機械学習入門

私たちが `Python` を学ぶのであれば、機械学習についてもきっと話さなければなりません。なぜなら、その多くのライブラリが Python で書かれているからです。まずはそれらをインストールして、少し遊んでみましょう。

Tensorflow

インストールしてみましょう。

```
$ pip install tensorflow
ERROR: Could not find a version that satisfies the requirement tensorflow
ERROR: No matching distribution found for tensorflow
```

このエラーメッセージは、指定されたバージョンの TensorFlow が見つからなかったことを示しています。この問題は、Python のバージョンが TensorFlow と互換性がない場合や、利用可能な TensorFlow のバージョンが存在しない場合に発生することがあります。解決策として、Python のバージョンを確認し、TensorFlow がサポートしているバージョンに合わせるか、または利用可能な TensorFlow のバージョンを指定してインストールを試みることができます。

```
$ type python
python は `/usr/local/Cellar/python@3.9/3.9.1_6/bin/python3' のエイリアスです
```

しかし、`Tensorflow 2` は `Python 3.5–3.8` のみをサポートしています。私たちが使用しているのは `3.9` です。

```
% type python3
python3 は /usr/bin/python3 です
% python3 -V
Python 3.8.2
```

私のシステム内の `python3` は `3.8.2` バージョンであることに気づきました。この Python バージョンに対応する `pip` はどこにインストールされているのでしょうか。

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

このコマンドは、Python のパッケージ管理ツールである pip のバージョンと、それがインストールされている場所を確認するものです。出力結果から、pip のバージョンが 21.0.1 であり、Python 3.8 の環境で使用されていることがわかります。また、pip が/Users/lzw/Library/Python/3.8/lib/python/site-packages/pip にインストールされていることも示されています。

対応する pip はここにあります。それでは、.zprofile ファイルを変更します。最近、私は shell を変更しました。.zprofile は以前の.bash_profile に相当します。1 行追加します。

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

このコードは、pip3 コマンドに対してエイリアスを設定しています。具体的には、pip3 を実行すると、指定されたパス/Users/lzw/Library/Python/3.8/bin/pip3 にある pip3 が実行されるようになります。これにより、システム全体の pip3 ではなく、特定のユーザー環境にインストールされた pip3 を使用することができます。

これで、python3 と pip3 を使って Tensorflow を楽しむことができます。

```
% pip3 install tensorflow
...
成功インストール: absl-py-0.12.0 astunparse-1.6.3 cachetools-4.2.1 certifi-2020.12.5 chardet-4.0.0 flatbu
```

多くのライブラリをインストールしました。公式サイトの例を使ってみました。

```
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
```

このコードは、MNIST データセットを読み込み、訓練データとテストデータを正規化するためのものです。具体的には、x_train と x_test の各ピクセル値を 255 で割ることで、0 から 1 の範囲にスケーリングしています。これにより、ニューラルネットワークの学習がより効率的になります。

```
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)
```

あなたはプロの翻訳者です。Jekyll ブログ投稿用のマークダウンファイルを翻訳しています。以下のテキスト

実行してみましょう。

```
$ /usr/bin/python3 tf.py
データをダウンロード中: https://storage.googleapis.com/tensorflow/tf-keras-datasets/mnist.npz
11493376/11490434 [==============================] - 10秒 1マイクロ秒/ステップ
[[ 0.15477428 -0.3877643   0.0994779   0.07474922 -0.26219758 -0.03550266
   0.32226565 -0.37141111  0.10925996 -0.0115255 ]]
```

データセットがダウンロードされ、結果が出力されたことが確認できます。

次に、画像分類の例を見てみましょう。

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

ヘルパーライブラリ

import numpy as np import matplotlib.pyplot as plt

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

エラーが発生しました。

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

このエラーメッセージは、Python が matplotlib というモジュールを見つけられなかったことを示しています。matplotlib はデータの可視化に使用される人気のあるライブラリです。このエラーを解決するには、以下の手順で matplotlib をインストールしてください。

1. ターミナルまたはコマンドプロンプトを開きます。
2. 以下のコマンドを実行して `matplotlib` をインストールします。

```
pip install matplotlib
```

これで `matplotlib` がインストールされ、エラーが解消されるはずです。もし `pip` がインストールされていない場合や、他の問題が発生した場合は、Python 環境の設定を確認してください。

