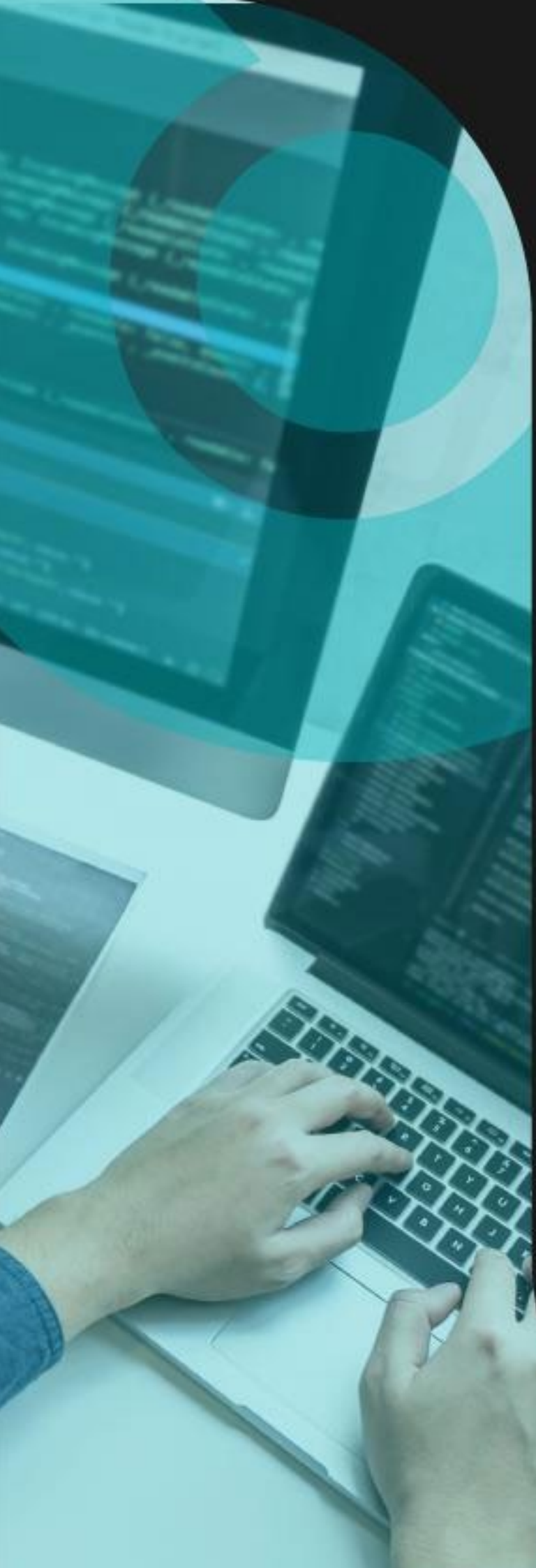


AWAKELAB

BASECAMP

Ciencia de Datos

Módulo: Fundamentos de deep learning.

