Intro DL

by **adsoft**

Deep Learning

Using Keras and Tensorflow you'll learn how to:

- create a **fully-connected** neural network architecture
- apply neural nets to two classic ML problems: **regression** and **classification**
- train neural nets with **stochastic gradient descent**, and
- improve performance with **dropout**, **batch normalization**, and other techniques

What is Deep Learning?

Some of the most impressive advances in artificial intelligence in recent years have been in the field of *deep learning*. *Natural language translation, image recognition, and game playing* are all tasks where deep learning models have neared or even exceeded human-level performance.

Deep learning is an approach to machine learning characterized by **deep stacks** of computations.

This depth of computation is what has enabled deep learning models to disentangle the kinds of complex and hierarchical patterns found in the most challenging real-world datasets

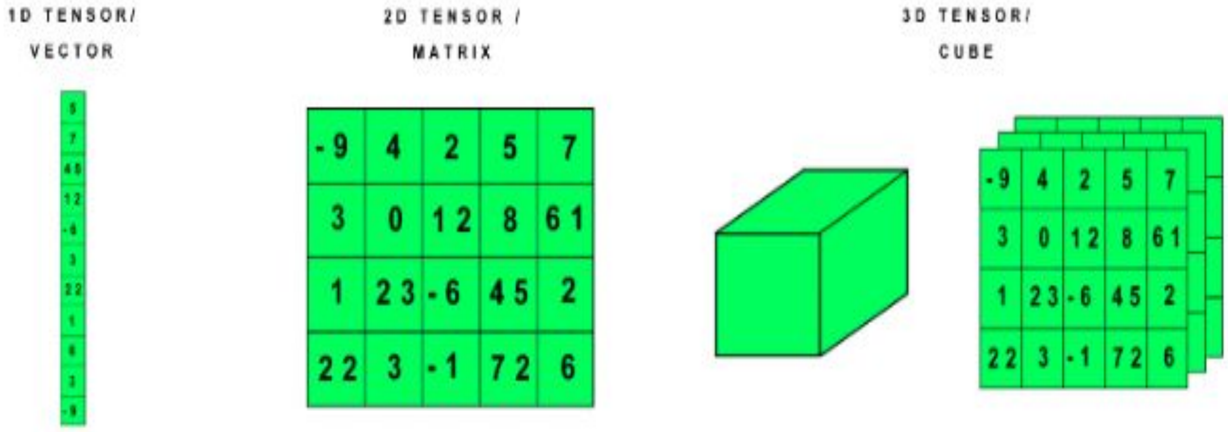
What is a tensor

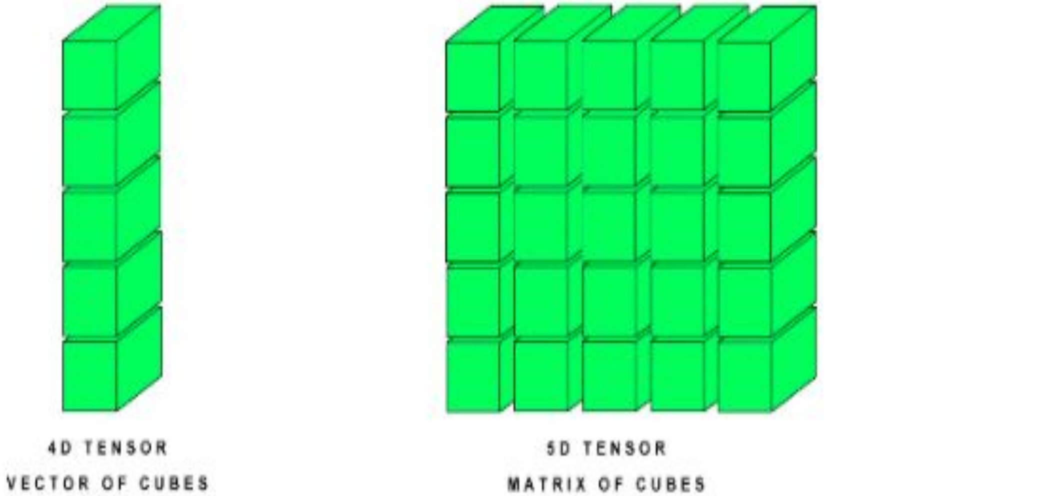
A tensor is the basic building block of modern machine learning.

At its core it's a data container.

Mostly it contains numbers. Sometimes it even contains strings, but that's rare. So think of it as a bucket of numbers

There are multiple sizes of tensors. Let's go through the most basic ones that you'll run across in deep learning, which will be between 0 and 5 dimensions





https://www.youtube.com/watch?v=bPPLCrjQCBQ create an virtual environment

$ python3 -m venv tensorflow-2.0

activate

$ source tensorflow-2.0/bin/activate

Tensorflow install

$ mkdir tensorflow
$ cd tensorflow

vi requirements.txt

tensorflow
numpy
keras

$ pip install -r requirements.txt

tensorflow - hello world

$ vi hello_world.py

```
import warnings
import logging, os

warnings.filterwarnings("ignore")
logging.disable(logging.WARNING)
os.environ["TF_CPP_MIN_LOG_LEVEL"] = "3"

import tensorflow as tf

# create a Tensor 0D, escalar
hello = tf.constant("hello world")
print(hello)

# to acces a Tensor value, call numpy()
print(hello.numpy())
```

run - hello world

$ python hello_world.py

```
tf.Tensor(b'hello world', shape=(), dtype=string)
b'hello world'
```

update - hello world (comments warnings, logs..)

```python
import warnings
import logging, os

#warnings.filterwarnings("ignore")
#logging.disable(logging.WARNING)
#os.environ["TF_CPP_MIN_LOG_LEVEL"] = "3"

import tensorflow as tf

# create a Tensor
hello = tf.constant("hello world")
print(hello)

# to acces a Tensor value, call numpy()
print(hello.numpy())
```

$python hello_world.py

```
2023-11-07 22:10:33.965172: I tensorflow/tsl/cuda/cudart_stub.cc:28] Could not find cuda drivers on your machine, GPU will not be used.
2023-11-07 22:10:33.996255: E tensorflow/compiler/xla/stream_executor/cuda/cuda_dnn.cc:9342] Unable to register cuDNN factory: Attempting to register
 factory for plugin cuDNN when one has already been registered
2023-11-07 22:10:33.996311: E tensorflow/compiler/xla/stream_executor/cuda/cuda_fft.cc:609] Unable to register cuFFT factory: Attempting to register
factory for plugin cuFFT when one has already been registered
2023-11-07 22:10:33.996404: E tensorflow/compiler/xla/stream_executor/cuda/cuda_blas.cc:1518] Unable to register cuBLAS factory: Attempting to regist
er factory for plugin cuBLAS when one has already been registered
2023-11-07 22:10:34.003410: I tensorflow/tsl/cuda/cudart_stub.cc:28] Could not find cuda drivers on your machine, GPU will not be used.
2023-11-07 22:10:34.003677: I tensorflow/core/platform/cpu_feature_guard.cc:182] This TensorFlow binary is optimized to use available CPU instruction
s in performance-critical operations.
To enable the following instructions: AVX2 FMA, in other operations, rebuild TensorFlow with the appropriate compiler flags.
2023-11-07 22:10:34.945045: W tensorflow/compiler/tf2tensorrt/utils/py_utils.cc:38] TF-TRT Warning: Could not find TensorRT
tf.Tensor(b'hello world', shape=(), dtype=string)
b'hello world'
```

see video:

https://www.youtube.com/watch?v=-P28LKWTzrI

tensor-operations.py (part 1)

