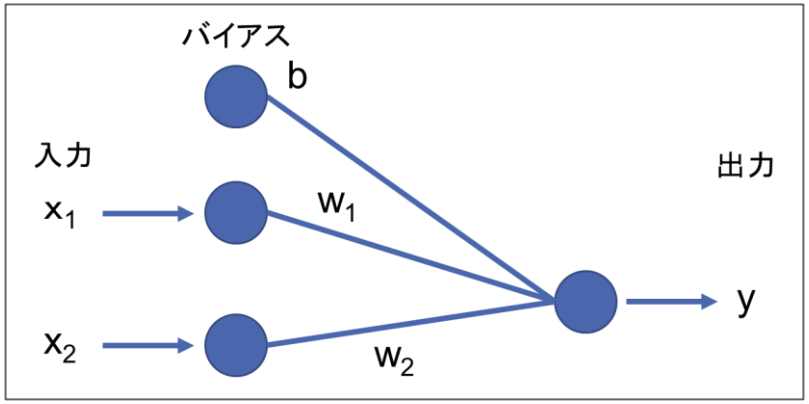
ニューラルネットワークによる分析・分類

単純パーセプトロン

入力と出力から成る最も基本的なニューロンモデル

入力Xに対して重みとバイアスを用いた計算により得られた値と教師データYの誤差を計算し、誤差が小さくなるように重みとバイアスを更新して学習

単純パーセプトロンをKerasで構築し、分類を試みる



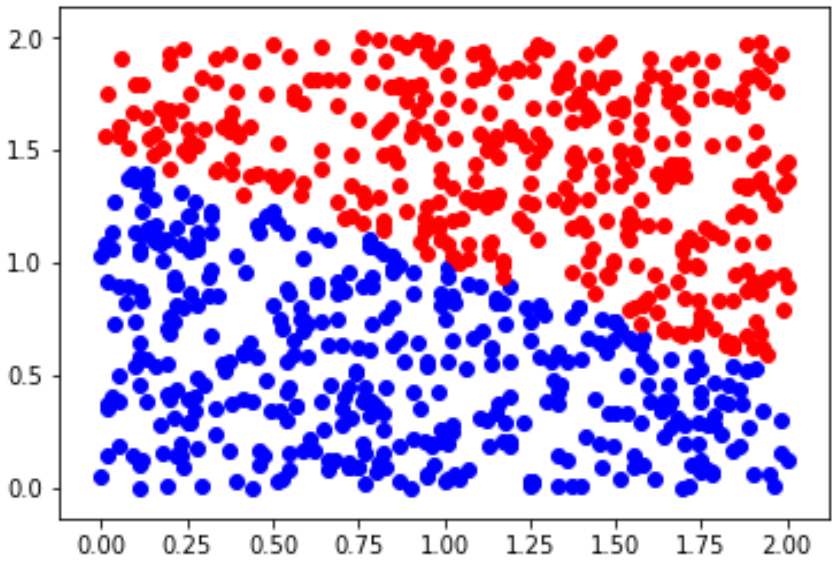
$$y = w_1 x_1 + w_2 x_2 + b$$

準備するデータセット

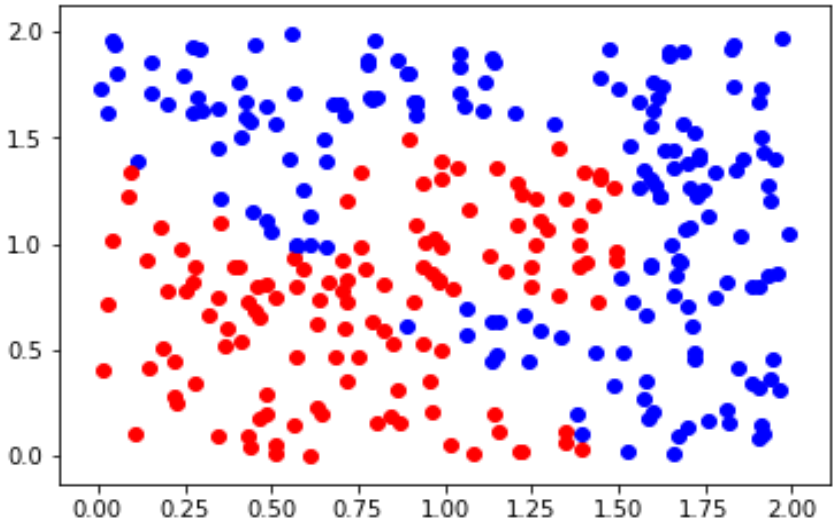
- 学習（訓練）用データセット
  - トレーニングデータセット（Training data set）
  - 入力と正解データのセット
- 学習評価用データセット
  - バリデーションデータセット（Validation data set）
  - 入力と正解データのセット
  - 学習（重みの更新）には使わないデータを評価として使う
  - 汎化性能のチェック
- 推測（予測）用データセット
  - テストデータセット（Test data set、Prediction data set）
  - 入力のみ