Aprendizaje Esperado

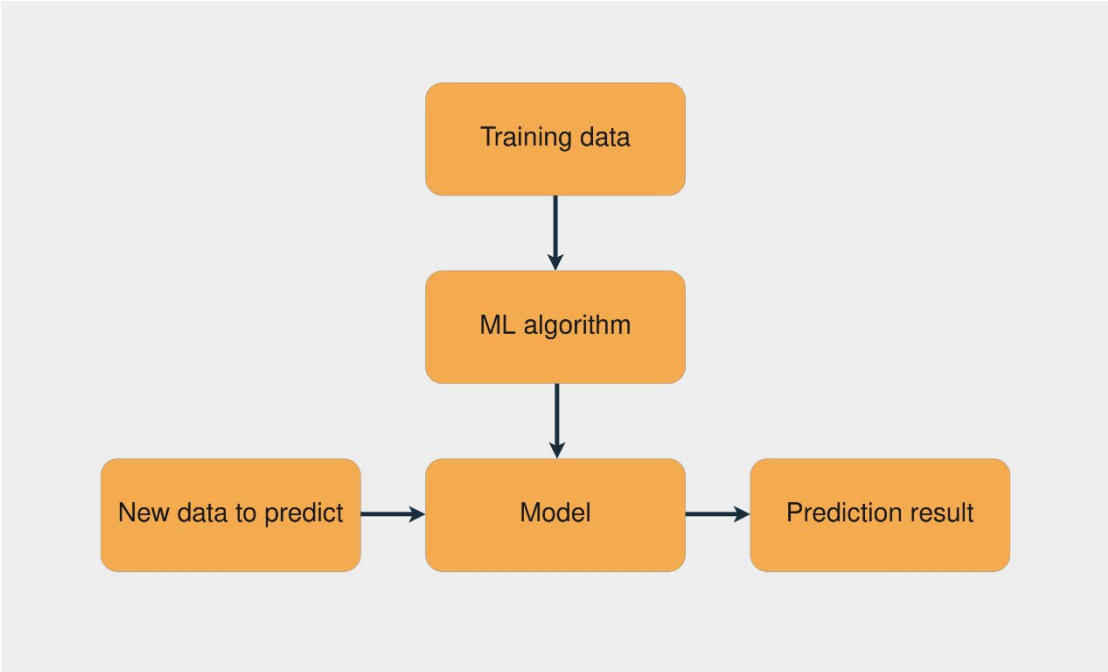
---

2. Elaborar un modelo predictivo utilizando redes neuronales en Python para resolver un problema de aprendizaje de máquina

---

Implementación de Redes Neuronales en Python

Una tarea común de aprendizaje automático es el aprendizaje supervisado, en el que se tiene un conjunto de datos con entradas y salidas conocidas. La tarea es usar este conjunto de datos para entrenar un modelo que prediga los resultados correctos en función de las entradas. La siguiente imagen presenta el flujo de trabajo para entrenar un modelo usando el aprendizaje supervisado:



Redes Neuronales: Conceptos Principales

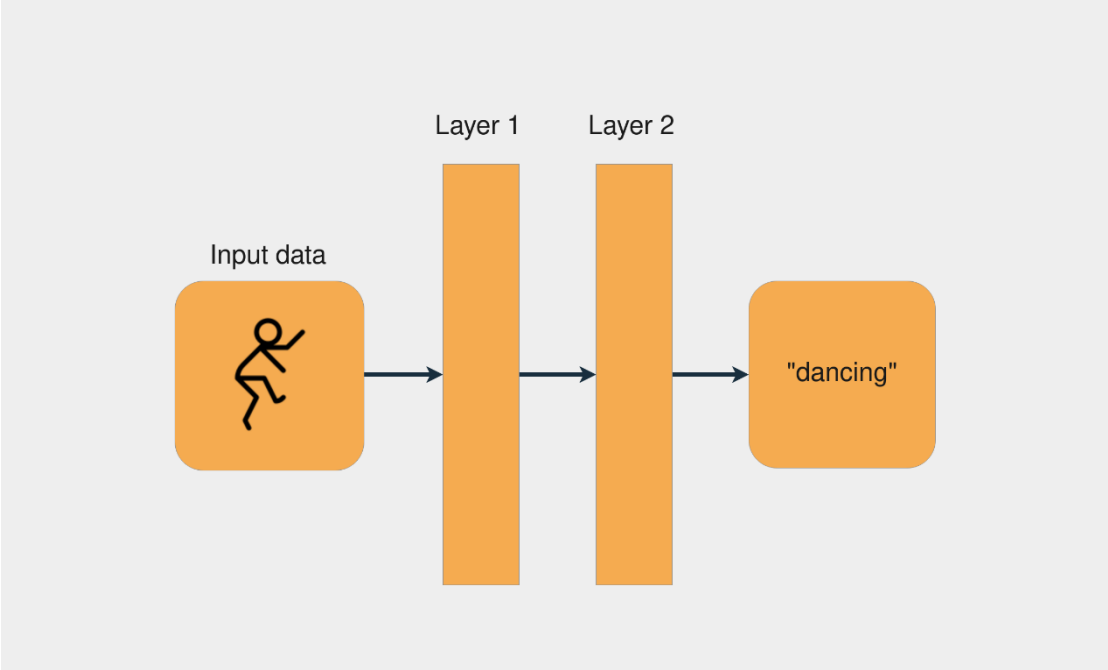
Una red neuronal es un sistema que aprende a hacer predicciones siguiendo estos pasos:

