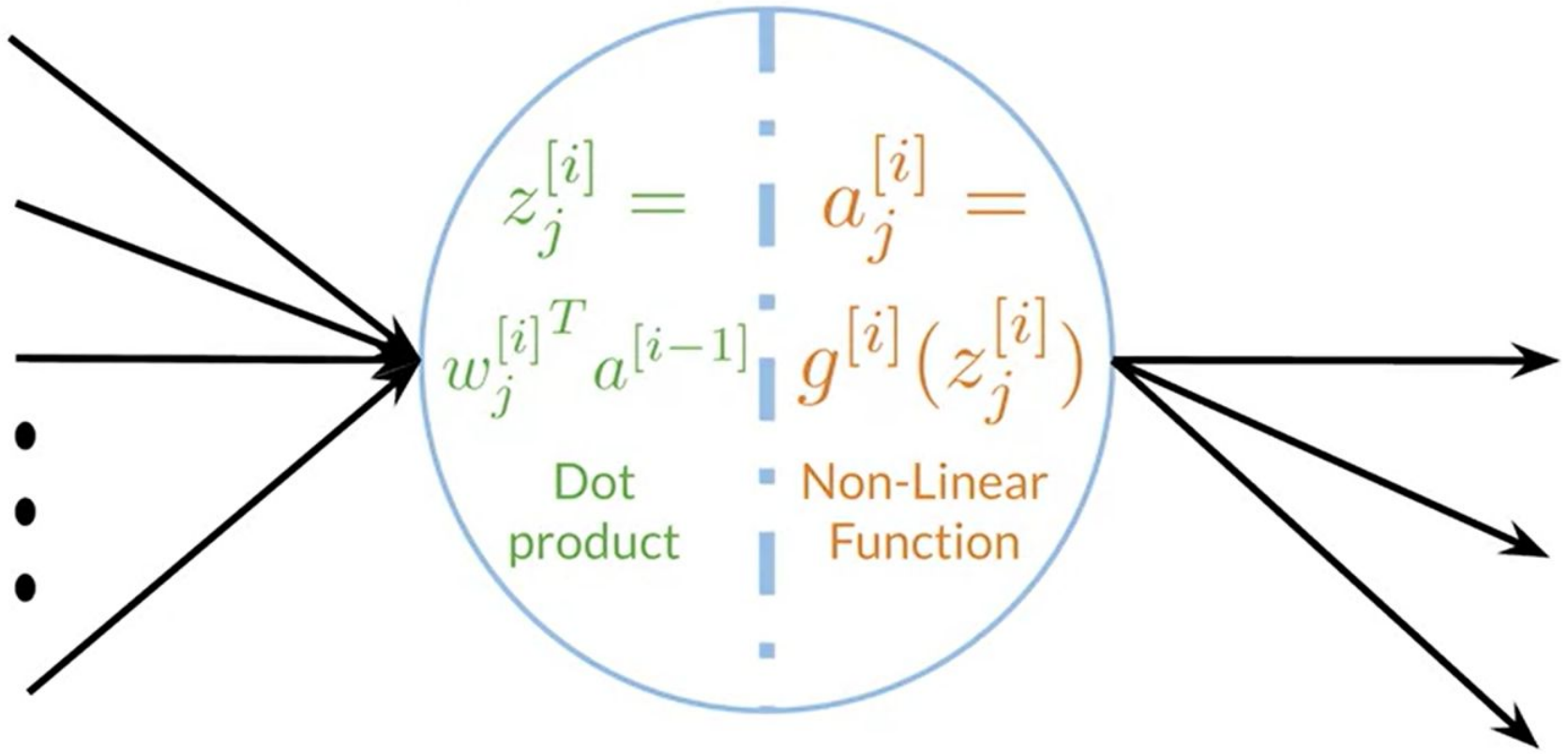
Word	Number
a	1
able	2
about	3
...	...
hand	615
...	...
happy	621
...	...
zebra	1000



