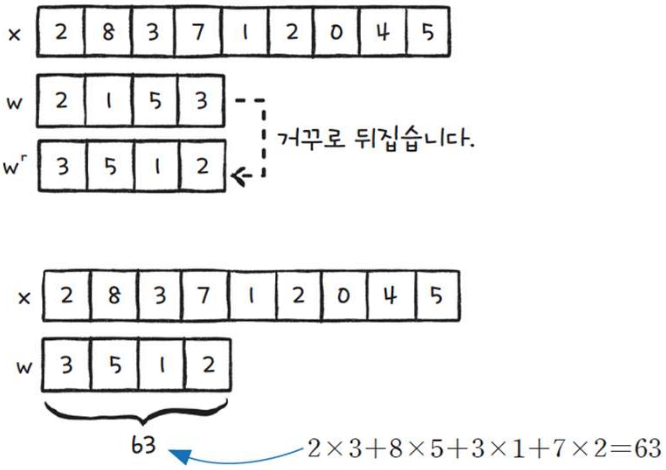
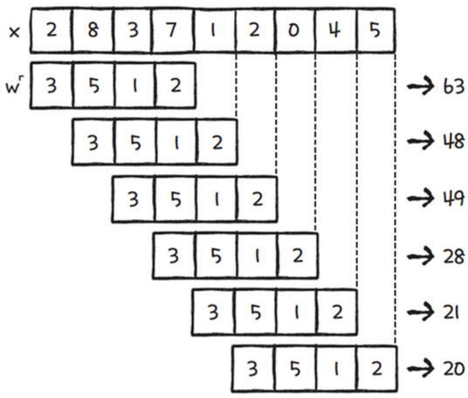
08 이미지를 분류합니다

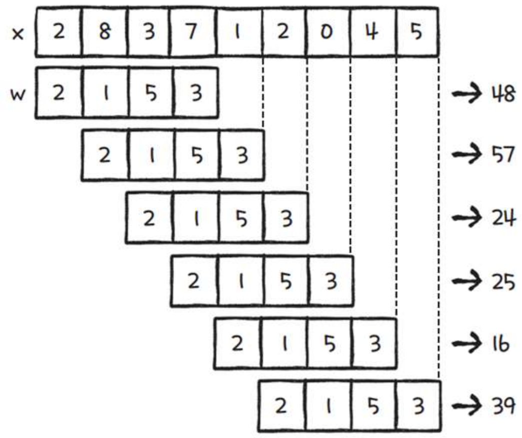
- 합성곱 신경망

합성곱 연산

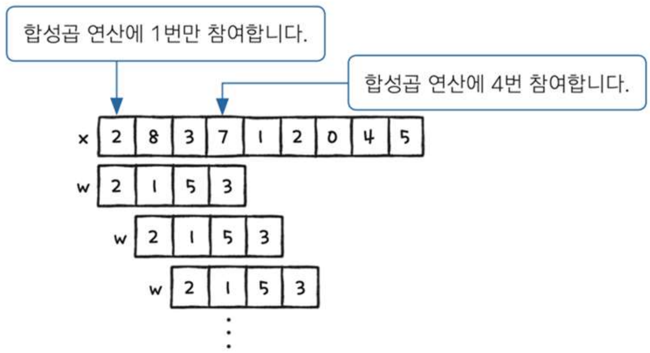




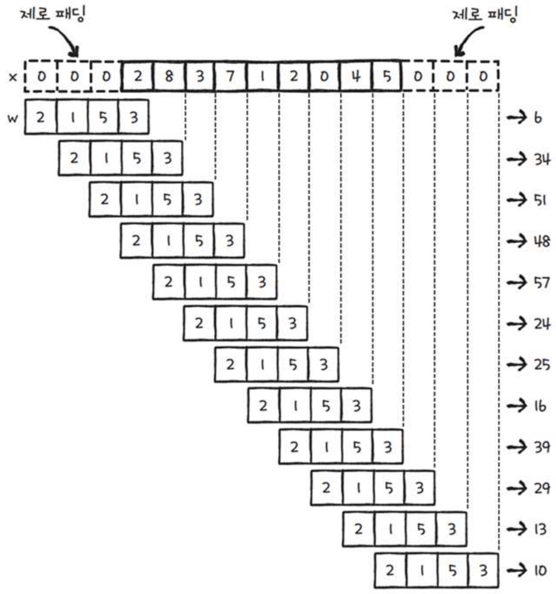
교차 상관 연산



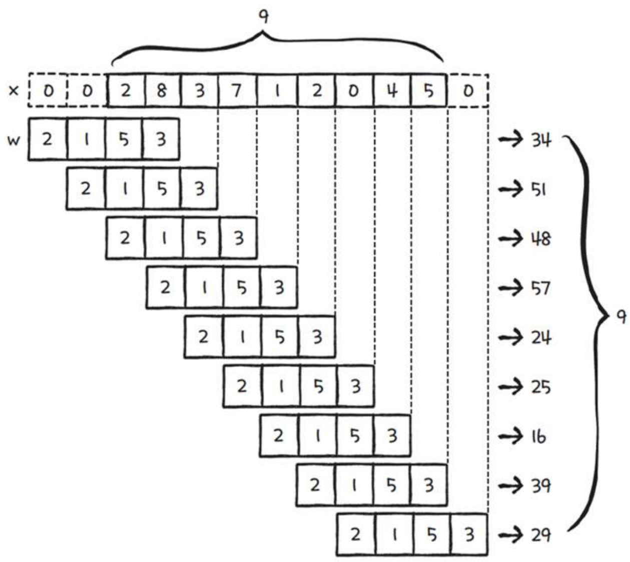
밸리드 패딩



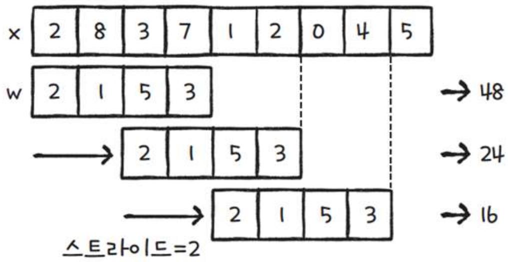
풀패딩


제로 패딩
제로 패딩
x 0 0 0 2 8 3 7 1 2 0 4 5 0 0 0
w 2 1 5 3
→ 6
→ 34
→ 51
→ 48
→ 57
→ 24
→ 25
→ 16
→ 39
→ 29
→ 13
→ 10

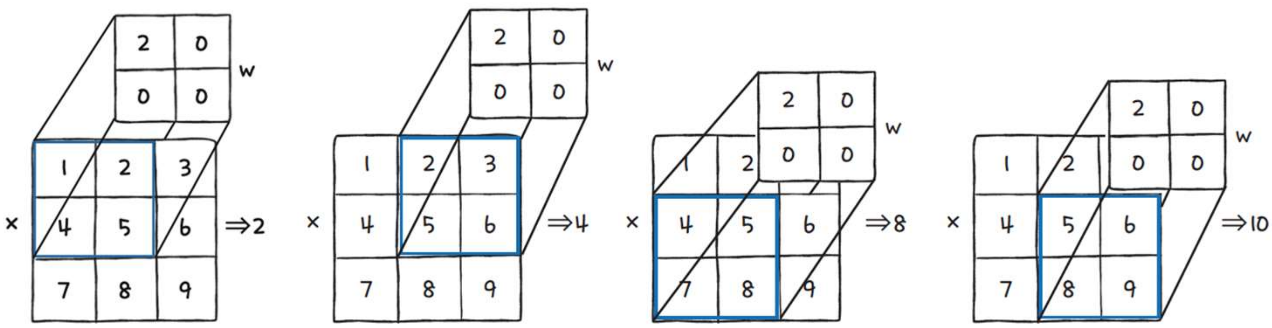
세임 패딩


9
x 0 0 2 8 3 7 1 2 0 4 5 0
w 2 1 5 3
→ 34
→ 51
→ 48
→ 57
→ 24
→ 25
→ 16
→ 39
→ 29
9