インストールしてみましょう。

```
% pip3 install matplotlib
```

正解です。

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

このコマンドは、Python 3 を使用して `image.py` というスクリプトを実行し、その結果として `2.4.1` というバージョン情報を出力しています。

以下はコピー＆ペーストの例コードです。

```
# コピー＆ペーストの例
text_to_copy = " これはコピーするテキストです。"
copied_text = text_to_copy  # ここでコピーが行われます

print(" コピーされたテキスト:", copied_text)
```

このコードは、`text_to_copy` 変数の内容を `copied_text` 変数にコピーし、それを出力します。

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

ヘルパーライブラリ

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 = ['T シャツ/トップス', 'ズボン', 'プルオーバー', 'ドレス', 'コート',
               'サンダル', 'シャツ', 'スニーカー', 'バッグ', 'アンクルブーツ']
print(train_images.shape)
print(len(train_labels))
```

結果が出力されました。ここに train_images、train_labels、test_images、test_labels があることに注目してください。これらは訓練データセットとテストデータセットに分かれています。

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

このコードブロックは、Python の NumPy 配列や Pandas データフレームの形状や要素数を表示する際の出力例を示しています。具体的には、以下のように解釈できます：

- (60000, 28, 28)：これは 3 次元の配列またはテンソルの形状を示しています。60000 個の要素があり、それぞれが 28x28 の 2 次元配列（例えば、画像データ）であることを意味します。
- 60000：これは配列の最初の次元のサイズ、つまり 60000 個の要素があることを示しています。

この出力は、機械学習のデータセット（例えば、MNIST データセット）を扱う際によく見られる形式です。

次に、画像を印刷してみましょう。

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

結果を見てみましょう。

```
[[  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]]
  ....
```

以下に結果の一部を抜粋しました。

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

出力は 28 です。したがって、これは横幅が 28 の行列であることが明確です。続けて印刷します。

```
    print(len(train_images[0][0][0])
TypeError: 'numpy.uint8' 型のオブジェクトには len() がありません
```

したがって、非常に明白です。各画像は 28*28*3 の配列です。最後の次元の配列には RGB 値が保存されています。しかし、私たちの考えが間違っている可能性があることに気づきました。

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

このコードは、train_images という多次元配列（またはリスト）の最初の要素の 2 番目の要素の 21 番目の要素を出力します。具体的には、train_images[0] が最初の要素、train_images[0][1] がその中の 2 番目の要素、そして train_images[0][1][20] がその中の 21 番目の要素を指します。このコードは、その値を表示するために使用されます。

```
0
```

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

このコードは、train_images というリスト（または配列）の最初の要素の 2 番目の要素を出力します。Python ではインデックスが 0 から始まるため、[0] が最初の要素を指し、[1] がその中の 2 番目の要素を指します。

```
[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]
```

（注：このコードブロックは数値のリストを表しており、翻訳の必要はありません。そのまま保持します。）

説明によると、各画像は 28×28 の配列です。しばらく試行錯誤した後、ついに秘密がわかりました。

まずは出力された図を見てみましょう。

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

このコードは、Python の Matplotlib ライブラリを使用して、トレーニングデータセットの最初の画像を表示するものです。各コマンドの役割は以下の通りです：

- `plt.figure()`: 新しい図を作成します。
- `plt.imshow(train_images[0])`: トレーニングデータセットの最初の画像を表示します。
- `plt.colorbar()`: 画像の横にカラーバーを表示します。
- `plt.grid(False)`: グリッドを非表示にします。
- `plt.show()`: 作成した図を表示します。

右側のカラーバーが見えますか。`0` から `250` まであります。これは元々、2 つの色の間のグラデーションです。しかし、どうやってどの 2 色を選ぶのか知っているのでしょうか。私たちはどこでそれを指定したのでしょうか。

次に、2 番目の図も印刷します。

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

このコードは、`train_images` というリストまたは配列の中の 2 番目の画像（インデックス 1）を表示するために使用されます。`plt.imshow()` は Matplotlib ライブラリの関数で、画像をプロットするために使われます。

面白いですね。これは `pyplot` の依存ライブラリのデフォルト設定なのでしょうか。引き続き、公式サイトに掲載されているコードを実行してみましょう。