3. Red neuronal básica con Trax

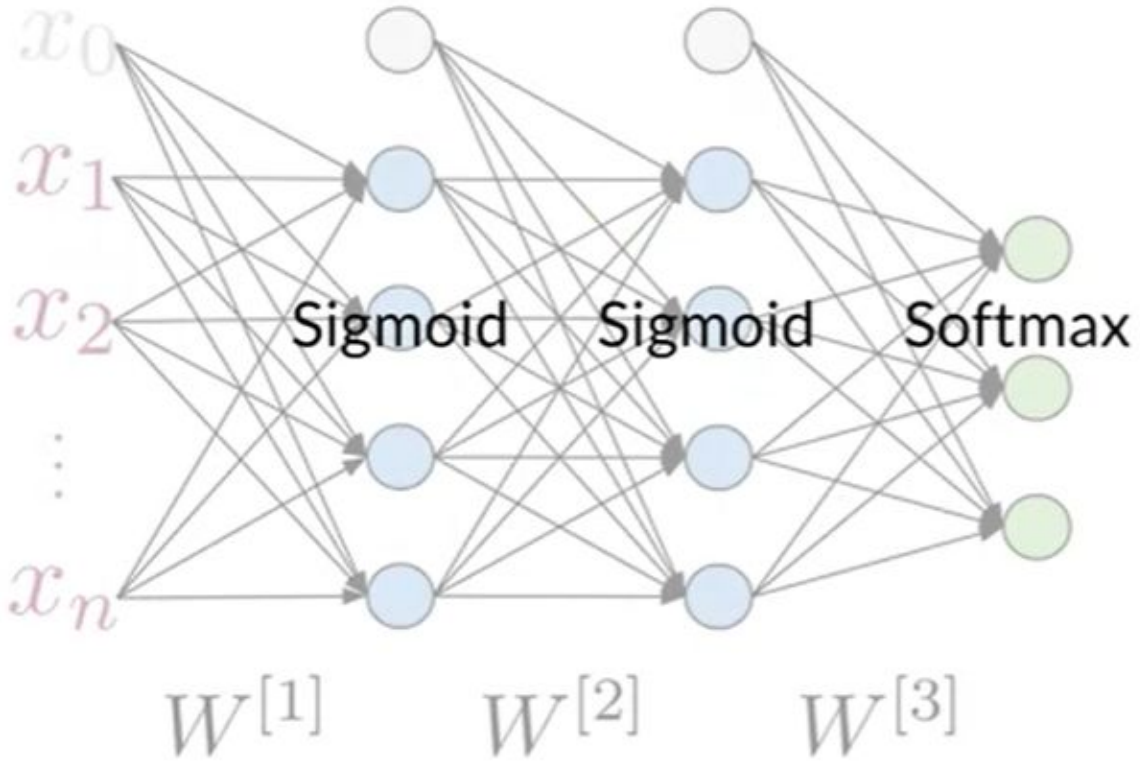
- Define una básica red neuronal básica usando Trax
- Beneficios de Trax

