Introducción al Aprendizaje Automático (Machine Learning)

Ya que estamos aprendiendo `Python`, también es imprescindible hablar sobre aprendizaje automático (machine learning). Esto se debe a que muchas de sus bibliotecas están escritas en Python. Para empezar, vamos a instalarlas y jugar un poco con ellas.

TensorFlow

TensorFlow es una biblioteca de código abierto desarrollada por Google para el aprendizaje automático y la inteligencia artificial. Es ampliamente utilizada para construir y entrenar modelos de aprendizaje profundo, como redes neuronales, y es compatible con una variedad de lenguajes de programación, siendo Python el más común. TensorFlow ofrece una gran flexibilidad y escalabilidad, lo que lo convierte en una herramienta esencial para investigadores y desarrolladores en el campo de la inteligencia artificial.

Instálalo.

```
$ pip install tensorflow
ERROR: No se pudo encontrar una versión que satisfaga el requisito tensorflow
ERROR: No se encontró ninguna distribución compatible para tensorflow

$ type python
python es un alias de `/usr/local/Cellar/python@3.9/3.9.1_6/bin/python3'
```

Sin embargo, `Tensorflow 2` solo es compatible con `Python 3.5-3.8`. Nosotros estamos usando `3.9`.

```
% type python3
python3 es /usr/bin/python3
% python3 -V
Python 3.8.2
```

Noté que la versión de `python3` en mi sistema es `3.8.2`. ¿Dónde se instala el `pip` correspondiente a esta versión de Python?

```
% python3 -m pip -V
pip 21.0.1 from /Users/lzw/Library/Python/3.8/lib/python/site-packages/pip (python 3.8)
```

El `pip` correspondiente está aquí. Entonces voy a modificar el archivo `.zprofile`. Recientemente cambié mi `shell`. `.zprofile` es equivalente al antiguo `.bash_profile`. Agrego una línea.

```
alias pip3=/Users/lzw/Library/Python/3.8/bin/pip3
```

De esta manera, usamos `python3` y `pip3` para trabajar con `Tensorflow`.

```
% pip3 install tensorflow
...
Instalación exitosa de absl-py-0.12.0 astunparse-1.6.3 cachetools-4.2.1 certifi-2020.12.5 chardet-4.0.0
```

He instalado muchas bibliotecas. Utilicé un ejemplo de la página oficial.

```
import tensorflow as tf
```

mnist = tf.keras.datasets.mnist

```
(x_train, y_train), (x_test, y_test) = mnist.load_data()
x_train, x_test = x_train / 255.0, x_test / 255.0

model = tf.keras.models.Sequential([
  tf.keras.layers.Flatten(input_shape=(28, 28)),
  tf.keras.layers.Dense(128, activation='relu'),
  tf.keras.layers.Dropout(0.2),
  tf.keras.layers.Dense(10)
])

predictions = model(x_train[:1]).numpy()
print(predictions)
```

El texto que has proporcionado es solo un bloque de código vacío. Si necesitas traducir algo específico dentro de ese bloque o si hay más contenido que necesitas traducir, por favor proporciona más detalles. ¡Estoy aquí para ayudarte! 😊

Ejecútalo.

```
$ /usr/bin/python3 tf.py
Descargando datos desde https://storage.googleapis.com/tensorflow/tf-keras-datasets/mnist.npz
11493376/11490434 [==============================] - 10s 1us/paso
[[ 0.15477428 -0.3877643   0.0994779   0.07474922 -0.26219758 -0.03550266
   0.32226565 -0.37141111  0.10925996 -0.0115255 ]]
```

Se puede ver que se descargó el conjunto de datos y luego se generó el resultado.

A continuación, veamos un ejemplo de clasificación de imágenes.

```
# TensorFlow y tf.keras
import tensorflow as tf
```

Bibliotecas auxiliares

import numpy as np import matplotlib.pyplot as plt

```
print(tf.__version__)
```

Error.

```
ModuleNotFoundError: No module named 'matplotlib'
```

Nota: El mensaje de error no se traduce, ya que es un mensaje técnico en inglés que se utiliza comúnmente en el entorno de programación.

