Androidで動かす
はじめてのDeepLearning

narrative nights株式会社
三好康祐

Jinnan Android Meetup vol.1

今日やること

1. DeepLearningのデモを動かしてみる
2. Javaで(Deepでない)ニューラルネットでデモを動かしてみる
3. TensorFlowでニューラルネットのデータを書き出してみる

ここまで

4. TensorFlowでDeepLearningで書き出したデータに差し替えてみる
5. 4.でやった内容の説明

できる限り専門用語なしで
説明します

https://github.com/miyosuda/intro-to-dl-android

まずこちらからソース一式をダウンロード.
中にAndroidStudioのプロジェクトが二つ.

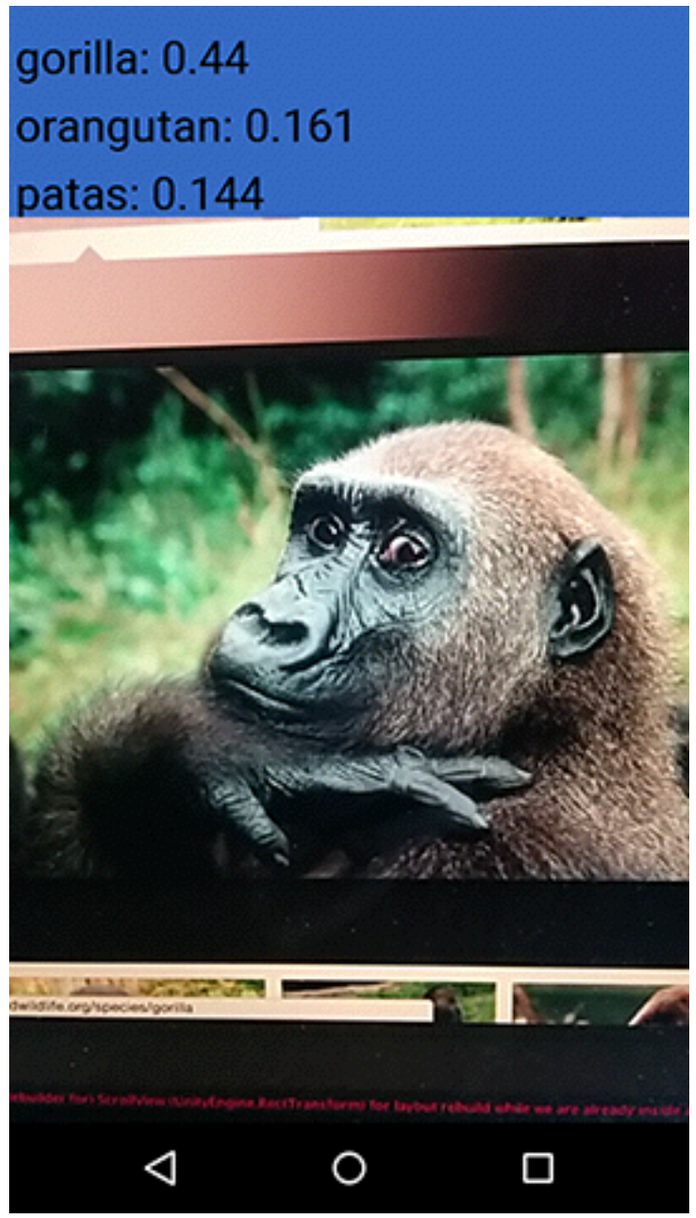
デモをビルドして実行

デモ1

- **ImageClassification**

  - TensorFlowを使った画像認識

  - カメラを向けるとリアルタイムに1000種類の中から一番近いと思われるものを提示する



デモ2

- HandWriting

- TensorFlowを使った手書き数字認識

- 0〜9の数字の認識

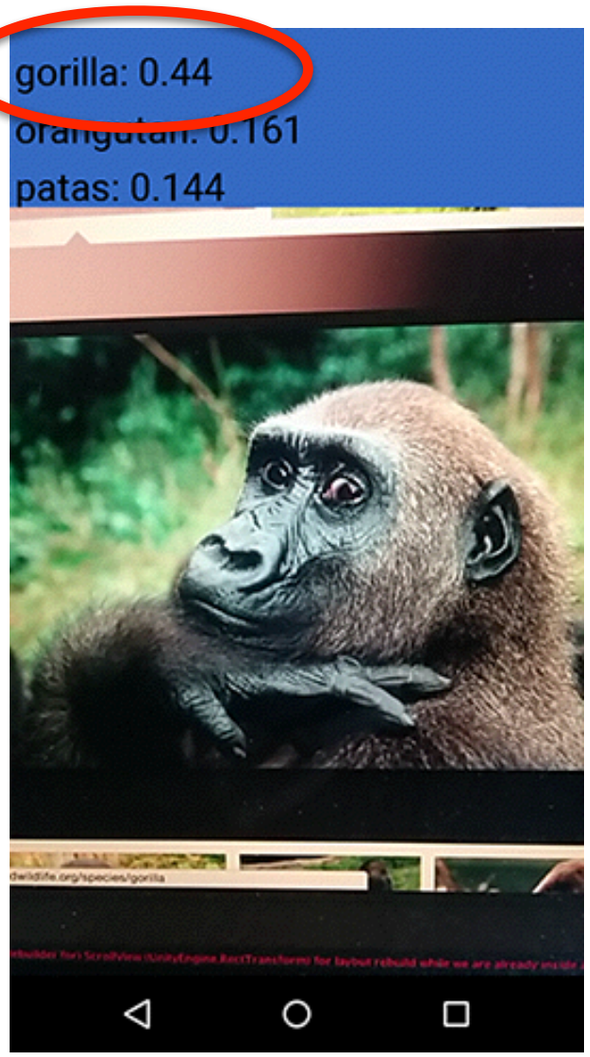
ニューラルネットとは

float[] ->　[ニューラルネット] -> **float[]**

floatの配列を入れるとfloat
の配列が出てくる

float[224*224*3] → ニューラルネット → **float[1000]**

224ピクセルx
224ピクセルx
RGB3色



キツネ = 0.00002
電車　= 0.00002
ゴリラ = 0.44
オラウータン= 0.161
…
…
ノートパソコン= 0.034
カルボナーラ　= 0.0001

1000個の候補の中から
一番値が大きかったのが
ゴリラ

float[28*28] → ニューラルネット → **float[10]**

28ピクセルx
28ピクセル
グレースケール

0 = 0.00002
1 = 0.00002
2 = 0.03
3 = 0.95
4 = 0.02
…
…
9 = 0.012

10個の値の中から