```
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()
```

このコードは、25 枚の訓練画像を 5x5 のグリッドで表示するためのものです。各画像の下には、その画像に対応するラベルが表示されます。`plt.xticks([])` と `plt.yticks([])` は、x 軸と y 軸の目盛りを非表示にします。`plt.grid(False)` は、グリッドを非表示にします。`plt.imshow` は、画像を表示し、`plt.xlabel` は、画像の下にラベルを表示します。最後に `plt.show()` を呼び出して、図を表示します。

ここに画像とその分類が表示されていることに気づきました。ついに `cmp` パラメータの正体がわかりました。もし `cmp` に何も指定しなければ、きっと先ほどのような色合いになるはずです。やはりその通りでした。

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

今回は `pyplot cmap` を検索します。いくつかの資料が見つかりました。

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

コードを修正します。

```
plt.figure(figsize=(10,10))
for i in range(25):
    plt.subplot(5,5,i+1)    ## この行を修正
    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()
```

しかし、エラーが発生しました。

```
ValueError: num は 1 <= num <= 10 でなければなりませんが、11 が指定されました
```

これは何を意味しているのでしょうか。以前の `5,5,i+1` は一体どういう意味だったのでしょうか。なぜ `2` に変更するとうまくいかないのでしょうか。直感的には 5 行 5 列という意味だとわかりますが、なぜこのエラーが発生するのでしょうか。`11` はどのように計算されたのでしょうか。`num` とは何を意味しているのでしょうか。`10` とは何を意味しているのでしょうか。`2*5=10` に気づきました。おそらく `i=11` のときにエラーが発生したのでしょう。`for i in range(10):` に変更したとき、以下の結果が得られました。

少しドキュメントを見てみると、`subplot(nrows, ncols, index, **kwargs)` ということがわかります。うん、これでかなり理解できました。

```
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()
```

このコードは、25 枚の訓練画像を 5x5 のグリッドで表示するためのものです。各画像の下には、その画像に対応するラベルが表示されます。`plt.figure(figsize=(10,10))` で図のサイズを設定し、`for` ループを使って 25 枚の画像を順番に表示しています。`plt.subplot(5,5,i+1)` で 5 行 5 列のサブプロットを作成し、`plt.imshow(train_images[i], cmap=plt.cm.Blues)` で画像を表示します。`plt.xlabel(class_names[train_labels[i]])` で画像のラベルを表示し、`plt.show()` で図を表示します。

`0 25` のようなものを `xticks` と呼ぶことに注意してください。このボックスを拡大縮小すると、異なる表示が得られます。

plot_scale

拡大縮小ボックスに注目すると、`xticks` と `xlabels` が異なる表示になることがわかります。

```
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'])
```

このコードは、Keras モデルをコンパイルするためのものです。以下に各パラメータの説明を示します：

- `optimizer='adam'`: モデルの最適化アルゴリズムとして Adam を使用します。Adam は、学習率を自動的に調整する効率的な最適化手法です。
- `loss=tf.keras.losses.SparseCategoricalCrossentropy(from_logits=True)`: 損失関数として Sparse Categorical Crossentropy を使用します。`from_logits=True` は、モデルの出力がロジット（softmax 関数を適用する前の値）であることを示しています。
- `metrics=['accuracy']`: モデルの評価指標として精度（accuracy）を使用します。これは、モデルの予測が正しいかどうかを測定するための一般的な指標です。

この設定により、モデルは Adam オプティマイザを使用して訓練され、損失関数として Sparse Categorical Crossentropy が使用され、訓練中に精度が監視されます。

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

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

print('\nテスト精度:', test_acc)
```

