# “Hello world” of deep learning

# Keras

If you want to learn theano:

http://speech.ee.ntu.edu.tw/~tlkagk/courses/MLDS_2015_2/Lecture/Theano%20DNN.ecm.mp4/index.html

http://speech.ee.ntu.edu.tw/~tlkagk/courses/MLDS_2015_2/Lecture/RNN%20training%20(v6).ecm.mp4/index.html


or theano

Very flexible

Need some effort to learn

Interface of TensorFlow or Theano

keras

Easy to learn and use

(still have some flexibility)

You can modify it if you can write TensorFlow or Theano

# Keras

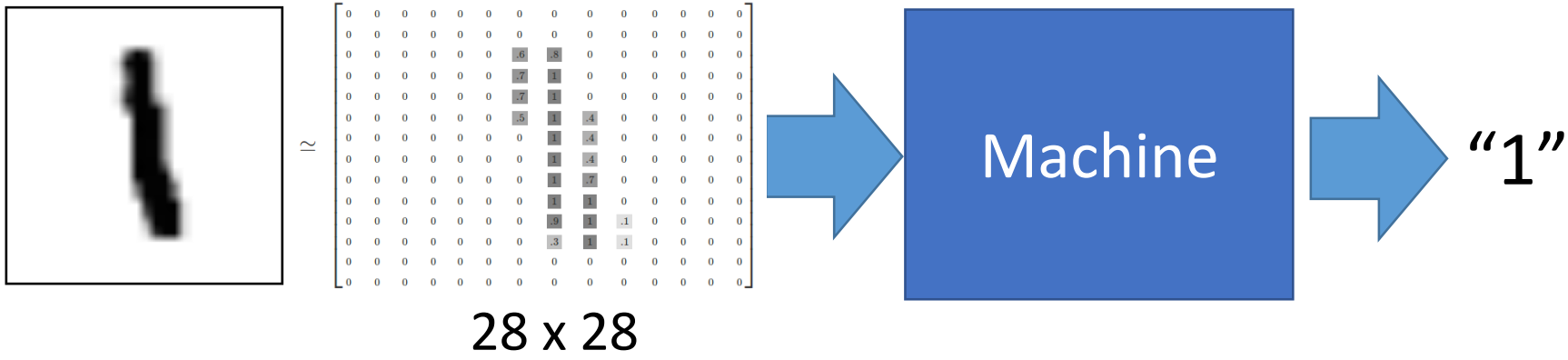
- François Chollet is the author of Keras.
  - He currently works for Google as a deep learning engineer and researcher.
- Keras means *horn* in Greek
- Documentation: http://keras.io/
- Example: https://github.com/fchollet/keras/tree/master/examples

感謝 沈昇勳 同學提供圖檔

# 使用 Keras 心得

# Example Application

- Handwriting Digit Recognition




MNIST Data: http://yann.lecun.com/exdb/mnist/

“Hello world” for deep learning

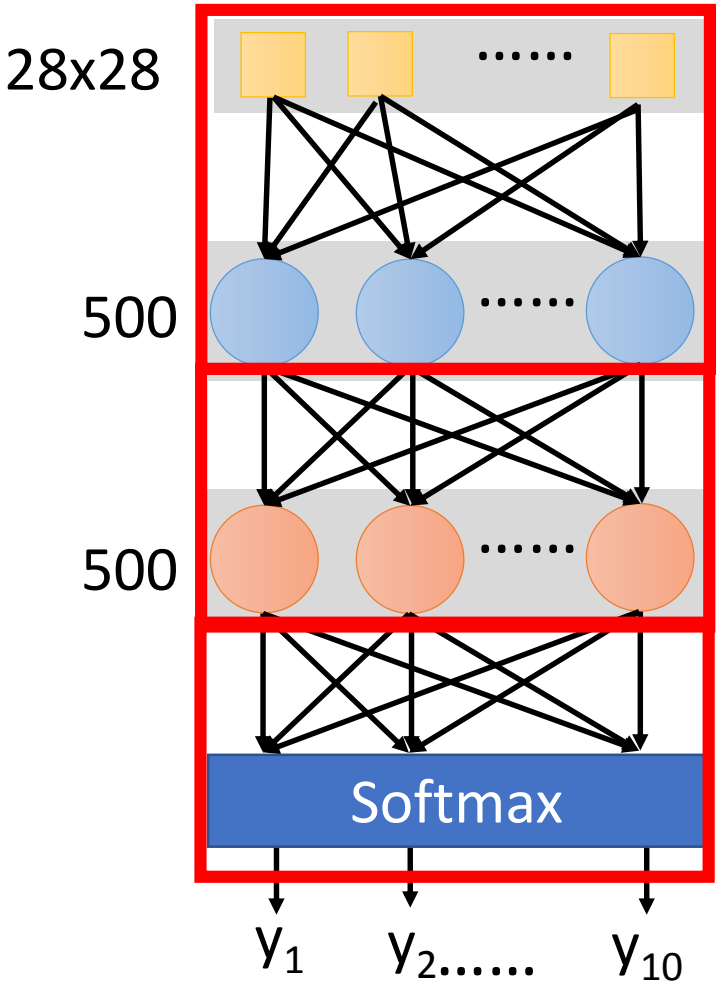
Keras provides data sets loading function: http://keras.io/datasets/

