1. Keras tutorial

- 1.1 Keras Basic(y=3x Regression) - ANN(단층 레이어)
- 1.2 Keras MNIST - ANN(단층 레이어)
- 1.3 Keras MNIST - DNN(다층 레이어)
- 1.4 Keras MNIST - 나만의 모델 만들기

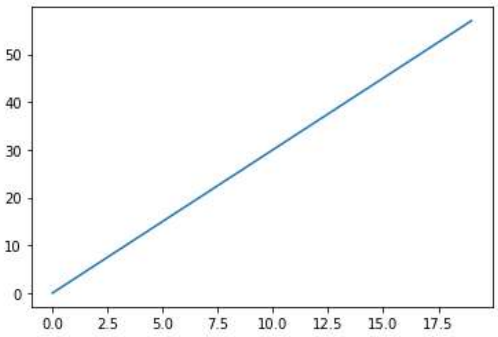
In [1]:
```
# Import Libraries
from tensorflow import keras
import numpy as np
import matplotlib.pyplot as plt
%matplotlib inline
```

1.1 Keras Basic Y = 3X ANN Regression

- 이번 실습에서는 Keras를 이용하여 1차함수를 예측하는 학습 모델을 만들고, 성능을 평가한다.

(1) 데이터셋 준비

In [2]:
```
# Training Data
x = np.arange(20) # 0, 1, 2, 3, ...., 19
y = x * 3 # 0, 3, 6, 9, ..., 57
plt.plot(x,y)

# Testing Data
x_test = np.arange(50,70) # 50, 51, 52, ..., 69
y_test = x_test*3 # 150, 153, 156, ..., 207

# Dimension of input and output
n_in = 1
n_out = 1
```



(2) Keras 모델링

- keras.layers : 모델을 구성하기 위한 layer들이 구현되어 있는 모듈
- keras.models : layer들을 묶어 모델을 정의하고 학습, 평가, 예측 등의 기능을 구현한 모듈

In [3]:
```
from tensorflow.keras import layers, models
```

Keras에서는 크게 Functional, Sequential 방식으로 모델을 구현

- Sequential : 모델에 필요한 layer들을 순차적으로 더해가는 방식으로 구현
- Function : 모델을 수식처럼 구현

<Sequential에 사용되는 Layer>

- Dense : Fully Connected Layer (https://keras.io/layers/core/#dense)

Sequential Modeling

In [4]:
```
def modeling_sequential(n_in, n_out):
    # Coding Time
    model = models.Sequential()
    model.add(layers.Dense(units =n_out, input_shape=(n_in,)))
    return model
```

In [5]:
```
class modeling_sequential_class(models.Sequential):
    def __init__(self, n_in, n_out):

        #멤버 변수로 모델에 사용할 변수 선언
        self.n_in = n_in
        self.n_out = n_out

        #상속받은 Sequential 클래스 초기화 후 레이어 추가
        super().__init__()
        self.add(layers.Dense(units =n_out, input_shape=(n_in,)))
```

<Functional에 사용되는 Layer>

- Input : 모델에 입력되는 데이터의 batch size를 제외한 shape을 결정
- Dense : Fully Connected Layer (https://keras.io/layers/core/#dense)

Functional Modeling

In [6]:
```
def modeling_functional(n_in, n_out):
    # Coding Time
    input = layers.Input(shape=(n_in,))
    y = layers.Dense(n_out)(input)
    model = models.Model(inputs = input, outputs = y)
    return model
```

In [7]:
```
class modeling_functional_class(models.Model):
    def __init__(self, n_in, n_out):

        #멤버 변수로 모델에 사용할 변수 및 레이어 선언
        self.n_in = n_in
        self.n_out = n_out
        input = layers.Input(shape=(n_in,))
        output = layers.Dense(n_out)

        # layer 연결
        x = input
        y = output(x)

        #상속받은 Model 클래스 초기화
        super().__init__(x, y)
```

(2)-2 모델 시각화

model.summary()나 plot_model(model) 활용

In [8]:
```
model =  modeling_sequential(n_in, n_out)
model.summary()
```

```
Model: "sequential"
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 dense (Dense)               (None, 1)                 2

=================================================================
Total params: 2
Trainable params: 2
Non-trainable params: 0
_________________________________________________________________
```

In [9]:
```
model =  modeling_sequential_class(n_in, n_out)
model.summary()
```

```
Model: "modeling_sequential_class"
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 dense_1 (Dense)             (None, 1)                 2

=================================================================
Total params: 2
Trainable params: 2
Non-trainable params: 0
_________________________________________________________________
```

In [10]:
```
model =  modeling_functional(n_in, n_out)
model.summary()
```

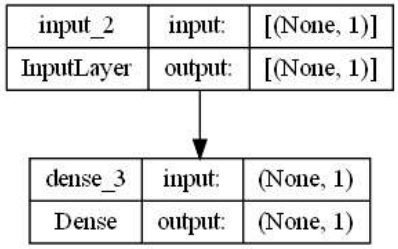
```
Model: "model"
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 input_1 (InputLayer)        [(None, 1)]               0

 dense_2 (Dense)             (None, 1)                 2

=================================================================
Total params: 2
Trainable params: 2
Non-trainable params: 0
_________________________________________________________________
```

In [11]:
```
model = modeling_functional_class(n_in, n_out)
model.summary()
```

```
Model: "modeling_functional_class"
_________________________________________________________________
 Layer (type)                 Output Shape              Param #
=================================================================
 input_2 (InputLayer)         [(None, 1)]               0

 dense_3 (Dense)              (None, 1)                 2

=================================================================
Total params: 2
Trainable params: 2
Non-trainable params: 0
_________________________________________________________________
```

In [12]:
```
from tensorflow.keras.utils import plot_model
plot_model(model, show_shapes=True)
```

Out[12]:



(3) 모델의 학습과정 설정(model.compile)

