

TensorFlowチュートリアル



TensorFlow

by icchi

2016/10/12

本チュートリアルの概要

- 狙い
 - 短時間でTensorFlowについて理解
 - TensorFlowのコードも読めるように
- ターゲット
 - 深層学習の仕組みを簡単に理解している人

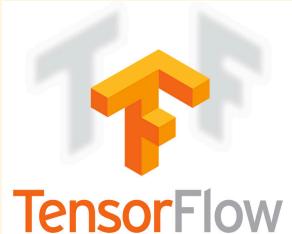
引用した解説資料について

- エンジニアとして 知っておくと幸せになれる
(かも知れない) 機械学習とTensorFlowのこと
- by Norhiro Shimoda
- DevFest Tokyo2016で発表された内容
- <https://goo.gl/1iZPl0>

目次

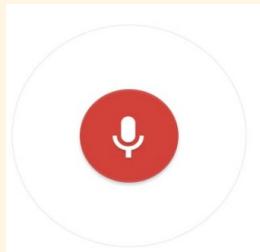
1. TensorFlowについて
 1. TensorFlowとは何か
 2. 演算処理(コードを見ながら)
 3. Google Cloud ML
2. 公式チュートリアルコードの解説(iPython)
 1. 初心者向けのmnist
 2. 玄人向けのmnist(CNN)

1. TensorFlowについて



TensorFlowとは？

- Googleによって2015年11月に公開されたオープンソース
 - Google内部で多数の使用実績があるらしい



OK Google
音声認識



Gmail
スパムフィルタ



Google Photo
画像の自動分類



Google 翻訳
翻訳の自動学習

TensorFlowにありがちな勘違い

TensorFlowは深層学習に
特化したツールである

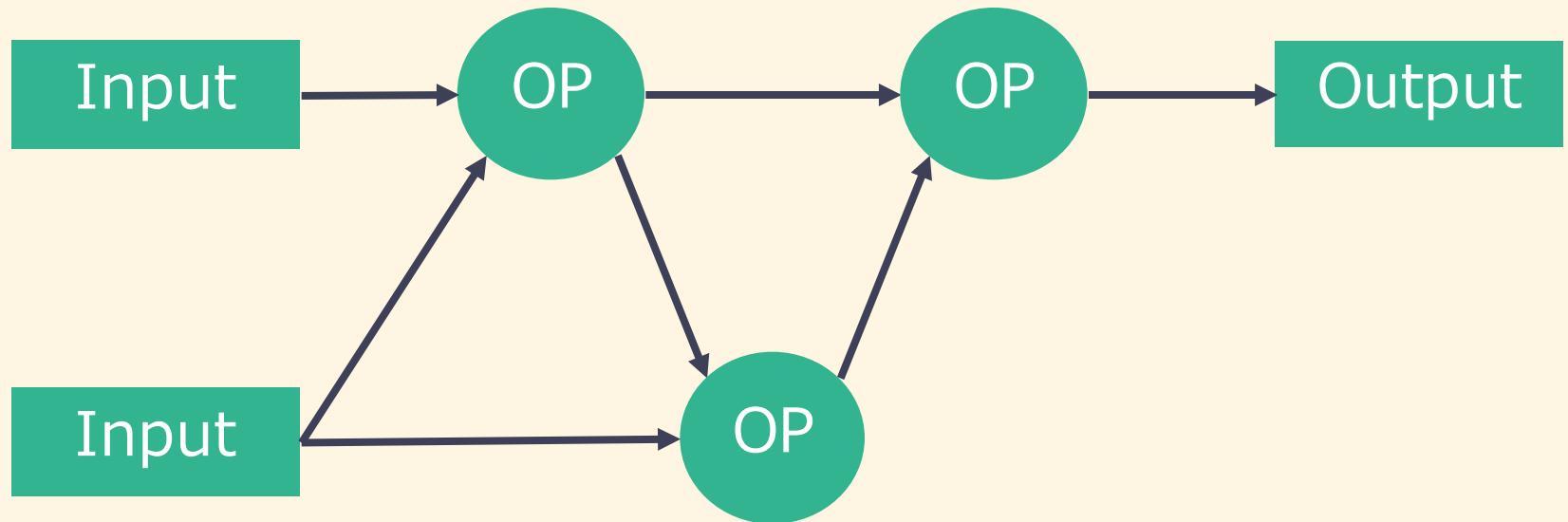
TensorFlowの正しい認識

TensorFlowはデータフローグラフを利用した数値計算のためのオープンソースのソフトウェアライブラリである

TensorFlow™ is an open source software library for numerical computation using data flow graphs.

TensorFlow公式サイトより

TensorFlowの超基本



グラフを意識しながら処理を記述して
最後にどびやっと実行する

TensorFlowのパラダイム

- はじめにテンソルの演算グラフを作る
- グラフの実行単位をセッションという
- 定数、変数、プレースホルダーを宣言できる
- 作ったグラフはデバイス（CPUやGPU）に展開して実行する
- 分散処理も可能である

TensorFlowのパラダイム

- はじめにテンソルの演算グラフを作る
- グラフの実行単位をセッションという
- 定数、変数、プレースホルダーを宣言できる
- 作ったグラフはデバイス（CPUやGPU）に展開して実行する
- 分散処理も可能である

ということを、Pythonをインターフェースとして
行っているフレームワーク

機械学習に関する便利なヘルパー関数がいっぱい
あるのがポイント

ここだけ押さえればTensorFlowが 分かる簡単な例

- 例 1 : 足し算
 - 演算 (operation)
- 例 2 : カウントアップ
 - 変数 (Variable)
- 例 3 : 入力値をいろいろと変える
 - プレースホルダー (Placeholder)
- 例 4 : セッションを使う
 - セッション (Session)
- 例 5 : 高ランクなテンソルの演算
 - テンソル (Tensor)

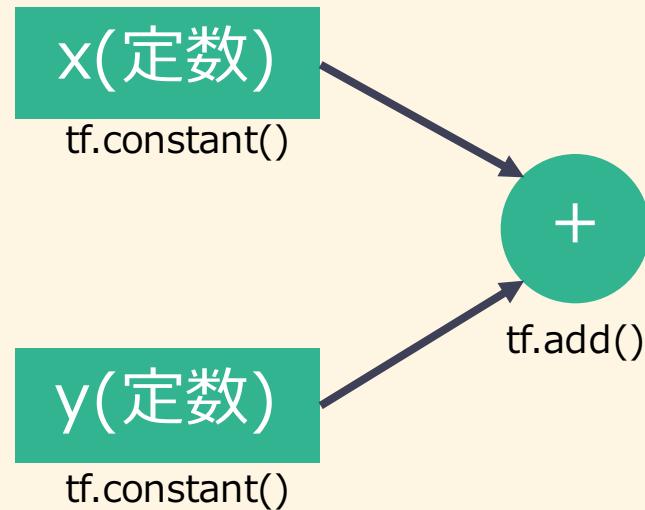
ここだけ押さえればTensorFlowが 分かる簡単な例

- 例 1 : 足し算
 - 演算 (operation)
- 例 2 : カウントアップ
 - 変数 (Variable)
- 例 3 : 入力値をいろいろと変える
 - プレースホルダー (Placeholder)
- 例 4 : セッションを使う
 - セッション (Session)
- 例 5 : 高ランクなテンソルの演算
 - テンソル (Tensor)

1+2 = 3の足し算

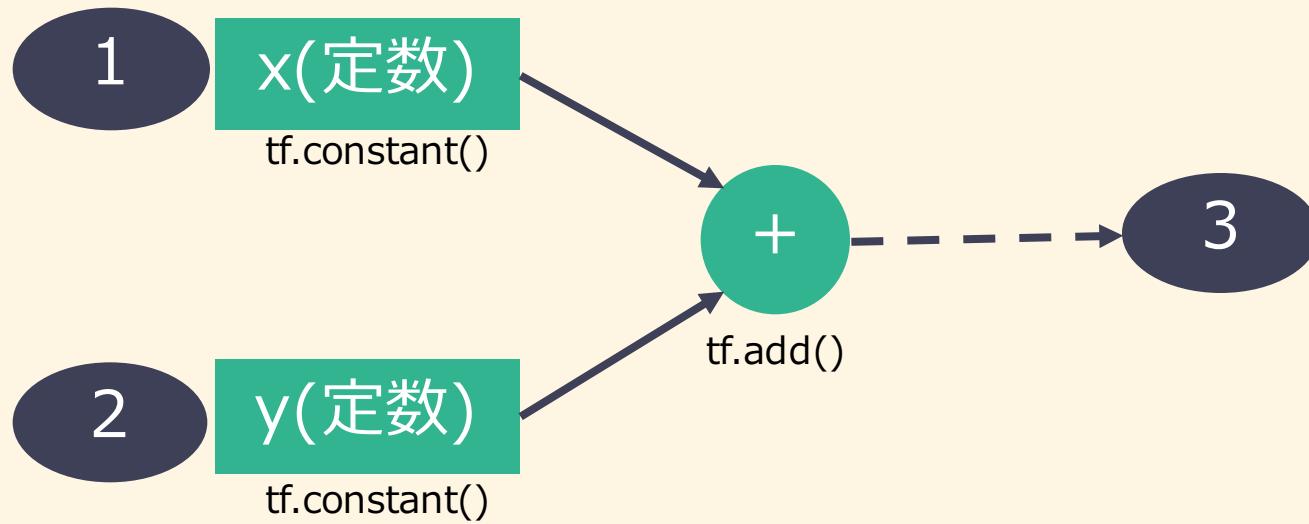
```
import tensorflow as tf  
-  
# 入力となる定数の定義  
x = tf.constant(1, name="x")  
y = tf.constant(2, name="y")  
-  
add_op = tf.add(x, y)  
-  
with tf.Session() as sess:  
    print(sess.run(add_op))
```