# Keras

Step 1: define a set of function

Step 2: goodness of function

Step 3: pick the best function




```
model = Sequential()
```

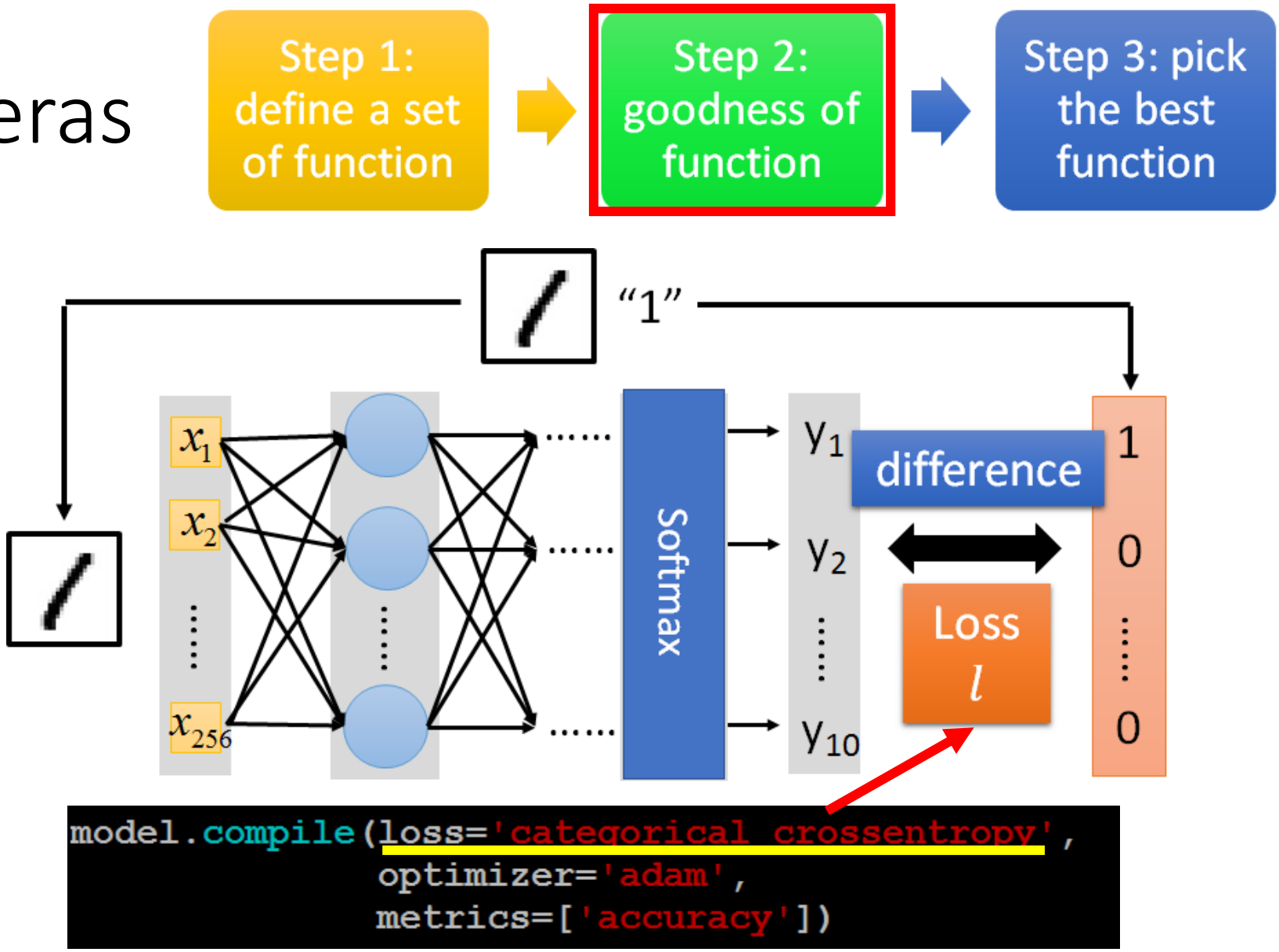
```
model.add( Dense( input_dim=28*28,
                  output_dim=500 ))
model.add( Activation('sigmoid') )
```

softplus, softsign, relu, tanh, hard_sigmoid, linear

```
model.add( Dense( output_dim=500 ) )
model.add( Activation('sigmoid') )
```

```
model.add( Dense(output_dim=10 ) )
model.add( Activation('softmax') )
```

# Keras




Several alternatives: https://keras.io/objectives/

# Keras


Step 3.1: Configuration

```
model.compile(loss='categorical_crossentropy',
              optimizer='adam',
              metrics=['accuracy'])
```

**SGD, RMSprop, Adagrad, Adadelta, Adam, Adamax, Nadam**

Step 3.2: Find the optimal network parameters

```
model.fit(x_train, y_train, batch_size=100, nb_epoch=20)
```

Training data (Images)

Labels (digits)

In the following slides

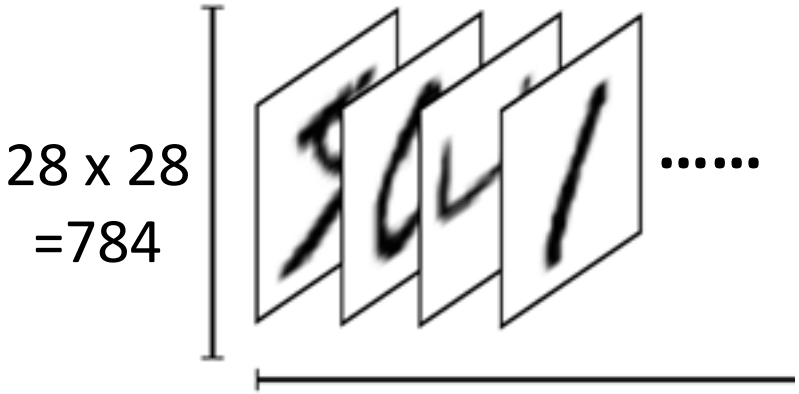
# Keras

Step 1: define a set of function → Step 2: goodness of function → Step 3: pick the best function

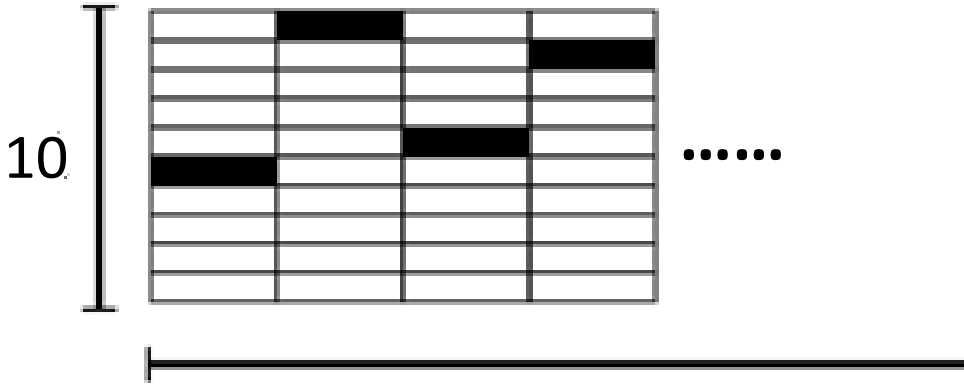
Step 3.2: Find the optimal network parameters

```
model.fit(x_train, y_train, batch_size=100, nb_epoch=20)
```

