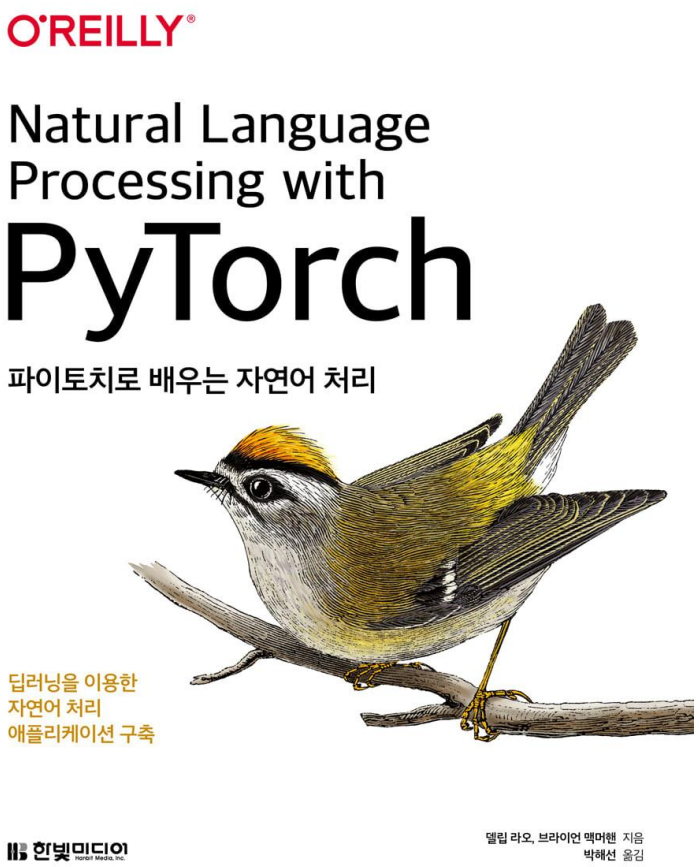
Data intelligence Lab

2022.7.6 ~ 7.13

Progress

인천대학교 컴퓨터 공학부 강병하

학습한 내용



4장 자연어 처리를 위한 피드 포워드 신경망

- 다층 퍼셉트론
- 합성곱 신경망

5장 단어와 타입 임베딩

- CBOW 임베딩 학습
- 사전 훈련된 임베딩을 사용한 전이 학습
  (뉴스 카테고리 분류)

다층 퍼셉트론(MLP)의 필요성

단층 퍼셉트론으로 XOR 게이트 문제 풀어보기

x_1	x_2	y
0	0	0
0	1	1
1	0	1
1	1	0

```
# 단층 퍼셉트론 구현
linear = nn.Linear(2, 1, bias=True)
sigmoid = nn.Sigmoid()
model = nn.Sequential(linear, sigmoid).to(device)
```

```
X = torch.FloatTensor([[0, 0], [0, 1], [1, 0], [1, 1]]).to(device)
Y = torch.FloatTensor([[0], [1], [1], [0]]).to(device)
```

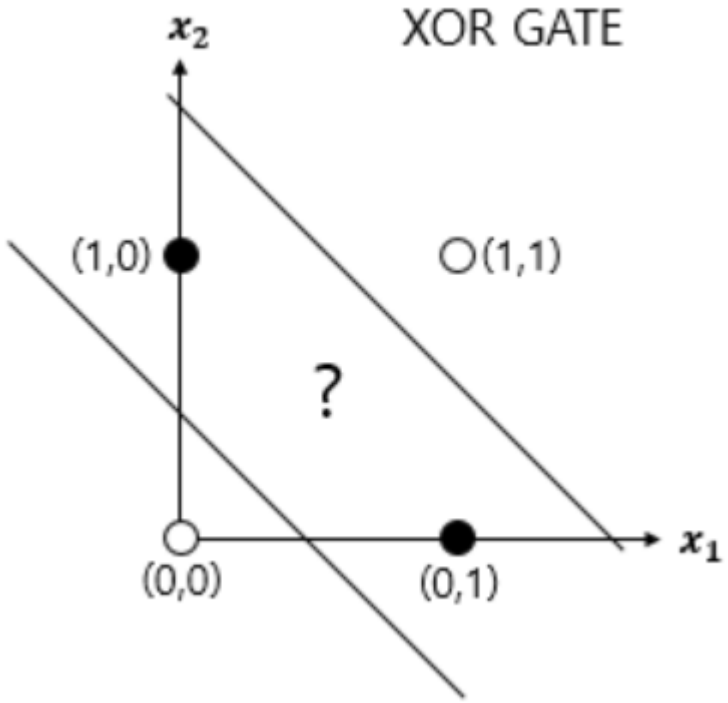
```
모델의 출력값(Hypothesis):  [[0.5]
 [0.5]
 [0.5]
 [0.5]]
모델의 예측값(Predicted):  [[0.]
 [0.]
 [0.]
 [0.]]
실제값(Y):  [[0.]
 [1.]
 [1.]
 [0.]]
정확도(Accuracy):  0.5
```