演算 (Operation)



演算がグラフのノードとなる（この場合は加算の演算）

演算 (Operation)



演算がグラフのノードとなる（この場合は加算の演算）

ここだけ押さえればTensorFlowが 分かる簡単な例

- 例 1 : 足し算
 - 演算 (operation)
- 例 2 : カウントアップ
 - 変数 (Variable)
- 例 3 : 入力値をいろいろと変える
 - プレースホルダー (Placeholder)
- 例 4 : セッションを使う
 - セッション (Session)
- 例 5 : 高ランクなテンソルの演算
 - テンソル (Tensor)

カウントアップ

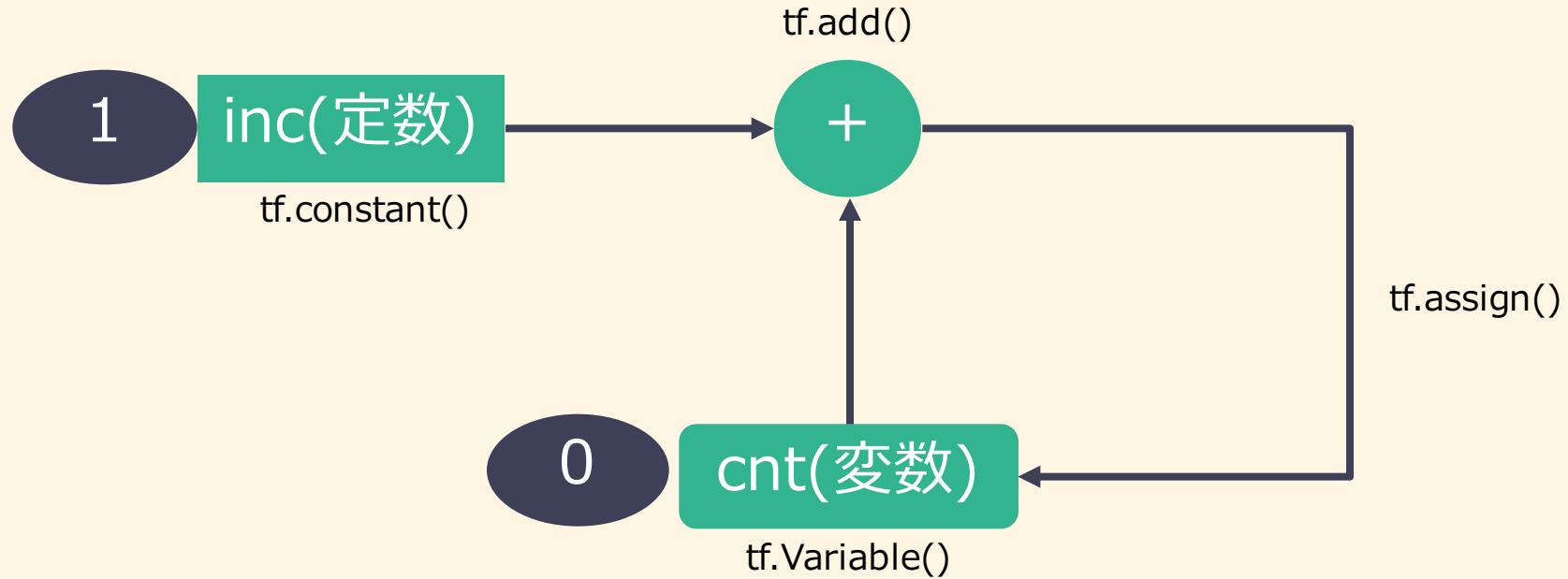
```
import tensorflow as tf

# cntという変数の定義
cnt = tf.Variable(0, name="cnt")
inc = tf.constant(1, name="inc")

# カウントアップ
add_op = tf.add(cnt, inc)
# cntの値を更新
up_op = tf.assign(cnt, add_op)

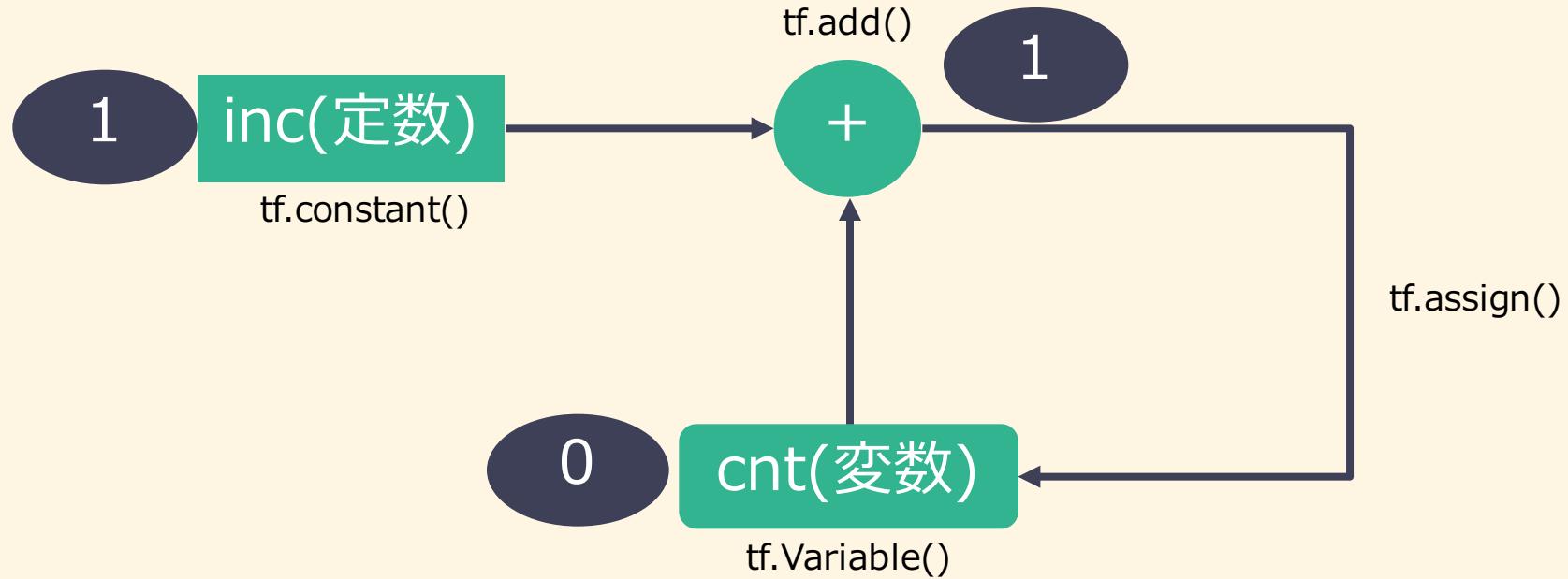
with tf.Session() as sess:
    sess.run(tf.initialize_all_variables())
    # カウントアップを計3回の実施
    print(sess.run(up_op))
    print(sess.run(up_op))
    print(sess.run(up_op))
```

変数(Variable)



