深層学習用ライブラリ
Keras入門

田中章詞（iTHES）

本日の内容

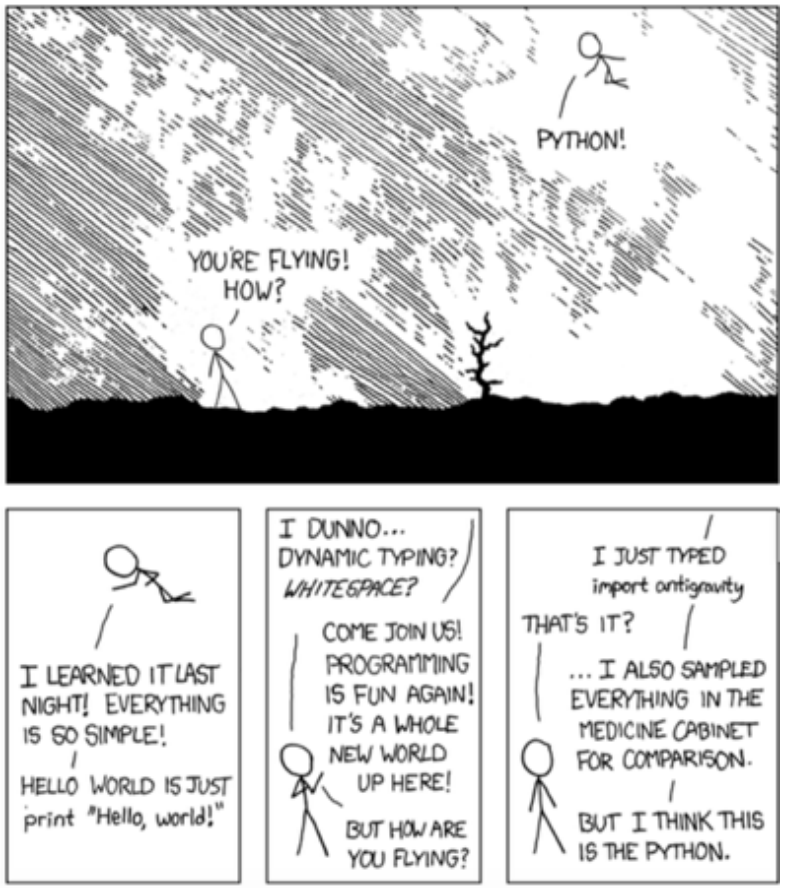
1．Kerasを使えるようになるまで

2．いろいろやってみる

3．深層学習の「Hello world」

1．Kerasを使えるようになるまで

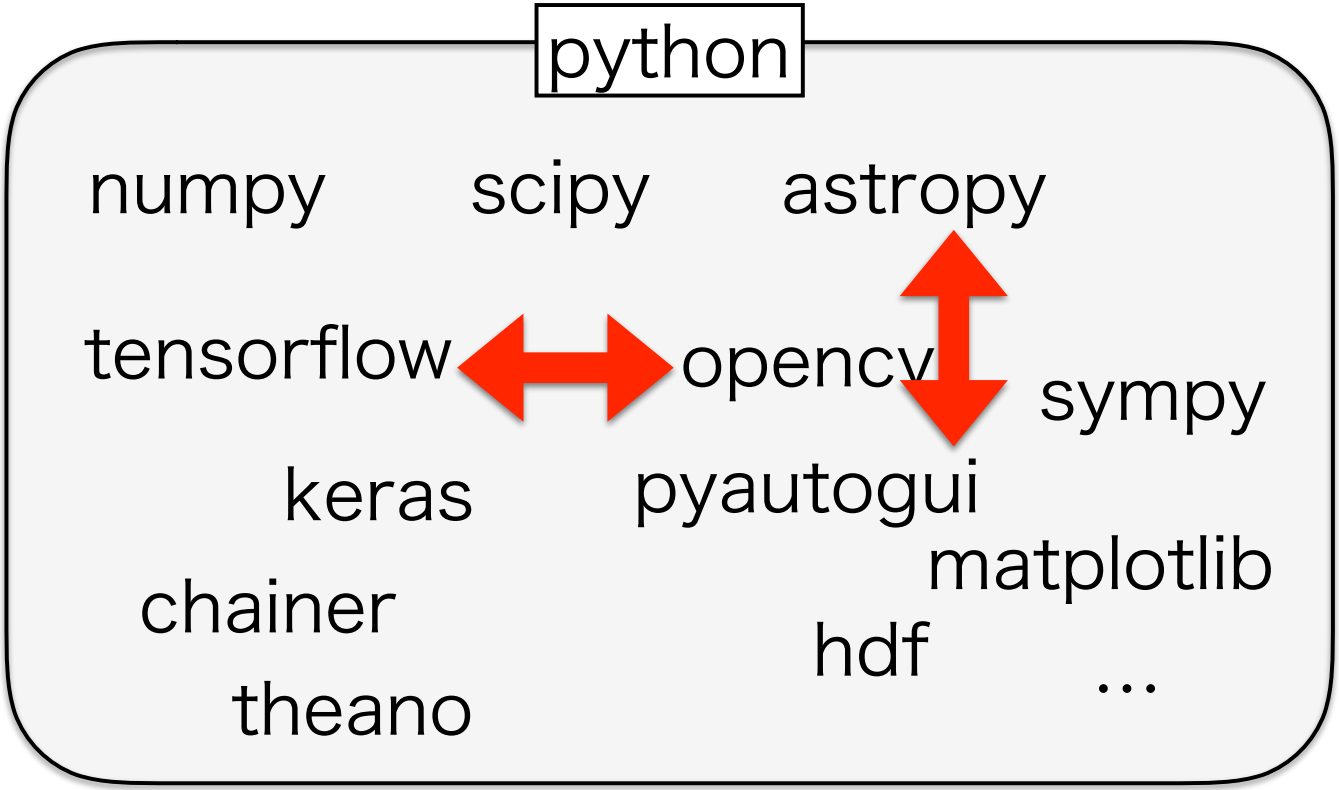
- python環境を整える
  - ・python + **ライブラリ**で使うと強力