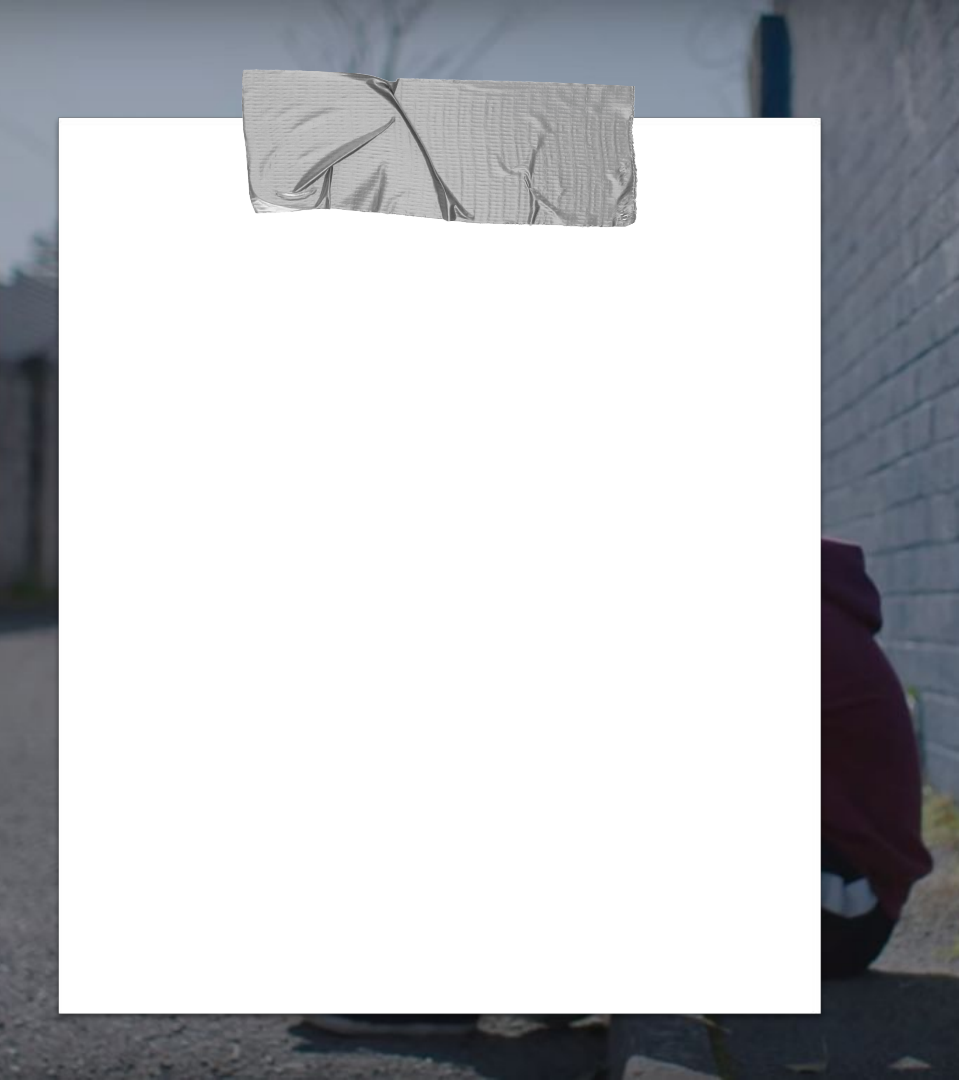
```
from __future__ import print_function
import warnings
import logging, os
warnings.filterwarnings("ignore")
logging.disable(logging.WARNING)
os.environ["TF_CPP_MIN_LOG_LEVEL"] = "3"

import tensorflow as tf
# define tensor constants
a = tf.constant(2)
b = tf.constant(3)
c = tf.constant(5)

# Various tensor operations
# Note: Tensor also support python operators (+, *, ...)
add = tf.add(a, b)
sub = tf.subtract(a, b)
mul = tf.multiply(a, b)
div = tf.divide(a, b)
```

tensor-operations.py (part 2)

```
# Access tensors value
print("add = ", add.numpy())
print("sub = ", sub.numpy())
print("mul = ", mul.numpy())
print("div = ", div.numpy())

# Some more operations.
mean = tf.reduce_mean([a, b, c])
sum = tf.reduce_sum([a, b, c])

# Access tensors value.
print("mean =", mean.numpy())
print("sum =", sum.numpy())

# Matrix multiplications.
matrix1 = tf.constant([[1., 2.], [3., 4.]])
matrix2 = tf.constant([[5., 6.], [7., 8.]])
product = tf.matmul(matrix1, matrix2)
print (product)
print (product.numpy())
```

$ python tensor-operations.py

```
add =  5
sub =  -1
mul =  6
div =  0.6666666666666666
mean = 3
sum = 10
tf.Tensor(
[[19. 22.]
 [43. 50.]], shape=(2, 2), dtype=float32)
[[19. 22.]
 [43. 50.]]
```



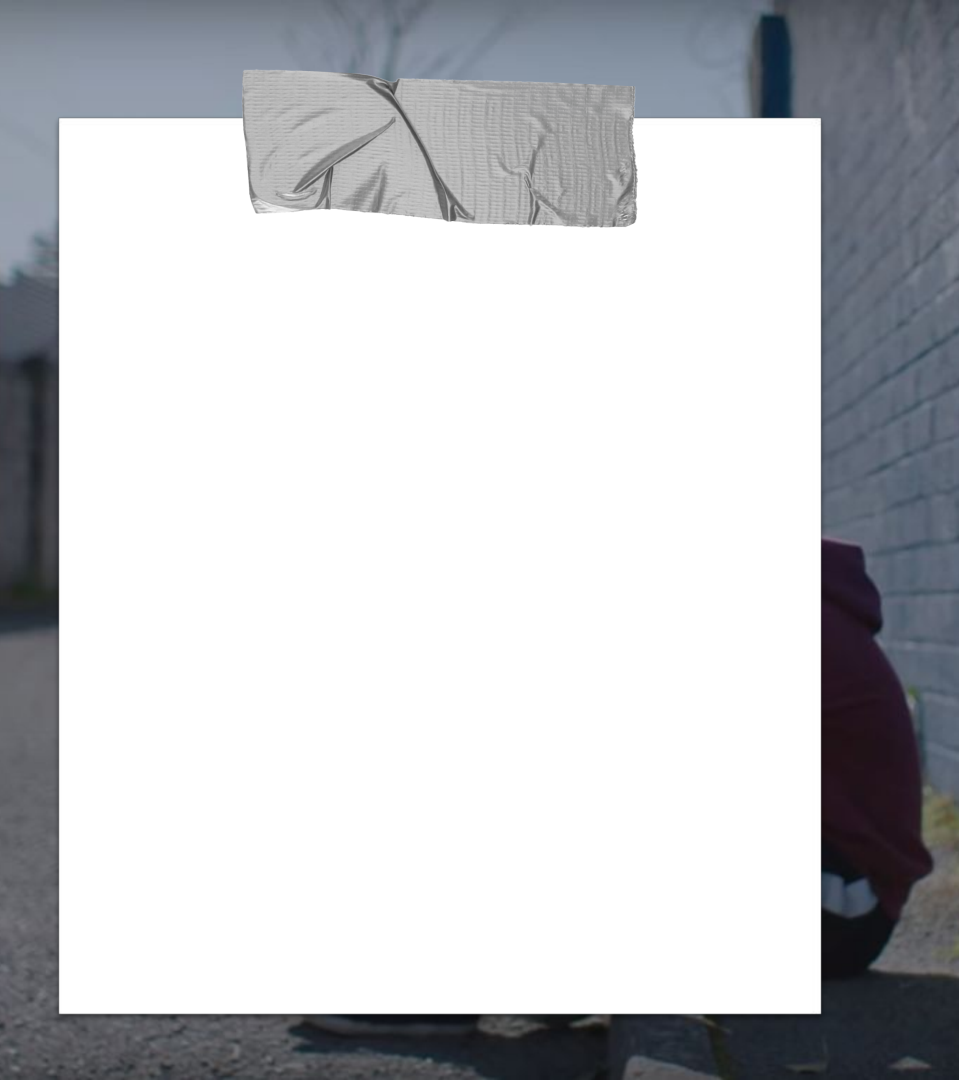
tensor-board.py (1)

$ mkdir graphs

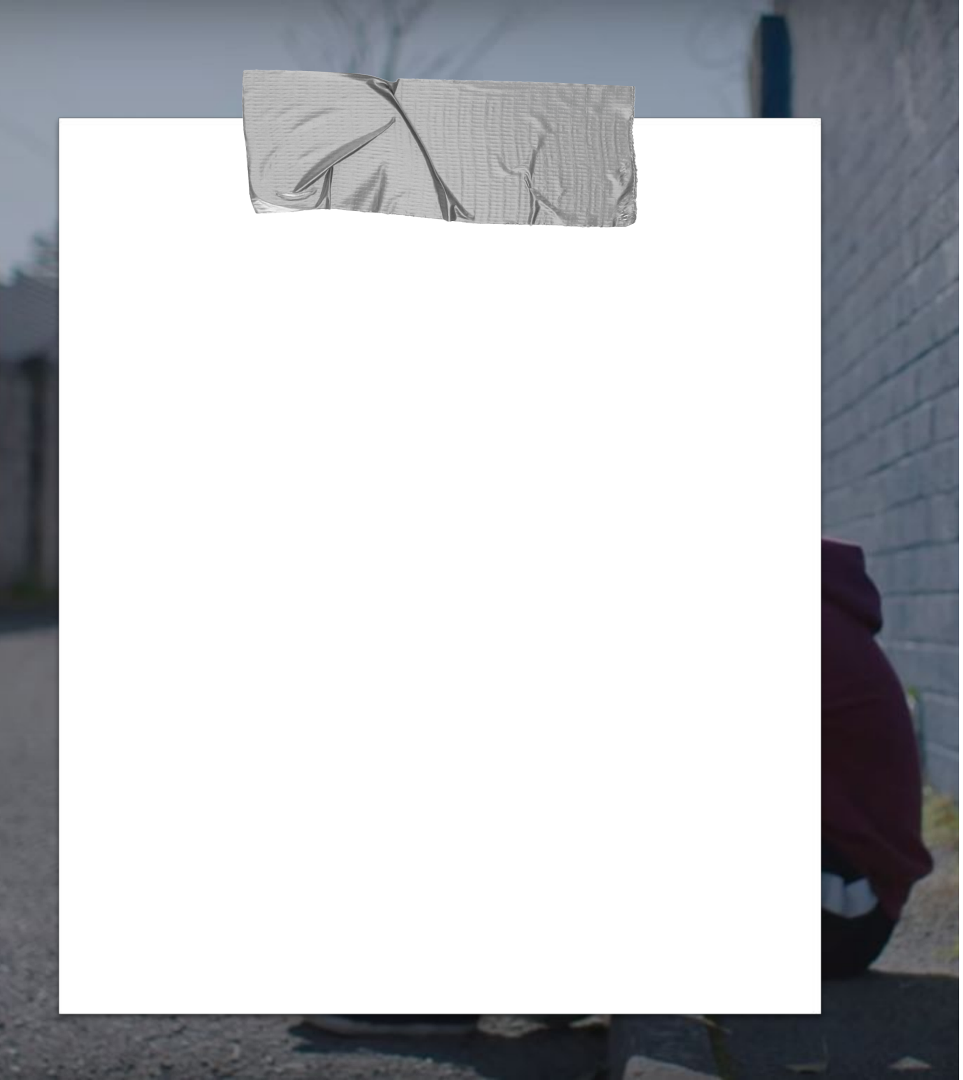
$ vi tensor-board.py

```
from __future__ import print_function
import warnings
import logging, os
warnings.filterwarnings("ignore")
logging.disable(logging.WARNING)
os.environ["TF_CPP_MIN_LOG_LEVEL"] = "3"
import tensorflow as tf
from tensorflow.python.ops import summary_ops_v2
# Graph
a = tf.Variable(2, name='a')
b = tf.Variable(3, name='b')
```

tensor-board.py (2)



```
@tf.function  # tf.function allows us to take a graph from a function
def graph_to_visualize(a, b):
    c = tf.add(a, b, name='Add')

# Visualize
writer = tf.summary.create_file_writer('./graphs')

with writer.as_default():
    graph = graph_to_visualize.get_concrete_function(a, b).graph
    # get graph from function
    summary_ops_v2.graph(graph.as_graph_def()) # visualize

writer.close()
```