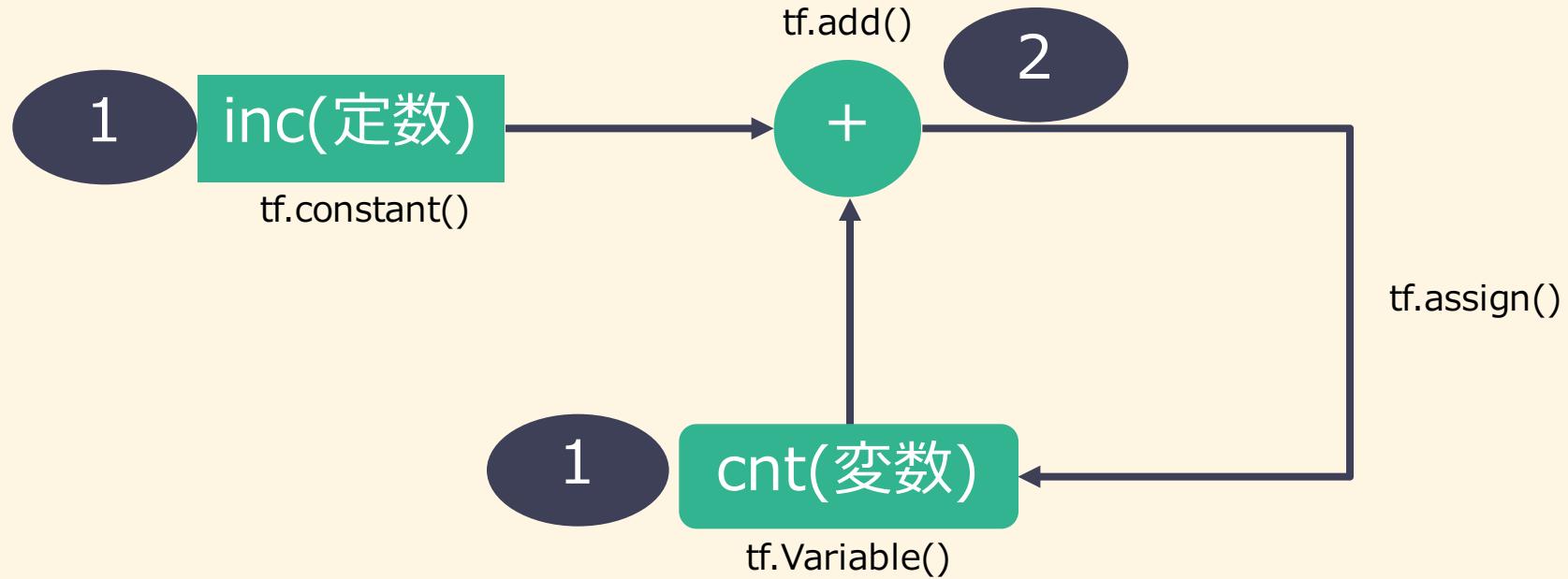
通常のプログラミング言語の変数などと同じように
代入可能な箱として変数がある

変数(Variable)



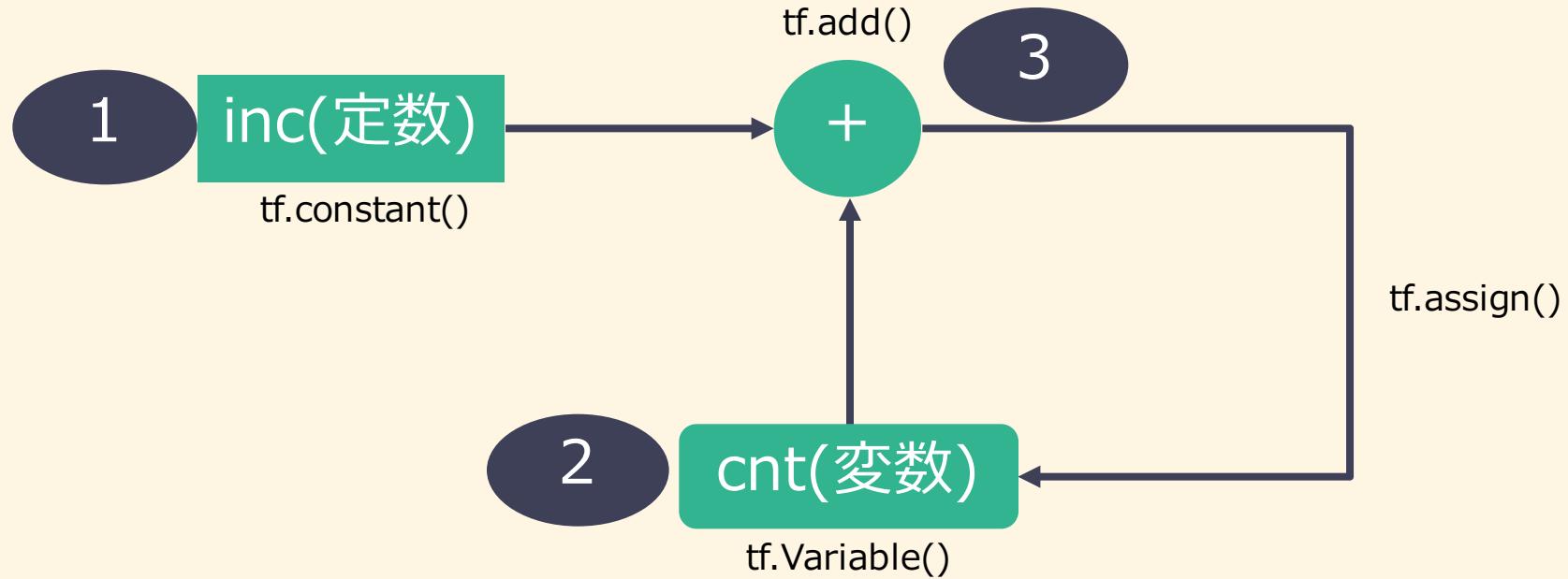
通常のプログラミング言語の変数などと同じように
代入可能な箱として変数がある

変数(Variable)



通常のプログラミング言語の変数などと同じように
代入可能な箱として変数がある

変数(Variable)



通常のプログラミング言語の変数などと同じように
代入可能な箱として変数がある

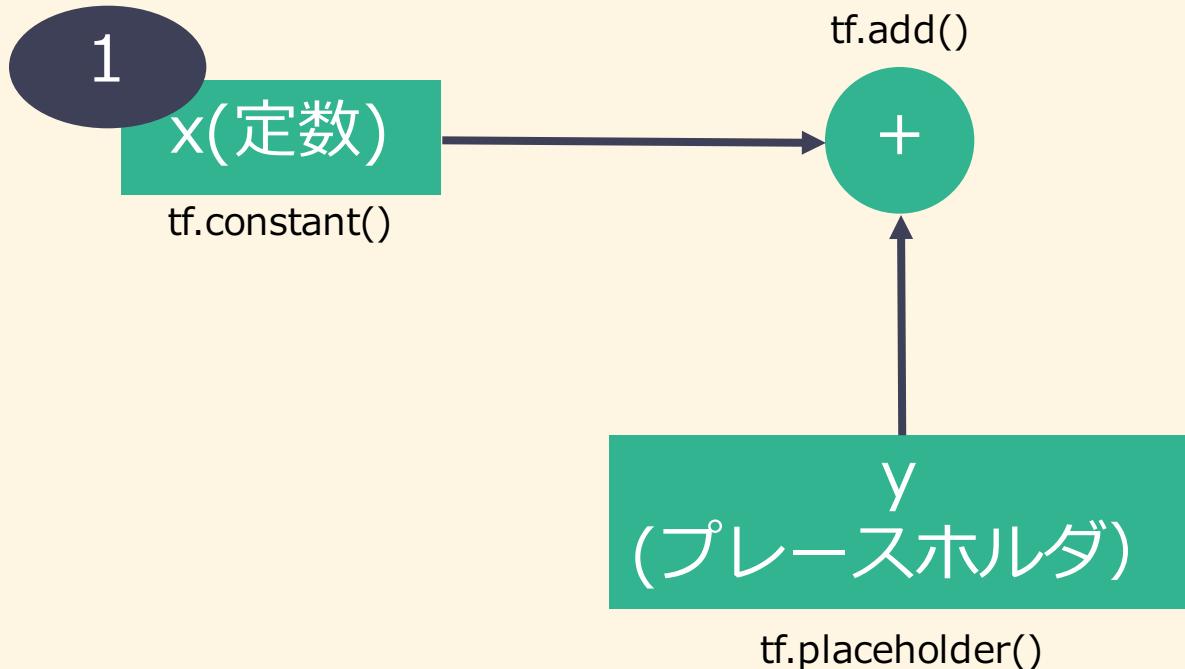
ここだけ押さえればTensorFlowが 分かる簡単な例

- 例 1 : 足し算
 - 演算 (operation)
- 例 2 : カウントアップ
 - 変数 (Variable)
- 例 3 : 入力値をいろいろと変える
 - プレースホルダー (Placeholder)
- 例 4 : セッションを使う
 - セッション (Session)
- 例 5 : 高ランクなテンソルの演算
 - テンソル (Tensor)