1: 飛んでる！どうして！
2: pythonだよ！
昨日の夜、勉強したんだ。全てが簡単なんだよ。
print(Hello world)と打ち込めば
"Hello, world"と出力されるんだ。
1: 良く分からないな。それは動的なの？
whitespaceなの？
2: こっちに来いよ！
プログラミングの楽しさを思い出すよ。
ここは全く新しい世界だよ！
1: でも、どうやって飛んだの？
2: "**import antigravity**"とタイプしたんだよ。
1: それだけ!?

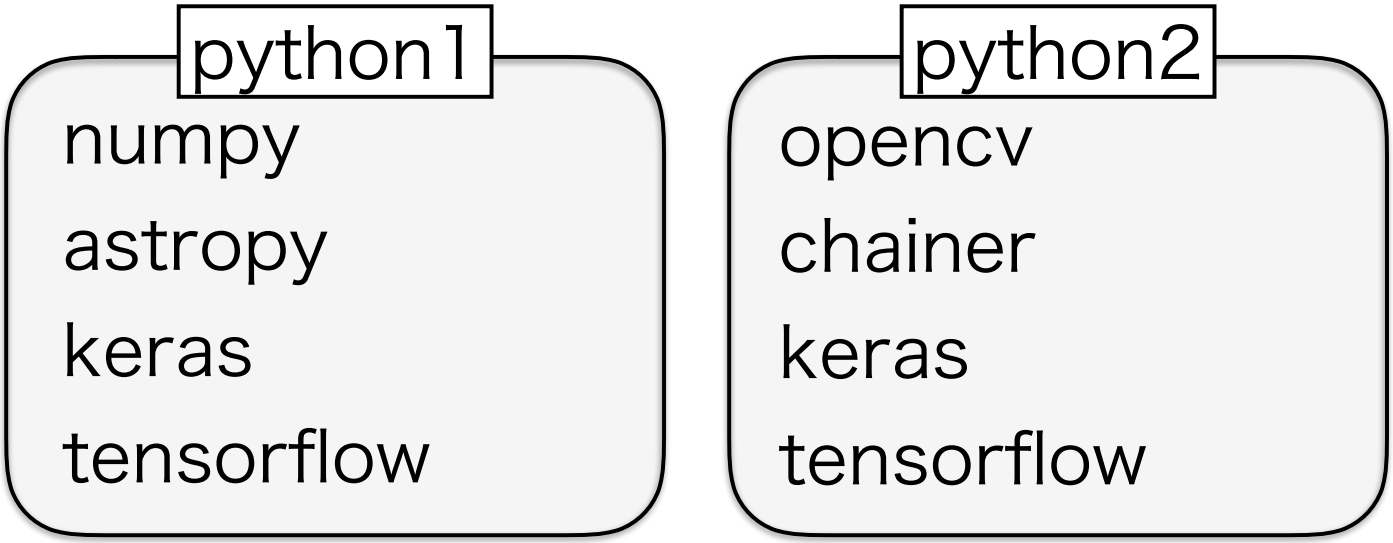
1．Kerasを使えるようになるまで

- python環境を整える
  - ・python + **ライブラリ**で使うと強力
    - numpy, scipy, astropy,…
    - tensorflow, keras, chainer, theano, …