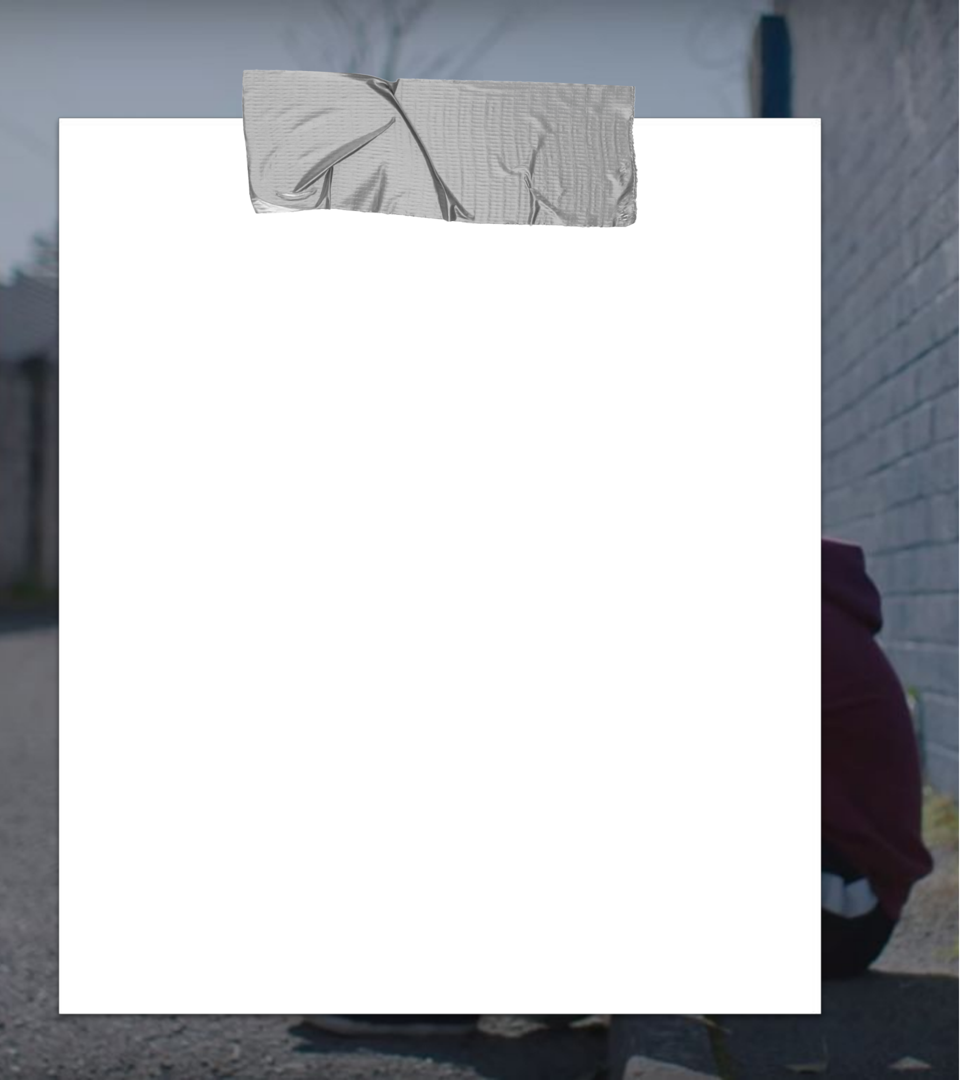


$ python tensor-board.py

$ ls graphs

events.out.tfevents.1699399936.codespaces-f115e4.16130.0.v2

$ tensorboard --logdir='./graphs' --host 0.0.0.0 --port 8080



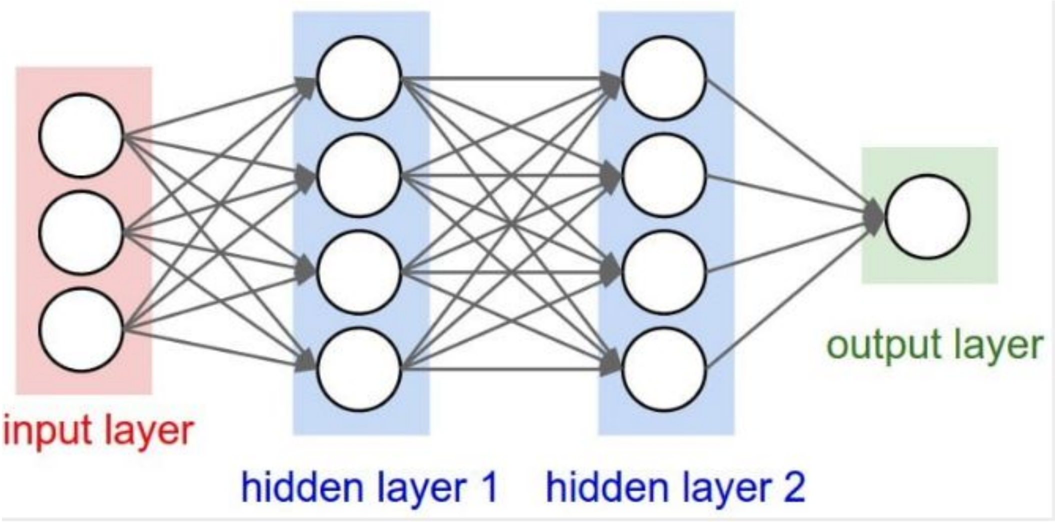
$ curl https://github.com/adsoftsito/tensorflow-2.0/blob/main/tensorboard_example2.py > tensor-board2.py

$ python tensor-board2.py

$ tensorboard --logdir='./graphs' --host 0.0.0.0 --port 8080

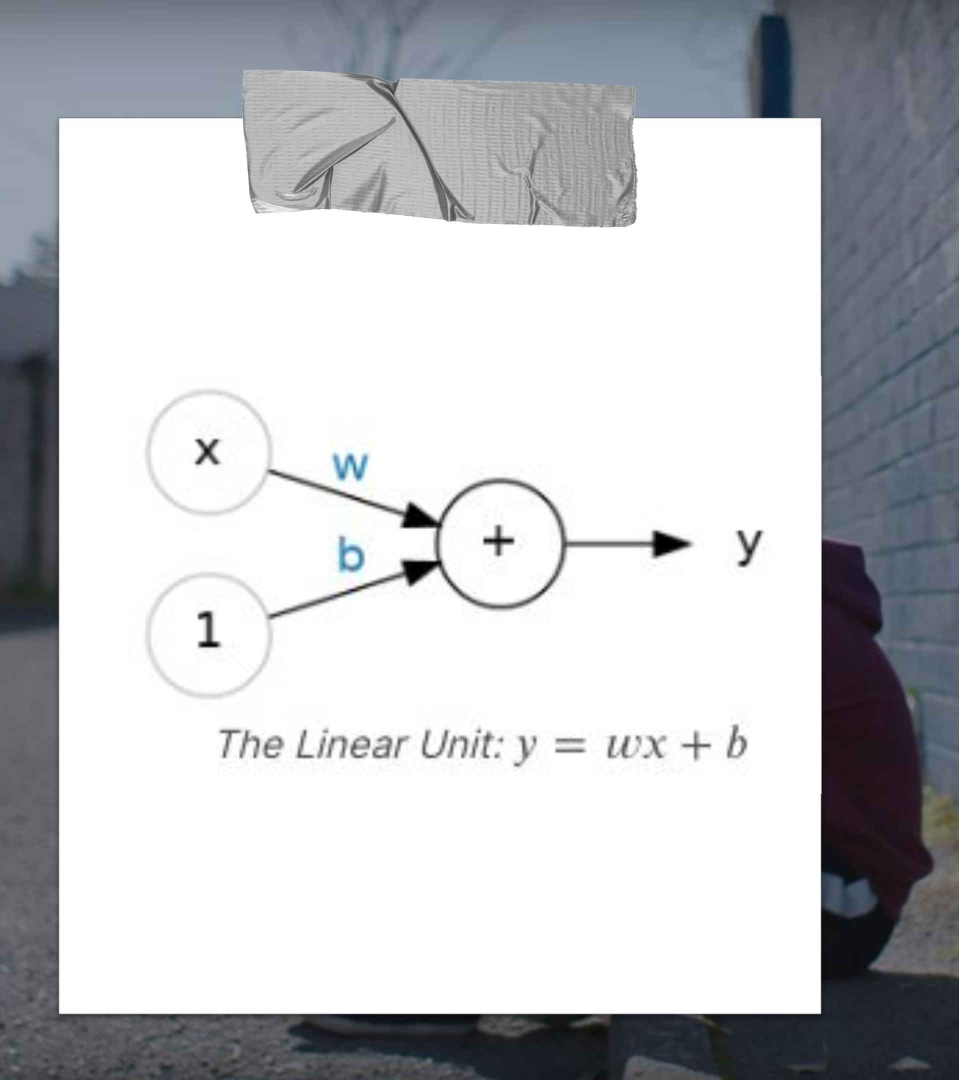
A Single Neuron

Learn about **linear units**, the building blocks of deep learning.