numpy array

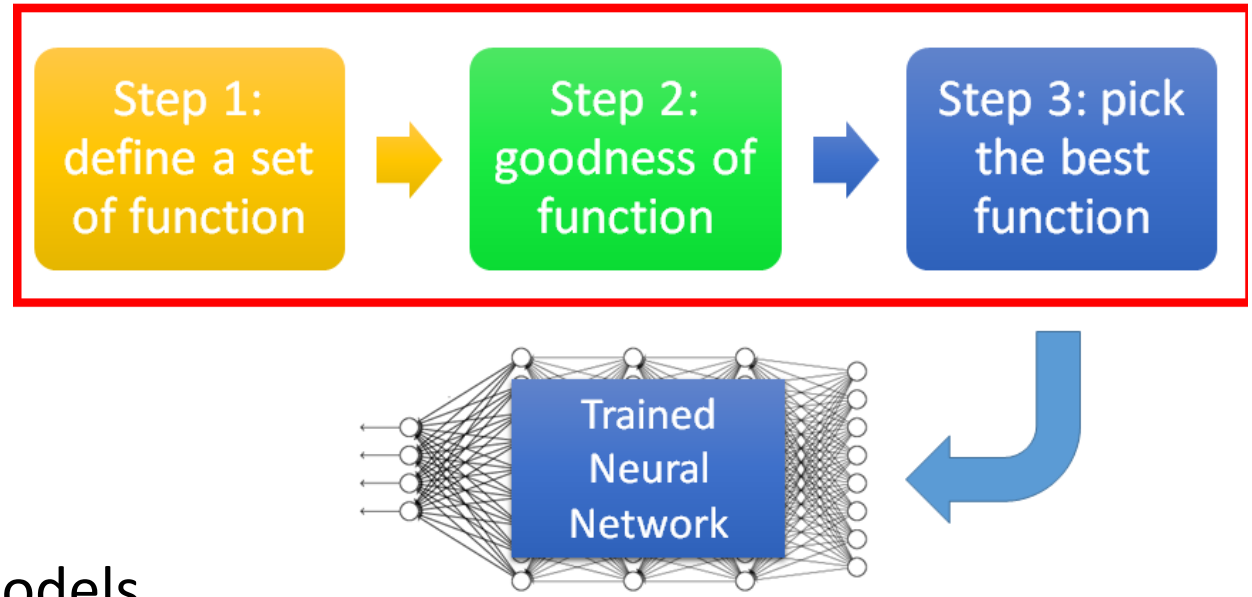



numpy array




https://www.tensorflow.org/versions/r0.8/tutorials/mnist/beginners/index.html

# Keras




Save and load models

http://keras.io/getting-started/faq/#how-can-i-save-a-keras-model

How to use the neural network (testing):

case 1:

```
score = model.evaluate(x_test,y_test)
print('Total loss on Testing Set:', score[0])
print('Accuracy of Testing Set:', score[1])
```

case 2:

```
result = model.predict(x_test)
```

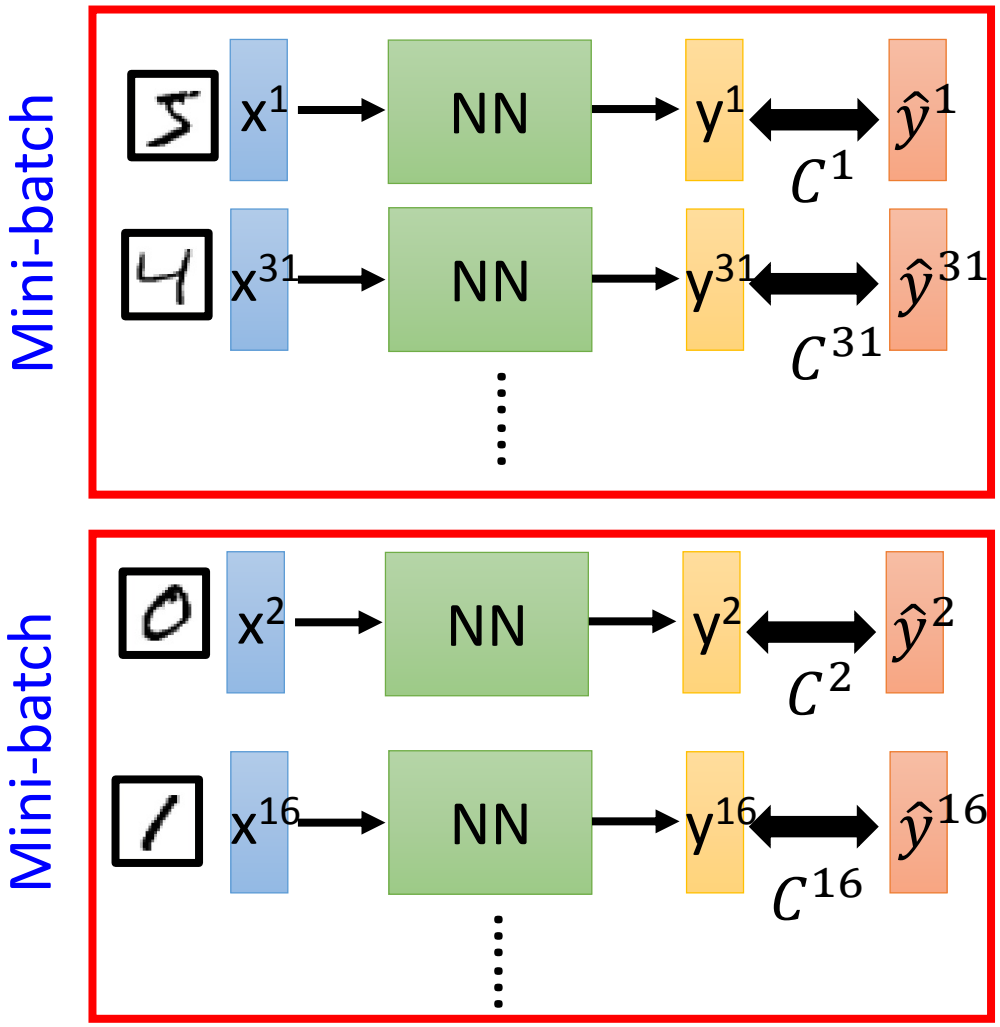
# Keras

- Using GPU to speed training
  - Way 1
    - THEANO_FLAGS=device=gpu0 python YourCode.py
  - Way 2 (in your code)
    - import os
    - os.environ["THEANO_FLAGS"] = "device=gpu0"

# Live Demo

We do not really minimize total loss!

# Mini-batch




- Randomly initialize network parameters
- Pick the 1st batch

  $L' = C^1 + C^{31} + \cdots$

  Update parameters once
- Pick the 2nd batch

  $L'' = C^2 + C^{16} + \cdots$

  Update parameters once

  ⋮
- Until all mini-batches have been picked

one epoch

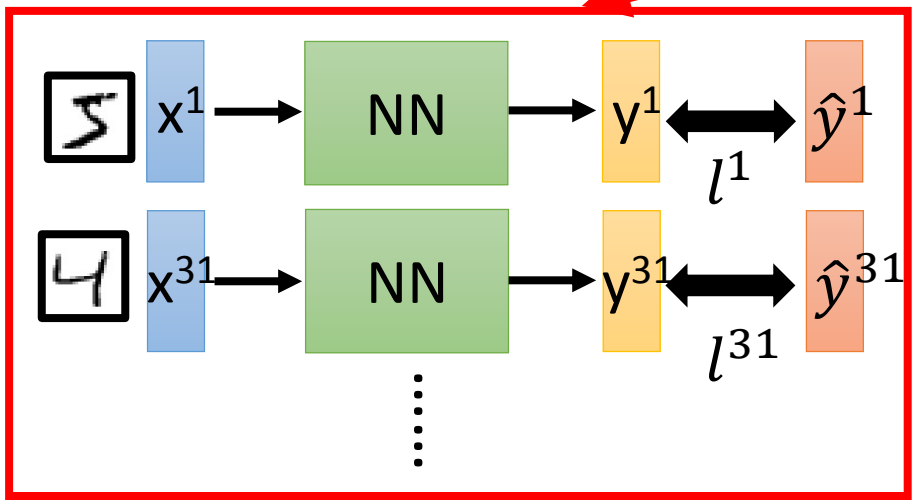
Repeat the above process

# Mini-batch

Batch size influences both ***speed*** and ***performance***. You have to tune it.

```
model.fit(x_train, y_train, batch_size=100, nb_epoch=20)
```

