機械学習エンジニアコース
Sprint

－ フレームワーク2_Keras －

DIVE INTO CODE

目的はなにか

フレームワークのコードを読めるようにする

フレームワークを習得し続けられるようになる

知っている理論の範囲をフレームワークで動かす

© 2020 DIVE INTO CODE Corp.

このスライドは?

ここでは、Kerasの基本的な知識を学びましょう

© 2020 DIVE INTO CODE Corp.

Kerasとは

Keras (κέρας) はギリシア語で**角**を意味します。
これは κέρας （角）/ κραίνω （遂行）と ἐλέφας （象牙）/ ἐλεφαίρομαι （欺瞞）の似た響きを楽しむ言葉遊びです。

https://keras.io/ja/

© 2020 DIVE INTO CODE Corp.

Kerasとは

Kerasの作者

Kerasの作者である **François Chollet** (フランソワ ショレ)は、人工知能研究者であり、2015年8月よりGoogleでDeep Learning研究およびTensorflow・Kerasの開発を行っています。
彼は自身の研究と実験のために2015年3月27日にKerasの最初のバージョンをGitHubにコミットしました。
Kerasは当時のほかのフレームワーク（Torch・Theano・Caffe）に比べて、効率的で使いやすく設計されたAPIであったため、多くの研究者や開発者が好んで使用しました。

François Cholletのパーソナルページ
https://fchollet.com/
François CholletのTwitter
https://twitter.com/fchollet?ref_src=twsrc%5Egoogle%7Ctwcamp%5Eserp%7Ctwgr%5Eauthor

© 2020 DIVE INTO CODE Corp.

Kerasとは

Kerasのバックエンド

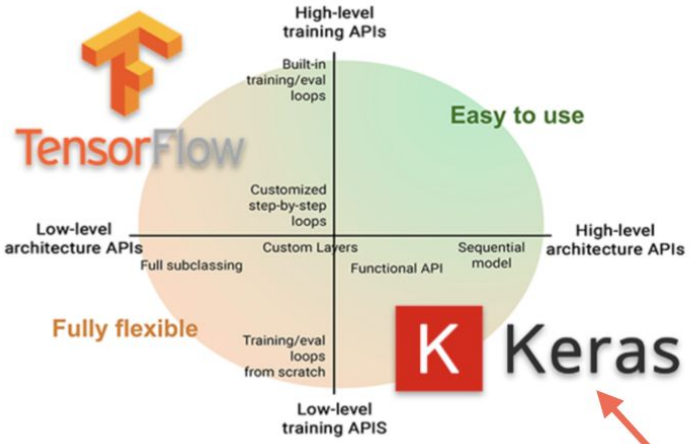
+

Kerasはネットワークをトレーニングするために**バックエンド**（= 計算エンジン; グラフ・トポロジーの構築、最適化、数値計算を実行する）を必要としました。
Kerasはバックエンドへアクセスする抽象的な言語集合と考えることができるため、バックエンドを交換することができました。
当初Kerasのデフォルトバックエンドは Teano でした。このころ、GoogleがTensorflowをリリースし、KerasはバックエンドとしてTensorflowをサポートしました。Tensorflowが人気のあるバックエンドとして定着したため、Kerasのv1.1.0からはTensorflowがデフォルトバックエンドとなりました。

© 2020 DIVE INTO CODE Corp.

Kerasとは

tf.keras の 登場



TensorFlow v1.10.0では、Tensorflowのパッケージ内にKerasを統合し、サブモジュールとして**tf.keras**が導入されました。さらに2019年6月、TensorFlow 2.0が発表された際に、tf.kerasは公式高レベルAPIになりました。

独立したKerasからtf.kerasへの移行はシンプルで、import文の書き換えで済みます。（コード表記はkerasをtf.kerasと書き換えればよい）

```
In [ ]:  1  from keras import ..

         1

In [ ]:  1  from tensorflow.keras import ..
```

© 2020 DIVE INTO CODE Corp.