Two or more hidden layers comprise a Deep Neural Network

The Linear Unit



The Linear Unit: $y = wx + b$

The input is **x**. Its connection to the neuron has a **weight** which is **w**.

Whenever a value flows through a connection, you multiply the value by the connection's weight. For the input x, what reaches the neuron is `w * x`. ***A neural network "learns" by modifying its weights***.

The b is a special kind of weight we call the **bias.** The bias doesn't have any input data associated with it; instead, we put a 1 in the diagram so that the value that reaches the neuron is just b (since `1 * b = b`). The bias enables the neuron to modify the output independently of its inputs.

The y is the value the neuron ultimately outputs. To get the output, the neuron sums up all the values it receives through its connections. This neuron's activation is `y = w * x + b`, or as a formula

$$y = wx + b$$

```python
from tensorflow import keras
from tensorflow.keras import layers


# Create a network with 1 linear unit
model = keras.Sequential([
    layers.Dense(units=1, input_shape=[3])
])
```

```python
model.summary()

w, b = model.weights

print(w)

print(b)
```

try linear_model :

$ curl https://github.com/adsoftsito/aiops/blob/main/linear.py > linear.py

$ python linear.py

pull serving

$ docker pull tensorflow/serving

```
$ docker run -d --name
serving_base
tensorflow/serving
```

```
$ docker cp linear-model
serving_base:/models/linear-model
```

```
$ docker commit --change "ENV MODEL_NAME
linear-model" serving_base
<user>/tensorflow-serving
```

```
$ docker run -d --name
my_serving_base -p 8501:8501
<user>/tensorflow-serving
```

tensor_client.py

```
import json
import numpy as np
import requests
# The server URL specifies the endpoint of your server
running the linear_model
# model with the name "linear_model" and using the predict
interface.
SERVER_URL =
'http://localhost:8501/v1/models/linear-model:predict'

def main():
  predict_request = '{"instances" : [ [0.0], [1.0], [2.0]
]}'
  # Send few actual requests and report average latency.
  total_time = 0
  num_requests = 10
  index = 0
  for _ in range(num_requests):
    response = requests.post(SERVER_URL,
data=predict_request)
    response.raise_for_status()
    total_time += response.elapsed.total_seconds()
    prediction = response.json()
    print (prediction)
  print('Prediction class: {}, avg latency: {} ms'.format(
      np.argmax(prediction), (total_time * 1000) /
num_requests))

if __name__ == '__main__':
  main()
```

$python tensor_client.py

aiops — m@MacBook-Pr

```
➜  aiops git:(main) ✗ python model_client.py
{'predictions': [[0.999998629], [2.99999857], [4.99999857]]}
{'predictions': [[0.999998629], [2.99999857], [4.99999857]]}
{'predictions': [[0.999998629], [2.99999857], [4.99999857]]}
{'predictions': [[0.999998629], [2.99999857], [4.99999857]]}
{'predictions': [[0.999998629], [2.99999857], [4.99999857]]}
{'predictions': [[0.999998629], [2.99999857], [4.99999857]]}
{'predictions': [[0.999998629], [2.99999857], [4.99999857]]}
{'predictions': [[0.999998629], [2.99999857], [4.99999857]]}
{'predictions': [[0.999998629], [2.99999857], [4.99999857]]}
{'predictions': [[0.999998629], [2.99999857], [4.99999857]]}
Prediction class: 0, avg latency: 338.3797 ms
➜  aiops git:(main) ✗
```

Github Actions with tensorflow

https://github.com/adsoftsito/tecnologias-construccion/blob/main/modelops_devops.docx

Stochastic gradient descent

https://www.analyticslane.com/2018/12/21/implementacion-del-metodo-descenso-del-gradiente-en-python/

Test SGD - sgd.py (part 1)

```
import numpy as np
# Creación de un conjunto de datos para entrenamiento
trX = np.linspace(-2, 2, 10)
trY = 5 * trX + 10.0

def gradient_func(W, x, b):
 return W*x + b

# Definición de los ajustes y parámetros iniciales
num_steps = 100
learningRate = 0.10
criteria = 1e-8
b = 1
W = 1
```

Test SGD - sgd.py (part 2)

```
# Proceso iterativo
for step in range(0, num_steps):
   b_gradient = 0
   W_gradient = 0
   N = float(len(trX))
   for i in range(0, len(trX)):
       b_gradient -= (2/N) * (trY[i] - gradient_func(W, trX[i], b))
       W_gradient -= (2/N) * (trY[i] - gradient_func(W, trX[i], b)) * trX[i]
       print(W_gradient)
   print('=============================')
   b = b - (learningRate * b_gradient)
   W = W - (learningRate * W_gradient)
   print(W_gradient, b_gradient)
   print(W, b)
   print('----')
   if max(abs(learningRate * b_gradient), abs(learningRate * W_gradient)) < criteria:
       break

# Impresión de los resultados
print("Los valores que se obtienen son:", W, b, "en pasos", step)
```

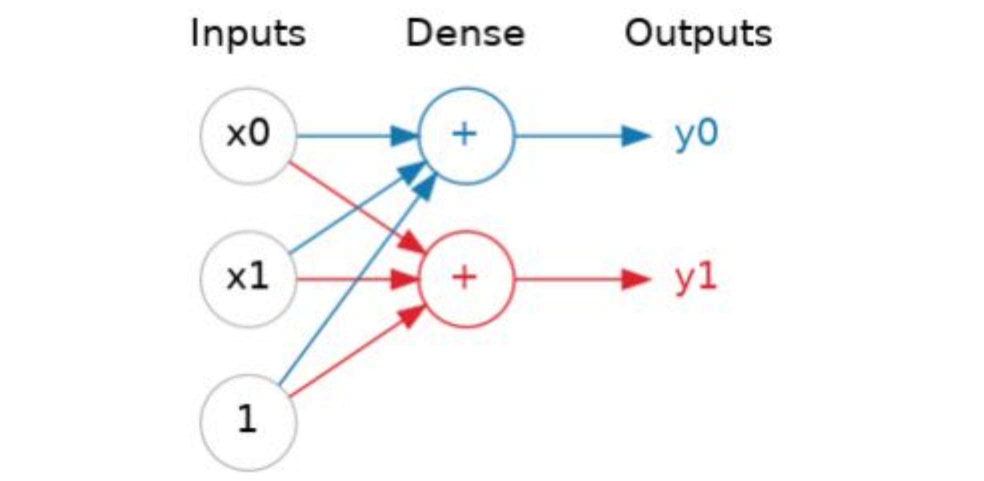
Complete a single neuron lesson

https://www.kaggle.com/code/ryanholbrook/a-single-neuron

Deep neural network lesson

Add hidden layers to your network to uncover complex relationships.

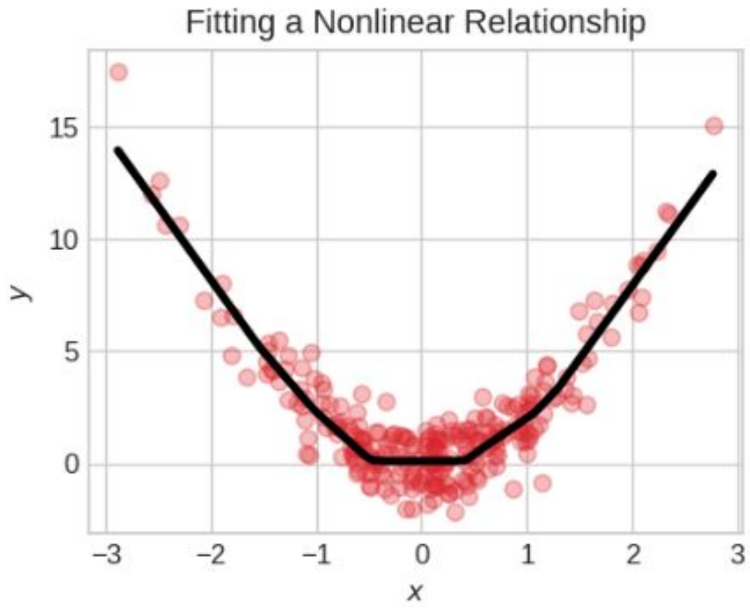
You could think of each layer in a neural network as performing some kind of relatively simple transformation. Through a deep stack of layers, a neural network can transform its inputs in more and more complex ways. In a well-trained neural network, each layer is a transformation getting us a little bit closer to a solution.



A dense layer of two linear units receiving two inputs and a bias.

Activation Function

What we need is something *nonlinear*. What we need are activation functions.