1. Tomando los datos de entrada
2. Haciendo una predicción
3. Comparación de la predicción con el resultado deseado
4. Ajustando su estado interno para predecir correctamente la próxima vez

Los vectores , **las capas** y **la regresión lineal** son algunos de los componentes básicos de las redes neuronales. Los datos se almacenan como vectores, y con Python almacena estos vectores en matrices. Cada capa transforma los datos que provienen de la capa anterior. Puede pensar en cada capa como un paso de ingeniería de características, porque cada capa extrae alguna representación de los datos que se obtuvieron anteriormente.

Una cosa interesante acerca de las capas de redes neuronales es que los mismos cálculos pueden extraer información de *cualquier* tipo de datos. Esto significa que no importa si está utilizando datos de imagen o datos de texto. El proceso para extraer información significativa y entrenar el modelo de aprendizaje profundo es el mismo para ambos escenarios.

En la imagen a continuación, puede ver un ejemplo de una arquitectura de red con dos capas:



Entorno en Python para redes neuronales

Existen dos formas para hacer esto:

Desde cero: este puede ser un buen ejercicio de aprendizaje, ya que le enseñará cómo funcionan las redes neuronales desde cero.

Uso de una biblioteca de redes neuronales: paquetes como Keras y TensorFlow simplifican la creación de redes neuronales al abstraer el código de bajo nivel. Si ya está familiarizado con el funcionamiento de las redes neuronales, esta es la forma más rápida y sencilla de crear una.

Independientemente del método que elija, trabajar con una red neuronal para hacer una predicción es esencialmente lo mismo:

1. Importar las bibliotecas . Por ejemplo: importar numpy como np
2. Definir/crear datos de entrada . Por ejemplo, use numpy para crear un conjunto de datos y una matriz de valores de datos.
3. Agregue pesos y sesgos (si corresponde) a las características de entrada. Estos son parámetros que se pueden aprender, lo que significa que se pueden ajustar durante el entrenamiento.
    - Pesos = parámetros de entrada que influyen en la salida

- Sesgo = un valor de umbral adicional agregado a la salida

4. Entrene la red con datos buenos y conocidos para encontrar los valores correctos para los pesos y sesgos.
5. Pruebe la red con un conjunto de datos de prueba para ver cómo funciona.
6. Ajuste el modelo con hiper parámetros (parámetros cuyos valores se utilizan para controlar el proceso de aprendizaje), calcule la precisión y realice una predicción.

Librería PyTorch

PyTorch es un marco con todas las funciones para crear modelos de aprendizaje profundo, que es un tipo de aprendizaje automático que se usa comúnmente en aplicaciones como el reconocimiento de imágenes y el procesamiento del lenguaje. Escrito en Python, es relativamente fácil de aprender y usar para la mayoría de los desarrolladores de aprendizaje automático. PyTorch se distingue por su excelente soporte para GPU y su uso de diferenciación automática de modo inverso, que permite modificar los gráficos de cálculo sobre la marcha. Esto lo convierte en una opción popular para la experimentación y creación de prototipos rápidos.

Para qué se utiliza PyTorch

Se sabe que el marco PyTorch es conveniente y flexible, con ejemplos que cubren el aprendizaje por refuerzo, la clasificación de imágenes y el procesamiento del lenguaje natural como los casos de uso más comunes .