Instálalo.

```
% pip3 install matplotlib
```

Correcto.

```
$ /usr/bin/python3 image.py
2.4.1
```

Ejemplo de código para copiar y pegar.

```
# TensorFlow y tf.keras
import tensorflow as tf
```

Bibliotecas auxiliares

import numpy as np import matplotlib.pyplot as plt

```
fashion_mnist = tf.keras.datasets.fashion_mnist

(train_images, train_labels), (test_images, test_labels) = fashion_mnist.load_data()

class_names = ['Camiseta/top', 'Pantalón', 'Jersey', 'Vestido', 'Abrigo',
               'Sandalia', 'Camisa', 'Zapatilla', 'Bolso', 'Bota de tobillo']
print(train_images.shape)
print(len(train_labels))
```

Se generaron los resultados. Observa que aquí tenemos `train_images`, `train_labels`, `test_images`, `test_labels`. Esto significa que los datos están divididos en un conjunto de entrenamiento y un conjunto de prueba.

```
(60000, 28, 28)
60000
```

Nota: El texto dentro del bloque de código no se traduce, ya que es un formato específico que no debe ser alterado.

A continuación, intentemos imprimir la imagen.

```
print(train_images[0])
```

Mira los resultados.

```
[[  0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0
    0   0   0   0   0   0   0   0   0   0]
 [  0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0
    0   0   0   0   0   0   0   0   0   0]
 [  0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0
    0   0   0   0   0   0   0   0   0   0]
 [  0   0   0   0   0   0   0   0   0   0   0   0   1   0   0  13  73   0
    0   1   4   0   0   0   0   1   1   0]
 [  0   0   0   0   0   0   0   0   0   0   0   0   3   0  36 136 127  62
   54   0   0   0   1   3   4   0   0   3]
 [  0   0   0   0   0   0   0   0   0   0   0   0   6   0 102 204 176 134
  144 123  23   0   0   0   0  12  10   0]
 [  0   0   0   0   0   0   0   0   0   0   0   0   0   0 155 236 207 178
```

```
  107 156 161 109  64  23  77 130  72  15]
 [  0   0   0   0   0   0   0   0   0   0   0   1   0  69 207 223 218 216
  216 163 127 121 122 146 141  88 172  66]]
  ....
```

Aquí se presenta un extracto de los resultados.

```
print(len(train_images[0][0]))
```

Se imprime 28. Así que está claro que es una matriz con un ancho de 28. Continuemos imprimiendo.

```
    print(len(train_images[0][0][0])
TypeError: el objeto de tipo 'numpy.uint8' no tiene len()
```

Entonces, está bastante claro. Cada imagen es una matriz de 28*28*3. La última dimensión de la matriz almacena los valores de rgb. Sin embargo, nos dimos cuenta de que nuestra idea podría estar equivocada.

```
print(train_images[0][1][20])
```

```
0
```

```
print(train_images[0][1])
```

```
[0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0]
```

Cada imagen es una matriz de 28*28. Después de jugar un rato, finalmente descubrimos el secreto.

Primero, echemos un vistazo a la gráfica generada.

```
plt.figure()
plt.imshow(train_images[0])
plt.colorbar()
plt.grid(False)
plt.show()
```

¿Ves la barra de colores a la derecha? `0` a `250`. Originalmente, esto es un degradado entre dos colores. Pero, ¿cómo sabe cuáles son esos dos colores? ¿Dónde se lo dijimos?

Luego, también imprimimos la segunda imagen.

```
plt.imshow(train_images[1])
```

Es muy interesante. ¿Será esto algo predeterminado de las bibliotecas dependientes de `pyplot`? Sigamos ejecutando el código proporcionado en el sitio web oficial.

```
plt.figure(figsize=(10,10))
for i in range(25):
    plt.subplot(5,5,i+1)
    plt.xticks([])
    plt.yticks([])
    plt.grid(False)
    plt.imshow(train_images[i], cmap=plt.cm.binary)
    plt.xlabel(class_names[train_labels[i]])
plt.show()
```

Observa que aquí se muestran las imágenes junto con sus categorías. Finalmente, hemos descubierto el parámetro `cmp`. Si no escribimos nada en `cmp`, definitivamente obtendremos el mismo esquema de colores que teníamos antes. Efectivamente.

```
    plt.imshow(train_images[i])
```

En esta ocasión, buscamos `pyplot cmap`. Encontramos algunos recursos.

```
    plt.imshow(train_images[i], cmap=plt.cm.PiYG)
```