いろんな値を入力する

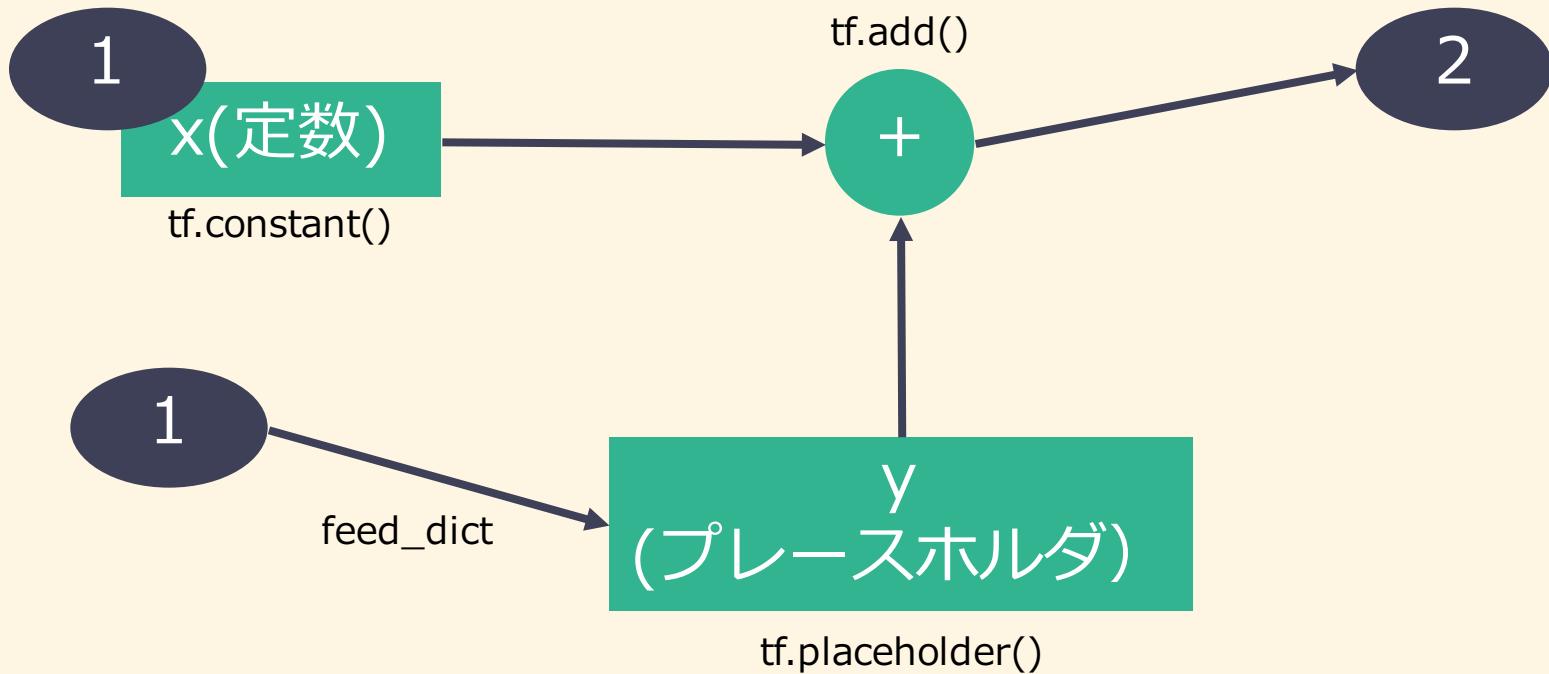
```
import tensorflow as tf\n\nx = tf.constant(1, name="x")\n# プレースホルダーという箱を用意する\ny = tf.placeholder(tf.int32, name="y")\n\nadd_op = tf.add(x, y)\n\nwith tf.Session() as sess:\n    sess.run(tf.initialize_all_variables())\n    print(sess.run(add_op, feed_dict={y:1}))\n    print(sess.run(add_op, feed_dict={y:3}))
```

プレースホルダー(Placeholder)



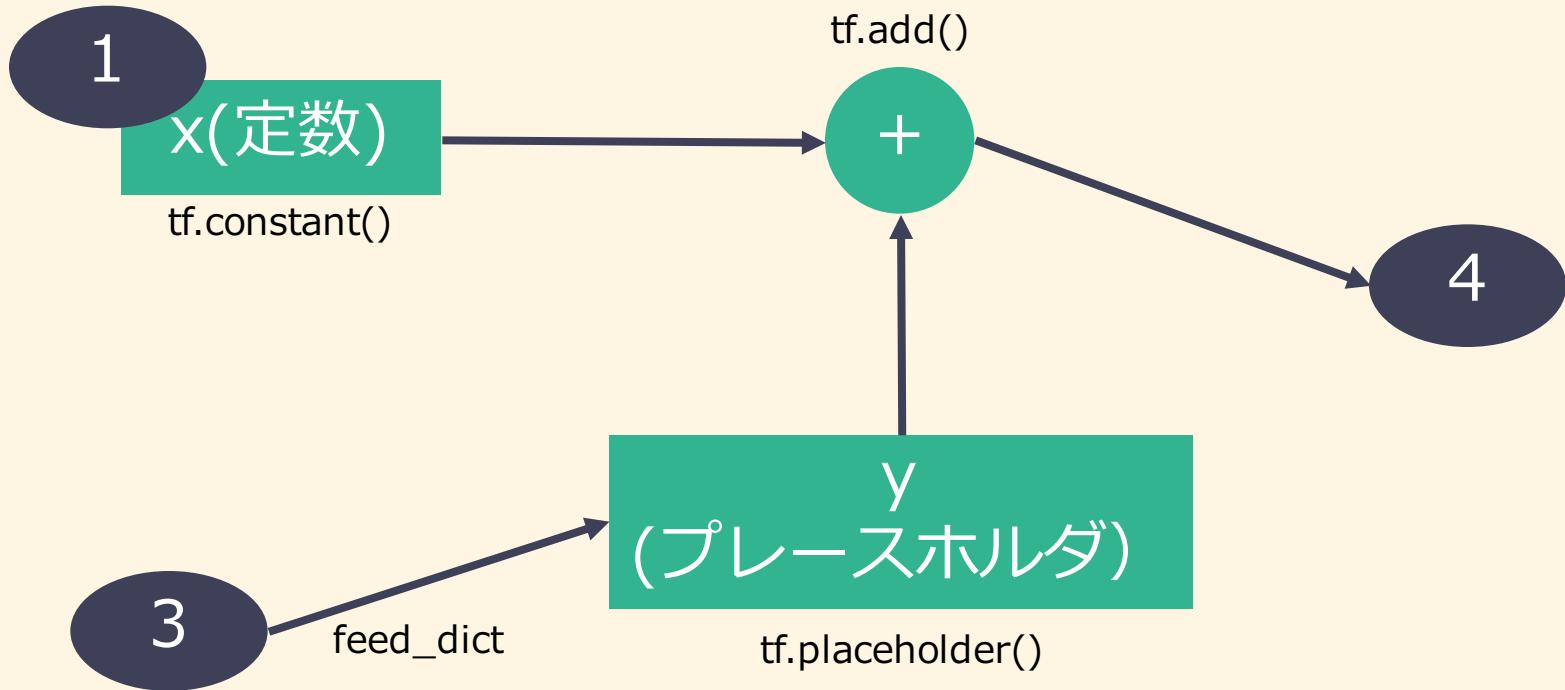
あらかじめ箱を作つておいて
実行時に好きな値を入力させる

プレースホルダー(Placeholder)



あらかじめ箱を作つておいて
実行時に好きな値を入力させる

プレースホルダー(Placeholder)



あらかじめ箱を作つておいて
実行時に好きな値を入力させる

ここだけ押さえればTensorFlowが 分かる簡単な例

- 例 1 : 足し算
 - 演算 (operation)
- 例 2 : カウントアップ
 - 変数 (Variable)
- 例 3 : 入力値をいろいろと変える
 - プレースホルダー (Placeholder)
- 例 4 : セッションを使う
 - セッション (Session)
- 例 5 : 高ランクなテンソルの演算
 - テンソル (Tensor)

