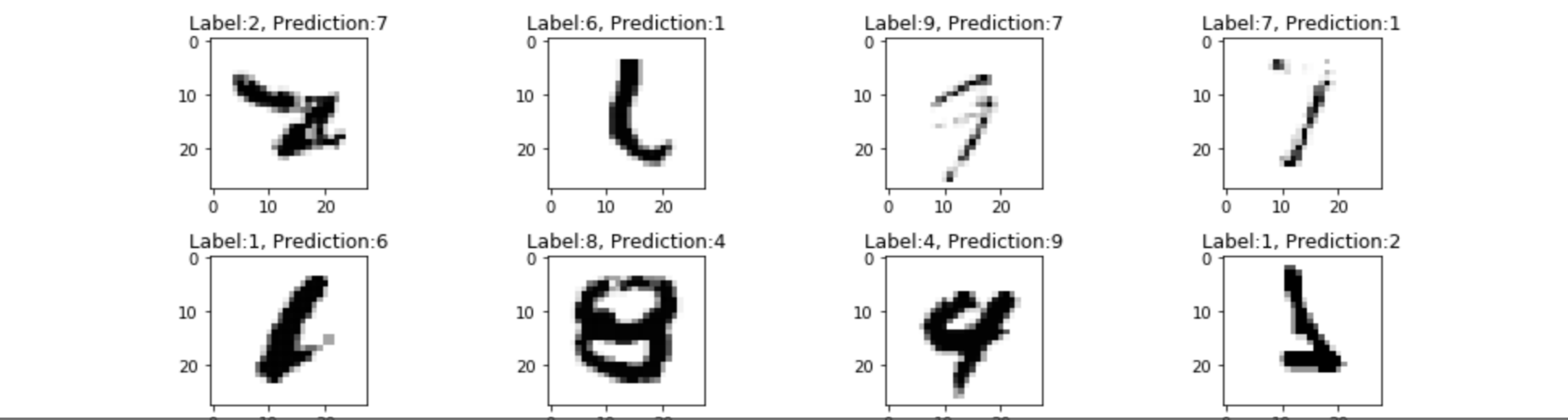
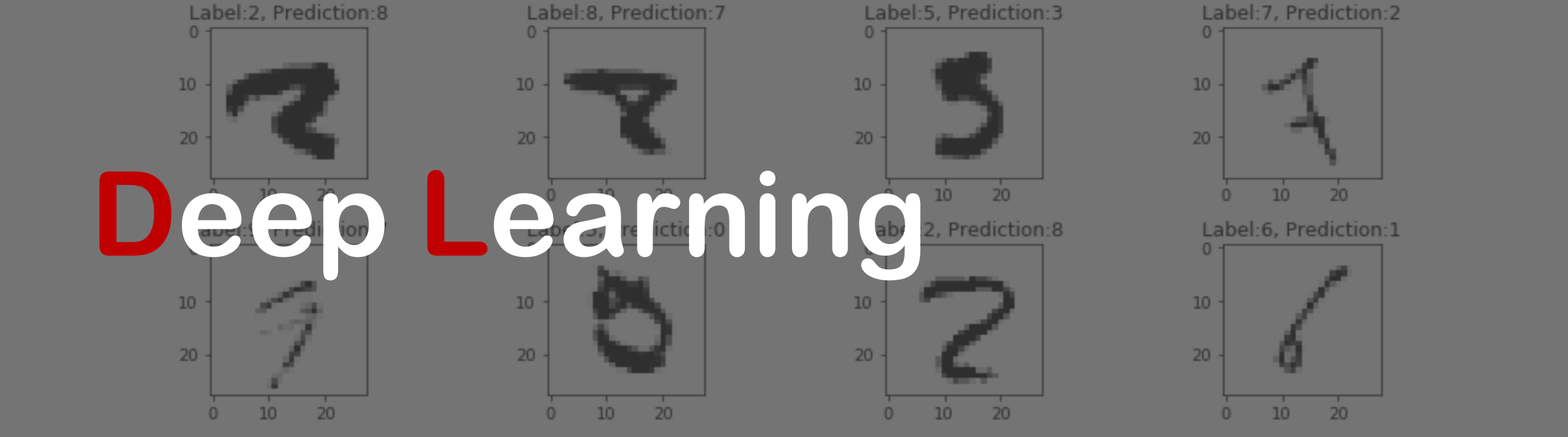
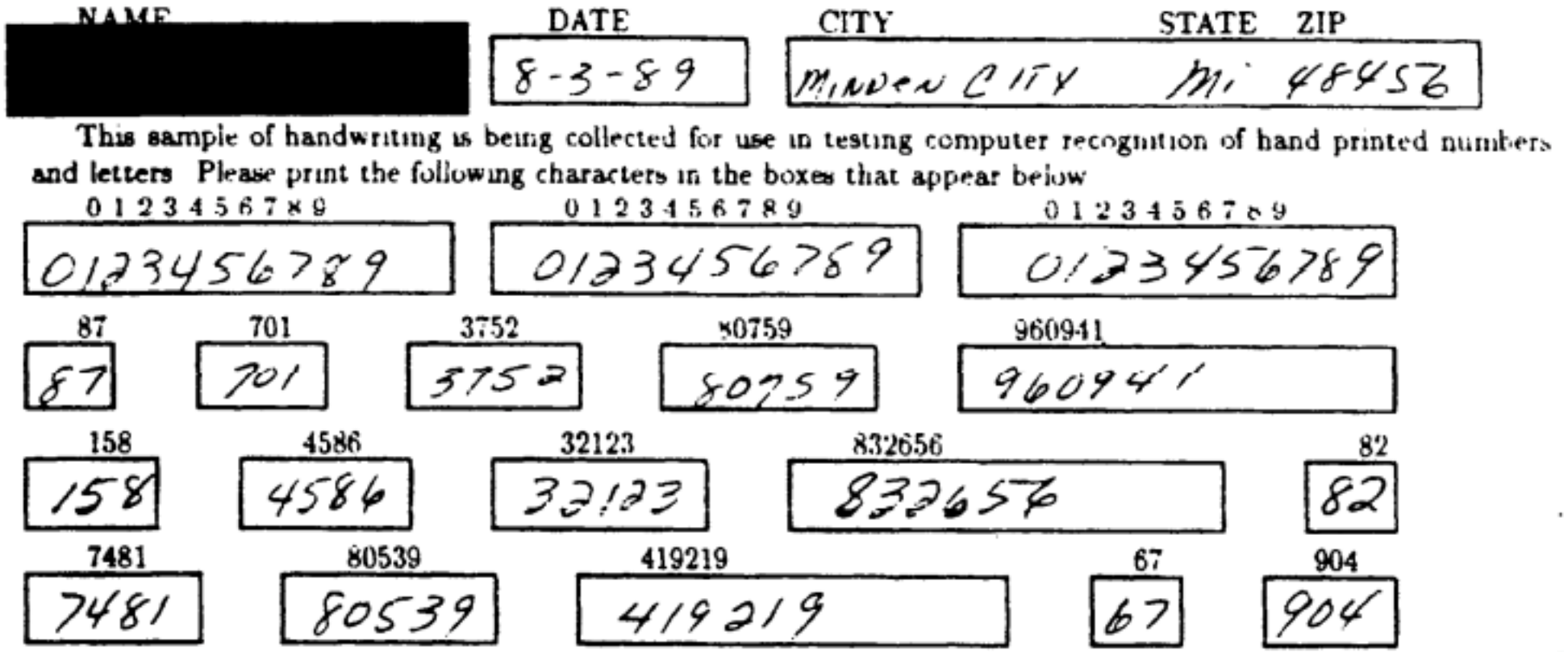




