

인공신경망의 층을 추가하는 명령어: `model.add()`

- 입력층 개수 설정하는 속성: `input_dim = n`

신경망을 요약해주는 명령어: `model.summary()`

```
model = Sequential()
model.add(Dense(10, input_dim=X_train.shape[1], activation='relu'))
model.add(Dense(30, activation='relu'))
model.add(Dense(40, activation='relu'))
model.add(Dense(1))
model.summary()
```

해당 코드의 `summary` 결과를 작성하시오:

Model: "sequential"

| Layer (type) | Output Shape | Param # |
|-----------------|--------------|---------|
| dense (Dense) | (None, 10) | 60 |
| dense_1 (Dense) | (None, 30) | 330 |
| dense_2 (Dense) | (None, 40) | 1240 |
| dense_3 (Dense) | (None, 1) | 41 |

=====
 Total params: 1,671
 Trainable params: 1,671
 Non-trainable params: 0

0~255의 값을 0~1로 변경하는 과정: 정규화

- 딥러닝을 위하여 2차원 배열 데이터를 (1차원 배열로) 변경이 필요하다

컨볼루션 층을 추가하는 함수: `model.add(Conv2D())`

- 커널의 크기를 지정해 주는 속성: `kernel_size(x, y)`
- 입력 모양을 지정해주는 속성: `input_shape(x, y, color)`

이미지 크기 줄이는 연산: 풀링/ pooling

| | | | |
|---|---|---|---|
| 1 | 0 | 1 | 0 |
| 0 | 4 | 2 | 0 |
| 0 | 1 | 6 | 1 |
| 0 | 0 | 1 | 0 |

해당 그림을 기반으로 max pooling

| | |
|---|---|
| 4 | 2 |
| 1 | 6 |

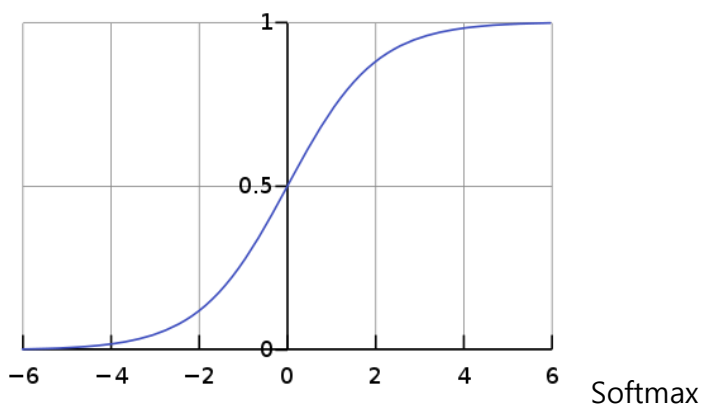
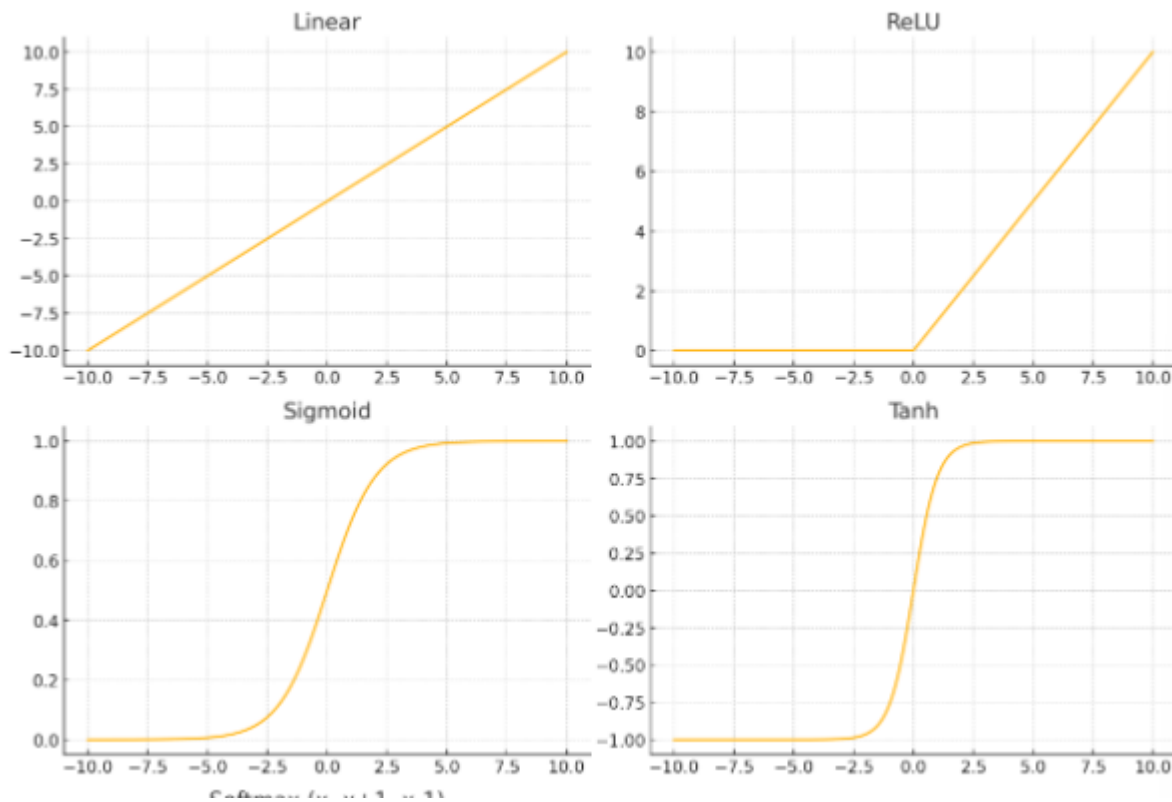
랜덤하게 특정 비율의 노드를 꺼주는 기법: Dropout