一番大きいのを提示

- “0.72” -> ただの数字
- {0.72, 0.45, 0.11} -> 配列
- { {0.72, 0.45, 0.11},
  {0.23, 1.24, 5.23},
  {2.23, 0.12, 1.12} } -> 二次元配列

- “0.72” -> 0階テンソル
- {0.72, 0.45, 0.11} -> 1階テンソル
- { {0.72, 0.45, 0.11},
  {0.23, 1.24, 5.23}, -> 2階テンソル
  {2.23, 0.12, 1.12} }

… -> 3階テンソル

…

…

テンソル → テンソルに対するいろんな計算 → **テンソル**

Javaでニューラルネットを動かす

- **HandWriting** プロジェクトソース内

- jp/narr/tensorflowmnist/MainActitivity.java
  42行目〜46行目

```
// TensorFlowを使った文字認識を行う場合
//private TensorFlowDigitDetector mDetector = new TensorFlowDigitDetector();

// Javaによる文字認識を行う場合
private JavaDigitDetector mDetector = new JavaDigitDetector();
```

- **TensorFlowDigitDetector の行をコメントアウトして、JavaDigitDetectorを動かす様に切り替えます**

・ JavaDigitDetecotor.java

input -> float[784]

28x28(=784)個のピクセル値を0.0～1.0にして、ベタにつなげて1次元の配列にしたもの

これに演算をかけて、10個のfloat値(“0”~”9”のそれぞれのもっともらしさ)を出す

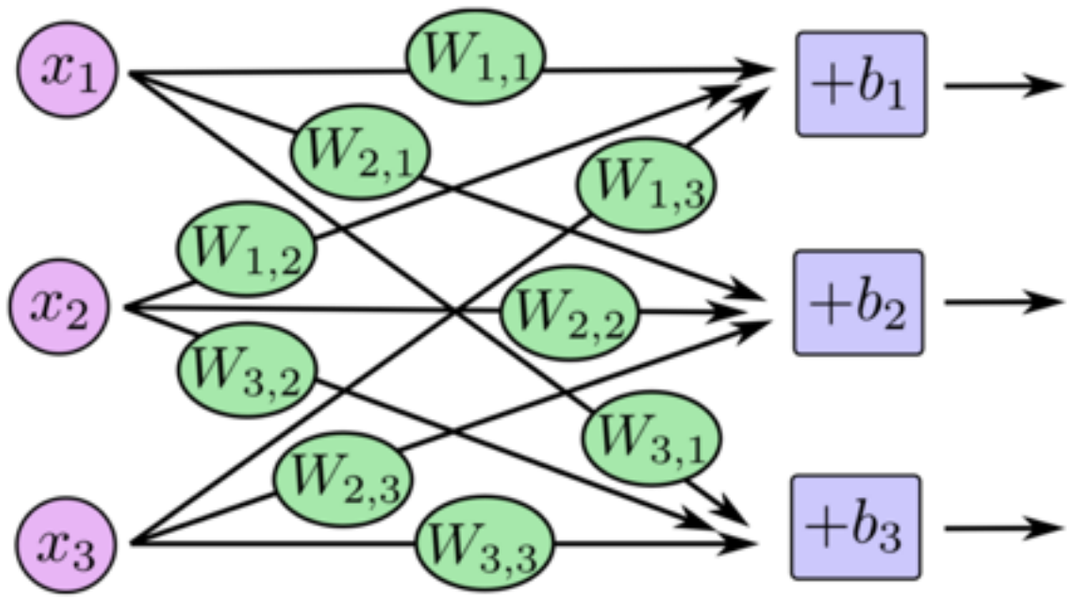
- JavaDigitDetecotor.java

```
// 784x10のweight値
private float[][] weightW = new float[784][10];
// 10個のbias値
private float[] biasB = new float[10];
```