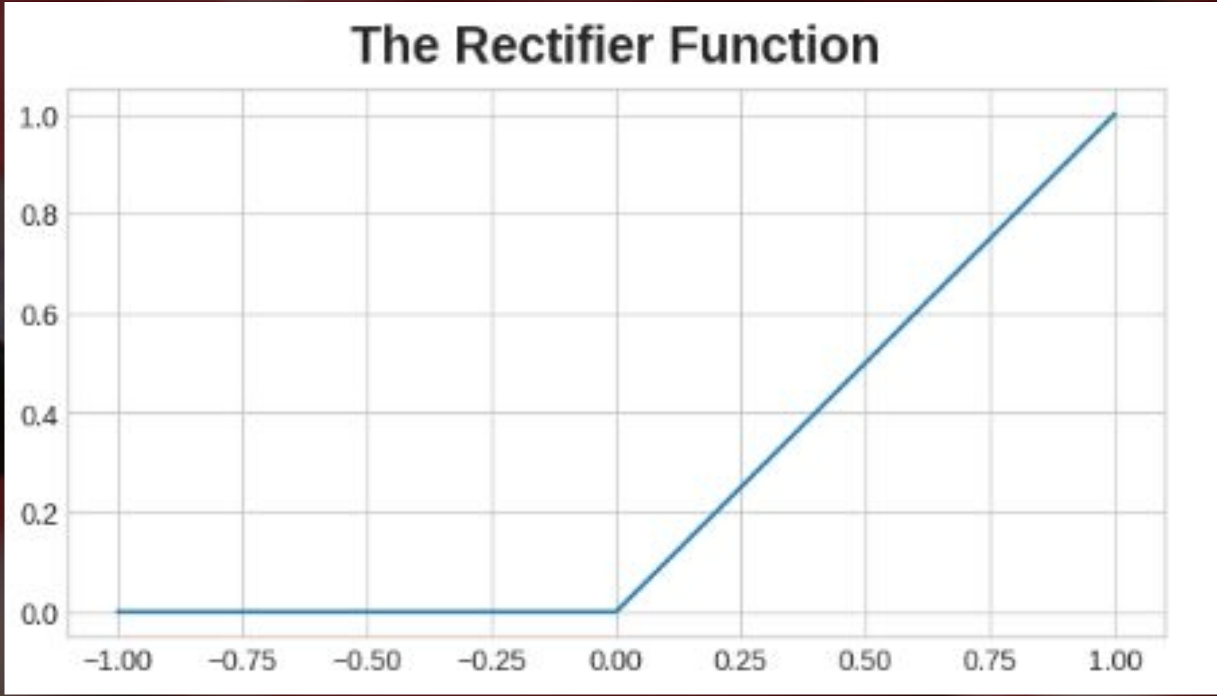
Without activation functions, neural networks can only learn linear relationships. In order to fit curves, we'll need to use activation functions.

Activation Function

An **activation function** is simply some function we apply to each of a layer's outputs (its *activations*). The most common is the *rectifier* function

ReLU - $max(0,x)$

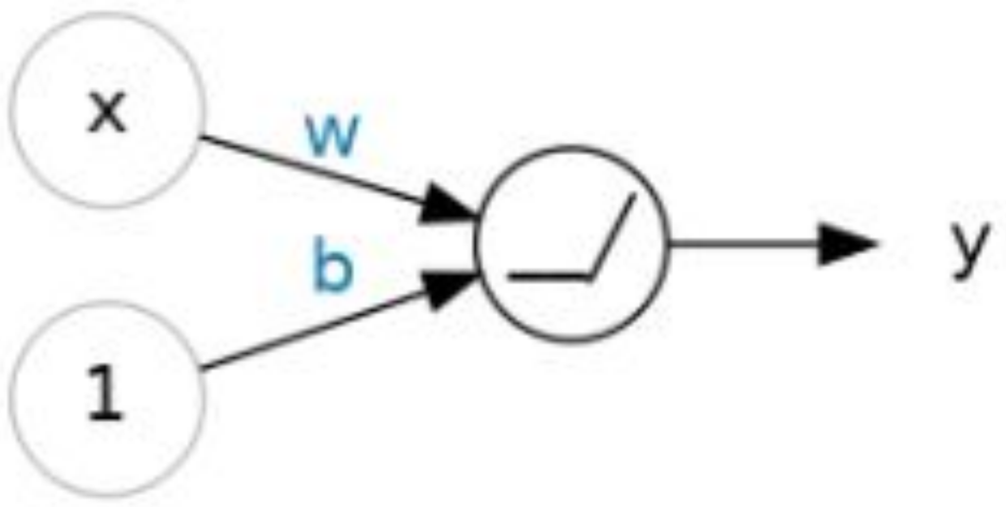
The rectifier function has a graph that's a line with the negative part "rectified" to zero. Applying the function to the outputs of a neuron will put a *bend* in the data, moving us away from simple lines.



ReLU - $max(0,x)$

When we attach the rectifier to a linear unit, we get a **rectified linear unit** or **ReLU**. (For this reason, it's common to call the rectifier function the "ReLU function".)

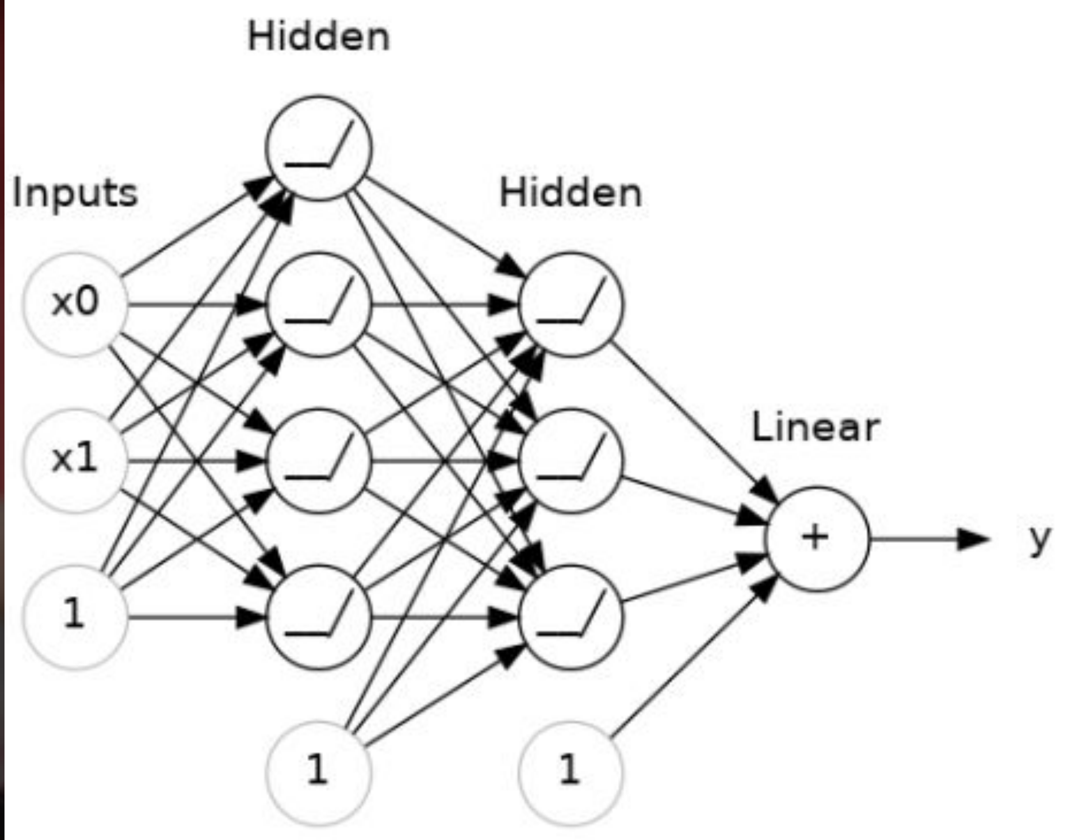
Applying a ReLU activation to a linear unit means the output becomes `max(0, w * x + b)`



A rectified linear unit.

ReLU - $max(0,x)$

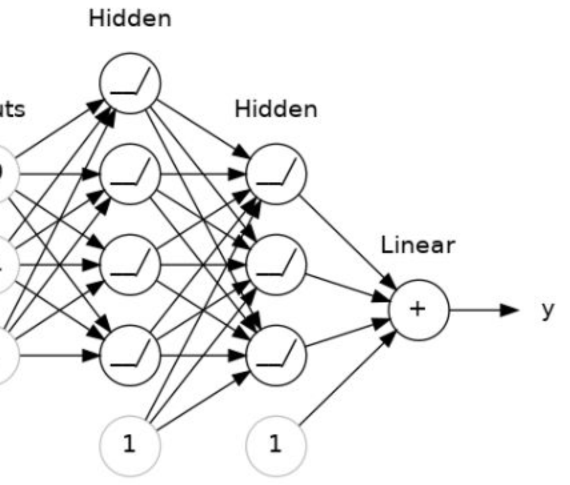
Now, notice that the final (output) layer is a linear unit (meaning, no activation function). That makes this network appropriate to a **regression task**, where we are trying to predict some arbitrary numeric value. Other tasks (like **classification**) might require an activation function on the output.



A stack of dense layers makes a "fully-connected" network.

The layers before the output layer are sometimes called **hidden** since we never see their outputs directly.