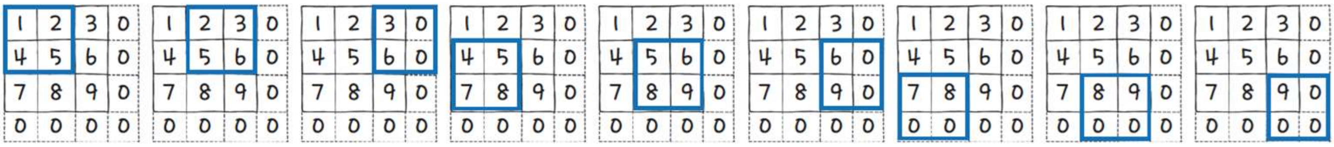
스트라이드



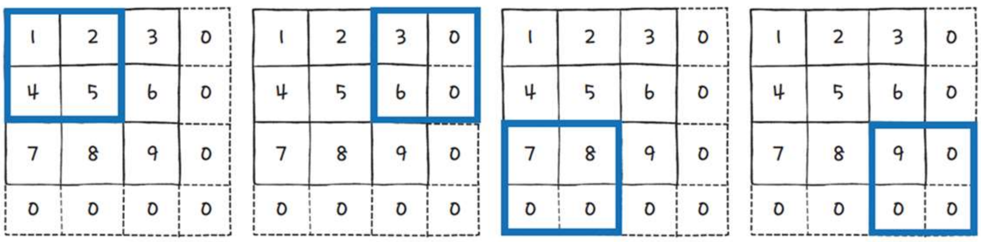
2차원 배열의 합성곱



2차원 배열의 세임 패딩

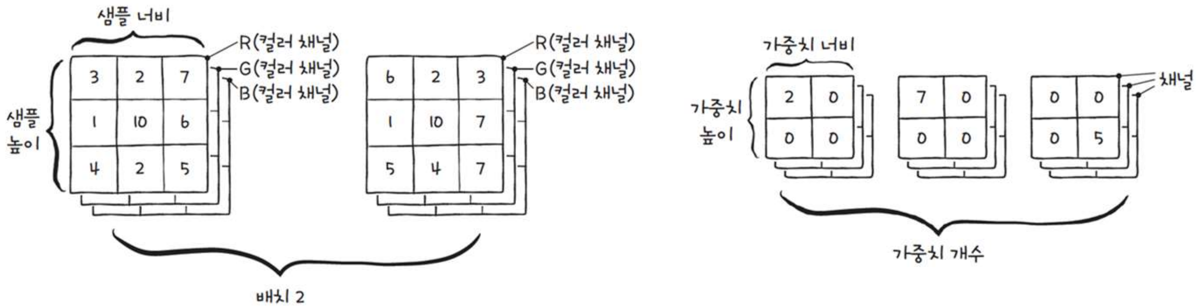


2차원 배열의 스트라이드

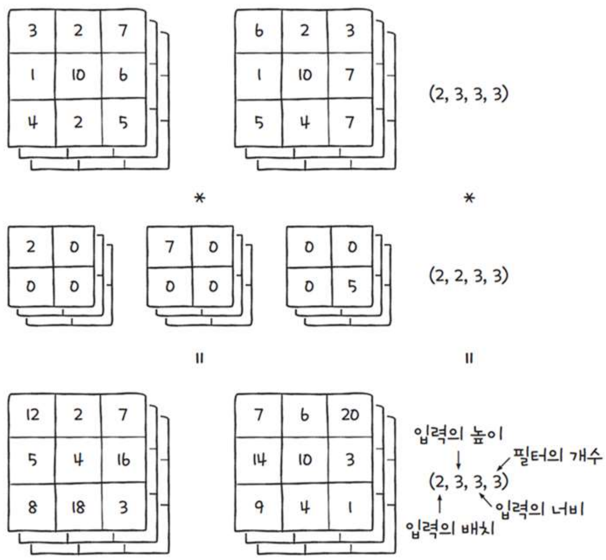


이미지 데이터와 커널

4차원 배열 사용

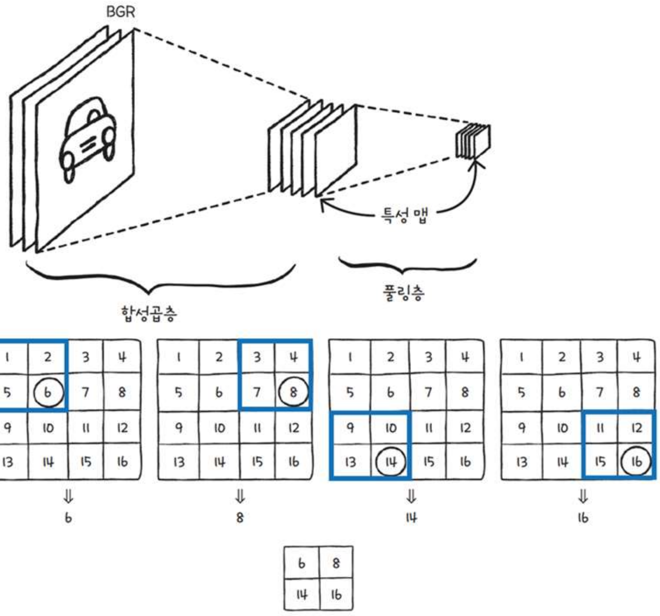


이미지 데이터의 합성곱 연산

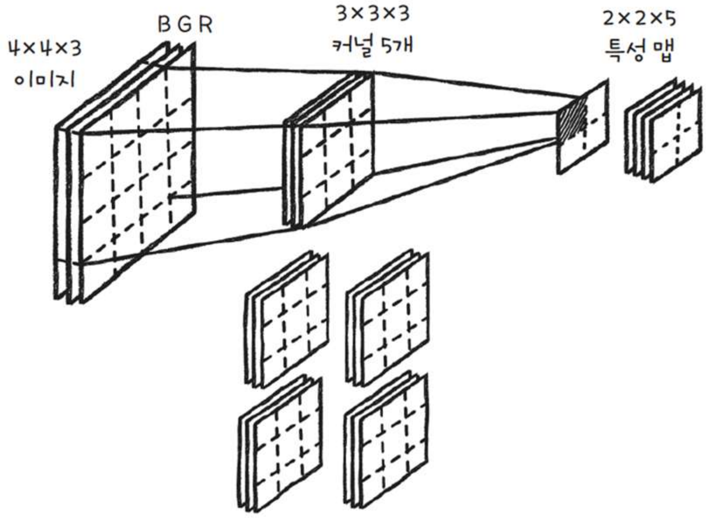


풀링

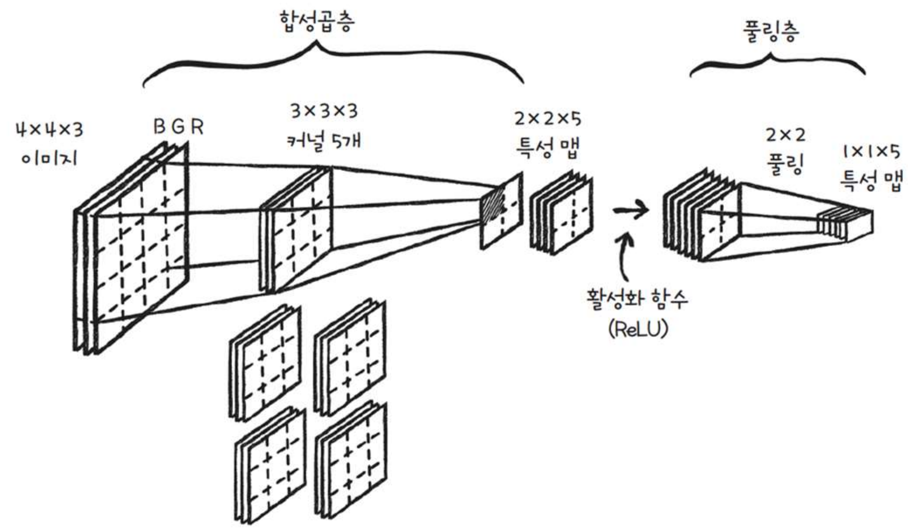
최대 풀링, 평균 풀링



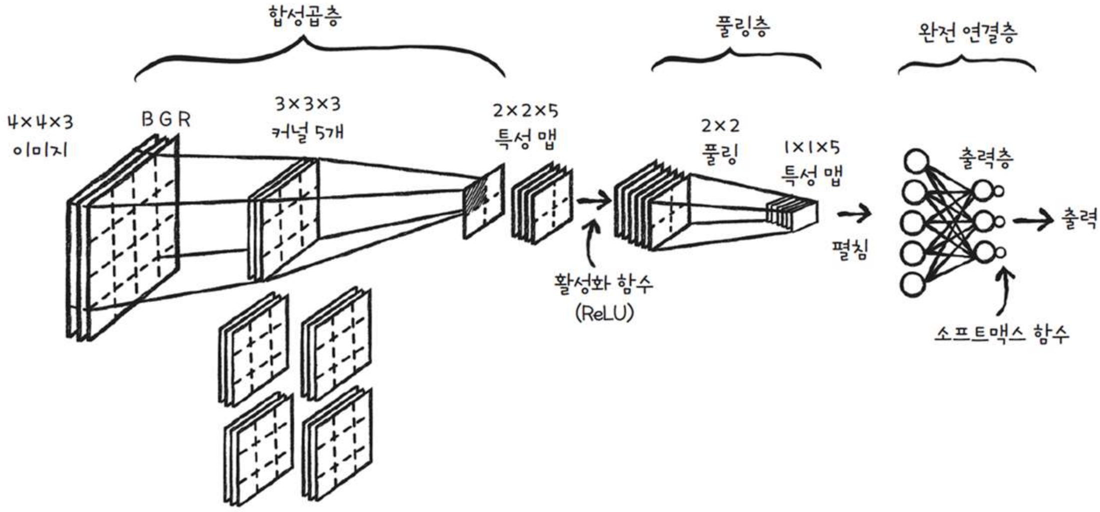
합성곱 층



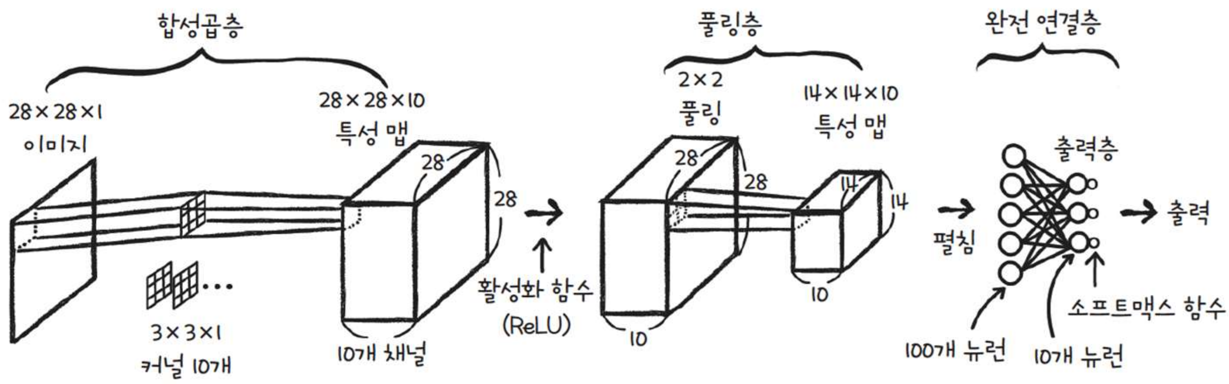
풀링 층


합성곱층
풀링층
4×4×3
이미지
BGR
3×3×3
커널 5개
2×2×5
특성 맵
2×2
풀링
1×1×5
특성 맵
활성화 함수
(ReLU)

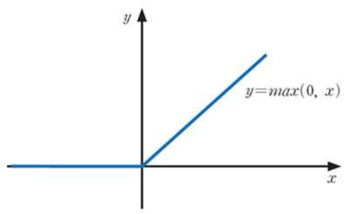
합성곱 신경망


합성곱층
풀링층
완전 연결층
4×4×3
이미지
BGR
3×3×3
커널 5개
2×2×5
특성 맵
2×2
풀링
1×1×5
특성 맵
출력층
출력
펼침
활성화 함수
(ReLU)
소프트맥스 함수

08-4 합성곱 신경망을 만들고 훈련합니다



정방향 계산



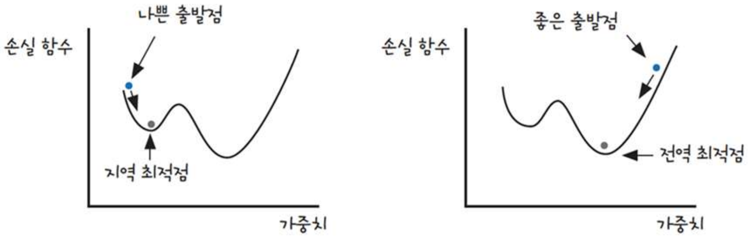

```
def forpass(self, x):
    # 3×3 합성곱 연산을 수행합니다.
    c_out = tf.nn.conv2d(x, self.conv_w, strides=1, padding='SAME') + self.conv_b
    # 렐루 활성화 함수를 적용합니다.
    r_out = tf.nn.relu(c_out)