演習

2種類のデータに対して分類を行う



線形分離可能な単純なデータ



線形分離不可能な入り組んだデータ

演習前に

データをColabにインポートしておく
「データの読み込みと描画」の4行目でデータフォルダを設定

```
datadir = "/content/drive/My Drive/data/"
```

ソースのデフォルトはGoogle Driveに直接dataフォルダを置いた例となっているので、自分の環境に合わせて書き換える
最後に / を付け忘れないように

Kerasの基本

model: ニューラルネットワークの定義

model.add: レイヤーの追加

model.compile: 定義したネットワークの構築

model.fit: ネットワークの学習

多くの場合、history = model.fit とし、学習結果をhistroyに保存しておいて結果のプロットを行う

公式ドキュメント：https://keras.io/ja/

活性化関数、損失誤差については後述

ここではsigmoidとmean_squared_errorを使用

KerasとTensorflowの関係

本講座では、Kerasの書式によってTensorflowを使用する

もともとKerasは独立したライブラリであったが、近年Tensorflow内に取り込まれた。ただし、その後も独立したライブラリとして存在している

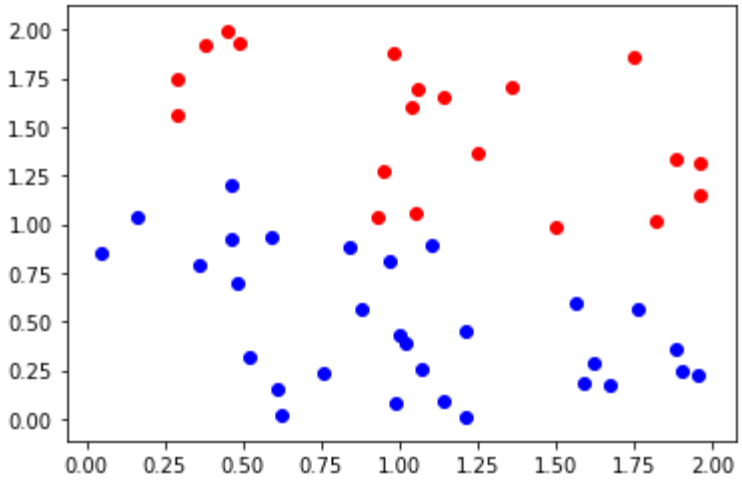
Kerasはバックエンドとして、Tensorflowの他にTheano, CNTKといったライブラリも使用できるが、ここではTensorflowのみを使用

純粋にTensorflowだけでニューラルネットワークを組む場合には、セッションという概念が必要になり、以下の記述が用いられるが、本講座ではこの書式は使用しない

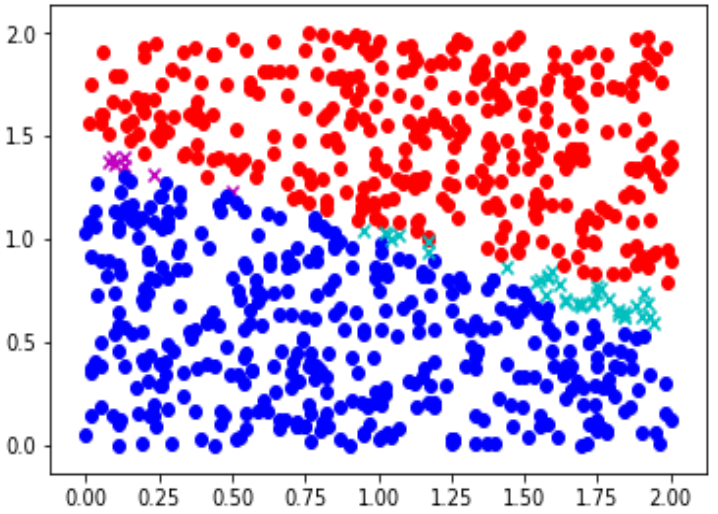
```
with tf.Session() as sess:
    print(sess.run())
```