- model.compile(loss, optimizer, metrics)
- optimizer(str) : optimizer instance
- loss(str) : loss function
- metrics(str) : list of metrics to be evaluated by the model during training and testing (https://www.tensorflow.org/api_docs/python/tf/keras/Model)

In [13]:
```
# Coding Time
model.compile(loss='mse', optimizer='sgd')
```

(4) 모델 학습시키기(model.fit)

- x : Input data
- y : Label of training input data
- batch_szie(int) : Number of samples per gradient update
- steps_per_epoch(int) : Total number of steps (batches of samples) before declaring one epoch finished and starting the next epoch
- epochs(int) : Number of epochs to train the model
- verbose : 0 = silent, 1 = progress bar, 2 = one line per epoch
- callbacks : List of callback instances
- validation_split(float) : Fraction of the training data to be used as validation data
- validation_data : (x_val, y_val)
- shuffle(bool) : Whether to shuffle the training data before each epoch
- history : 학습과정이 담겨있는 데이터 송출 (https://www.tensorflow.org/api_docs/python/tf/keras/Model)

In [14]:
```
# Coding Time
history = model.fit(x, y, batch_size=5, epochs=100, validation_split=0.2)
```

```
Epoch 1/100
4/4 [==============================] - 2s 44ms/step - loss: 102.3446 - val_loss: 32.1718
Epoch 2/100
4/4 [==============================] - 0s 7ms/step - loss: 4.2392 - val_loss: 1.5640
Epoch 3/100
4/4 [==============================] - 0s 6ms/step - loss: 0.3560 - val_loss: 0.1200
Epoch 4/100
4/4 [==============================] - 0s 5ms/step - loss: 0.0119 - val_loss: 0.0914
Epoch 5/100
4/4 [==============================] - 0s 5ms/step - loss: 0.0160 - val_loss: 0.0696
Epoch 6/100
4/4 [==============================] - 0s 5ms/step - loss: 0.0122 - val_loss: 0.0725
Epoch 7/100
4/4 [==============================] - 0s 5ms/step - loss: 0.0101 - val_loss: 0.0366
Epoch 8/100
4/4 [==============================] - 0s 5ms/step - loss: 0.0082 - val_loss: 0.0515
Epoch 9/100
4/4 [==============================] - 0s 6ms/step - loss: 0.0155 - val_loss: 0.0395
Epoch 10/100
4/4 [==============================] - 0s 5ms/step - loss: 0.0104 - val_loss: 0.2372
Epoch 11/100
4/4 [==============================] - 0s 5ms/step - loss: 0.0679 - val_loss: 0.0211
Epoch 12/100
4/4 [==============================] - 0s 5ms/step - loss: 0.0086 - val_loss: 0.0496
Epoch 13/100
4/4 [==============================] - 0s 5ms/step - loss: 0.0075 - val_loss: 0.0060
Epoch 14/100
4/4 [==============================] - 0s 5ms/step - loss: 0.0123 - val_loss: 1.9271e-04
Epoch 15/100
4/4 [==============================] - 0s 5ms/step - loss: 0.0077 - val_loss: 0.0306
Epoch 16/100
4/4 [==============================] - 0s 5ms/step - loss: 0.0058 - val_loss: 0.0015
Epoch 17/100
4/4 [==============================] - 0s 5ms/step - loss: 0.0059 - val_loss: 0.0077
Epoch 18/100
4/4 [==============================] - 0s 5ms/step - loss: 0.0057 - val_loss: 0.0081
Epoch 19/100
4/4 [==============================] - 0s 5ms/step - loss: 0.0050 - val_loss: 0.0438
Epoch 20/100
4/4 [==============================] - 0s 5ms/step - loss: 0.0081 - val_loss: 0.2210
Epoch 21/100
4/4 [==============================] - 0s 5ms/step - loss: 0.0097 - val_loss: 0.0445
Epoch 22/100
4/4 [==============================] - 0s 5ms/step - loss: 0.0337 - val_loss: 0.4224
Epoch 23/100
4/4 [==============================] - 0s 5ms/step - loss: 0.0417 - val_loss: 0.0750
Epoch 24/100
4/4 [==============================] - 0s 5ms/step - loss: 0.0118 - val_loss: 0.0252
Epoch 25/100
4/4 [==============================] - 0s 5ms/step - loss: 0.0073 - val_loss: 1.4391e-04
Epoch 26/100
4/4 [==============================] - 0s 5ms/step - loss: 0.0045 - val_loss: 7.3620e-04
Epoch 27/100
4/4 [==============================] - 0s 5ms/step - loss: 0.0052 - val_loss: 0.0366
Epoch 28/100
4/4 [==============================] - 0s 5ms/step - loss: 0.0076 - val_loss: 0.0057
Epoch 29/100
4/4 [==============================] - 0s 5ms/step - loss: 0.0032 - val_loss: 0.0255
Epoch 30/100
4/4 [==============================] - 0s 5ms/step - loss: 0.0043 - val_loss: 0.0175
Epoch 31/100
4/4 [==============================] - 0s 5ms/step - loss: 0.0058 - val_loss: 0.0472
Epoch 32/100
4/4 [==============================] - 0s 5ms/step - loss: 0.0060 - val_loss: 0.0345
Epoch 33/100
4/4 [==============================] - 0s 5ms/step - loss: 0.0073 - val_loss: 5.4137e-04
Epoch 34/100
4/4 [==============================] - 0s 5ms/step - loss: 0.0037 - val_loss: 0.0129
Epoch 35/100
4/4 [==============================] - 0s 5ms/step - loss: 0.0033 - val_loss: 0.0144
Epoch 36/100
4/4 [==============================] - 0s 5ms/step - loss: 0.0035 - val_loss: 2.3315e-04
Epoch 37/100
4/4 [==============================] - 0s 5ms/step - loss: 0.0043 - val_loss: 0.0093
Epoch 38/100
4/4 [==============================] - 0s 5ms/step - loss: 0.0024 - val_loss: 0.0125
Epoch 39/100
4/4 [==============================] - 0s 5ms/step - loss: 0.0022 - val_loss: 0.0110
Epoch 40/100
4/4 [==============================] - 0s 5ms/step - loss: 0.0020 - val_loss: 0.0132
Epoch 41/100
4/4 [==============================] - 0s 5ms/step - loss: 0.0025 - val_loss: 0.0202
Epoch 42/100
4/4 [==============================] - 0s 5ms/step - loss: 0.0022 - val_loss: 0.0027
Epoch 43/100
4/4 [==============================] - 0s 5ms/step - loss: 0.0018 - val_loss: 0.0111
Epoch 44/100
4/4 [==============================] - 0s 5ms/step - loss: 0.0037 - val_loss: 0.0134
Epoch 45/100
4/4 [==============================] - 0s 5ms/step - loss: 0.0024 - val_loss: 0.0042