    # 2×2 최대 풀링을 적용합니다.
    p_out = tf.nn.max_pool2d(r_out, ksize=2, strides=2, padding='VALID')
    # 첫 번째 배치 차원을 제외하고 출력을 일렬로 펼칩니다.
    f_out = tf.reshape(p_out, [x.shape[0], -1])
    z1 = tf.matmul(f_out, self.w1) + self.b1      # 첫 번째 층의 선형식을 계산합니다.
    a1 = tf.nn.relu(z1)                           # 활성화 함수를 적용합니다.
    z2 = tf.matmul(a1, self.w2) + self.b2         # 두 번째 층의 선형식을 계산합니다.
    return z2
```

역방향 계산

```
def training(self, x, y):
    m = len(x)                                  # 샘플 개수를 저장합니다.
    with tf.GradientTape( ) as tape:
        z = self.forpass(x)                     # 정방향 계산을 수행합니다.
        # 손실을 계산합니다.
        loss = tf.nn.softmax_cross_entropy_with_logits(y, z)
        loss = tf.reduce_mean(loss)

    weights_list = [self.conv_w, self.conv_b,
                    self.w1, self.b1, self.w2, self.b2]
    # 가중치에 대한 그레이디언트를 계산합니다.
    grads = tape.gradient(loss, weights_list)
    # 가중치를 업데이트합니다.
    self.optimizer.apply_gradients(zip(grads, weights_list))
```

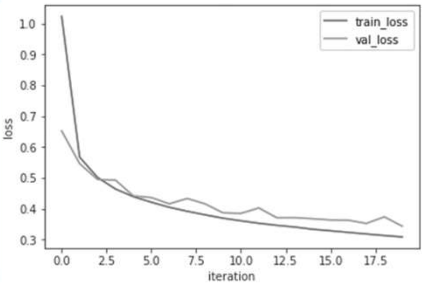
가중치 초기화

```
def init_weights(self, input_shape, n_classes):
    g = tf.initializers.glorot_uniform( )
    self.conv_w = tf.Variable(g((3, 3, 1, self.n_kernels)))
    self.conv_b = tf.Variable(np.zeros(self.n_kernels), dtype=float)
    n_features = 14 * 14 * self.n_kernels
    self.w1 = tf.Variable(g((n_features, self.units)))          # (특성 개수, 은닉층의 크기)
    self.b1 = tf.Variable(np.zeros(self.units), dtype=float)  # 은닉층의 크기
    self.w2 = tf.Variable(g((self.units, n_classes)))           # (은닉층의 크기, 클래스 개수)
    self.b2 = tf.Variable(np.zeros(n_classes), dtype=float)   # 클래스 개수
```



ConvolutionNetowrk 훈련

```
cn = ConvolutionNetwork(n_kernels=10, units=100, batch_size=128, learning_rate=0.01)
cn.fit(x_train, y_train_encoded, x_val=x_val, y_val=y_val_encoded, epochs=20)
에포크 0 .................................
에포크 1 .................................
.
.
.
에포크 19 ...............................
```



케라스로 합성곱 신경망 만들기

```
from tensorflow.keras.layers import Conv2D, MaxPooling2D, Flatten, Dense
```

```
conv1 = tf.keras.Sequential()
conv1.add(Conv2D(10, (3, 3), activation='relu', padding='same', input_shape=(28, 28, 1)))
conv1.add(MaxPooling2D((2, 2)))
conv1.add(Flatten())
conv1.add(Dense(100, activation='relu'))
conv1.add(Dense(10, activation='softmax'))
```

```
conv1.summary()
```

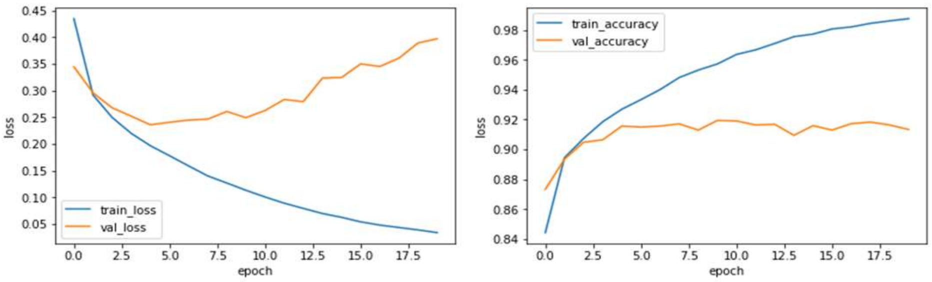
```
Model: "sequential"
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
conv2d (Conv2D)              (None, 28, 28, 10)        100
_________________________________________________________________
max_pooling2d (MaxPooling2D) (None, 14, 14, 10)        0
_________________________________________________________________
flatten (Flatten)            (None, 1960)              0
_________________________________________________________________
dense (Dense)                (None, 100)               196100
_________________________________________________________________
dense_1 (Dense)              (None, 10)                1010
=================================================================
Total params: 197,210
Trainable params: 197,210
Non-trainable params: 0
_________________________________________________________________
```

케라스 모델 훈련하기

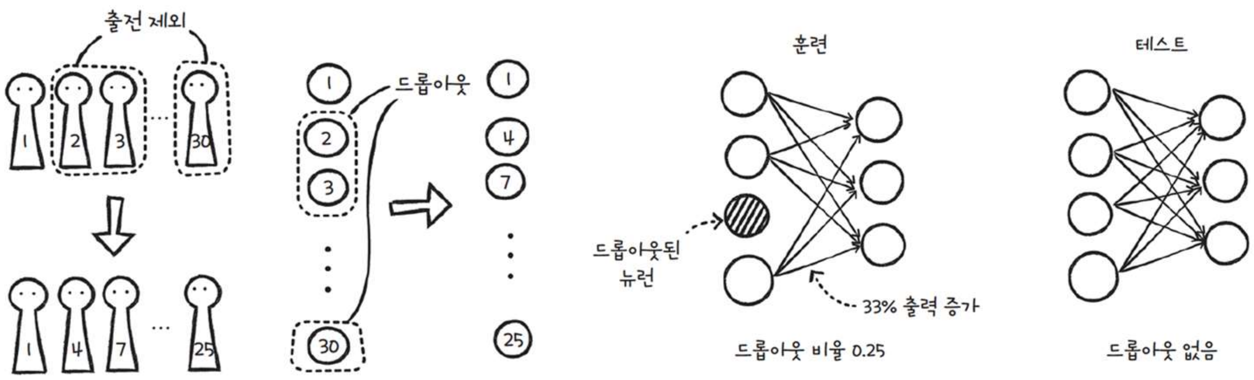
```
conv1.compile(optimizer='adam', loss='categorical_crossentropy',
              metrics=['accuracy'])
```

```
history = conv1.fit(x_train, y_train_encoded, epochs=20,
                    validation_data=(x_val, y_val_encoded))
```

```
Train on 48000 samples, validate on 12000 samples
Epoch 1/20
48000/48000 [==============================] - 6s 120us/sample - loss: 0.4349 - accuracy: 0.84
Epoch 2/20
48000/48000 [==============================] - 5s 100us/sample - loss: 0.2920 - accuracy: 0.89
```



드롭아웃



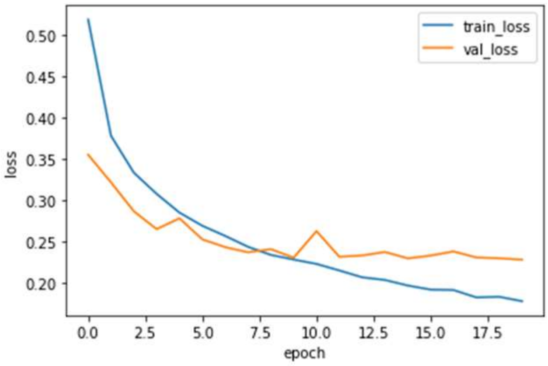
신경망에 드롭아웃 추가하기

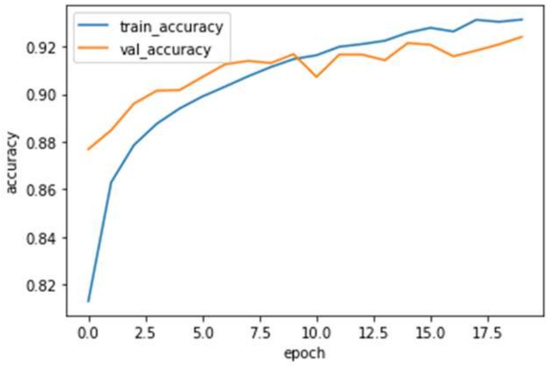
```
conv2 = tf.keras.Sequential()
conv2.add(Conv2D(10, (3, 3), activation='relu', padding='same', input_shape=(28, 28, 1)))
conv2.add(MaxPooling2D((2, 2)))
conv2.add(Flatten())
conv2.add(Dropout(0.5))
conv2.add(Dense(100, activation='relu'))
conv2.add(Dense(10, activation='softmax'))
```

```
conv2.summary()
```

```
Model: "sequential_1"
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
conv2d_1 (Conv2D)            (None, 28, 28, 10)        100
_________________________________________________________________
max_pooling2d_1 (MaxPooling2 (None, 14, 14, 10)        0
_________________________________________________________________
flatten_1 (Flatten)          (None, 1960)              0
_________________________________________________________________
dropout (Dropout)            (None, 1960)              0
_________________________________________________________________
dense_2 (Dense)              (None, 100)               196100
_________________________________________________________________
dense_3 (Dense)              (None, 10)                1010
=================================================================
Total params: 197,210
Trainable params: 197,210
Non-trainable params: 0
_________________________________________________________________
```

신경망에 드롭아웃 추가하기

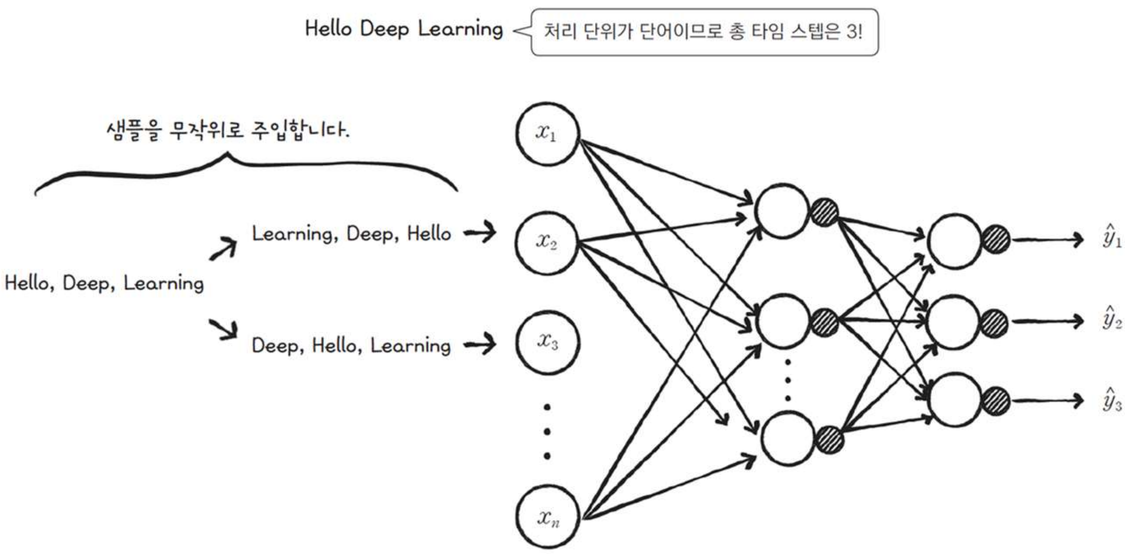




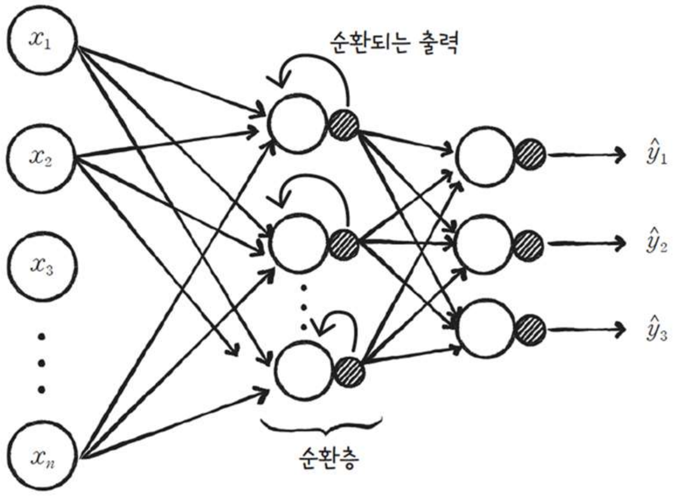
09 텍스트를 분류합니다

- 순환 신경망

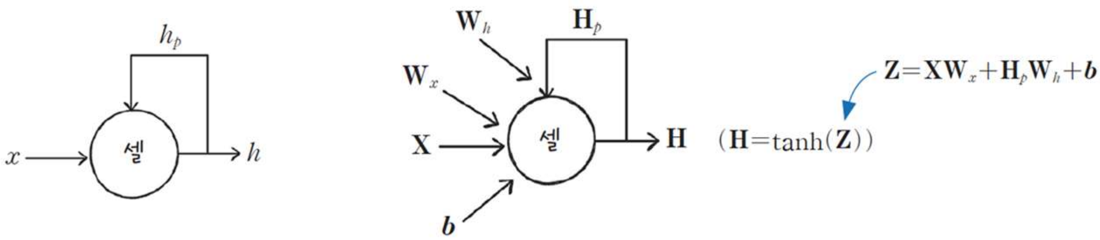
순차 데이터



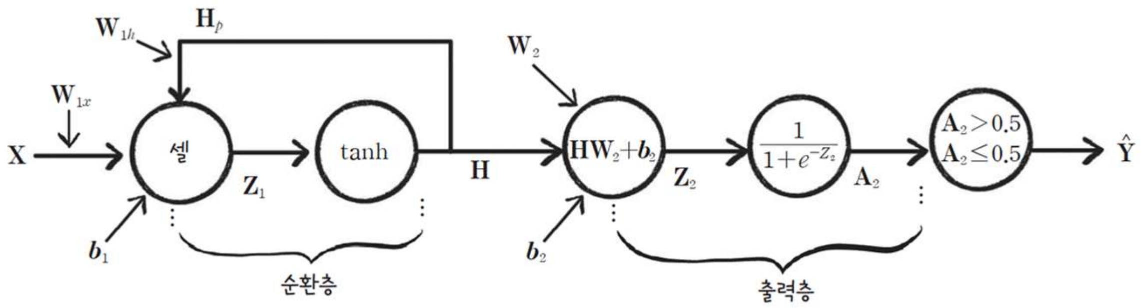
순환 신경망


순환되는 출력
x_1
x_2
x_3
x_n
$\hat{y}_1$
$\hat{y}_2$
$\hat{y}_3$
순환층

순환 셀

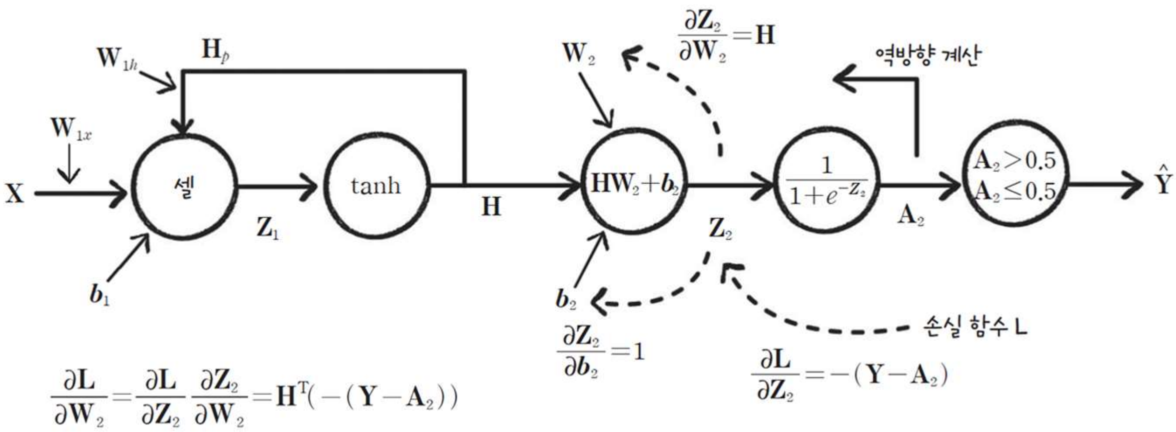

h_p
x
셀
h
$\mathbf{W}_h$
$\mathbf{H}_p$
$\mathbf{W}_x$
$\mathbf{X}$
셀
$\mathbf{H}$
$\boldsymbol{b}$
$(\mathbf{H}=\tanh(\mathbf{Z}))$
$\mathbf{Z}=\mathbf{X}\mathbf{W}_x+\mathbf{H}_p\mathbf{W}_h+\boldsymbol{b}$

정방향 계산



순환층의 정방향 계산	출력층의 정방향 계산
$Z_1=XW_{1x}+H_pW_{1h}+b_1$ $H=\tanh(Z_1)$	$Z_2=HW_2+b_2$ $A_2=\mathrm{sigmoid}(Z_2)$

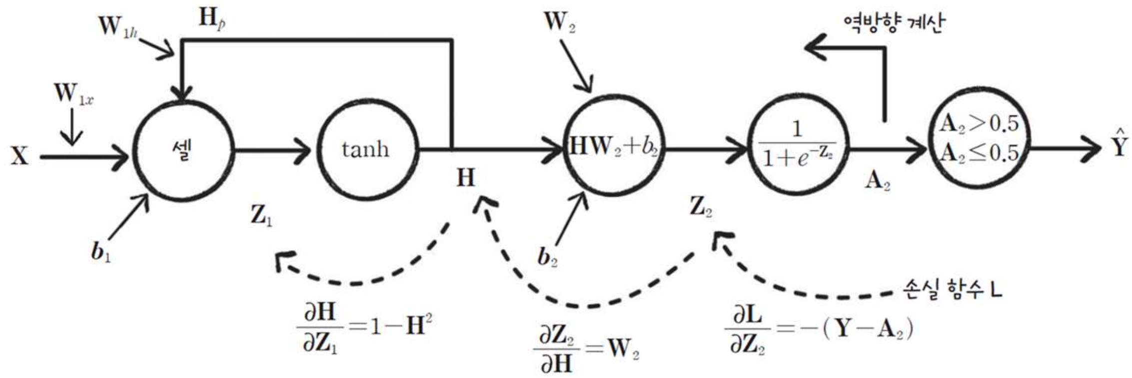
역방향 계산-1



$$\frac{\partial L}{\partial W_2}=\frac{\partial L}{\partial Z_2}\frac{\partial Z_2}{\partial W_2}=H^T(-(Y-A_2))$$

$$\frac{\partial L}{\partial b_2}=\frac{\partial L}{\partial Z_2}\frac{\partial Z_2}{\partial b_2}=1^T(-(Y-A_2))$$

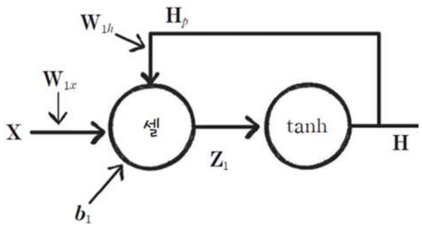
역방향 계산-2



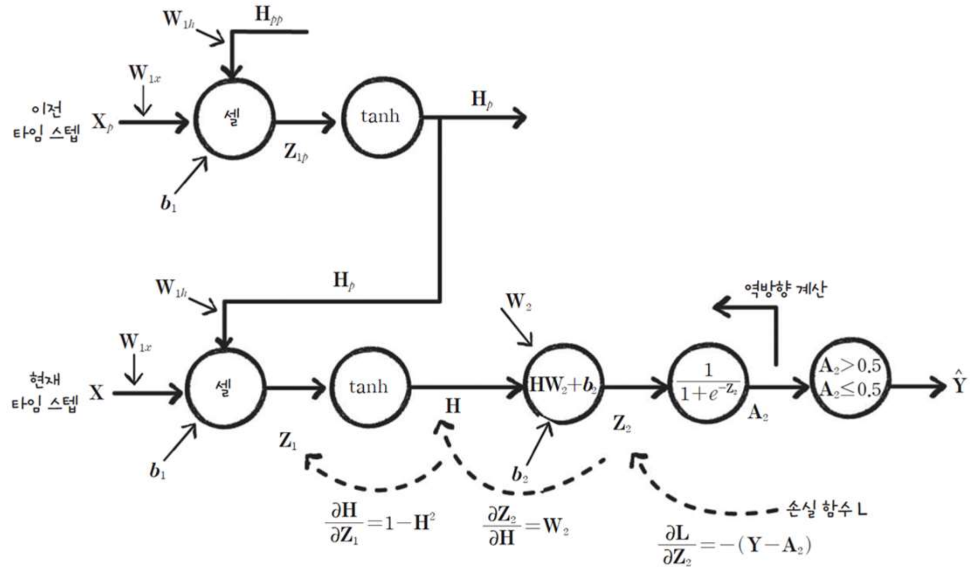
$$\frac{\partial \mathbf{H}}{\partial \mathbf{Z}_1}=\frac{\partial}{\partial \mathbf{Z}_1}\tanh(\mathbf{Z}_1)=1-\tanh^2(\mathbf{Z}_1)=1-\mathbf{H}^2 \qquad \frac{\partial \mathbf{Z}_2}{\partial \mathbf{H}}=\frac{\partial}{\partial \mathbf{H}}(\mathbf{H}\mathbf{W}_2+\boldsymbol{b}_2)=\mathbf{W}_2$$

역방향 계산-3

$$\frac{\partial \mathbf{Z}_1}{\partial \mathbf{W}_{1h}}=\frac{\partial}{\partial \mathbf{W}_{1h}}(\mathbf{X}\mathbf{W}_{1x}+\mathbf{H}_p\mathbf{W}_{1h}+\boldsymbol{b}_1)=\mathbf{H}_p\ ?$$



타임 스텝을 거슬러 역전파



타임스텝을 거슬러 역전파

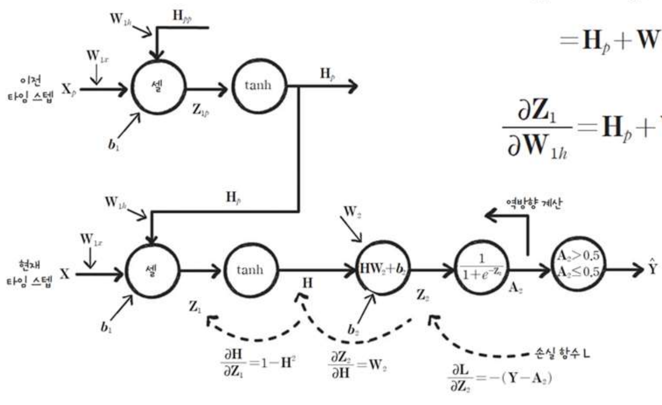
$$Z_{1p}=X_pW_{1x}+H_{pp}W_{1h}+b_1$$

$$H_p=\tanh(Z_{1p})$$

$$\frac{\partial Z_1}{\partial W_{1h}}=\frac{\partial}{\partial W_{1h}}(XW_{1x}+H_pW_{1h}+b_1)=\frac{\partial}{\partial W_{1h}}(H_pW_{1h})=H_p\frac{\partial W_{1h}}{\partial W_{1h}}+W_{1h}\frac{\partial H_p}{\partial W_{1h}}$$

$$=H_p+W_{1h}\frac{\partial H_p}{\partial W_{1h}}=H_p+W_{1h}\frac{\partial H_p}{\partial Z_{1p}}\frac{\partial Z_{1p}}{\partial W_{1h}}=H_p+W_{1h}\odot(1-H_p^2)\frac{\partial Z_{1p}}{\partial W_{1h}}$$

$$\frac{\partial Z_1}{\partial W_{1h}}=H_p+W_{1h}\odot(1-H_p^2)(H_{pp}+W_{1h}\odot(1-H_{pp}^2)\frac{\partial Z_{1pp}}{\partial W_{1h}})$$



타임스텝을 거슬러 역전파-2

$$\frac{\partial \mathbf{Z}_1}{\partial \mathbf{W}_{1h}} = \mathbf{H}_p + \mathbf{H}_{pp}\mathbf{W}_{1h} \odot (1-\mathbf{H}_p^2) + \mathbf{H}_{ppp}\mathbf{W}_{1h} \odot (1-\mathbf{H}_p^2) \odot \mathbf{W}_{1h} \odot (1-\mathbf{H}_{pp}^2) + \cdots$$

$$\frac{\partial \mathbf{Z}_1}{\partial \mathbf{W}_{1x}} = \mathbf{X} + \mathbf{X}\mathbf{W}_{1h} \odot (1-\mathbf{H}_p^2) + \mathbf{X}\mathbf{W}_{1h} \odot (1-\mathbf{H}_p^2) \odot \mathbf{W}_{1h} \odot (1-\mathbf{H}_{pp}^2) + \cdots$$

$$\frac{\partial \mathbf{Z}_1}{\partial \boldsymbol{b}_1} = 1 + \mathbf{W}_{1h} \odot (1-\mathbf{H}_p^2) + \mathbf{W}_{1h} \odot (1-\mathbf{H}_p^2) \odot \mathbf{W}_{1h} \odot (1-\mathbf{H}_{pp}^2) + \cdots$$

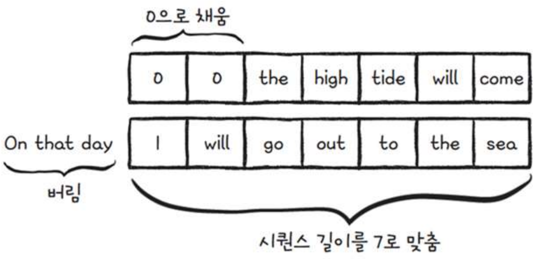
09-2 순환 신경망을 만들고 텍스트를 분류합니다

데이터 로드

```
import numpy as np
from tensorflow.keras.datasets import imdb

(x_train_all, y_train_all), (x_test, y_test) = imdb.load_data(skip_top=20, num_words=100)
```

```
print(x_train_all.shape, y_train_all.shape)

(25000,) (25000,)
```

샘플 길이 맞추고 원-핫 인코딩하기

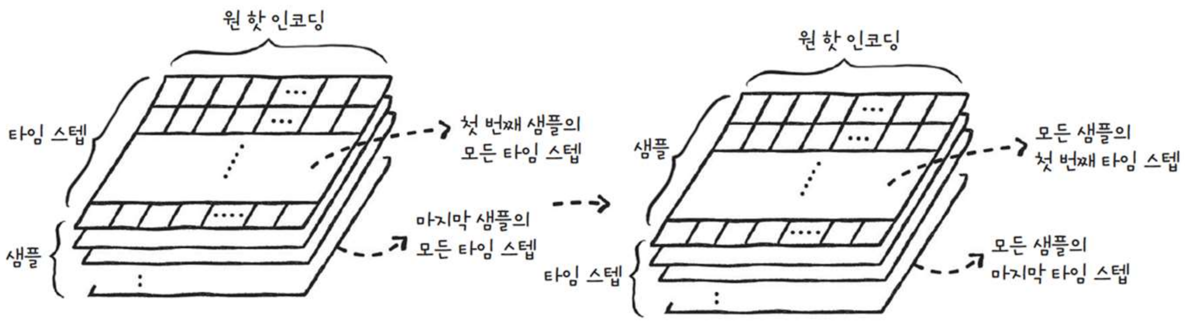



```
from tensorflow.keras.preprocessing import sequence

maxlen=100
x_train_seq = sequence.pad_sequences(x_train, maxlen=maxlen)
x_val_seq = sequence.pad_sequences(x_val, maxlen=maxlen)
```

```
from tensorflow.keras.utils import to_categorical

x_train_onehot = to_categorical(x_train_seq)
x_val_onehot = to_categorical(x_val_seq)
```

배치 차원 바꾸기

```
# 배치 차원과 타임 스텝 차원을 바꿉니다.
seq = np.swapaxes(x, 0, 1)
```



정방향 계산

```
...
# 순환층의 선형식을 계산합니다.
for x in seq:
    z1 = np.dot(x, self.w1x) + np.dot(self.h[-1], self.w1h) + self.b1
    h = np.tanh(z1)                     # 활성화 함수를 적용합니다.
    self.h.append(h)                    # 역전파를 위해 은닉 상태를 저장합니다.
    z2 = np.dot(h, self.w2) + self.b2  # 출력층의 선형식을 계산합니다.
return z2
```

역방향 계산

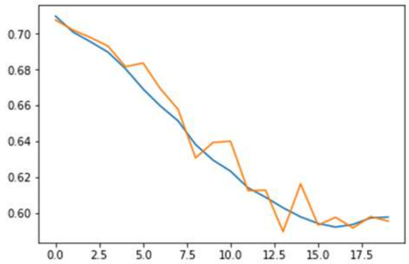
```
def backprop(self, x, err):
    m = len(x)          # 샘플 개수

    # 출력층의 가중치와 절편에 대한 그레이디언트를 계산합니다.
    w2_grad = np.dot(self.h[-1].T, err) / m
    b2_grad = np.sum(err) / m
    # 배치 차원과 타임 스텝 차원을 바꿉니다.
    seq = np.swapaxes(x, 0, 1)

    w1h_grad = w1x_grad = b1_grad = 0
    # 셀 직전까지 그레이디언트를 계산합니다.
    err_to_cell = np.dot(err, self.w2.T) * (1 - self.h[-1] ** 2)
    # 모든 타임 스텝을 거슬러 가면서 그레이디언트를 전파합니다.
    for x, h in zip(seq[::-1][:10], self.h[:-1][::-1][:10]):
        w1h_grad += np.dot(h.T, err_to_cell)
        w1x_grad += np.dot(x.T, err_to_cell)
        b1_grad += np.sum(err_to_cell, axis=0)
        # 이전 타임 스텝의 셀 직전까지 그레이디언트를 계산합니다.
        err_to_cell = np.dot(err_to_cell, self.w1h) * (1 - h ** 2)
```

$$\frac{\partial \mathbf{Z}_1}{\partial \mathbf{W}_{1h}} = \mathbf{H}_p + \mathbf{H}_{pp}\mathbf{W}_{1h} \odot (1 - \mathbf{H}_p^2) + \mathbf{H}_{ppp}\mathbf{W}_{1h} \odot (1 - \mathbf{H}_p^2) \odot \mathbf{W}_{1h} \odot (1 - \mathbf{H}_{pp}^2) + \cdots$$

```
    return w1h_grad, w1x_grad, b1_grad, w2_grad, b2_grad
```

모델 훈련

```
rn = RecurrentNetwork(n_cells=32, batch_size=32, learning_rate=0.01)

rn.fit(x_train_onehot, y_train, epochs=20, x_val=x_val_onehot, y_val=y_val)

에포크 0 ............................................................................
에포크 1 ............................................................................
에포크 2 ............................................................................
...
```



텐서플로로 순환 신경망을 만들기

```
from tensorflow.keras.models import Sequential
from tensorflow.keras.layers import Dense, SimpleRNN
```

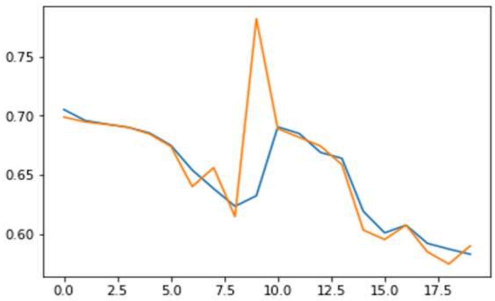
```
model = Sequential()

model.add(SimpleRNN(32, input_shape=(100, 100)))
model.add(Dense(1, activation='sigmoid'))

model.summary()
```