- assets内のテキストファイル(w.csv, b.csv)からロード

```
public int detectDigit(int[] pixels) {
  …
   float[] input = new float[784];
   …

   // 10個の出力値を準備
   float[] output = new float[10];

   // input と weightW の掛け算を行う
   for (int j = 0; j < 784; ++j) {
      for (int i = 0; i < 10; ++i) {
         output[i] += input[j] * weightW[j][i];
      }
   }

   // それに biasB を足す
   for (int i = 0; i < 10; ++i) {
      output[i] += biasB[i];
   }

    …
}
```

output = input * W + b



784個　　10個

- **softMax()**

10個の値を**合計して1.0**になる様にするexp()を使った計算　**(各出力値も0.0~1.0の間になる)**

exp()の結果は必ず0より大きい正の値になる

例:

exp(10) -> 22026.5

exp(5) -> 148.4

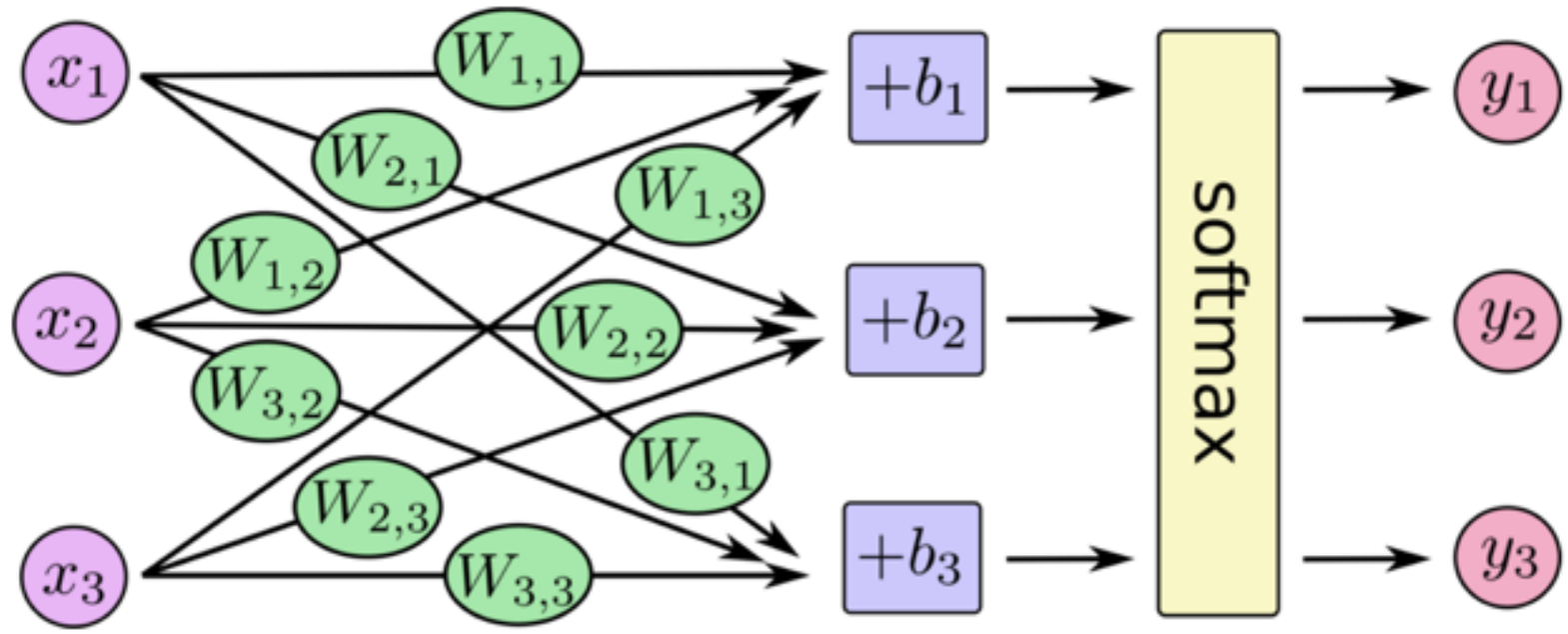
exp(0) -> 1.0

exp(-5) -> 0.00674

exp(-10) -> 0.0000454

```
private float[] softMax(float[] values) {
   // 各値のexp値
   float[] expValues = new float[values.length];

   float expSum = 0.0f; // 各値のexp値の合計
   for (int i = 0; i < values.length; ++i) {
      // 各値のexp値を出す
      float exp = (float) Math.exp(values[i]);
      expValues[i] = exp;
      expSum += exp; // 合計値を加算
   }

   // 合計値で割って、全部のexp値の合計が1になる様にする
   for (int i = 0; i < values.length; ++i) {
      expValues[i] /= expSum;
   }
   // 結果はそれぞれ0.0~1.0の間の値
   return expValues;
}
```

output = softmax(input * W + b)