Mini-batch




100 examples in a mini-batch

Batch size = 1 ➡

Stochastic gradient descent

- Pick the 1st batch

  $L' = C^1 + C^{31} + \cdots$

  Update parameters once

- Pick the 2nd batch

  $L'' = C^2 + C^{16} + \cdots$

  Update parameters once

  ⋮

- Until all mini-batches have been picked

Repeat 20 times

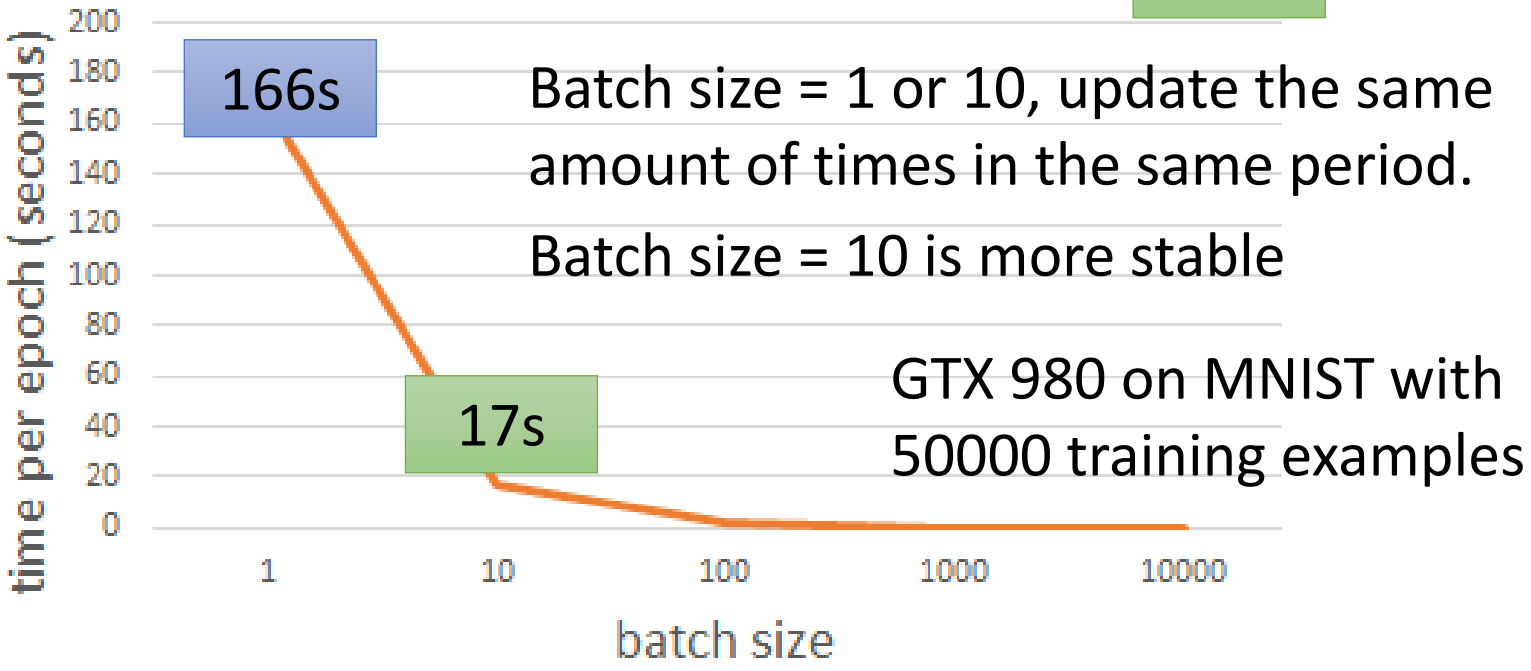
one epoch

# Speed

- Smaller batch size means more updates in one epoch
  - E.g. 50000 examples
  - batch size = 1, 50000 updates in one epoch
  - batch size = 10. 5000 updates in one epoch

166s 1 epoch

17s 10 epoch

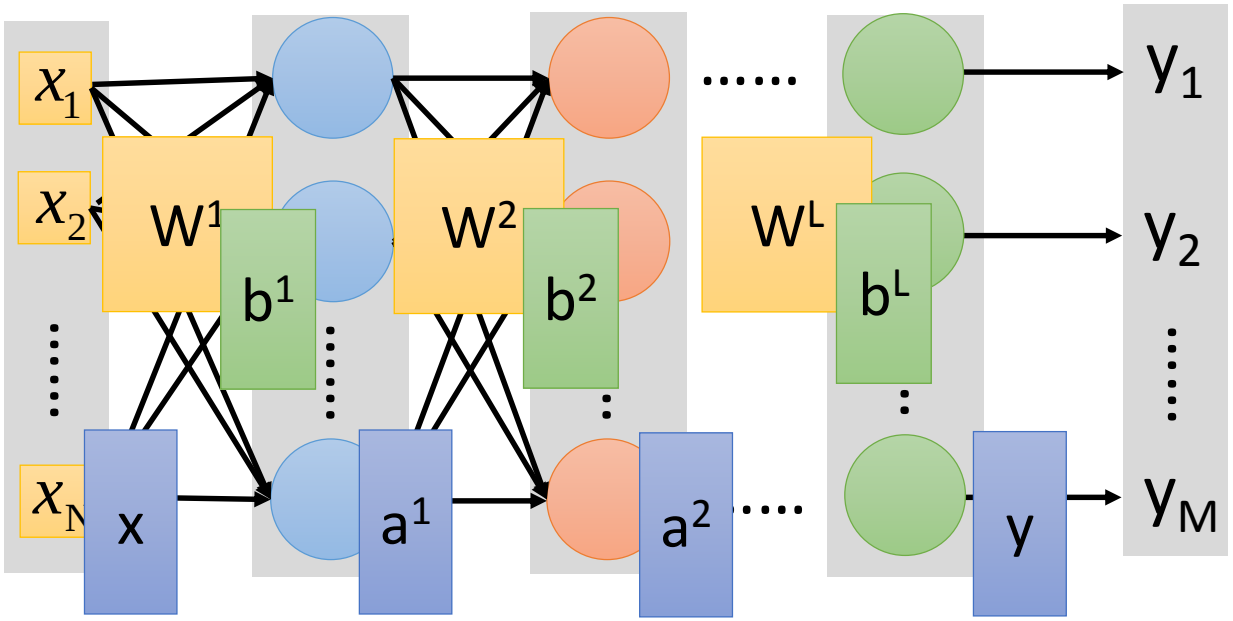



Batch size = 1 or 10, update the same amount of times in the same period.