Ejemplos de negocios, investigación y educación

- **Procesamiento del lenguaje natural (NLP):** desde Siri hasta Google Translate, las redes neuronales profundas han permitido avances en la comprensión del lenguaje natural por parte de las máquinas. La mayoría de estos modelos tratan el lenguaje como una secuencia plana de palabras o caracteres y utilizan un tipo de modelo llamado red neuronal recurrente (RNN) para procesar esta secuencia. Pero muchos lingüistas piensan que el lenguaje se entiende mejor como un árbol jerárquico de frases, por lo que se ha investigado mucho en modelos de aprendizaje profundo conocidos como redes neuronales recursivas que tienen en cuenta esta estructura. Si bien estos modelos son notoriamente difíciles de implementar e ineficientes de ejecutar, PyTorchhace que estos y otros modelos complejos de procesamiento de lenguaje natural sean mucho más fáciles. Salesforce está utilizando PyTorch para NLP y aprendizaje multitarea.
- **Investigación:** PyTorch es una opción popular para la investigación debido a su facilidad de uso, flexibilidad y creación rápida de prototipos. La Universidad de Stanford está utilizando la flexibilidad de PyTorch para investigar de manera eficiente nuevos enfoques algorítmicos.
- Educación: Udacity está educando a los innovadores de IA utilizando PyTorch.

Por qué PyTorch es importante para...

- **Los científicos de datos**

PyTorch es relativamente fácil de aprender para los programadores que están familiarizados con Python. Ofrece una depuración fácil, API simples y compatibilidad con una amplia gama de extensiones integradas de Python. Su modelo de ejecución dinámica también es excelente para la creación de prototipos, aunque extrae cierta sobrecarga de rendimiento.

- **Los desarrolladores de software**

PyTorch admiten una variedad de funciones que hacen que la implementación de los modelos de IA sea rápida y sencilla. También tiene un rico ecosistema de bibliotecas como Captum (para la interpretación de modelos), skorch (compatibilidad con scikit-learn), etc.

Instalación de Pytorch

```
!pip install torch


# Looking in indexes: https://pypi.org/simple,
https://us-python.pkg.dev/colab-wheels/public/simple/

# Requirement already satisfied: torch in
/usr/local/lib/python3.9/dist-packages (1.13.1+cu116)

# Requirement already satisfied: typing-extensions in
/usr/local/lib/python3.9/dist-packages (from torch)
(4.5.0)
```

```
import torch
```

Librería Keras

Keras es una API de aprendizaje profundo y redes neuronales de François Chollet que es capaz de ejecutarse sobre Tensorflow (Google), Theano o CNTK (Microsoft). Para citar el maravilloso libro de François Chollet, *Aprendizaje profundo con Python*:

Keras es una biblioteca a nivel de modelo que proporciona componentes básicos de alto nivel para desarrollar modelos de aprendizaje profundo. No maneja operaciones de bajo nivel como la manipulación y diferenciación de tensores. En cambio, se basa en una biblioteca de tensores especializada y bien optimizada para hacerlo, que actúa como el motor de fondo de Keras (Fuente).

¿Para qué se usa?

Es una excelente manera de comenzar a experimentar con redes neuronales sin tener que implementar cada capa y pieza por su cuenta. Por ejemplo, **Tensorflow** es una excelente biblioteca de aprendizaje automático, pero debe implementar una gran cantidad de código repetitivo para ejecutar un modelo.

¿Cómo se instala?

```
from tensorflow import keras
```

TensorFlow

TensorFlow fue desarrollado por Google y lanzado como código abierto en 2015. Surgió del software de aprendizaje automático de Google, que fue refactorizado y optimizado para su uso en producción.