ここでモデルを定義する方法に注目すると、クラス `Sequential` が使用されています。これらのパラメータ、`28,28`、`128`、`relu`、`10` に注意してください。`compile` と `fit` が必要であることに気づきます。`fit` は「フィット」、つまりモデルをデータに適合させることを意味します。`28,28` は画像のサイズであることに気づきます。

```
エポック 1/10
1875/1875 [==============================] - 2s 928us/ステップ - 損失: 0.6331 - 精度: 0.7769
エポック 2/10
1875/1875 [==============================] - 2s 961us/ステップ - 損失: 0.3860 - 精度: 0.8615
エポック 3/10
1875/1875 [==============================] - 2s 930us/ステップ - 損失: 0.3395 - 精度: 0.8755
エポック 4/10
1875/1875 [==============================] - 2s 1ms/ステップ - 損失: 0.3071 - 精度: 0.8890
```

```
エポック 5/10
1875/1875 [==============================] - 2s 1ms/ステップ - 損失: 0.2964 - 精度: 0.8927
エポック 6/10
1875/1875 [==============================] - 2s 985us/ステップ - 損失: 0.2764 - 精度: 0.8955
エポック 7/10
1875/1875 [==============================] - 2s 961us/ステップ - 損失: 0.2653 - 精度: 0.8996
エポック 8/10
1875/1875 [==============================] - 2s 1ms/ステップ - 損失: 0.2549 - 精度: 0.9052
エポック 9/10
1875/1875 [==============================] - 2s 1ms/ステップ - 損失: 0.2416 - 精度: 0.9090
エポック 10/10
1875/1875 [==============================] - 2s 1ms/ステップ - 損失: 0.2372 - 精度: 0.9086
313/313 - 0s - 損失: 0.3422 - 精度: 0.8798
```

テスト精度: 0.879800021648407

モデルがトレーニングされました。パラメータを調整しましょう。

```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)
])
```

`Dense` の最初のパラメータを修正します。

```
エポック 1/10
1875/1875 [==============================] - 2秒 714マイクロ秒/ステップ - 損失: 6.9774 - 精度: 0.3294
エポック 2/10
1875/1875 [==============================] - 1秒 715マイクロ秒/ステップ - 損失: 1.3038 - 精度: 0.4831
エポック 3/10
1875/1875 [==============================] - 1秒 747マイクロ秒/ステップ - 損失: 1.0160 - 精度: 0.6197
エポック 4/10
1875/1875 [==============================] - 1秒 800マイクロ秒/ステップ - 損失: 0.7963 - 精度: 0.6939
エポック 5/10
1875/1875 [==============================] - 2秒 893マイクロ秒/ステップ - 損失: 0.7006 - 精度: 0.7183
```

```
エポック 6/10
1875/1875 [==============================] - 1秒 747マイクロ秒/ステップ - 損失: 0.6675 - 精度: 0.7299
エポック 7/10
1875/1875 [==============================] - 1秒 694マイクロ秒/ステップ - 損失: 0.6681 - 精度: 0.7330
エポック 8/10
1875/1875 [==============================] - 1秒 702マイクロ秒/ステップ - 損失: 0.6675 - 精度: 0.7356
エポック 9/10
1875/1875 [==============================] - 1秒 778マイクロ秒/ステップ - 損失: 0.6508 - 精度: 0.7363
エポック 10/10
1875/1875 [==============================] - 1秒 732マイクロ秒/ステップ - 損失: 0.6532 - 精度: 0.7350
313/313 - 0秒 - 損失: 0.6816 - 精度: 0.7230
```

テスト精度: 0.7229999899864197

`Test accuracy`の前後で変化が確認できます。`Epoch`というのは`fit`関数が出力するログです。バッチサイズが`128`の場

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

上記のコードは、train_labels というリストの内容とその長さを表示しています。train_labels には、訓練データのラベルが格納されており、その長さは 60,000 です。

これは、これらのカテゴリを 0 から 9 の数字で表すことを意味します。ちょうど class_names も 10 個あります。

```
class_names = ['Tシャツ/トップス', 'ズボン', 'プルオーバー', 'ドレス', 'コート',
               'サンダル', 'シャツ', 'スニーカー', 'バッグ', 'アンクルブーツ']
```

もう一度修正してみましょう。

```python
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'])
```

このコードは、Keras モデルをコンパイルするためのものです。以下に各パラメータの説明を示します：

- `optimizer='adam'`: モデルの最適化アルゴリズムとして Adam を使用します。Adam は、勾配降下法の一種で、学習率を自動的に調整するため、広く使われています。
- `loss=tf.keras.losses.SparseCategoricalCrossentropy(from_logits=True)`: 損失関数として Sparse Categorical Crossentropy を使用します。`from_logits=True` は、モデルの出力がロジット（未正規化のスコア）であることを示しています。
- `metrics=['accuracy']`: モデルの評価指標として精度（accuracy）を使用します。これは、モデルの性能を評価するために使用されます。

この設定により、モデルは Adam オプティマイザを使用して訓練され、損失関数として Sparse Categorical Crossentropy が適用され、精度が評価指標として使用されます。

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

エラーが発生しました。

```
tensorflow.python.framework.errors_impl.InvalidArgumentError:  ラベル値として9を受け取りましたが、これは有効
         [[node sparse_categorical_crossentropy/SparseSoftmaxCrossEntropyWithLogits/SparseSoftmaxCrossEn
```

関数呼び出しスタック: train_function

`Sequential`の3番目のパラメータである`Dense`のパラメータを`15`に変更すればOKです。結果に大きな違いはありません。

```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'])
```

このコードは、Keras モデルをコンパイルするための設定を行っています。以下に各パラメータの説明を日本語で示します。

- `optimizer='adam'`: オプティマイザとして Adam を使用します。Adam は、適応的な学習率を提供するため、多くの場合で効果的な最適化アルゴリズムです。
- `loss=tf.keras.losses.SparseCategoricalCrossentropy(from_logits=True)`: 損失関数として Sparse Categorical Crossentropy を使用します。`from_logits=True` は、モデルの出力がロジット（softmax を適用する前の値）であることを示しています。
- `metrics=['accuracy']`: 評価指標として精度（accuracy）を使用します。これは、モデルの性能を評価するための指標です。

この設定により、モデルは Adam オプティマイザを使用して訓練され、損失関数として Sparse Categorical Crossentropy が適用され、精度が評価指標として使用されます。

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

このコードは、ニューラルネットワークモデルを訓練するためのものです。`train_images` と `train_labels` はそれぞれ訓練データの画像とそのラベルを表しています。`epochs=15` は、モデルを 15 回繰り返し訓練することを指定しています。以前は 10 回だったのが、15 回に増やされています。

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

print('\nテスト精度:', test_acc)
```

```
エポック 1/15
1875/1875 [==============================] - 2s 892us/ステップ - 損失: 6.5778 - 精度: 0.3771
エポック 2/15
1875/1875 [==============================] - 2s 872us/ステップ - 損失: 1.3121 - 精度: 0.4910
エポック 3/15
1875/1875 [==============================] - 2s 909us/ステップ - 損失: 1.0900 - 精度: 0.5389
エポック 4/15
1875/1875 [==============================] - 1s 730us/ステップ - 損失: 1.0422 - 精度: 0.5577
エポック 5/15
1875/1875 [==============================] - 1s 709us/ステップ - 損失: 0.9529 - 精度: 0.5952
エポック 6/15
1875/1875 [==============================] - 1s 714us/ステップ - 損失: 0.9888 - 精度: 0.5950
```

```
エポック 7/15
1875/1875 [==============================] - 1s 767us/ステップ - 損失: 0.8678 - 精度: 0.6355
エポック 8/15
1875/1875 [==============================] - 1s 715us/ステップ - 損失: 0.8247 - 精度: 0.6611
エポック 9/15
1875/1875 [==============================] - 1s 721us/ステップ - 損失: 0.8011 - 精度: 0.6626
エポック 10/15
1875/1875 [==============================] - 1s 711us/ステップ - 損失: 0.8024 - 精度: 0.6622
エポック 11/15
1875/1875 [==============================] - 1s 781us/ステップ - 損失: 0.7777 - 精度: 0.6696
エポック 12/15
1875/1875 [==============================] - 1s 724us/ステップ - 損失: 0.7764 - 精度: 0.6728
エポック 13/15
1875/1875 [==============================] - 1s 731us/ステップ - 損失: 0.7688 - 精度: 0.6767
エポック 14/15
1875/1875 [==============================] - 1s 715us/ステップ - 損失: 0.7592 - 精度: 0.6793
エポック 15/15
1875/1875 [==============================] - 1s 786us/ステップ - 損失: 0.7526 - 精度: 0.6792
313/313 - 0s - 損失: 0.8555 - 精度: 0.6418
```

テスト精度: 0.6417999863624573

注意を15に変更します。違いはあまりありません。`tf.keras.layers.Dense(88, activation='relu')`が重要です。128を

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

このコードは、Keras を使用して確率モデルを作成しています。probability_model は、既存の model に Softmax 層を追加したシーケンシャルモデルです。Softmax 層は、モデルの出力を確率分布に変換するために使用されます。

次に予測を行います。Sequential が上記と同じであることに注意してください。パラメータ model と tf.keras.layers.Softmax() に注目してください。

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

このコードは、TensorFlow と Keras を使用して、モデルの出力に Softmax 層を追加し、テスト画像に対する予測を行うものです。以下に日本語で説明します。

1. `probability_model` という名前の新しいシーケンシャルモデルを作成します。このモデルは、既存の `model` と、その後に追加される `Softmax` 層で構成されています。Softmax 層は、モデルの出力を確率分布に変換します。
2. `predictions` 変数には、`probability_model` を使用して `test_images` に対する予測結果が格納されます。この予測結果は、各クラスに対する確率として出力されます。

このコードは、分類タスクにおいて、モデルの最終的な出力を確率として解釈するために使用されます。

```
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)
```

このコードは、Matplotlib の `xlabel` 関数を使用して、X 軸のラベルを設定しています。ラベルは、予測されたクラス名、その予測確率（パーセンテージ）、および実際のクラス名を含むフォーマット文字列です。`color` パラメータは、ラベルの色を指定します。

- `class_names[predicted_label]`：予測されたクラスの名前
- `100*np.max(predictions_array)`：予測確率をパーセンテージで表示

- class_names[true_label]：実際のクラスの名前
- color=color：ラベルの色を指定

このラベルは、モデルの予測結果と実際のラベルを視覚的に比較するために使用されます。

```
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)
```

この関数は、指定されたインデックス i の予測結果 predictions_array と正解ラベル true_label を使用して、予測値の配列を棒グラフとしてプロットします。グラフの X 軸には 0 から 9 までの数字が表示され、Y 軸の目盛りは非表示になります。棒グラフの色は灰色（#777777）で、Y 軸の範囲は 0 から 1 に設定されます。最後に、predictions_array の中で最も高い値を持つインデックスを predicted_label として取得します。

```
thisplot[predicted_label].set_color('red')
thisplot[true_label].set_color('blue')
```

このコードは、thisplot というオブジェクトの特定のインデックス（predicted_label と true_label）に対応する要素の色を変更しています。predicted_label に対応する要素の色を赤に、true_label に対応する要素の色を青に設定しています。

```
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()
```

これは、この画像が 99% の確率で Ankle boot であることを示しています。plot_image は左側の画像を表示し、plot_value_array は右側のグラフを出力することに注意してください。

```
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()
```

このコードは、指定された行数 (num_rows) と列数 (num_cols) に基づいて、画像とその予測結果をプロットするためのものです。num_images は行数と列数を掛け合わせた総画像数を表します。plt.figure で図のサイズを設定し、for ループ内で各画像とその予測結果をサブプロットとして配置します。plot_image 関数は画像を表示し、plot_value_array 関数は予測結果の配列を表示します。最後に plt.tight_layout() でレイアウトを整え、plt.show() で図を表示します。

ここでは、より多くのテスト結果を表示しているだけであることに気づきました。したがって、使用の流れはおおよそ理解しています。しかし、背後でどのように計算されているかはまだわかりません。それらがどのように使用されるかはわかっていますが、その背後には微積分があります。では、微積分をどのように理解すればよいのでしょうか。

例えば、1 から 100 までの数字を当てるゲームがあるとします。毎回あなたが数字を推測し、私が「小さい」か「大きい」かを教えます。あなたが 50 と推測したら、私は「小さい」と言います。次にあなたが 80 と推測したら、私は「大きい」と言います。あなたが 65 と推測したら、私は「大きい」と言います。あなたが 55 と推測したら、私は「小さい」と言います。最後にあなたが 58 と推測したら、私は「当たり」と言います。

機械学習は、このようなプロセスを背後でシミュレートするものです。ただし、より複雑です。多くの「1 から 100」の範囲があり、多くの数を推測する必要があります。同時に、毎回の推測には多くの計算が必要です。そして、大きいか小さいかを判断するたびに、多くの計算を行います。

PyTorch

PyTorch は、Python 向けのオープンソースの機械学習ライブラリです。Facebook の AI 研究チームによって開発され、柔軟性と使いやすさが特徴です。PyTorch は、動的な計算グラフを採用し

ており、これによりモデルのデバッグや変更が容易になります。また、GPU を活用した高速な計算が可能で、深層学習モデルの開発や研究に広く利用されています。PyTorch は、研究者や開発者にとって強力なツールであり、最新の AI 技術を迅速に実装するためのプラットフォームとして人気を集めています。

インストールしてみましょう。これは 3.9 バージョンの Python をサポートしています。

```
$ pip install torch torchvision
Collecting torch
  Downloading torch-1.8.0-cp39-none-macosx_10_9_x86_64.whl (120.6 MB)
     |                  | 120.6 MB 224 kB/s
Collecting torchvision
  Downloading 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
  Downloading 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
```

確認してみましょう。

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

エラーが発生しました。

```
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: 部分的に初期化されたモジュール 'torch' には 'rand' 属性がありません（おそらく循環インポートが原
```

このエラーメッセージを Google で検索してみました。原因は、私たちのファイルも `torch` という名前だったためです。名前が重複していました。名前を変更したら、正しく動作しました。

```
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]])
```

このコードブロックは、PyTorch のテンソルを表示しています。テンソルは、5 行 3 列の行列で、各要素は浮動小数点数です。具体的には、以下のような値を持っています：

- 1 行目: [0.5520, 0.9446, 0.5543]
- 2 行目: [0.6192, 0.0908, 0.8726]
- 3 行目: [0.0223, 0.7685, 0.9814]
- 4 行目: [0.4019, 0.5406, 0.3861]
- 5 行目: [0.5485, 0.6040, 0.2387]

このテンソルは、機械学習モデルの重みや入力データなど、さまざまな用途で使用されることがあります。

例を見つけました。

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

import torch
import math
dtype = torch.float
device = torch.device("cpu")
# device = torch.device("cuda:0") # GPU で実行する場合はこの行のコメントを解除してください
```

ランダムな入力と出力データを作成

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

重みをランダムに初期化

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):
    # 順伝播：予測値 y を計算
    y_pred = a + b * x + c * x ** 2 + d * x ** 3

# 損失を計算して表示
loss = (y_pred - y).pow(2).sum().item()
if t % 100 == 99:
    print(t, loss)

# 損失に対するa，b，c，dの勾配を計算するためのバックプロパゲーション
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()

# 勾配降下法を使用して重みを更新
a -= learning_rate * grad_a
b -= learning_rate * grad_b
c -= learning_rate * grad_c
d -= learning_rate * grad_d
```

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

実行してみましょう。

```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
結果: y = -0.011828352697193623 + 0.8360244631767273 x + 0.002040589228272438 x^2 + -0.0903836563229560
```

numpy ライブラリだけを使ったコードを見てみましょう。

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

ランダムな入力と出力データを作成する

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

重みをランダムに初期化

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):
    # 順伝播: 予測値 y を計算
    # y = a + b x + c x^2 + d x^3
    y_pred = a + b * x + c * x ** 2 + d * x ** 3

# 損失を計算して表示
loss = np.square(y_pred - y).sum()
```

```
if t % 100 == 99:
    print(t, loss)

# 損失に対するa, b, c, dの勾配を計算するためのバックプロパゲーション
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()

# 重みの更新
a -= learning_rate * grad_a
b -= learning_rate * grad_b
c -= learning_rate * grad_c
d -= learning_rate * grad_d

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

これは 2 つの計算方法であることに注意してください。

これらの例では、まず一連の x と y を生成しています。次に、三次方程式を仮定します。その後、いくつかの方法を使用して係数を反復計算します。これらのアルゴリズムはどのようなものなのでしょうか。2000 回ループしていることに注意してください。各反復でより正確にフィットさせています。ここでは詳細には触れません。

最後に

現在のところ、私たちは機械学習の背後にある計算方法を理解していません。しかし、それは今のところ重要ではありません。上記のような知識を使って、私たちはすでに多くのことを行うことができます。また、機械学習を使ってテキストやオーディオなどを処理することもできます。数十の例を試してみてから、その原理を学ぶのも遅くはありません。

練習

- 学生は上記のように探索を行います。