Batch size = 10 is more stable

GTX 980 on MNIST with 50000 training examples

# Speed - Matrix Operation



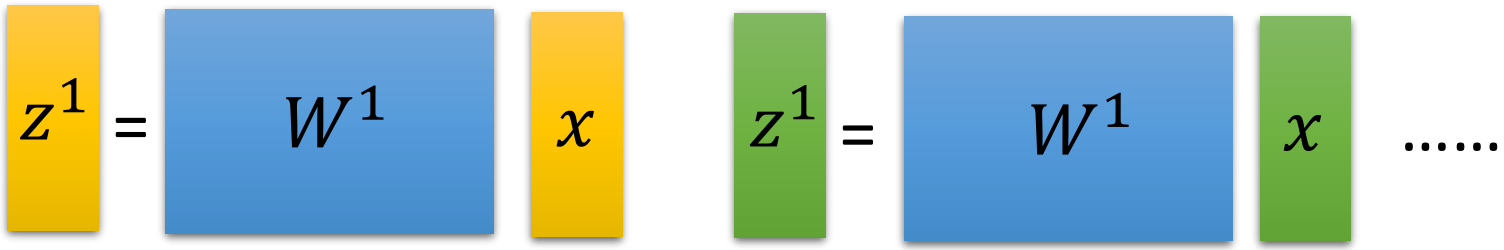

$$\mathbf{y} = f(\mathbf{x})$$ Forward pass (Backward pass is similar)

$$= \sigma(\mathbf{W}^L \cdots \sigma(\mathbf{W}^2 \sigma(\mathbf{W}^1 \mathbf{x} + \mathbf{b}^1) + \mathbf{b}^2) \cdots + \mathbf{b}^L)$$

# Speed - Matrix Operation

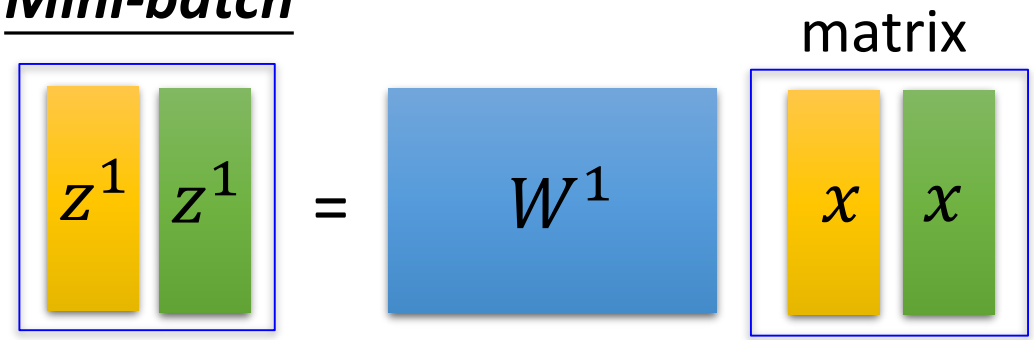
- Why mini-batch is faster than stochastic gradient descent?

***Stochastic Gradient Descent***




***Mini-batch***

matrix

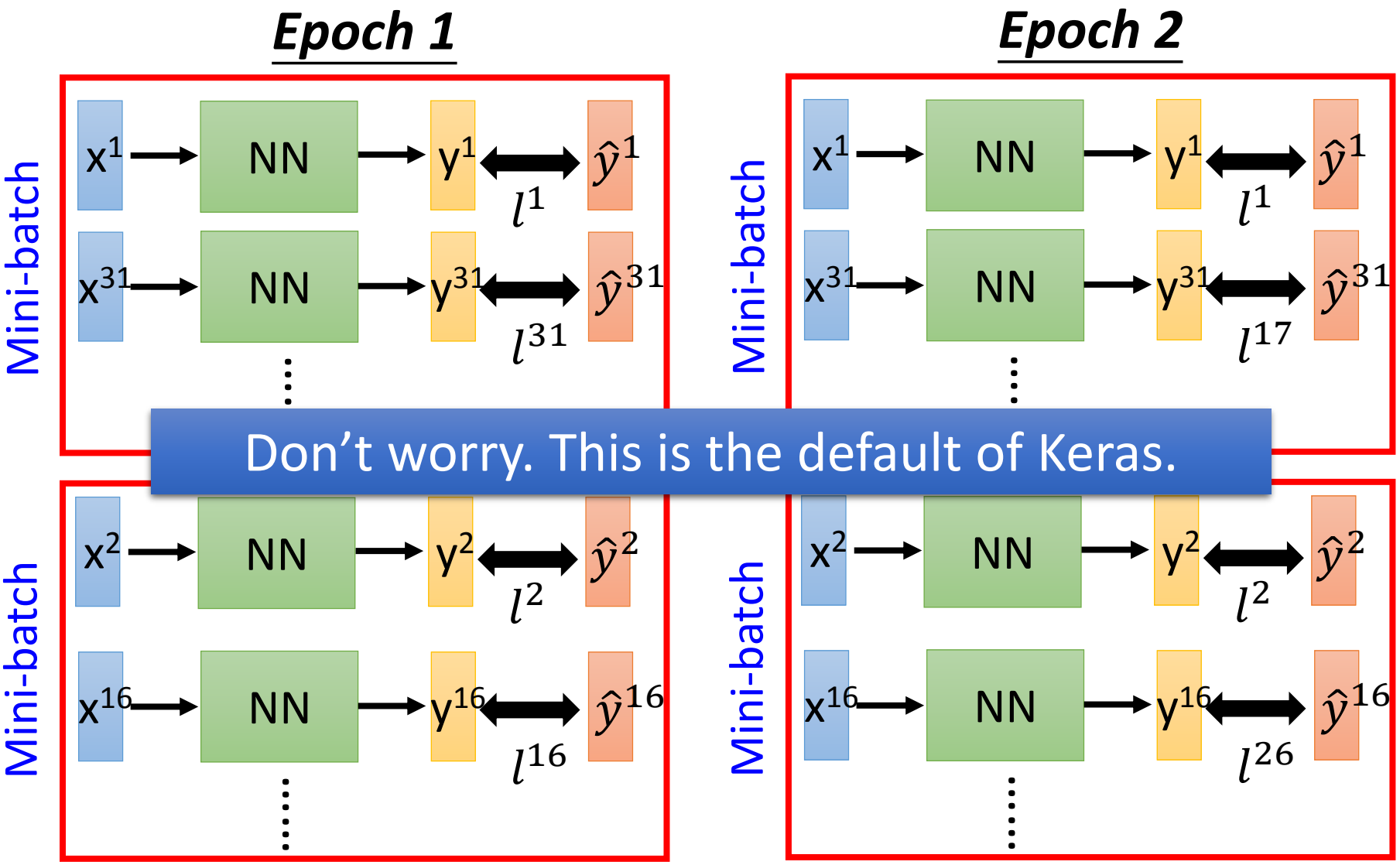



Practically, which one is faster?