Deep Learning



- **NIST** 데이터 셋 **(National Institute of Standards and Technology)**

- **MNIST** 데이터 셋 **(Modified National Institute of Standards and Technology)**

- **28*28** 픽셀의 **0~9** 사이의 숫자 이미지와 레이블로 구성된 셋
- 머신 러닝을 공부하는 사람들이 입문용으로 사용함
- **60000**개의 훈령용 셋과 **10000**개의 실험용 셋으로 구성
- 이 데이터를 데리고, 학습한 다음, 실제 정답률이 어느 정도인지 맞춰 봐요 **~~**

- **MNIST** 필기 숫자를 학습하는 과정...
- 원래 이 예제는 **CNN(Convolutional Neural Network)**이 정석이라고 알려져 있지만
- 우리는 딥러닝의 기초에 집중하도록 하자.

In [1]:

```
import matplotlib.pyplot as plt
%matplotlib inline

import tensorflow as tf
import keras
from keras.models import Sequential
from keras.layers import Dense
import numpy as np
np.random.seed(13)

print('TensorFlow version : ', tf.__version__)
print('Keras version : ', keras.__version__)
print('Numpy version : ', np.__version__)
```

[Keras] Simple Layer for MNIST

In [2]:
```
# load data
(X_train, y_train), (X_test, y_test) = keras.datasets.mnist.load_data()
```

In [3]:
```
# flatten 28*28 images to a 784 vector for each image
num_pixels = X_train.shape[1] * X_train.shape[2]
X_train = X_train.reshape(X_train.shape[0], num_pixels).astype('float32')
X_test = X_test.reshape(X_test.shape[0], num_pixels).astype('float32')
```

In [4]:
```
# normalize inputs from 0-255 to 0-1
X_train = X_train / 255
X_test = X_test / 255
```

- **Keras**는 **MNIST** 데이터를 가지고 있다.
- 입력 데이터로 사용할 수 있도록 데이터를 수정한다.

[Keras] Simple Layer for MNIST

In [5]:

```
import random
samples = random.choices(population=range(0,60000), k=16)
samples
```

```
[33409,
 6650,
 58164,
 19194,
 56267,
 7066,
 36630,
 28517,
 28310,
```

[Keras] Simple Layer for MNIST

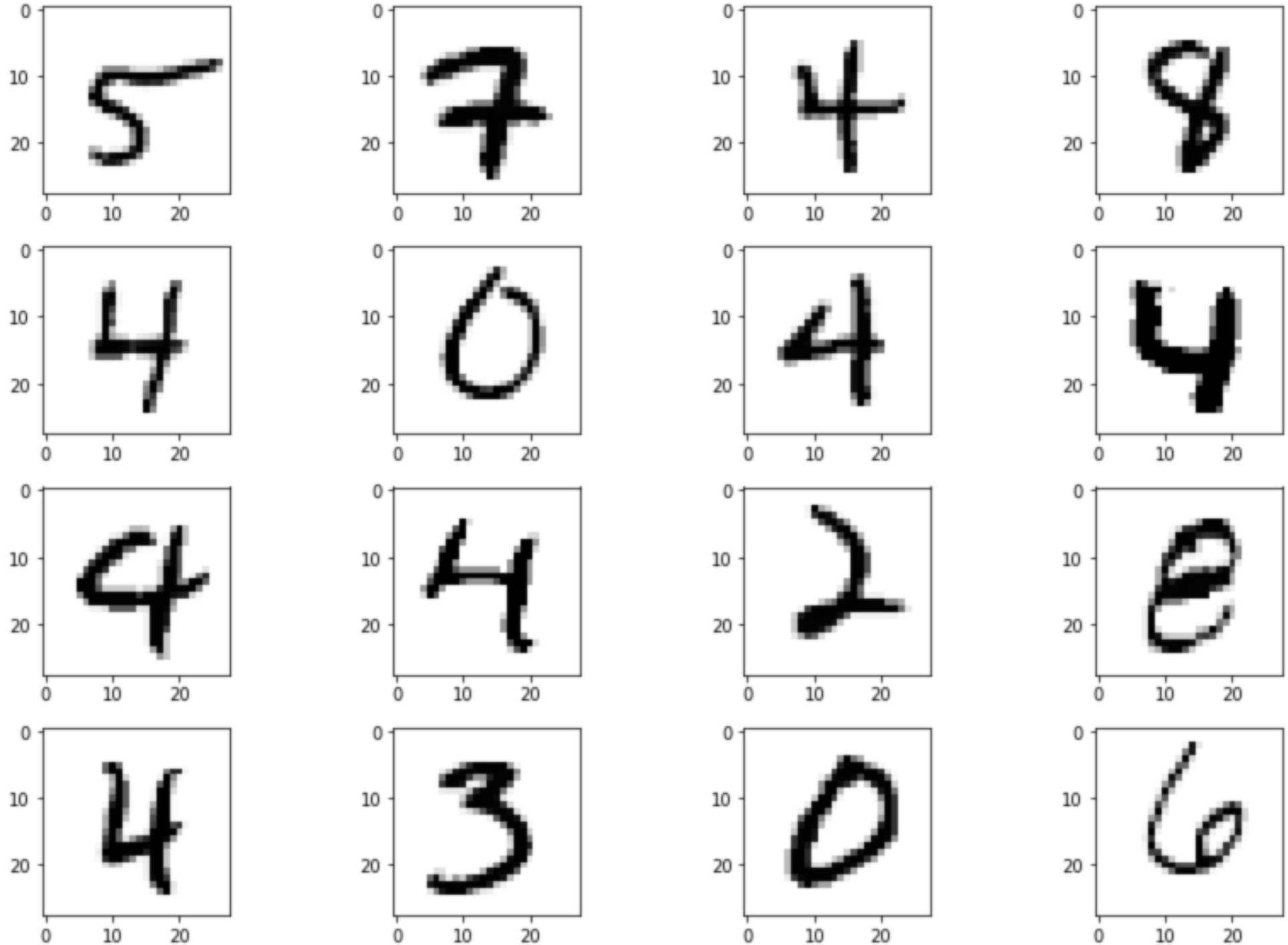
In [6]:

```
count = 0
nrows = ncols = 4
plt.figure(figsize=(12,8))

for n in samples:
    count += 1
    plt.subplot(nrows, ncols, count)
    plt.imshow(X_train[n].reshape(28, 28), cmap='Greys', interpolation='nearest')

plt.tight_layout()
plt.show()
```