"단층 퍼셉트론(하나의 선형층)으로는 XOR 문제를 풀 수 없다." → ***Why?***

다층 퍼셉트론(MLP)의 필요성

단층 퍼셉트론으로 XOR 게이트 문제 풀어보기

x_1	x_2	y
0	0	0
0	1	1
1	0	1
1	1	0



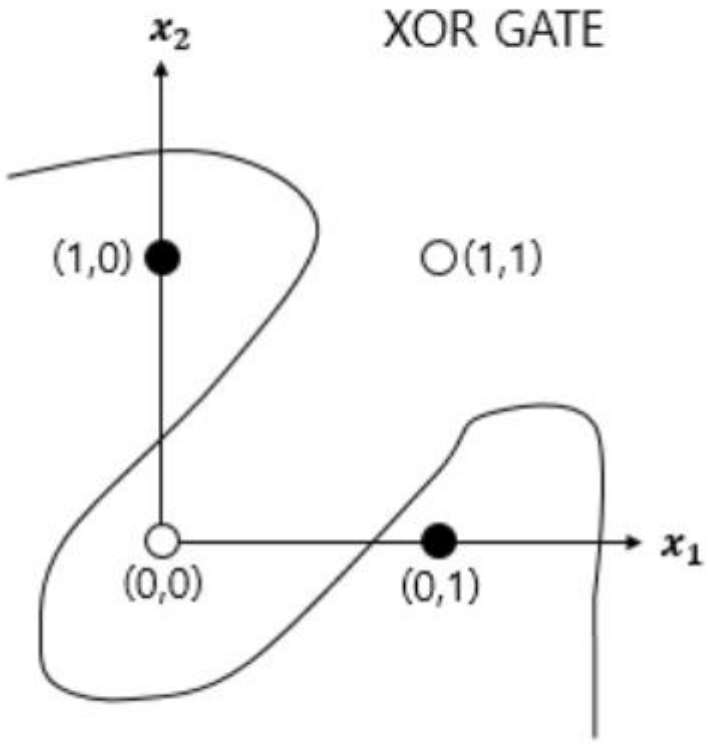
하얀 원과 검은 원을 직선 하나로 나누는 것(분류)은 불가능

→ 단층 퍼셉트론은 **선형 영역에 대해서만** 분리 가능

다층 퍼셉트론(MLP)의 필요성

단층 퍼셉트론으로 XOR 게이트 문제 풀어보기

x_1	x_2	y
0	0	0
0	1	1
1	0	1
1	1	0

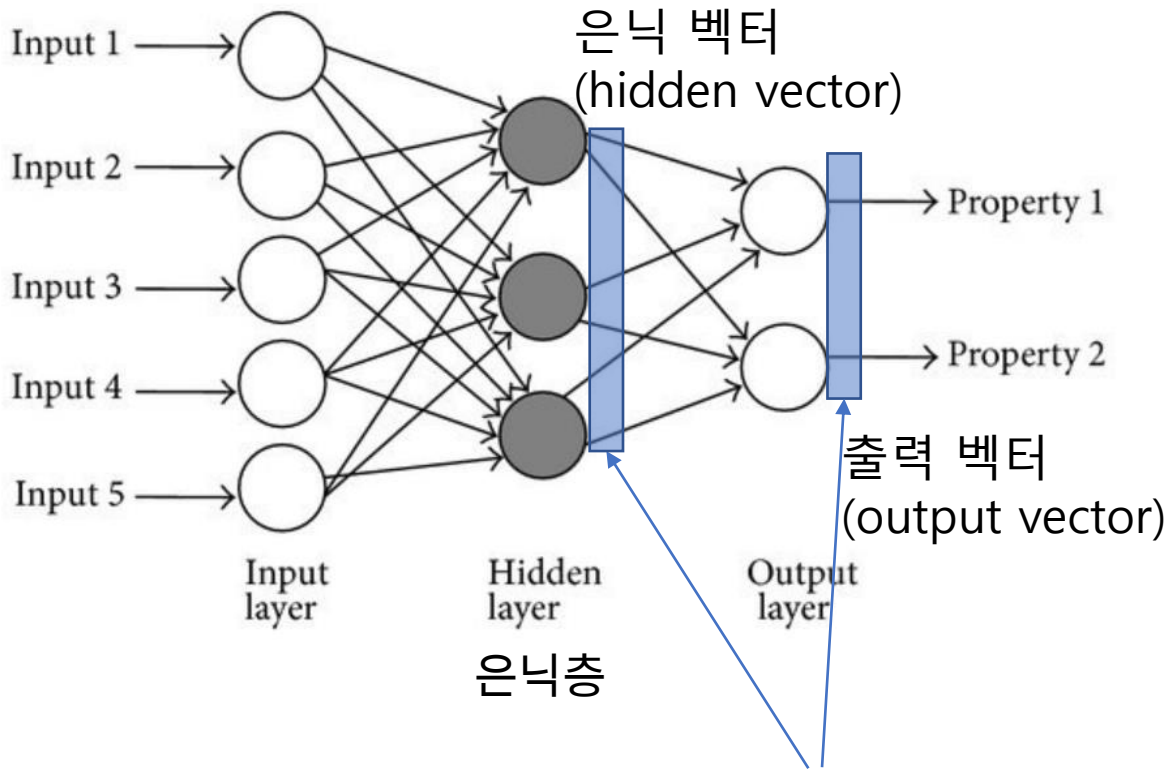


곡선(비선형성)을 사용하면 가능!

How? ➔ **다층 퍼셉트론**

다층 퍼셉트론(MLP)

: 많은 퍼셉트론이 있는 층을 여러 개 쌓아 올린 구조 (은닉층이 추가됨)



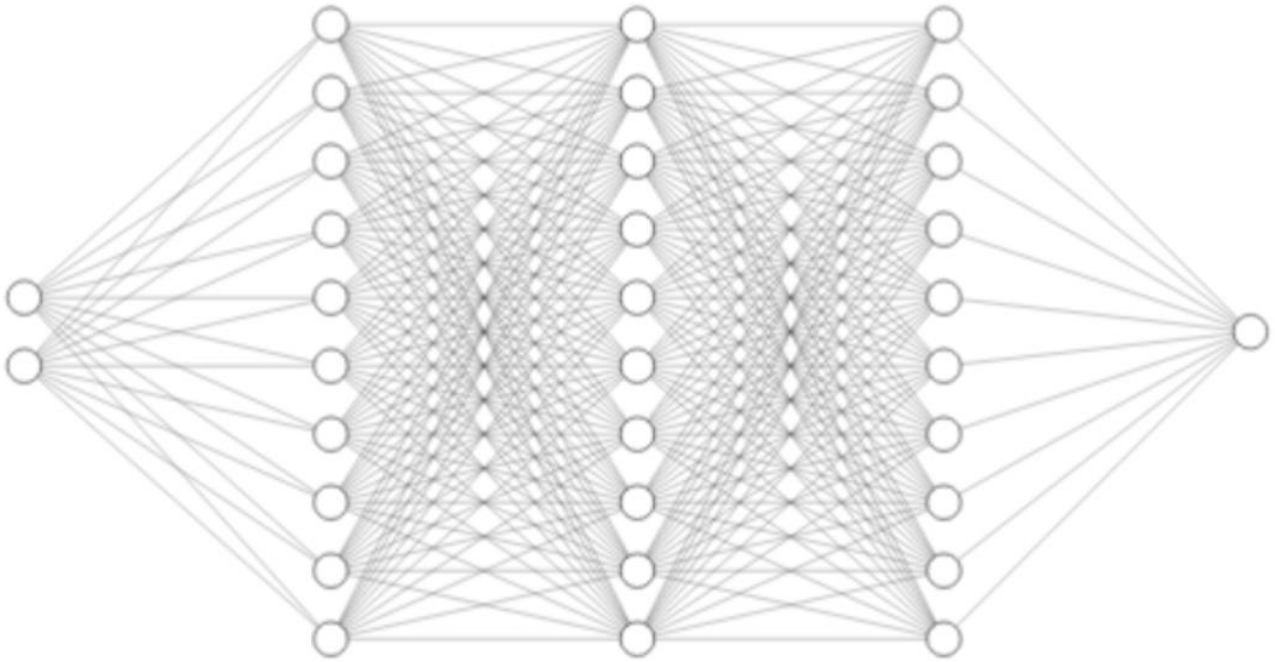
은닉 벡터
(hidden vector)