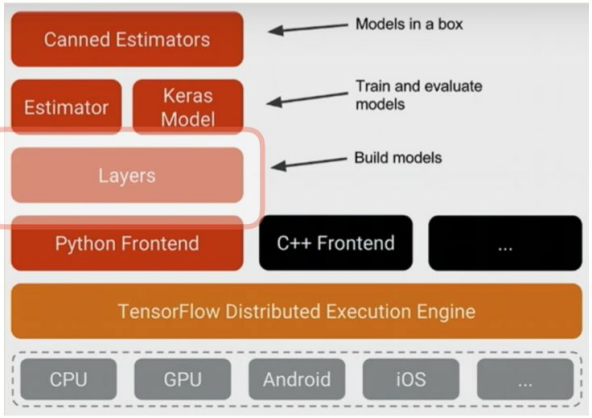
Kerasとは

tf.layers から tf.keras.layersへ

Tensorflow v1.13以前は、標準的に**モデル構築**においてtf.layers APIが用いてきました。
v1.13以降はtf.layersは非推奨となり（warningが出るようになった）、Tensorflow v2.0以降では、tf.keras.layersで置き換えられるようになりました。

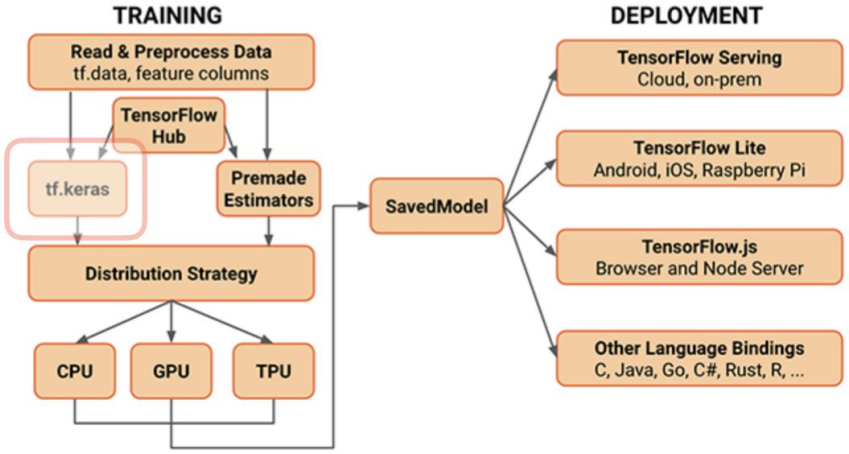
https://github.com/tensorflow/tensorflow/issues/26144

研究、実験、モデルの準備/量子化、プロダクションへの展開がより早く、効率的になった。



Tensorflow v1.13以前

フレームワーク
設計の変遷



Tensorflow v2.x

© 2020 DIVE INTO CODE Corp.

Kerasとは

tf.keras によるモデル構築には、
次の3つの記法を用意しています。

（用途に応じて好きな記法を選べます）

リンク:

https://www.tensorflow.org/api_docs/python/tf/keras/Sequential

https://www.tensorflow.org/guide/keras/functional

https://www.tensorflow.org/api_docs/python/tf/keras/Model

Sequential Model API (dead simple!)

Sequential( )をインスタンス化し、**レイヤ**のインスタンスをadd( )メソッドにより線形にスタックすることでモデルが構築される。モデルの構造に制約がある。（single-input, single-output）

functional API (like Lego!)

レイヤのインスタンスを関数的に呼び出すことができ、戻り値として tensor（多次元配列）[1]を返す。レイヤからのアウトプットは次のレイヤのインプットになる。**Model( )**のインスタンスが、インプットとアウトプットのレイヤを引数にとり、互いに繋ぐことで複雑な構造のモデルを構築できる。（Multi-input, multi-output）

[1] テンソルとは：https://www.tensorflow.org/tutorials/eager/basics?hl=ja

Model subclassing (fully-customizable!)

開発者が柔軟にモデルを構築したいときに向いている。

© 2020 DIVE INTO CODE Corp.