- 어떻게 생겼는지 몇 개만 그려보자

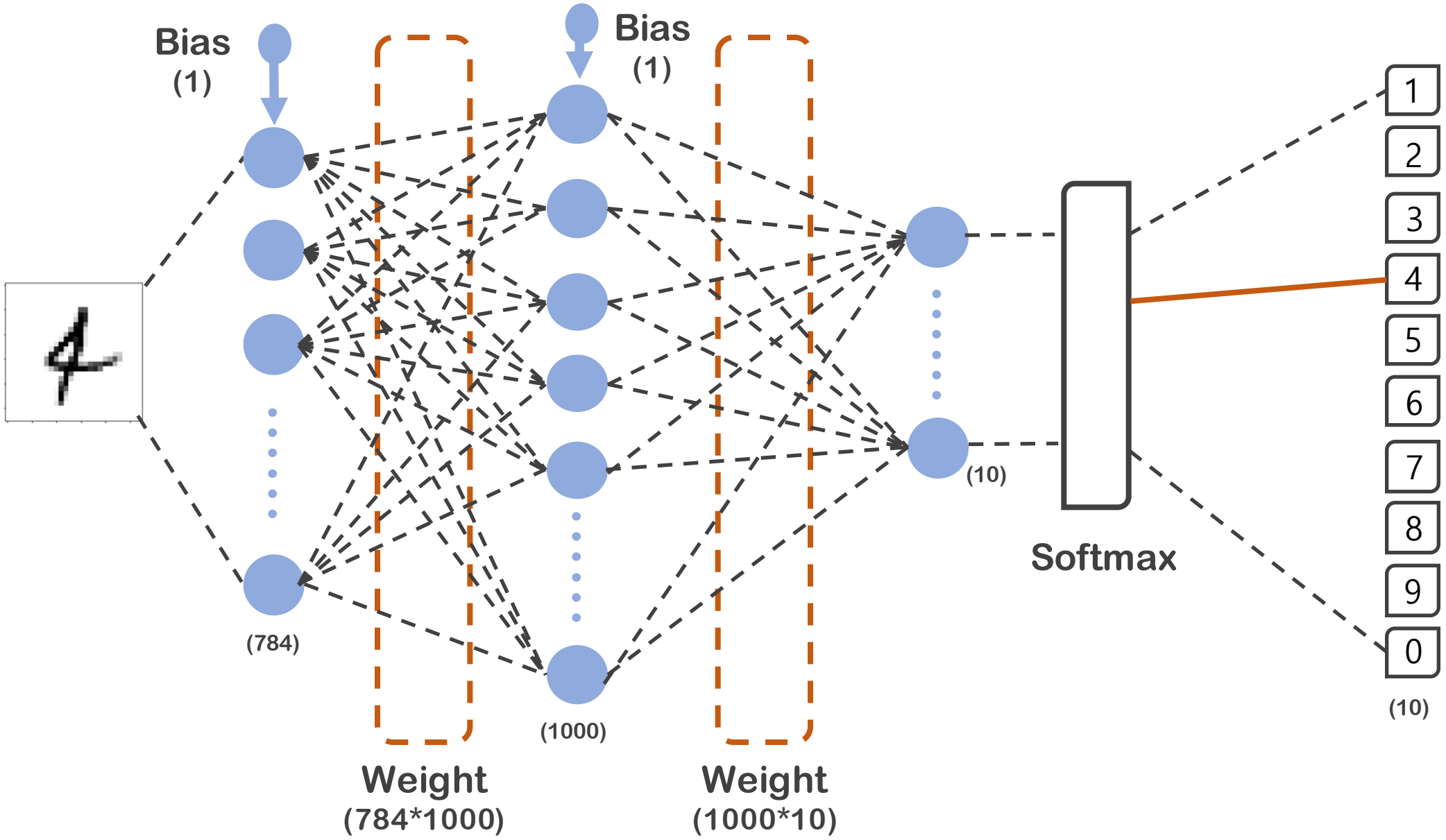


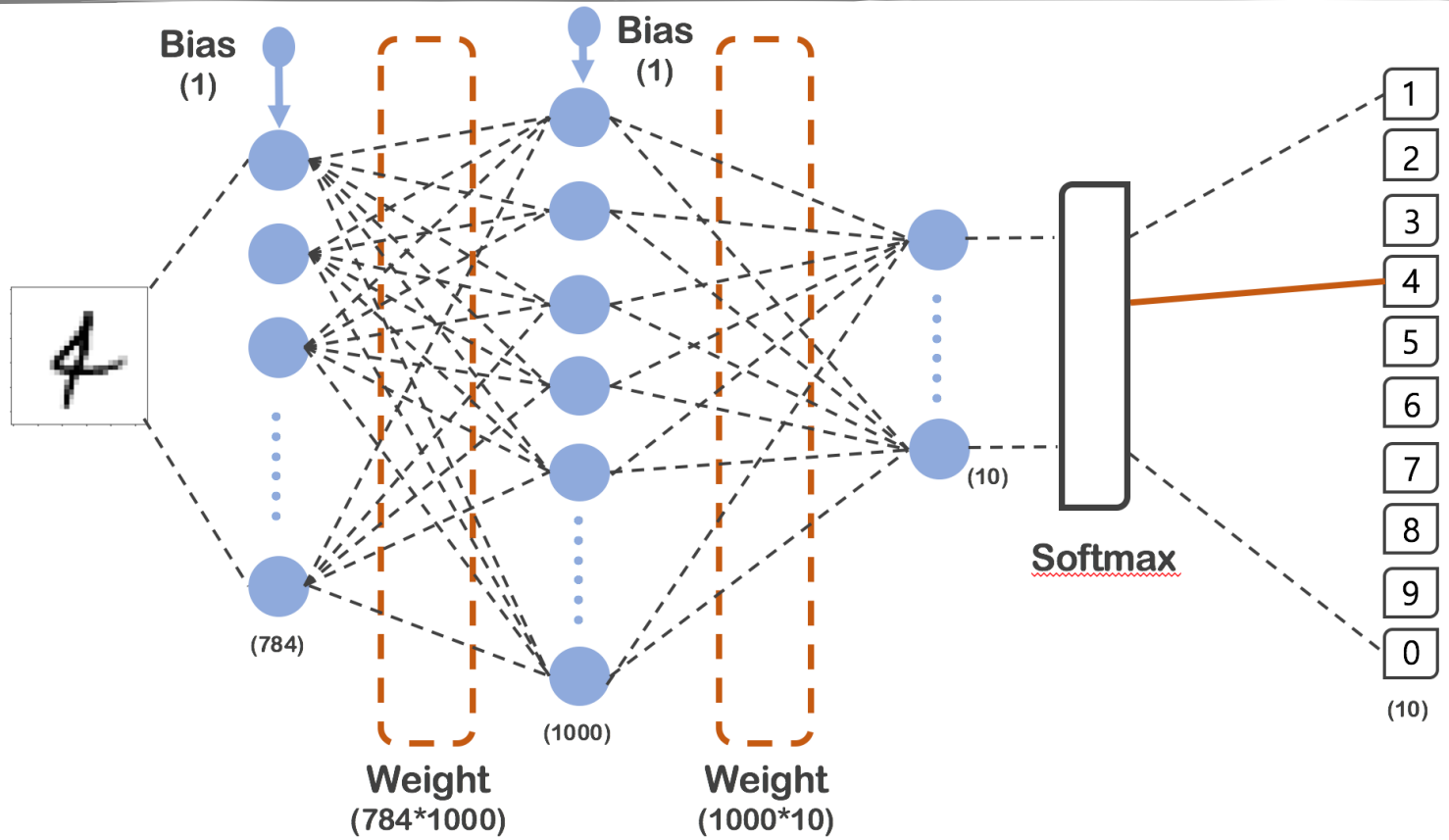
In [7]:

```
# one hot encode outputs
y_train = keras.utils.np_utils.to_categorical(y_train)
y_test = keras.utils.np_utils.to_categorical(y_test)