784個　　10個　　10個

10個の出力の内最大のものが、output[3]なら、
文字が”3”だと推測

```
// 784x10のweight値
private float[][] weightW = new float[784][10];
// 10個のbias値
private float[] biasB = new float[10];
```

次に、ファイルから読み込んだこれらの値を
TensorFlowを使って書き出してみます

MacにTensorFlowをインストールします

https://github.com/miyosuda/intro-to-dl-android/wiki/TensorFlow%E3%81%AEMac%E3%81%B8%E3%81%AE%E3%82%A4%E3%83%B3%E3%82%B9%E3%83%88%E3%83%BC%E3%83%AB%E6%89%8B%E9%A0%86

こちらの手順でインストール

Weightの書き出し

python-scripts/example1.pyを実行します

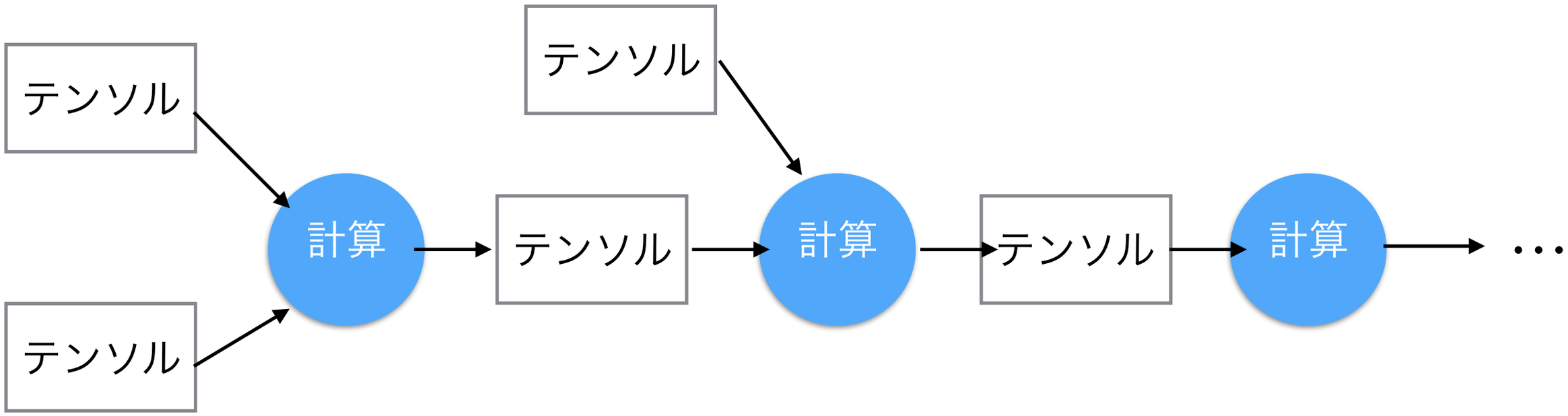
```
$ source ~/tensorflow/bin/activate
(tensorflow)$ cd (#intro-to-dl-on-androidのディレクトリ#)/python-scripts
(tensorflow)$ python example1.py
```

TensorFlowの流れ

1. グラフの定義

2. グラフの実行

グラフ



MNIST

- 訓練用の55000組の手書き画像(28x28ピクセル)と正解データ(0〜9)
- 10000組の確認用データ(画像+正解データ)
- 違いは訓練用と確認用に分けただけ