출력 벡터
(output vector)

은닉층

비선형(직선 1개로는 그릴 수 없는) 활성화 함수 적용

다층 퍼셉트론(MLP) – 심층 신경망(DNN)



은닉 층이 **2개 이상**인 신경망

→ **심층 신경망**(Deep Neural Network, DNN)

심층 신경망을 학습 시키기

→ **딥 러닝**(Deep Learning)

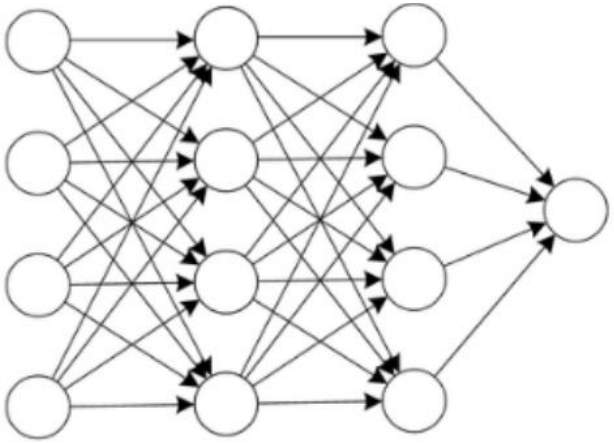
다층 퍼셉트론(MLP) in PyTorch

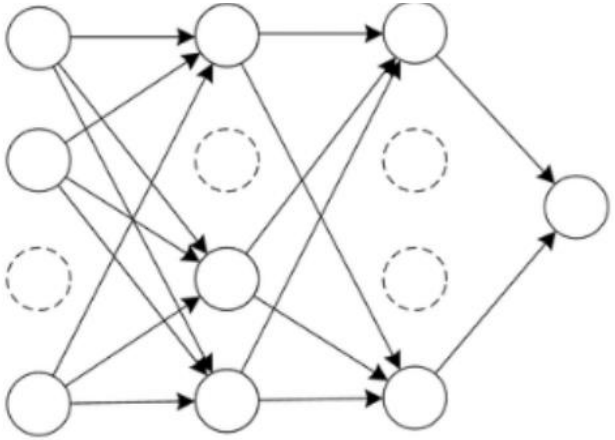
```
import torch.nn as nn
import torch.nn.functional as F

class MultilayerPerceptron(nn.Module):
  def __init__(self, input_dim, hidden_dim, output_dim):
    super(MultilayerPerceptron, self).__init__()
    self.fc1 = nn.Linear(input_dim, hidden_dim)   # hidden_dim : 출력 특성의 개수
    self.fc2 = nn.Linear(hidden_dim, output_dim)  # 한 층의 출력 개수는 다음 층의 입력 개수와 같아야 함

  def forward(self, x_in, apply_softmax = False):
    intermediate = F.relu(self.fc1(x_in))
    output = self.fc2(intermediate)

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

두 Linear 층 사이에는
필수적으로 비선형 활성화 함수 적용

다층 퍼셉트론(MLP)의 규제 : 드롭아웃(dropout)

인접한 두 층에 속하는 유닛 사이의 연결을 랜덤하게 끊는 것(훈련시에만) → 과대적합을 막음





랜덤하게 유닛을 끊으면 어떤 유닛도 다른 유닛에 전적으로 의존하지 못함 → **모델 안정**