El nombre "TensorFlow" describe cómo organizar y realizar operaciones en los datos. La estructura de datos básica para TensorFlow y PyTorch es un tensor. Cuando usa TensorFlow, realiza operaciones en los datos de estos tensores mediante la creación de un gráfico de flujo de datos con

estado, algo así como un diagrama de flujo que recuerda eventos pasados.

¿Para qué se usa?

Un `Session` objeto es una clase para ejecutar operaciones de TensorFlow. Contiene el entorno en el que `Tensor` se evalúan las `Operation` objetos y se ejecutan los objetos, y puede poseer recursos como tf.Variableobjetos. La forma más común de usar a `Session` es como administrador de contexto.

En TensorFlow 2.0, aún puede crear modelos de esta manera, pero es más fácil usar la ejecución ansiosa, que es la forma en que Python funciona normalmente. La ejecución ansiosa evalúa las operaciones de inmediato, por lo que puede escribir su código utilizando el flujo de control de Python en lugar del flujo de control de gráfico.

¿Cómo se instala?

```
!pip install tensorflow

import tensorflow as tf
```

Regresión con Redes Neuronales

Al modelar la relación entre las variables como lineal, puede expresar la variable dependiente como una **suma ponderada** de las variables independientes. Entonces, cada variable independiente se multiplicará por un vector llamado `weight`. Además de los pesos y las variables independientes, también agrega otro vector: el **sesgo** . Establece el resultado cuando todas las demás variables independientes son iguales a cero.

Como ejemplo del mundo real de cómo construir un modelo de regresión lineal, imagina que quieres entrenar un modelo para predecir el precio de las casas según el área y la antigüedad de la casa. Decides modelar esta relación usando una regresión lineal. El siguiente bloque de código muestra cómo puede escribir un modelo de regresión lineal para el problema indicado en pseudocódigo:

```
price = (weights_area * area) + (weights_age * age) + bias
```

En el ejemplo anterior, hay dos pesos: `weights_area` y `weights_age`. El proceso de entrenamiento consiste en ajustar los pesos y el sesgo para que el modelo pueda predecir el valor del precio correcto. Para lograrlo, deberá calcular el error de predicción y actualizar los pesos en consecuencia.

Estos son los conceptos básicos de cómo funciona el mecanismo de la red neuronal. Ahora es el momento de ver cómo aplicar estos conceptos usando Python.

Regresión Lineal con RN

Vamos a utilizar tres bibliotecas básicas para este modelo, numpy, matplotlib y TensorFlow.

- **Numpy:** agrega soporte para arreglos y matrices grandes y multidimensionales, junto con una gran colección de funciones matemáticas de alto nivel. En nuestro caso, vamos a generar datos con la ayuda de Numpy.
- **Matplotlib:** esta es una biblioteca de trazado para Python, visualizamos los resultados finales usando gráficos en Matplotlib.
- **Tensorflow:** esta biblioteca tiene un enfoque particular en el entrenamiento y la inferencia de redes neuronales

profundas. Podemos importar directamente las capas y entrenar, probar funciones sin tener que escribir todo el programa.

```
import numpy as np

import matplotlib.pyplot as plt

import tensorflow as tf
```

Generación de datos

Podemos generar nuestros propios datos numéricos para este proceso usando la función np.unifrom() que genera datos uniformes. Aquí, usamos dos variables de entrada xs y zs, agregamos algo de ruido para distribuir aleatoriamente los puntos de datos y, finalmente, la variable de destino se define como y=2*xs-3*zs+5+noise. El tamaño del conjunto de datos es 1000.

```
observations=1000

xs=np.random.uniform(-10,10,(observations,1))

zs=np.random.uniform(-10,10,(observations,1))

generated_inputs=np.column_stack((xs,zs))

noise=np.random.uniform(-10,10,(observations,1))

generated_target=2*xs-3*zs+5+noise
```

Después de generar los datos, guárdalos en un archivo .npz, para que pueda usarse para el entrenamiento.

```
np.savez('TF_intro',input=generated_inputs,targets=generat
ed_target)
```

```
training_data=np.load('TF_intro.npz')
```

Definición del modelo

Aquí, vamos a usar la capa densa de TensorFlow para hacer el modelo e importar el descenso de gradiente estocástico del optimizador de Keras.