実行環境を分ける

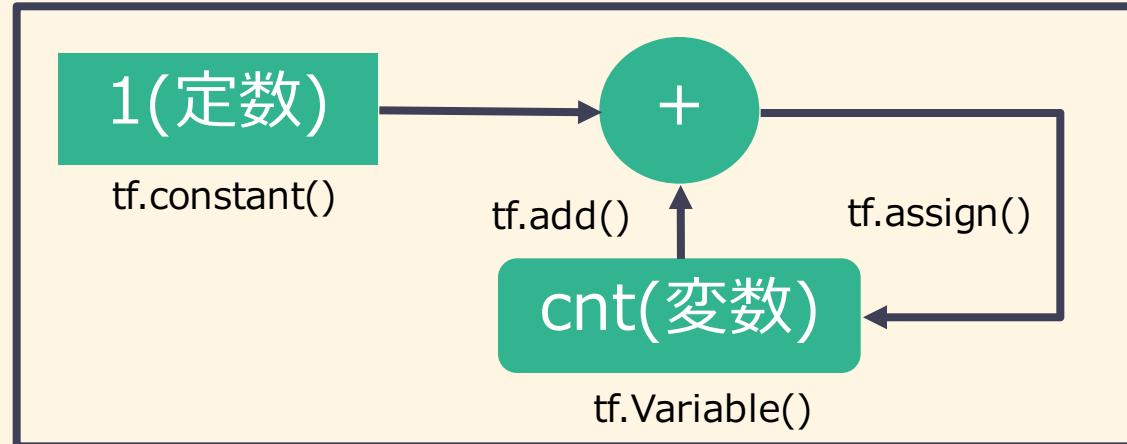
```
add_op = tf.add(cnt, inc)
up_op = tf.assign(cnt, add_op)

# 同じグラフで実行
with tf.Session() as sess:
    sess.run(tf.initialize_all_variables())
    print(sess.run(up_op))
    print(sess.run(up_op))
    print(sess.run(up_op))

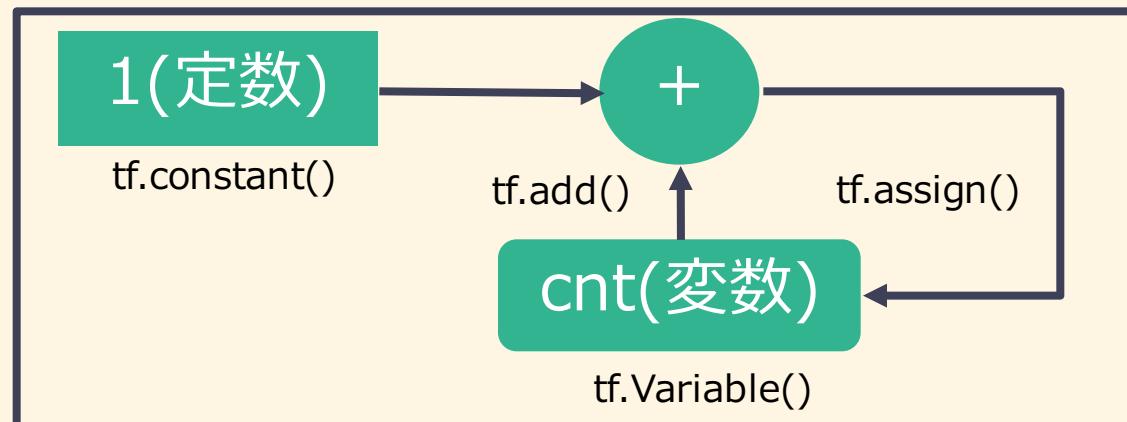
# 同じグラフでもセッションを分けると異なる実行環境となる
with tf.Session() as sess:
    sess.run(tf.initialize_all_variables())
    print(sess.run(up_op))
    print(sess.run(up_op))
    print(sess.run(up_op))
```

セッション

tf.Session()



tf.Session()

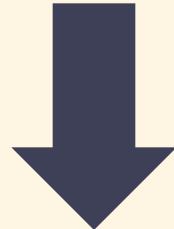


セッションにより
グラフの実行環境が
まるっと独立する
名前空間のようなもの

ここだけ押さえればTensorFlowが 分かる簡単な例

- 例 1 : 足し算
 - 演算 (operation)
- 例 2 : カウントアップ
 - 変数 (Variable)
- 例 3 : 入力値をいろいろと変える
 - プレースホルダー (Placeholder)
- 例 4 : セッションを使う
 - セッション (Session)
- 例 5 : 高ランクなテンソルの演算
 - テンソル (Tensor)

ここまで話題を
多次元の値に拡張



テンソル (Tensor)
のランクを大きくする

テンソル

今までの話はここ

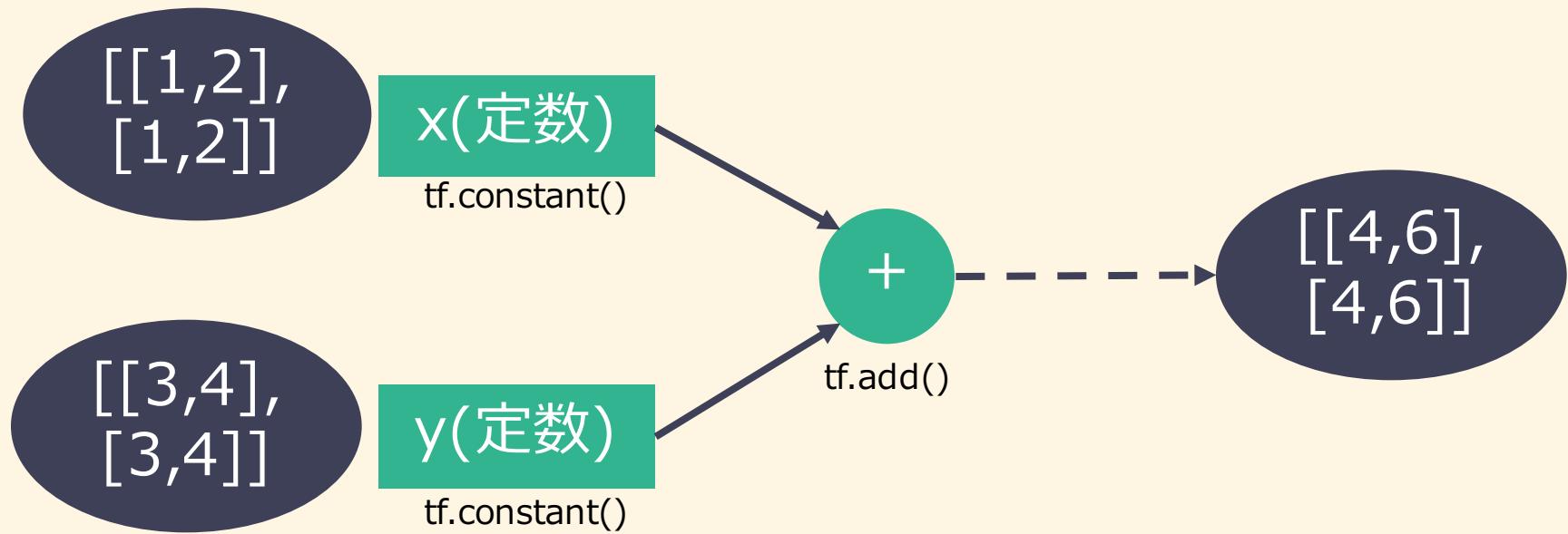
ランク	どうなる
0	スカラ（要はただの数値）
1	ベクトル（配列）
2	行列（2次元配列）
3	3次元配列（行列に厚みがある感じ）
n	n次元配列（イメージできない世界）

要は取り扱うデータ構造のことだと思ってください

多次元版の足し算

```
import tensorflow as tf\n\n# ランク2のテンソル（行列）の定義\nx = tf.constant([[1,2], [1,2]], tf.int32, name="x")\ny = tf.constant([[3,4], [3,4]], tf.int32, name="y")\n\nadd_op = tf.add(x, y)\n\nwith tf.Session() as sess:\n    print(sess.run(x))\n    print(sess.run(y))\n    # 行列の足し算\n    print(sess.run(add_op))
```