- 드롭아웃 25% 적용 코드: `model.add(Dropout(0.25))`

2차원 배열을 1차원으로 변경하는 기법: Flatten

- Flatten 적용 코드: `model.add(Flatten())`
- Flatten은 (Dense) 적용 전에 적용을 해야한다

CNN에서 마지막 활성화 함수는: Softmax



`to_categorical()`의 역할은? 원 핫 인코딩

입력된 텍스트를 잘게 나누는 과정: 토큰화

토큰화 함수: `text_to_word_sequence()`

원 핫 인코딩의 공간적 낭비를 해결하기 위하여 등장: 단어 임베딩/ word embedding

- 단어간의 (유사도)를 기준으로 분류

- 입력될 단어수 16개, 출력 벡터 크기는 4로 임베딩 하는 코드: Embedding(16, 4)

```
# 데이터를 불러와 학습셋, 테스트셋으로 나눕니다.
(X_train, y_train), (X_test, y_test) = reuters.load_data(num_words=1000,
test_split=0.2)

# 데이터를 확인해 보겠습니다.
category = np.max(y_train) + 1
print(category, '카테고리')
print(len(X_train), '학습용 뉴스 기사')
print(len(X_test), '테스트용 뉴스 기사')
print(X_train[0])

# 단어의 수를 맞추어 줍니다.
X_train = sequence.pad_sequences(X_train, maxlen=100)
X_test = sequence.pad_sequences(X_test, maxlen=100)
```

```
# 원-핫 인코딩 처리를 합니다.
y_train = to_categorical(y_train)
y_test = to_categorical(y_test)

# 모델의 구조를 설정합니다.
model = Sequential()
model.add(Embedding(1000, 100))
model.add(LSTM(100, activation='tanh'))
model.add(Dense(46, activation='softmax'))
```

```
# 모델의 실행 옵션을 정합니다.
model.compile(loss='categorical_crossentropy', optimizer='adam',
metrics=['accuracy'])

# 학습의 조기 중단을 설정합니다.
early_stopping_callback = EarlyStopping(monitor='val_loss', patience=5)

# 모델을 실행합니다.
history = model.fit(X_train, y_train, batch_size=20, epochs=200,
validation_data=(X_test, y_test), callbacks=[early_stopping_callback])

# 테스트 정확도를 출력합니다.
print("\n Test Accuracy: %.4f" % (model.evaluate(x_test, y_test)[1]))
```

```
# 학습셋과 테스트셋의 오차를 저장합니다.
y_vloss = history.history['val_loss']
y_loss = history.history['loss']

# 그래프로 표현해 보겠습니다.
x_len = np.arange(len(y_loss))
plt.plot(x_len, y_vloss, marker='.', c="red", label='Testset_loss')
plt.plot(x_len, y_loss, marker='.', c="blue", label='Trainset_loss')

# 그래프에 그리드를 주고 레이블을 표시하겠습니다.
plt.legend(loc='upper right')
plt.grid()
plt.xlabel('epoch')
plt.ylabel('loss')
plt.show()
```

LSTM의 약자: Long-Term Short Memory