Un gradiente es la pendiente de una función. Mide el grado en que una variable cambia con los cambios de otra variable. Matemáticamente, el descenso de gradiente es una función convexa cuya salida es la derivación parcial de un conjunto de parámetros de sus entradas. Cuanto mayor sea la pendiente, más empinada será la pendiente.

A partir de un valor inicial, Gradient Descent se ejecuta iterativamente para encontrar los valores de parámetro óptimos para encontrar el valor mínimo posible para la función de costo dada. La palabra "estocástico" se refiere a un sistema o proceso de probabilidad aleatoria. Por lo tanto, en Stochastic Gradient Descent, algunas muestras se seleccionan aleatoriamente, en lugar del conjunto de datos para cada iteración.

Dado que la entrada tiene 2 variables, tamaño de entrada = 2 y tamaño de salida = 1.

Establecimos la tasa de aprendizaje en 0,02, que no es ni demasiado alta ni demasiado baja, y el valor de época = 100.

```
input_size=2

output_size=1

models = tf.keras.Sequential([


tf.keras.layers.Dense(output_size)

                             ])
```

```
custom_optimizer=tf.keras.optimizers.SGD(learning_rate=0.02)

models.compile(optimizer=custom_optimizer,loss='mean_squared_error')

models.fit(training_data['input'],training_data['targets'],epochs=100,verbose=1)
```

Obtener pesos y sesgos

Podemos imprimir los valores predichos de pesos y sesgos y también almacenarlos.

```
models.layers[0].get_weights()

[array([[ 3.0473251],

        [-2.6085448]], dtype=float32), array([4.739199], dtype=float32)]
```

Aquí, la primera matriz representa los pesos y la segunda matriz representa los sesgos. Podemos observar claramente que los valores predichos de los pesos están muy cerca del valor real de los pesos.

```
weights=models.layers[0].get_weights()[0]

bias=models.layers[0].get_weights()[1]
```

Predicción y Precisión

Después de la predicción utilizando los pesos y sesgos dados, se obtiene una puntuación final de RMSE de 0,02866, que es bastante baja.

RMSE se define como el error cuadrático medio de la raíz. El error cuadrático medio toma la diferencia para cada valor observado y predicho. La fórmula para el error RMSE se da como:

$$RMSD = \sqrt{\frac{\sum_{i=1}^{N} (x_i - \hat{x}_i)^2}{N}}$$

```
out=training_data['targets'].round(1)

from sklearn.metrics import mean_squared_error

mean_squared_error(generated_target, out, squared=False)

0.0282032616762168
```

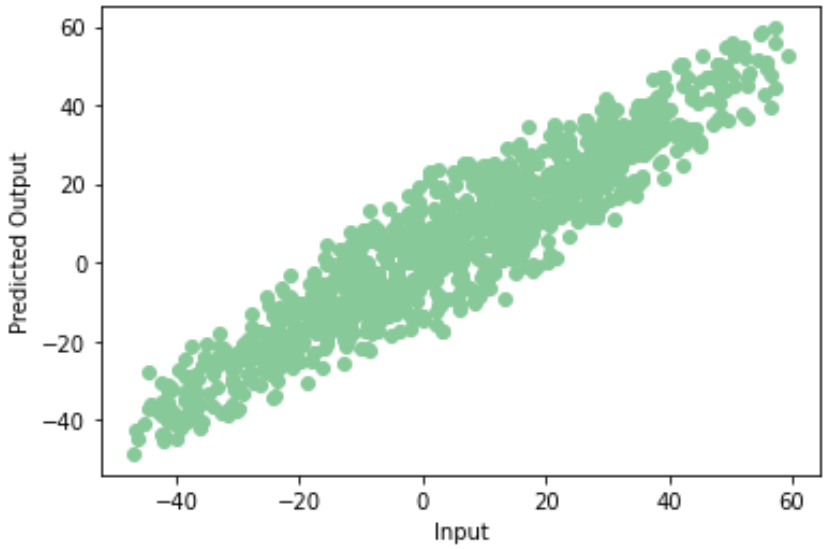
Si trazamos los datos pronosticados en un diagrama de dispersión, obtenemos un gráfico como este

```
plt.scatter(np.squeeze(models.predict_on_batch(training_da
ta['input'])),np.squeeze(training_data['targets']),c='#88c
999')

plt.xlabel('Input')

plt.ylabel('Predicted Output')

plt.show();
```