num_classes = y_test.shape[1]
```

- **Y값들은 One-hot 인코딩을 해서 행렬 연산의 크기를 맞춰준다.**
- **5 → [0, 0, 0, 0, 1, 0, 0, 0, 0, 0]**

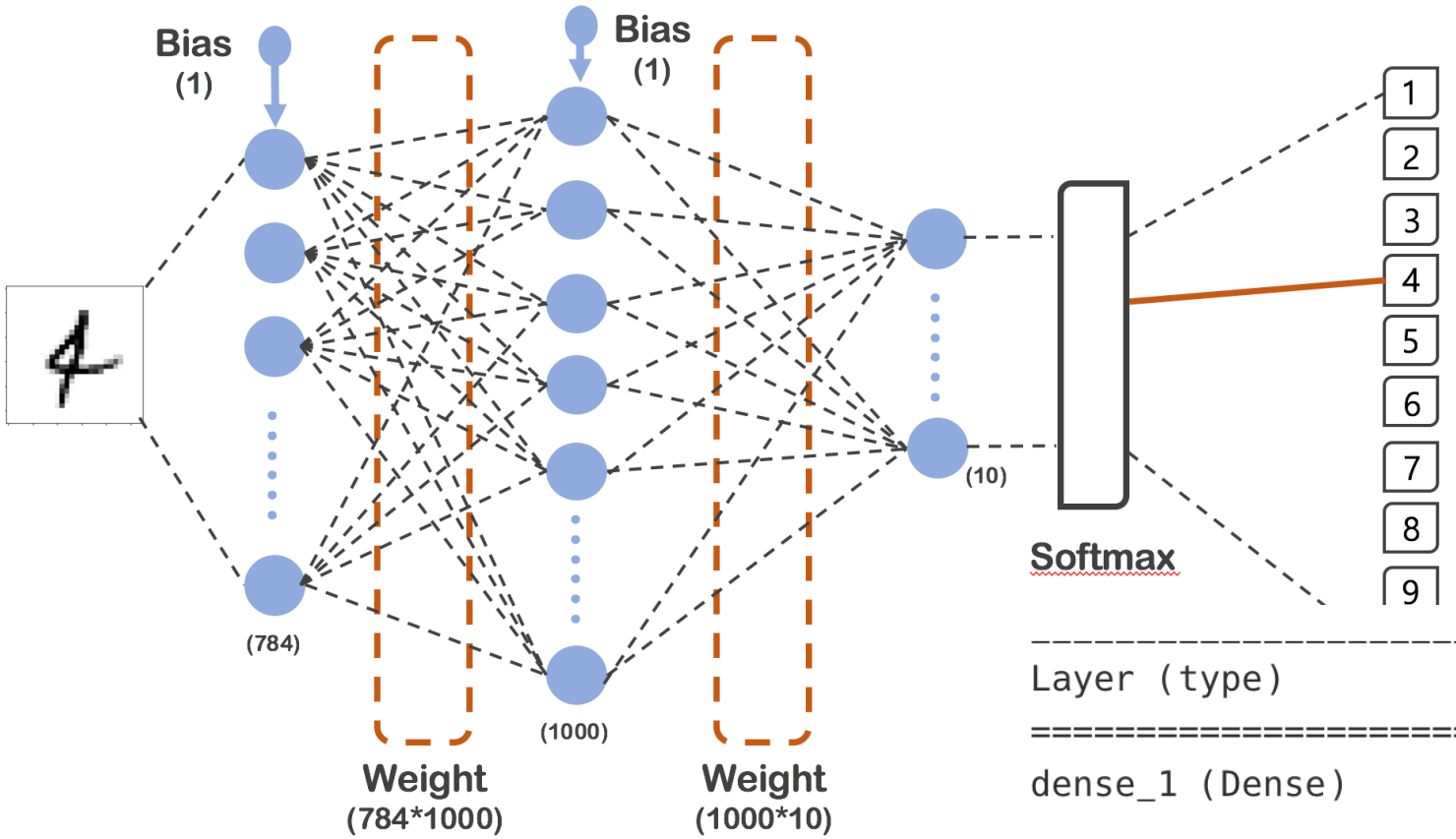
- 데이터는 뭐 그렇다 치자...
- 학습할 모델을 어떻게 정하지**???**