Trax
Neural Network

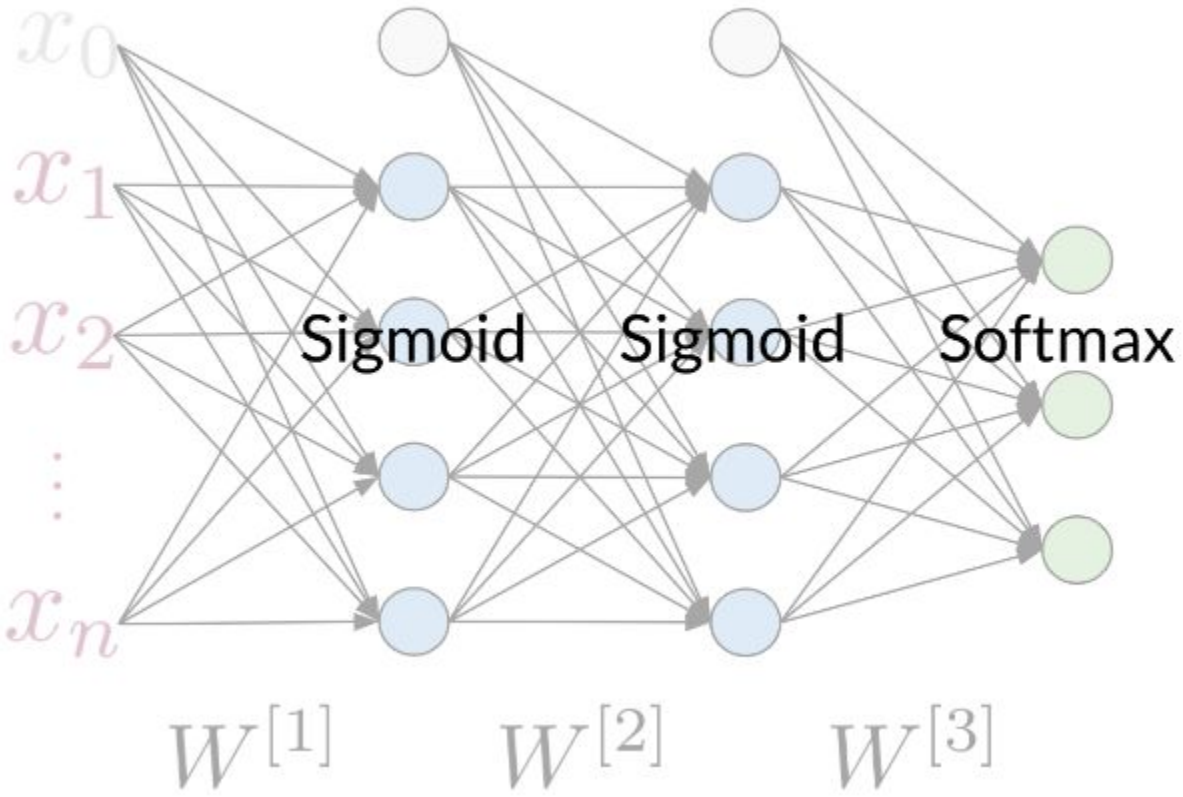


Trax
Neural Network

Es una biblioteca integral para el deep learning.



Trax
En código




```
from trax import layers as tl
Model = tl.Serial(
        tl.Dense(4),
        tl.Sigmoid(),
        tl.Dense(4),
        tl.Sigmoid(),
        tl.Dense(3),
        tl.Softmax())
```

Trax
Beneficios

Trax tiene varias ventajas:

- Código limpio y rápido
- Funciona rápido en CPU, GPU y TPU
- Computación paralela
- Registre cálculos algebraicos y lo almacena en orden para la evaluación de gradientes

Tensorflow Pytorch JAX

¿Por qué Trax y no TensorFlow o PyTorch?

TensorFlow y PyTorch son frameworks extensos que pueden hacer casi cualquier cosa en el deep learning. Ofrecen mucha flexibilidad, pero eso a menudo significa verbosidad de sintaxis y tiempo adicional para codificar.

Trax es mucho más conciso. Se ejecuta en un backend de TensorFlow, pero le permite entrenar modelos con comandos de 1 línea.

¿Por qué no Keras entonces?

Keras ahora es parte de Tensorflow desde 2.0 en adelante. Además, Trax es bueno para implementar nuevos algoritmos de última generación como Transformers, Reformers, BERT porque Google Brain Team lo mantiene activamente para tareas avanzadas de aprendizaje profundo.

¿Cómo codificar en Trax?

La construcción de modelos en Trax se basa en 2 conceptos clave: capas y combinadores.

- Las capas Trax son objetos simples que procesan datos y realizan cálculos.
- Se pueden encadenar en capas compuestas mediante los combinadores de Trax, lo que le permite crear capas y modelos de cualquier complejidad.

Trax, JAX, TensorFlow

Se sabe que Trax usa Tensorflow como backend, pero también usa la biblioteca JAX para acelerar el cálculo. (JAX es como una versión mejorada y optimizada de numpy).
JAX: Computación numérica de alto rendimiento.

```
import trax.fastmath.numpy as np
```

El código fuente de Trax se puede encontrar en Github: Trax

Biblioteca JAX: JAX

¿Por qué Trax?

¿Qué es lo que espera o desea de una biblioteca de aprendizaje profundo?

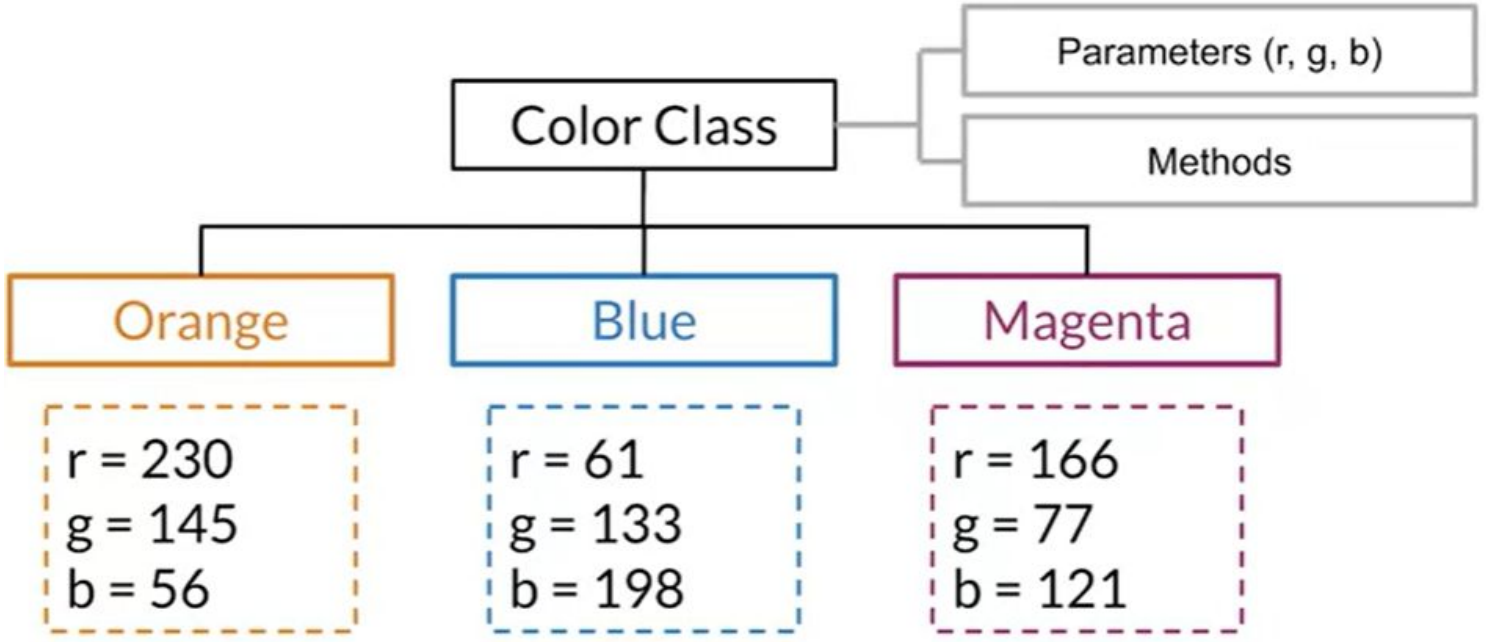
¿Por qué Trax?