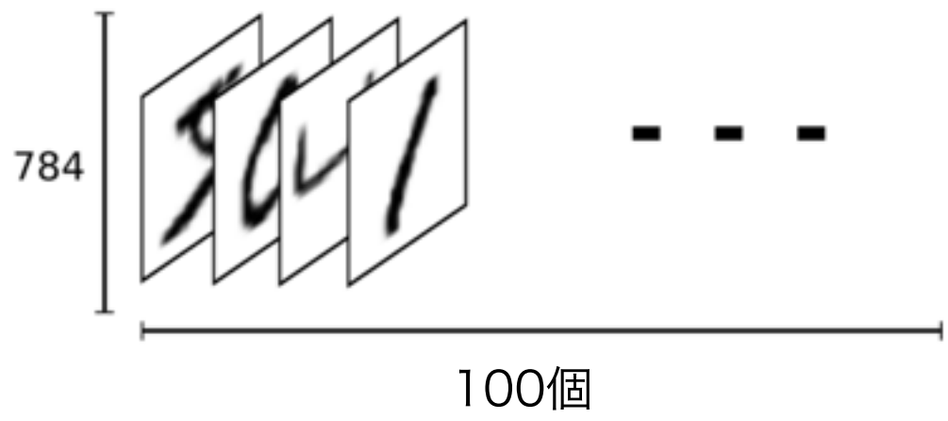
https://github.com/miyosuda/intro-to-dl-android/blob/master/python-scripts/example1.py

```
# [グラフ定義]

# 入力用の値のいれ物(=PlaceHolder)を作成する. ここには画像データをテンソルにしたものが
# 入ってくる.
# Noneは"数が未定"を表す. 学習時はここが100になり、確認時は10000になる.
# なので、学習時は(100x784)のテンソル、確認時は(10000x784)のテンソルになる.
x = tf.placeholder(tf.float32, [None, 784])

# 784x10個の重み. 学習により変化していく.
W = tf.Variable(tf.zeros([784, 10]))

# 10個のBias値. 学習により変化していく.
b = tf.Variable(tf.zeros([10]))

# (x * W + b)の結果をsoftmax関数に入れ、その結果をyとする.
# yは学習時は(100x10)のテンソル. 確認時は(10000x10)のテンソルになる.
y = tf.nn.softmax(tf.matmul(,x W) + b)
```

```
# 損失関数(正解とどれくらいずれているかを表すもの)を定義していく
# y_ は正解データを入れる入れもの.
# Noneとなっているが、学習時にはここが100になり、
# y_は(100, 10)のテンソルとなる.
y_ = tf.placeholder(tf.float32, [None, 10])

# ニューラルネットの出した10個の値と正解の10個の値(正解部分だけが1の配列)を
# もちいて、どれくらいずれていたか、を出す.
# 小さければ小さいほど正解に近かった事を表す値.(合計をひとつのスカラー値として集めたもの)
# 100個分の合計を求める
cross_entropy = -tf.reduce_sum(y_ * tf.log(y))

# 上記のずれを小さくする様に学習させるOptimizerを用意する.
train_step = tf.train.GradientDescentOptimizer(0.01).minimize(cross_entropy)
```