Epoch 46/100
4/4 [==============================] - 0s 5ms/step - loss: 0.0018 - val_loss: 0.0098
Epoch 47/100
4/4 [==============================] - 0s 5ms/step - loss: 0.0026 - val_loss: 0.0054
Epoch 48/100
4/4 [==============================] - 0s 5ms/step - loss: 0.0025 - val_loss: 0.0113
Epoch 49/100
4/4 [==============================] - 0s 5ms/step - loss: 0.0017 - val_loss: 0.0057
Epoch 50/100
4/4 [==============================] - 0s 5ms/step - loss: 0.0016 - val_loss: 0.0039
```

```
Epoch 51/100
4/4 [==============================] - 0s 5ms/step - loss: 0.0043 - val_loss: 0.0090
Epoch 52/100
4/4 [==============================] - 0s 5ms/step - loss: 0.0014 - val_loss: 0.0092
Epoch 53/100
4/4 [==============================] - 0s 5ms/step - loss: 0.0015 - val_loss: 0.0161
Epoch 54/100
4/4 [==============================] - 0s 5ms/step - loss: 0.0021 - val_loss: 1.7485e-05
Epoch 55/100
4/4 [==============================] - 0s 5ms/step - loss: 0.0011 - val_loss: 0.0040
Epoch 56/100
4/4 [==============================] - 0s 5ms/step - loss: 9.7933e-04 - val_loss: 0.0055
Epoch 57/100
4/4 [==============================] - 0s 5ms/step - loss: 9.9249e-04 - val_loss: 0.0077
Epoch 58/100
4/4 [==============================] - 0s 5ms/step - loss: 0.0022 - val_loss: 2.2935e-04
Epoch 59/100
4/4 [==============================] - 0s 5ms/step - loss: 9.3436e-04 - val_loss: 8.9884e-05
Epoch 60/100
4/4 [==============================] - 0s 5ms/step - loss: 0.0011 - val_loss: 0.0069
Epoch 61/100
4/4 [==============================] - 0s 5ms/step - loss: 0.0018 - val_loss: 6.9954e-04
Epoch 62/100
4/4 [==============================] - 0s 5ms/step - loss: 0.0017 - val_loss: 0.0060
Epoch 63/100
4/4 [==============================] - 0s 5ms/step - loss: 8.7633e-04 - val_loss: 3.2037e-04
Epoch 64/100
4/4 [==============================] - 0s 5ms/step - loss: 9.8642e-04 - val_loss: 0.0112
Epoch 65/100
4/4 [==============================] - 0s 5ms/step - loss: 0.0033 - val_loss: 0.0025
Epoch 66/100
4/4 [==============================] - 0s 5ms/step - loss: 0.0023 - val_loss: 0.0089
Epoch 67/100
4/4 [==============================] - 0s 5ms/step - loss: 0.0011 - val_loss: 0.0034
Epoch 68/100
4/4 [==============================] - 0s 5ms/step - loss: 9.3668e-04 - val_loss: 0.0050
Epoch 69/100
4/4 [==============================] - 0s 5ms/step - loss: 0.0012 - val_loss: 0.0045
Epoch 70/100
4/4 [==============================] - 0s 5ms/step - loss: 0.0011 - val_loss: 0.0081
Epoch 71/100
4/4 [==============================] - 0s 5ms/step - loss: 0.0043 - val_loss: 0.0032
Epoch 72/100
4/4 [==============================] - 0s 5ms/step - loss: 5.3086e-04 - val_loss: 0.0019
Epoch 73/100
4/4 [==============================] - 0s 5ms/step - loss: 6.8048e-04 - val_loss: 0.0046
Epoch 74/100
4/4 [==============================] - 0s 5ms/step - loss: 0.0013 - val_loss: 2.0700e-04
Epoch 75/100
4/4 [==============================] - 0s 5ms/step - loss: 4.6501e-04 - val_loss: 0.0032
Epoch 76/100
4/4 [==============================] - 0s 5ms/step - loss: 5.4558e-04 - val_loss: 2.4290e-04
Epoch 77/100
4/4 [==============================] - 0s 5ms/step - loss: 4.8460e-04 - val_loss: 0.0034
Epoch 78/100
4/4 [==============================] - 0s 5ms/step - loss: 4.4763e-04 - val_loss: 0.0044
Epoch 79/100
4/4 [==============================] - 0s 5ms/step - loss: 4.4019e-04 - val_loss: 0.0051
Epoch 80/100
4/4 [==============================] - 0s 5ms/step - loss: 0.0013 - val_loss: 0.0056
Epoch 81/100
4/4 [==============================] - 0s 9ms/step - loss: 4.7106e-04 - val_loss: 0.0019
Epoch 82/100
4/4 [==============================] - 0s 6ms/step - loss: 4.2625e-04 - val_loss: 0.0019
Epoch 83/100
4/4 [==============================] - 0s 5ms/step - loss: 0.0011 - val_loss: 0.0038
Epoch 84/100
4/4 [==============================] - 0s 5ms/step - loss: 0.0013 - val_loss: 0.0013
Epoch 85/100
4/4 [==============================] - 0s 5ms/step - loss: 7.7979e-04 - val_loss: 1.7492e-04
Epoch 86/100
4/4 [==============================] - 0s 5ms/step - loss: 3.6943e-04 - val_loss: 0.0040
Epoch 87/100
4/4 [==============================] - 0s 5ms/step - loss: 0.0011 - val_loss: 7.4381e-04
Epoch 88/100
4/4 [==============================] - 0s 5ms/step - loss: 2.5794e-04 - val_loss: 0.0020
Epoch 89/100
4/4 [==============================] - 0s 5ms/step - loss: 3.1308e-04 - val_loss: 9.8511e-04
Epoch 90/100
4/4 [==============================] - 0s 5ms/step - loss: 2.7690e-04 - val_loss: 9.7898e-04
Epoch 91/100
4/4 [==============================] - 0s 5ms/step - loss: 5.2144e-04 - val_loss: 0.0041
Epoch 92/100
4/4 [==============================] - 0s 5ms/step - loss: 6.8726e-04 - val_loss: 3.2261e-04
Epoch 93/100
4/4 [==============================] - 0s 5ms/step - loss: 2.0518e-04 - val_loss: 0.0013
Epoch 94/100
4/4 [==============================] - 0s 5ms/step - loss: 2.5590e-04 - val_loss: 5.6630e-04
Epoch 95/100
4/4 [==============================] - 0s 5ms/step - loss: 1.8360e-04 - val_loss: 0.0012
Epoch 96/100
4/4 [==============================] - 0s 5ms/step - loss: 2.5334e-04 - val_loss: 7.9708e-04
Epoch 97/100
4/4 [==============================] - 0s 5ms/step - loss: 1.6520e-04 - val_loss: 0.0011
Epoch 98/100
4/4 [==============================] - 0s 5ms/step - loss: 3.7641e-04 - val_loss: 6.9453e-04
Epoch 99/100
4/4 [==============================] - 0s 5ms/step - loss: 1.7440e-04 - val_loss: 5.9684e-04
Epoch 100/100
4/4 [==============================] - 0s 5ms/step - loss: 3.5578e-04 - val_loss: 4.9640e-04
```