¿Qué es lo que espera o desea de una biblioteca de aprendizaje profundo?

- Programadores sean eficientes. (Costo hh programador)
- El código se ejecuta rápido. (Costo computacional)

Clases, Subclases y Herencia

Clase: es una forma de definir propiedades y métodos comunes para objetos similares. En otras palabras, las clases definen variables y comportamientos comunes para los objetos asociados con ellas



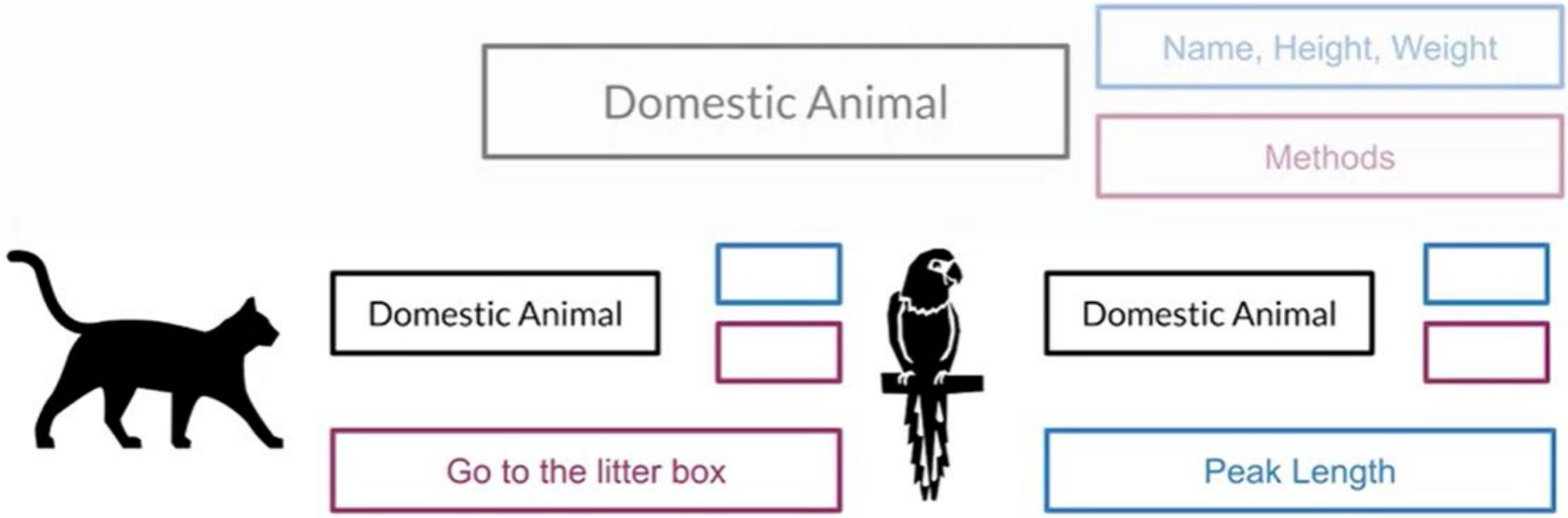
Clases en Python

```
class MyClass:

    def __init__(self, y):
        self.y = y

    def my_method(self,x):
        return x + self.y

    def __call__(self, x):
        return self.my_method(x)
```

```
f = MyClass(7)

print(f(3))
```

Subclases

Cuando tiene varias clases que comparten algunos parámetros y métodos, es conveniente definirlas como subclases de otra clase, a menudo conocida como clase principal.



