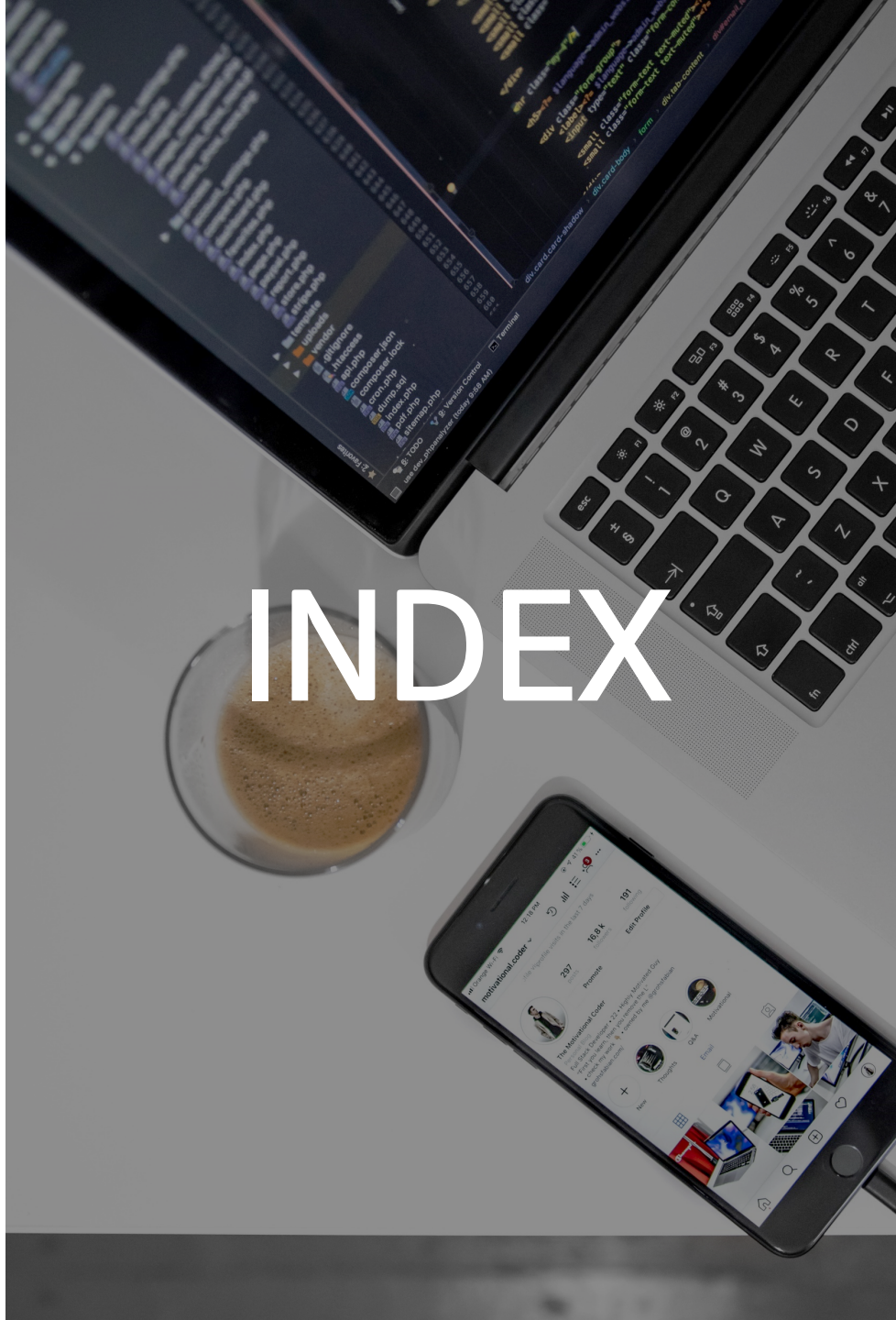
Summer NLP

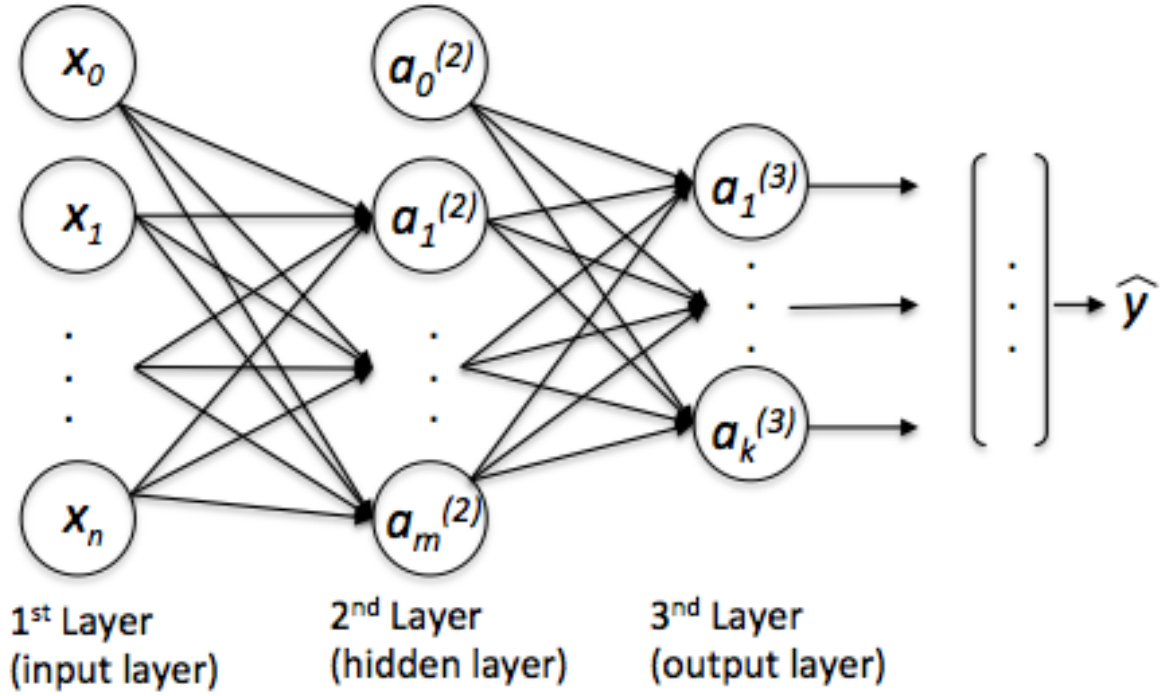
Lec 03.Feed Forward
Neural Network for NLP

파이토치로 배우는 자연어 처리 CHAPTER 4

INDEX



다층 퍼셉트론



다층 퍼셉트론 (Multi Layer Perceptron)

퍼셉트론의 Layer를 수직적으로 쌓아서 만든 Network

다층 퍼셉트론

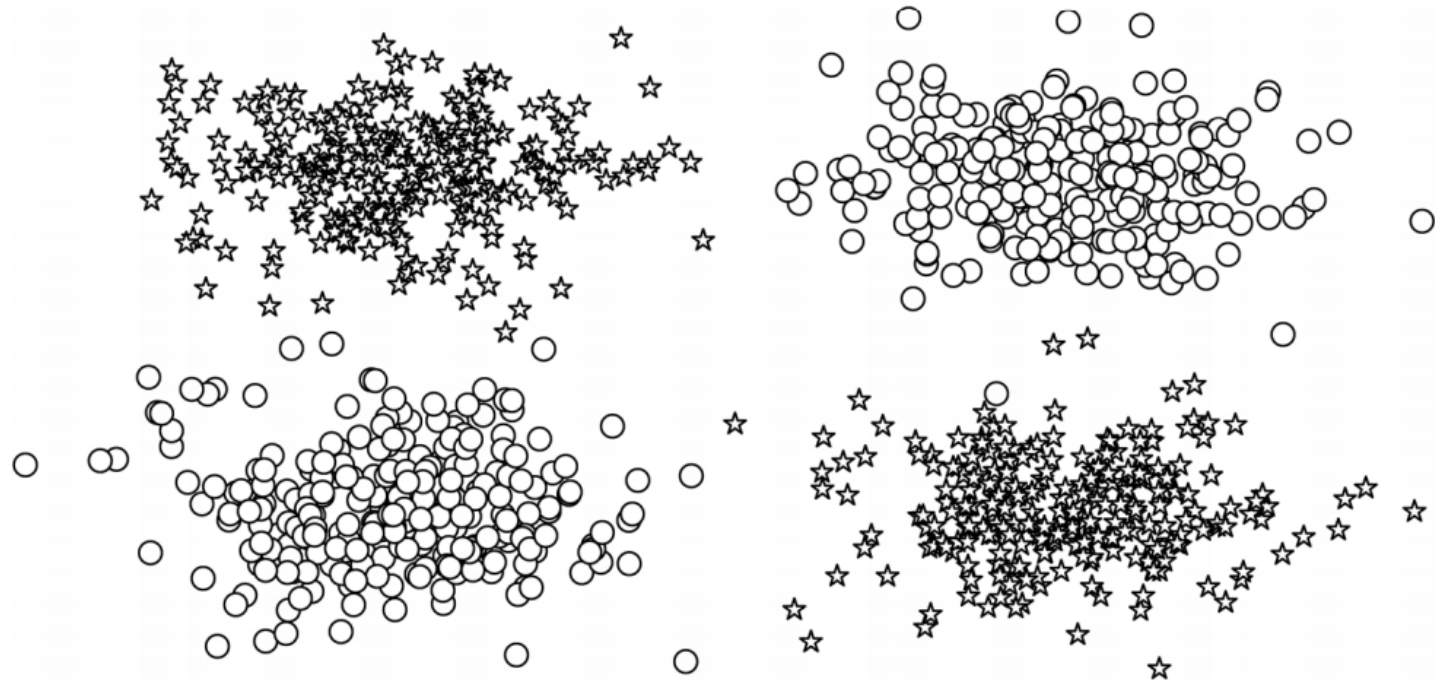


Figure 4-1. Two classes in the XOR dataset plotted as circles and stars. Notice how no single line can separate the two classes.

다층 퍼셉트론 (Multi Layer Perceptron)

Perceptron은 선형적으로만 학습되어 XOR 등을 학습할 수 없는 문제점을 해결

다층 퍼셉트론

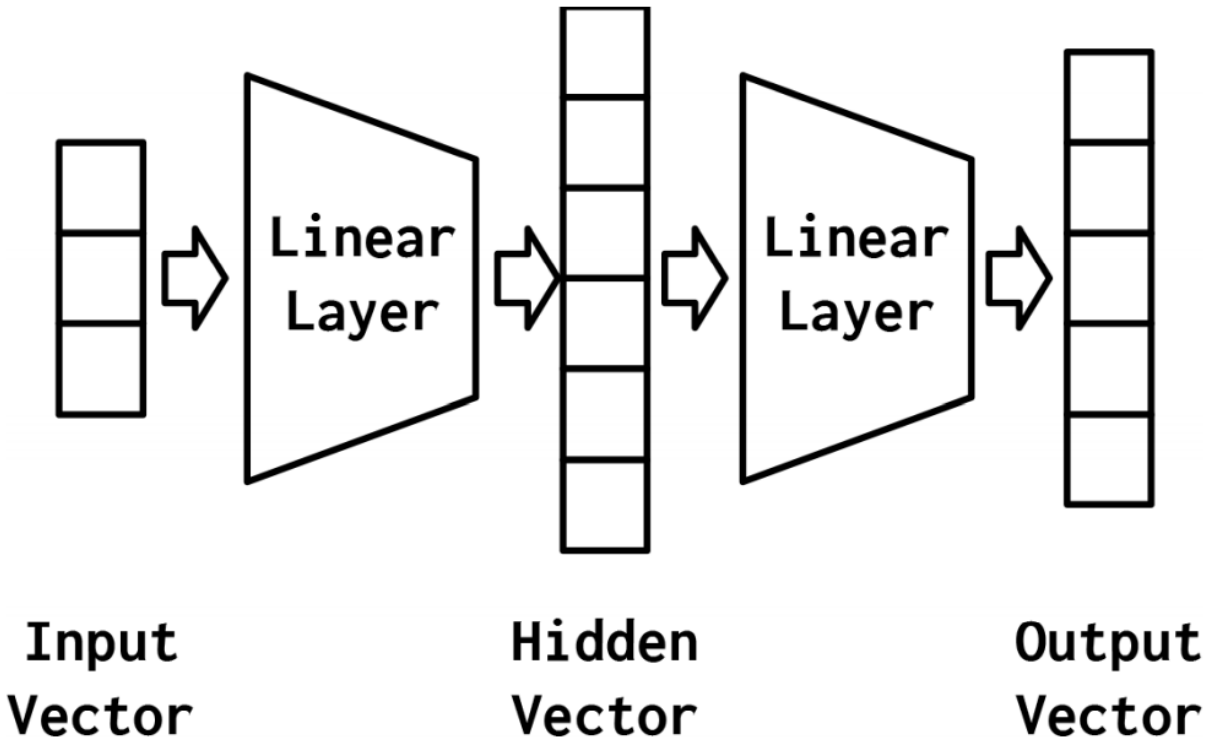


Figure 4-2. A visual representation of an MLP with two linear layers and three stages of representation—the input vector, the hidden vector, and the output vector.

다층 퍼셉트론 (Multi Layer Perceptron)

Hidden Layer는 Hidden Vector를 만들고, 여러 Layer를 거쳐 Output Vector를 만듦

다층 퍼셉트론

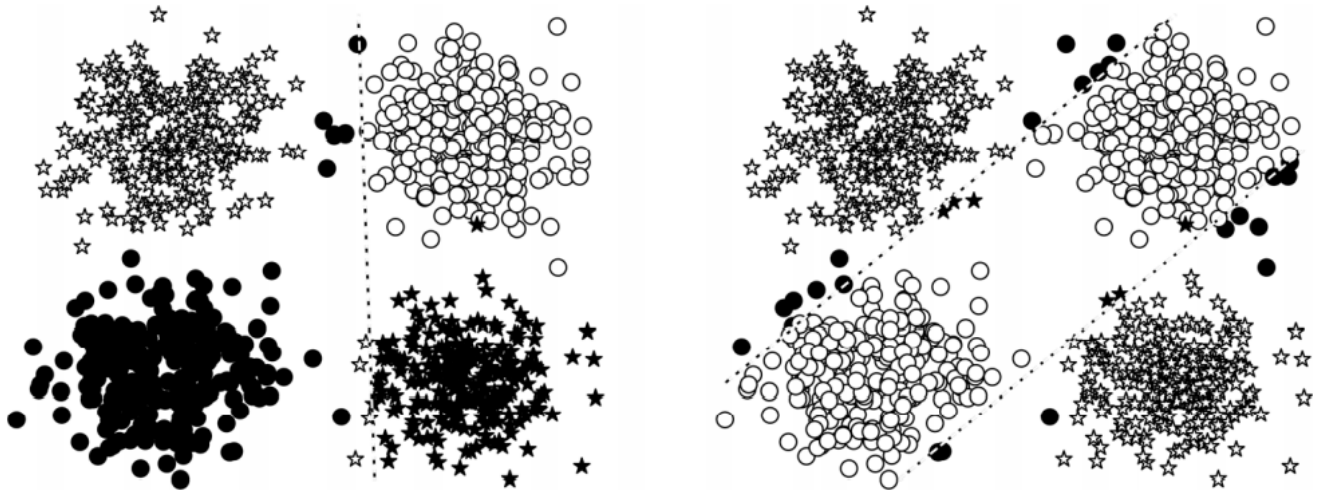


Figure 4-3. The learned solutions from the perceptron (left) and MLP (right) for the XOR problem. The true class of each data point is the point's shape: star or circle. Incorrect classifications are filled in with black and correct classifications are not filled in. The lines are the decision boundaries of each model. In the left panel, a perceptron learns a decision boundary that cannot correctly separate the circles from the stars. In fact, no single line can. In the right panel, an MLP has learned to separate the stars from the circles.

Perceptron vs MLP

Perceptron은 단 하나의 Decision Boundary를 가져서 분류가 힘듦

다층 퍼셉트론

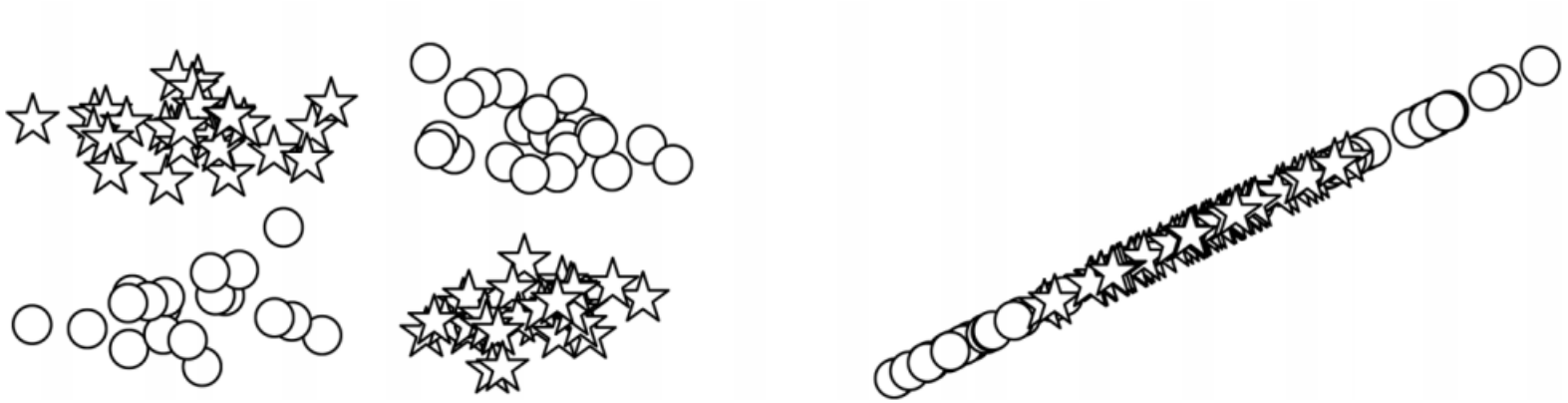


Figure 4-5. The input and output representations of the perceptron. Because it does not have an intermediate representation to group and reorganize as the MLP can, it cannot separate the circles and stars.

Perceptron vs MLP

Perceptron은 다음과 같은 중간 표현을 학습, 선형적으로 분류 불가능

다층 퍼셉트론

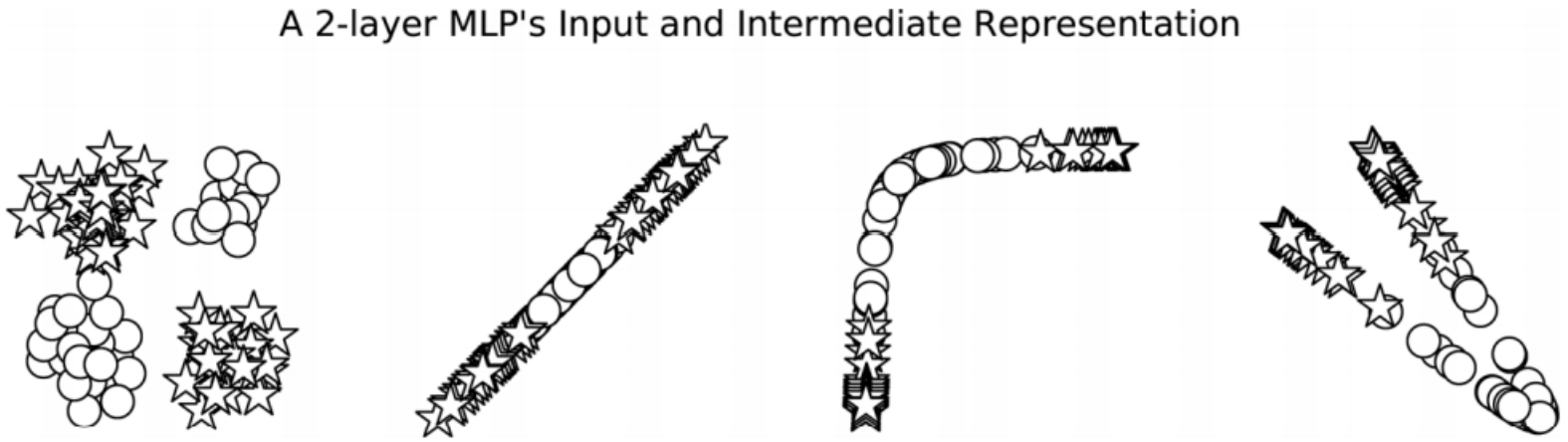


Figure 4-4. The input and intermediate representations for an MLP. From left to right: (1) the input to the network, (2) the output of the first linear module, (3) the output of the first nonlinearity, and (4) the output of the second linear module. As you can see, the output of the first linear module groups the circles and stars, whereas the output of the second linear module reorganizes the data points to be linearly separable.

Perceptron vs MLP

MLP는 입력 데이터를 변형하여 최종 Layer에서는 선형적으로 분류 가능

다층 퍼셉트론

Example 4-1. Multilayer perceptron using PyTorch

```
import torch.nn as nn
import torch.nn.functional as F

class MultilayerPerceptron(nn.Module):
    def __init__(self, input_dim, hidden_dim, output_dim):
        """
        Args:
            input_dim (int): the size of the input vectors
            hidden_dim (int): the output size of the first Linear layer
            output_dim (int): the output size of the second Linear layer
        """
        super(MultilayerPerceptron, self).__init__()
        self.fc1 = nn.Linear(input_dim, hidden_dim)
        self.fc2 = nn.Linear(hidden_dim, output_dim)

    def forward(self, x_in, apply_softmax=False):
        """The forward pass of the MLP
        Args:
            x_in (torch.Tensor): an input data tensor
                x_in.shape should be (batch, input_dim)
            apply_softmax (bool): a flag for the softmax activation
                should be false if used with the cross-entropy losses
        Returns:
            the resulting tensor. tensor.shape should be (batch, output_dim)
        """
        intermediate = F.relu(self.fc1(x_in))
        output = self.fc2(intermediate)

        if apply_softmax:
            output = F.softmax(output, dim=1).
        return output
```

Example 4-2. An example instantiation of an MLP

Input[0]

```
batch_size = 2 # number of samples input at once
input_dim = 3
hidden_dim = 100
output_dim = 4

# Initialize model
mlp = MultilayerPerceptron(input_dim, hidden_dim, output_dim)
print(mlp)
```

Output[0]

```
MultilayerPerceptron(
  (fc1): Linear(in_features=3, out_features=100, bias=True)
  (fc2): Linear(in_features=100, out_features=4, bias=True)
  (relu): ReLU()
)
```

MLP in PyTorch

nn.Linear와 ReLU를 통해 MLP 적용

MLP로 **성씨 분류하기**

	surname	nationality
0	Woodford	English
1	Coté	French
2	Kore	English
3	Koury	Arabic
4	Lebzak	Russian

Surname Dataset

성씨를 국적에 따라 분류된 Dataset

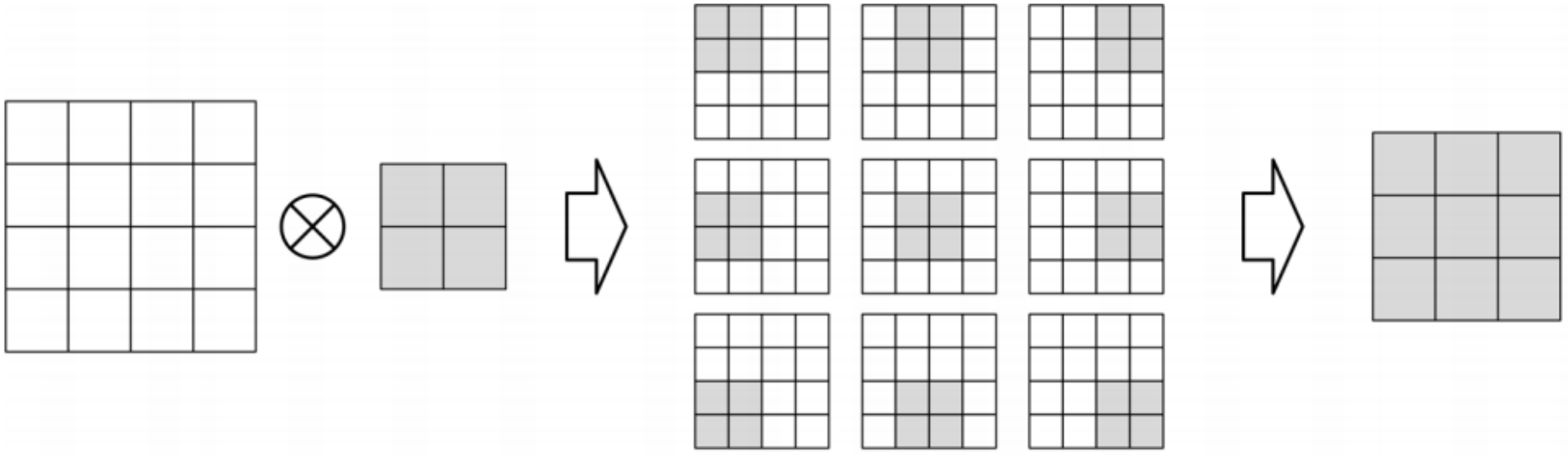
MLP로 **성씨 분류하기**

Notebook Link

Lets Code!

With Colab

합성곱 **신경망**



An input matrix
and a convolutional kernel

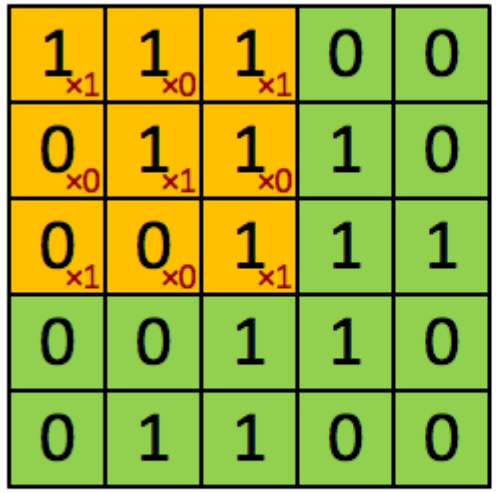
The kernel being applied to the
input matrix

The output matrix
(feature map)

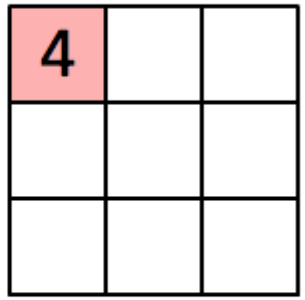
Convolutional Neural Network

공간상의 부분 구조를 감지하는데 적합한 신경망

합성곱 **신경망**



Image

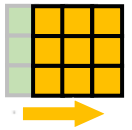


Convolved Feature

Terminology

Channel : RGB input –〉 3 Channels

Filter (Kernel) : Kernel size 3x3



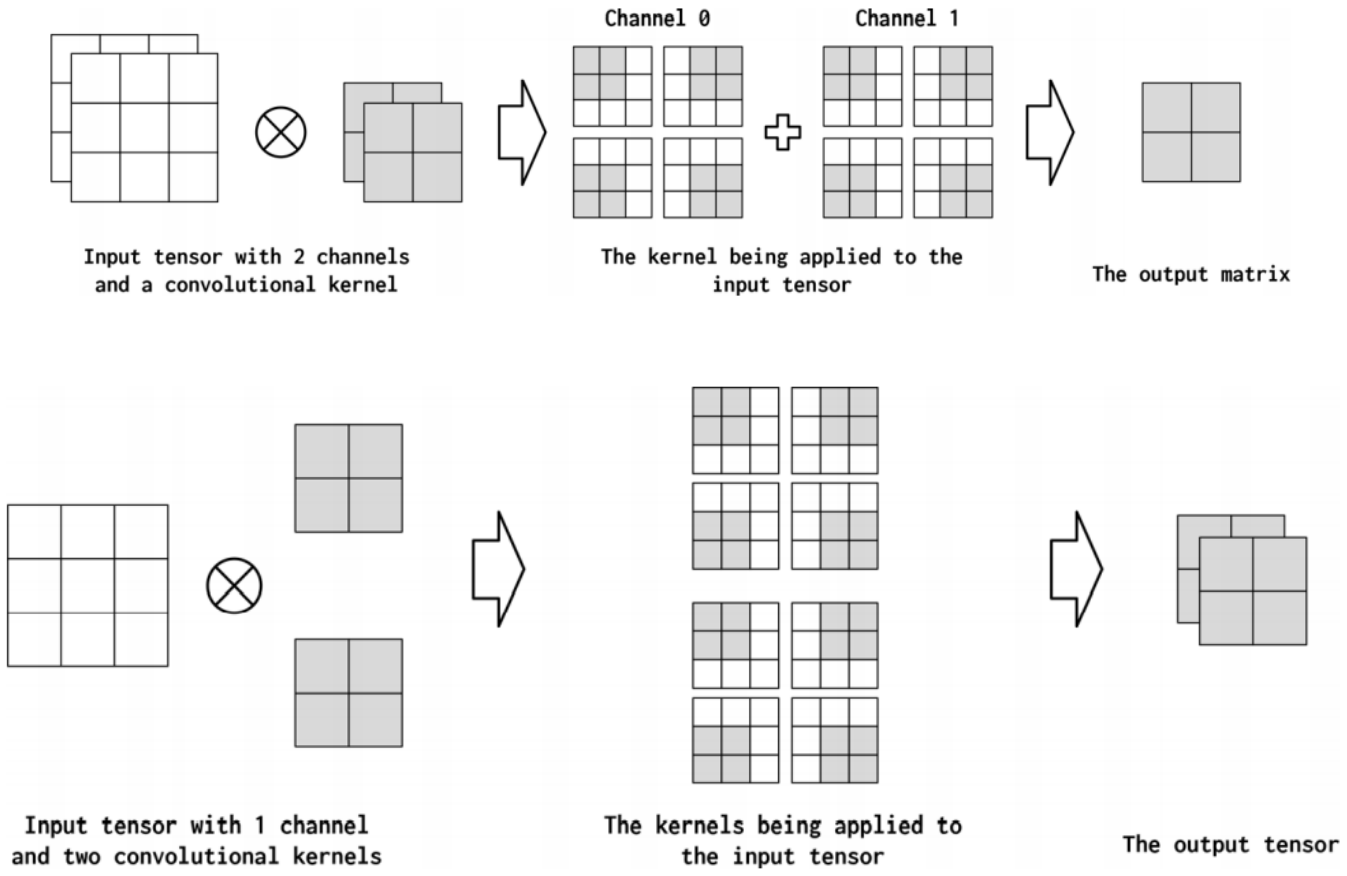
Stride : 1 step

Padding (Zero-padding) : 1 pixel

Convolutional Neural Network

CNN에서 사용되는 Hyperparameters

합성곱 **신경망**



Channel

Channel

입력 각 포인트에 있는 특성(Feature)의 차원. 이미지의 경우 각 픽셀은 RGB, 즉 3차원

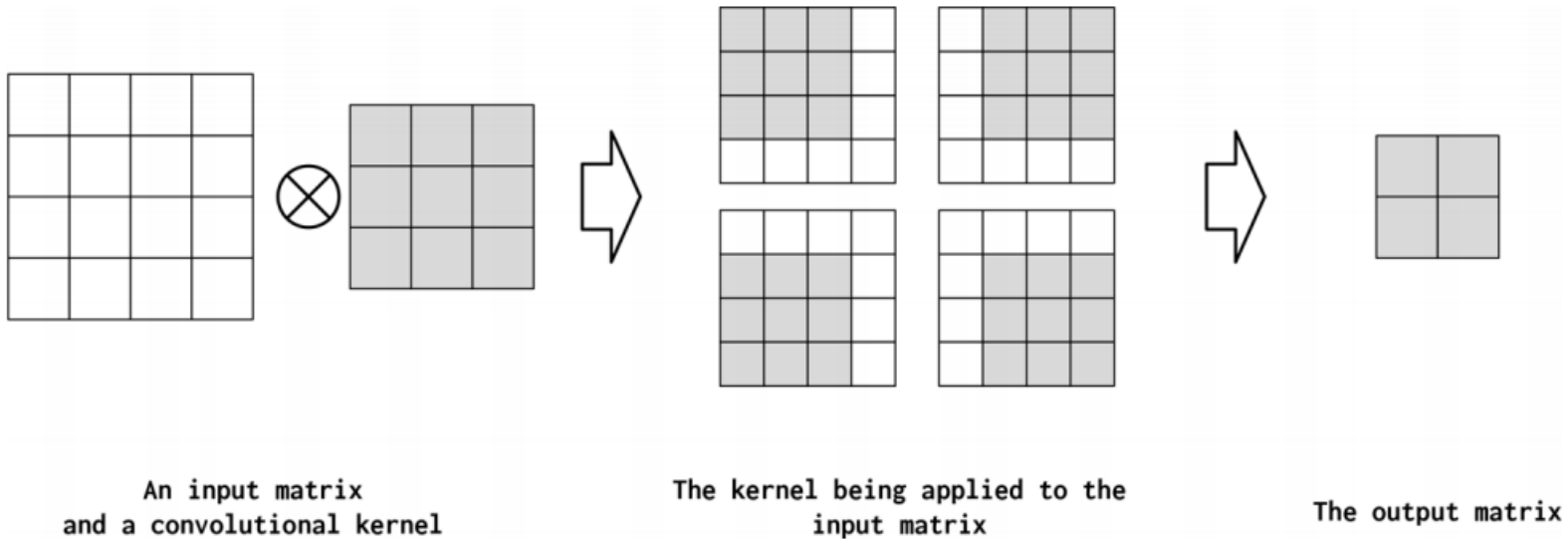
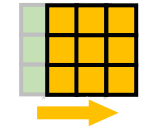


Figure 4-9. A convolution with kernel_size=3 is applied to the input matrix. The result is a trade-off: more local information is used for each application of the kernel to the matrix, but the output size is smaller.

Kernel Size

Convolution 연산에 사용되는 커널 행렬의 크기



Stride

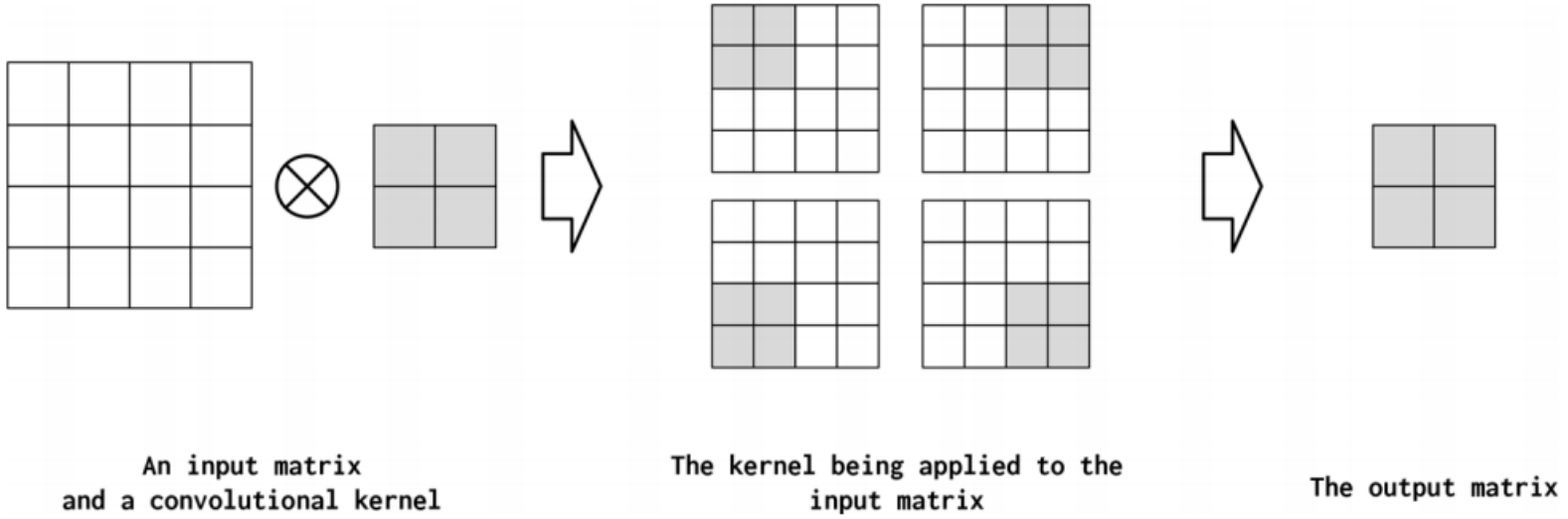
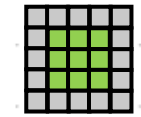


Figure 4-10. A convolutional kernel with kernel_size=2 applied to an input with the hyperparameter stride equal to 2. This has the effect that the kernel takes larger steps, resulting in a smaller output matrix. This is useful for subsampling the input matrix more sparsely.

Stride

Convolution 연산 간의 Step 크기



Padding

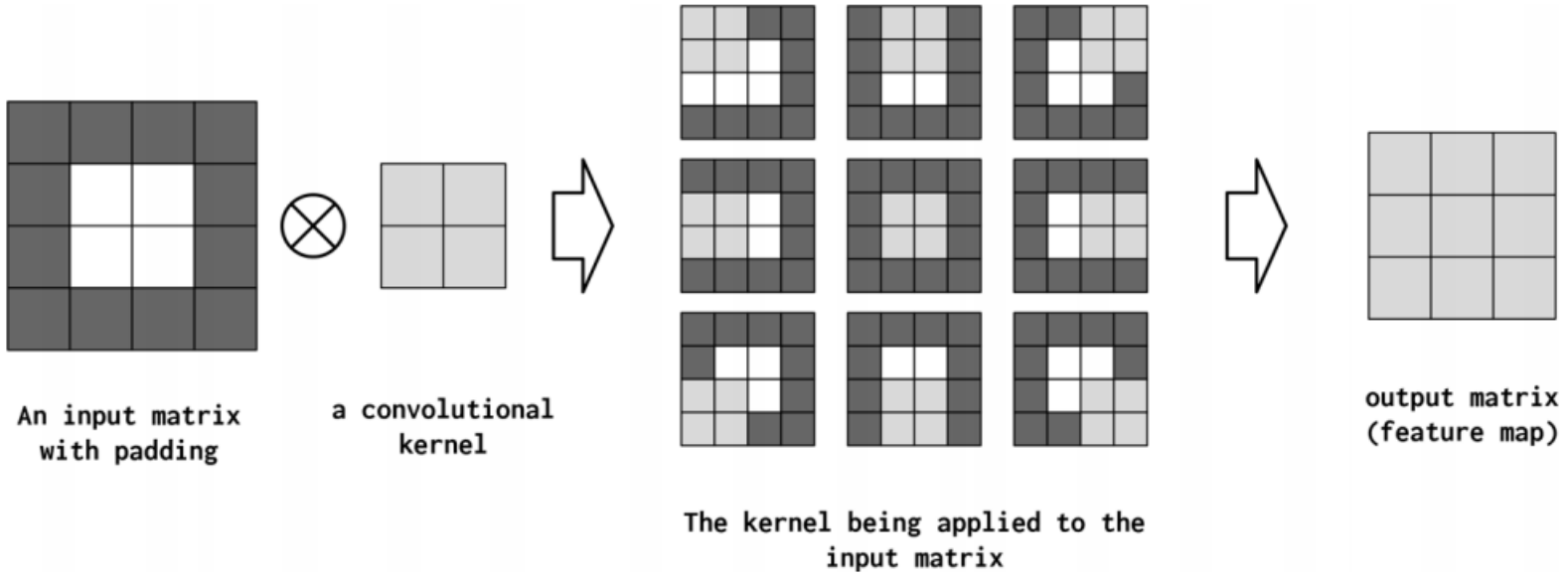
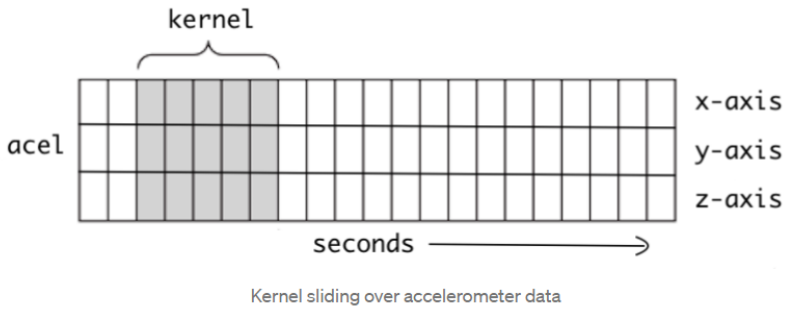


Figure 4-11. A convolution with kernel_size=2 applied to an input matrix that has height and width equal to 2. However, because of padding (indicated as dark-gray squares), the input matrix's height and width can be made larger. This is most commonly used with a kernel of size 3 so that the output matrix will exactly equal the size of the input matrix.

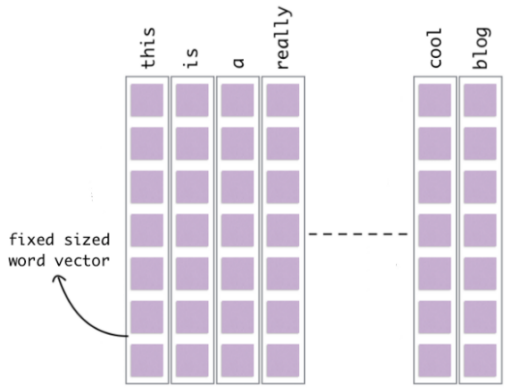
Padding

합성곱 결과 Output의 크기가 줄어드는 현상을 방지하기 위해 앞뒤로 값을 채움

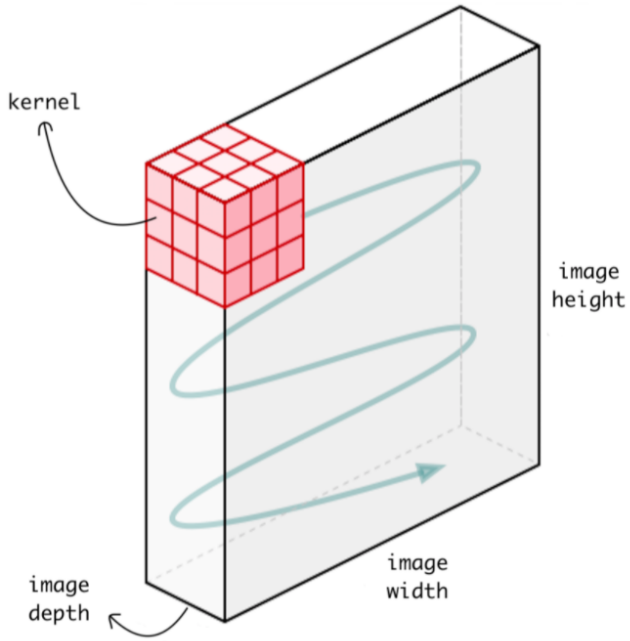
합성곱 **신경망**

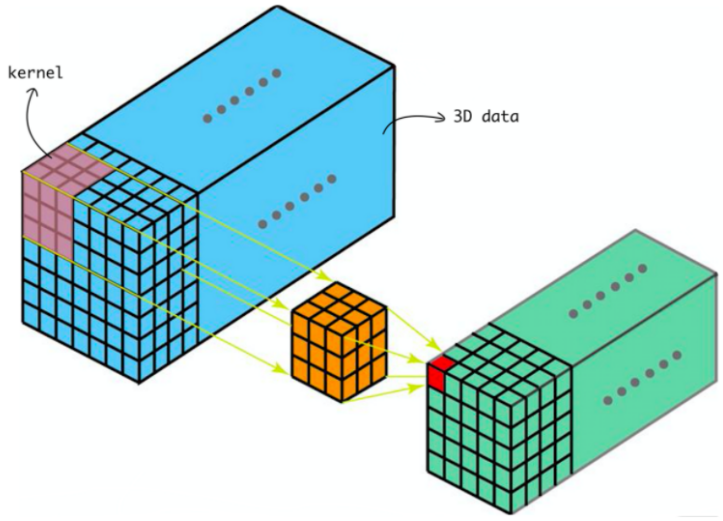


Kernel sliding over accelerometer data



Text data as Time Series





Kernel sliding on 3D data

Conv1d **Conv2d** **Conv3d**

Dimensions in CNN

Kernel의 이동 방향에 따라 차원을 결정. 보통 이미지는 2d, 텍스트는 1d

합성곱 **신경망**

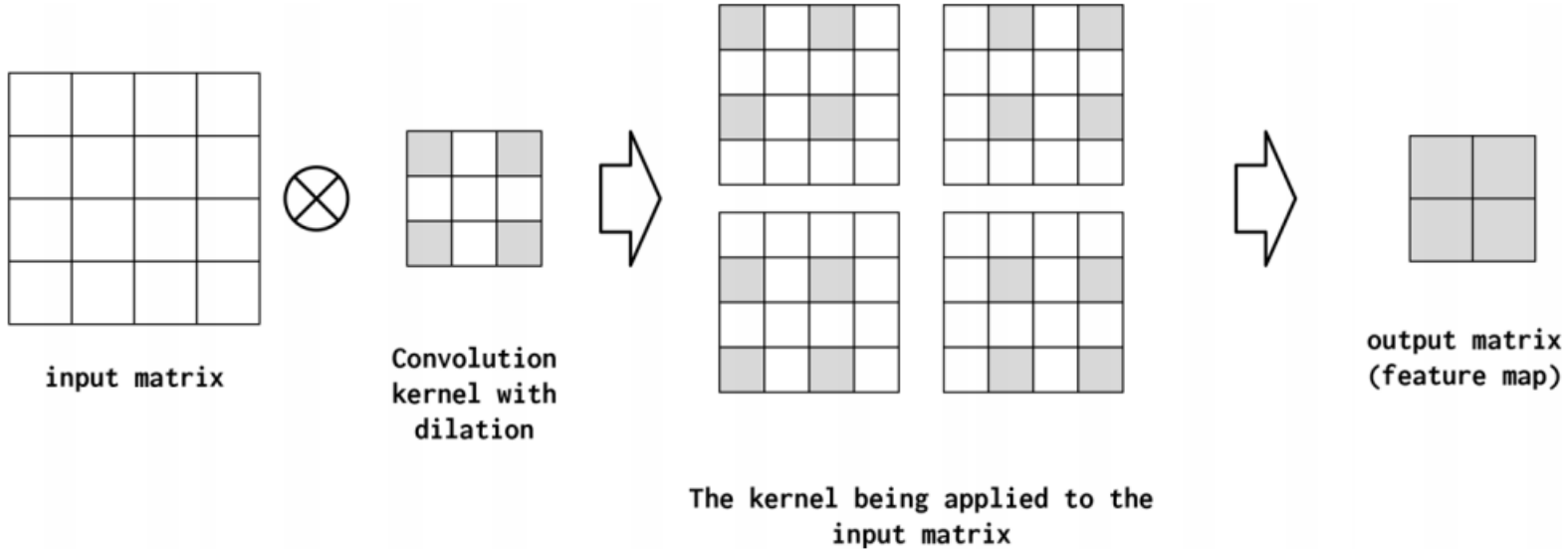


Figure 4-12. A convolution with kernel_size=2 applied to an input matrix with the hyperparameter dilation=2. The increase in dilation from its default value means the elements of the kernel matrix are spread further apart as they multiply the input matrix. Increasing dilation further would accentuate this spread.

Dilation

Kernel이 입력에 적용되는 방식을 결정 : 커널 행렬의 원소 사이의 Step

합성곱 **신경망**

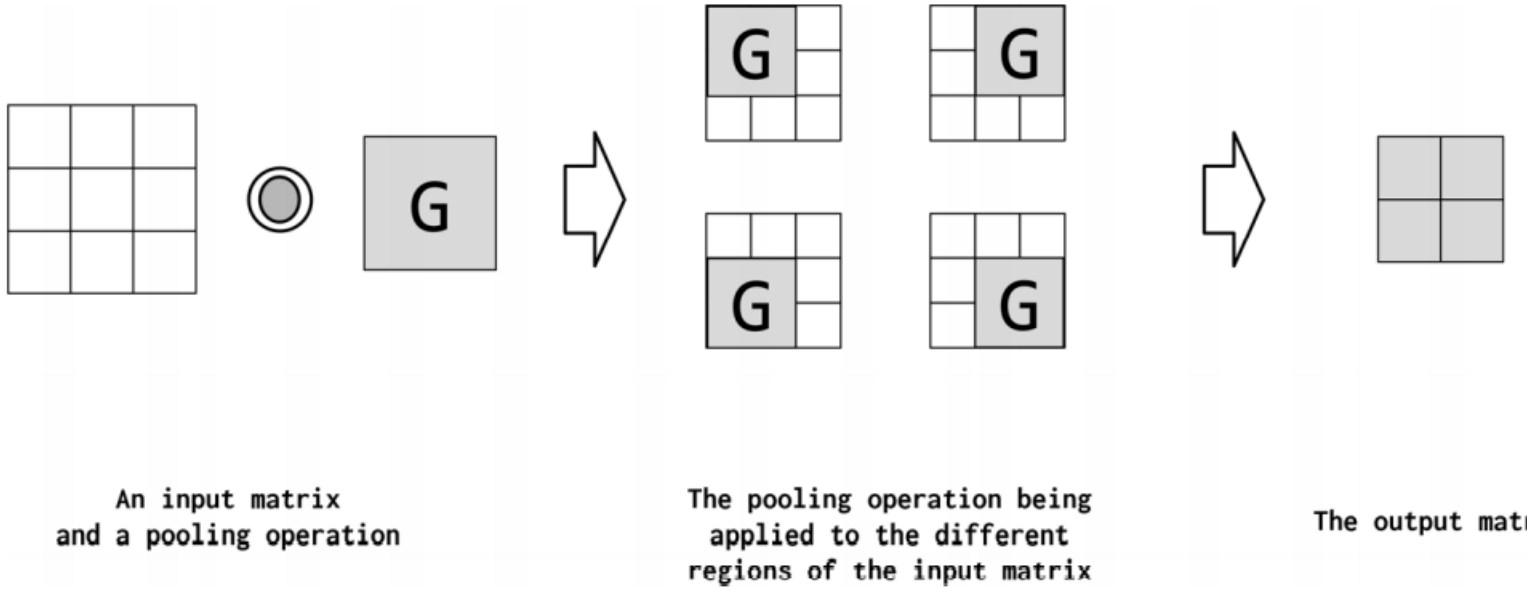


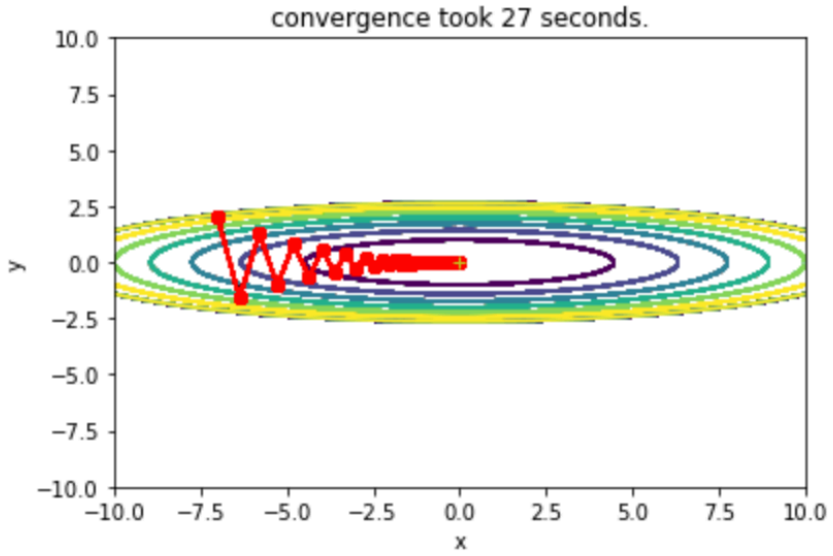
Figure 4-13. The pooling operation as shown here is functionally identical to a convolution: it is applied to different positions in the input matrix. However, rather than multiply and sum the values of the input matrix, the pooling operation applies some function G that pools the values. G can be any operation, but summing, finding the max, and computing the average are the most common.

Pooling

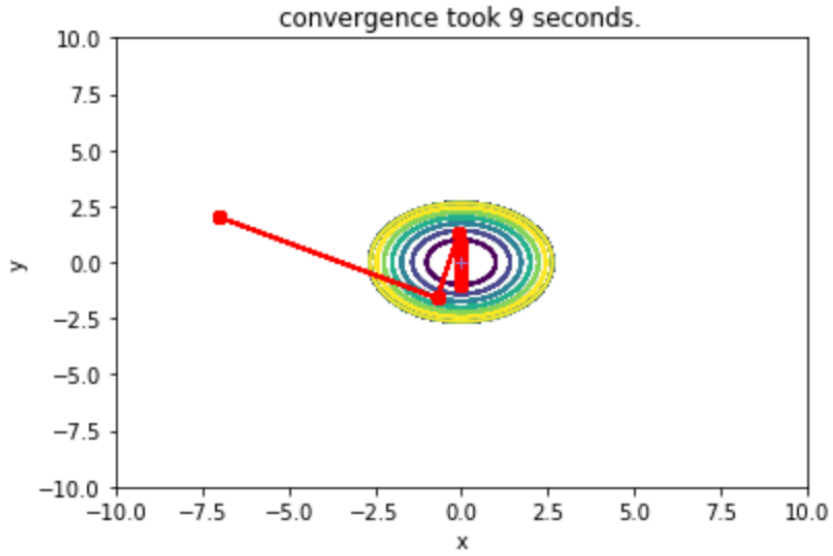
고차원 Feature Map을 저차원 Feature Map으로 요약하는 연산

합성곱 **신경망**

un-Normalized

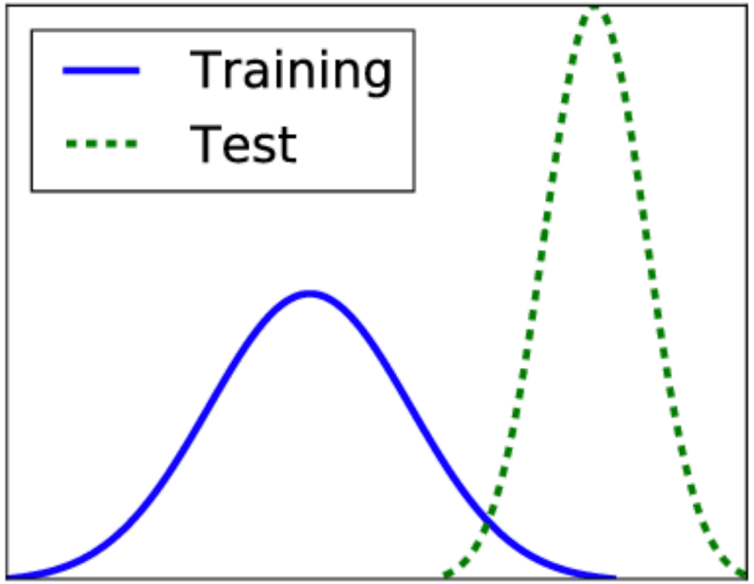


Normalized



Batch Normalization

활성화값의 평균이 0이고 단위분산이 되도록 만듦



Batch Normalization

실제 데이터가 훈련 데이터와 분포가 달라지는 문제를 해결

합성곱 **신경망**

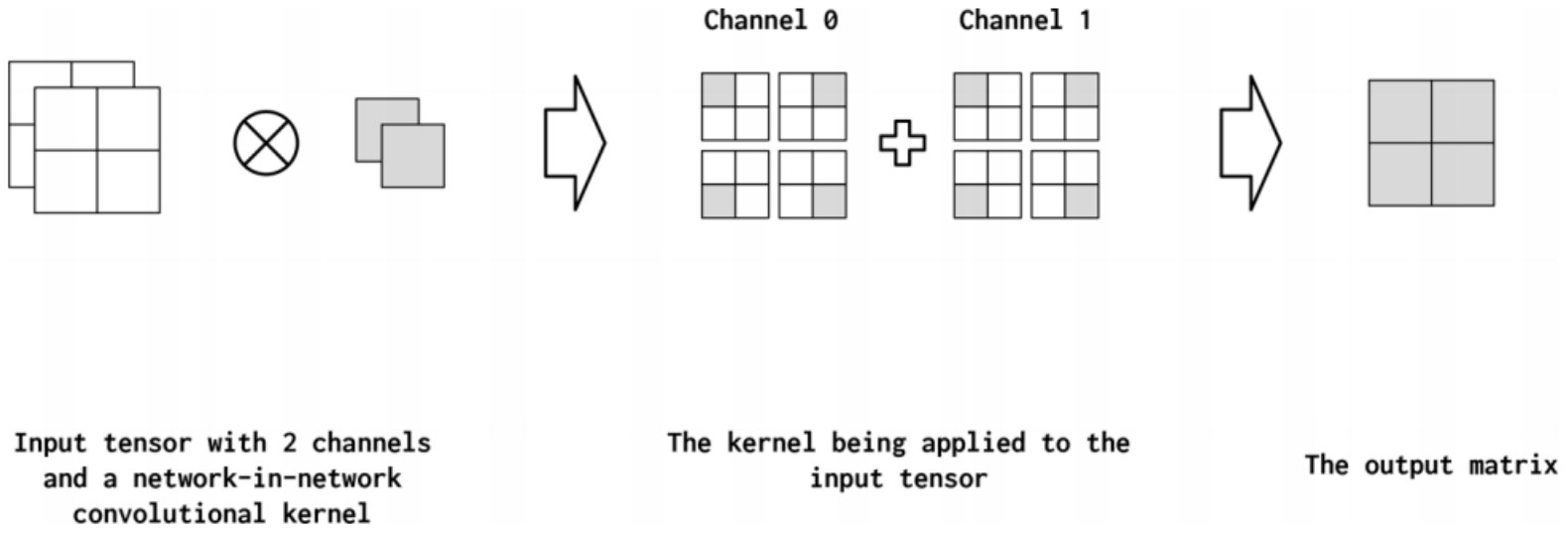
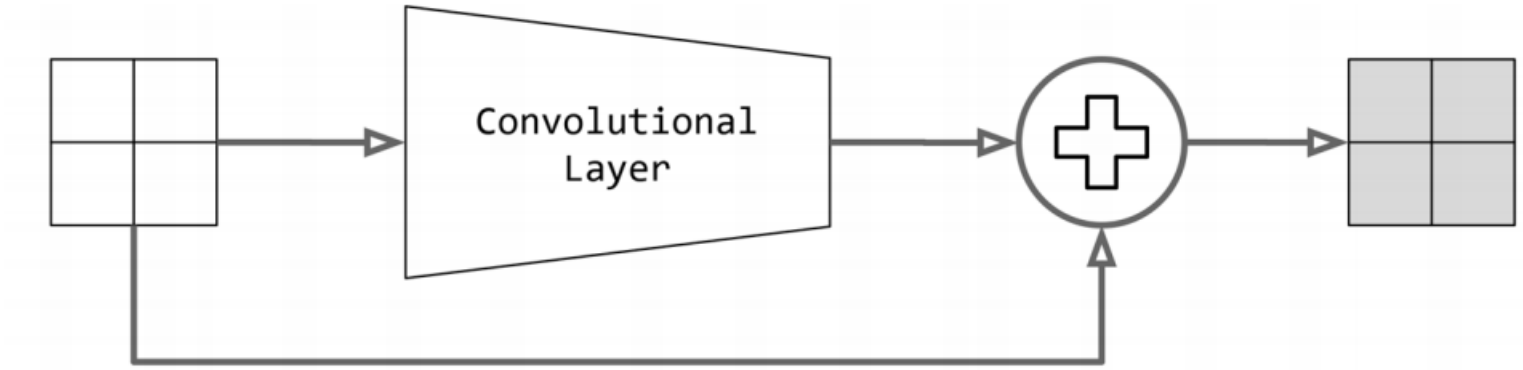


Figure 4-14. An example of a 1×1 convolution operation in action. Observe how the 1×1 convolution operation reduces the number of channels from two to one.

NiN Connection (1x1 Convolution)

입력되는 Feature Map의 Channel을 낮추어주기 위해 사용

합성곱 **신경망**



A convolutional layer is applied to the input matrix to create a matrix of the same size.

The input matrix is added to the output of the convolutional layer.

Residual Connection

원본 입력에 합성곱 결과를 더하여 Gradient를 건너띄어서 전달하고, 문제를 단순화시킴

CNN으로 **성씨 분류하기**

Notebook Link

Lets Code!

With Colab


Thank You