Modifica el código.

```
plt.figure(figsize=(10,10))
for i in range(25):
    plt.subplot(5,5,i+1)    ## Cambia esta línea
    plt.xticks([])
    plt.yticks([])
    plt.grid(False)
```

```
    plt.imshow(train_images[i], cmap=plt.cm.Blues)
    plt.xlabel(class_names[train_labels[i]])
plt.show()
```

Sin embargo, ocurrió un error.

```
ValueError: num debe ser 1 <= num <= 10, no 11
```

¿Qué significa esto? ¿Qué significa exactamente el `5,5,i+1` anterior? ¿Por qué no funciona cuando se cambia a `2`? Aunque intuitivamente sabemos que probablemente significa 5 filas y 5 columnas, ¿por qué se produce este error? ¿Cómo se calcula el `11`? ¿Qué significa `num`? ¿Qué significa el `10`? Notamos que `2*5=10`. Entonces, tal vez el error ocurre cuando `i=11`. Cuando se cambia a `for i in range(10):`, se obtienen los siguientes resultados.

Esto es un vistazo rápido a la documentación, donde aprendemos que `subplot(nrows, ncols, index, **kwargs)`. Mmm, hasta aquí lo tenemos bastante claro.

```
plt.figure(figsize=(10,10))
for i in range(25):
    plt.subplot(5,5,i+1)
    # plt.xticks([])
    plt.yticks([])
    plt.grid(False)
    plt.imshow(train_images[i], cmap=plt.cm.Blues)
    plt.xlabel(class_names[train_labels[i]])
plt.show()
```

Nota que `0 25` se llama `xticks`. Cuando ampliamos o reducimos este cuadro, se mostrará de manera diferente.

plot_scale

Observa que al hacer zoom o reducir el cuadro, los `xticks` y `xlabels` se mostrarán de manera diferente.

```
model = tf.keras.Sequential([
    tf.keras.layers.Flatten(input_shape=(28, 28)),
    tf.keras.layers.Dense(128, activation='relu'),
    tf.keras.layers.Dense(10)
])
```

```
model.compile(optimizer='adam',
              loss=tf.keras.losses.SparseCategoricalCrossentropy(from_logits=True),
              metrics=['accuracy'])
```

model.fit(train_images, train_labels, epochs=10)

test_loss, test_acc = model.evaluate(test_images, test_labels, verbose=2)

```
print('\nPrecisión en la prueba:', test_acc)
```

Observa la forma en que se define el modelo aquí, utilizando la clase `Sequential`. Presta atención a estos parámetros: `28,28`, `128`, `relu`, `10`. Nota que es necesario `compile` y `fit`. `fit` significa ajustar o entrenar el modelo. Observa que `28,28` corresponde al tamaño de la imagen.

```
Época 1/10
1875/1875 [==============================] - 2s 928us/paso - pérdida: 0.6331 - precisión: 0.7769
Época 2/10
1875/1875 [==============================] - 2s 961us/paso - pérdida: 0.3860 - precisión: 0.8615
Época 3/10
1875/1875 [==============================] - 2s 930us/paso - pérdida: 0.3395 - precisión: 0.8755
Época 4/10
1875/1875 [==============================] - 2s 1ms/paso - pérdida: 0.3071 - precisión: 0.8890
Época 5/10
1875/1875 [==============================] - 2s 1ms/paso - pérdida: 0.2964 - precisión: 0.8927
Época 6/10
1875/1875 [==============================] - 2s 985us/paso - pérdida: 0.2764 - precisión: 0.8955
Época 7/10
1875/1875 [==============================] - 2s 961us/paso - pérdida: 0.2653 - precisión: 0.8996
Época 8/10
1875/1875 [==============================] - 2s 1ms/paso - pérdida: 0.2549 - precisión: 0.9052
Época 9/10
1875/1875 [==============================] - 2s 1ms/paso - pérdida: 0.2416 - precisión: 0.9090
Época 10/10
1875/1875 [==============================] - 2s 1ms/paso - pérdida: 0.2372 - precisión: 0.9086
313/313 - 0s - pérdida: 0.3422 - precisión: 0.8798
```

Precisión de la prueba: 0.879800021648407

El modelo ya ha sido entrenado. Vamos a ajustar los parámetros.

```shell
model = tf.keras.Sequential([
    tf.keras.layers.Flatten(input_shape=(28, 28)),
    tf.keras.layers.Dense(28, activation='relu'),    # 128 -> 28
    tf.keras.layers.Dense(10)
])
```