매번 랜덤하게 선택된 유닛 → 서로 다른 신경망을 앙상블하여 사용하는 것 같은 효과

"술에 취한 상태에서 어떤 작업을 반복 수행하는 방법을 배울 수 있다면 술이 깼을 때는 더 잘할 수 있어야 한다"

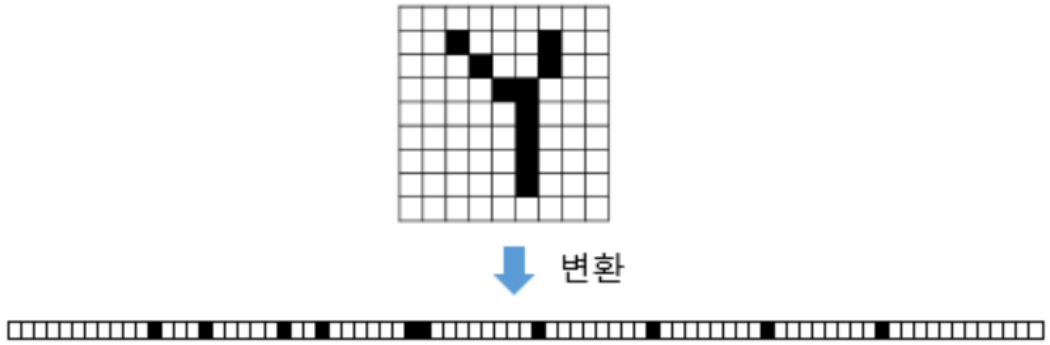
-Stephen Merity

NLP에서 다층 퍼셉트론(MLP)의 한계

MLP는 텍스트의 **순차 패턴**을 감지하기에 적합하지 않음

{나는, 밥을 , 먹었다 , 굶었다}

밥을 먹었다 = 먹었다 밥을 = { 0, 1, 1, 0 }



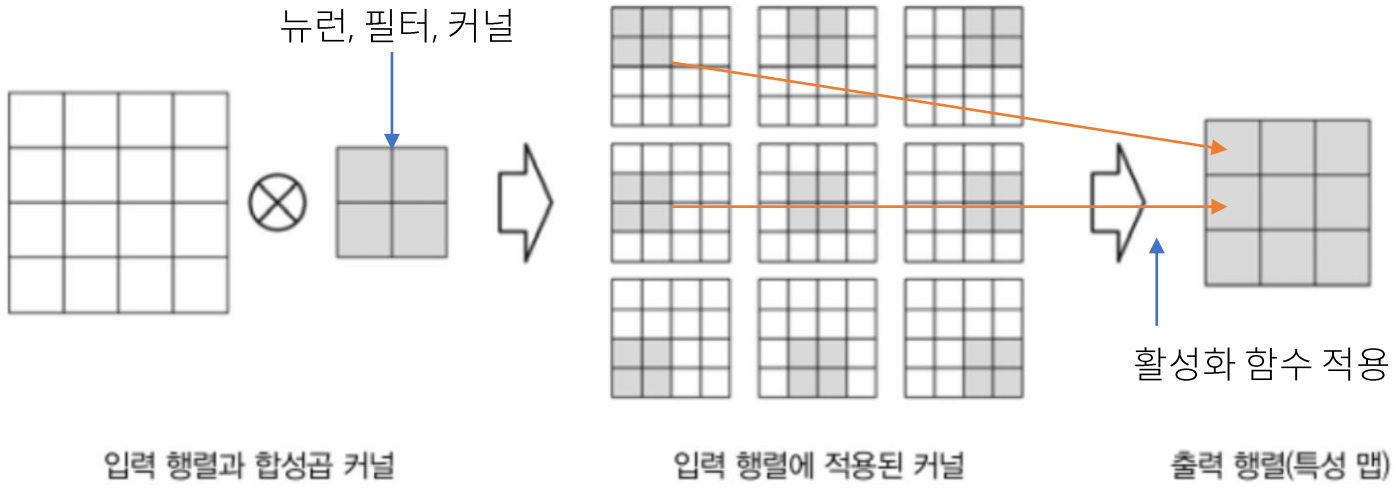
공간적 구조(순서)정보 유실

어떻게 하면 단어의 순서(or 문자의 순서) 패턴을 학습시킬 수 있을까? ➔ CNN, RNN

합성곱 신경망(CNN)

공간상의 부분 구조를 감지하는 데 적합한 합성곱 신경망 CNN(Convolutional Neural Networks)

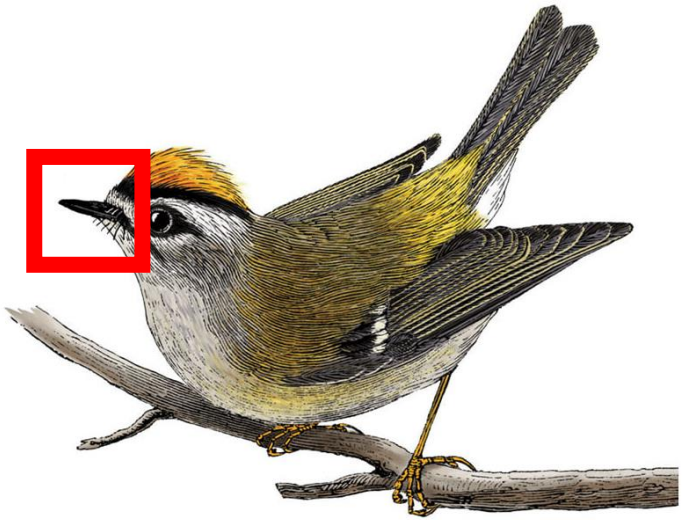
→ 이미지, 영상 데이터 처리

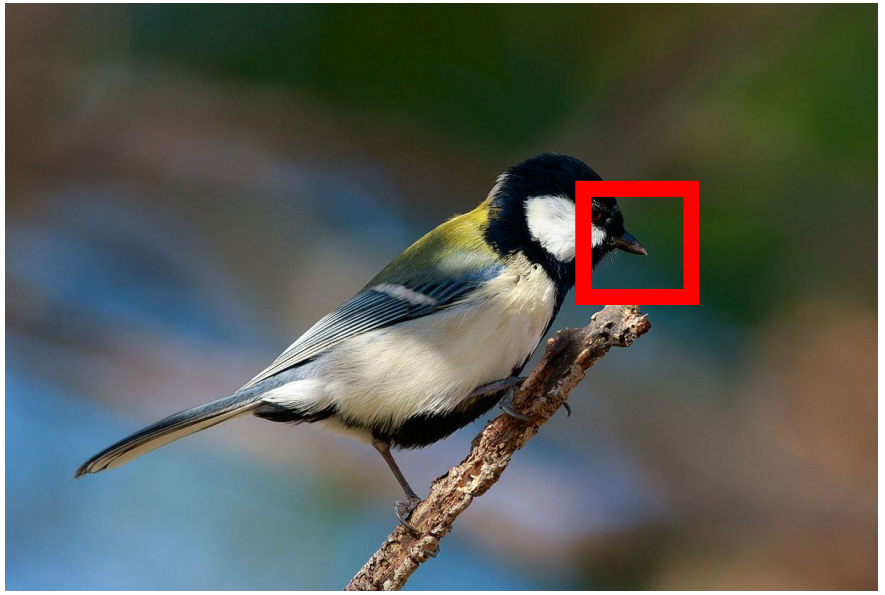


입력 데이터에 도장을 여러 번 찍어 (Σ 각 입력x가중치) 유용한 특성만 골라 내기(=압축)

합성곱 신경망(CNN)

“이것은 새 입니까?”





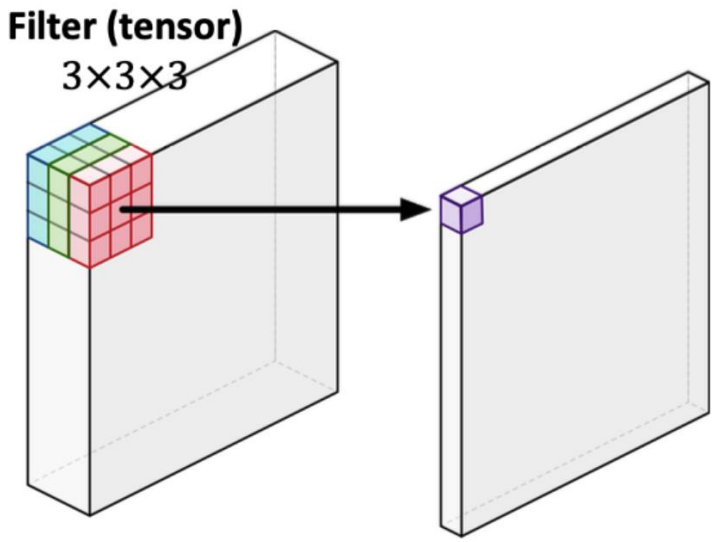
전체가 아닌 **부분 구조(패턴)**을 감지하여 판단.

합성곱 신경망(CNN) - 채널

입력 각 포인트에 있는 특성 차원

흑백 이미지 -> 채널 수 1 (밝기 0~255)

컬러 이미지 -> 채널 수 3 (R,G,B)



텍스트 데이터 -> 채널 수 = 어휘 사전(Vocabulary)의 크기

합성곱 신경망(CNN) – 커널 크기

도장의 크기 (커널, 필터 행렬의 크기) → 얻어지는 **정보의 양**

커널 크기가 작을수록 작고 자주 등장하는 패턴 감지

커널 크기가 클수록 (아주 가끔 등장하는 패턴) 큰 패턴 감지