(5) 모델 평가하기

모델 객체의 evaluate 함수로 test 데이터에 대한 모델의 성능을 평가

- x : Input data
- y : Label of testing input data
- batch_size(int) : Number of samples per batch of computation
- steps(int) : Total number of steps (batches of samples) before declaring the evaluation round finished
- verbose : 0 = silent, 1 = progress bar, 2 = one line per epoch

In [15]:
```
# Coding Time
loss= model.evaluate(x_test, y_test, batch_size=20)
print('loss : %.4f'%(loss))
```

```
1/1 [==============================] - 0s 9ms/step - loss: 0.0169
loss : 0.0169
```

(6) 모델 사용하기

모델 객체의 predict 함수로 input 데이터에 대한 모델의 예측결과를 반환

- x : Input data
- batch_size(int) : Number of samples per batch
- steps(int) : Total number of steps (batches of samples) before declaring the prediction round finished
- verbose : 0 = silent(recommend), 1 = progress bar, 2 = one line per epoch

In [16]:
```
new_x = np.arange(100,120)
true_y = new_x*3

pred_y = model.predict(new_x, batch_size=20, verbose = 0)
pred_y = np.reshape(pred_y,(-1,))
for y in zip(new_x, true_y, pred_y):
    print("x: %.2f,  y : %.2f,  y_predict : %.2f"%(y[0], y[1], y[2]))
```

```
x: 100.00,  y : 300.00,  y_predict : 299.77
x: 101.00,  y : 303.00,  y_predict : 302.77
x: 102.00,  y : 306.00,  y_predict : 305.76
x: 103.00,  y : 309.00,  y_predict : 308.76
x: 104.00,  y : 312.00,  y_predict : 311.76
x: 105.00,  y : 315.00,  y_predict : 314.76
x: 106.00,  y : 318.00,  y_predict : 317.75
x: 107.00,  y : 321.00,  y_predict : 320.75
x: 108.00,  y : 324.00,  y_predict : 323.75
x: 109.00,  y : 327.00,  y_predict : 326.75
x: 110.00,  y : 330.00,  y_predict : 329.74
x: 111.00,  y : 333.00,  y_predict : 332.74
x: 112.00,  y : 336.00,  y_predict : 335.74
x: 113.00,  y : 339.00,  y_predict : 338.73
x: 114.00,  y : 342.00,  y_predict : 341.73
x: 115.00,  y : 345.00,  y_predict : 344.73
x: 116.00,  y : 348.00,  y_predict : 347.73
x: 117.00,  y : 351.00,  y_predict : 350.72
x: 118.00,  y : 354.00,  y_predict : 353.72
x: 119.00,  y : 357.00,  y_predict : 356.72
```

1.2 MNIST Classification with Linear Classifier

이번 실습에서는 keras로 linear 모델을 생성하고, MNIST dataset을 학습하여 성능을 평가한다.

In [17]:
```
import tensorflow.keras.utils as utils
from tensorflow.keras import datasets
from tensorflow.keras.models import Sequential, Model
from tensorflow.keras.layers import Input, Dense, Activation

import numpy as np
import matplotlib.pyplot as plt
```

(1) 데이터셋 다운로드

Download the MNIST dataset

MNIST dataset은 28x28 사이즈의 이미지들로 0~9까지의 숫자 10개의 손글씨 이미지로 이루어져있다.

60,000장이 training에 사용되고 10,000장이 test에 사용된다.