1．Kerasを使えるようになるまで

- python環境を整える
  - ・python + **ライブラリ**で使うと強力
    - numpy, scipy, astropy,…
    - tensorflow, keras, chainer, theano, …
  - ・仮想環境をいくつか分けて作る方法



1．Kerasを使えるようになるまで

- python環境を整える
  - ・python + **ライブラリ**で使うと強力
    - numpy, scipy, astropy,…
    - tensorflow, keras, chainer, theano, …
  - ・仮想環境をいくつか分けて作る方法
    - Mac … pyenv, virtualenv等
    - Windows … Anaconda等

1．Kerasを使えるようになるまで

- python環境を整える
  - ・python + **ライブラリ**で使うと強力
    - numpy, scipy, astropy,…
    - tensorflow, keras, chainer, theano, …
  - ・仮想環境をいくつか分けて作る方法
    - Mac … pyenv, virtualenv等
    - Windows … Anaconda等
  - ・入っていると便利なもの
    jupyterコマンド … ノートの作成(HTML, Tex)
    pipコマンド … ライブラリインストールが容易に

1．Kerasを使えるようになるまで

- 私の環境

  Mac OS X バージョン10.11.5 (El Capitan)
  pyenvでanaconda3-4.1.0

- Kerasのインストール：

```
akinori-no-MacBook-Pro:~ akinoritanaka$
```

１．Kerasを使えるようになるまで

- 私の環境

  Mac OS X バージョン10.11.5 (El Capitan)
  pyenvでanaconda3-4.1.0

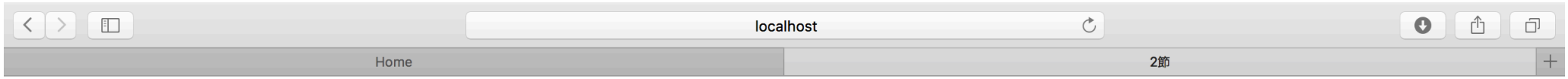
- Kerasのインストール：

  注意：
  $ pip install tensorflow ← まずこれ必要
  $ pip install keras

本日の内容

２．いろいろやってみる

３．深層学習の「Hello world」



2.いろいろやってみる

?

2-1. Kerasのインポート

In [1]:
```
import keras
from keras.models import Sequential
from keras.layers import Dense, Activation
```

```
Using TensorFlow backend.
```

それぞれの意味：

Sequential：*モデルの型*
Dense：*線形変換*
Activation：*活性化関数*

2-2. ニューラルネットをつくってみる

他にも作り方はありますが、
今回は**Sequential**クラスを使ってみます。
まずはそのオブジェクトをつくります：

In [2]:
```
Test_NN = Sequential() # 空のニューラルネット
```

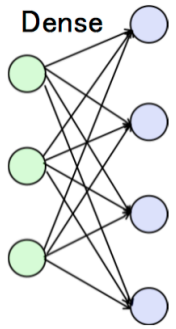
In [3]:
```
Test_NN
```

Out[3]:
```
<keras.models.Sequential at 0x103fc7470>
```

.add(...)でニューラルネットに構造...を与えることができます：

```
In [4]: Test_NN.add(Dense(4, input_dim=3))
```