Bias
(1)
Bias
(1)
(784)
(1000)
(10)
Softmax
1
2
3
4
5
6
7
8
9
0
(10)
Weight
(784*1000)
Weight
(1000*10)



In [9]:

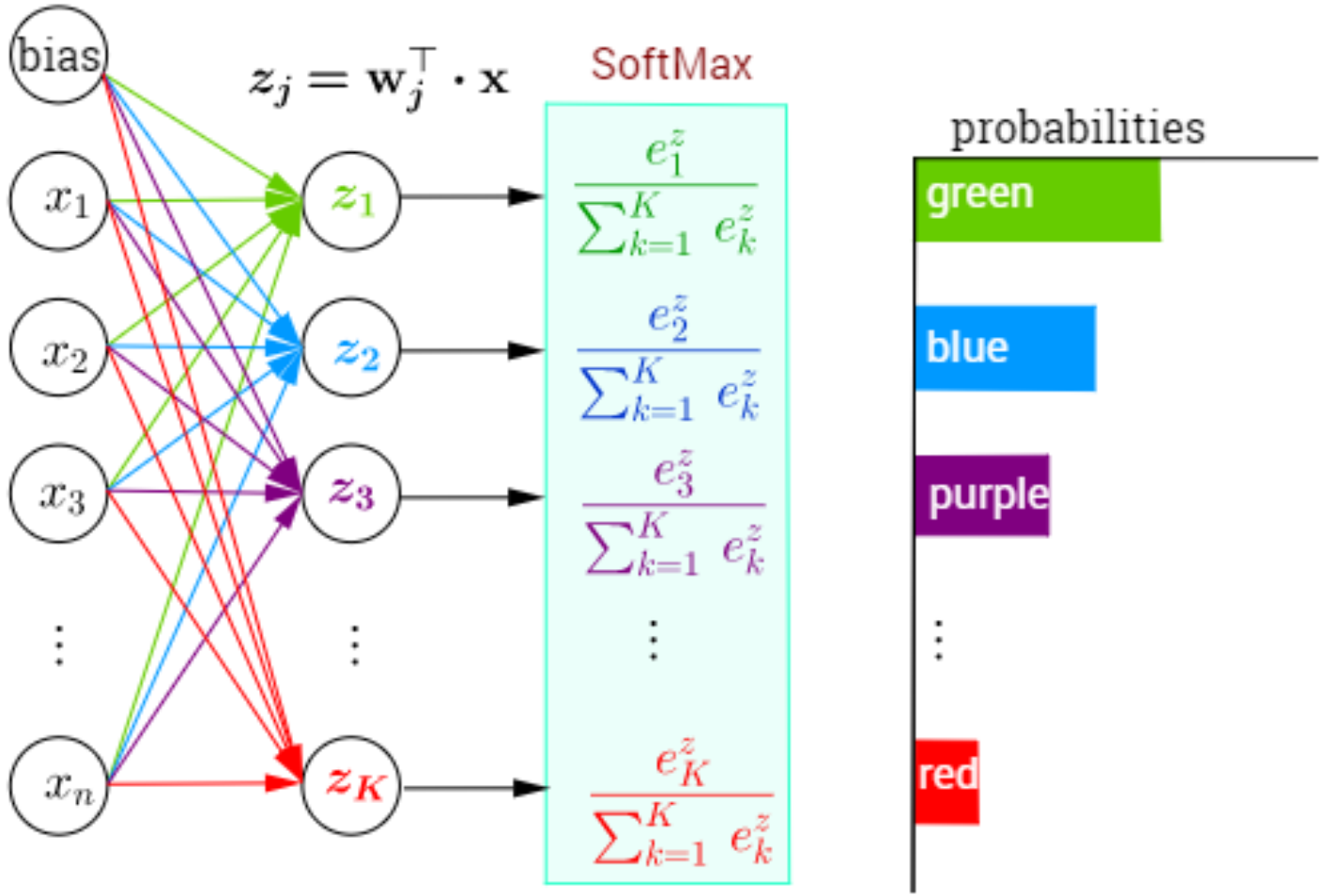
```
model = Sequential()
model.add(Dense(1000, input_dim=num_pixels, activation='relu'))
model.add(Dense(num_classes,activation='softmax'))
model.summary()
```




```
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
dense_1 (Dense)              (None, 1000)              785000
_________________________________________________________________
dense_2 (Dense)              (None, 10)                10010
=================================================================
Total params: 795,010
Trainable params: 795,010
Non-trainable params: 0
_________________________________________________________________
```

Multi-Class Classification with NN and SoftMax Function

$$\mathbf{z} = \begin{bmatrix} z_1 \\ z_2 \\ z_3 \\ \vdots \\ z_K \end{bmatrix} = \begin{bmatrix} \mathbf{w}_1^\top \\ \mathbf{w}_2^\top \\ \mathbf{w}_3^\top \\ \vdots \\ \mathbf{w}_K^\top \end{bmatrix} \begin{bmatrix} x_1 \\ x_2 \\ x_3 \\ \vdots \\ x_n \end{bmatrix}$$