Subclases en Python

Trabajar con subclases es conveniente porque sólo necesita especificar parámetros y métodos comunes en la clase principal y agregar las propiedades distintivas a cada subclase.

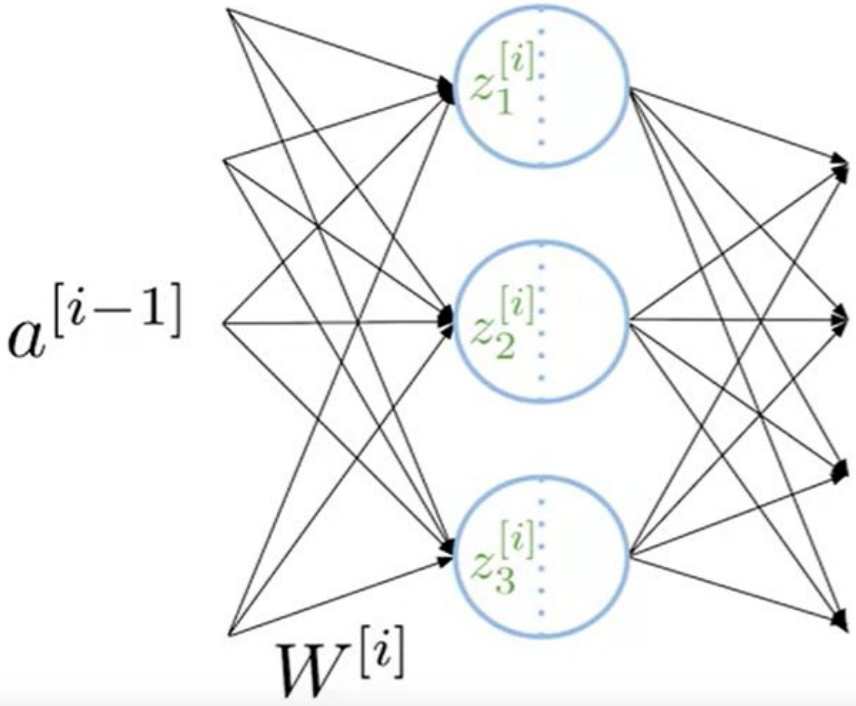
```
class MyClass:

    def __init__(self,y):
        self.y = y

    def my_method(self,x):
        return x + self.y

    def __call__(self,x):
        return self.my_method(x)
```

```
class SubClass(MyClass):

    def my_method(self,x):
        return x + self.y**2
```

```
f = SubClass(7)

print(f(3))
```

Dense Layer

Hay dos capas principales que se usan muy comúnmente en la mayoría de las redes neuronales. Hay una capa densa, que le permite pasar de una capa a otra dentro de la red, y luego hay una capa ReLU que mantiene estable su red.