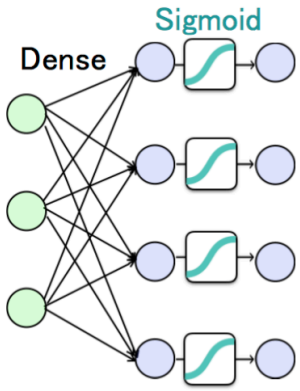
現在のニューラルネットの形状は：



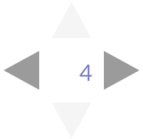
これでは単なる線形変換なのでもう少し足してみます：

まず活性化関数として**sigmoid**を入れてみます：

In [5]:
```
Test_NN.add(Activation('sigmoid'))
```



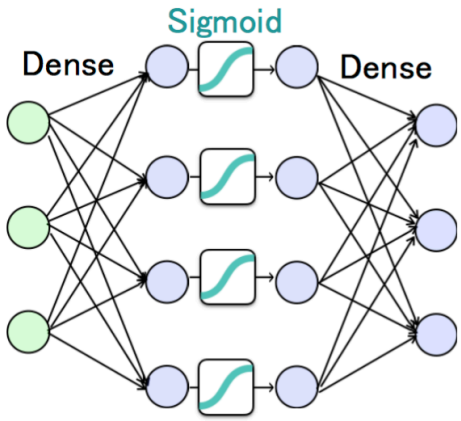
$$\text{sigmoid} : x \rightarrow \frac{1}{1+e^{-x}}$$



次に3次元への線形変換を再度**Dense**で入れてみます：

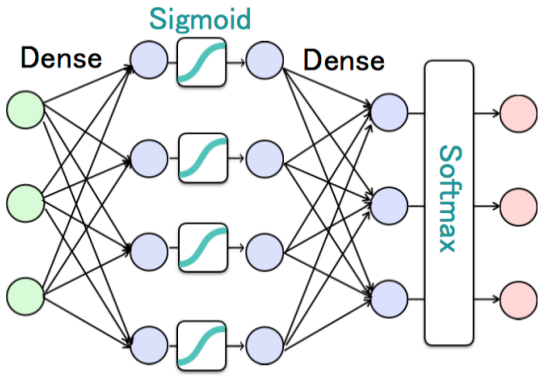
In [6]:
```
Test_NN.add(Dense(3))
```

２回目の**Dense**には入力サイズを明示しなくてOKです。

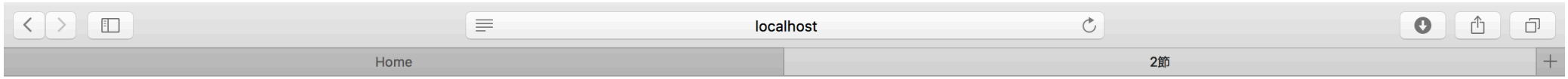


最後にsoftmax関数をはさみます：

```
In [7]: Test_NN.add(Activation('softmax'))
```



$$\mathrm{softmax} : \vec{x} = (x_i) \rightarrow \frac{e^{x_i}}{\sum_i e^{e^{x_i}}}$$



サマリーを出すと構造が整理して表示されるため便利です：

In [8]:
```
Test_NN.summary()
```

```
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
dense_1 (Dense)              (None, 4)                 16
_________________________________________________________________
activation_1 (Activation)    (None, 4)                 0
_________________________________________________________________
dense_2 (Dense)              (None, 3)                 15
_________________________________________________________________
activation_2 (Activation)    (None, 3)                 0
=================================================================
Total params: 31
Trainable params: 31
Non-trainable params: 0
_________________________________________________________________
```

2-3. ニューラルネットの学習準備

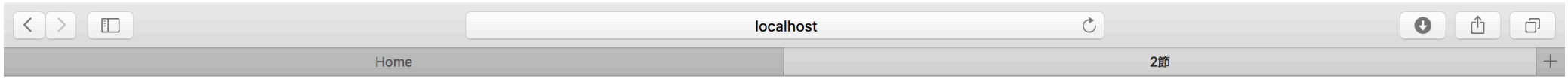
作ったTest_NNに.compileという関数があります：

In [9]:
```
Test_NN.compile(optimizer='adam',
                loss='categorical_crossentropy')
```

これで学習の際の

> ***optimizer***：*勾配法*
> ***loss***：*誤差関数*

が設定できました。あとは学習データを用意します。



とりあえず適当につくります（入力3次元、出力3次元）：

In [10]:

```
import numpy as np
# 5つの入力データ
x = np.array([
        [0,1,2],
        [1,3,1],
        [3,1,-1],
        [5,2,0],
        [0,8,0]
    ])
# それに対応した5つの正解データ
y = np.array([
        [1,0,0],
        [1,0,0],
        [0,1,0],
        [0,0,1],
        [0,1,0]
    ])
```