ReLU - $max(0,x)$



tack of dense layers makes a "fully-connected" network.

```
from tensorflow import keras
from tensorflow.keras import layers

model = keras.Sequential([
    # the hidden ReLU layers
    layers.Dense(units=4, activation='relu', input_shape=[2]),
    layers.Dense(units=3, activation='relu'),
    # the linear output layer
    layers.Dense(units=1),
])
```

model.summary()

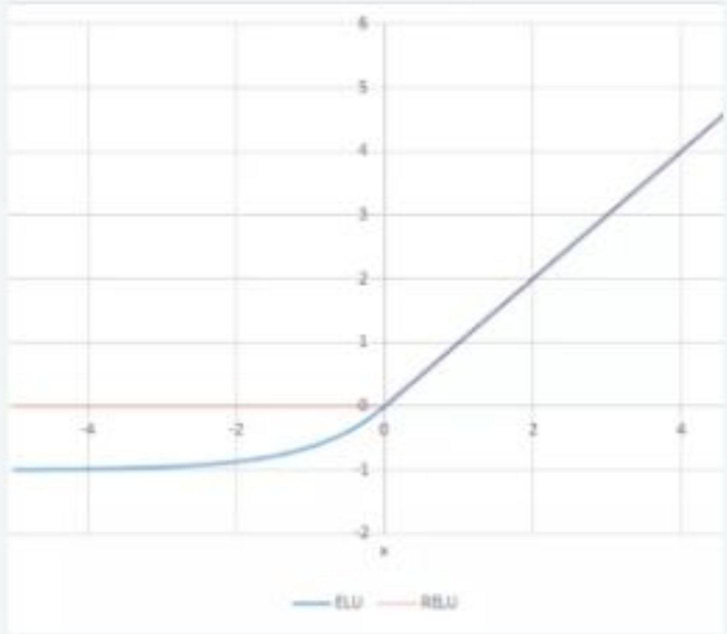
elu

$$R(z) = \begin{cases} z & z > 0 \\ \alpha.(e^z - 1) & z <= 0 \end{cases}$$

Note: e=2.71

xponential Linear Unit or its widely
nown name ELU is a function that tend to
onverge cost to zero faster and produce
nore accurate results.

ifferent to other activation functions,
LU has a extra alpha constant which
hould be positive number.



```
def elu(z,alpha):
	return z if z >= 0 else alpha*(e^z -1)
```

elu

$$R(z) = \begin{cases} z & z > 0 \\ \alpha.(e^z - 1) & z <= 0 \end{cases}$$

ros

- ELU becomes smooth slowly until its output equal to -α whereas RELU sharply smoothes.
- ELU is a strong alternative to ReLU.
- Unlike to ReLU, ELU can produce negative outputs.

ons

- For x > 0, it can blow up the activation with the output range of [0, inf].

selu

Scaled Exponential Linear Unit

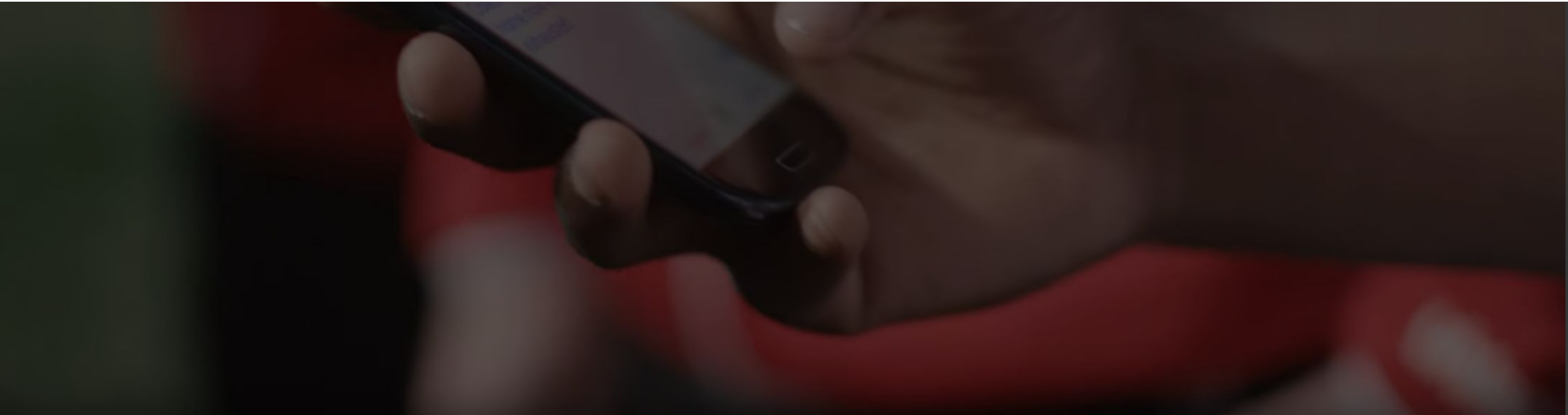
$$f(x) = \lambda x \qquad \text{if } \; x > 0$$

$$f(x) = \lambda\alpha(e^x - 1) \qquad \text{if } \; x \leq 0$$

/here λ and α are the following approximate values:

$$\lambda \approx 1.0507009873554804934193349852946$$

$$a \approx 1.6732632423543772848170429916717$$

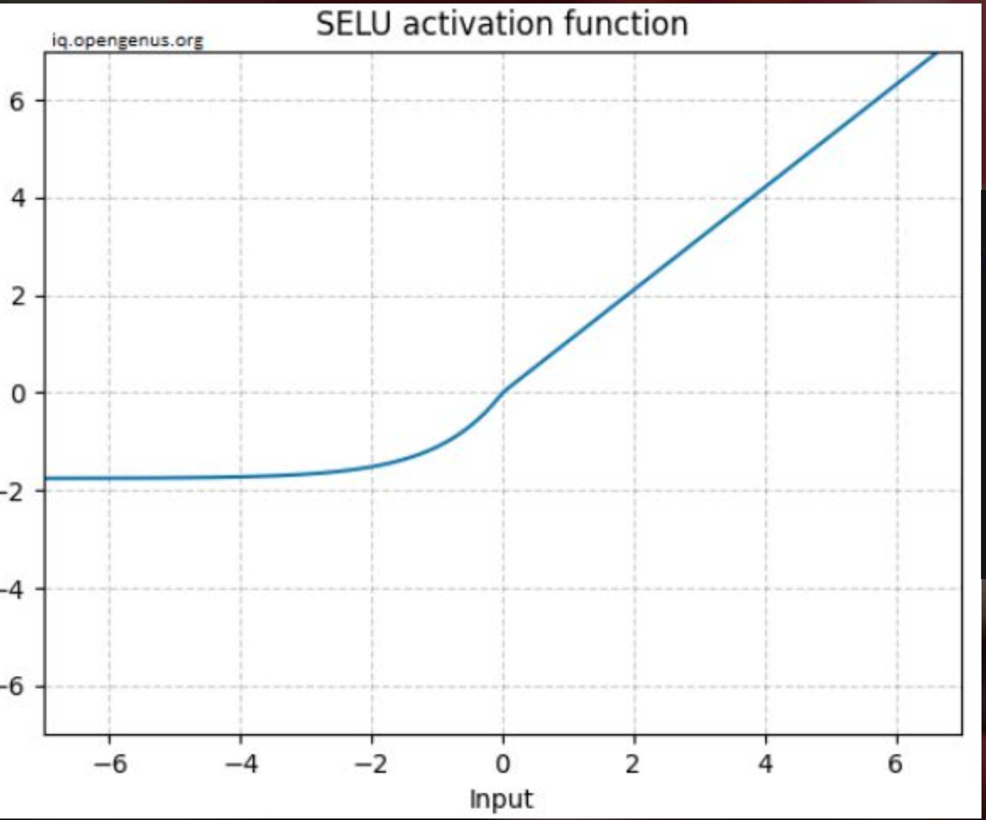


selu

Scaled Exponential Linear Unit

$$f(x) = \lambda x \qquad \text{if } x > 0$$

$$f(x) = \lambda\alpha(e^x - 1) \qquad \text{if } x \leq 0$$




```
def SELU(x, lambdaa = 1.0507, alpha = 1.6732):
    if x >= 0:
        return lambdaa * x
    else:
        return lambdaa * alpha * (np.exp(x) - 1)
```

sigmoid

gmoid takes a real value as input and

utputs another value between 0 and 1.