入力となるテンソルが違うだけ

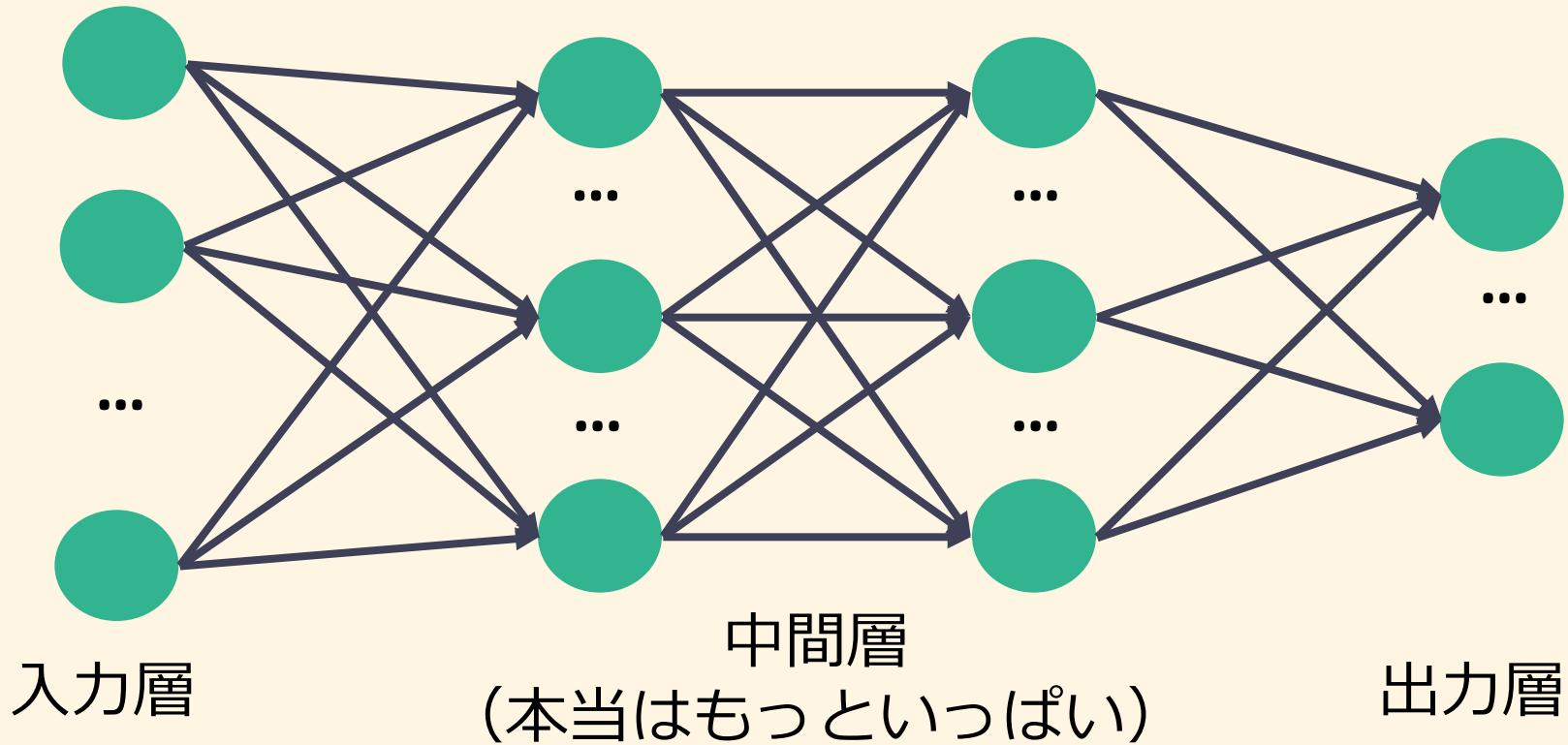


演算がグラフのノードとなる（この場合は加算の演算）

TensorFlowで押さえるべき基本
は以上です

このパラダイムに従って深層（機械）学習のアルゴリズムを記述する必要があります

深層学習について



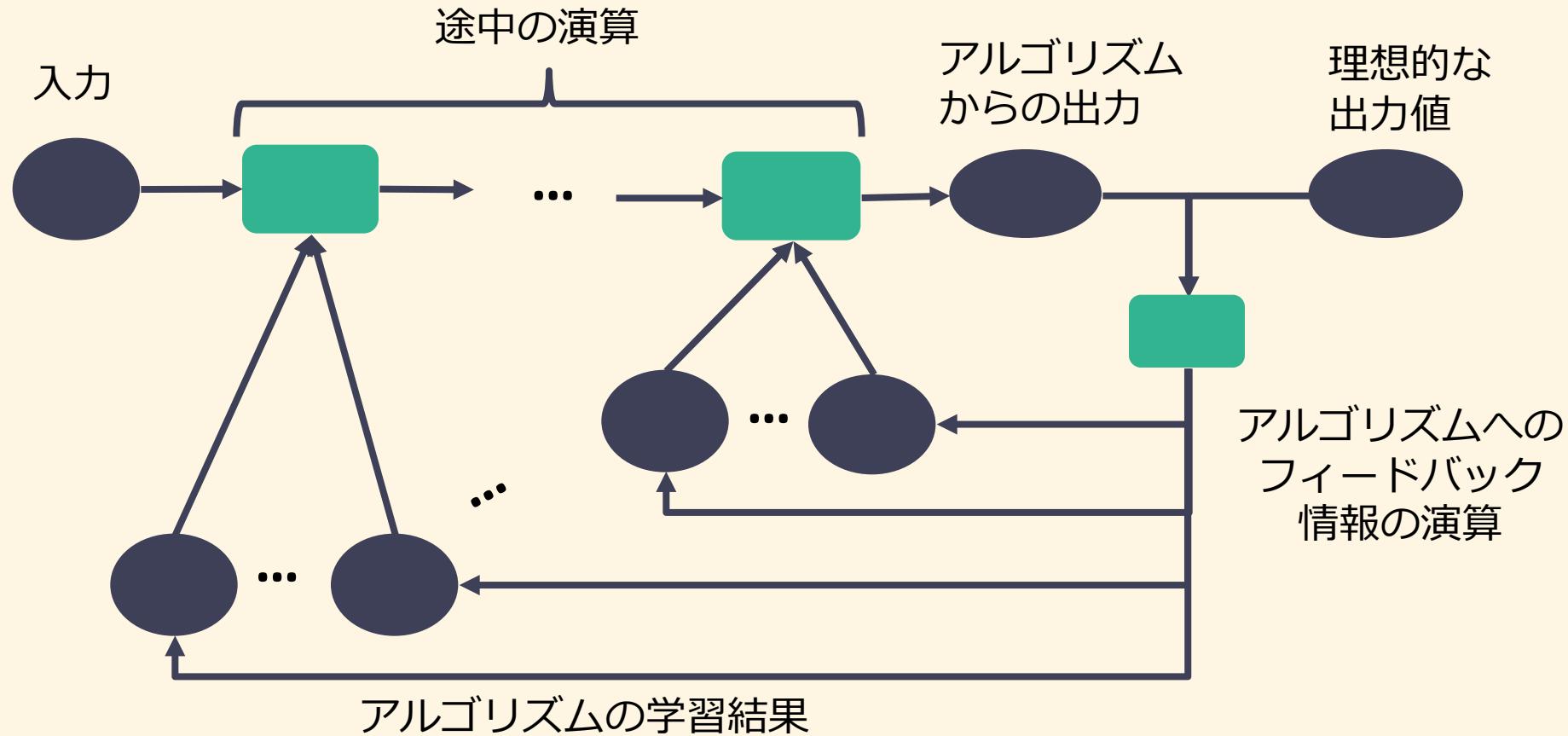
ディープラーニング、深層学習とは、多層構造の
ニューラルネットワークを用いた機械学習である。

Wikipediaより

ニューラルネットワークの
グラフのノード一箇一箇が
TensorFlowの
グラフのノードになって…

と、思いがちなので
若干ややこしい

TensorFlowでやる場合の深層学習 (正確には機械学習全般)



例えば中間層が3層の
ニューラルネットワークだと
こうなる

```
# 入力層
x = tf.placeholder(tf.float32, [None, 4], name='input')

# 第1層
W1 = tf.Variable(tf.truncated_normal([4, 10], stddev=0.5, name='weight1'))
b1 = tf.Variable(tf.constant(0.0, shape=[10], name='bias1'))
h1 = tf.nn.relu(tf.matmul(x,W1) + b1)

# 第2層
W2 = tf.Variable(tf.truncated_normal([10, 20], stddev=0.5, name='weight2'))
b2 = tf.Variable(tf.constant(0.0, shape=[20], name='bias2'))
h2 = tf.nn.relu(tf.matmul(h1,W2) + b2)

# 第3層
W3 = tf.Variable(tf.truncated_normal([20, 10], stddev=0.5, name='weight3'))
b3 = tf.Variable(tf.constant(0.0, shape=[10], name='bias3'))
h3 = tf.nn.relu(tf.matmul(h2,W3) + b3)

# 出力層
W4 = tf.Variable(tf.truncated_normal([10, 3], stddev=0.5, name='weight4'))
b4 = tf.Variable(tf.constant(0.0, shape=[3], name='bias4'))
y = tf.nn.softmax(tf.matmul(h3,W4) + b4)
```