```
text = ['You', 'can', 'always', 'recognize', 'truth', 'by', 'its', 'beauty', 'and', 'simplicity', '.']
n = 3
for i in range(len(text)-n+1):
  print(text[i:i+3])
```

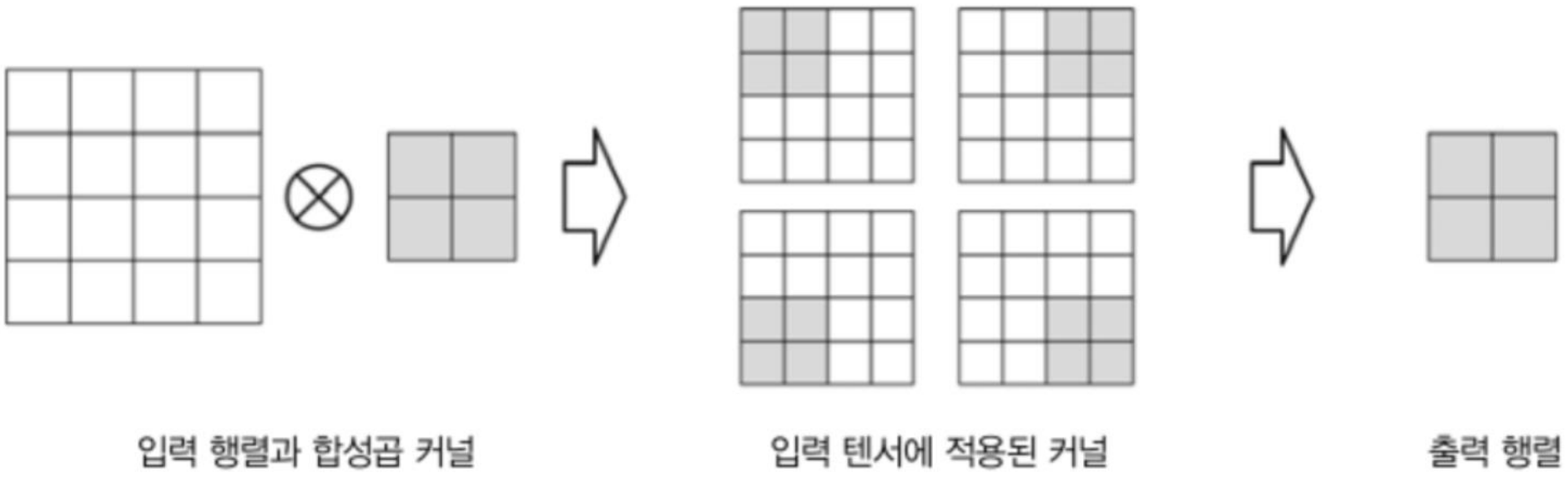
```
['You', 'can', 'always']
['can', 'always', 'recognize']
['always', 'recognize', 'truth']
['recognize', 'truth', 'by']
['truth', 'by', 'its']
['by', 'its', 'beauty']
['its', 'beauty', 'and']
['beauty', 'and', 'simplicity']
['and', 'simplicity', '.']
```

텍스트 데이터 -> 커널 크기 = n-그램의 크기

합성곱 신경망(CNN) – 스트라이드(stride)

도장을 어느 정도 촘촘하게(or 듬성듬성하게) 찍을 지

스트라이드 값이 클수록 듬성듬성 샘플링 ➔ 출력 행렬 크기는 작아짐

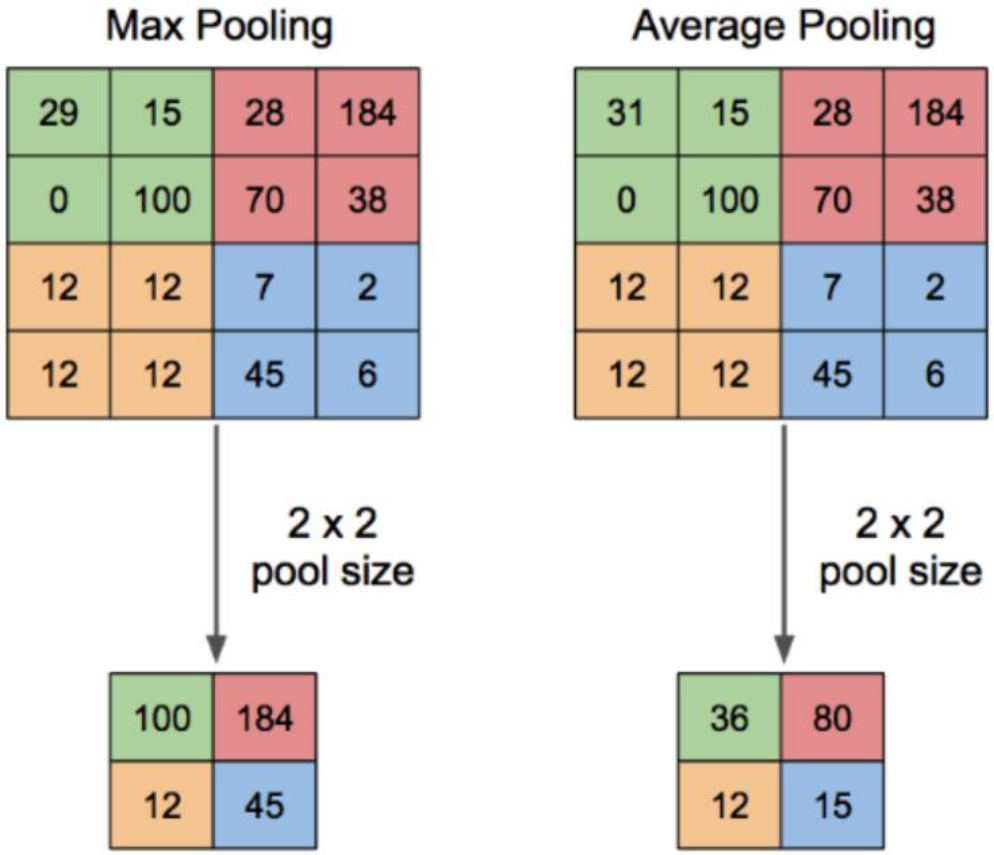


입력 행렬과 합성곱 커널 | 입력 텐서에 적용된 커널 | 출력 행렬

스트라이드 = 2

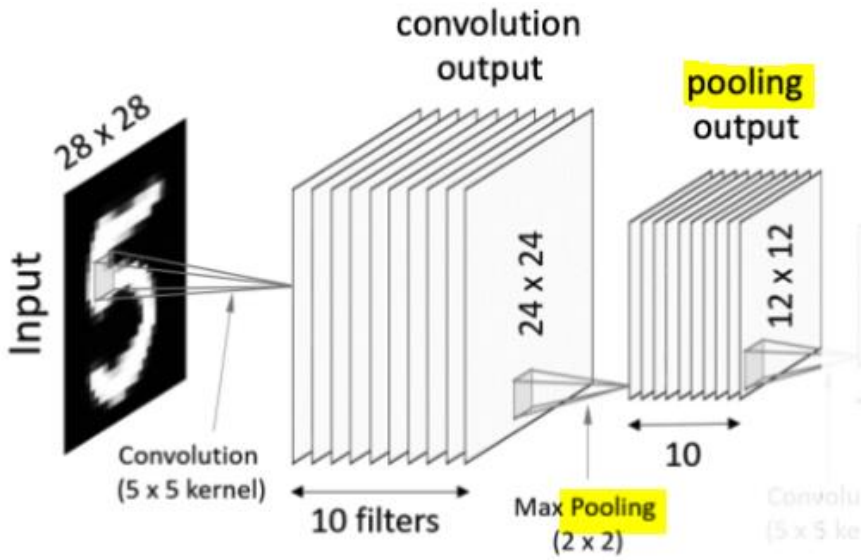
합성곱 신경망(CNN) – 풀링(pooling)

고차원 (합성곱의 출력)특성 맵을 저차원 특성 맵으로 요약 (다운 샘플링)

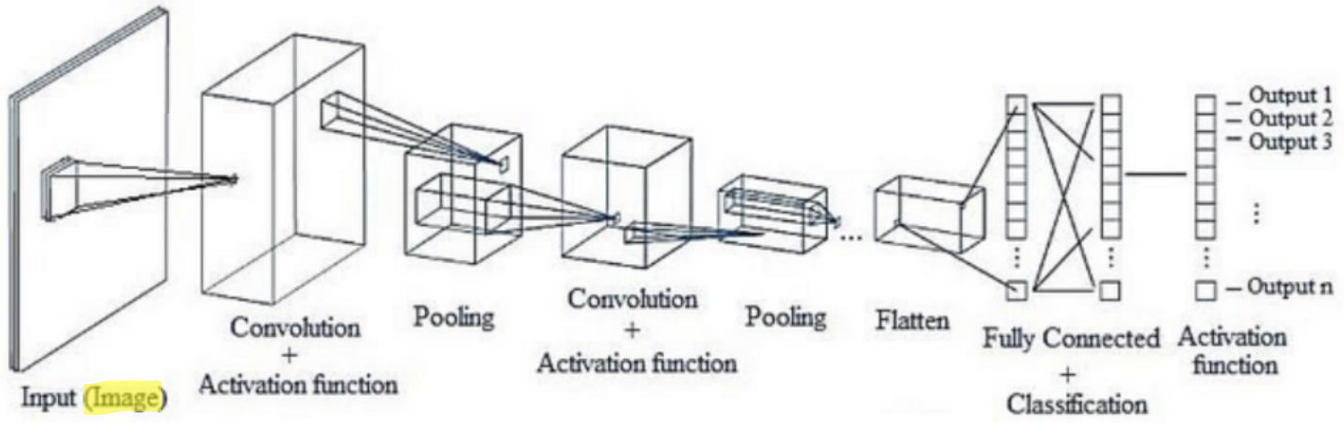


합성곱 연산의 중첩되는 특징 때문에 많은 특성 중복 가능성

➔ 중복 가능성이 높은 특성맵을 저차원으로 요약

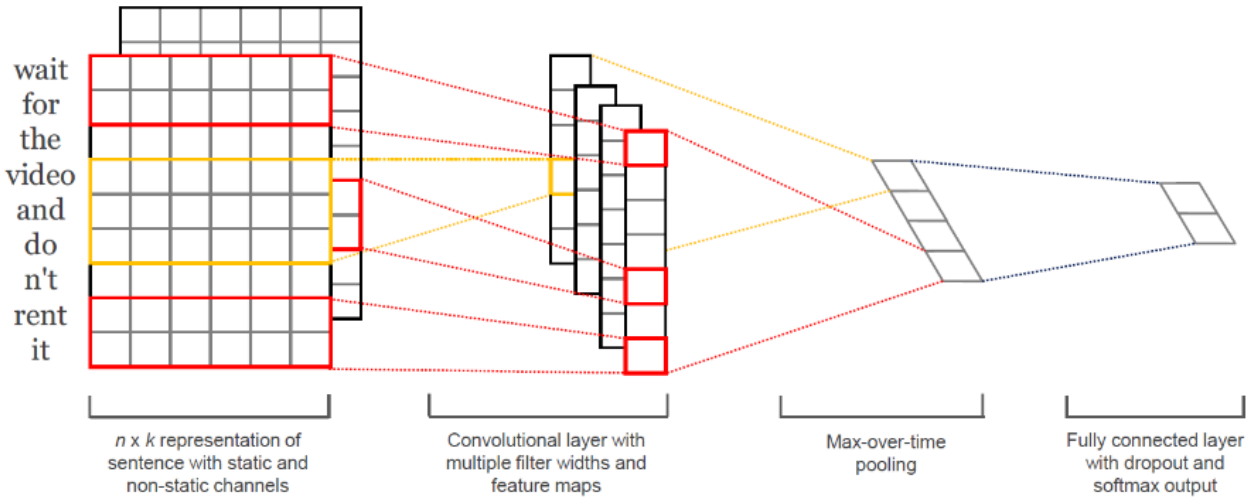


합성곱 신경망(CNN)의 전체 구조



임베딩된 p차원의 단어 벡터

단어	V1	V2	V3	V4	...	Vp-2	Vp-1	Vp
W1								
W2								
W3								
...								
Wn-2								
Wn-1								
Wn								



필터(커널), 바이그램(n-그램의 n=2)

| 단어와 타입 임베딩 : 단어 임베딩(word embedding)

NLP 작업은 여러 종류의 **이산적인 타입(type)**을 다뤄야 함.

단어, 문자, 품사 태깅, 개체명 등

이산 타입(단어) ➔ 밀집 벡터 '표현 학습', '임베딩' : 이산 타입과 벡터 공간 포인트 사이의 매핑 학습

```
dog   -1.242  -0.360  0.573  0.367  0.600  -0.189  1.273 ...
cat   -0.964  -0.610  0.674  0.351  0.413  -0.212  1.380 ...
```

단어 임베딩(word embedding)

단어 임베딩은 어떤 NLP 작업에서도 성능 향상을 기대할 수 있다.

➔ NLP의 '스리라차'

단어 임베딩 학습 방법

```
dog   -1.242  -0.360  0.573  0.367  0.600  -0.189  1.273 ...
cat   -0.964  -0.610  0.674  0.351  0.413  -0.212  1.380 ...
```

ㄴ임베딩 벡터(embedding vector)는 어떻게 만들어지는 것인가?

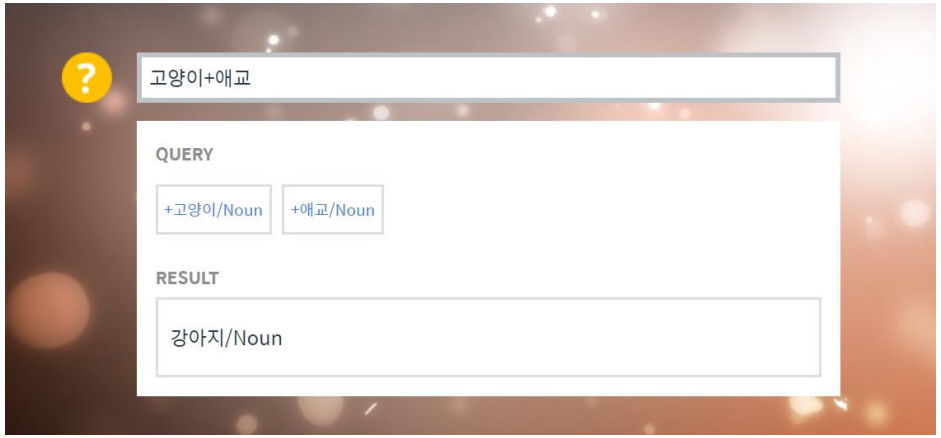
1. 단어를 랜덤한 값을 가지는 밀집 벡터로 변환
2. 보조 작업 지도학습 → **가중치 변형하듯 단어 밀집 벡터도 변환**
   - 단어 시퀀스에서 다음 단어 예측
   - 앞과 뒤의 단어 시퀀스 사이의 누락된 단어 예측
   - 단어가 주어지면 위치에 관계 없이 window안에 등장할 단어 예측 등..

통계적, 언어적 속성 감지 → 규칙적으로 나타나는 구문과 의미 관계 인코딩

Word2Vec

Word2Vec

단어간 유사도를 반영할 수 있도록 단어의 의미를 벡터화



http://w.elnn.kr/search/

한국어 단어에 대해 벡터 연산 가능

각 단어 벡터 = 단어간 유사도를 반영한 값

두 가지 방식

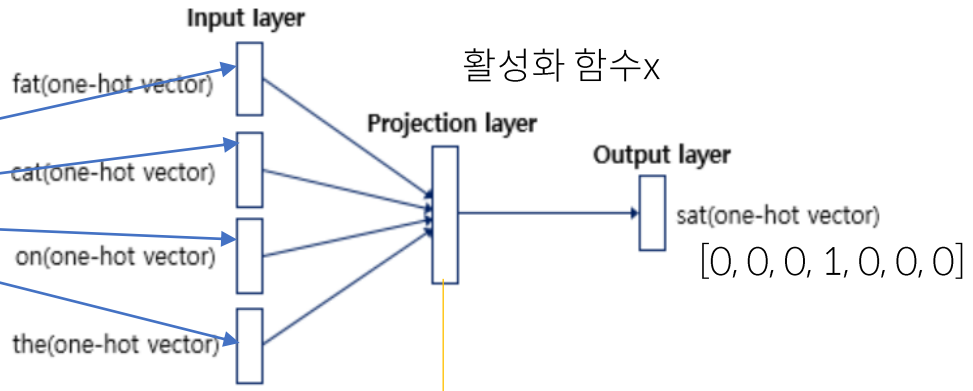
- CBOW(Continuous Bag of Words) → 주변 단어를 가지고 중간에 있는 단어 예측
- Skip-Gram → 중간에 있는 단어로 주변 단어 예측

Word2Vec : CBOW

CBOW(Continuous Bag of Words) → 주변에 있는 단어들로 중간에 있는 단어 예측

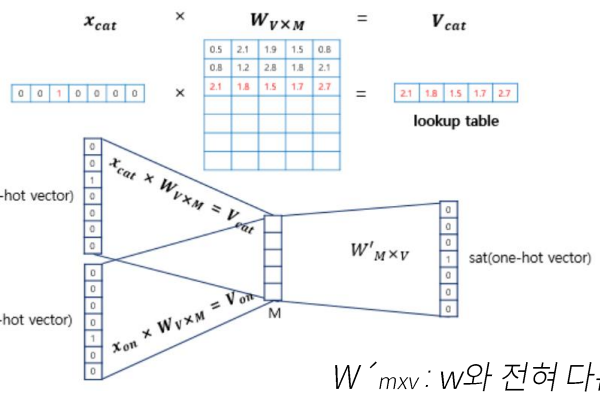
중심 단어 / 주변 단어

중심 단어	주변 단어
The fat cat sat on the mat	[1, 0, 0, 0, 0, 0]	[0, 1, 0, 0, 0, 0], [0, 0, 1, 0, 0, 0]
The fat cat sat on the mat	[0, 1, 0, 0, 0, 0]	[1, 0, 0, 0, 0, 0], [0, 0, 1, 0, 0, 0], [0, 0, 0, 1, 0, 0]
The fat cat sat on the mat	[0, 0, 1, 0, 0, 0]	[1, 0, 0, 0, 0, 0], [0, 1, 0, 0, 0, 0], [0, 0, 0, 1, 0, 0], [0, 0, 0, 0, 1, 0]
The fat cat sat on the mat	[0, 0, 0, 1, 0, 0]	[0, 1, 0, 0, 0, 0], [0, 0, 1, 0, 0, 0], [0, 0, 0, 0, 1, 0], [0, 0, 0, 0, 0, 1]