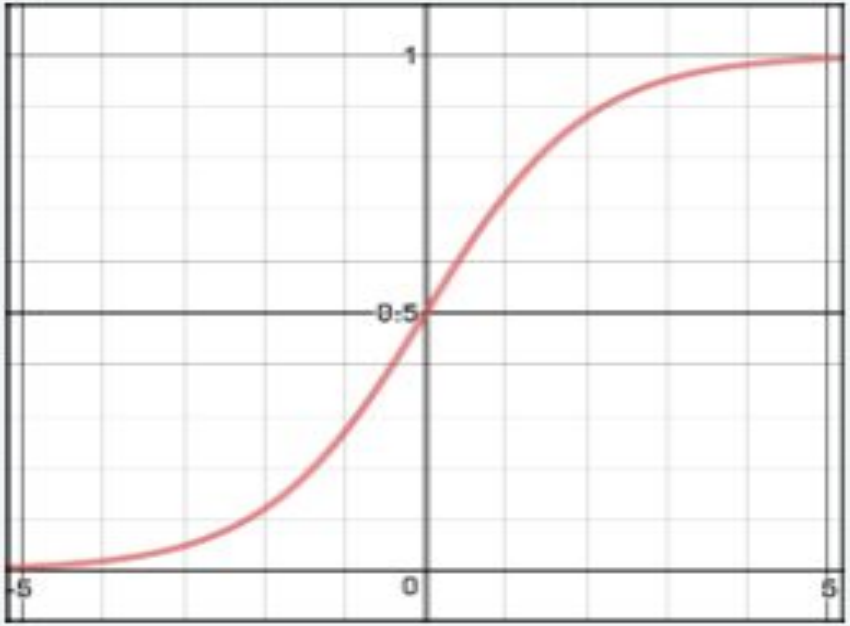
s easy to work with and has all the nice

operties of activation functions: it’s

on-linear, continuously differentiable,

onotonic, and has a fixed output range.

$$S(z) = \frac{1}{1 + e^{-z}}$$



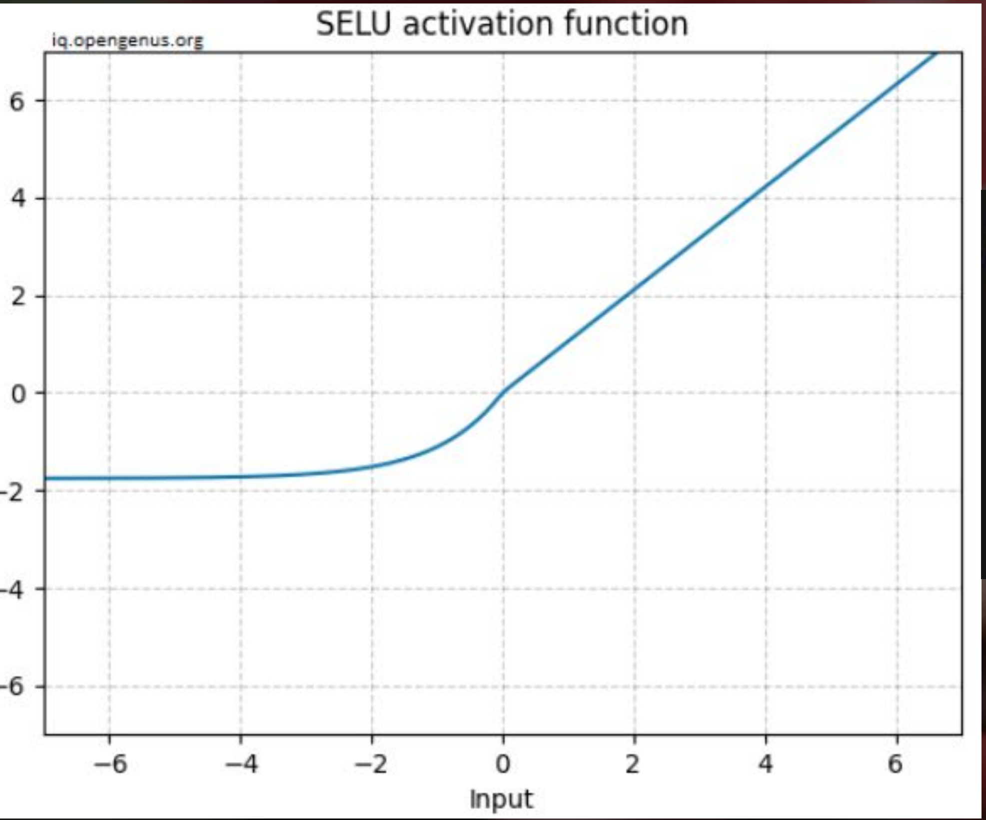

```
def sigmoid(z):
    return 1.0 / (1 + np.exp(-z))
```

selu

Scaled Exponential Linear Unit

$$f(x) = \lambda x \qquad \text{if } x > 0$$

$$f(x) = \lambda\alpha(e^x - 1) \qquad \text{if } x \leq 0$$




```python
def SELU(x, lambdaa = 1.0507, alpha = 1.6732):
    if x >= 0:
        return lambdaa * x
    else:
        return lambdaa * alpha * (np.exp(x) - 1)
```

sigmoid

os

- It is **nonlinear** in nature. Combinations of this function are also nonlinear!
- It will give an analog activation unlike step function.
- It has a smooth gradient too.
- It's good for a **classifier**.
- The output of the activation function is always going to be in range (0,1) compared to (-inf, inf) of linear function. So we have our activations bound in a range. Nice, it won't blow up the activations then.

Cons

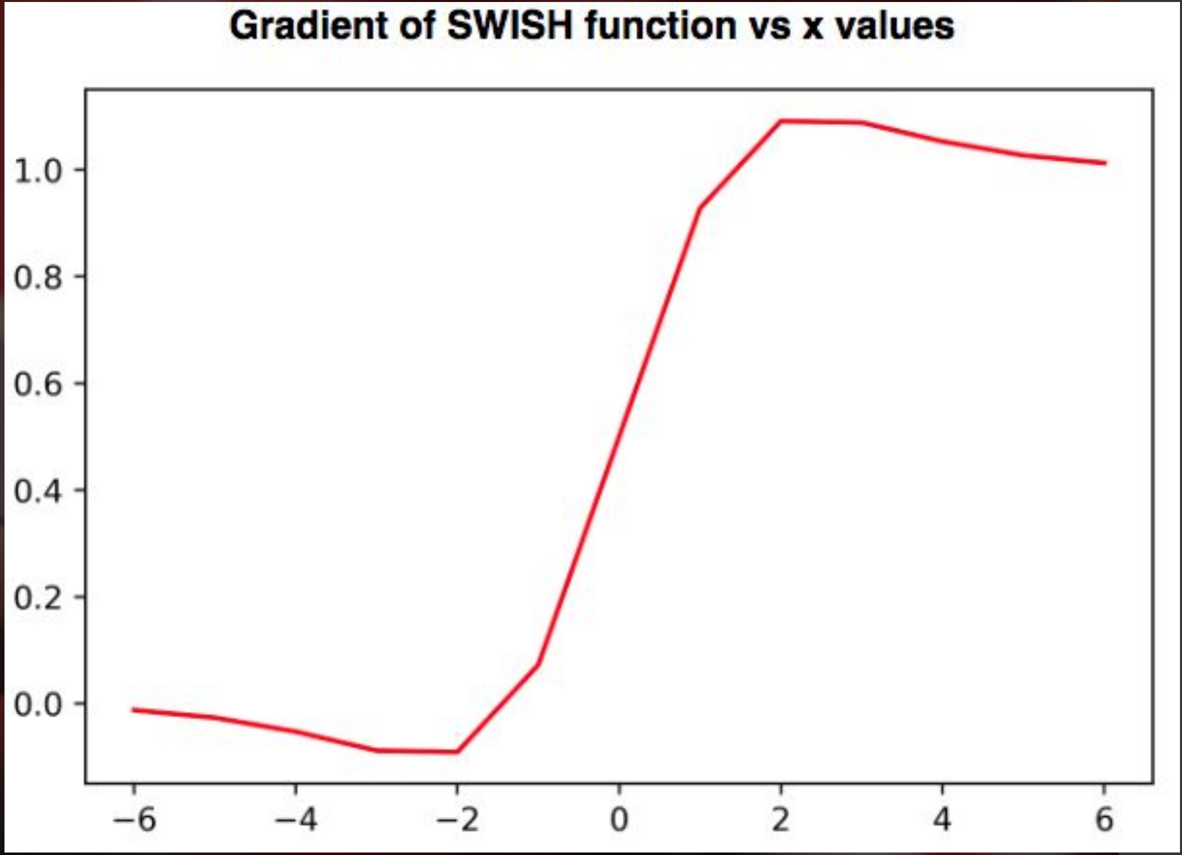
- Towards either end of the sigmoid function, the Y values tend to respond very less to changes in X.
- It gives rise to a problem of "vanishing gradients".
- Its output isn't zero centered. It makes the gradient updates go too far in different directions. 0 < output < 1, and it makes optimization harder.
- Sigmoids saturate and kill gradients.
- The network refuses to learn further or is drastically slow (depending on use case and until gradient /computation gets hit by floating point value limits)