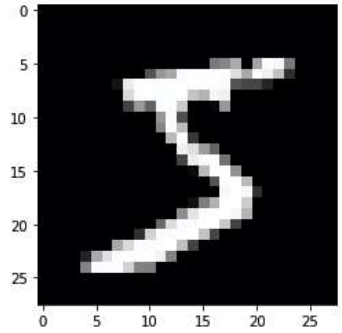
In [18]:
```
# Coding Time
(X_train, Y_train),(X_test, Y_test) = datasets.mnist.load_data()
print(X_train.shape, Y_train.shape)
```

```
(60000, 28, 28) (60000,)
```

In [19]:
```
print('label : ',Y_train[0])
plt.imshow(X_train[0], cmap='gray')
```

```
label :  5
```

Out[19]:
```
<matplotlib.image.AxesImage at 0x20c9f745040>
```



In [20]:
```
# Coding Time
# Flatten and Normalization
X_train_flat = X_train.reshape(60000, 28*28).astype('float32')/255.0
X_test_flat = X_test.reshape(10000, 28*28).astype('float32')/255.0

# One-hot Encoding
Y_train_onehot = utils.to_categorical(Y_train)
Y_test_onehot = utils.to_categorical(Y_test)

print(Y_train_onehot)
print(X_train_flat.shape, Y_train_onehot.shape)
```

```
[[0. 0. 0. ... 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. 1. 0.]]
(60000, 784) (60000, 10)
```

(2) Keras 모델링

<사용되는 Layer>

- Input : 모델에 입력되는 데이터의 batch size를 제외한 shape을 결정(https://keras.io/layers/core/#input)
- Dense : Fully Connected Layer(https://keras.io/layers/core/#dense)
- Activation : Activation Function을 정의, 문자열로 입력(https://keras.io/layers/core/#activation)

In [21]:
```
n_in = 28*28   #784
n_out = np.shape(Y_test_onehot)[1] # 10
```

Sequential Modeling

In [22]:
```
def linear_model_seq(n_in, n_out):
    # Coding Time
    model = Sequential()
    model.add(Dense(units =n_out, input_shape=(n_in,), activation='softmax'))
    return model
```

Functional Modeling

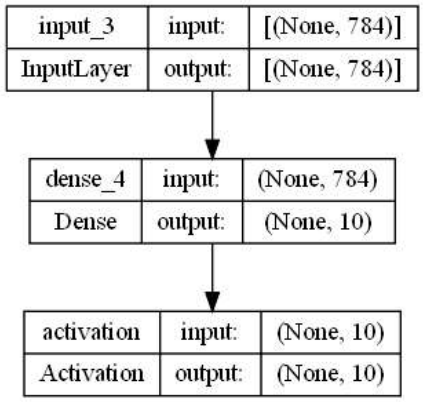
In [23]:
```
def linear_model_func(n_in, n_out):
    # Coding Time
    input = Input(shape=(n_in,))
    h = Dense(n_out)(input)
    y = Activation('softmax')(h)
    model = Model(inputs = input, outputs = y)
    return model
```

In [24]:
```
model = linear_model_func(n_in, n_out)
model.summary()
```

```
Model: "model_1"
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 input_3 (InputLayer)        [(None, 784)]             0

 dense_4 (Dense)             (None, 10)                7850

 activation (Activation)     (None, 10)                0

=================================================================
Total params: 7,850
Trainable params: 7,850
Non-trainable params: 0
_________________________________________________________________
```

In [25]:
```
plot_model(model, show_shapes=True)
```

Out[25]:



(3) 모델의 학습과정 설정

accuracy를 측정할 수 있는 문제라면 metric에 accuracy를 설정 (https://keras.io/models/model/)

In [26]:
```
# Coding Time
model.compile(loss='categorical_crossentropy',
              optimizer='sgd',
              metrics=['accuracy'])
```

(4) 모델 학습시키기

In [27]:
```
# Coding Time
history = model.fit(X_train_flat, Y_train_onehot, batch_size=128, epochs=20, validation_split=0.2)
```

```
Epoch 1/20
375/375 [==============================] - 1s 2ms/step - loss: 1.4125 - accuracy: 0.6535 - val_loss: 0.8990 - val_accuracy: 0.8257
Epoch 2/20
375/375 [==============================] - 1s 2ms/step - loss: 0.7964 - accuracy: 0.8288 - val_loss: 0.6577 - val_accuracy: 0.8576
Epoch 3/20
375/375 [==============================] - 1s 2ms/step - loss: 0.6444 - accuracy: 0.8510 - val_loss: 0.5612 - val_accuracy: 0.8708
Epoch 4/20
375/375 [==============================] - 1s 2ms/step - loss: 0.5714 - accuracy: 0.8624 - val_loss: 0.5079 - val_accuracy: 0.8770
Epoch 5/20
375/375 [==============================] - 1s 2ms/step - loss: 0.5271 - accuracy: 0.8693 - val_loss: 0.4738 - val_accuracy: 0.8826
Epoch 6/20
375/375 [==============================] - 1s 2ms/step - loss: 0.4966 - accuracy: 0.8747 - val_loss: 0.4495 - val_accuracy: 0.8865
Epoch 7/20
375/375 [==============================] - 1s 2ms/step - loss: 0.4742 - accuracy: 0.8790 - val_loss: 0.4312 - val_accuracy: 0.8907
Epoch 8/20
375/375 [==============================] - 1s 2ms/step - loss: 0.4568 - accuracy: 0.8824 - val_loss: 0.4171 - val_accuracy: 0.8930
Epoch 9/20
375/375 [==============================] - 1s 2ms/step - loss: 0.4428 - accuracy: 0.8847 - val_loss: 0.4056 - val_accuracy: 0.8954
Epoch 10/20
375/375 [==============================] - 1s 2ms/step - loss: 0.4311 - accuracy: 0.8869 - val_loss: 0.3961 - val_accuracy: 0.8973
Epoch 11/20
375/375 [==============================] - 1s 2ms/step - loss: 0.4214 - accuracy: 0.8890 - val_loss: 0.3880 - val_accuracy: 0.8991
Epoch 12/20
375/375 [==============================] - 1s 2ms/step - loss: 0.4129 - accuracy: 0.8911 - val_loss: 0.3810 - val_accuracy: 0.9007
Epoch 13/20
375/375 [==============================] - 1s 2ms/step - loss: 0.4056 - accuracy: 0.8923 - val_loss: 0.3750 - val_accuracy: 0.9012
Epoch 14/20
375/375 [==============================] - 1s 2ms/step - loss: 0.3991 - accuracy: 0.8936 - val_loss: 0.3697 - val_accuracy: 0.9024
Epoch 15/20
375/375 [==============================] - 1s 2ms/step - loss: 0.3933 - accuracy: 0.8950 - val_loss: 0.3648 - val_accuracy: 0.9035
Epoch 16/20
375/375 [==============================] - 1s 2ms/step - loss: 0.3881 - accuracy: 0.8954 - val_loss: 0.3607 - val_accuracy: 0.9038
Epoch 17/20
375/375 [==============================] - 1s 2ms/step - loss: 0.3834 - accuracy: 0.8963 - val_loss: 0.3568 - val_accuracy: 0.9058
Epoch 18/20
375/375 [==============================] - 1s 2ms/step - loss: 0.3792 - accuracy: 0.8972 - val_loss: 0.3534 - val_accuracy: 0.9054
Epoch 19/20
375/375 [==============================] - 1s 2ms/step - loss: 0.3753 - accuracy: 0.8983 - val_loss: 0.3501 - val_accuracy: 0.9064
Epoch 20/20
375/375 [==============================] - 1s 2ms/step - loss: 0.3716 - accuracy: 0.8992 - val_loss: 0.3471 - val_accuracy: 0.9068
```