2-4. 学習

作ったTest_NNの.fitという関数に

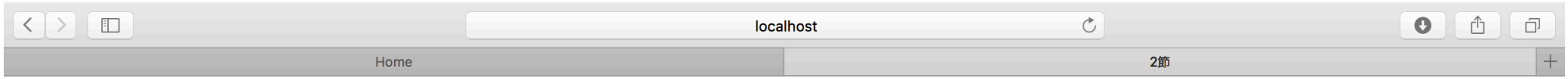
> *入力データ（x）と正解データ（y）*
> *何回勾配器を回すか（epochs）*

を渡せば学習開始します：

In [11]:
```
Test_NN.fit(x, y, epochs=3)
```

```
Epoch 1/3
5/5 [==============================] - 0s - loss: 1.2731
Epoch 2/3
5/5 [==============================] - 0s - loss: 1.2703
Epoch 3/3
5/5 [==============================] - 0s - loss: 1.2676
```

Out[11]:
```
<keras.callbacks.History at 0x11ddb2f28>
```



3回だと少ないのでたくさん回してみます：

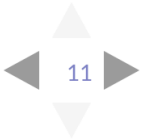
In [12]:
```
Test_NN.fit(x, y, epochs=500)  # 7秒
```

...

テストしてみます（.predict関数）：

In [13]:
```
Test_NN.predict(x), y
```

Out[13]:
```
(array([[ 0.62236255,  0.18385088,  0.19378662],
        [ 0.45650768,  0.34380636,  0.19968592],
        [ 0.16992027,  0.47773021,  0.35234958],
        [ 0.22151317,  0.41758704,  0.36089975],
        [ 0.24036226,  0.61675429,  0.14288346]], dtype=float32),
 array([[1, 0, 0],
        [1, 0, 0],
        [0, 1, 0],
        [0, 0, 1],
        [0, 1, 0]]))
```



2-5. ここまでのまとめ

In [ ]:

```
# ライブラリ等インポート
import keras ...

# モデルをつくる
Test_NN = Sequential()
Test_NN.add(...)

# モデルをコンパイルする
Test_NN.compile(optimizer=..., loss=...)

# 学習用データを用意する
(x, y) = (np.array(...), np.array(...))

# モデルを学習
Test_NN.fit(x, y, epochs=...)

# モデルをテスト
Test_NN.predict(x)
```

2-6. 素数を学ばせてみる

もう少し意味ありげな例で遊んでみましょう

In [14]:
```
import prime
x, y = prime.get_xy()
```

$$x[n] = [(n \bmod 2), \ldots, (n \bmod 11)],\ y[n] = \begin{cases} [1] & n\text{が素数} \\ [0] & n\text{が素数でない} \end{cases}$$

In [15]:
```
n = 23 # 149まで
x[n], y[n]
```

Out[15]:
```
(array([1, 2, 3, 2, 1]), array([1]))
```

In [20]:
```
x.shape, y.shape
```

Out[20]: ((150, 5), (150, 1))

150個の5次元の入力(x)、1次元の答(y)、ですので、

prime_NN：5次元→素数確率

というのをつくってみます：

In [16]:
```
prime_NN = Sequential()
prime_NN.add(Dense(20, input_dim=5))
prime_NN.add(Activation('sigmoid'))
prime_NN.add(Dense(1))
prime_NN.add(Activation('sigmoid'))
```

localhost

Home | 2節

In [17]:
```
prime_NN.compile(optimizer='adam',
                 loss='binary_crossentropy')
```

n = 0, 1 は除き、かつ100までのみ学習

In [26]:
```
prime_NN.fit(x[2:100], y[2:100], epochs=500, batch_size=50) # 15秒ほど
```

...

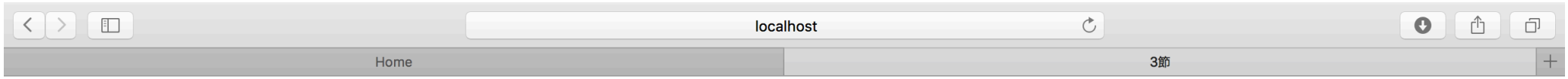
テスト：100〜150の間の素数を正しく見積もれるか

In [27]:
```
predictions = []
answers = []
for n in range(100, 150):
    if prime_NN.predict(x[n].reshape(1,5))>0.5:
        predictions.append(n)
    if y[n] == 1:
        answers.append(n)
print('予言', predictions)
print('答え', answers)
```

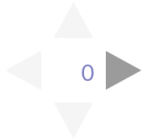
```
予言 [101, 103, 107, 109, 113, 119, 125, 131, 133, 137, 139, 143, 149]
答え [101, 103, 107, 109, 113, 127, 131, 137, 139, 149]
```