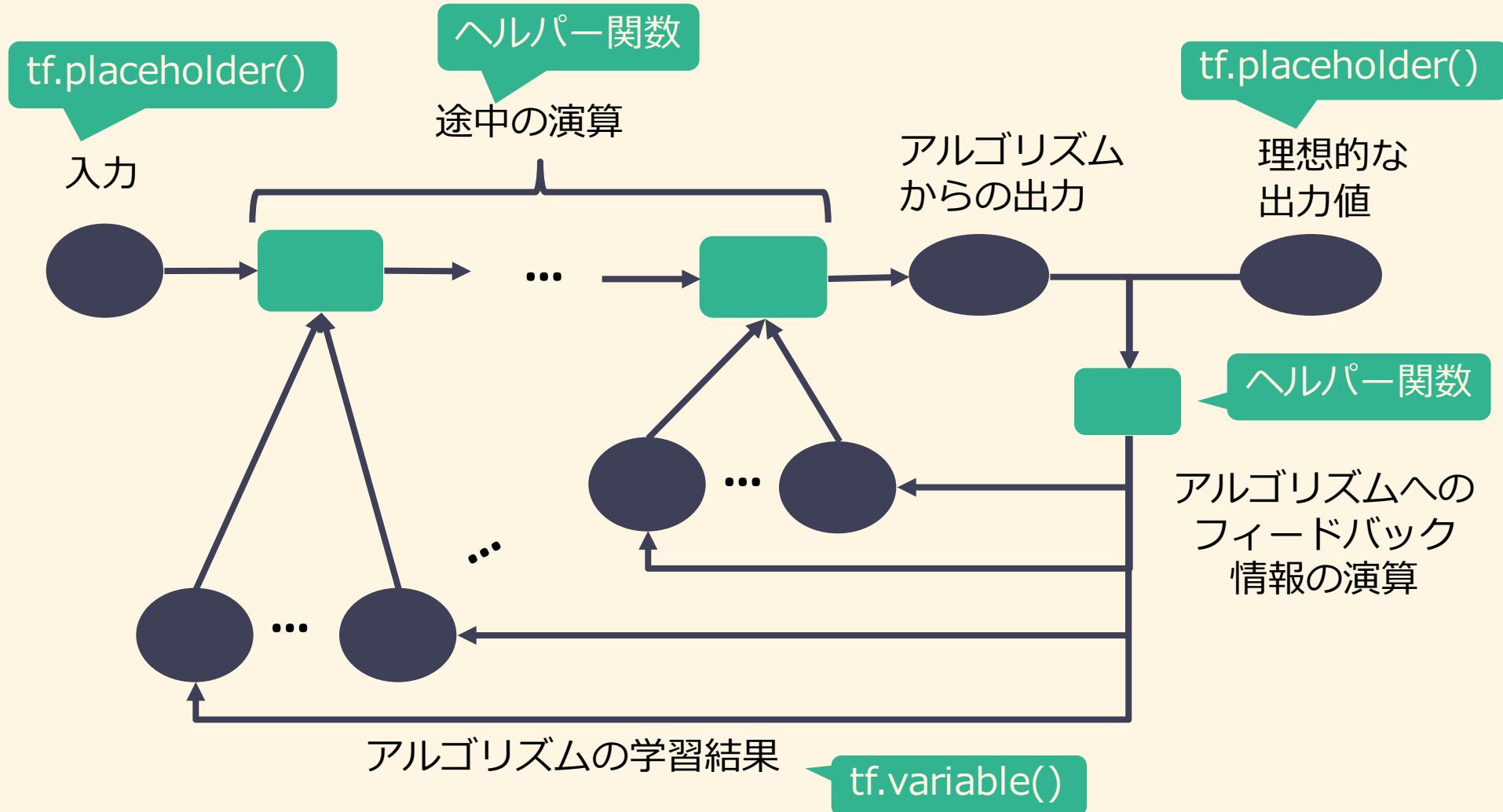
```
# 理想的な出力値
y_ = tf.placeholder(tf.float32, [None, 3], name='teacher_signal')

# 理想的な出力値との比較
cross_entropy = tf.reduce_mean(-tf.reduce_sum(y_ * tf.log(y), reduction_indices=[1]))

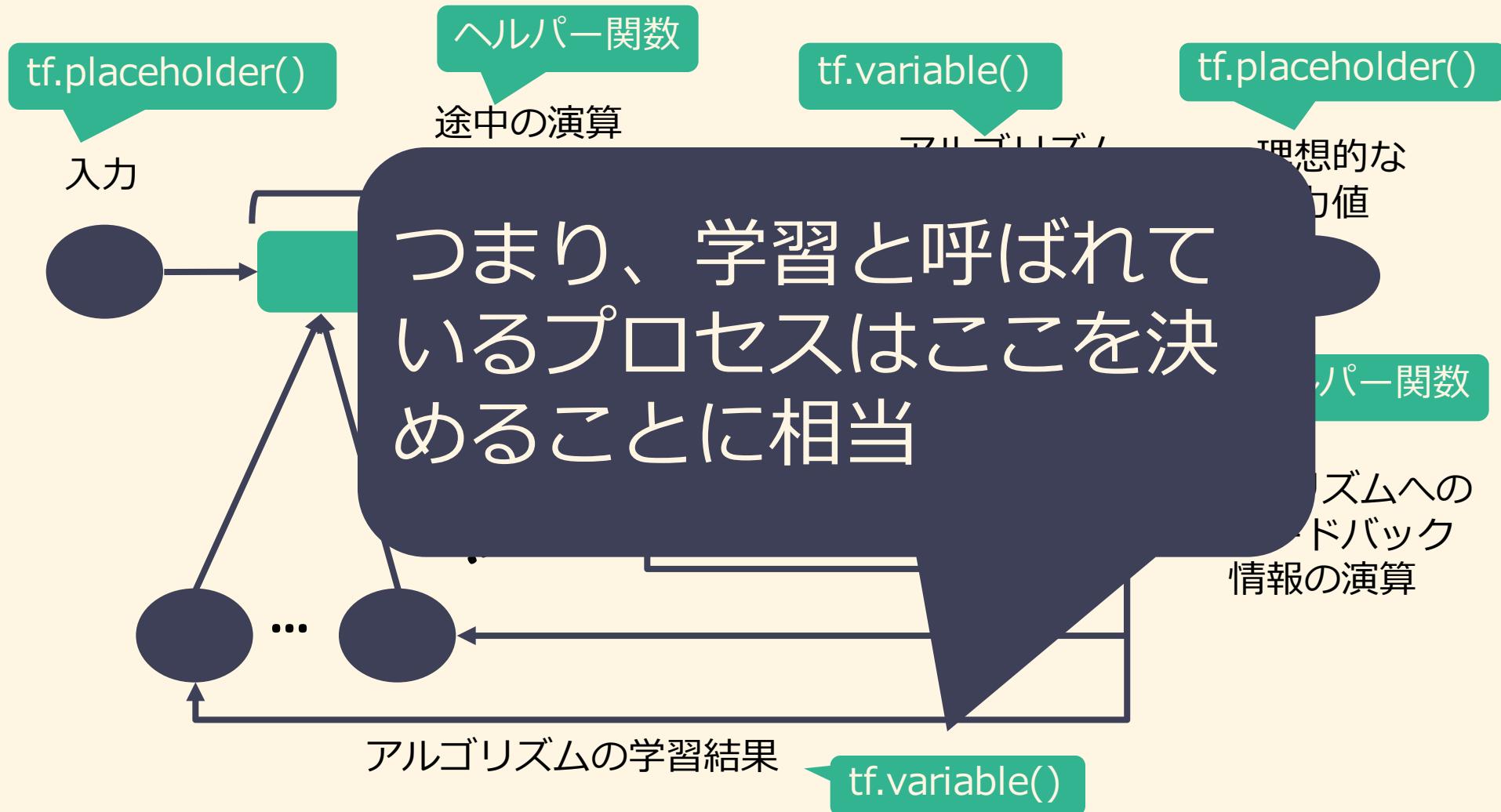
# 学習処理と呼ばれているもの
train_step = tf.train.GradientDescentOptimizer(0.01).minimize(cross_entropy)

with tf.Session() as sess:
    sess.run(tf.initialize_all_variables())
    # 学習処理自体の実行
    for i in range(1000):
        sess.run(train_step, feed_dict={x: train_x, y_: one_hot_labels(train_y)})
```

TensorFlowでやる場合の深層学習 (正確には機械学習全般)



TensorFlowでやる場合の深層学習 (正確には機械学習全般)





深層（機械）学習は時間かかるけど
学習済みのモデルは小さくなるし、
新しいデータの適用する時は早い云々…

TensorFlowの文脈では…



学習とは…

- ・良さげなtf.Variableの値を探索すること

学習済みのモデルとは…

- ・グラフの構造そのもの
- ・グラフ構造に含まれるtf.Variableの値



学習済みのモデルの適用とは…

- ・tf.Variableが決まった後のグラフに
データを通すこと

いざ取り組もうとすると
TensorFlowのパラダイムを意識しつつ
使っているヘルパー関数の
意味を理解した上で
プログラミングする必要があり
けっこう敷居が高い

TensorFlowに関する事実（再掲）



Pythonをインターフェースとした深層学習に特化した便利ツール



Python版だと機械学習のためのヘルパー関数が多数提供された機械学習と相性の良いデータフロープログラミングのフレームワーク
※当然、深層学習もターゲットにしているので相性は良い

結果、こういう評価になりがち

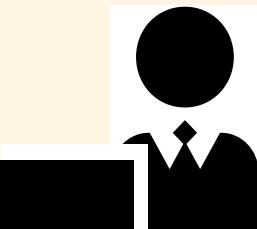
データサイエンティスト評



Chainerとかと比べて、セッションとかよくわからない概念が一杯出てきて使いにくいくらいですね

そもそも深層学習の一般的な知識がないので、どんな風にグラフを構築したり、ヘルパー関数やパラメータをどう選択すればいいかよくわからないんですよね

エンジニア評



逆にGoogleには
エンジニアリング力の高い数学
がバリバリできる超優秀な方が
沢山いらっしゃるということ

実はこれが今日言いた
かったことの一つ

それはさておき

当然同じような問題を
みんなが抱えているため
こういう解決策が出ています

tf.contrib.learn(旧skflow)

- Pythonの機械学習ライブラリであるScikit-learnのように関数呼び出しの感覚でTensorFlowを使いたい！として始まったプロジェクト
 - グラフなどを意識しないで済む高レベルAPIを提供
- TensorFlowのV0.8で本体に取り込まれた
- 一緒に深層学習以外のアルゴリズムも提供されている
 - 線形分類や線形回帰など

中間層が3層のニューラルネットワークのtf.contrib.learn版



Keras

- ニューラルネットワークに関する高レイヤな記述を行うことができるPythonのライブラリ
 - 要は深層学習用のDSL的なもの
 - 記述スタイルはChainerっぽい
- バックエンドとしてTheanoやTensorFlowを選択可能なのでTensorFlow専用というわけではない
- TensorFlow特有の知識は必要ない