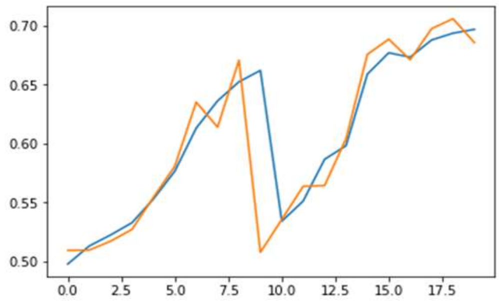
```
Model: "sequential"
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
simple_rnn (SimpleRNN)       (None, 32)                4256
_________________________________________________________________
dense (Dense)                (None, 1)                 33
=================================================================
Total params: 4,289
Trainable params: 4,289
Non-trainable params: 0
_________________________________________________________________
```

모델 훈련

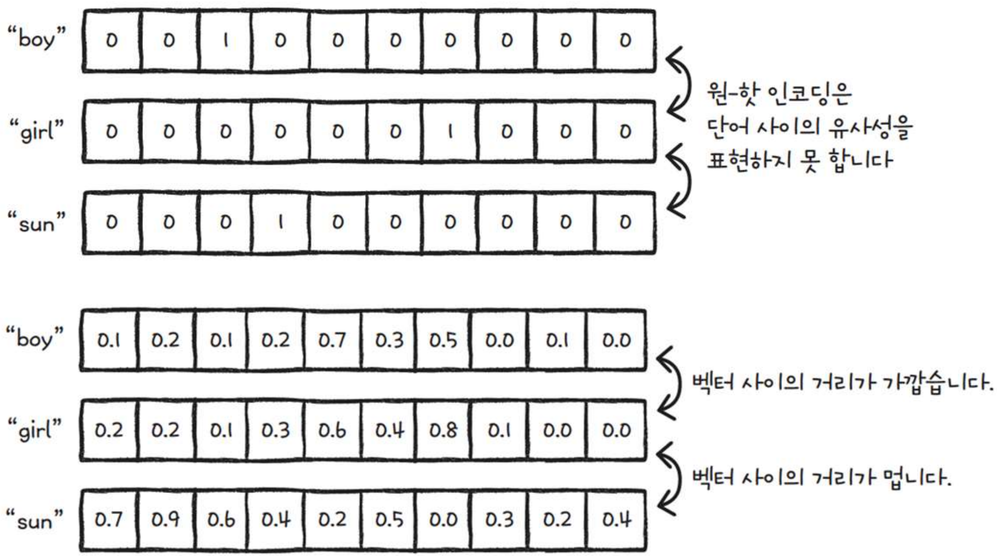
```
model.compile(optimizer='sgd', loss='binary_crossentropy', metrics=['accuracy'])

history = model.fit(x_train_onehot, y_train, epochs=20, batch_size=32,
                    validation_data=(x_val_onehot, y_val))
```

```
Train on 20000 samples, validate on 5000 samples
Epoch 1/20
20000/20000 [==============================] - 19s 935us/sample - loss: 0.7051 - accuracy: 0.4
Epoch 2/20
20000/20000 [==============================] - 17s 864us/sample - loss: 0.6958 - accuracy: 0.5
Epoch 3/20
```





워드 임베딩

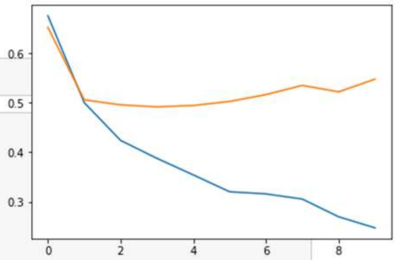


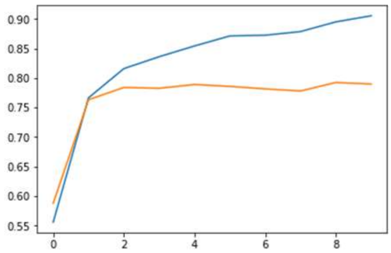
워드 임베딩 신경망 만들기

```python
from tensorflow.keras.layers import Embedding
```

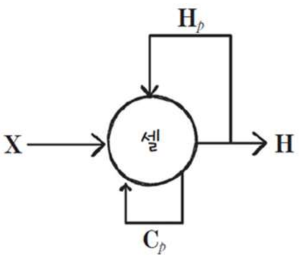
```python
model_ebd = Sequential()

model_ebd.add(Embedding(1000, 32))
model_ebd.add(SimpleRNN(8))
model_ebd.add(Dense(1, activation='sigmoid'))

model_ebd.summary()
```

```
Model: "sequential_1"
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
embedding (Embedding)        (None, None, 32)          32000
_________________________________________________________________
simple_rnn_1 (SimpleRNN)     (None, 8)                 328
_________________________________________________________________
dense_1 (Dense)              (None, 1)                 9
=================================================================
Total params: 32,337
Trainable params: 32,337
Non-trainable params: 0
_________________________________________________________________
```





09-4 LSTM 순환 신경망을 만들고 텍스트를 분류합니다

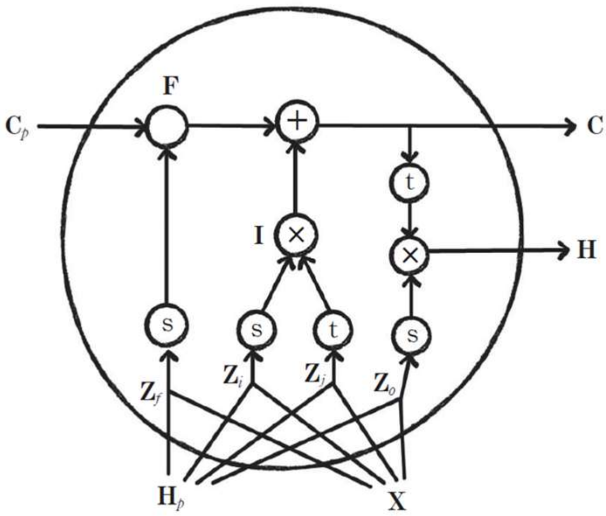


$\mathbf{F} = \mathrm{C}_p \times \mathrm{sigmoid}(\mathbf{Z}_f)$

$\mathbf{I} = \mathrm{sigmoid}(\mathbf{Z}_i) \times \tanh(\mathbf{Z}_j)$

$\mathrm{C} = \mathbf{F} + \mathbf{I}$

$\mathbf{H} = \tanh(\mathrm{C}) \times \mathrm{sigmoid}(\mathbf{Z}_o)$

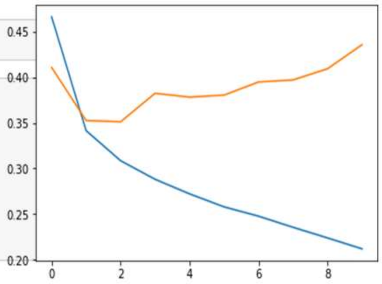


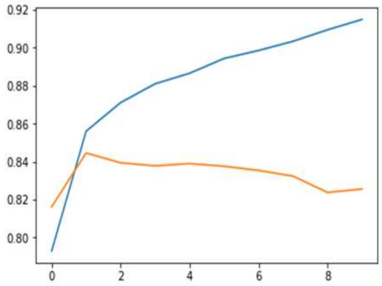
텐서플로로 LSTM 순환 신경망 만들기

```
from tensorflow.keras.layers import LSTM
```

```
model_lstm = Sequential()

model_lstm.add(Embedding(1000, 32))
model_lstm.add(LSTM(8))
model_lstm.add(Dense(1, activation='sigmoid'))

model_lstm.summary()
```

```
Model: "sequential_2"
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
embedding_1 (Embedding)      (None, None, 32)          32000
_________________________________________________________________
lstm (LSTM)                  (None, 8)                 1312
_________________________________________________________________
dense_2 (Dense)              (None, 1)                 9
=================================================================
Total params: 33,321
Trainable params: 33,321
Non-trainable params: 0
_________________________________________________________________
```





감사합니다