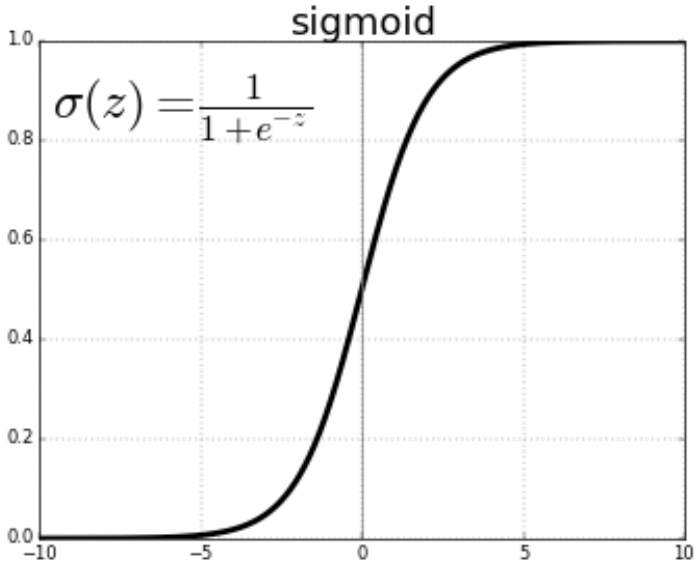


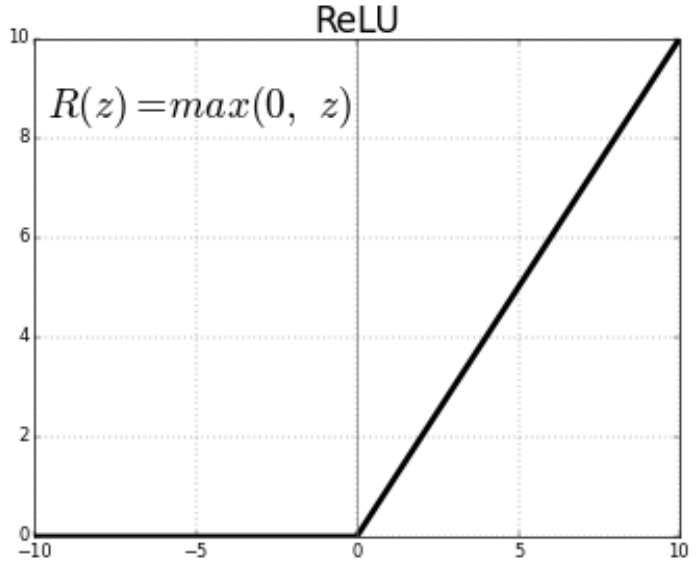
- **Softmax???**
- 범주형으로 결과를 가져야할 때 많이 사용하는 **activation function** 중 하나
- 전체 결과 **[1,2,3]**를 전체 합이 **1**이 되도록 출력을 가진다.

  → **[0.17, 0.33, 0.5]**

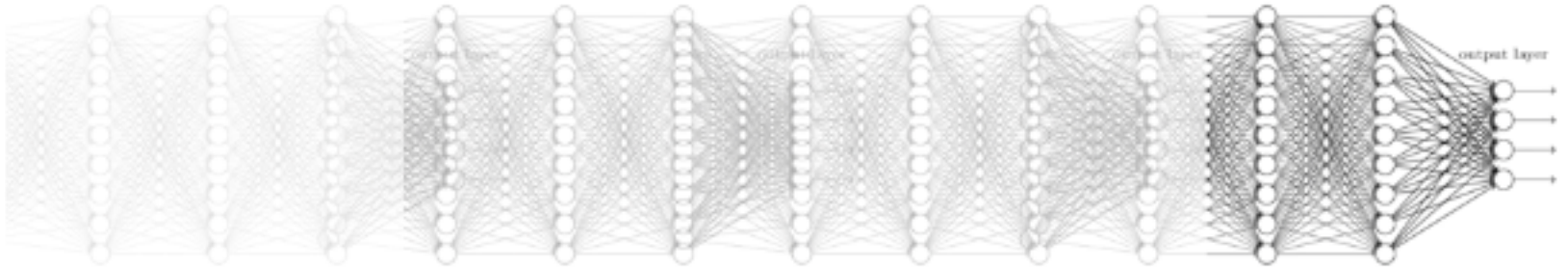
- 응?? 활성 **Activation Function** 함수???
  - 개별 뉴런의 입력의 총합을 출력으로 변환하는 함수
  - **non-linear** 함수를 사용함
  - 그 이유는 합성될때 유의미한 결과를 얻기 위해서임
  - **if h(x)=ax** 이고, **3**번 합성하면 **h(h(h(x)))**이어도, 그냥 **bx**형태여서 의미가 없기 때문에 **non-linear** 함수를 사용함

- 근데 **ReLU(Rectified Linear Unit)**는 뭐지???





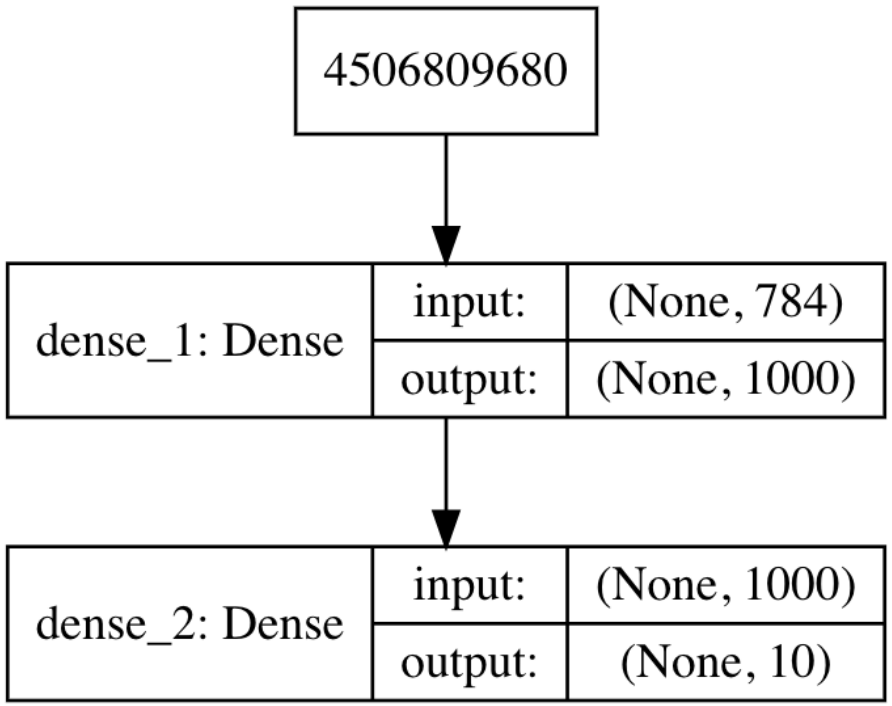
- 에러를 좀 더 크게 가지게 해서 **Gradient Vanishing** 문제를 해결하려 한다