Modifica el primer parámetro de `Dense`.

```
Época 1/10
1875/1875 [==============================] - 2s 714us/paso - pérdida: 6.9774 - precisión: 0.3294
Época 2/10
1875/1875 [==============================] - 1s 715us/paso - pérdida: 1.3038 - precisión: 0.4831
Época 3/10
1875/1875 [==============================] - 1s 747us/paso - pérdida: 1.0160 - precisión: 0.6197
Época 4/10
1875/1875 [==============================] - 1s 800us/paso - pérdida: 0.7963 - precisión: 0.6939
Época 5/10
1875/1875 [==============================] - 2s 893us/paso - pérdida: 0.7006 - precisión: 0.7183
Época 6/10
1875/1875 [==============================] - 1s 747us/paso - pérdida: 0.6675 - precisión: 0.7299
Época 7/10
1875/1875 [==============================] - 1s 694us/paso - pérdida: 0.6681 - precisión: 0.7330
Época 8/10
1875/1875 [==============================] - 1s 702us/paso - pérdida: 0.6675 - precisión: 0.7356
Época 9/10
1875/1875 [==============================] - 1s 778us/paso - pérdida: 0.6508 - precisión: 0.7363
Época 10/10
1875/1875 [==============================] - 1s 732us/paso - pérdida: 0.6532 - precisión: 0.7350
313/313 - 0s - pérdida: 0.6816 - precisión: 0.7230
```

Precisión de la prueba: 0.7229999899864197

Se observa un cambio en la `Test accuracy` antes y después. `Epoch` es un registro de salida de la funci

```python
print(train_labels)
[9 0 0 ... 3 0 5]
print(len(train_labels))
60000
```

Esto significa que se utilizan los números del `0 al 9` para representar estas categorías. Coincidentemente, `class_names` también tiene 10 elementos.

```
class_names = ['Camiseta', 'Pantalón', 'Jersey', 'Vestido', 'Abrigo',
               'Sandalia', 'Camisa', 'Zapatilla', 'Bolso', 'Bota tobillera']
```

Vamos a hacer algunos cambios más.

```
model = tf.keras.Sequential([
    tf.keras.layers.Flatten(input_shape=(28, 28)),
    tf.keras.layers.Dense(28, activation='relu'),
    tf.keras.layers.Dense(5)   # 10 -> 5
])

model.compile(optimizer='adam',
              loss=tf.keras.losses.SparseCategoricalCrossentropy(from_logits=True),
              metrics=['accuracy'])

model.fit(train_images, train_labels, epochs=10)
```

Se produjo un error.

```
tensorflow.python.framework.errors_impl.InvalidArgumentError: Se recibió un valor de etiqueta de 9 que
         [[nodo sparse_categorical_crossentropy/SparseSoftmaxCrossEntropyWithLogits/SparseSoftmaxCrossE
```

Pila de llamadas de funciones: train_function

```
Cambia el tercer parámetro de `Sequential`, que es `Dense`, a `15` y eso debería funcionar. La diferenc
```

```python
model = tf.keras.Sequential([
    tf.keras.layers.Flatten(input_shape=(28, 28)),
```

```
    tf.keras.layers.Dense(28, activation='relu'),
    tf.keras.layers.Dense(15)
])

model.compile(optimizer='adam',
              loss=tf.keras.losses.SparseCategoricalCrossentropy(from_logits=True),
              metrics=['accuracy'])
```

En este código, el modelo se compila utilizando el optimizador 'adam', la función de pérdida `SparseCategoricalCrossentropy` (con `from_logits=True`) y se mide la precisión (`accuracy`) como métrica.