中間層が3層のニューラルネットワークのKeras版

```
# モデルの定義
model = Sequential()
```
隠れ層の定義
1層目
model.add(Dense(input_dim=4, output_dim=10))
model.add(Activation('relu'))
2層目
model.add(Dense(input_dim=10, output_dim=20))
model.add(Activation('relu'))
3層目
model.add(Dense(input_dim=20, output_dim=10))
model.add(Activation('relu'))
```
# 出力層
model.add(Dense(output_dim=3))
model.add(Activation('softmax'))
```
モデルに関する種々の定義
model.compile(loss = 'sparse_categorical_crossentropy',
 optimizer = 'adagrad',
 metrics = ['accuracy'])
モデルの学習
model.fit(iris.data, iris.target, nb_epoch = 2000, batch_size = 1)
```

# これなら、こういう評価になる？？

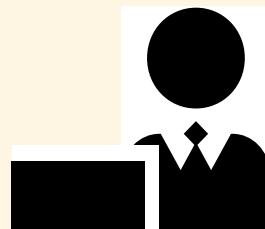
## データサイエンティスト評（想像）



大規模データも分散処理で対応できるし、深層学習の処理の記述に集中できます。（想像）

深層学習とかはよくわからないですが、関数を呼び出す時のパラメータを色々いじるとなんとなく動くところまでは持っていくので楽ですね。（想像）

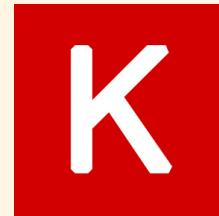
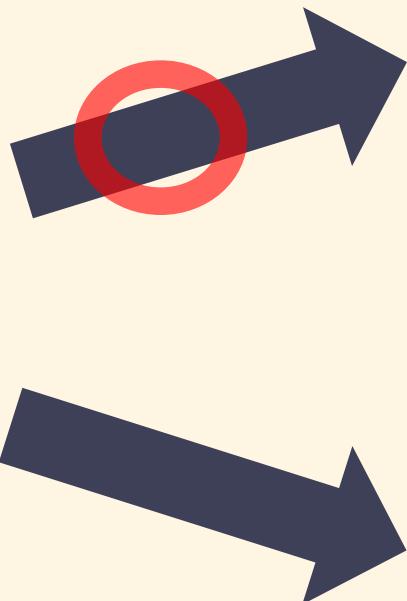
## エンジニア評（想像）



# TensorFlowとKeras



データサイエンティスト



TensorFlowを意識しないで  
アルゴリズムに集中できる



生のTensorFlowを  
書くのは辛い

深層学習界隈で  
インターフェースのデ  
ファクトになるのはなん  
なんでしょうね・・・？

TensorFlowの  
話をするなら外せない  
CloudMLの話も少し



# 9/29にPublic Betaに！



Why Google Products Solutions Launcher Pricing Customers Documentation Support Partn



## Google Cloud Machine Learning: now open to all with new professional services and education programs

Thursday, September 29, 2016

Posted by Riku Inoue, Product Manager, Google Cloud Platform

Earlier this year at GCP NEXT, we introduced new [Cloud Machine Learning products](#) with the intention to change the way [businesses operate](#) and [create new customer experiences](#), while deepening the insights derived from data. Today, we want to share how Google aims to help more businesses benefit from the advancements in machine learning, while making it easier for them to use it.

[Google Cloud Machine Learning](#) is now publicly available in beta and can empower all businesses to easily train quality machine learning models at a faster rate. With its powerful distributed training capability, you can train models on terabytes of data within hours, instead of waiting for days. Integrated with [Google Cloud Platform](#) (GCP), Cloud Machine Learning is a fully-managed service that can scale and creates a rich environment across [TensorFlow](#) and cloud computing tools such as [Google Cloud Dataflow](#), [BigQuery](#), [Cloud Storage](#) and [Cloud Datalab](#).

出典：<https://cloud.google.com/blog/big-data/2016/09/google-cloud-machine-learning-now-open-to-all-with-new-professional-services-and-education-programs>

# CloudMLでできること

- TensorFlowによる学習をクラウド上で実行できる
  - GCS経由でのデータのやり取りなどのお作法がある
  - マシンパワーがたくさん使えるので分散処理やパラメータの自動チューニングなどができる！
- 学習済みのモデルをサービスとしてクラウド上で提供できる
  - gcloud経由
  - RestAPI経由

# CloudMLでできること

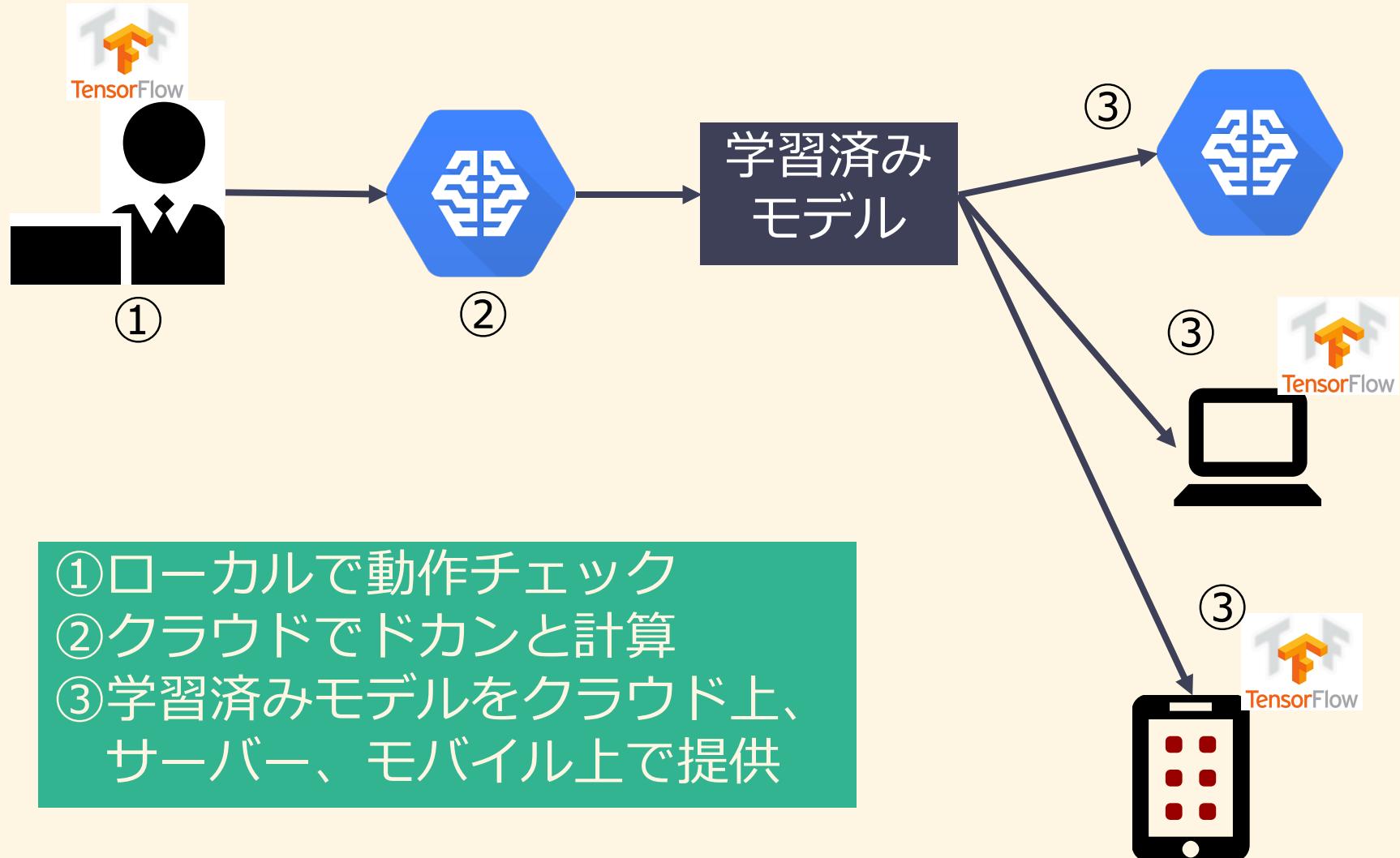
- TensorFlowによる学習をクラウド上で実行できる
  - GCS経由でのデータのやり取りなどのお作法がある
  - マシンパワーがたくさん使えるので分散処理やパラメータの自動チューニングなどができる！
- 学習済みのモデルをサービスとしてクラウド上で提供でTensorFlowのチュートリアルを
  - gcloud 手元のPCで動かしてみると、この領域での
  - REST GPUやクラウドの重要性が分かります

# チュートリアルでこうなる



出典：<http://gadgetm.jp/I0000291>

# CloudMLも絡めたワークフロー



## 2. 公式チュートリアルコードの 解説