Nuestro modelo entrena correctamente con muy poco error. Este es el final de su primera red neuronal. Tenga en cuenta que cada vez que entrenamos el modelo podemos obtener un valor diferente de precisión, pero no diferirá mucho.

Clasificación con Redes Neuronales

Tipos de clasificación

- Suponga que desea predecir si una persona tiene diabetes o no. Si te enfrentas a este tipo de situación, hay dos posibilidades, ¿verdad? Eso se llama **clasificación binaria.**
- Suponga que desea identificar si una foto es de un juguete, una persona o un gato, ¿verdad? esto se llama **Clasificación Multiclase** porque hay más de dos opciones.
- Suponga que desea decidir qué categorías deben asignarse a un artículo. Si es así, se llama **Clasificación Multi-etiqueta** , porque un artículo podría tener más de una categoría asignada. Tomemos nuestra explicación a través de este artículo. Podemos asignar categorías como "Aprendizaje profundo, TensorFlow, Clasificación", etc. a este artículo.

Pasos en el modelado de redes neuronales para clasificación con Tensorflow

En TensorFlow hay etapas fijas para crear un modelo:

- **Creación de un modelo** : reúna las capas de una red neuronal utilizando la API funcional o secuencial
- **Compilar un modelo** : definir cómo se debe medir el rendimiento de un modelo y cómo se debe mejorar (función de pérdida y optimizador)
- **Ajuste de un modelo**: dejar que un modelo encuentre patrones en los datos

NOTA: Esta actividad se hará en conjunto con el grupo curso.

Actividad: Red neuronal para clasificación con Tensorflow

Características especiales

TensorFlow tiene una base de usuarios grande y bien establecida y una gran cantidad de herramientas para ayudar a producir aprendizaje automático. Para el desarrollo móvil, tiene API para JavaScript y Swift, y TensorFlow Lite le permite comprimir y optimizar modelos para dispositivos de Internet de las cosas.

Puede comenzar a usar TensorFlow rápidamente debido a la gran cantidad de datos, modelos preentrenados y **cuadernos de Google Colab** que proporcionan tanto Google como terceros.

Referencias

[1] Control de Python

https://realpython.com/

[2] Librería Keras

https://unipython.com/introduccion-y-como-instalar-keras-anaconda/#:~:text=Keras%20es%20una%20biblioteca%20de,la%20investigaci%C3%B3n%20como%20el%20desarrollo.

[3] Regresión con Redes Neuronales

https://sitiobigdata.com/2018/10/01/redes-neuronales-profundas-problemas-regresion/

[4] Tipo de redes neuronales

https://www.diegocalvo.es/clasificacion-de-redes-neuronales-artificiales/

[5] Clasificación con redes neuronales

https://scielo.isciii.es/scielo.php?script=sci_arttext&pid=S1578-84232015000100003

Material Complementario

[1] Regresión con redes neuronales

https://www.youtube.com/watch?v=u9Cg0jguzlQ

[2] Tipos de redes neuronales – Clasificación

https://www.youtube.com/watch?v=xj4MVkd1HXA


AWAKELAB