線形分離可能なデータの結果

訓練データで学習し、評価用データで検証する



訓練データ



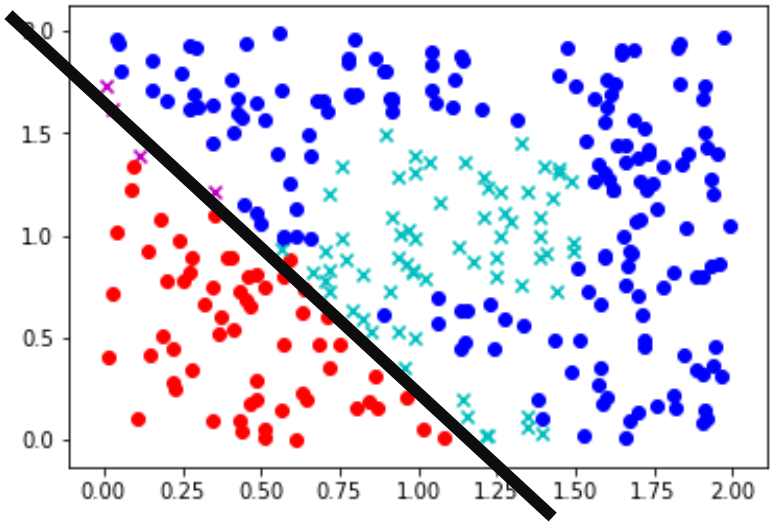
評価用データ

線形分類しかできない

単純パーセプトロンによる入力、重み、バイアス、出力の式は

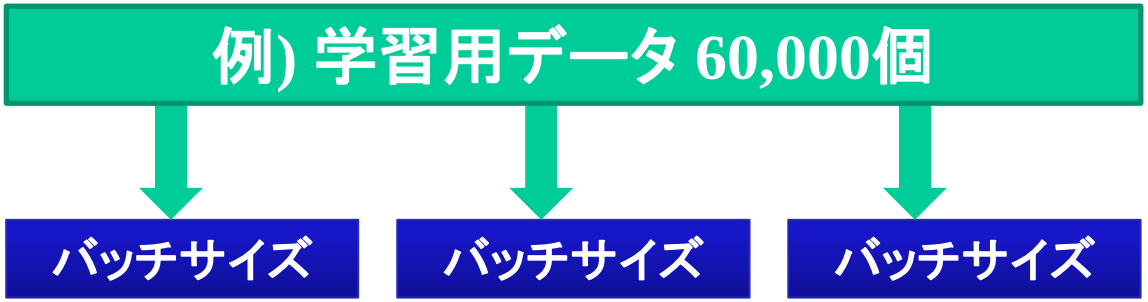
$$y = w_i x_i + b \rightarrow y = ax + b$$

すなわち、単なる1次式、線形回帰、2値予測になる
複雑なデータに対しても、次のように無理矢理1次式で分類してしまう
そこで、中間層と非線形関数の導入（活性化関数）でこれを解決する



他パラメータ

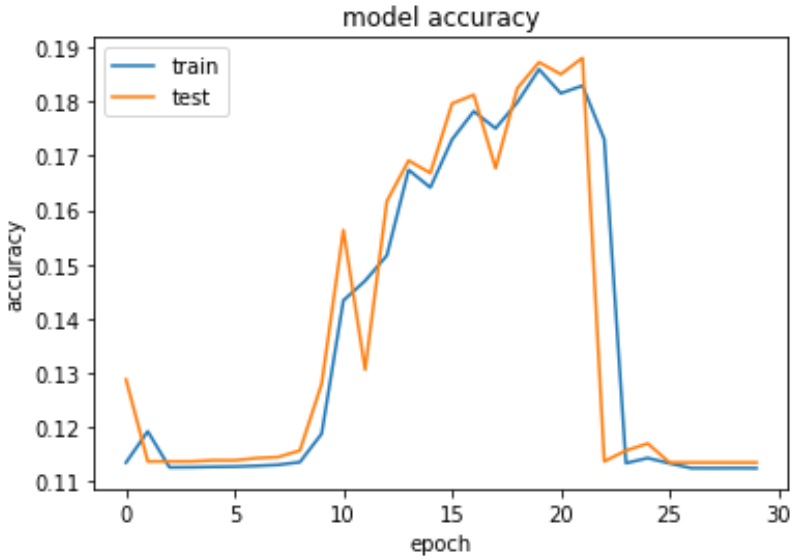
さらにバッチ数、エポック数というパラメータもある
学習用データをある程度のひとまとまりで学習するが、その数のこと
当然学習データ数より多くはできない。大抵2の乗数で設計
ある程度はバッチサイズがないと学習効果が出ない
全部で何回学習するかがエポック数。最初は少なめで傾向をつかむ