学習時の入力

- 学習時は入力画像(28x28ピクセルのfloat値)を100個分まとめて入れている

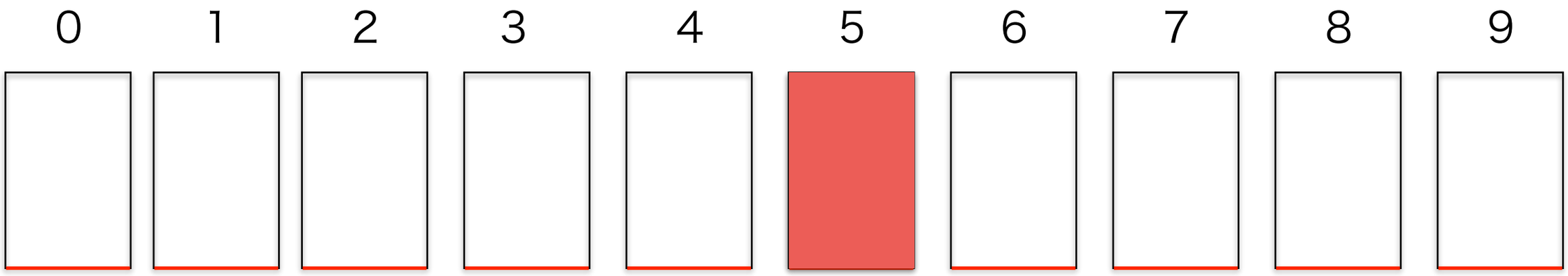


- 出力(float[10])も100個まとめて出てくる

正解データ

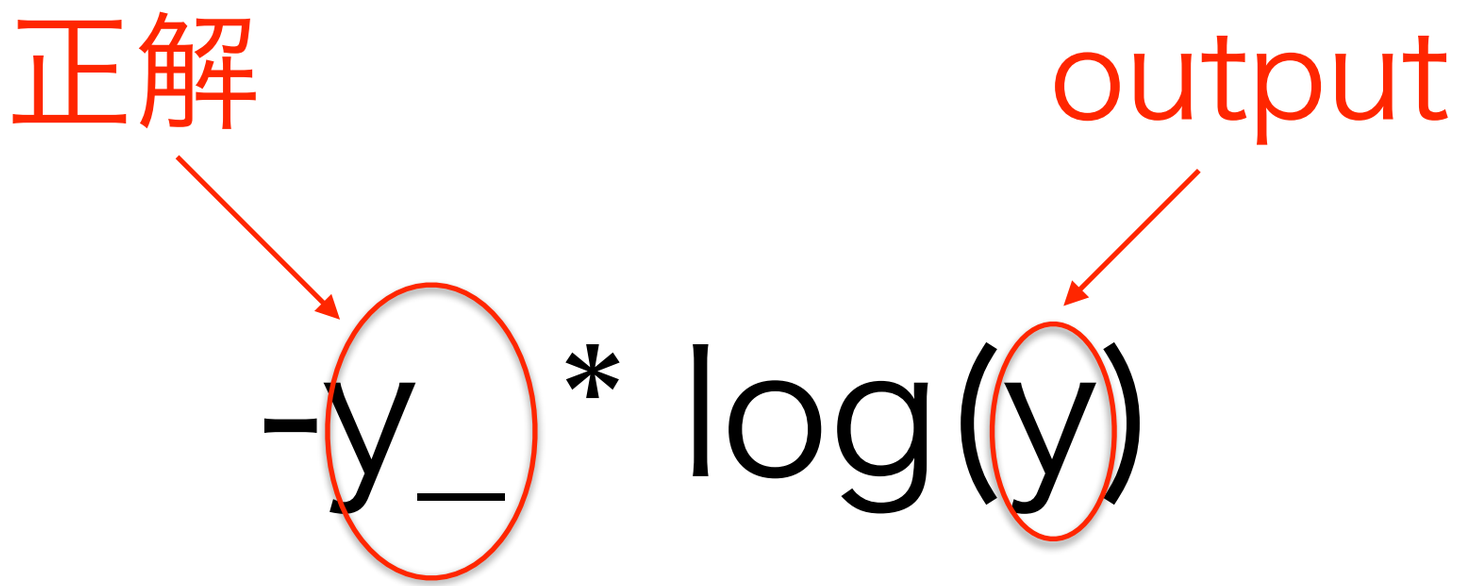
- 正解データ(0~9のどれになるか)もoutputと同じ様に10個の値の配列となっている