swish

$$S(z) = \frac{1}{1+e^{-z}}$$

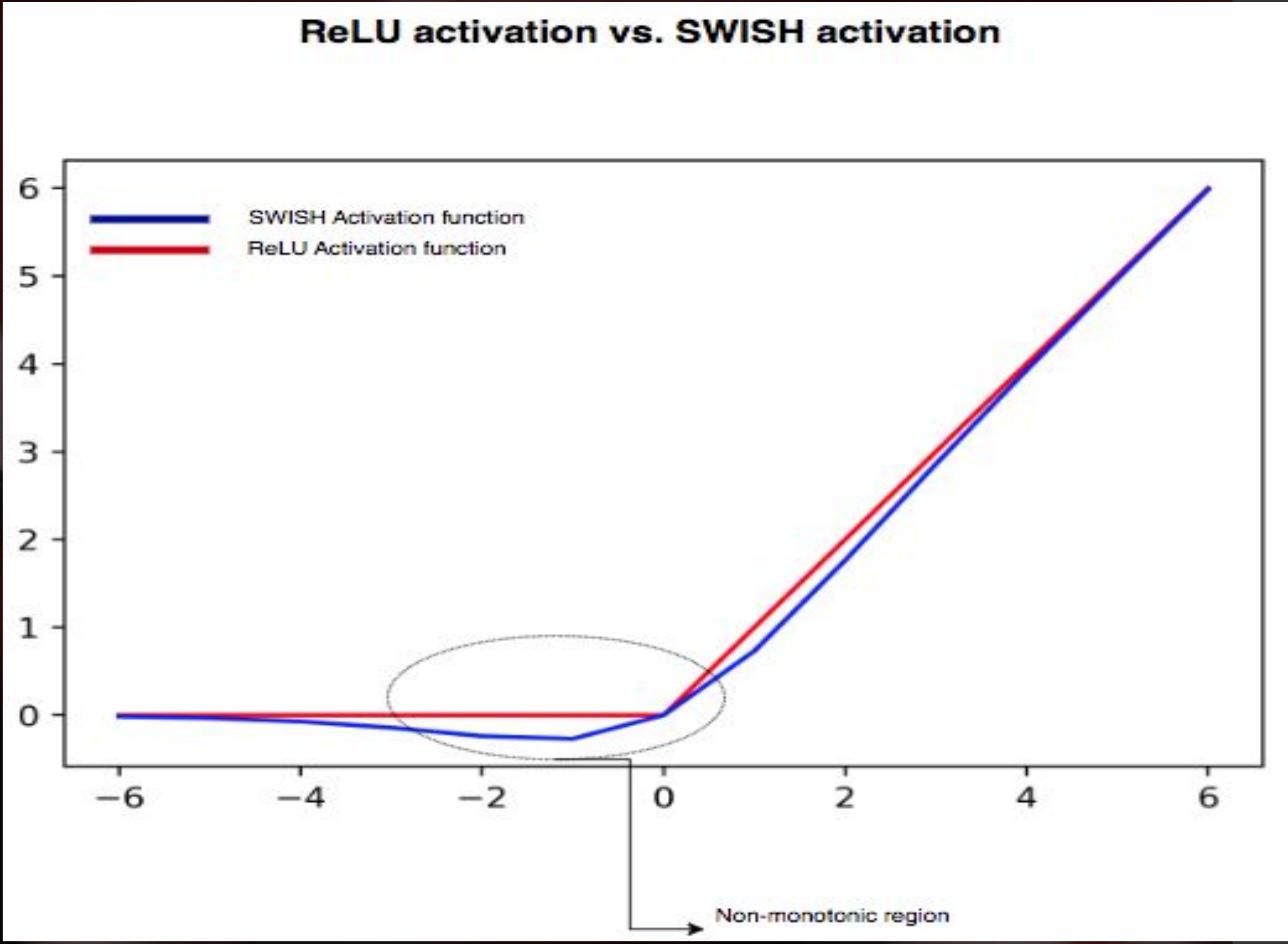
$$Swish(x) = \frac{x}{1+e^{-x}}$$

he Swish function, or the
elf-gated function, is just
nother activation function
roposed by **Google**. It has
een proposed as a possible
nprovisation to the already
xisting **sigmoid function**.



swish

$$Swish(x) = \frac{x}{1+e^{-x}}$$



swish

$$Swish(x) = \frac{x}{1 + e^{-x}}$$

Advantages of the Swish function

Here are some of the advantages of using a Swish function:

- The function **allows data normalization** and leads to quicker convergence and learning of the neural network.
- It works better in deep neural networks that require LSTM, compared to ReLU.
- With deeper neural networks requiring minor updates to the gradient during backpropagation, the update is not enough. This leads to the **vanishing gradient problem** in the case of the sigmoid and ReLU activation functions. Swish can work around and prevent the vanishing gradient program, and hence allow training for small gradient updates.

tanh

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

anh squashes a real-valued number

o the range [-1, 1].

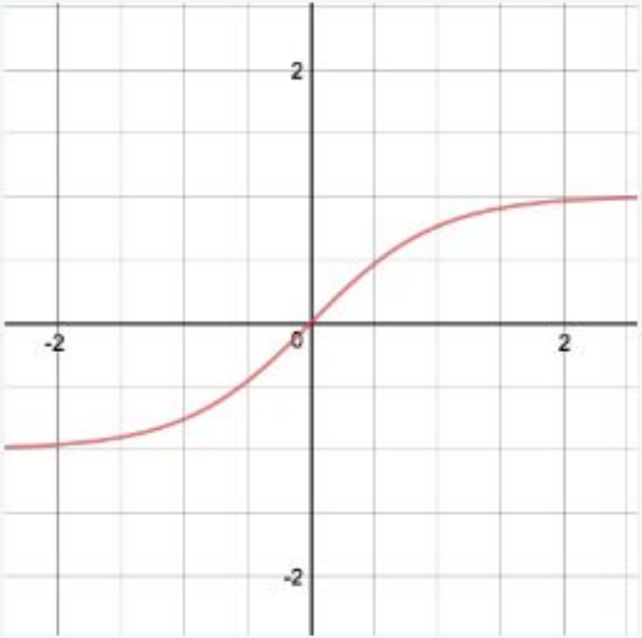
's **non-linear**. But unlike Sigmoid,

s output is zero-centered. Therefore,

practice the tanh non-linearity is

ways preferred to the sigmoid

onlinearity.




```
def tanh(z):
	return (np.exp(z) - np.exp(-z)) / (np.exp(z) + np.exp(-z))
```

tanh

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

Pros

- The gradient is **stronger** for **tanh** than **sigmoid** (derivatives are steeper).

Cons

- Tanh also has the vanishing gradient problem.

softmax

oftmax function calculates the **probabilities distribution** of the event over ‘n’ ifferent events.

n general way of saying, this function will calculate the **probabilities of each arget class over all possible target classes**. Later the calculated probabilities vill be helpful for determining the target class for the given inputs.

Built neural network from scratch

https://anderfernandez.com/blog/como-programar-una-red-neuronal-desde-0-en-python/

Challenge
maps

$ streamlit run maps.py

```
import pandas as pd
import numpy as np
import streamlit as st
```

Las 3 librerías que vamos a usar serán pandas para la manipulación de los datos; Streamlit para construir la aplicación Web y numpy para la generación de los puntos GPS aleatorios.

```
map_data = pd.DataFrame(
   np.random.randn(1000, 2) / [50, 50] + [37.76, -122.4],
   columns=['lat', 'lon'])
```

La primera sección que va a tener nuestra aplicación Web será la correspondiente al título y a un encabezado de nuestro sitio web.

```
# Create the title for the web app
st.title("San francisco Map")
st.header("Using Streamlit and Mapbox")
```

Posteriormente vamos a desplegar el conjunto de datos GPS para que el usuario pueda visualizar la información que vamos a pintar en el mapa.

```
st.map(map_data)
```

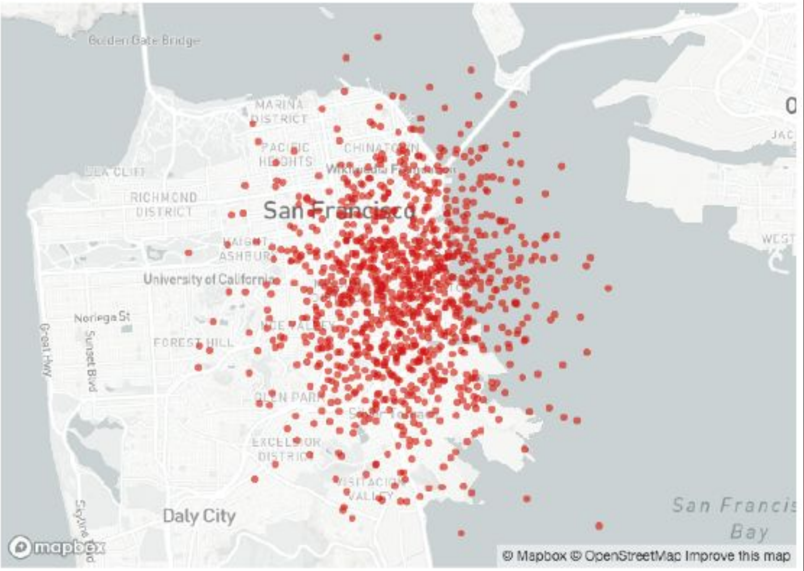
Challenge maps

- update def circulo (https://github.com/adsoftsito/tecnologias-construccion/blob/main/neural_network_numpy.py)

```
def circulo(num_datos = 100,R = 1, minimo = 0,maximo= 1):
  pi = math.pi
  r = R * np.sqrt(stats.truncnorm.rvs(minimo, maximo, size= num_datos)) * 10
  theta = stats.truncnorm.rvs(minimo, maximo, size= num_datos) * 2 * pi *10
  #print(len(r))
  #print(len(theta))
  x = np.cos(theta) * r
  y = np.sin(theta) * r

  y = y.reshape((num_datos,1))
  x = x.reshape((num_datos,1))

  #Vamos a reducir el numero de elementos para que no cause un Overflow
  x = np.round(x,3)
  y = np.round(y,3)

  df = np.column_stack([x,y])
  #print(df)
  return(df)
```

San francisco Map

Using Streamlit and Mapbox



Challenge maps

- generate 2 datasets
    - 150 city 1 (América)
    - 150 city 2 (Europa, Asia, Africa)
- Train neural network in numpy from scratch

Challenge maps

David vs Goliat

numpy

tensorflow/keras

- update https://github.com/adsoftsito/aiops/blob/main/linear.py
    - generate 2 datasets (150 city 1 - 150 city 2)
    - update keras layers
    - train neural network in keras
    - deploy to okteto cloud
    - try in postman

Challenge maps

- validate with excel (20 gps points) for 2 epochs (https://docs.google.com/spreadsheets/d/1GABf2jSFXd5rvPxJSTVoMK_ooPmykxpk1KZDDZ9_Dbg/edit?usp=sharing)

¡Buena suerte!

adsoft