

# MNIST 손글씨 예측과 오류 확인

# 실습 파일

- 21-6-mnist-application.ipynb

# MNIST 딥러닝 구현 전 소스, 약 97% 정확도

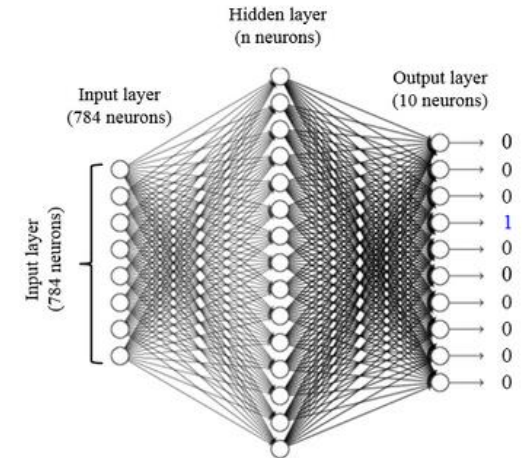
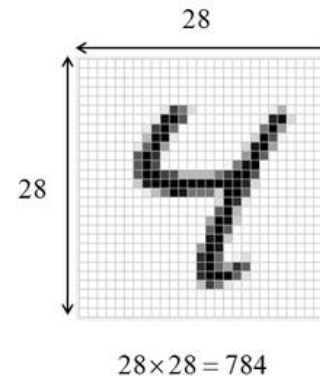
```
#####
import tensorflow as tf

# mnist 모듈 준비
mnist = tf.keras.datasets.mnist

# MNIST 데이터셋을 훈련과 테스트 데이터로 로드하여 준비
(x_train, y_train), (x_test, y_test) = mnist.load_data()

# 샘플 값을 정수 (0~255)에서 부동소수 (0~1)로 변환
x_train, x_test = x_train / 255.0, x_test / 255.0
```

```
# 층을 차례대로 쌓아 tf.keras.Sequential 모델을 생성
model = tf.keras.models.Sequential([
    tf.keras.layers.Flatten(input_shape=(28, 28)),
    tf.keras.layers.Dense(128, activation='relu'),
    tf.keras.layers.Dropout(.2),
    tf.keras.layers.Dense(10, activation='softmax')
])
```



figures) <http://neuralnetworksanddeeplearning.com/chap1.html>

```
# 훈련에 사용할 옵티마이저(optimizer)와 손실 함수, 출력정보를 선택
model.compile(optimizer='adam',
              loss='sparse_categorical_crossentropy',
              metrics=['accuracy'])

# 모델 요약 표시
model.summary()

# 모델을 훈련 데이터로 총 5번 훈련
model.fit(x_train, y_train, epochs=5)

# 모델을 테스트 데이터로 평가
model.evaluate(x_test, y_test)
```

# 테스트 데이터의 첫 번째 손글씨 예측 결과를 확인

## • model.predict(input)

- input 값
  - 모델의 fit(), evaluate()에 입력과 같은 형태가 필요
    - 28X28 이미지가 여러 개인 3차원
- 첫 번째 손글씨만 알아보더라도 3차원 배열로 입력
  - 슬라이스해서 사용, x\_test[:1]
  - pred\_result = model.predict(x\_test[:1])

## • 결과

- 정수 ?
  - 손글씨 값의 정수
- 실제
  - (1, 10)의 이차원 배열
- 결과
  - 10개의 0~1의 실수

```
[33] 1 # 테스트 데이터의 첫 번째 손글씨 예측 결과를 확인
      2 print(x_test[:1].shape)
      3
      4 pred_result = model.predict(x_test[:1])
      5 print(pred_result.shape)
      6 print(pred_result)
      7 print(pred_result[0])
```

```
↳ (1, 28, 28)
   (1, 10)
   [[8.7629097e-12  4.7056760e-14  2.5735870e-12  1.3529770e-07  1.9923079e-21
     1.6554103e-12  2.3112234e-21  9.9999988e-01  2.5956004e-10  3.6446388e-10]]
   [[8.7629097e-12  4.7056760e-14  2.5735870e-12  1.3529770e-07  1.9923079e-21
     1.6554103e-12  2.3112234e-21  9.9999988e-01  2.5956004e-10  3.6446388e-10]]
```

이게 과연 무엇  
일까?

# 정답으로 나온 10개의 실수는 확률 값

- 0~1
  - 확률 값?
  - 10 개 합이 1
- One hot encoding
  - 하나의 자리만 1, 나머지는 모두 0
- argmax() 로
  - 가장 큰 수의 위치 첨자를 반환

```
import numpy as np
```

```
# 10 개의 수를 더하면?
```

```
one_pred = pred_result[0]
print(one_pred.sum())
```

```
# 혹시 가장 큰 수가 있는 첨자가 결과
```

```
one = np.argmax(one_pred)
print(one)
```

[8.7629097e-12  
4.7056760e-14  
2.5735870e-12  
1.3529770e-07  
1.9923079e-21  
1.6554103e-12  
2.3112234e-21  
9.9999988e-01  
2.5956004e-10  
3.6446388e-10]

0	0	첨자
0	1	
0	2	
0	3	
0	4	
0	5	
0	6	
1	7	
0	8	
0	9	

.99 정도로  
가장 큰 수

확정적이라면 이러한 원핫  
인코딩으로 직접 표현할 수  
있으나 딥러닝 결과는 왼쪽  
의 확률 값으로 표현

# Tensorflow 메소드

- **tf.reduce\_sum(), tf.argmax()**

```
import numpy as np

# 10 개의 수를 더하면?
one_pred = pred_result[0]
print(tf.reduce_sum(one_pred))
print(tf.reduce_sum(one_pred).numpy())

# 혹시 가장 큰 수가 있는 첨자가 결과
print(tf.argmax(one_pred).numpy())
```



```
import numpy as np

# 10 개의 수를 더하면?
one_pred = pred_result[0]
print(one_pred.sum())

# 혹시 가장 큰 수가 있는 첨자가 결과
one = np.argmax(one_pred)
print(one)
```



```
1.0
7
```

[8] `import numpy as np`

```
# 10 개의 수를 더하면?
one_pred = pred_result[0]
print(tf.reduce_sum(one_pred))
print(tf.reduce_sum(one_pred).numpy())











# 혹시 가장 큰 수가 있는 첨자가 결과
print(tf.argmax(one_pred).numpy())
```













```
tf.Tensor(1.0, shape=(), dtype=float32)
1.0
7
```

# One Hot Encoding의 이해

- 데이터가 취할 수 있는 모든 단일 범주에 대해 하나의 새 열을 생성
  - 독, 전기, 물, 지면, 벌레 및 유령이라는 6 개의 새로운 열이 필요
  - 모든 행 (포켓 몬스터)에서 범주에 속하는 경우 1을, 그렇지 않으면 0을 배치
    - 첫 번째 예 (Koffing)의 경우 독에 1을, 나머지 유형에 0을 저장

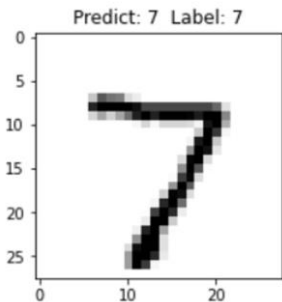
Sprite	Name	HP	Attack	Defense	Sp_Atk	Sp_Def	Speed	Type
	Koffing	40	65	95	60	45	35	Poison
	Pikachu	35	55	40	50	50	90	Electric
	Shellder	30	65	100	45	25	40	Water
	Krabby	30	105	90	25	25	50	Water
	Voltorb	40	30	50	55	55	100	Electric
	Cubone	50	50	95	40	50	35	Ground
	Magikarp	20	10	55	15	20	80	Water
	Pineco	50	65	90	35	35	15	Bug
	Misdreavus	60	60	60	85	85	85	Ghost
	Phanpy	90	60	60	40	40	40	Ground

Sprite	Name	HP	Attack	Defense	Sp_Atk	Sp_Def	Speed	Poison	Electric	Water	Ground	Bug	Ghost
	Koffing	40	65	95	60	45	35	1	0	0	0	0	0
	Pikachu	35	55	40	50	50	90	0	1	0	0	0	0
	Shellder	30	65	100	45	25	40	0	0	1	0	0	0
	Krabby	30	105	90	25	25	50	0	0	1	0	0	0
	Voltorb	40	30	50	55	55	100	0	1	0	0	0	0
	Cubone	50	50	95	40	50	35	0	0	0	1	0	0
	Magikarp	20	10	55	15	20	80	0	0	1	0	0	0
	Pineco	50	65	90	35	35	15	0	0	0	0	1	0
	Misdreavus	60	60	60	85	85	85	0	0	0	0	0	1
	Phanpy	90	60	60	40	40	40	0	0	0	1	0	0

# MNIST 예측 결과인 확률

- 각 위치(첨자의 값)의 값일 확률로 결과



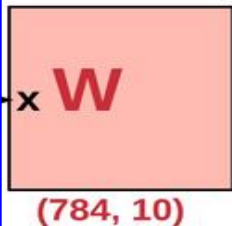
```
# 층을 차례대로 쌓아 tf.keras.models.Sequential 모델을 생성
model = tf.keras.models.Sequential([
    tf.keras.layers.Flatten(input_shape=(28, 28)),
    tf.keras.layers.Dense(128, activation='relu'),
    tf.keras.layers.Dropout(0.2),
    tf.keras.layers.Dense(10, activation='softmax')
])
```

[8.7629097e-12  
4.7056760e-14  
2.5735870e-12  
1.3529770e-07  
1.9923079e-21  
1.6554103e-12  
2.3112234e-21  
**9.9999988e-01**  
2.5956004e-10  
3.6446388e-10]

.99 정도  
로 가장  
큰 수

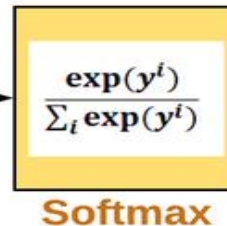
0  
1  
2  
3  
4  
5  
6  
**7**  
8  
9

Input of size (1, 784)



0.0  
1.5  
0.7  
0.4  
0.1  
0.1  
0.2  
3.5  
0.1  
0.1

Logits  
(1, 10)



0.02  
0.09  
0.04  
0.03  
0.02  
0.02  
0.03  
0.70  
0.02  
0.02

Probabilities  
( $P_n$ )

Cross-entropy

0  
0  
0  
0  
0  
0  
0  
1  
0  
0

One-hot-encoded  
label ( $L_n$ )



# 배열에서 가장 큰 값의 첨자 구하기

## • 메소드 np.argmax()

- 2차원에서 내부 행의 argmax를 구하려면

- axis=1

```
[46] 1 import numpy as np
      2
      3 #####
      4 # 원핫 인코딩과 argmax 학습
      5 print(np.argmax([5, 4, 10, 1, 2]))
      6 print(np.argmax([3, 1, 4, 9, 6, 7, 2]))
      7 print(np.argmax([[0.1, 0.8, 0.1], [0.7, 0.2, 0.1], [0.2, 0.1, 0.7]], axis=1))
```

```
↳ 2
   3
   [1 0 2]
```

## • 메소드 tf.argmax()

```
[11] import numpy as np

#####
# 원핫 인코딩과 argmax 학습
print(tf.argmax([5, 4, 10, 1, 2]))
print(tf.argmax([3, 1, 4, 9, 6, 7, 2]))
print(tf.argmax([[0.1, 0.8, 0.1], [0.7, 0.2, 0.1], [0.2, 0.1, 0.7]], axis=1))
```

```
↳ tf.Tensor(2, shape=(), dtype=int64)
   tf.Tensor(3, shape=(), dtype=int64)
   tf.Tensor([1 0 2], shape=(3,), dtype=int64)
```

# MNIST 손글씨 예측과 결과 확인

# 실제 손글씨를 그려 결과와 비교

## • 맞은 결과 7

### – 예측

- **predict()**
- 예측 결과는 원핫 인코딩 확률 값
  - 다시 **argmax()**로 변환

```
import numpy as np
```

```
# 10 개의 수를 더하면?
```

```
one_pred = pred_result[0]
print(one_pred.sum())
```

```
# 혹시 가장 큰 수가 있는 첨자가 결과
```

```
one = np.argmax(one_pred)
print(one)
```

```
import matplotlib.pyplot as plt
```

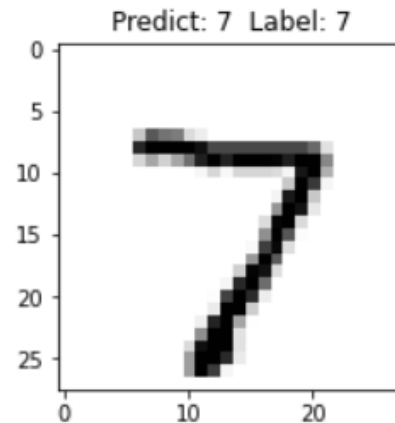
```
plt.figure(figsize=(5, 3))
```

```
tmp = "Predict: " + str(one) + " Label: " + str(y_test[0])
```

```
plt.title(tmp)
```

```
plt.imshow(x_test[0], cmap='Greys')
```

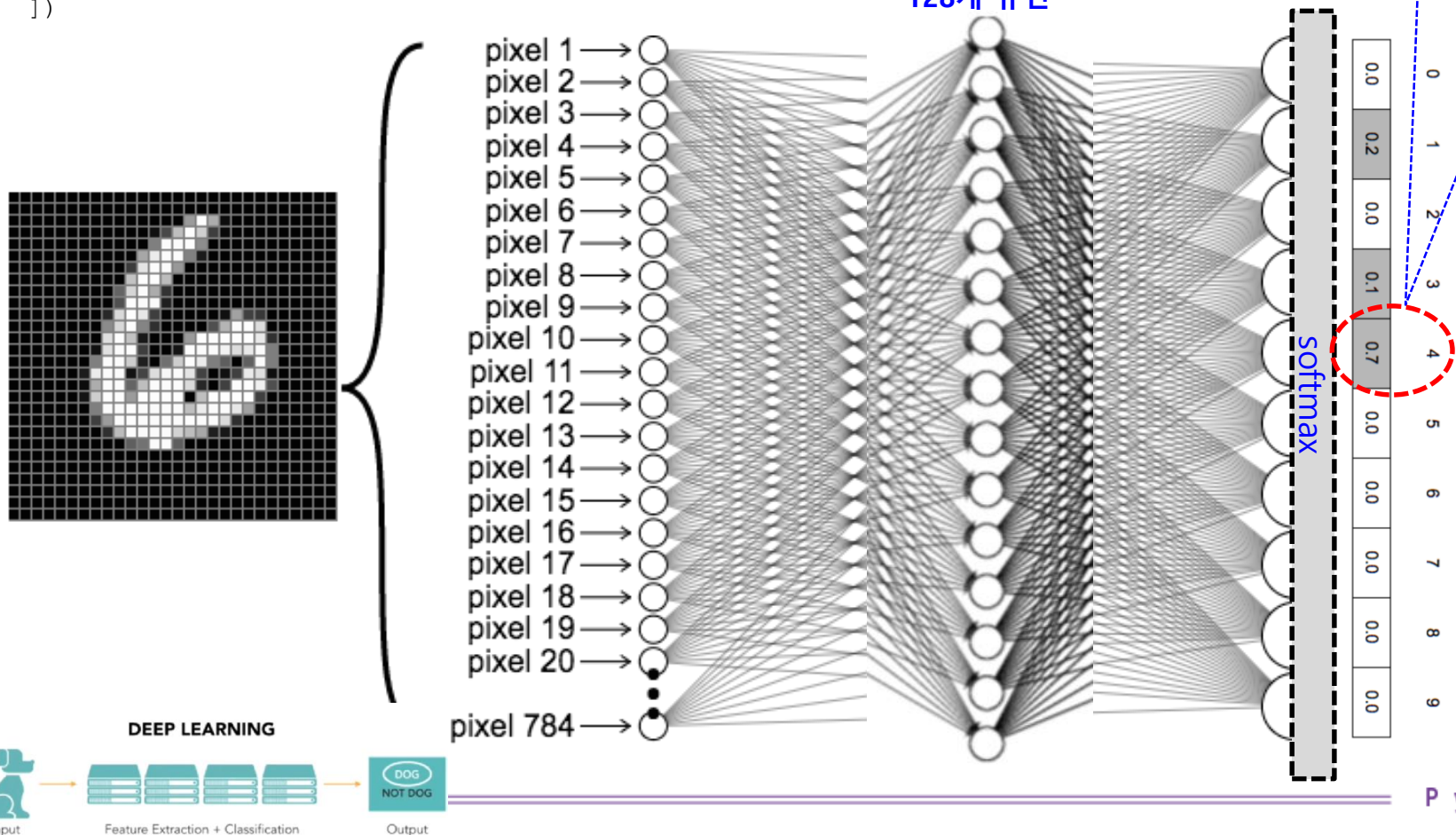
```
1.0
7
<matplotlib.image.AxesImage at 0x7f4a7890f2b0>
```



# 활성화 함수 softmax()

# 층을 차례대로 쌓아 `tf.keras.models.Sequential` 모델을 생성

```
model = tf.keras.models.Sequential([
    tf.keras.layers.Flatten(input_shape=(28, 28)),
    tf.keras.layers.Dense(128, activation='relu'),
    tf.keras.layers.Dropout(0.2),
    tf.keras.layers.Dense(10, activation='softmax')
])
```

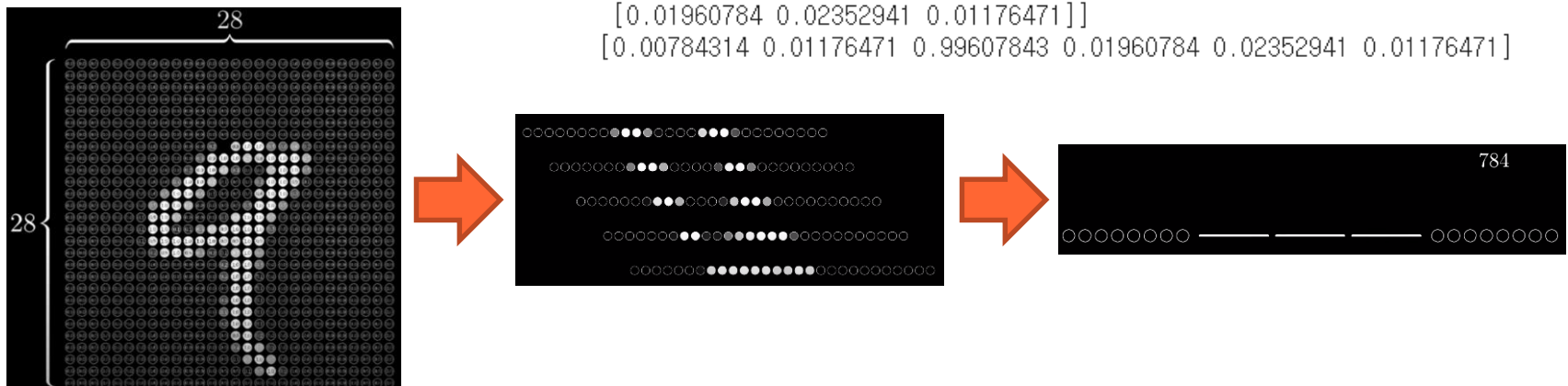


# 평탄화 메소드 flatten

## • 메소드 `ary.flatten()`

```
[50] 1 #####
      2 # 간단한 자료 처리
      3 import numpy as np
      4
      5 x = np.array([2, 3, 254, 5, 6, 3])
      6 x = x / 255.0
      7 print(x)
      8
      9 x = x.reshape(2, 3)
     10 print(x)
     11
     12 x = x.flatten()
     13 print(x)
```

```
↳ [0.00784314 0.01176471 0.99607843 0.01960784 0.02352941 0.01176471]
   [[0.00784314 0.01176471 0.99607843]
    [0.01960784 0.02352941 0.01176471]]
   [0.00784314 0.01176471 0.99607843 0.01960784 0.02352941 0.01176471]
```



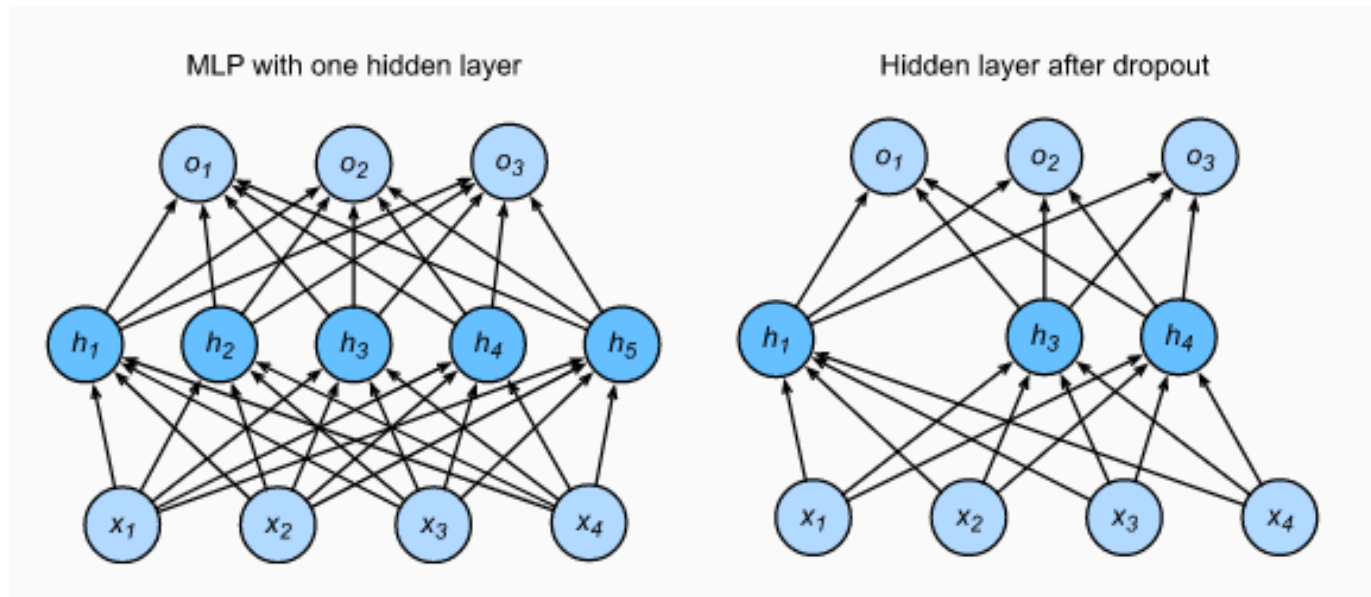
# 드롭아웃 개념

## • 2012년

- 토론토(Toronto) 대학의 힌튼(Hinton) 교수와 그의 제자들이 개발

## • 층에서 결과 값을 일정 비율로 제거하는 방법

- 오버피팅(overfitting) 문제를 해결하는 정규화(regularization) 목적을 위해서 필요
  - 학습 데이터에 지나치게 집중해 실제 Test에서는 결과가 더 나쁘게 나오는 현상
- $h_2$ 와  $h_5$ 를 0으로 지정하여 사용되지 않게 조정



# 드롭아웃 정리

- **tf.keras.layers.Dropout(0.2)**
  - 확률 값은 0.2~0.5를 주로 사용
- **Dropout**
  - 훈련 단계보다 더 많은 유닛이 활성화되기 때문에 균형을 맞추기 위해 층의 출력 값을 드롭아웃 비율만큼 줄이는 방법
  - 일반적으로 훈련단계에서 적용
    - 드롭아웃을 층에 적용하면 훈련하는 동안 층의 출력 특성을 랜덤하게 끄(off)
      - 즉 0으로 지정
  - 훈련하는 동안 어떤 입력 샘플에 대해 [0.2, 0.5, 1.3, 0.8, 1.1] 벡터를 출력하는 층이 있다고 가정
    - 벡터에서 몇 개의 원소가 랜덤하게 0이 됨
      - 예를 들면, [0, 0.5, 1.3, 0, 1.1]가 됨
      - .2라면 "드롭아웃 비율"은 0이 되는 특성의 비율인 20%
  - 테스트 단계에서는 어떤 유닛도 드롭아웃하지 않음
- **tf.keras에서는 Dropout 층을 이용해 네트워크에 드롭아웃을 추가**
  - 이 층은 바로 이전 층의 출력에 드롭아웃을 적용

# tf.keras.layers.Dropout()

## • 드롭아웃 계층

- 학습 시간 동안 각 단계에서 속도의 빈도로 입력 단위를 무작위로 0으로 설정
  - 과적합이 방지
- 0으로 설정되지 않은 입력
  - 전체 입력의 합계가 변경되지 않도록  $1/(1 - \text{rate})$  씩 증가

## • 지정한 비율로 0 지정

- 0이 아닌 값
  - $1/(1-.2)$ 배로 증가
  - $1 \times 1/(1-.2)$ 
    - 1.25

항상 비율에 맞는 0이 되지 않음  
총합도 마찬가지로

```
1 tf.random.set_seed(0)
2 layer = tf.keras.layers.Dropout(.2, input_shape=(2,))
3 data = np.arange(10).reshape(5, 2).astype(np.float32)
4 print(data)
5 print(data.sum())
6 outputs = layer(data, training=True)
7 print(outputs)
8 print(outputs.numpy().sum())
```

```
[[0. 1.]
 [2. 3.]
 [4. 5.]
 [6. 7.]
 [8. 9.]]
45.0
tf.Tensor(
[[ 0.    1.25]
 [ 2.5   3.75]
 [ 5.    6.25]
 [ 7.5   8.75]
 [10.    0.  ]], shape=(5, 2), dtype=float32)
45.0
```



# 테스트 데이터 모두 예측해 보기

```

from random import sample
import numpy as np

# x_test로 직접 결과 처리
pred_result = model.predict(x_test)
print(pred_result.shape)
print(pred_result[0])
print(np.argmax(pred_result[0]))

# 원핫 인코딩을 일반 데이터로 변환
pred_labels = np.argmax(pred_result, axis=1)
# 예측한 답 출력
print(pred_labels)
# 실제 정답 출력
print(y_test)
#####

```

```

(10000, 10)
[3.1434331e-07 2.5280498e-08 6.2301833e-06 9.8214645e-05 3.8718386e-11
6.4800524e-08 7.3486254e-14 9.9989331e-01 7.0924173e-08 1.7541108e-06]
7
[7 2 1 ... 4 5 6]
[7 2 1 ... 4 5 6]

```

MNIST 손글씨  
입의 20개 정답과 예측,  
그리고 그림 그리기

# 임의의 20개 예측 값과 정답

## 예측 값과 20개의 첨자 구하기

- 리스트 pred\_result
  - 모델의 예측 결과, 확률 값
- 리스트 pred\_labels
  - 모델의 예측 결과, 정수
- 리스트 samples
  - 출력할 20개의 첨자 리스트

```
#####
from random import sample
import numpy as np

# 예측한 softmax의 확률이 있는 리스트 pred_result
pred_result = model.predict(x_test)

# 실제 예측한 정답이 있는 리스트 pred_labels
pred_labels = np.argmax(pred_result, axis=1)

#랜덤하게 20개의 훈련용 자료를 예측 값과 정답, 그림을 그려 보자.
nrows, ncols = 5, 4 #출력 가로 세로 수
samples = sorted(sample(range(len(x_test)), nrows * ncols)) # 출력할 첨자 선정
```

# 임의의 20개 예측 값과 정답, 손글씨 그리기

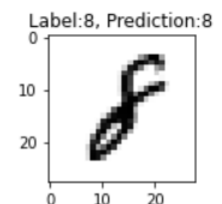
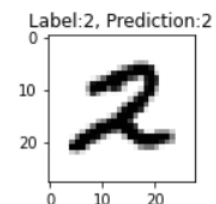
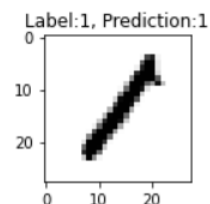
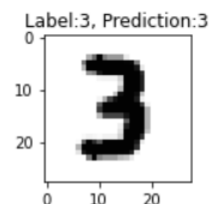
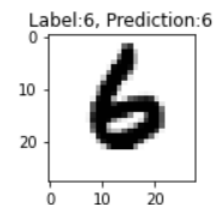
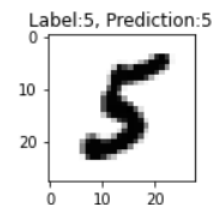
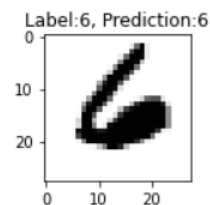
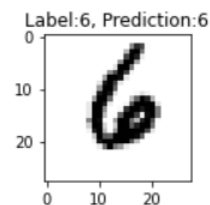
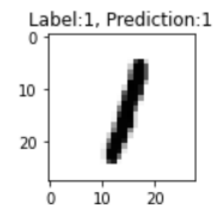
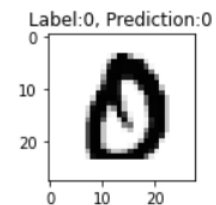
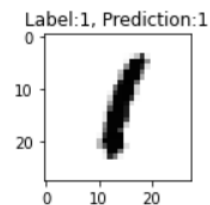
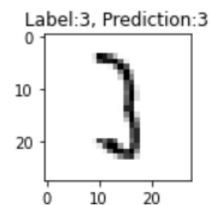
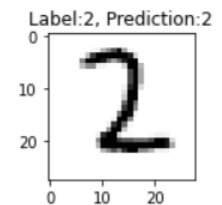
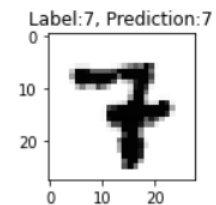
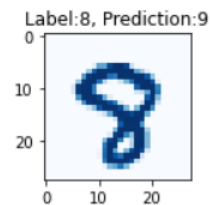
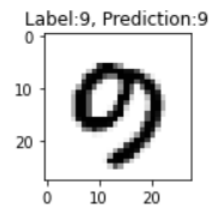
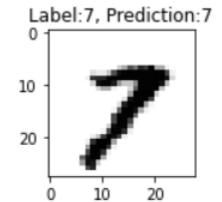
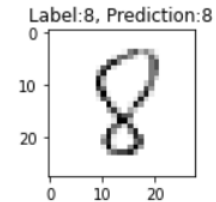
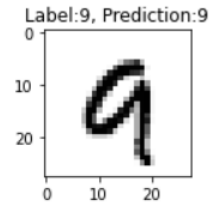
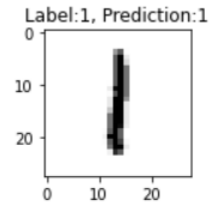
- `pred_labels[n] == y_test[n]`
  - 예측이 맞는 경우
  - 리스트 `pred_labels`
    - 모델의 예측 결과, 정수
  - 리스트 `y_test`
    - 훈련 데이터 정답
- 예측이 틀린 것은 'Blues'로 그리기

```
# 임의의 20개 그리기
count = 0
plt.figure(figsize=(12,10))
for n in samples:
    count += 1
    plt.subplot(nrows, ncols, count)
    # 예측이 틀린 것은 파란색으로 그리기
    cmap = 'Greys' if ( pred_labels[n] == y_test[n]) else 'Blues'
    plt.imshow(x_test[n].reshape(28, 28), cmap=cmap, interpolation='nearest')
    tmp = "Label:" + str(y_test[n]) + ", Prediction:" + str(pred_labels[n])
    plt.title(tmp)

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

# 임의의 20개 샘플 예측 값과 정답 그리기

- 8을 9로 예측



# 임의의 20개 샘플 예측 값과 정답 그리기 소스

```
#####
from random import sample
import numpy as np

# 예측한 softmax의 확률이 있는 리스트 pred_result
pred_result = model.predict(x_test)

# 실제 예측한 정답이 있는 리스트 pred_labels
pred_labels = np.argmax(pred_result, axis=1)

#랜덤하게 20개의 훈련용 자료를 예측 값과 정답, 그림을 그려 보자.
samples = sorted(sample(range(len(x_test)), nrows * ncols)) # 출력할 첨자 선정

# 임의의 20개 그리기
count = 0
nrows, ncols = 5, 4
plt.figure(figsize=(12,10))
for n in samples:
    count += 1
    plt.subplot(nrows, ncols, count)
    # 예측이 틀린 것은 파란색으로 그리기
    cmap = 'Greys' if (pred_labels[n] == y_test[n]) else 'Blues'
    plt.imshow(x_test[n].reshape(28, 28), cmap=cmap, interpolation='nearest')
    tmp = "Label:" + str(y_test[n]) + ", Prediction:" + str(pred_labels[n])
    plt.title(tmp)

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

MNIST 손글씨  
예측이 틀린 임의 20개  
정답과 예측,  
그리고 그림 그리기

# 예측이 틀린 20개 찾기

- 틀린 것을 임의의 20개를 찾아 첨자를 리스트 samples에 저장

```
from random import sample
import numpy as np

#####
# 예측 틀린 것 첨자를 저장할 리스트
mispred = []
# 예측한 softmax의 확률이 있는 리스트 pred_result
pred_result = model.predict(x_test)

# 실제 예측한 정답이 있는 리스트 pred_labels
pred_labels = np.argmax(pred_result, axis=1)

for n in range(0, len(y_test)):
    if pred_labels[n] != y_test[n]:
        mispred.append(n)
print("정답이 틀린 수", len(mispred))

# 랜덤하게 틀린 것 20개의 첨자 리스트 생성
samples = sample(mispred, 20)
print(samples)
```

예측이 틀린 조건

`pred_labels[n] != y_test[n]`

```
mispred = [n for n in range(0, len(y_test))
            if pred_labels[n] != y_test[n]]
```

정답이 틀린 수 195

[6625, 6093, 3946, 5676, 9587, 8311, 3520, 9679, 3558, 4571, 2953, 1112, 3503, 5642, 2369, 6400, 4551, 1247, 3943, 5734]



# 예측이 잘못된 20개 샘플로 그리기

- 틀린 첨자 저장
  - mispred
  - 196개 중 랜덤하게 20개 선택
- 5행 4열로 그리기

```
# 틀린 것 20개 그리기
count = 0
nrows, ncols = 5, 4
plt.figure(figsize=(12,10))
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(y_test[n]) + ", Prediction:" + str(pred_labels[n])
    plt.title(tmp)

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

정답이 틀린 수 195

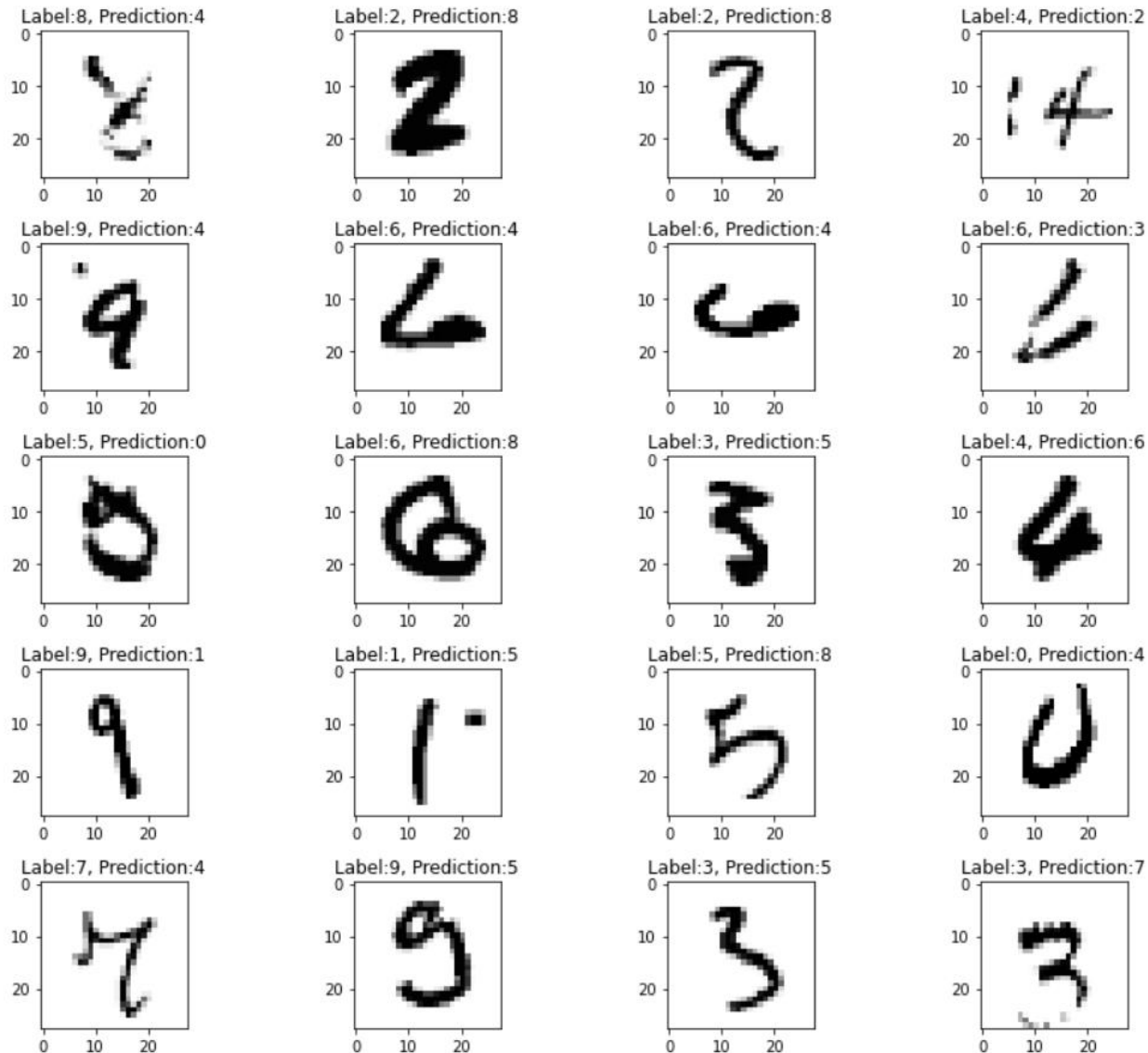
[6625, 6093, 3946, 5676, 9587, 8311, 3520, 9679, 3558, 4571, 2953, 1112, 3503, 5642, 2369, 6400, 4551, 1247, 3943, 5734]

# 예측이 잘못된 20개 샘플



정답이 틀린 수 195

[6625, 6093, 3946, 5676, 9587, 8311, 3520, 9679, 3558, 4571, 2953, 1112, 3503, 5642, 2369, 6400, 4551, 1247, 3943, 5734]



# 예측이 잘못된 20개 그리기 소스

```

from random import sample
import numpy as np

#####
# 예측 틀린 것 첨자를 저장할 리스트
mispred = []
# 예측한 softmax의 확률이 있는 리스트 pred_result
pred_result = model.predict(x_test)

# 실제 예측한 정답이 있는 리스트 pred_labels
pred_labels = np.argmax(pred_result, axis=1)

for n in range(0, len(y_test)):
    if pred_labels[n] != y_test[n]:
        mispred.append(n)
print('정답이 틀린 수', len(mispred))

# 랜덤하게 틀린 것 20개의 첨자 리스트 생성
samples = sample(mispred, 20)
print(samples)

# 틀린 것 20개 그리기
count = 0
nrows, ncols = 5, 4
plt.figure(figsize=(12,10))
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(y_test[n]) + ", Prediction:" + str(pred_labels[n])
    plt.title(tmp)

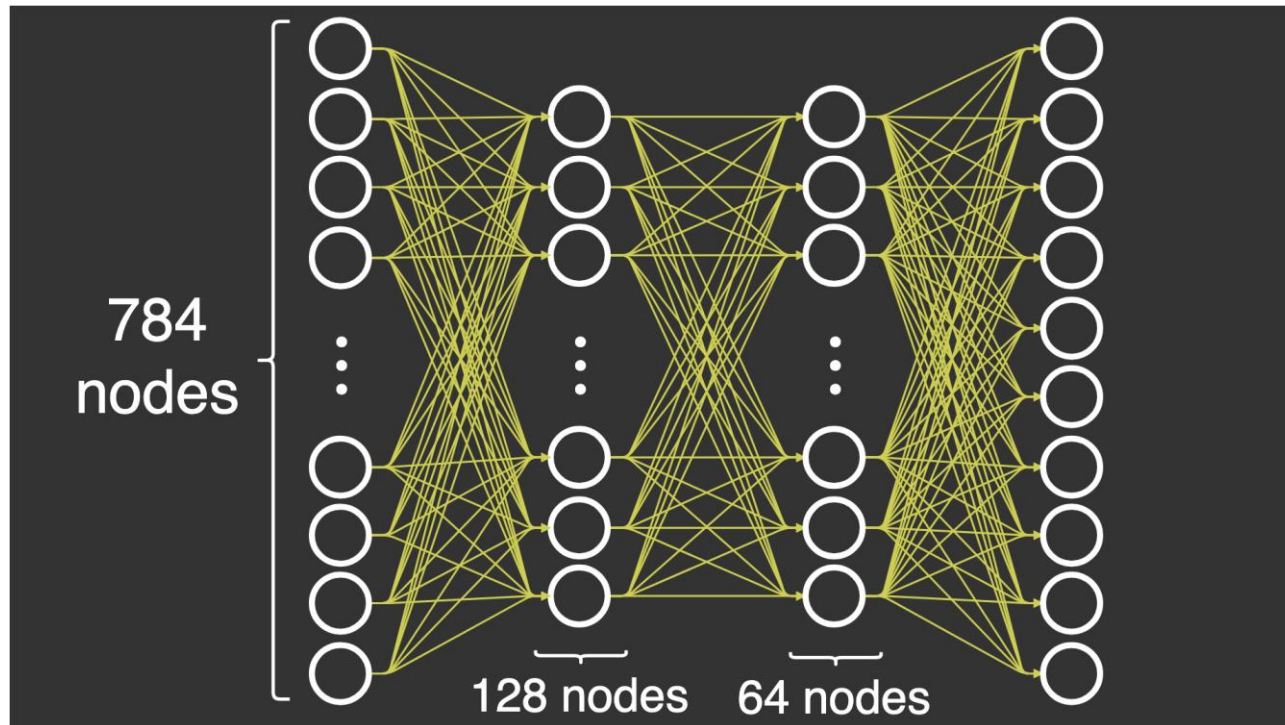
plt.tight_layout()
plt.show()
#####

```

# MNIST 손글씨 다양한 구현

# 중간층을 늘리고 훈련 횟수를 증가

- **중간층 2개, 출력층**
  - 128개 뉴런, 64개 뉴런, 10개 출력
- **훈련 횟수 20회**
  - epochs=20



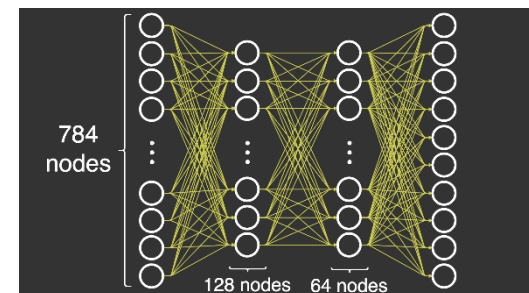
# 주요 소스

```
# 층을 차례대로 쌓아 tf.keras.Sequential 모델을 생성
model = tf.keras.models.Sequential([
    tf.keras.layers.Flatten(input_shape=(28, 28)),
    tf.keras.layers.Dense(128, activation='relu'),
    tf.keras.layers.Dropout(.2),
    tf.keras.layers.Dense(64, activation='relu'),
    tf.keras.layers.Dropout(.2),
    tf.keras.layers.Dense(10, activation='softmax')
])

# 훈련에 사용할 옵티마이저(optimizer)와 손실 함수, 출력정보를 선택
model.compile(optimizer='adam',
              loss='sparse_categorical_crossentropy',
              metrics=['accuracy'])

# 모델 요약 표시
model.summary()

# 모델을 훈련 데이터로 총 5번 훈련
model.fit(x_train, y_train, epochs=20)
```



# 전 소스

## • 약 98% 이상 정답 예측

```
#####
import tensorflow as tf

# mnist 모듈 준비
mnist = tf.keras.datasets.mnist
# MNIST 데이터셋을 훈련과 테스트 데이터로 로드하여 준비
(x_train, y_train), (x_test, y_test) = mnist.load_data()
# 샘플 값을 정수(0~255)에서 부동소수(0~1)로 변환
x_train, x_test = x_train / 255.0, x_test / 255.0

# 층을 차례대로 쌓아 tf.keras.Sequential 모델을 생성
model = tf.keras.models.Sequential([
    tf.keras.layers.Flatten(input_shape=(28, 28)),
    tf.keras.layers.Dense(128, activation='relu'),
    tf.keras.layers.Dropout(.2),
    tf.keras.layers.Dense(64, activation='relu'),
    tf.keras.layers.Dropout(.2),
    tf.keras.layers.Dense(10, activation='softmax')

])

# 훈련에 사용할 옵티마이저(optimizer)와 손실 함수, 출력정보를 선택
model.compile(optimizer='adam', loss='sparse_categorical_crossentropy',
              metrics=['accuracy'])

# 모델 요약 표시
model.summary()
# 모델을 훈련 데이터로 총 5번 훈련
model.fit(x_train, y_train, epochs=20)
# 모델을 테스트 데이터로 평가
model.evaluate(x_test, y_test)
```

# 메소드 `flatten()` 미사용

- 먼저 `reshape()`로 평탄화 작업을 수행한 후 `Dense()` 층 사용

```
import tensorflow as tf

mnist = tf.keras.datasets.mnist
(x_train, y_train), (x_test, y_test) = mnist.load_data()
# 샘플 값을 정수(0~255)에서 부동소수(0~1)로 변환
x_train, x_test = x_train / 255.0, x_test / 255.0
# 먼저 reshape()로 평탄화 작업을 수행한 후
x_train = x_train.reshape((60000, 28*28))
x_test = x_test.reshape((10000, 28*28))

# 층을 차례대로 쌓아 tf.keras.models.Sequential 모델을 생성
model = tf.keras.models.Sequential([
    #tf.keras.layers.Flatten(input_shape=(28, 28)),
    tf.keras.layers.Dense(128, activation='relu', input_shape=(28 * 28,)),
    tf.keras.layers.Dropout(0.2),
    tf.keras.layers.Dense(10, activation='softmax')
])
# 모델 요약 표시
model.summary()

# 훈련에 사용할 옵티마이저(optimizer)와 손실 함수, 출력정보를 모델에 설정
model.compile(optimizer='adam',
              loss='sparse_categorical_crossentropy',
              metrics=['accuracy'])
# metrics=['accuracy', 'mse'])

# 모델을 훈련 데이터로 총 5번 훈련
model.fit(x_train, y_train, epochs=5)
# 모델을 테스트 데이터로 평가
model.evaluate(x_test, y_test)
```