(5) 모델 평가하기

In [28]:
```
# Coding Time
loss_and_accuracy = model.evaluate(X_test_flat, Y_test_onehot, batch_size=128)
print('loss : %.4f, accruracy : %.4f'%(loss_and_accuracy[0],loss_and_accuracy[1]))
```

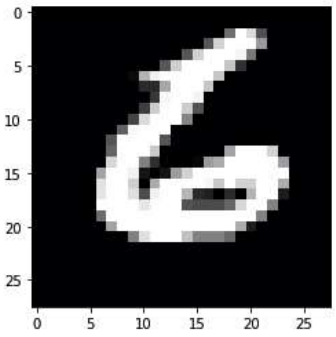
```
79/79 [==============================] - 0s 2ms/step - loss: 0.3476 - accuracy: 0.9068
loss : 0.3476, accruracy : 0.9068
```

(6) 모델 사용하기

In [29]:
```
# Coding Time
test_data = X_test[-1].reshape(1,28*28)
pred_y = model.predict(test_data)
pred_y = pred_y.argmax()

print('real_label : {}, predict_label : {}'.format(Y_test[-1], pred_y))
plt.imshow(X_test[-1], cmap='gray')
```

```
1/1 [==============================] - 0s 41ms/step
real_label : 6, predict_label : 6
<matplotlib.image.AxesImage at 0x20cf016b7c0>
```

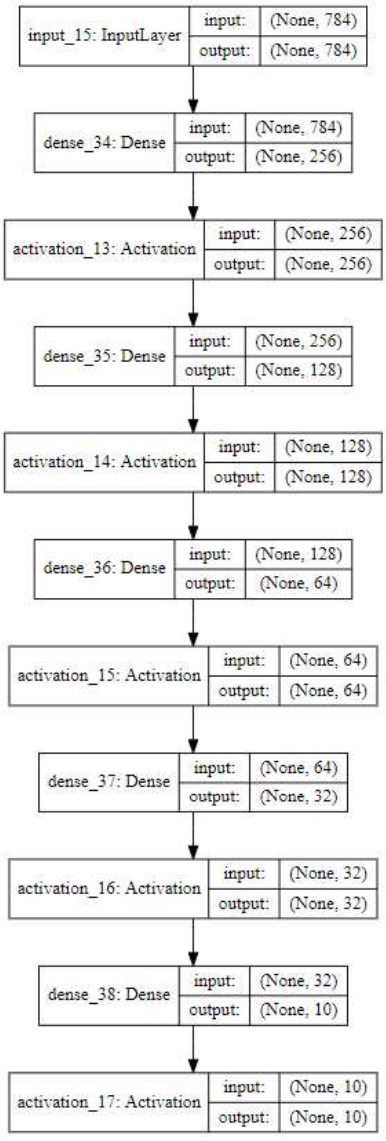
Out[29]:



1.3 MNIST Classification with DNN (To Do)

모델의 설명을 보고 DNN 코드를 만들어보자

(2) Keras 모델링



<맨 마지막 activation은 softmax로 하고 이외의 activation은 relu를 사용>

In [30]:

```
def DNN_seq(n_in, n_out):
    model = Sequential()
    model.add(Dense(units = 256, input_shape=(n_in,), activation='relu'))
    model.add(Dense(units = 128, input_shape=(256,), activation='relu'))
    model.add(Dense(units = 64, input_shape=(128,), activation='relu'))
    model.add(Dense(units = 32, input_shape=(64,), activation='relu'))
    model.add(Dense(units = n_out, input_shape=(32,), activation='softmax'))
    return model

def DNN_func(n_in, n_out):
    input = Input(shape=(n_in,))
    x = Dense(256)(input)
    x = Activation('relu')(x)
    x = Dense(128)(x)
    x = Activation('relu')(x)
```

```
    x = Dense(64)(x)
    x = Activation('relu')(x)
    x = Dense(32)(x)
    x = Activation('relu')(x)
    x = Dense(n_out)(x)
    y = Activation('softmax')(x)
    model = Model(inputs = input, outputs = y)
    return model

model = DNN_seq(n_in, n_out)
model.summary()
```

```
Model: "sequential_1"
_________________________________________________________________
 Layer (type)                 Output Shape              Param #
=================================================================
 dense_5 (Dense)              (None, 256)               200960

 dense_6 (Dense)              (None, 128)               32896

 dense_7 (Dense)              (None, 64)                8256

 dense_8 (Dense)              (None, 32)                2080

 dense_9 (Dense)              (None, 10)                330

=================================================================
Total params: 244,522
Trainable params: 244,522
Non-trainable params: 0
_________________________________________________________________
```