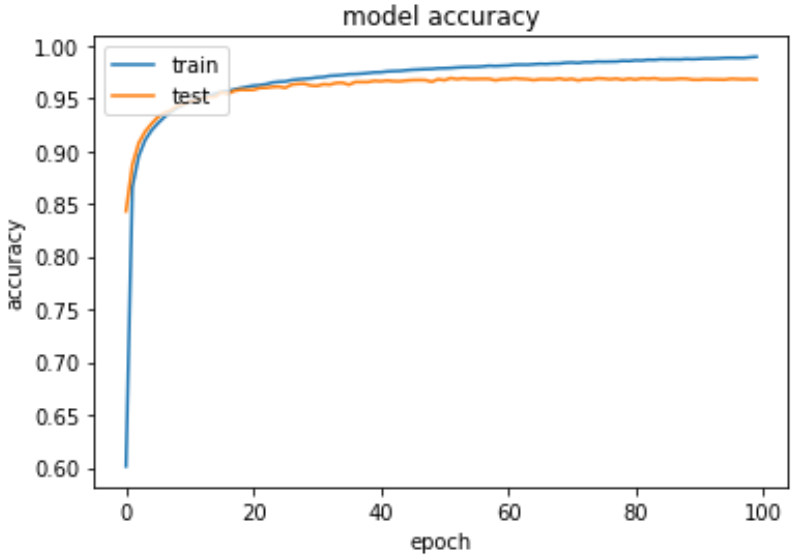
結果をどう見るか

loss（損失）が極力小さく、accuracy（正確さ）が極力大きいのが望ましい
結果のグラフで現れるのはaccuracyなので、右肩上がりが望ましい

悪い例



良い例



ロジスティック回帰

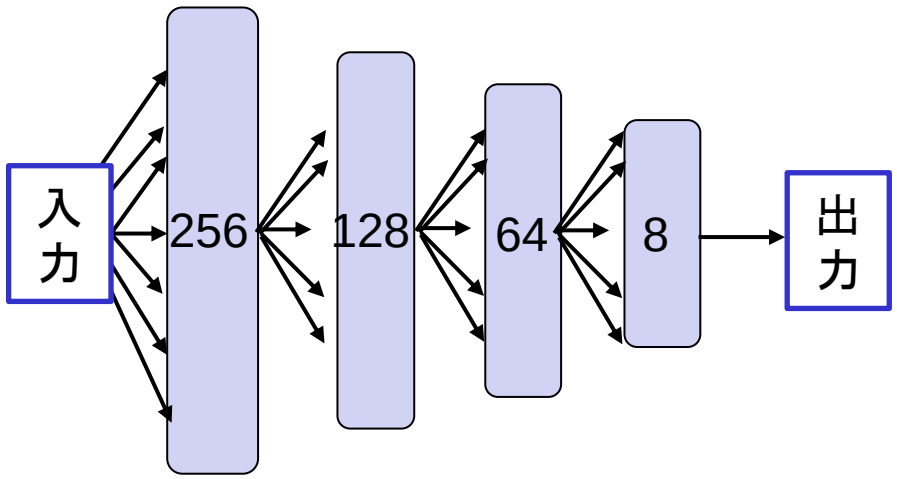
単純な線形分離のコードに対し、シグモイド関数を用いた活性化関数を追加すると、ロジスティック回帰

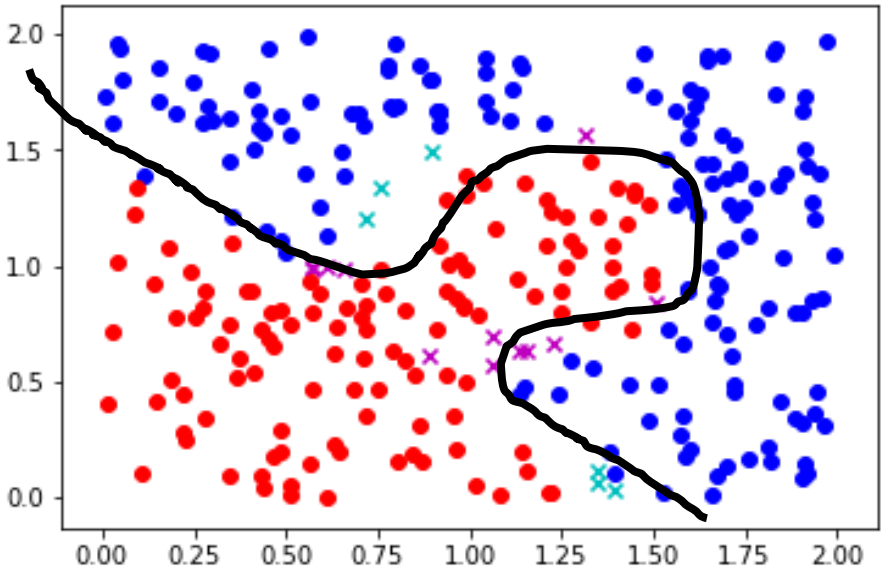
ただし、入力層と出力層のみのモデルでは線形分類しかできない

そのため、多層構造に変更する

DNN(Deep Neural Network)の演習

入出力含め6層のDNNを作成し、分類する
model.add で層を追加可能





ネットワークの設計

Kerasは krs.Sequential()に追記する形で組み立てていく

```
model = krs.Sequential()
model.add(krs.layers.Dense(256, input_shape=(train_data.shape
[1],)))
model.add(krs.layers.Activation("tanh"))
```