In [10]:

```
from IPython.display import SVG
from keras.utils.vis_utils import model_to_dot
%matplotlib inline

SVG(model_to_dot(model, show_shapes=True).create(prog='dot', format='svg'))
```



In [11]:

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

- 학습 시작 ~
- **loss → categorical crossentropy**
- **Optimizer → adam**

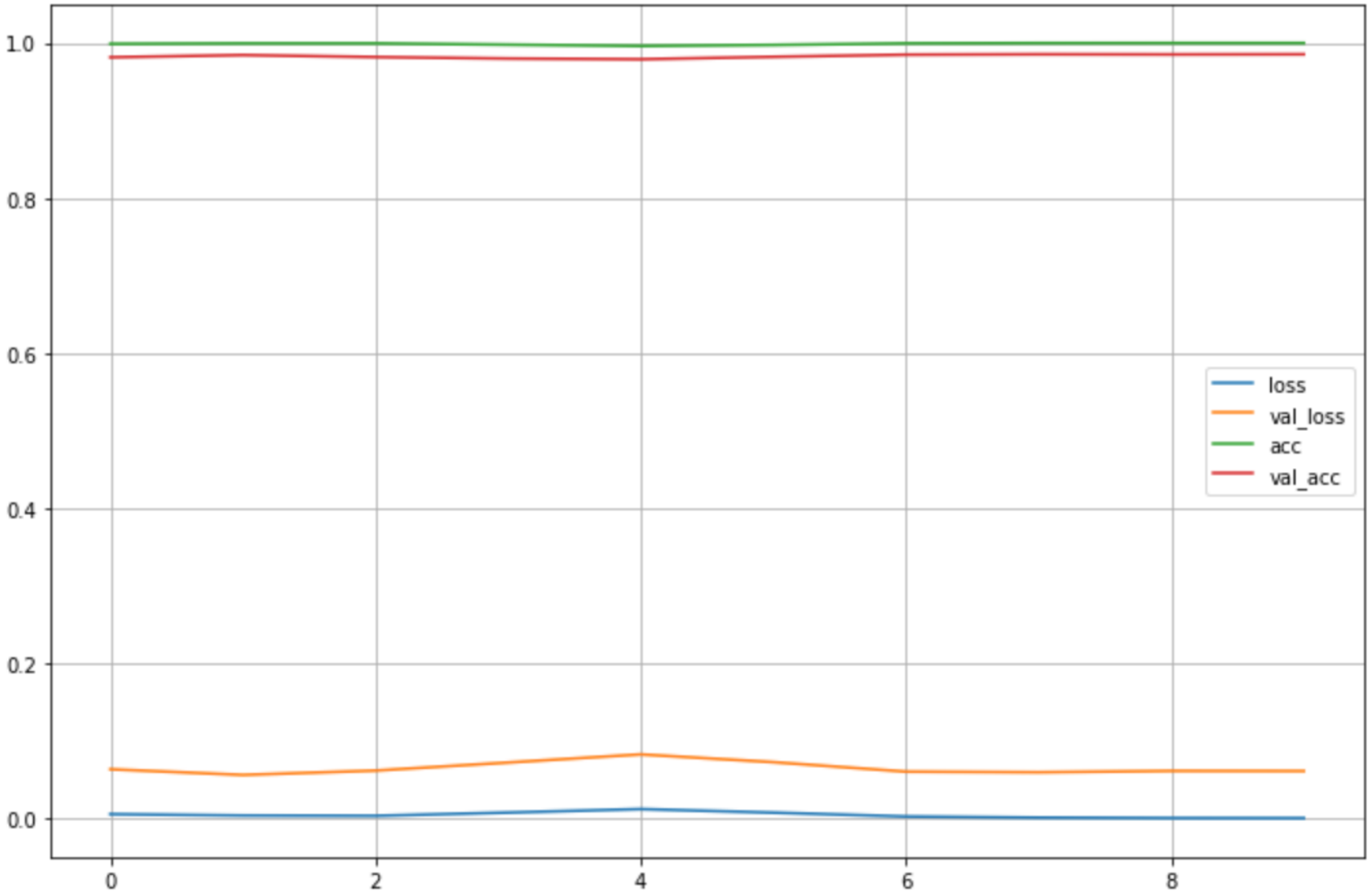
In [*]:

```
hist = model.fit(X_train, y_train,
                 validation_data=(X_test, y_test),
                 epochs=10, batch_size=200, verbose=1)
```

```
Train on 60000 samples, validate on 10000 samples
Epoch 1/10
60000/60000 [==============================] - 3s 57us/step - loss: 0.0054 - acc: 0.999
0 - val_loss: 0.0633 - val_acc: 0.9816
Epoch 2/10
60000/60000 [==============================] - 3s 57us/step - loss: 0.0037 - acc: 0.999
6 - val_loss: 0.0560 - val_acc: 0.9845
```

In [14]:

```
plt.figure(figsize=(12,8))
plt.plot(hist.history['loss'])
plt.plot(hist.history['val_loss'])
plt.plot(hist.history['acc'])
plt.plot(hist.history['val_acc'])
plt.legend(['loss','val_loss', 'acc','val_acc'])
plt.grid()
plt.show()
```


1.0
0.8
0.6
0.4
0.2
0.0
0
2
4
6
8
loss
val_loss
acc
val_acc

In [15]:

```
score = model.evaluate(X_test, y_test, verbose=0)
print('Test loss:', score[0])
print('Test accuracy:', score[1])
```

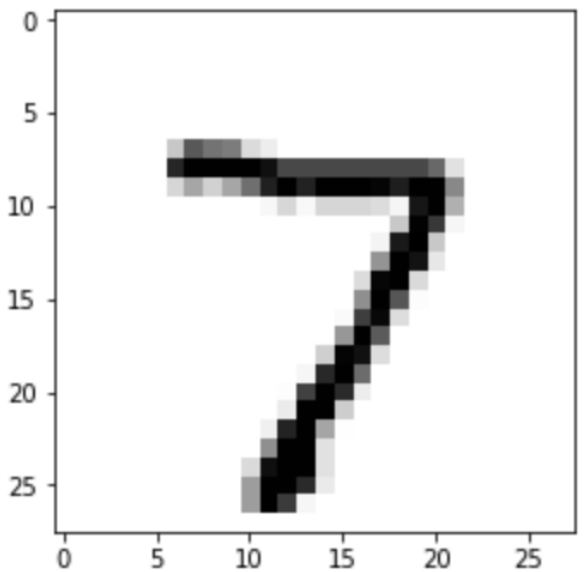
```
Test loss: 0.06105822477729034
Test accuracy: 0.9855
```

- 테스트 데이터에 대해서 검증해야겠지....