本日の内容

３．深層学習の「Hello world」



3. 深層学習の「Hello world」



3-1. MNISTデータ

手書き数字の学習データ・セット

In [1]:
```
import keras
import numpy as np
from keras.datasets import mnist
```

```
Using TensorFlow backend.
```

In [2]:
```
(x_tr, y_tr), (x_te, y_te) = mnist.load_data()
# 初回時は https://s3.amazonaws.com/img-datasets/mnist.npz
# からダウンロードされます
```

In [3]:
```
x_tr.shape, x_te.shape, y_tr.shape, y_te.shape
```

Out[3]:
```
((60000, 28, 28), (10000, 28, 28), (60000,), (10000,))
```

localhost

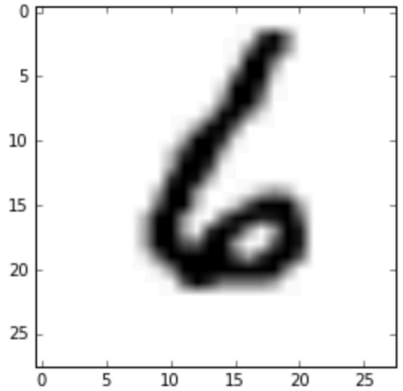
Home | 3節

どんなものか見てみます：

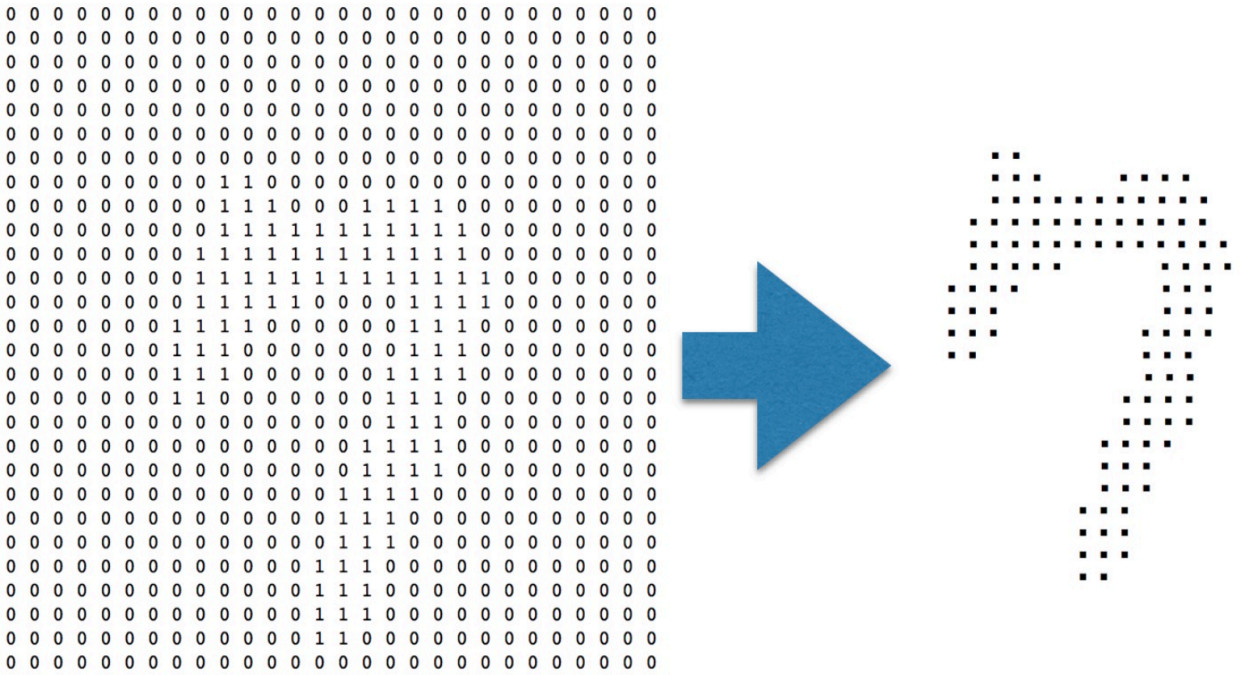
In [4]:
```
import matplotlib.pyplot as plt
%matplotlib inline
```

In [11]:
```
data_index = np.random.randint(60000)
print('label=', y_tr[data_index])
plt.imshow(x_tr[data_index], cmap =plt.cm.gray_r); plt.show()
```

```
label= 6
```