例: 正解が5の場合

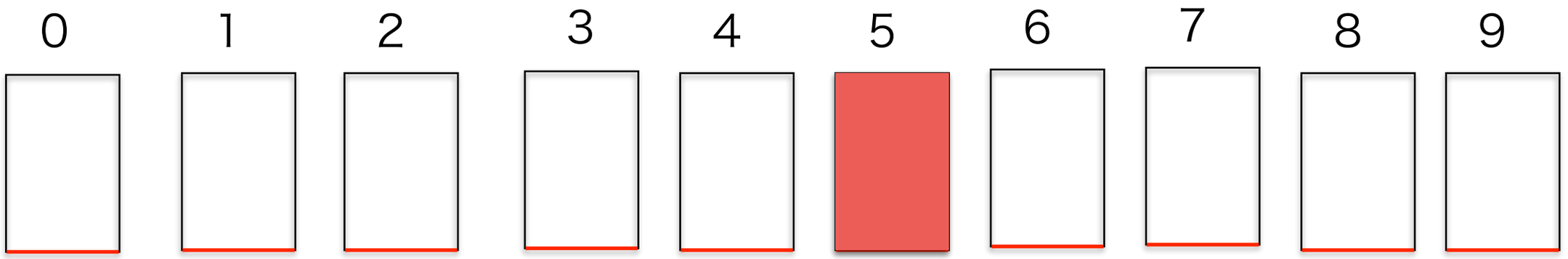


y_ = {0,0,0,0,0,1,0,0,0,0}

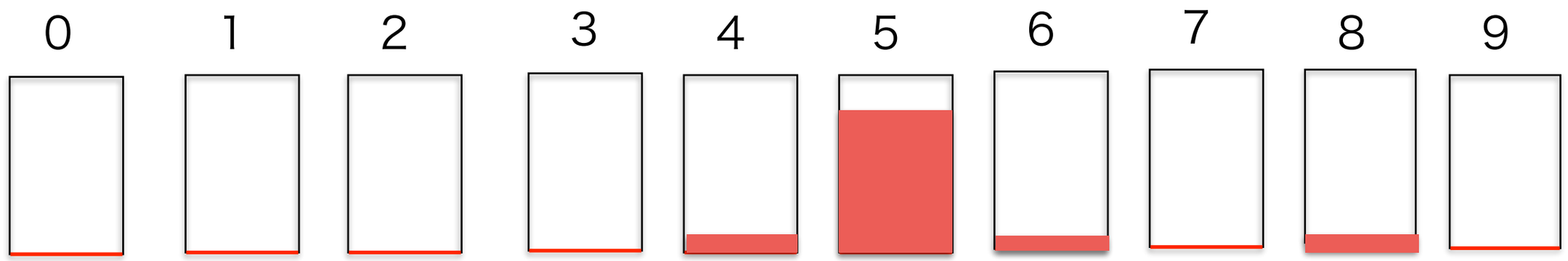
正解と出力のずれ



正解



output



0.0〜1.0の値の**log()**

log(**0.0**)　　-> **-∞**

log(**0.0001**) -> **-9.21**

log(**0.1**)　　-> **-2.3**

log(**0.5**)　　-> **-0.69**

log(**0.9**)　　-> **-0.11**

log(**1.0**)　　-> **0.0**

0.0以下の負の数になる

マイナスを掛けると

-log(**0.0**) -> **∞**

-log(**0.0001**) -> **9.21**

-log(**0.1**) -> **2.3**

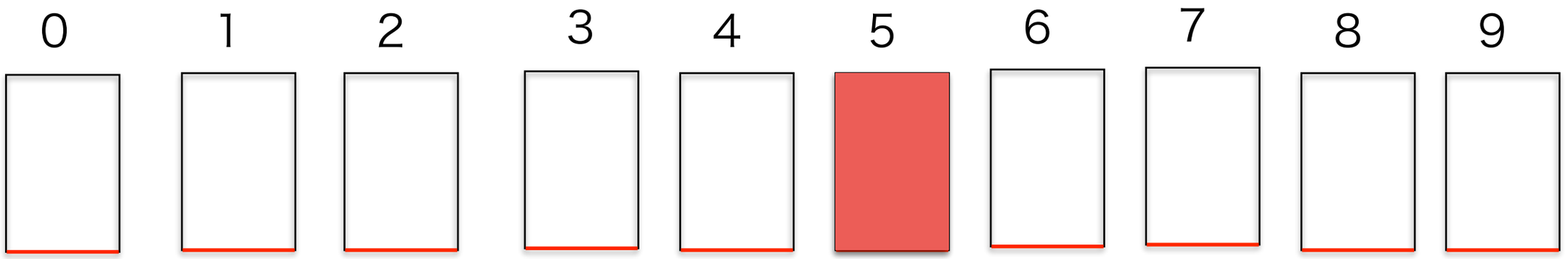
-log(**0.5**) -> **0.69**

-log(**0.9**) -> **0.11**

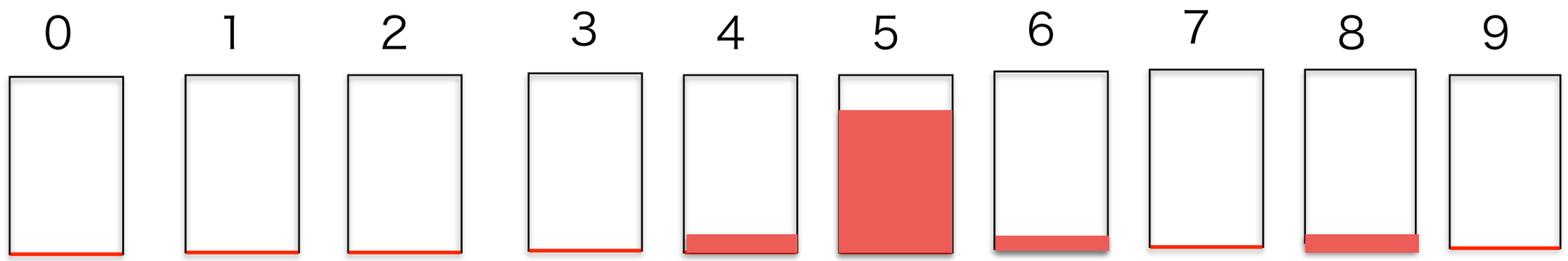
-log(**1.0**) -> **0.0**

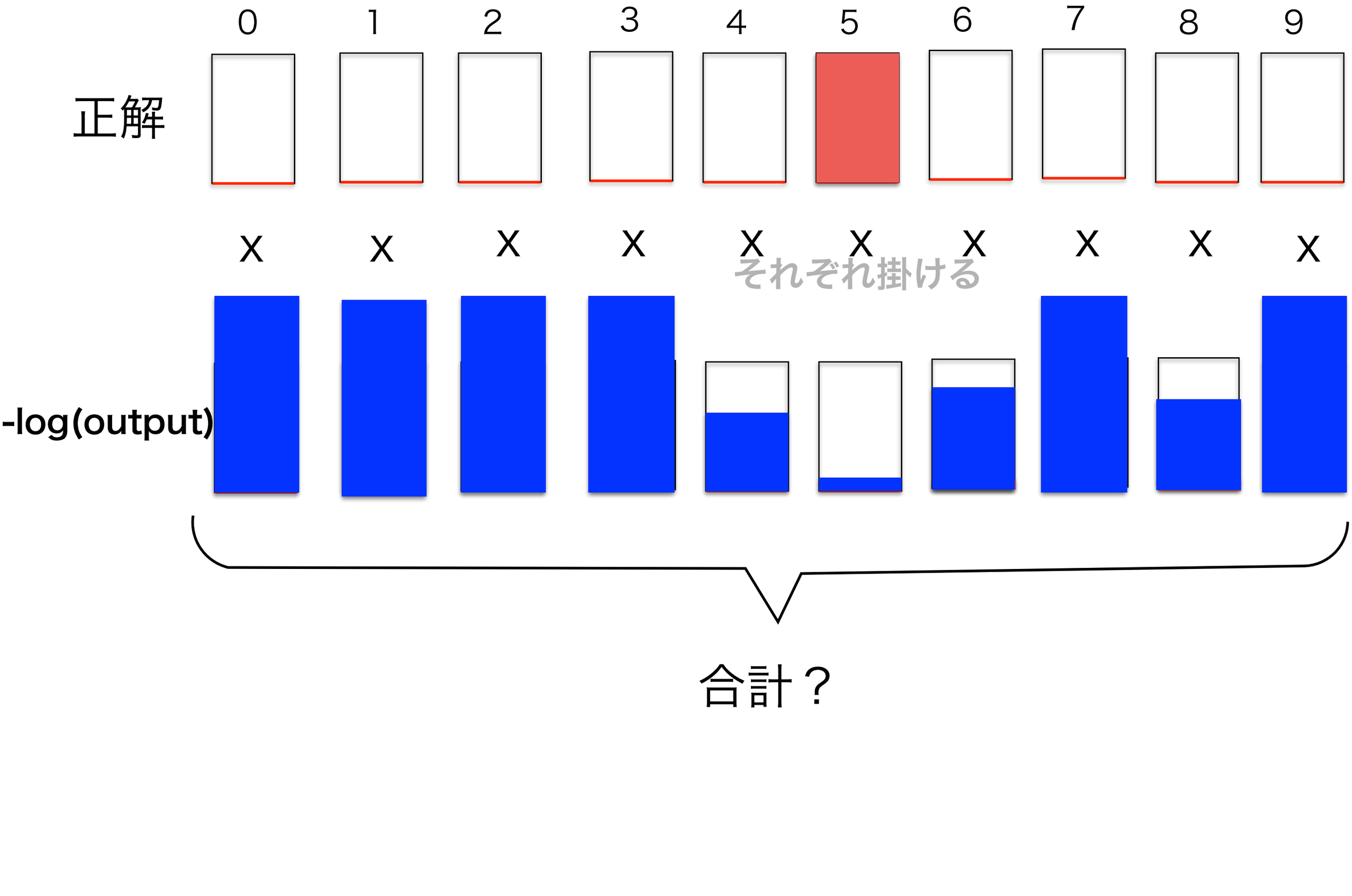
-log(1.0)が最小で0.0

正解



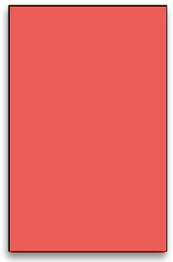
output




0
1
2
3
4
5
6
7
8
9
正解
X
X
X
X
X
X
X
X
X
X
それぞれ掛ける
-log(output)
合計？

ここしか残らない

5

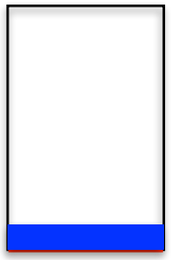


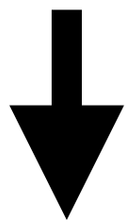
正解

掛ける

X

-log(output)





求めようとしているずれの値

-y_ * log(y)

output[5]が1.0(=正解と完全に一致)ならばこの値は0

output[5]が1.0から小さくなればなるほど、
この値は大きくなる

=>この値が小さければ小さいほどより正解に