model.fit(train_images, train_labels, epochs=15) # 10 -> 15

test_loss, test_acc = model.evaluate(test_images, test_labels, verbose=2)

```
print('\nPrecisión en la prueba:', test_acc)
```

```
Época 1/15
1875/1875 [==============================] - 2s 892us/paso - pérdida: 6.5778 - precisión: 0.3771
Época 2/15
1875/1875 [==============================] - 2s 872us/paso - pérdida: 1.3121 - precisión: 0.4910
Época 3/15
1875/1875 [==============================] - 2s 909us/paso - pérdida: 1.0900 - precisión: 0.5389
Época 4/15
1875/1875 [==============================] - 1s 730us/paso - pérdida: 1.0422 - precisión: 0.5577
Época 5/15
1875/1875 [==============================] - 1s 709us/paso - pérdida: 0.9529 - precisión: 0.5952
Época 6/15
1875/1875 [==============================] - 1s 714us/paso - pérdida: 0.9888 - precisión: 0.5950
Época 7/15
1875/1875 [==============================] - 1s 767us/paso - pérdida: 0.8678 - precisión: 0.6355
Época 8/15
1875/1875 [==============================] - 1s 715us/paso - pérdida: 0.8247 - precisión: 0.6611
Época 9/15
1875/1875 [==============================] - 1s 721us/paso - pérdida: 0.8011 - precisión: 0.6626
Época 10/15
1875/1875 [==============================] - 1s 711us/paso - pérdida: 0.8024 - precisión: 0.6622
```

```
Época 11/15
1875/1875 [==============================] - 1s 781us/paso - pérdida: 0.7777 - precisión: 0.6696
Época 12/15
1875/1875 [==============================] - 1s 724us/paso - pérdida: 0.7764 - precisión: 0.6728
Época 13/15
1875/1875 [==============================] - 1s 731us/paso - pérdida: 0.7688 - precisión: 0.6767
Época 14/15
1875/1875 [==============================] - 1s 715us/paso - pérdida: 0.7592 - precisión: 0.6793
Época 15/15
1875/1875 [==============================] - 1s 786us/paso - pérdida: 0.7526 - precisión: 0.6792
313/313 - 0s - pérdida: 0.8555 - precisión: 0.6418
```

Precisión de la prueba: 0.6417999863624573

Nota: Cambié a 15. La diferencia no es grande. `tf.keras.layers.Dense(88, activation='relu'),` es impor

```python
probability_model = tf.keras.Sequential([model,
                                         tf.keras.layers.Softmax()])
```

A continuación, vamos a hacer una predicción. Observa que `Sequential` es igual que antes. Fíjate en los parámetros `model` y `tf.keras.layers.Softmax()`.

```
probability_model = tf.keras.Sequential([model,
                                         tf.keras.layers.Softmax()])
predictions = probability_model.predict(test_images)

def plot_image(i, predictions_array, true_label, img):
  true_label, img = true_label[i], img[i]
  plt.grid(False)
  plt.xticks([])
  plt.yticks([])

plt.imshow(img, cmap=plt.cm.binary)

predicted_label = np.argmax(predictions_array)
if predicted_label == true_label:
```

```
    color = 'blue'
else:
    color = 'red'

plt.xlabel("{} {:2.0f}% ({})".format(class_names[predicted_label],
                                     100*np.max(predictions_array),
                                     class_names[true_label]),
                                     color=color)
```

Traducción:

```
plt.xlabel("{} {:2.0f}% ({})".format(class_names[predicted_label],
                                     100*np.max(predictions_array),
                                     class_names[true_label]),
                                     color=color)
```

En este caso, el código no necesita ser traducido, ya que es una línea de código en Python que utiliza variables y funciones específicas. Sin embargo, si deseas una explicación en español, sería algo como:

```
plt.xlabel("{} {:2.0f}% ({})".format(class_names[predicted_label],
                                     100*np.max(predictions_array),
                                     class_names[true_label]),
                                     color=color)
```

Aquí, `plt.xlabel` está configurando la etiqueta del eje x de un gráfico. La etiqueta se forma utilizando el nombre de la clase predicha (`class_names[predicted_label]`), el porcentaje de confianza de la predicción (`100*np.max(predictions_array)`), y el nombre de la clase real (`class_names[true_label]`). El color de la etiqueta se establece con el parámetro `color`.

```
def plot_value_array(i, predictions_array, true_label):
  true_label = true_label[i]
  plt.grid(False)
  plt.xticks(range(10))
  plt.yticks([])
  thisplot = plt.bar(range(10), predictions_array, color="#777777")
  plt.ylim([0, 1])
  predicted_label = np.argmax(predictions_array)
```

```
thisplot[predicted_label].set_color('rojo')
thisplot[true_label].set_color('azul')

i = 0
plt.figure(figsize=(6,3))
plt.subplot(1,2,1)
plot_image(i, predictions[i], test_labels, test_images)
plt.subplot(1,2,2)
plot_value_array(i, predictions[i],  test_labels)
plt.show()
```