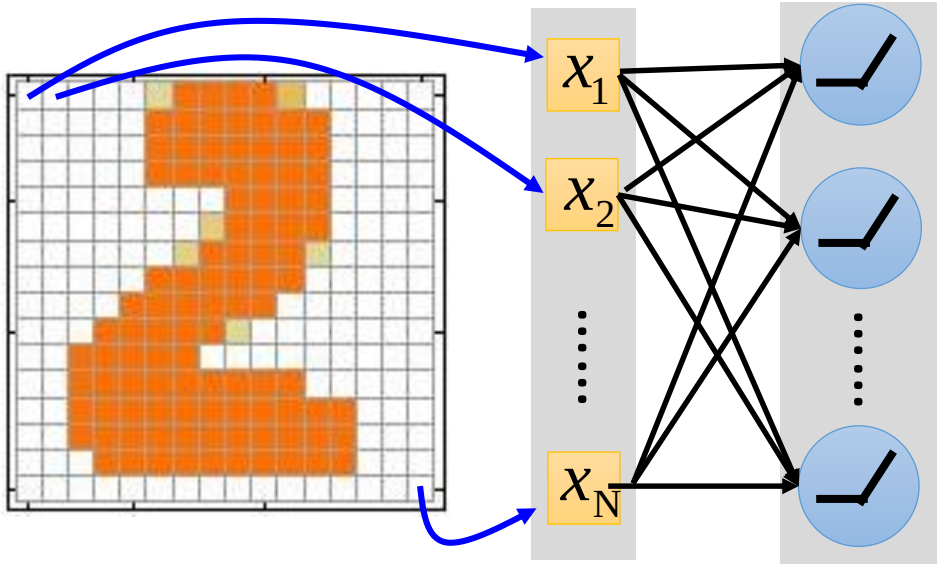
# Performance

- Larger batch size yields more efficient computation.
  - However, it can yield worse performance

# Shuffle the training examples for each epoch

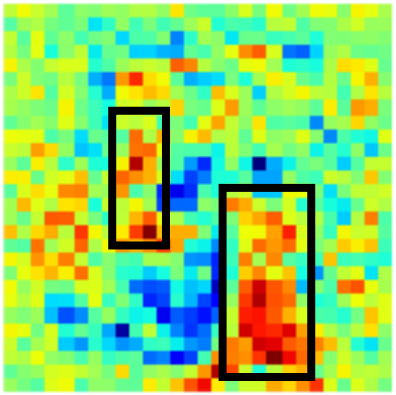


# Analysis




When did the neuron has the largest output?

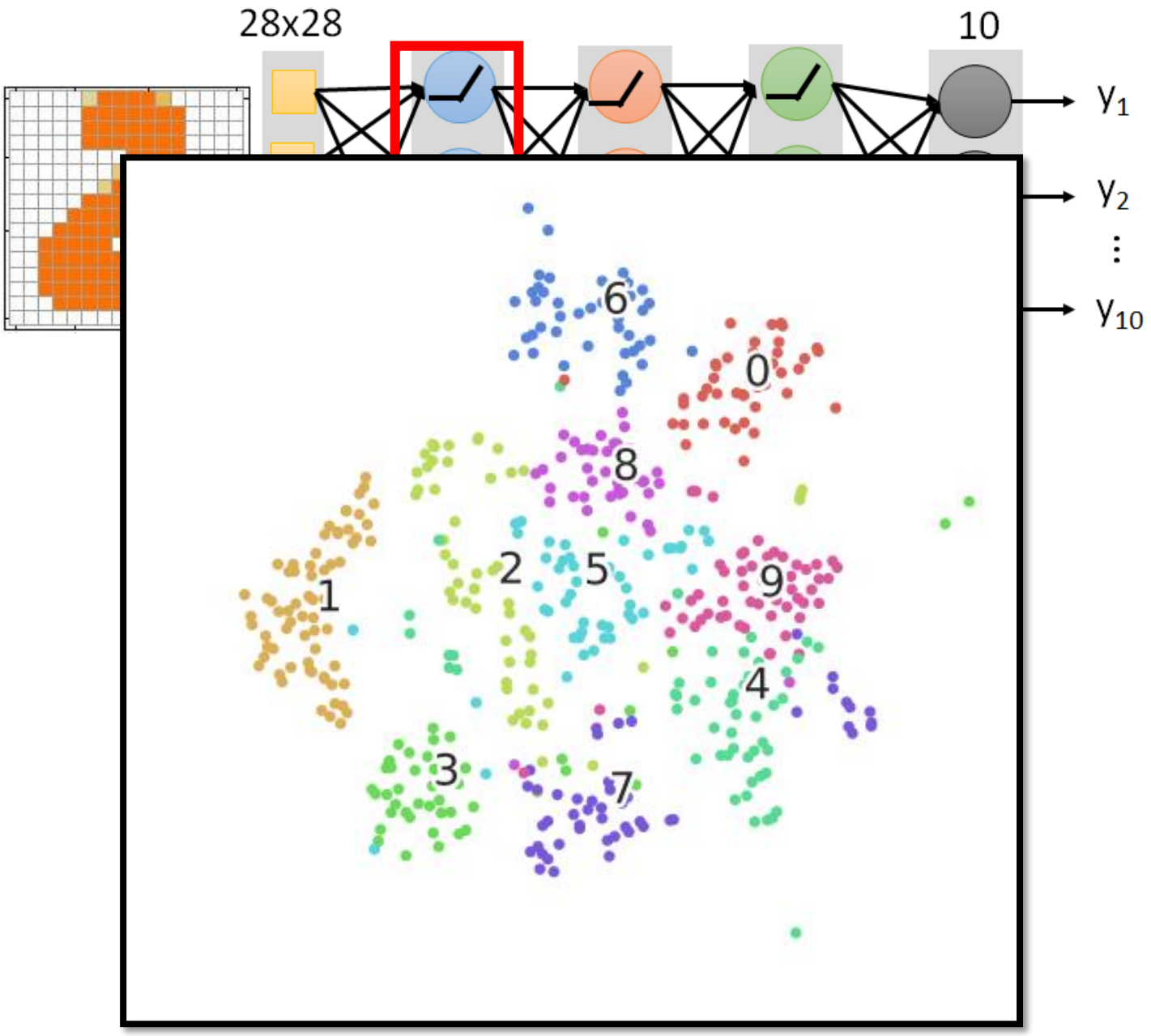
Arranging the weights according to the pixels they connected