Kerasとは

Keras Sequential model API

ベースライン

Keras ガイド： https://keras.io/ja/models/sequential/

tf.kerasガイド： https://www.tensorflow.org/api_docs/python/tf/keras/Sequential

p7のように、import文をこう書けば、
以前の tf.layers のコードが再利用できる

```
import tensorflow as tf
from tensorflow import keras
from tensorflow.keras import layers

model = keras.Sequential()
model.add(layers.Dense(20, activation='relu',
input_shape=(10,)))         # 出力ユニット数:20
model.add(layers.Dense(20, activation='relu'))
model.add(layers.Dense(20,activation='softmax'))
```

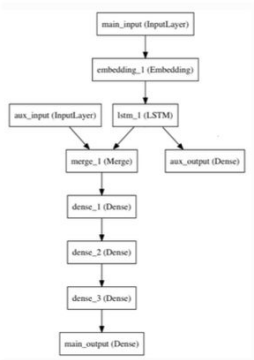
```
optimizer = keras.optimizers.RMSprop()
loss = keras.losses.SparseCategoricalCrossentropy(from_logits=True)
metrics=['accuracy']

model.compile(optimizer=optimizer, loss=loss, metrics=metrics)
model.fit(x, y, epochs=10, batch_size=32)
```

© 2020 DIVE INTO CODE Corp.

Kerasとは

Keras functional API

ベースライン



Functional APIでは、複雑なネットワークも書ける

Keras ガイド： https://keras.io/ja/getting-started/functional-api-guide/

tf.kerasガイド： https://www.tensorflow.org/guide/keras/functional

```
import tensorflow as tf
from tensorflow import keras
from tensorflow.keras import layers

inputs = keras.Input(shape=(10,))

x = layers.Dense(20, activation='relu')(inputs)
x = layers.Dense(20, activation='relu')(x)

outputs = layers.Dense(10,activation='softmax')(x)
```

```
optimizer = keras.optimizers.RMSprop()
loss = keras.losses.SparseCategoricalCrossentropy(from_logits=True)
metrics=['accuracy']

model = keras.Model(inputs, outputs)
model.compile(optimizer=optimizer, loss=loss, metrics=metrics)
model.fit(x, y, epochs=10, batch_size=32)
```

© 2020 DIVE INTO CODE Corp.

Kerasとは

Model subclassing

ベースライン

Keras ガイド： https://keras.io/models/about-keras-models/#model-subclassing

tf.kerasガイド： https://www.tensorflow.org/api_docs/python/tf/keras/Model

```
import tensorflow as tf
from tensorflow import keras
from tensorflow.keras import layers

class MyModel(keras.Model):
   def __init__(self):
      super(MyModel, self).__init__()
      self.dense1 = layers.Dense(20,
                                 activation='relu')
      self.dense2 = layers.Dense(20,
                                 activation='relu')
      self.dense1 = layers.Dense(10,
                                 activation='softmax')
   def call(self, inputs):
      x = self.dense1(x)
      x = self.dence2(x)
      return self.dense3(x)

optimizer = keras.optimizers.RMSprop()
loss = keras.losses.SparseCategoricalCrossentropy(from_logits=True)
metrics=['accuracy']

model = MyModel()
model.compile(optimizer=optimizer, loss=loss, metrics=metrics)
model.fit(x, y, epochs=10, batch_size=32)
```

© 2020 DIVE INTO CODE Corp.

Kerasとは

Model subclassing

ベースライン

例：LeNet クラス

```
class LeNet(tf.keras.Model):
    def __init__(self):
        super(LeNet, self).__init__()
        self.conv2d_1 = tf.keras.layers.Conv2D(filters=6,
                            kernel_size=(3, 3),
                           activation='relu',
                            input_shape=(32,32,1))

        self.average_pool = tf.keras.layers.AveragePooling2D()

        self.conv2d_2 = tf.keras.layers.Conv2D(filters=16,
                            kernel_size=(3, 3),
                            activation='relu')

        self.flatten = tf.keras.layers.Flatten()
        self.fc_1 = tf.keras.layers.Dense(120, activation='relu')
        self.fc_2 = tf.keras.layers.Dense(84, activation='relu')
        self.out = tf.keras.layers.Dense(10,activation='softmax')

    def call(self, input):
        x = self.conv2d_1(input)
        x = self.average_pool(x)
        x = self.conv2d_2(x)
        x = self.average_pool(x)
        x = self.flatten(x)
        x = self.fc_2(self.fc_1(x))
        return self.out(x)

lenet = LeNet()
```

© 2020 DIVE INTO CODE Corp.

Kerasとは

この後の流れ

Sequential model API と **functional API** の
基本的な記法をさらにチェック！

© 2020 DIVE INTO CODE Corp.

Keras Sequential model API

Keras Sequential model API

モデル構築のベースライン

Keras の sequential.py

https://github.com/keras-team/keras/blob/master/keras/engine/sequential.py

①Sequential()のインスタンスを作成しレイヤをスタックする

②Sequential()のインスタンスをcompileする

ここでoptimizer、loss、metricsを指定

③インスタンスにfitし、評価、predictする

④インスタンスをsave

epochごとに上書きし、最後の重みを保存

```python
import keras
from keras.models import Sequential
from keras.layers import Dense

#Create Sequential model with Dense layers, using the add method
model = Sequential()

#Dense implements the operation:
#        output = activation(dot(input, kernel) + bias)
#Units are the dimensionality of the output space for the layer,
#     which equals the number of hidden units
#Activation and loss functions may be specified by strings or classes
model.add(Dense(units=64, activation='relu', input_dim=100))
model.add(Dense(units=10, activation='softmax'))

#The compile method configures the model's learning process
model.compile(loss='categorical_crossentropy',
              optimizer='sgd',
              metrics=['accuracy'])

#The fit method does the training in batches
# x_train and y_train are Numpy arrays --just like in the Scikit-Learn
model.fit(x_train, y_train, epochs=5, batch_size=32)

#The evaluate method calculates the losses and metrics
#     for the trained model
loss_and_metrics = model.evaluate(x_test, y_test, batch_size=128)

#The predict method applies the trained model to inputs
#     to generate outputs
classes = model.predict(x_test, batch_size=128)
```

© 2020 DIVE INTO CODE Corp.

Keras Sequential model API

Sequential model APIの基本

モデル構築の書き方いろいろ

①Sequentialインスタンスを作り、レイヤのインスタンスのリストを渡す書き方

②Sequentialインスタンスを作り、addメソッドでレイヤのインスタンスをスタックしていく書き方

model.layersを実行すると、以下のようなレイヤのリストが返ってくる

```
[<keras.layers.core.Dense at 0x12dd97588>,
 <keras.layers.core.Activation at 0x12dd972e8>,
 <keras.layers.core.Dense at 0x12dd97278>,
 <keras.layers.core.Activation at 0x12dd97e48>]
```

```
model = Sequential([
    Dense(32, input_shape=(784,)),
    Activation('relu'),
    Dense(10),
    Activation('softmax'),
])
```

①

```
model = Sequential()
model.add(Dense(32, input_dim=784))
model.add(Activation('relu'))
```

②

© 2020 DIVE INTO CODE Corp.

Keras Sequential model API

Sequential model APIの基本

入力形状の書き方いろいろ

①input_shape引数を用いる場合

②input_dim引数を用いる場合

③batch_input_shape引数を用いる場合

④明示的に入力形状を書かない場合

model.weightsを実行すると、以下のような重みのリストが返ってくる

```
[<tf.Variable 'dense_1/kernel:0' shape=(500, 32) dtype=float32_ref>,
 <tf.Variable 'dense_1/bias:0' shape=(32,) dtype=float32_ref>,
 <tf.Variable 'dense_2/kernel:0' shape=(32, 32) dtype=float32_ref>,
 <tf.Variable 'dense_2/bias:0' shape=(32,) dtype=float32_ref>]
```

④の場合、fitの前に**model.weights**を実行すると、[]（空のリスト）が返ってくる。

```
# Optionally, the first layer can receive an `input_shape` argument:
model = Sequential()
model.add(Dense(32, input_shape=(500,)))
# Afterwards, we do automatic shape inference:
model.add(Dense(32))

# This is identical to the following:
model = Sequential()
model.add(Dense(32, input_dim=500))

# And to the following:
model = Sequential()
model.add(Dense(32, batch_input_shape=(None, 500)))

# Note that you can also omit the `input_shape` argument:
# In that case the model gets built the first time you call `fit` (or other
# training and evaluation methods).
model = Sequential()
model.add(Dense(32))
model.add(Dense(32))
model.compile(optimizer=optimizer, loss=loss)
# This builds the model for the first time:
model.fit(x, y, batch_size=32, epochs=10)
```

①

②

③

④

fitするまで
重みを保持しない

© 2020 DIVE INTO CODE Corp.

Keras Sequential model API

注意点

同じノートブックに連続してインスタンスを作ると…

レイヤの通し番号（dense_3のように）が繋がってしまう（①②）。

そこで、**K.clear_session( )を実行する**と（③）、レイヤの通し番号が（dense_1に）リセットされる（④）。別のインスタンスを作るときはclear_sessionを挿入すること推奨します。

※ Kとは、Keras backend APIのことを指します。「from keras import backend as K」という具合にimportします。

K.clear_session()は、現在のTFグラフ（一つのノートブック上で繋がっている）を破棄して新しいものを作成する。古いモデルやレイヤーの乱雑さを防ぎます。

https://www.tensorflow.org/api_docs/python/tf/keras/backend/clear_session

処理が終わった後に、AttributeError: 'NoneType' object has no attribute 'TF_NewStatus' というエラーが出る場合は、K.clear_session()で回避する処置が取られています。
https://github.com/tensorflow/tensorflow/issues/8652

```
model = Sequential()
model.add(Dense(32, input_shape=(500,)))
model.add(Dense(32))
model.weights
```

①

executed in 44ms, finished 20:53:45 2019-06-02

```
[<tf.Variable 'dense_1/kernel:0' shape=(500, 32) dtype=float32_ref>,
 <tf.Variable 'dense_1/bias:0' shape=(32,) dtype=float32_ref>,
 <tf.Variable 'dense_2/kernel:0' shape=(32, 32) dtype=float32_ref>,
 <tf.Variable 'dense_2/bias:0' shape=(32,) dtype=float32_ref>]
```

```
model = Sequential()
model.add(Dense(32, input_shape=(500,)))
model.add(Dense(32))
model.weights
```

②

executed in 46ms, finished 20:53:47 2019-06-02

```
[<tf.Variable 'dense_3/kernel:0' shape=(500, 32) dtype=float32_ref>,
 <tf.Variable 'dense_3/bias:0' shape=(32,) dtype=float32_ref>,
 <tf.Variable 'dense_4/kernel:0' shape=(32, 32) dtype=float32_ref>,
 <tf.Variable 'dense_4/bias:0' shape=(32,) dtype=float32_ref>]
```

インスタンスの名前を変えたとしても上と繋がっちゃう

```
K.clear_session()
```

③

```
model = Sequential()
model.add(Dense(32, input_shape=(500,)))
model.add(Dense(32))
model.weights
```

executed in 42ms, finished 20:53:48 2019-06-02

```
[<tf.Variable 'dense_1/kernel:0' shape=(500, 32) dtype=float32_ref>,
 <tf.Variable 'dense_1/bias:0' shape=(32,) dtype=float32_ref>,
 <tf.Variable 'dense_2/kernel:0' shape=(32, 32) dtype=float32_ref>,
 <tf.Variable 'dense_2/bias:0' shape=(32,) dtype=float32_ref>]
```

④

© 2020 DIVE INTO CODE Corp.

Keras functional API

Keras functional API

モデル構築のベースライン

functional.ipynb
https://github.com/tensorflow/docs/blob/master/site/en/guide/keras/functional.ipynb

input_layer.py
https://github.com/keras-team/keras/blob/master/keras/engine/input_layer.py#L114

core.py
https://github.com/keras-team/keras/blob/master/keras/layers/core.py

①Input()とレイヤのインスタンスからtf.Tensorを出力

②Model()のインスタンスに①の出力を渡す

③Model()のインスタンスをcompileする

ここでoptimizer、loss、metricsを指定

③インスタンスにfitし、評価、predictする

④インスタンスをsave

epochごとに上書きし、最後の重みを保存

```
from keras.layers import Input, Dense
from keras.models import Model

# This returns a tensor
inputs = Input(shape=(784,))

# a layer instance is callable on a tensor, and returns a tensor
x = Dense(64, activation='relu')(inputs)
x = Dense(64, activation='relu')(x)
predictions = Dense(10, activation='softmax')(x)

# This creates a model that includes
# the Input layer and three Dense layers
model = Model(inputs=inputs, outputs=predictions)
model.compile(optimizer='rmsprop',
              loss='categorical_crossentropy',
              metrics=['accuracy'])
model.fit(data, labels)  # starts training
```

※評価とpredictは省略

© 2020 DIVE INTO CODE Corp.

Keras functional API

Keras functional API の疑問

レイヤのインスタンスに接する括弧は何？

```
# a layer instance is callable on a tensor, and returns a tensor
x = Dense(64, activation='relu')(inputs)
x = Dense(64, activation='relu')(x)
predictions = Dense(10, activation='softmax')(x)
```

コレ

レイヤはそのインスタンスを**関数として呼び出す**ことが可能なクラス（関数っぽく呼び出すとcallの処理が発動する）。

Pythonで関数呼び出し可能なクラスを作ってみよう

```
# 関数呼び出し可能なLayerのクラスを作る

class My_Layer:

    def __init__(self, a):

        self.a = a

        print("My_Layer init")

    def __call__(self, b):

        print("My_Layer call")

        add = b + self.a

        print("足し算：{}".format(add))
```

executed in 5ms, finished 07:50:17 2019-09-06

```
# インスタンス化する

"""
input: 3

"""

layer = My_Layer(3)

print(layer)
```

executed in 5ms, finished 07:50:17 2019-09-06

```
My_Layer init
<__main__.My_Layer object at 0x12b0d3b38>
```

```
# インスタンスを関数っぽく呼んでみる

"""
input: 2

"""

layer(2)
```

executed in 5ms, finished 07:50:18 2019-09-06

```
My_Layer call
足し算：5
```

© 2020 DIVE INTO CODE Corp.

Keras の設計思想

Keras の 設計思想

ユーザーのためにハンバーガーを作る2種類のコードを考えてみましょう。

① チェックボックス主導型のデザイン（レストラン側の手続きを記述する）

```
cooked_burger = cook_burger(burger,
    grill_model='GR12',  # バーガー調理器
    time_on_grill=120,  # 調理時間120秒
    grill_temperature=150  # 150度で加熱
)
```

② ユーザー主導型のデザイン（ユーザーの体験に焦点を当てる）

```
cooked_burger = cook_burger(burger,
    level='medium rare'
)                       # ユーザー変数levelは、ユーザーの注文を表し、
                        # time_on_grillやgrill_temperatureへマッピングする
```

Cholletは、②のようにユーザー中心の思考で記述できるようにKerasを設計しました。

© 2020 DIVE INTO CODE Corp.

Keras の設計思想

Keras の 設計思想

それぞれの設計における変数の例

Tensorflow v1.xの低レベルAPIに出現するような変数だね！

チェックボックス主導型の変数：

graph、session、scope、buffer、param_group

ユーザー主導型の変数：

layer、model、optimizer、weights、initializer

© 2020 DIVE INTO CODE Corp.

Keras の設計思想

Keras の 設計思想

また、Keras のAPI設計にCholletは scikit-learn を参照しています。Kerasに登場するmodel.fitやmodel.predictはまさにscikit-learn的記法といえます。

(例：scikit-learnの線形回帰モデル)

```
from sklearn.linear_model import LinearRegression
model = LinearRegression (fit_intercept = False)
model.fit (X、y)
y_pred = model.predict (X_pred)
```

© 2020 DIVE INTO CODE Corp.

フレームワーク2_Keras
完

© 2020 DIVE INTO CODE Corp.