データ＝28 × 28の行列：



（ほんとうは0~255の値が入っています。）

3-2. 畳み込みニューラルネットを作る

In [12]:
```
from keras.models import Sequential
from keras.layers import Dense, Activation, Convolution2D, MaxPooling2D,
```

In [13]:
```
MNIST_NN = Sequential()
MNIST_NN.add(Convolution2D(10, (5, 5), input_shape=(28, 28, 1), padding=
MNIST_NN.add(MaxPooling2D(2, 2))
MNIST_NN.add(Convolution2D(10, (5, 5), padding='same'))
MNIST_NN.add(MaxPooling2D(2, 2))
MNIST_NN.add(Convolution2D(10, (5, 5), padding='same'))
MNIST_NN.add(MaxPooling2D(2, 2))
MNIST_NN.add(Flatten())
MNIST_NN.add(Dense(10, activation='softmax'))
```

Convolution2D(枚数, サイズ, input_shape)
input_shape=(行サイズ, 列サイズ, チャンネル数)
MaxPooling2D(サイズ)

In [14]:
```
MNIST_NN.summary()
```

```
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
conv2d_1 (Conv2D)            (None, 28, 28, 10)        260
_________________________________________________________________
max_pooling2d_1 (MaxPooling2 (None, 14, 14, 10)        0
_________________________________________________________________
conv2d_2 (Conv2D)            (None, 14, 14, 10)        2510
_________________________________________________________________
max_pooling2d_2 (MaxPooling2 (None, 7, 7, 10)          0
_________________________________________________________________
conv2d_3 (Conv2D)            (None, 7, 7, 10)          2510
_________________________________________________________________
max_pooling2d_3 (MaxPooling2 (None, 3, 3, 10)          0
_________________________________________________________________
flatten_1 (Flatten)          (None, 90)                0
_________________________________________________________________
dense_1 (Dense)              (None, 10)                910
=================================================================
Total params: 6,190
Trainable params: 6,190
Non-trainable params: 0
_________________________________________________________________
```

3-3. 学習の準備

データをニューラルネットに渡す用に整形します：

In [15]:
```
from keras.utils import np_utils

X_tr = x_tr.reshape(60000, 28, 28, 1)/255
X_te = x_te.reshape(10000, 28, 28, 1)/255
Y_tr = np_utils.to_categorical(y_tr, 10)
Y_te = np_utils.to_categorical(y_te, 10)

X_tr.shape, X_te.shape, Y_tr.shape, Y_te.shape
```

Out[15]:
```
((60000, 28, 28, 1), (10000, 28, 28, 1), (60000, 10), (10000, 10))
```

あとはMNIST_NNをコンパイルして.fitで学習させればよいです

3-4. 学習のテクニック

- 正解割合もモニターする方法：

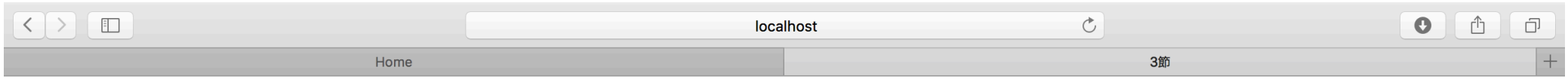
In [16]:
```
MNIST_NN.compile(optimizer='adam', loss='categorical_crossentropy',
                 metrics=['accuracy'])
```

- テスト誤差も監視し、**早期終了**

In [17]:
```
from keras.callbacks import EarlyStopping
MNIST_NN.fit(X_tr, Y_tr, epochs=1, batch_size=100,
             validation_data=(X_te, Y_te), callbacks=[EarlyStopping()])
```

```
Train on 60000 samples, validate on 10000 samples
Epoch 1/1
60000/60000 [==============================] - 49s - loss: 0.3694 - acc:
0.8877 - val_loss: 0.1164 - val_acc: 0.9631
```

Out[17]:
```
<keras.callbacks.History at 0x10964e0f0>
```



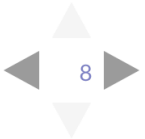
3-5. 学習後のモデルの保存、読込

保存は**.save**

In [18]:
```
MNIST_NN.save('MNIST_NN.h5')
```

読み込みは **load_model**というのを使います：

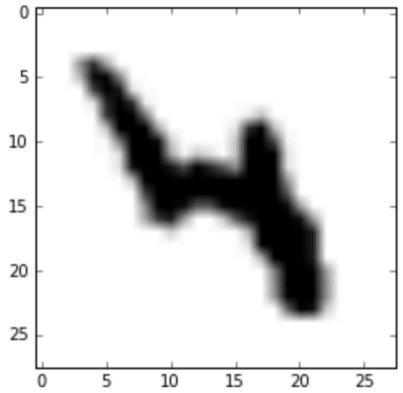
In [19]:
```
from keras.models import load_model
Loaded_NN = load_model('MNIST_NN.h5')
```



Loaded_NNにちゃんと学習済みモデルが入っているかチェック：

In [27]:
```
data_index = np.random.randint(10000)
predicted = Loaded_NN.predict(x_te[data_index].reshape(1,28,28,1))
print('予言=',np.argmax(predicted))
plt.imshow(x_te[data_index], cmap =plt.cm.gray_r)
plt.show()
```

予言= 4

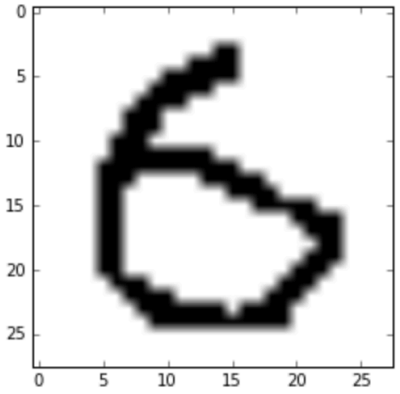


3-6. 「知らない」データを入れる

In [28]:
```
MNIST_NN_99 = load_model('MNIST_NN_99.h5')
```

In [30]:
```
from tegaki import Scribble, get28image
Scribble(); img = get28image('canvas.jpg')
print('予言：', np.argmax(MNIST_NN_99.predict(img.reshape(1,28,28,1))))
plt.imshow(img, cmap =plt.cm.gray_r) ;plt.show()
```

予言： 5



localhost

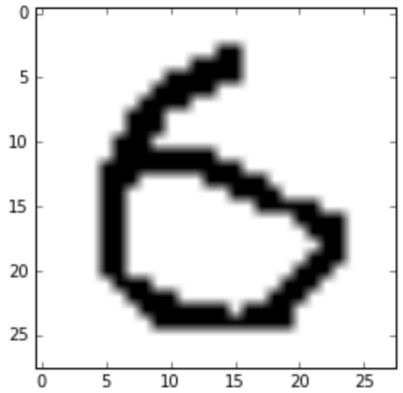
Home | 3節

#MNIST = 60,000, #**E**MNIST = 240,000

In [31]:
```
EMNIST_NN_99 = load_model('EMNIST_NN_99.h5')
```

In [32]:
```
from tegaki import get28image; img = get28image('canvas.jpg')
print('MNIST：', np.argmax(MNIST_NN_99.predict(img.reshape(1,28,28,1))), 'EMNIST：
plt.imshow(img, cmap =plt.cm.gray_r) ;plt.show()
```

```
MNIST： 5 EMNIST： 6
```

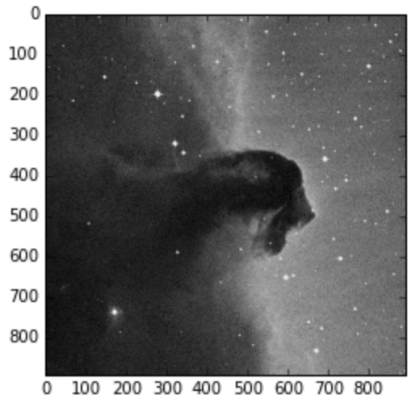


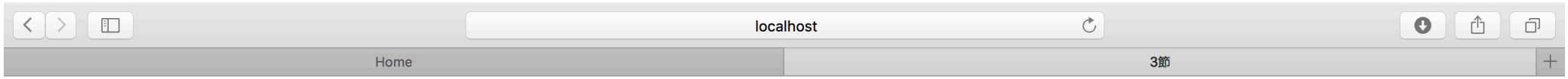
Astropy

In [33]:
```
import astropy.io.fits as pyfits
from astropy.utils.data import download_file
from astropy.io import fits
```

In [34]:
```
image_file = download_file('http://data.astropy.org/tutorials/FITS-image
```

In [35]:
```
image_data = fits.getdata(image_file)
plt.imshow(image_data, cmap='gray')
plt.show()
```





おわり