最初の入力

活性化関数

この書式でも可

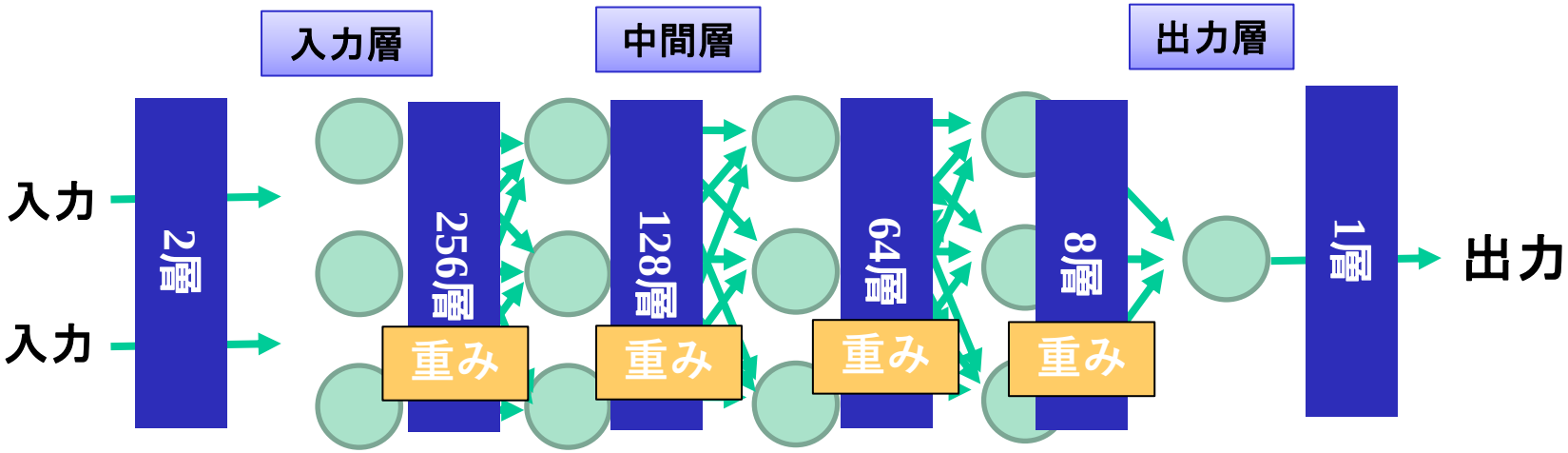
```
model.add(krs.layers.Dense(64, activation= "tanh"))
```

ニューラルネットワークの層の数

活性化関数

使用するネットワーク

すべての層が全結合された、最も基本的なDeep Neural Network
入力は画素数そのもの、出力はラベル数。途中の層は適当に決めている
もちろん増えれば増えるほど精度は向上するが、計算が遅くなる



多次元データ分類

Kaggleの天気データを用いて、データの整形と分類を実施する

元データ
https://www.kaggle.com/selfishgene/historical-hourly-weather-data

ライセンス：Open Data Commons Open Database License
CC BY-SA とほぼ同様。共有、創作、翻案は自由。ただし、配布にはライセンスの継承が必要