Esto indica que hay un 99% de probabilidad de que la imagen sea una `Ankle boot`. Observa que `plot_image` muestra la imagen de la izquierda, mientras que `plot_value_array` genera el gráfico de la derecha.

```
num_rows = 5
num_cols = 3
num_images = num_rows*num_cols
plt.figure(figsize=(2*2*num_cols, 2*num_rows))
for i in range(num_images):
  plt.subplot(num_rows, 2*num_cols, 2*i+1)
  plot_image(i, predictions[i], test_labels, test_images)
  plt.subplot(num_rows, 2*num_cols, 2*i+2)
  plot_value_array(i, predictions[i], test_labels)
plt.tight_layout()
plt.show()
```

Observa que aquí solo se muestran más resultados de pruebas. Por lo tanto, el flujo de uso lo tenemos bastante claro. Sin embargo, aún no sabemos cómo se calcula detrás de escena. Pero sabemos cómo usarlos. Detrás de ellos está el cálculo. ¿Cómo entender el cálculo?

Por ejemplo, hay un número entre 1 y 100 que tienes que adivinar. Cada vez que adivinas, te digo si es más pequeño o más grande. Adivinas 50. Te digo que es más pequeño. Adivinas 80. Te digo que es más grande. Adivinas 65. Te digo que es más grande. Adivinas 55. Te digo que es más pequeño. Adivinas 58. Te digo, "¡Sí, has adivinado correctamente!"

El aprendizaje automático, en esencia, simula un proceso similar en segundo plano. Solo que es un poco más complejo. Podría haber muchos números entre `1 y 100`, y se deben adivinar

muchos números. Además, cada vez que se adivina, se realizan muchos cálculos. Y cada vez que se determina si el número es mayor o menor, también se realizan muchos cálculos.

PyTorch

Instálalo. Esto es compatible con Python versión `3.9`.

```
$ pip install torch torchvision
Collecting torch
  Descargando torch-1.8.0-cp39-none-macosx_10_9_x86_64.whl (120.6 MB)
     |                  | 120.6 MB 224 kB/s
Collecting torchvision
  Descargando torchvision-0.9.0-cp39-cp39-macosx_10_9_x86_64.whl (13.1 MB)
     |                  | 13.1 MB 549 kB/s
Requirement already satisfied: numpy in /usr/local/lib/python3.9/site-packages (from torch) (1.20.1)
Collecting typing-extensions
  Descargando typing_extensions-3.7.4.3-py3-none-any.whl (22 kB)
Requirement already satisfied: pillow>=4.1.1 in /usr/local/lib/python3.9/site-packages (from torchvisio
Installing collected packages: typing-extensions, torch, torchvision
Successfully installed torch-1.8.0 torchvision-0.9.0 typing-extensions-3.7.4.3
```

Verifícalo.

```
import torch
x = torch.rand(5, 3)
print(x)
```

Se produjo un error.

```
Traceback (most recent call last):
  File "torch.py", line 1, in <module>
    import torch
  File "torch.py", line 2, in <module>
    x = torch.rand(5, 3)
AttributeError: el módulo 'torch' está parcialmente inicializado y no tiene el atributo 'rand' (probabl
```

Busqué el mensaje de error en Google. Resulta que el problema era porque nuestro archivo también se llamaba `torch`. Había un conflicto de nombres. Lo cambié y luego funcionó correctamente.

```
tensor([[0.5520, 0.9446, 0.5543],
        [0.6192, 0.0908, 0.8726],
        [0.0223, 0.7685, 0.9814],
        [0.4019, 0.5406, 0.3861],
        [0.5485, 0.6040, 0.2387]])
```

Encontré un ejemplo.

```
# -*- coding: utf-8 -*-

import torch
import math
dtype = torch.float
device = torch.device("cpu")
# device = torch.device("cuda:0") # Descomenta esto para ejecutar en GPU
```

Crear datos de entrada y salida aleatorios

x = torch.linspace(-math.pi, math.pi, 2000, device=device, dtype=dtype) y = torch.sin(x)