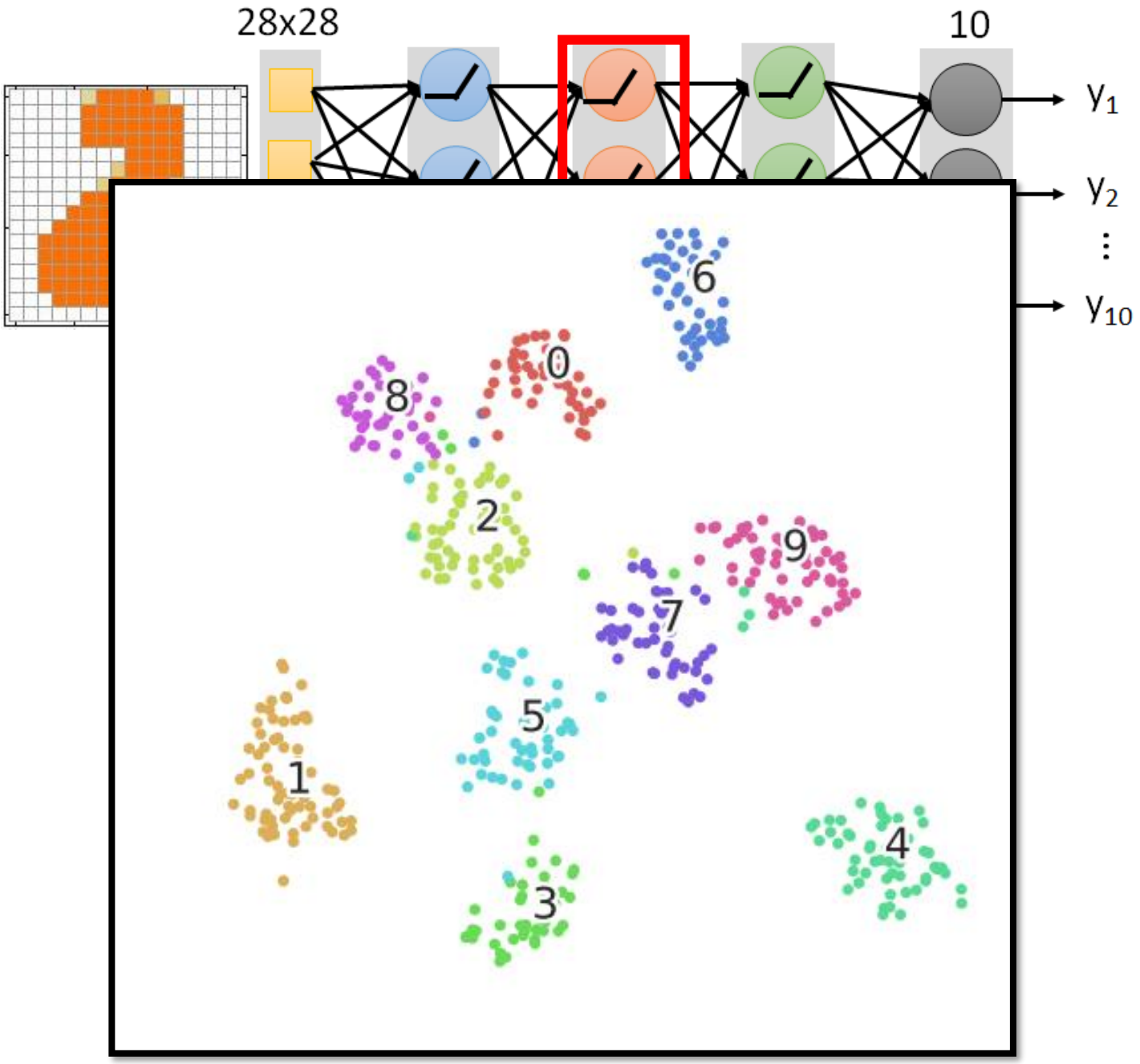


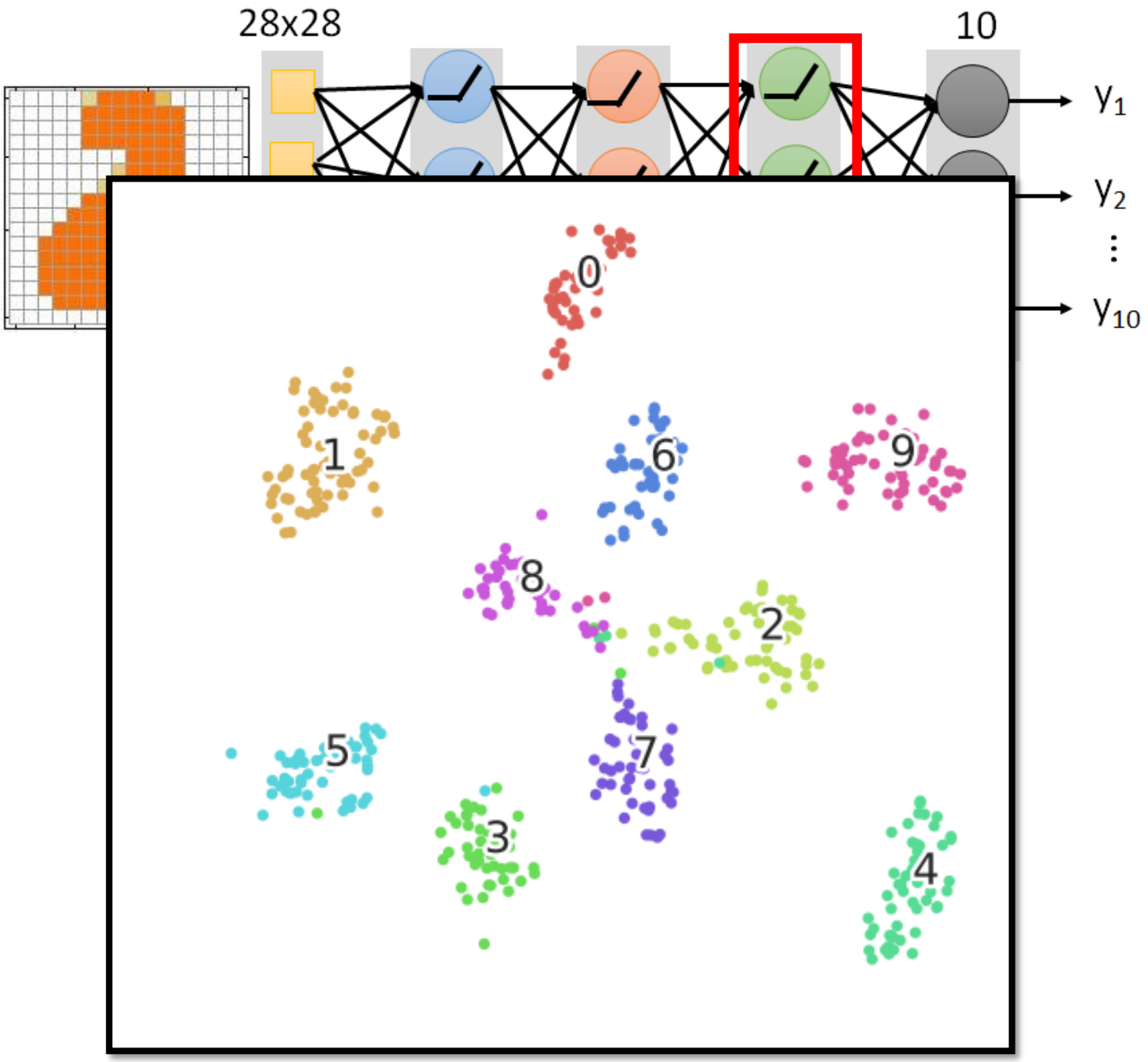
Red: positive

Blue: negative

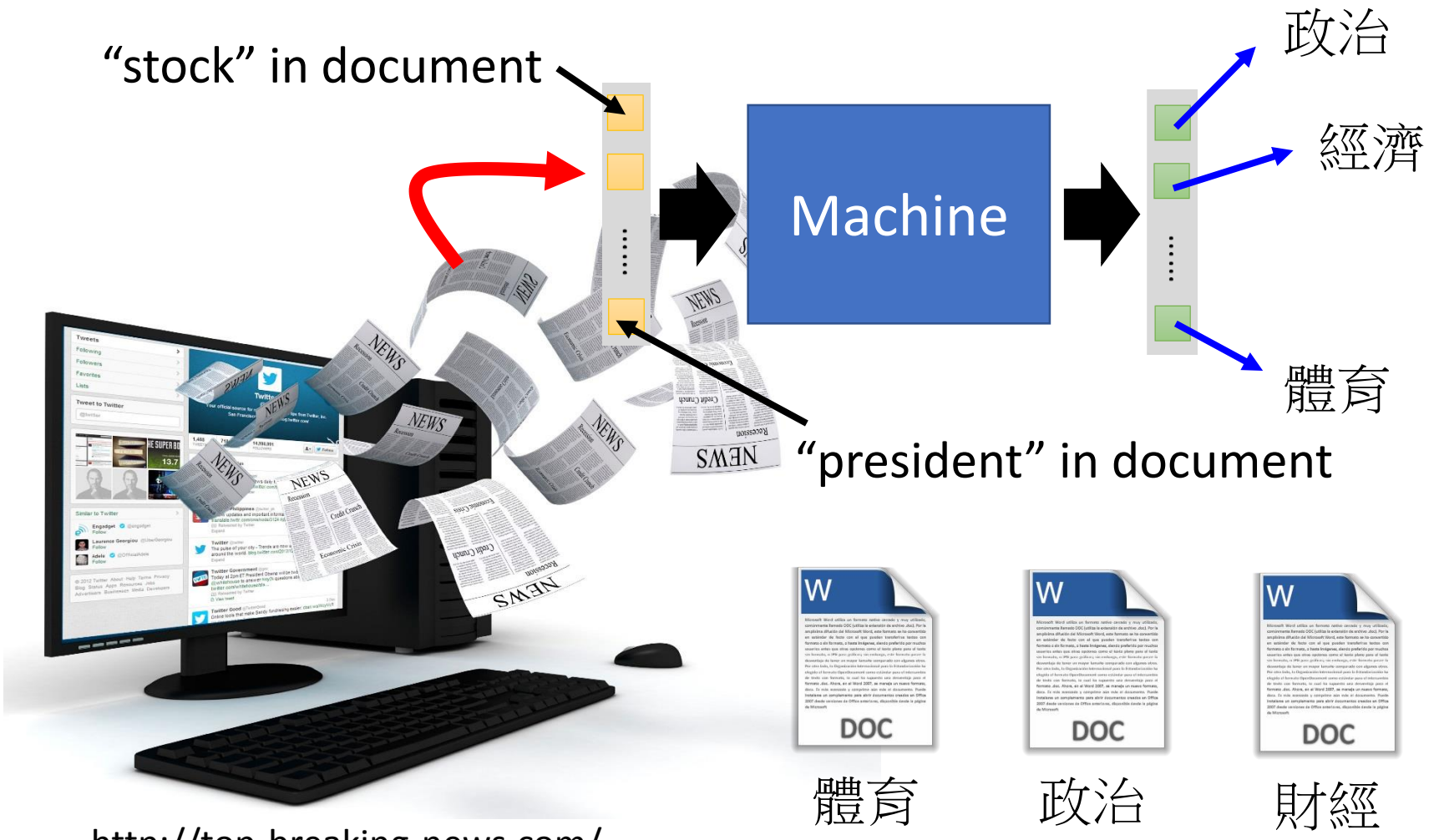
The neurons in the first layer usually detect part of the digits.


28x28
10
$y_1$
$y_2$
$y_{10}$
6
0
8
1
2
5
9
4
3
7


28x28
10
$y_1$
$y_2$
$y_{10}$
6
0
8
2
9
7
5
1
4
3


28x28
10
$y_1$
$y_2$
$y_{10}$
0
1
2
3
4
5
6
7
8
9

# Try another task




http://top-breaking-news.com/

# Try another task

```
In [8]: x_train.shape
Out[8]: (8982, 1000)

In [9]: y_train.shape
Out[9]: (8982, 46)

In [10]: x_test.shape
Out[10]: (2246, 1000)

In [11]: y_test.shape
Out[11]: (2246, 46)
```

```
In [12]: x_train[0]
Out[12]:
array([ 0.,  1.,  1.,  0.,  1.,  1.,  1.,  1.,  1.,  0.,  0.,  1.,  1.,
        1.,  0.,  1.,  0.,  0.,  1.,  0.,  0.,  1.,  1.,  0.,  1.,  0.,
        0.,  1.,  0.,  0.,  1.,  1.,  0.,  0.,  0.,  1.,  0.,  0.,  0.,
        0.,  0.,  0.,  0.,  0.,  0.,  0.,  0.,  0.,  0.,  0.,  0.,  0.,