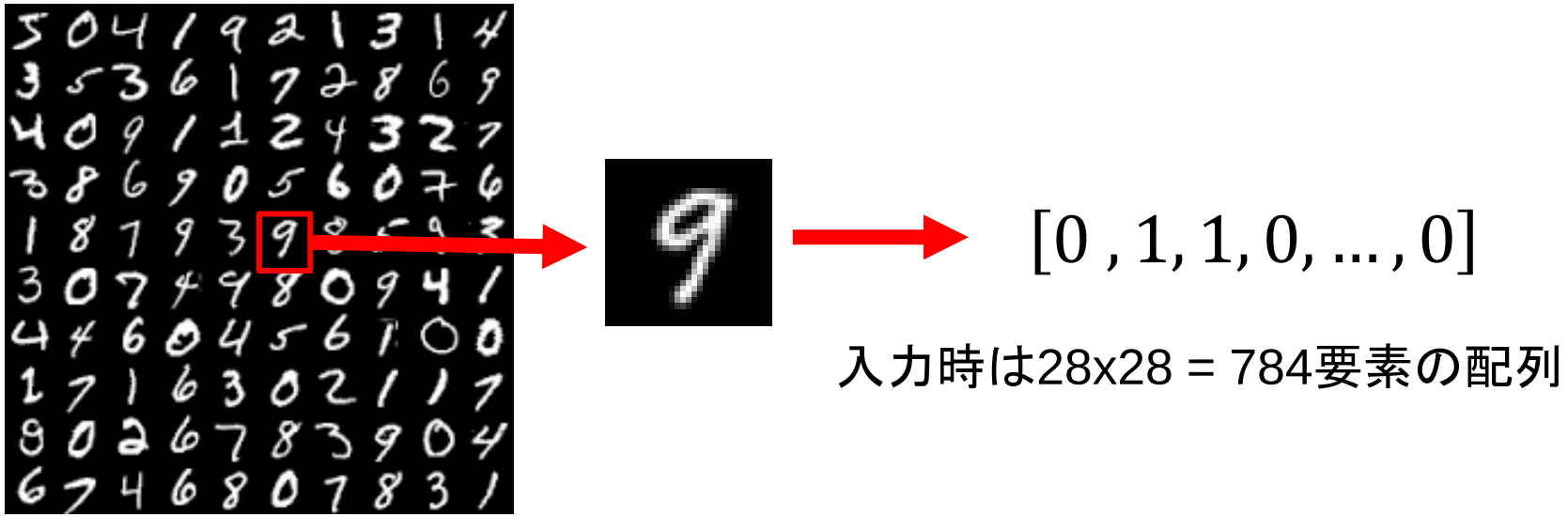
データ整形

NNに入力できるように天候データのうち、Denverの2017年データを整理
Humidity（湿度）, Pressure（気圧）,Temperature（気温）,
Wind_direction（風向）, Wind_speed（風速）, Weather（天気）を抽出
天気はweather_description.csvに書かれていて34種類ある

これを4種類に分けておく　晴れ：0、曇り：1、雨：2、雪：3
天気例）
broken clouds,曇り
drizzle,霧雨
dust,降塵
few clouds,少し曇り
moderate rain,中くらいの雨
overcast clouds,陰気な雲
proximity shower rain,すぐ近くでにわか雨

MNISTの分類

Mixed National Institute of Standards and Technology database
NY大学のYann LeCun教授らによる手書き文字DB
http://yann.lecun.com/exdb/mnist/
手書き文字を28x28で揃えたフォーマット
様々なエンジンでチュートリアルに用いられている
Kerasでは mnist.load_data() で自動的にダウンロードされ格納される



MNISTデータの取り寄せ

基本的にKerasのチュートリアルやGithubにサンプルソースがある
MNISTデータの取り寄せはこの1行でOK

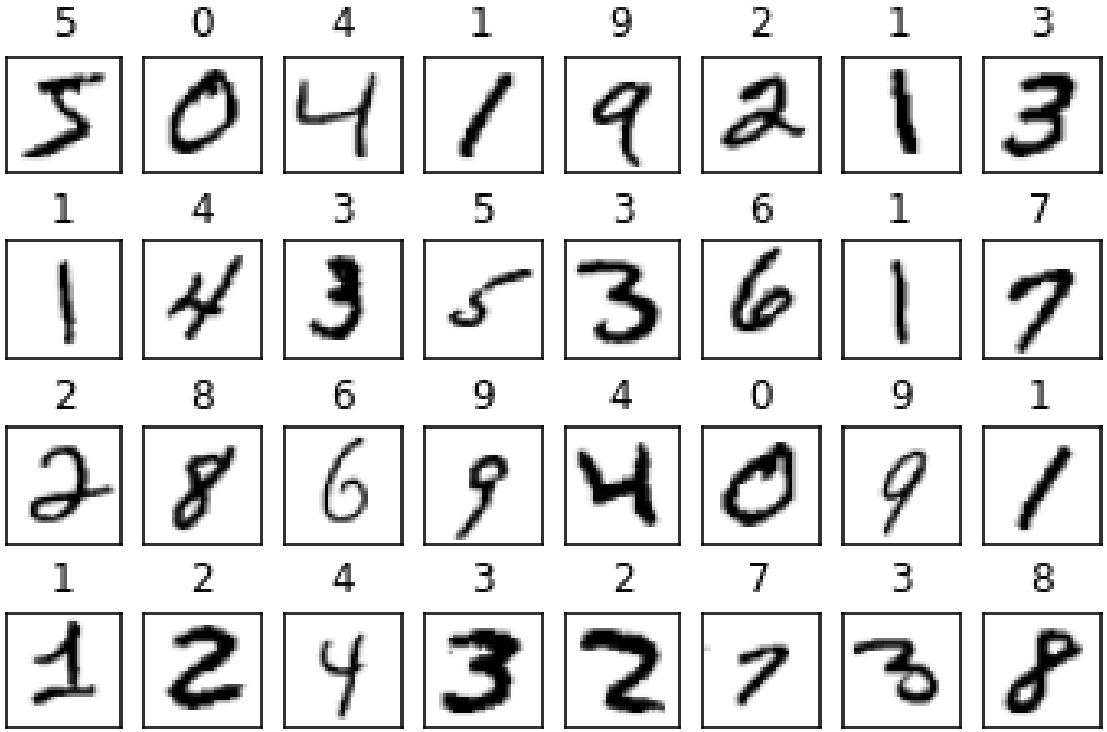
```
(x_train, y_train), (x_test, y_test) = mnist.load_data()
```

x_train ： 学習用（訓練・教師用）データ
y_train ： 学習用（訓練・教師用）ラベル
x_test ： 検証用データ
y_test ： 検証用ラベル
学習用データは60,000個、検証用データは10,000個