Inicializar pesos aleatoriamente

a = torch.randn((), device=device, dtype=dtype) b = torch.randn((), device=device, dtype=dtype) c = torch.randn((), device=device, dtype=dtype) d = torch.randn((), device=device, dtype=dtype)

```
learning_rate = 1e-6
for t in range(2000):
    # Paso hacia adelante: calcular la predicción y
    y_pred = a + b * x + c * x ** 2 + d * x ** 3

# Calcular e imprimir la pérdida
loss = (y_pred - y).pow(2).sum().item()
if t % 100 == 99:
    print(t, loss)
```

```
# Retropropagación para calcular los gradientes de a, b, c, d con respecto a la pérdida
grad_y_pred = 2.0 * (y_pred - y)
grad_a = grad_y_pred.sum()
grad_b = (grad_y_pred * x).sum()
grad_c = (grad_y_pred * x ** 2).sum()
grad_d = (grad_y_pred * x ** 3).sum()

# Actualizar pesos usando el descenso de gradiente
a -= learning_rate * grad_a
b -= learning_rate * grad_b
c -= learning_rate * grad_c
d -= learning_rate * grad_d
```

print(f'Resultado: y = {a.item()} + {b.item()} x + {c.item()} x^2 + {d.item()} x^3')

Ejecútalo.

```shell
99 1273.537353515625
199 849.24853515625
299 567.4786987304688
399 380.30291748046875
499 255.92752075195312
599 173.2559814453125
699 118.2861328125
799 81.72274780273438
899 57.39331817626953
999 41.198158264160156
1099 30.41307830810547
1199 23.227672576904297
1299 18.438262939453125
1399 15.244369506835938
1499 13.113286972045898
1599 11.690631866455078
1699 10.740333557128906
1799 10.105220794677734
```

```
1899 9.6804780960083
1999 9.39621353149414
Resultado: y = -0.011828352697193623 + 0.8360244631767273 x + 0.002040589228272438 x^2 + -0.09038365632
```

Aquí tienes el código utilizando solo la biblioteca `numpy`:

```
import numpy as np

# Crear un array de ejemplo
array = np.array([1, 2, 3, 4, 5])

# Realizar operaciones básicas
suma = np.sum(array)
media = np.mean(array)
maximo = np.max(array)

print("Suma:", suma)
print("Media:", media)
print("Máximo:", maximo)
```

Este código crea un array con `numpy`, realiza algunas operaciones básicas como la suma, la media y el valor máximo, y luego imprime los resultados.

```
# -*- coding: utf-8 -*-
import numpy as np
import math
```

Crear datos de entrada y salida aleatorios

x = np.linspace(-math.pi, math.pi, 2000) y = np.sin(x)

Inicializar pesos aleatoriamente

a = np.random.randn() b = np.random.randn() c = np.random.randn() d = np.random.randn()

```
learning_rate = 1e-6
for t in range(2000):
    # Paso hacia adelante: calcular la y predicha
    # y = a + b x + c x^2 + d x^3
    y_pred = a + b * x + c * x ** 2 + d * x ** 3

# Calcular e imprimir la pérdida
loss = np.square(y_pred - y).sum()
if t % 100 == 99:
    print(t, loss)

# Retropropagación para calcular los gradientes de a, b, c, d con respecto a la pérdida
grad_y_pred = 2.0 * (y_pred - y)
grad_a = grad_y_pred.sum()
grad_b = (grad_y_pred * x).sum()
grad_c = (grad_y_pred * x ** 2).sum()
grad_d = (grad_y_pred * x ** 3).sum()

# Actualizar pesos
a -= learning_rate * grad_a
b -= learning_rate * grad_b
c -= learning_rate * grad_c
d -= learning_rate * grad_d

print(f'Resultado: y = {a} + {b} x + {c} x^2 + {d} x^3')
```

Observa que hay dos formas de calcularlo.

En estos dos ejemplos, primero se genera un conjunto de valores para `x` e `y`. Luego, se asume que se trata de una ecuación cúbica. A continuación, se utilizan algunos métodos para calcular iterativamente los coeficientes. ¿Cómo son estos algoritmos? Observa que se realiza un ciclo de `2000` iteraciones, y en cada una de ellas se ajusta con mayor precisión. Por ahora, no entraremos en detalles sobre esto.

Finalmente

Actualmente, no entendemos cómo se realizan los cálculos detrás del aprendizaje automático. Sin embargo, por el momento, eso no es importante. Con el conocimiento similar al que hemos

mencionado anteriormente, ya podemos hacer muchas cosas. También podemos utilizar el aprendizaje automático para procesar texto, audio, etc. Cuando hayamos explorado docenas de ejemplos, será el momento adecuado para aprender los principios.

Práctica

- Los estudiantes exploran de la manera mencionada anteriormente.