(3) 모델의 학습과정 설정

<optimizer를 adam으로 설정>

In [31]:
```
model.compile(loss='categorical_crossentropy',
              optimizer='adam',
              metrics=['accuracy'])
```

(4) 모델 학습시키기

<batch size를 256, epoch을 30, train 데이터 중 30%를 validation 데이터로 사용, verbose 모드는 2>

In [32]:
```
history = model.fit(X_train_flat, Y_train_onehot, batch_size=256, epochs=30, validation_split=0.3, verbose =2)
```

```
Epoch 1/30
165/165 - 1s - loss: 0.4872 - accuracy: 0.8536 - val_loss: 0.1910 - val_accuracy: 0.9457 - 1s/epoch - 7ms/step
Epoch 2/30
165/165 - 1s - loss: 0.1535 - accuracy: 0.9542 - val_loss: 0.1419 - val_accuracy: 0.9582 - 550ms/epoch - 3ms/step
Epoch 3/30
165/165 - 1s - loss: 0.1024 - accuracy: 0.9687 - val_loss: 0.1204 - val_accuracy: 0.9644 - 555ms/epoch - 3ms/step
Epoch 4/30
165/165 - 1s - loss: 0.0762 - accuracy: 0.9768 - val_loss: 0.1215 - val_accuracy: 0.9642 - 559ms/epoch - 3ms/step
Epoch 5/30
165/165 - 1s - loss: 0.0551 - accuracy: 0.9833 - val_loss: 0.1028 - val_accuracy: 0.9720 - 548ms/epoch - 3ms/step
Epoch 6/30
165/165 - 1s - loss: 0.0434 - accuracy: 0.9869 - val_loss: 0.1142 - val_accuracy: 0.9673 - 550ms/epoch - 3ms/step
Epoch 7/30
165/165 - 1s - loss: 0.0356 - accuracy: 0.9884 - val_loss: 0.1136 - val_accuracy: 0.9686 - 578ms/epoch - 4ms/step
Epoch 8/30
165/165 - 1s - loss: 0.0256 - accuracy: 0.9923 - val_loss: 0.1044 - val_accuracy: 0.9733 - 554ms/epoch - 3ms/step
Epoch 9/30
165/165 - 1s - loss: 0.0213 - accuracy: 0.9936 - val_loss: 0.1046 - val_accuracy: 0.9738 - 541ms/epoch - 3ms/step
Epoch 10/30
165/165 - 1s - loss: 0.0150 - accuracy: 0.9957 - val_loss: 0.1090 - val_accuracy: 0.9733 - 559ms/epoch - 3ms/step
Epoch 11/30
165/165 - 1s - loss: 0.0129 - accuracy: 0.9960 - val_loss: 0.1190 - val_accuracy: 0.9724 - 592ms/epoch - 4ms/step
Epoch 12/30
165/165 - 1s - loss: 0.0160 - accuracy: 0.9949 - val_loss: 0.1289 - val_accuracy: 0.9704 - 548ms/epoch - 3ms/step
Epoch 13/30
165/165 - 1s - loss: 0.0159 - accuracy: 0.9948 - val_loss: 0.1092 - val_accuracy: 0.9752 - 552ms/epoch - 3ms/step
Epoch 14/30
165/165 - 1s - loss: 0.0113 - accuracy: 0.9964 - val_loss: 0.1093 - val_accuracy: 0.9749 - 568ms/epoch - 3ms/step
Epoch 15/30
165/165 - 1s - loss: 0.0074 - accuracy: 0.9978 - val_loss: 0.1262 - val_accuracy: 0.9749 - 591ms/epoch - 4ms/step
Epoch 16/30
165/165 - 1s - loss: 0.0066 - accuracy: 0.9980 - val_loss: 0.1398 - val_accuracy: 0.9733 - 555ms/epoch - 3ms/step
Epoch 17/30
165/165 - 1s - loss: 0.0106 - accuracy: 0.9966 - val_loss: 0.1431 - val_accuracy: 0.9723 - 547ms/epoch - 3ms/step
Epoch 18/30
165/165 - 1s - loss: 0.0115 - accuracy: 0.9963 - val_loss: 0.1306 - val_accuracy: 0.9735 - 560ms/epoch - 3ms/step
Epoch 19/30
165/165 - 1s - loss: 0.0107 - accuracy: 0.9969 - val_loss: 0.1241 - val_accuracy: 0.9757 - 587ms/epoch - 4ms/step
Epoch 20/30
165/165 - 1s - loss: 0.0085 - accuracy: 0.9974 - val_loss: 0.1259 - val_accuracy: 0.9742 - 541ms/epoch - 3ms/step
Epoch 21/30
165/165 - 1s - loss: 0.0081 - accuracy: 0.9973 - val_loss: 0.1271 - val_accuracy: 0.9746 - 523ms/epoch - 3ms/step
Epoch 22/30
165/165 - 1s - loss: 0.0118 - accuracy: 0.9964 - val_loss: 0.1340 - val_accuracy: 0.9754 - 534ms/epoch - 3ms/step
Epoch 23/30
165/165 - 0s - loss: 0.0023 - accuracy: 0.9994 - val_loss: 0.1297 - val_accuracy: 0.9767 - 489ms/epoch - 3ms/step
Epoch 24/30
165/165 - 1s - loss: 0.0016 - accuracy: 0.9994 - val_loss: 0.1285 - val_accuracy: 0.9773 - 520ms/epoch - 3ms/step
Epoch 25/30
165/165 - 1s - loss: 0.0016 - accuracy: 0.9996 - val_loss: 0.1525 - val_accuracy: 0.9738 - 514ms/epoch - 3ms/step
Epoch 26/30
165/165 - 1s - loss: 4.7470e-04 - accuracy: 1.0000 - val_loss: 0.1359 - val_accuracy: 0.9771 - 550ms/epoch - 3ms/step
Epoch 27/30
165/165 - 1s - loss: 1.3349e-04 - accuracy: 1.0000 - val_loss: 0.1311 - val_accuracy: 0.9788 - 541ms/epoch - 3ms/step
Epoch 28/30
165/165 - 1s - loss: 7.5073e-05 - accuracy: 1.0000 - val_loss: 0.1326 - val_accuracy: 0.9789 - 514ms/epoch - 3ms/step
Epoch 29/30
165/165 - 1s - loss: 5.6021e-05 - accuracy: 1.0000 - val_loss: 0.1337 - val_accuracy: 0.9789 - 501ms/epoch - 3ms/step
Epoch 30/30
165/165 - 1s - loss: 4.7533e-05 - accuracy: 1.0000 - val_loss: 0.1352 - val_accuracy: 0.9792 - 512ms/epoch - 3ms/step
```