MNISTの中味

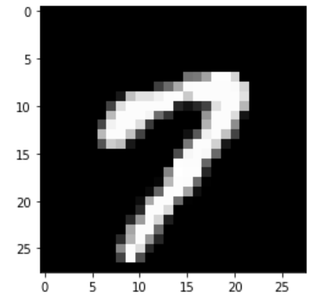
x_trainは多次元リスト

x_train[0]が数字の5, x_train[1]が数字の0 と、それぞれ28x28の画素データ



実際の中味は……

0-255の数値データ




```
[[  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   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   0   0   0   0]
 [  0   0   0   0   0   0   0   0   0   0   0   0   0   0   0 115 121 162 253 253 213   0   0   0   0   0   0   0]
 [  0   0   0   0   0   0   0   0   0   0   0   0  63 107 170 251 252 252 252 252 250 214   0   0   0   0   0   0]
 [  0   0   0   0   0   0   0   0  25 192 226 226 241 252 253 202 252 252 252 252 252 225   0   0   0   0   0   0]
 [  0   0   0   0   0   0   0  68 223 252 252 252 252 252  39  19  39  65 224 252 252 183   0   0   0   0   0   0]
 [  0   0   0   0   0   0   0 186 252 252 252 245 108  53   0   0   0 150 252 252 220  20   0   0   0   0   0   0]
 [  0   0   0   0   0   0  70 242 252 252 222  59   0   0   0   0   0 178 252 252 141   0   0   0   0   0   0   0]
 [  0   0   0   0   0   0 185 252 252 194  67   0   0   0   0  17  90 240 252 194  67   0   0   0   0   0   0   0]
 [  0   0   0   0   0   0  83 205 190  24   0   0   0   0   0 121 252 252 209  24   0   0   0   0   0   0   0   0]
 [  0   0   0   0   0   0   0   0   0   0   0   0   0   0  77 247 252 248 106   0   0   0   0   0   0   0   0   0]
 [  0   0   0   0   0   0   0   0   0   0   0   0   0   0 253 252 252 102   0   0   0   0   0   0   0   0   0   0]
 [  0   0   0   0   0   0   0   0   0   0   0   0   0 134 255 253 253  39   0   0   0   0   0   0   0   0   0   0]
 [  0   0   0   0   0   0   0   0   0   0   0   0   6 183 253 252 107   2   0   0   0   0   0   0   0   0   0   0]
 [  0   0   0   0   0   0   0   0   0   0   0  10 102 252 253 163  16   0   0   0   0   0   0   0   0   0   0   0]
 [  0   0   0   0   0   0   0   0   0   0  13 168 252 252 110   2   0   0   0   0   0   0   0   0   0   0   0   0]
 [  0   0   0   0   0   0   0   0   0   0  41 252 252 217   0   0   0   0   0   0   0   0   0   0   0   0   0   0]
 [  0   0   0   0   0   0   0   0   0  40 155 252 214  31   0   0   0   0   0   0   0   0   0   0   0   0   0   0]
 [  0   0   0   0   0   0   0   0   0 165 252 252 106   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0]
 [  0   0   0   0   0   0   0   0  43 179 252 150  39   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0]
 [  0   0   0   0   0   0   0   0 137 252 221  39   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0]
 [  0   0   0   0   0   0   0   0  67 252  79   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]]
```

ニューラルネットワークに入力するには

画像を2次元構造のまま高度に扱う手法（畳み込み）があるが、これは次回取り上げる

今回のような単純なDeep Neural Network(DNN)に入力するには、2次元構造ではなく、1次元の必要がある

そのため、[28, 28]のデータを[784]の巨大なリストにしてしまう

```
x_train_in = x_train.reshape(mnist_num, 28*28)
```

配列の形を変えるのがNumpyのreshape

正解ラベルの変換

正解ラベルもOne-hot形式に変換する必要がある

One-hot：正解が2 → **[0, 0, 1, 0, 0, 0, 0, 0, 0, 0]**

これはKerasの関数で1発変換

```
y_train_in = krs.utils.to_categorical(y_train, num_classes)
```