近い値をoutputできるニューラルネット

この値を小さくする様に**W**と**b**を調整していく

学習後の**W**と**b**を書き出したものが、
Javaから読み込んで使っていた
weightWと**biasB**

Android用TensorFlow ライブラリの利用

- TensorFlowのAndroid用ライブラリを用いれば、さっきのデモでJavaで書いていた部分を代わりに処理してくれる

- Android側のTensorFlowライブラリが読み込める形式で書き出す必要がある。

- 書き出し用のスクリプト
  python-scripts/example2.py
  (example1.pyと同じものをTensorFlowライブラリ用形式に書き出し)

- グラフ構造とWeightの値をまとめて書き出し/読み込みをする。

jp/narr/tensorflowmnist/MainActitivity.java

```
// TensorFlowを使った文字認識を行う場合
private TensorFlowDigitDetector mDetector = new TensorFlowDigitDetector();
```

```
// Javaによる文字認識を行う場合
//private JavaDigitDetector mDetector = new JavaDigitDetector();
```

TensorFlowDigitDetectorを利用する

TensotFlowDigitDetecotor.java

JNI(c++のnative)側でassetsからニューラルネットの学習済みのデータ**(=グラフ構造とVariableの中身をまとめたもの)**を読みこんで実行

TensotFlowDigitDetecotor.java

データを差し替えるだけで、ニューラルネットの中身をより複雑にしたもの(DeepLearning で学習させたもの)が実行できる

学習&データ書き出し用のスクリプト
python-scripts/example3.py

詳細はこちら

『TesorFlow: Pythonで学習したデータをAndroidで実行』

http://qiita.com/miyosuda/items/e53ad2efeed0ff040606

今回は以上になります