(5) 모델 평가하기

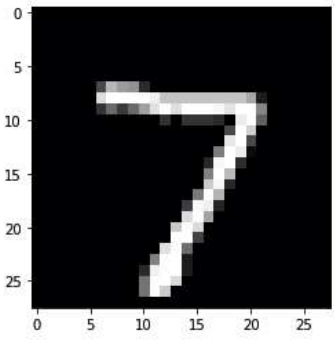
In [33]:
```
loss_and_accuracy = model.evaluate(X_test_flat, Y_test_onehot, batch_size=128)
print('loss : %.4f, accruracy : %.4f'%(loss_and_accuracy[0],loss_and_accuracy[1]))
```

```
79/79 [==============================] - 0s 2ms/step - loss: 0.1055 - accuracy: 0.9820
loss : 0.1055, accruracy : 0.9820
```

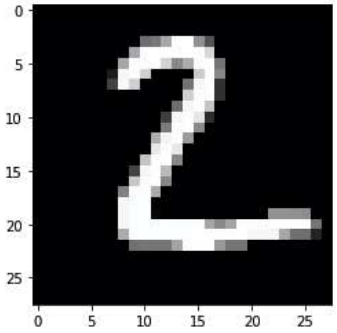
(6) 모델 사용하기

In [34]:
```
for i in range(5):
    test_data = X_test[i].reshape(1,28*28)
    pred_y = model.predict(test_data, verbose=0)
    pred_y = pred_y.argmax()

    print('real_label : {}, predict_label : {}'.format(Y_test[i], pred_y))
    plt.imshow(X_test[i], cmap='gray')
    plt.show()
```

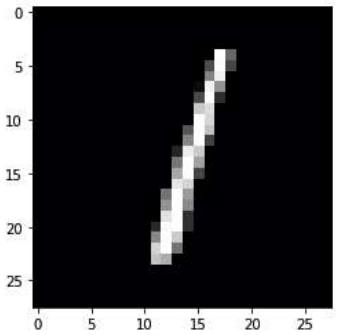
```
real_label : 7, predict_label : 7
```



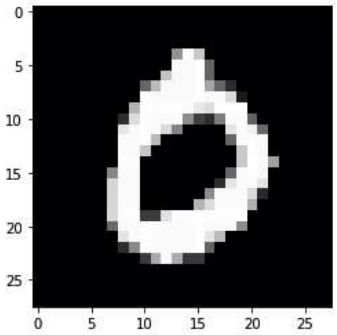

```
real_label : 2, predict_label : 2
```



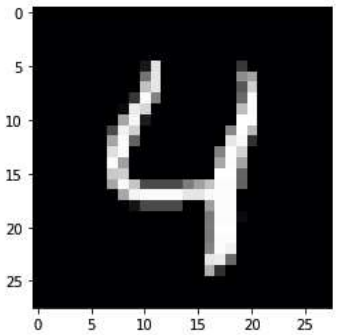
real_label : 1, predict_label : 1



real_label : 0, predict_label : 0



real_label : 4, predict_label : 4



(7) 모델 저장하고 불러오기

저장하기 : model 객체의 내부 함수인 save() .h5 형식으로 저장할 수 있음 [1.모델의 구조, 2.학습된 파라미터, 3.complie() 설정]

In [35]:
```
# Coding Time
model.save('latest_model.h5')
```

불러오기 : load_model 함수로 .h5 파일에서 모델을 불러올 수 있음

In [36]:
```
# Coding Time
from tensorflow.keras.models import load_model
model = load_model('latest_model.h5')
model.fit(X_train_flat, Y_train_onehot, batch_size=256, epochs=3, validation_split=0.3, verbose=2)
```

```
Epoch 1/3
165/165 - 1s - loss: 4.0850e-05 - accuracy: 1.0000 - val_loss: 0.1364 - val_accuracy: 0.9793 - 865ms/epoch - 5ms/step
Epoch 2/3
165/165 - 1s - loss: 3.6166e-05 - accuracy: 1.0000 - val_loss: 0.1377 - val_accuracy: 0.9789 - 562ms/epoch - 3ms/step
Epoch 3/3
165/165 - 1s - loss: 3.1927e-05 - accuracy: 1.0000 - val_loss: 0.1389 - val_accuracy: 0.9789 - 543ms/epoch - 3ms/step
```

Out[36]:
```
<keras.callbacks.History at 0x20cf12e95b0>
```

==========

1.4 Keras MNIST - 모델의 성능을 직접 높혀보자

- DNN의 구조를 바꾸어 나만의 모델을 만들어보자
- 목표 정확도: 평가 셋에 대해 98.5% 만들기
- 바꿀 수 있는 하이퍼 파라미터: Learning Rate, Batch size, Epochs, Optimizer, Activation Function, 모델 레이어 구조 등

(2) Keras 모델링

In [ ]:

(3) 모델의 학습과정 설정

In [ ]:

(4) 모델 학습시키기

In [ ]:

(5) 모델 평가하기

In [ ]: