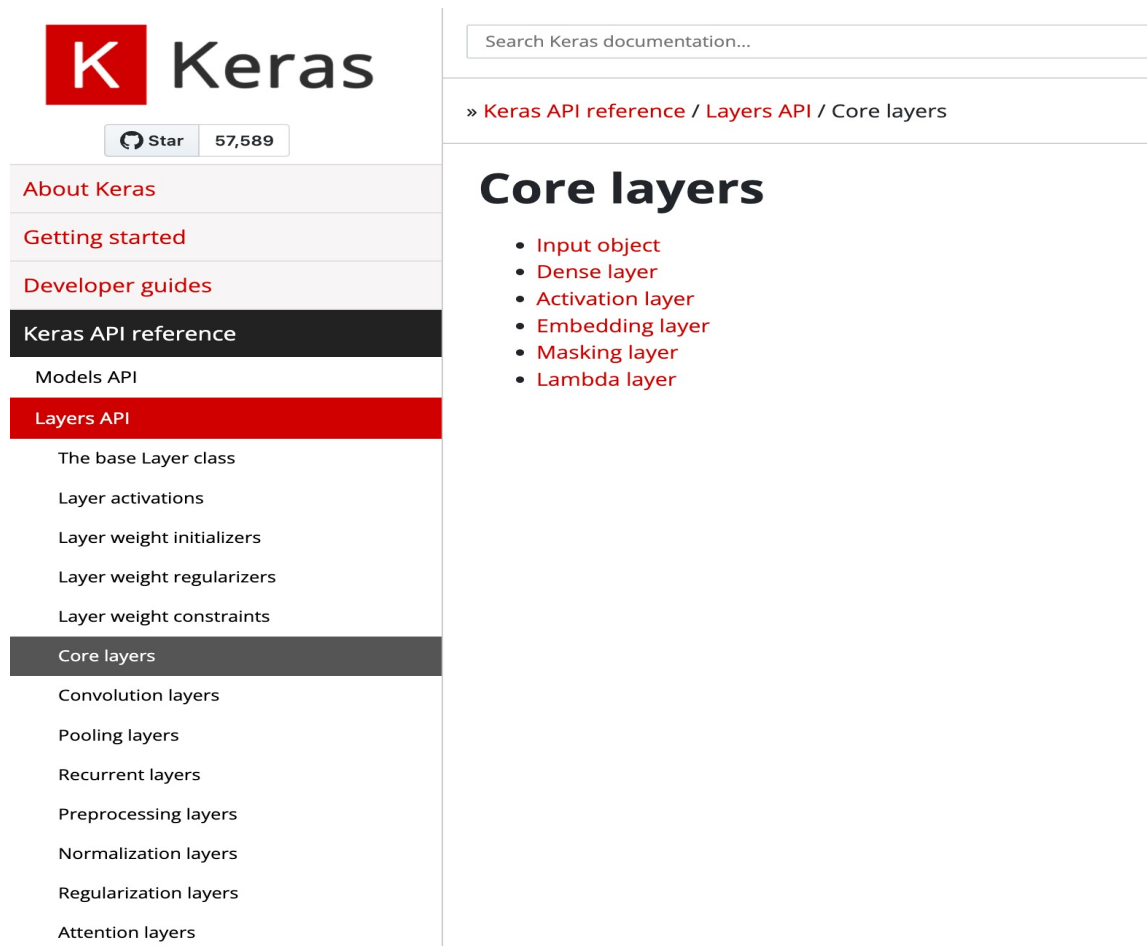


Keras를 이용한 심층신경망(Deep Neural Network) 구현(1/2)

□ 심층신경망을 구현을 위한 Keras



The screenshot displays the Keras documentation website. On the left, a sidebar contains a navigation menu with the following items: 'About Keras', 'Getting started', 'Developer guides', 'Keras API reference' (highlighted in black), 'Models API', 'Layers API' (highlighted in red), 'The base Layer class', 'Layer activations', 'Layer weight initializers', 'Layer weight regularizers', 'Layer weight constraints', 'Core layers' (highlighted in dark gray), 'Convolution layers', 'Pooling layers', 'Recurrent layers', 'Preprocessing layers', 'Normalization layers', 'Regularization layers', and 'Attention layers'. The main content area at the top features a search bar labeled 'Search Keras documentation...', a breadcrumb trail '» Keras API reference / Layers API / Core layers', and a section titled 'Core layers' which lists the following components: 'Input object', 'Dense layer', 'Activation layer', 'Embedding layer', 'Masking layer', and 'Lambda layer'.

K Keras

Star 57,589

About Keras

Getting started

Developer guides

Keras API reference

Models API

Layers API

The base Layer class

Layer activations

Layer weight initializers

Layer weight regularizers

Layer weight constraints

Core layers

Convolution layers

Pooling layers

Recurrent layers

Preprocessing layers

Normalization layers

Regularization layers

Attention layers

Search Keras documentation...

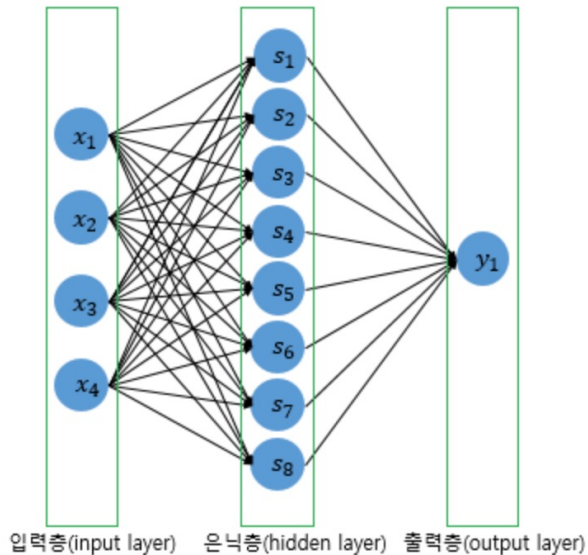
» Keras API reference / Layers API / Core layers

Core layers

- Input object
- Dense layer
- Activation layer
- Embedding layer
- Masking layer
- Lambda layer

Keras를 이용한 심층신경망(Deep Neural Network) 구현(2/2)

□ Keras를 이용한 DNN의 구성



```
model.summary()
```

Layer (type)	Output Shape	Param #
dense_1 (Dense)	(None, 8)	40
dense_2 (Dense)	(None, 1)	9
Total params: 49		
Trainable params: 49		
Non-trainable params: 0		

```
model = Sequential()  
model.add(Dense(8, input_dim=4, activation='relu'))  
model.add(Dense(1, activation='sigmoid')) # 출력층
```

□ Dense_1의 파라미터 갯수: bias(8개) + 가중치(32, 4×8)

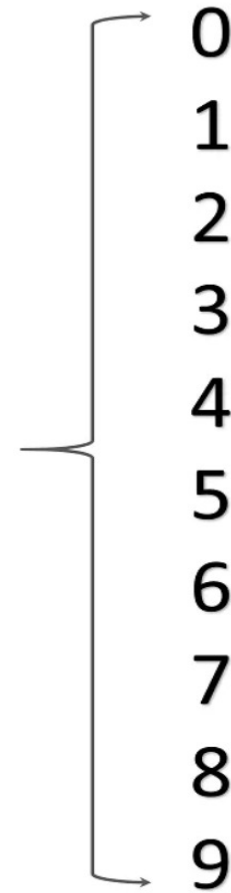
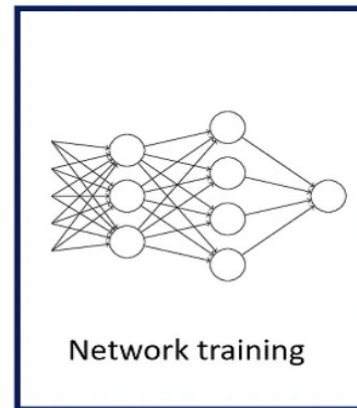
□ Dense_2의 파라미터 갯수: bias(1개) + 가중치(8)

분류용 심층신경망(Deep Neural Network) 구현 (1/6)

□ DNN을 이용한 MNIST 분류기



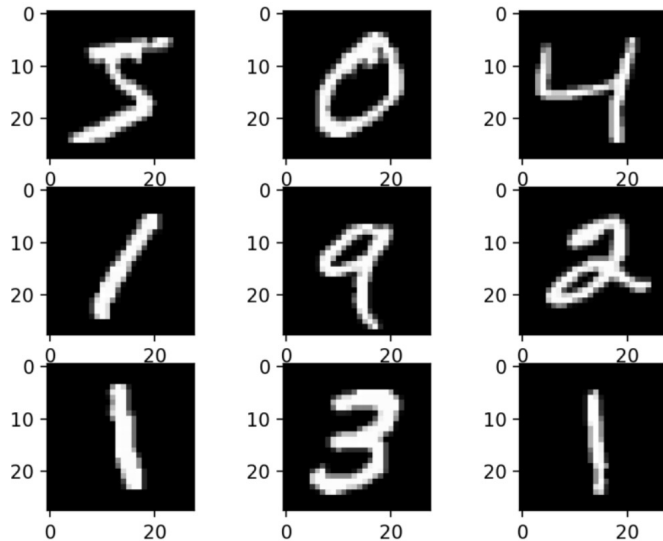
Data & Labels



분류용 심층신경망(Deep Neural Network) 구현 (2/6)

□ 영상 데이터베이스 (MNIST)

- 학습용 Gray 영상: 60,000장(밝기 0-255)
- 테스트용 Gray 영상: 10,000장 (밝기 0-255)



```
# 1. 학습 및 테스트데이터 준비
(X_train, y_train), (X_test, y_test) = mnist.load_data()

Y_train = np_utils.to_categorical(y_train)
Y_test = np_utils.to_categorical(y_test)
```

- 영상 레이블을 원 핫 인코딩(One-hot encoding)

레이블: '0'

[1 0 0 0 0 0 0 0 0]

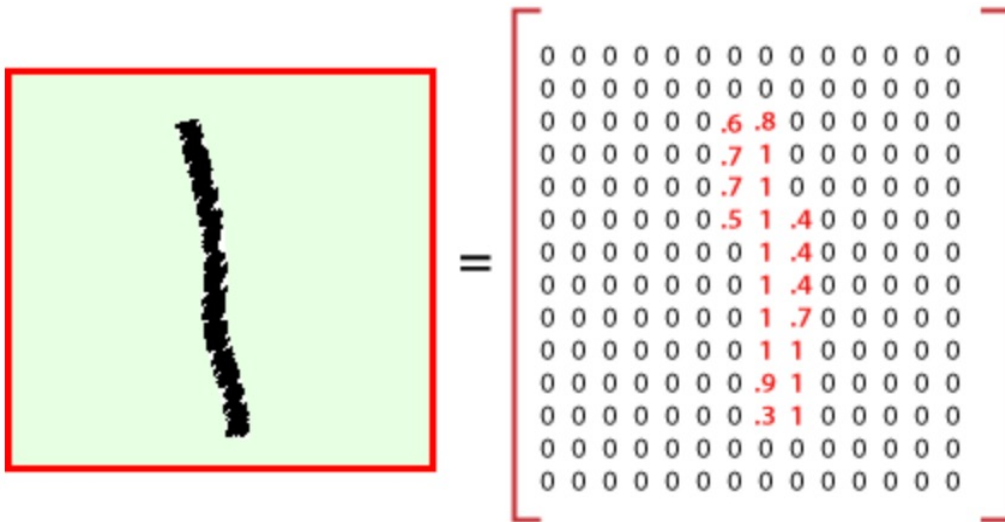
레이블: '1'

[0 1 0 0 0 0 0 0 0]

분류용 심층신경망(Deep Neural Network) 구현 (3/6)

영상 DB (MNIST)

- 영상크기: $28 * 28 = 784$



```
L, W, H = X_train.shape
X_train = X_train.reshape(-1, W * H)
X_test = X_test.reshape(-1, W * H)

X_train = X_train / 255.0 # 정규화
X_test = X_test / 255.0 # 정규화
```

- 영상들을 748 크기의 정규화된 벡터로 만듦

[illegible]

[첫째 줄]

[둘째 줄]

[셋째 줄]

분류용 심층신경망(Deep Neural Network) 구현 (4/6)

□ MNIST 분류용 DNN 구조(1/3)

```
# 2. 분류 DNN 분류기 모델링
Nin = X_train.shape[1] #784
Nh_l = [100, 50]
Nout = 10 # number of class

dnn_cls = Sequential()
dnn_cls.add(Dense(Nh_l[0], activation='relu', input_shape=(Nin,)))
dnn_cls.add(Dense(Nh_l[1], activation='relu'))
dnn_cls.add(Dense(Nout, activation='softmax'))
dnn_cls.compile(loss='categorical_crossentropy', optimizer='adam', metrics=['accuracy'])

#dnn_cls.summary()
```

- Dense_0의 파라미터 개수(78,500개): bias(100개) + 가중치(78,400, 784×100)
- Dense_1의 파라미터 개수(5,050개): bias(50개) + 가중치(5,000, 100×50)
- Dense_2의 파라미터 개수(61개): bias(10개) + 가중치(500, 50×10)

Layer (type)	Output Shape	Param #
dense (Dense)	(None, 100)	78500
dense_1 (Dense)	(None, 50)	5050
dense_2 (Dense)	(None, 10)	510
Total params: 84,060		
Trainable params: 84,060		
Non-trainable params: 0		

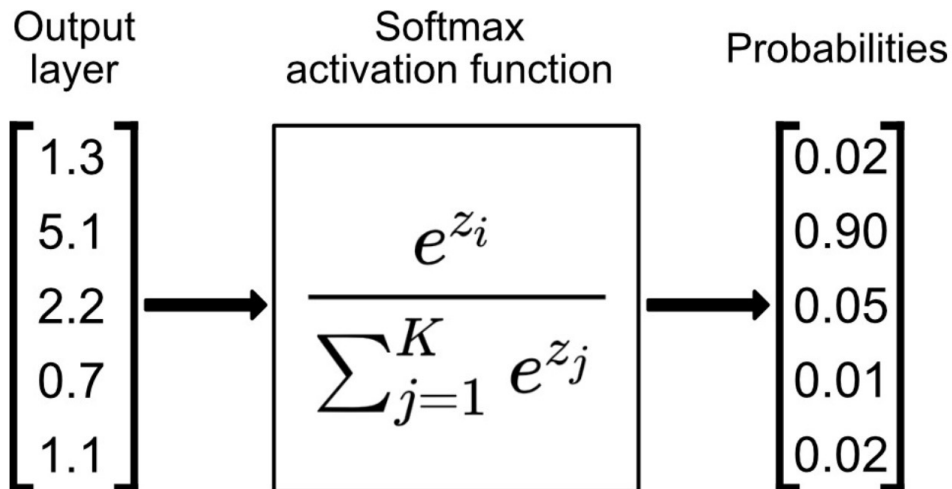
분류용 심층신경망(Deep Neural Network) 구현 (5/6)

□ MNIST 분류용 DNN 구조(2/3)

```
# 2. 분류 DNN 분류기 모델링
Nin = X_train.shape[1] #784
Nh_l = [100, 50]
Nout = 10 # number of class

dnn_cls = Sequential()
dnn_cls.add(Dense(Nh_l[0], activation='relu', input_shape=(Nin,)))
dnn_cls.add(Dense(Nh_l[1], activation='relu'))
dnn_cls.add(Dense(Nout, activation='softmax'))
dnn_cls.compile(loss='categorical_crossentropy', optimizer='adam', metrics=['accuracy'])

#dnn_cls.summary()
```



분류용 심층신경망(Deep Neural Network) 구현 (6/6)

□ MNIST 분류용 DNN 구조(3/3)

2. 분류 DNN 분류기 모델링

Nin = X_train.shape[1] #784

Nh_l = [100, 50]

Nout = 10 # number of class

dnn_cls = Sequential()

dnn_cls.add(Dense(Nh_l[0], activation='relu', input_shape=(Nin,)))

dnn_cls.add(Dense(Nh_l[1], activation='relu'))

dnn_cls.add(Dense(Nout, activation='softmax'))

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

#dnn_cls.summary()

4. categorical_crossentropy

categorical_crossentropy는 분류해야 할 클래스가 3개 이상인 경우, 즉 멀티클래스 분류에 사용됩니다. 라벨이 [0,0,1,0,0], [1,0,0,0,0], [0,0,0,1,0]과 같이 one-hot 형태로 제공될 때 사용됩니다. 공식은 다음과 같습니다.

$$L = -\frac{1}{N} \sum_{j=1}^N \sum_{i=1}^C t_{ij} \log(y_{ij}) \quad \dots(\text{categorical crossentropy})$$

여기서 C는 클래스의 갯수입니다.

이번에도 샘플이 하나만 있다고 가정하고, 실제값과 예측값이 완전히 일치하는 경우의 손실함수값을 살펴 보겠습니다. 0이 나와야 합니다. 실제값과 예측값이 모두 [1 0 0 0]이라고 가정하겠습니다.

$$L = -(1\log 1 + 0\log 0 + 0\log 0 + 0\log 0 + 0\log 0) = 0$$

계산했더니 0이 나왔습니다. 이번에는 실제값은 [1 0 0 0], 예측값은 [0 1 0 0]인 경우의 손실함수 값을 구해보겠습니다.

$$L = -(1\log 0 + 0\log 1 + 0\log 0 + 0\log 0 + 0\log 0) = \infty$$

계산했더니 양의 무한대가 나왔습니다. 일반적으로 예측값은 [0.02 0.94 0.02 0.01 0.01]와 같은 식으로 나오기 때문에 양의 무한대가 나오지는 않지만, 큰 값이 나오는 것만은 분명합니다. 이러한 특성을 가지고 있기 때문에 categorical_crossentropy는 멀티클래스 분류 문제의 손실함수로 사용되기에 적합합니다.

분류용 심층신경망(Deep Neural Network) 성능 (1/3)

□ MNIST 분류용 DNN 학습(1/3)

```
# 2. 분류 DNN 분류기 모델링
Nin = X_train.shape[1] #784
Nh_l = [100, 50]
Nout = 10 # number of class

dnn_cls = Sequential()
dnn_cls.add(Dense(Nh_l[0], activation='relu', input_shape=(Nin,)))
dnn_cls.add(Dense(Nh_l[1], activation='relu'))
dnn_cls.add(Dense(Nout, activation='softmax'))
dnn_cls.compile(loss='categorical_crossentropy', optimizer='adam', metrics=['accuracy'])

#dnn_cls.summary()
```

□ 가중치 학습방법('adam', 동작 설명은 아래 웹페이지 참조) 및 손실함수

❖ <https://onevision.tistory.com/entry/Optimizer-%EC%9D%98-%EC%A2%85%EB%A5%98%EC%99%80-%ED%8A%B9%EC%84%B1-Momentum-RMSProp-Adam>

분류용 심층신경망(Deep Neural Network) 성능 (2/3)

□ MNIST 분류용 DNN 학습(2/3)

2. 분류 DNN 분류기 모델링

```
Nin = X_train.shape[1] #784
```

```
Nh_l = [100, 50]
```

```
Nout = 10 # number of class
```

```
dnn_cls = Sequential()
```

```
dnn_cls.add(Dense(Nh_l[0], activation='relu', input_shape=(Nin,)))
```

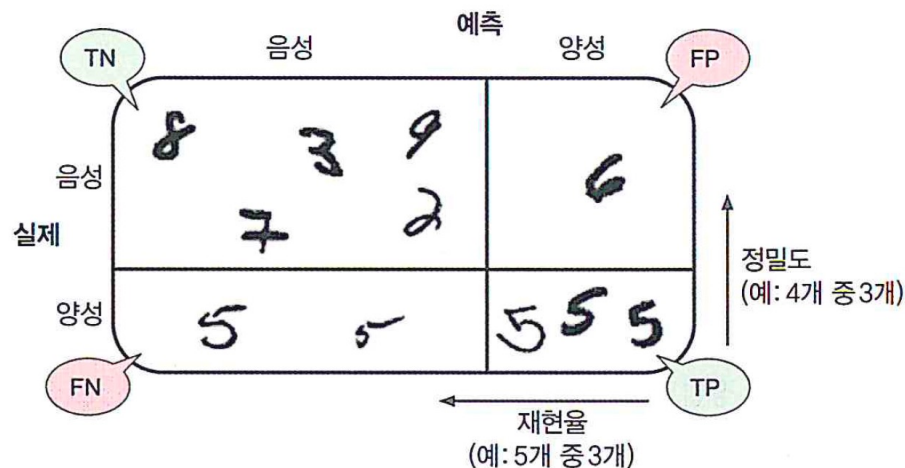
```
dnn_cls.add(Dense(Nh_l[1], activation='relu'))
```

```
dnn_cls.add(Dense(Nout, activation='softmax'))
```

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

```
#dnn_cls.summary()
```

□ Accuracy



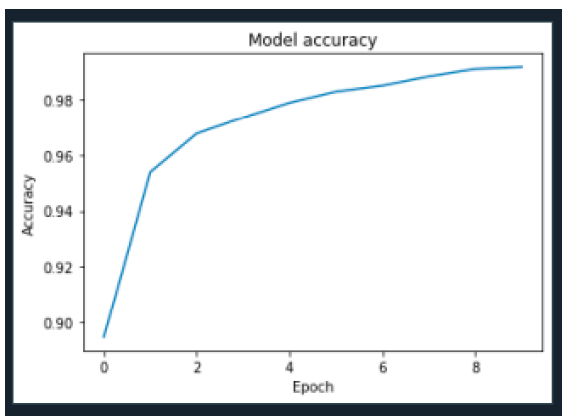
분류용 심층신경망(Deep Neural Network) 성능 (3/3)

□ MNIST 분류용 DNN 학습 (3/3)

```
# 3. Deep Neural Network 분류기 학습 및 성능평가 #####  
history = dnn_cls.fit(X_train, Y_train, epochs=10, batch_size=100, validation_split=0.2)  
performace_test = dnn_cls.evaluate(X_test, Y_test, batch_size=100)  
  
print('Test Loss and Accuracy ->', performace_test)  
  
#plt.plot(history.history['loss'])  
plt.plot(history.history['accuracy'])  
plt.title('Model accuracy')  
plt.ylabel('Accuracy')  
plt.xlabel('Epoch')
```

□ DNN 학습: epoch는 학습 단계수, batch_size는 한번에 학습되는 데이터 갯수

- ❖ epoch는 학습 단계수, validation_split을 이용하여 overfitting 여부를 확인
- ❖ batch_size는 한번에 학습되는 데이터 갯수, 숫자가 클수록 학습 속도 빠름



Test Loss and Accuracy -> [0.08797961473464966, 0.9761999845504761]