        0.,  0.,  1.,  0.,  0.,  0.,  0.,  0.,  0.,  0.,  0.,  0.,  0.,
        0.,  0.,  0.,  0.,  0.,  1.,  1.,  0.,  0.,  0.,  0.,  1.,  1.,
        0.,  0.,  0.,  0.,  0.,  0.,  0.,  0.,  0.,  0.,  0.,  1.,  0.,
        0.,  0.,  0.,  0.,  0.,  1.,  0.,  1.,  0.,  0.,  0.,  0.,  0.,
        0.,  0.,  0.,  0.,  0.,  0.,  0.,  1.,  0.,  0.,  0.,  0.,  0.,
        0.,  0.,  1.,  0.,  1.,  0.,  0.,  0.,  0.,  0.,  0.,  0.,  0.,
        0.,  0.,  0.,  0.,  0.,  0.,  0.,  0.,  1.,  0.,  0.,  1.,  1.,
        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.,  1.,  0.,  0.,  0.,  0.,  0.,  0.,  0.,  0.,
        0.,  0.,  1.,  0.,  0.,  0.,  0.,  0.,  0.,  0.,  0.,  0.,  1.,
        0.,  1.,  0.,  0.,  0.,  0.,  0.,  0.,  0.,  0.,  0.,  0.,  0.,
        0.,  0.,  0.,  0.,  0.,  0.,  0.,  0.,  0.,  0.,  0.,  0.,  0.,
```

```
In [13]: y_train[0]
Out[13]:
array([ 0.,  0.,  0.,  1.,  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.])
```

# Live Demo