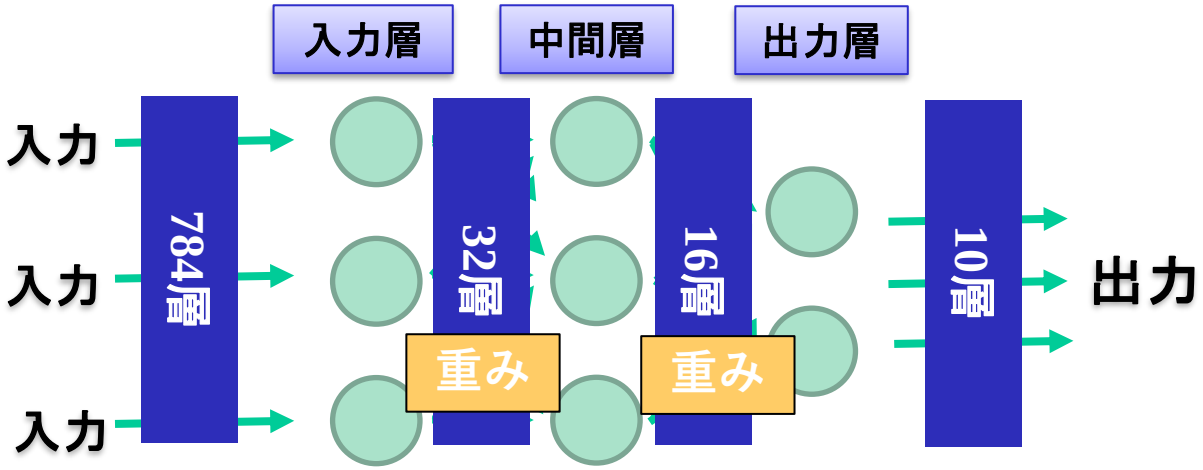
なぜこの形式が必要か

ニューラルネットワークでの分類問題は、あらかじめ設定されたラベルに対する**確率**が出力される

例）**[0, 0, 0.82, 0.04, 0, 0.1, 0.02, 0, 0.01, 0.01]** → 一番確率が高いのが2

使用するネットワーク

すべての層が全結合された、最も基本的なDeep Neural Network
入力は画素数そのもの、出力はラベル数。途中の層は適当に決めている
もちろん増えれば増えるほど精度は向上するが、計算が遅くなる



結果を良くするには

バッチサイズを増やす
ニューロンをもっと増やす
層自体を増やす

これをコピペしていけば層が増える

```
model.add(krs.layers.Dense(16, activation='relu'))
```

重みデータの保存と読み込み

指定回数の学習を終えると、各層の間のパラメータ（重み）ができあがる
これを次のコードで保存

```
model.save("mnist_predict_model.h5", include_optimizer=True)
```

保存しておくことで、良い結果のパラメータを次のコードで読み込める

```
model = krs.models.load_model("mnist_predict_model.h5", compile
 = True)
```

.h5はHDF5フォーマットのファイル
時系列データ保存用の標準フォーマットで、機械学習エンジンではほぼデファクトスタンダートのファイル形式

学習結果の検証

検証は、model.predictで行う　※検証用データをすべて引数で渡す

```
test_result = model.predict(x_test_in)
```

print(test_result[z])の結果は以下になるので

```
[0.08148291 0.1121015  0.10275728 0.10229278 0.09919951
0.09352663 0.09991397 0.10545233 0.10157305 0.10170012]
```

Numpyのargmaxで最も大きい値のインデックスを調べると

```
result =np.argmax(test_result,1)
print(result[z])
1
```

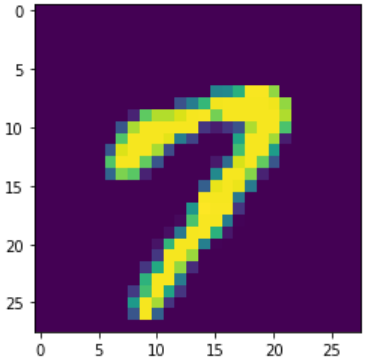
手書き文字で分類させてみる

この学習結果に、自分で書いた手書き文字を読ませてみる（用意済み）
手順は以下の通り

1. フォルダ内にpngファイルを格納し、Pythonで一括読み込み
2. OpenCVで画像ファイルを読み込む
3. Numpy配列に変換し、768個の1行配列にさらに直す

```
hand_image = images_tmp.reshape(filecount, 28*28)
predictions = model.predict(hand_image)
print(np.argmax(predictions,1))
```

根本的にデータを直す



実はもっと効率的なやり方：データの正規化
何もしなければカラー画像として計算されている
もともとモノクロ画像、さらに計算上0-1の範囲の方がよい
値が大きくなると損失関数が発散したり振動したりしやすくなる
元データはintなのでfloatに変換し、255で割る

```
x_train_nr = x_train   他データに移す
x_test_nr = x_test
x_train_nr = x_train_nr.astype('float32')   型変換
x_train_nr /= 255   255で割る
```

これでとても良い結果になる。**データの前処理はとても大事！**