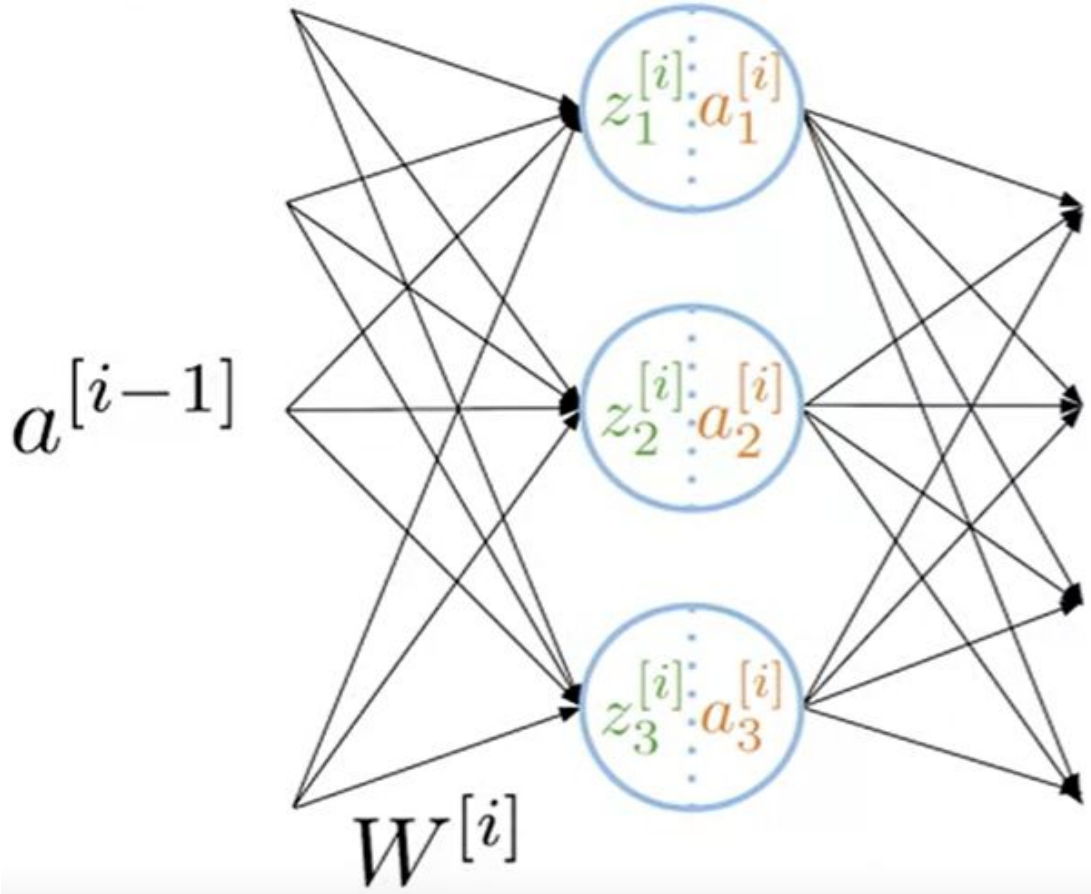
$$z_j^{[i]} = w_j^{[i]T} a^{[i-1]}$$

Dense layer

$$z^{[i]} = W^{[i]} a^{[i-1]}$$

Trainable parameters

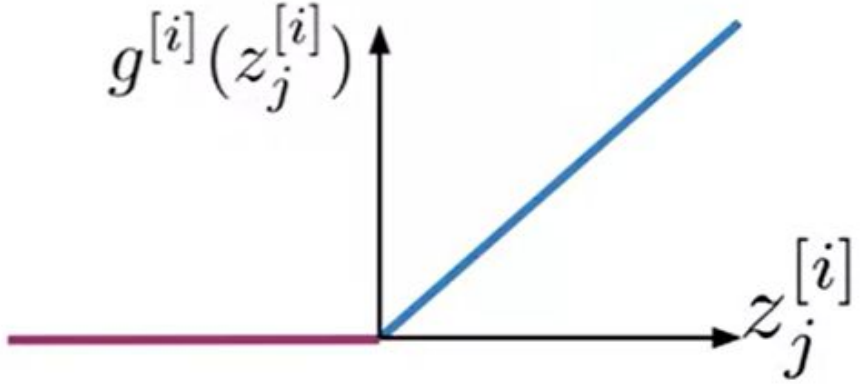
Layer ReLU



$$a_j^{[i]} = g^{[i]}(z_j^{[i]})$$

ReLU = Rectified linear unit

$$g(z^{[i]}) = \max(0, z^{[i]})$$



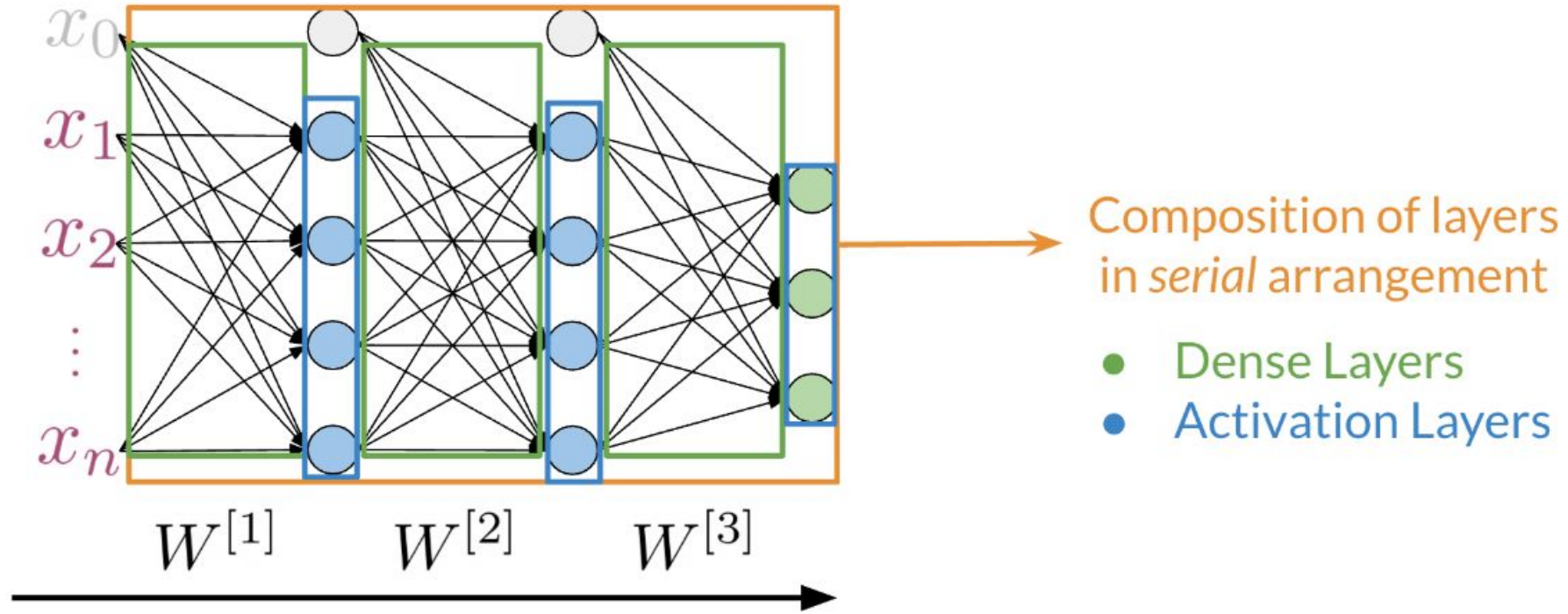
Dense Layer y ReLU

Dense Layer $\longrightarrow z^{[i]} = W^{[i]} a^{[i-1]}$