In [20]:

```
n = 0
plt.imshow(X_test[n].reshape(28, 28), cmap='Greys', interpolation='nearest')
plt.show()

print('The Answer is ', model.predict_classes(X_test[n].reshape(1,784)))
```




```
The Answer is  [7]
```

- 틀린 결과만 추려서 보자... 왜 틀렸는지 한 번 볼까 ~

In [21]:
```
predicted_result = model.predict(X_test)
predicted_labels = np.argmax(predicted_result, axis=1)
predicted_labels
```

```
array([7, 2, 1, ..., 4, 5, 6])
```

In [22]:
```
test_labels = np.argmax(y_test, axis=1)
test_labels
```

```
array([7, 2, 1, ..., 4, 5, 6])
```

In [23]:

```
wrong_result = []

for n in range(0, len(test_labels)):
    if predicted_labels[n] != test_labels[n]:
        wrong_result.append(n)

len(wrong_result)
```

```
145
```

In [24]:

```
import random
samples = random.choices(population=wrong_result, k=16)
samples
```

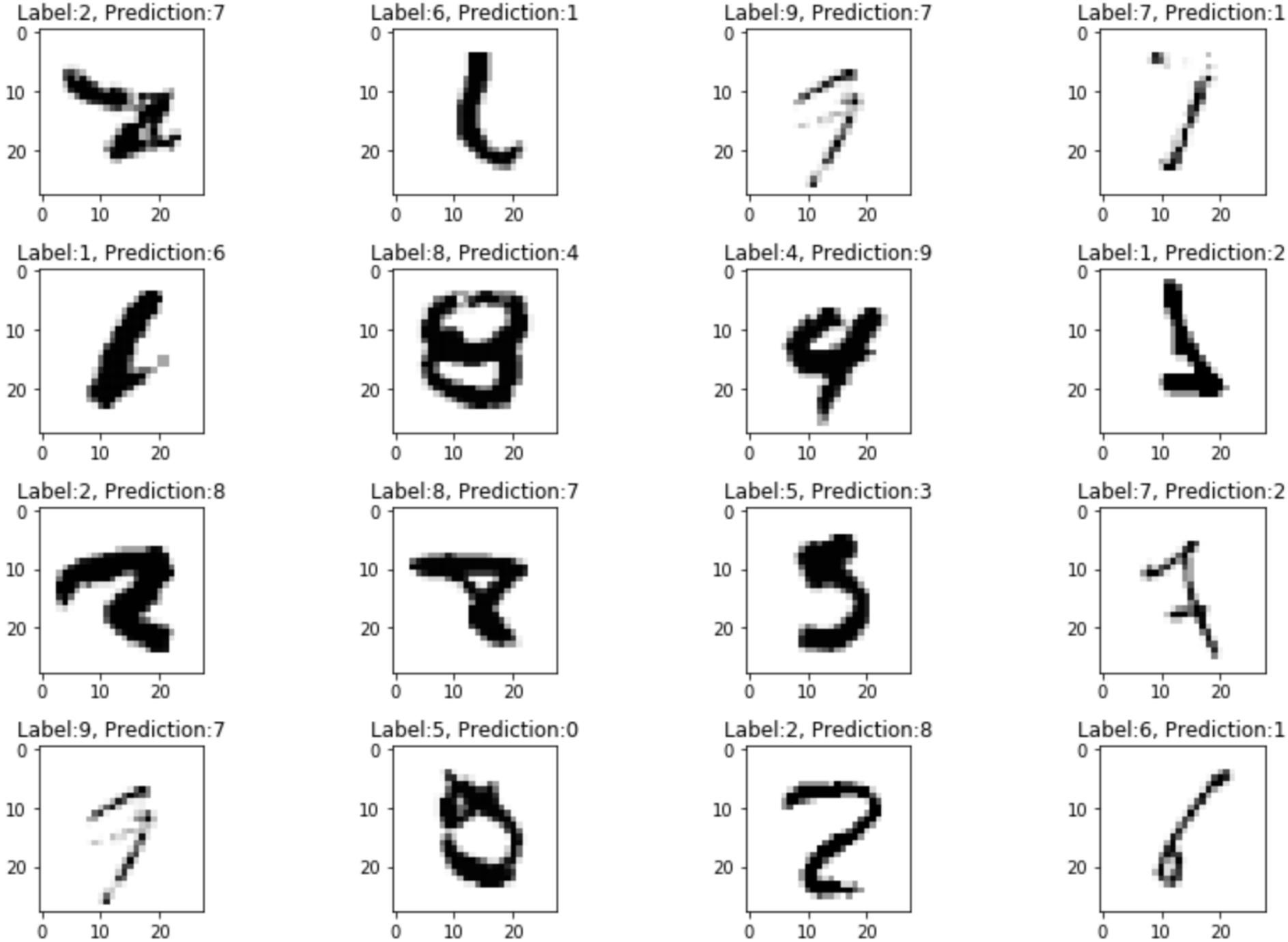
```
[4176,
 2654,
 6571,
 6576,
 6783,
```

In [25]:

```
count = 0
nrows = ncols = 4
plt.figure(figsize=(12,8))

for n in samples:
    count += 1
    plt.subplot(nrows, ncols, count)
    plt.imshow(X_test[n].reshape(28, 28), cmap='Greys', interpolation='nearest')
    tmp = "Label:" + str(test_labels[n]) + ", Prediction:" + str(predicted_labels[n])
    plt.title(tmp)

plt.tight_layout()
plt.show()
```


Label:2, Prediction:7
Label:6, Prediction:1
Label:9, Prediction:7
Label:7, Prediction:1
Label:1, Prediction:6
Label:8, Prediction:4
Label:4, Prediction:9
Label:1, Prediction:2
Label:2, Prediction:8
Label:8, Prediction:7
Label:5, Prediction:3
Label:7, Prediction:2
Label:9, Prediction:7
Label:5, Prediction:0
Label:2, Prediction:8
Label:6, Prediction:1