ReLU Layer $\longrightarrow g(z^{[i]}) = \max(0, z^{[i]})$

Serial Layer

Una capa en serie le permite componer capas en un arreglo en serie:



Es una composición de subcapas. Estas capas suelen ser capas densas seguidas de capas de activación.

Gradiente en Trax

$$f(x) = 3x^2 + x$$

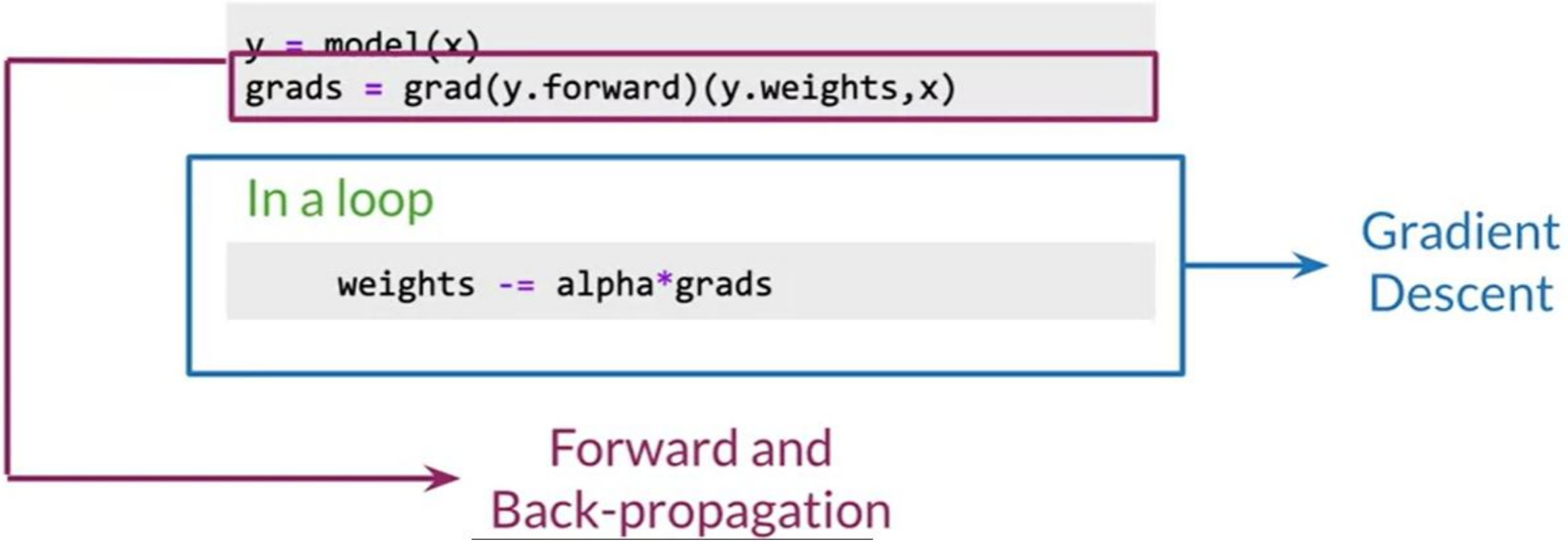
$$\frac{\delta f(x)}{\delta x} = 6x + 1$$

Gradient

```
def f(x):
      return 3*x**2 + x

grad_f = trax.math.grad(f)
```

Returns a function

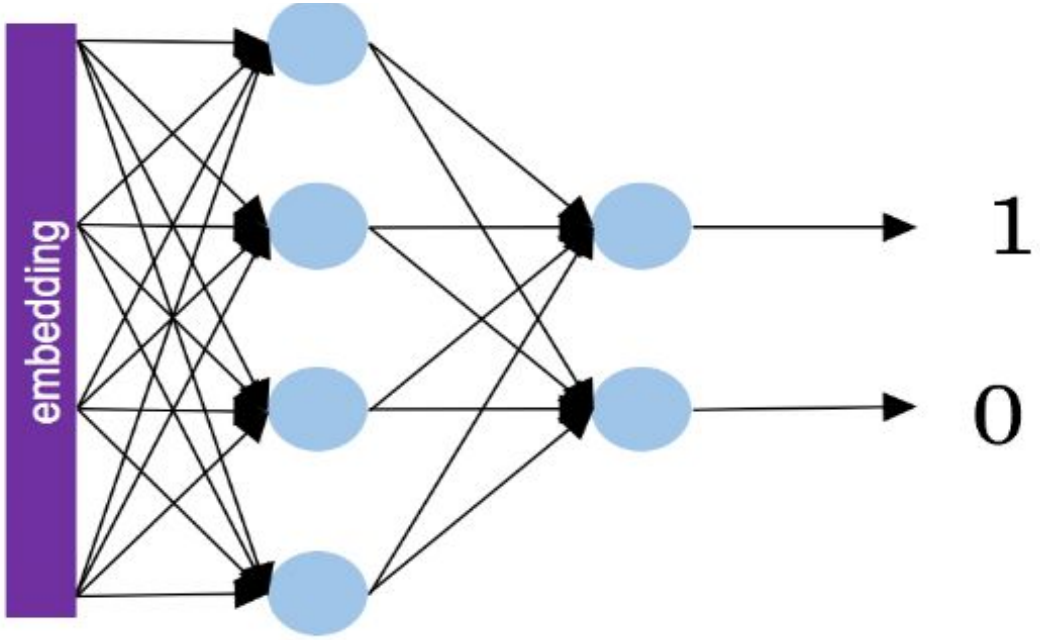
Gradiente en Trax



Laboratorio N°01 - Trax

- Instalar Trax
- Implemente la función de activación de ReLU en una clase.
- Implemente la función forward en una clase Dense
- Implemente un clasificador utilizando redes neuronales. Esta es la arquitectura modelo que implementará.

Utiliza la biblioteca de capas de Trax **tl**



Laboratorio N°01 - Trax

- Implemente la función de activación de ReLU en una clase.

```
class Relu(Layer):
    def forward(self, x):

        activation = #completar

        return activation
```

```
x = np.array([[-2.0, -1.0, 0.0], [0.0, 1.0, 2.0]], dtype=float)
relu_layer = Relu()
print("Test data is:")
print(x)
print("Output of Relu is:")
print(relu_layer(x))
```

Laboratorio N°01 - Trax

- Implemente la función forward en una clase Dense

```
class Dense(Layer):

    def __init__(self, n_units, init_stdev=0.1):
        self._n_units = n_units
        self._init_stdev = init_stdev
    def forward(self, x):
        dense = #codigo aquí
        return dense

    def init_weights_and_state(self, input_signature, random_key):
        input_shape = input_signature.shape
        w = self._init_stdev * random.normal(key = random_key, shape = (input_shape[-1], self._n_units))
        self.weights = w
```

Laboratorio N°01 - Trax

- Implemente la función forward en una clase Dense

```
# Testing your Dense layer
dense_layer = Dense(n_units=10)
random_key = random.get_prng(seed=0)
z = np.array([[2.0, 7.0, 25.0]])

dense_layer.init(z, random_key)
print("Weights are\n ",dense_layer.weights) #Returns pesos aleatorios
print("Foward function output is ", dense_layer(z))
```

Referencias Bibliográficas

Documentación oficial de Trax mantenida por el equipo de Google Brain:

https://trax-ml.readthedocs.io/en/latest/

Trax Código fuente en GitHub:

https://github.com/google/trax

Biblioteca JAX:

https://jax.readthedocs.io/en/latest/index.html

Gracias :)

Inteligencia computacional: la naturaleza como fuente de inspiración.