The fat cat sat on the mat	[0, 0, 0, 0, 1, 0]	[0, 0, 0, 1, 0, 0], [0, 0, 0, 0, 1, 0], [0, 0, 0, 0, 0, 1]
The fat cat sat on the mat	[0, 0, 0, 0, 0, 1]	[0, 0, 0, 0, 1, 0], [0, 0, 0, 0, 0, 1]



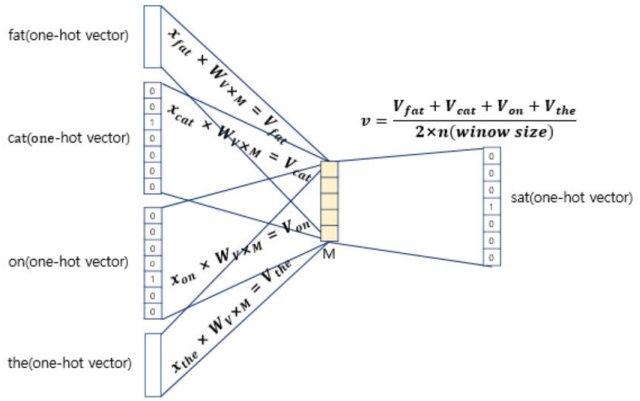
활성화 함수x

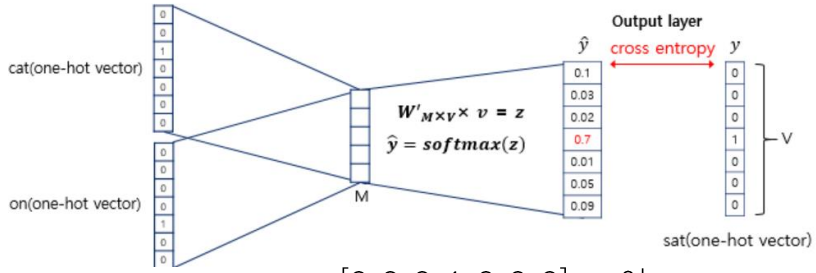
Lookup table 연산



W´mxv : w와 전혀 다른 가중치 행렬

Wvxm(가중치) : 초기에는 매우 작은 랜덤값 → 반복 훈련하여 수정



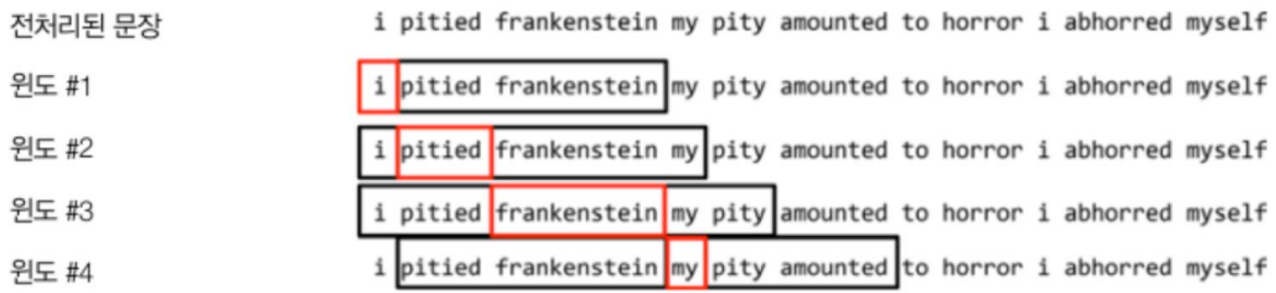


[0, 0, 0, 1, 0, 0, 0] sat이 중심 단어일 확률 0.7

→ W와 W´중 어떤 것을 임베딩 벡터로 사용할지 결정 (or 둘의 평균치)

| Word2Vec : CBOW 학습하기 - 데이터셋

Dataset : 메리 셸리의 소설 <프랑켄슈타인>



각 문장의 토큰 리스트를 순회하며 지정된 크기로 양쪽 묶기 -> 중심 단어는 target

```
cbow_data.head()
```

	context	target	split
0	, or the modern prometheus	frankenstein	train
1	frankenstein or the modern prometheus by	,	train
2	frankenstein , the modern prometheus by mary	or	train
3	frankenstein , or modern prometheus by mary wo...	the	train
4	frankenstein , or the prometheus by mary wolls...	modern	train

| Word2Vec : CBOW 학습하기 - 모델

```python
class CBOWClassifier(nn.Module): # Simplified cbow Model
    def __init__(self, vocabulary_size, embedding_size, padding_idx=0):
        """
        매개변수:
            vocabulary_size (int): 어휘 사전 크기, 임베딩 개수와 예측 벡터 크기를 결정합니다
            embedding_size (int): 임베딩 크기
            padding_idx (int): 기본값 0; 임베딩은 이 인덱스를 사용하지 않습니다
        """
        super(CBOWClassifier, self).__init__()

        self.embedding = nn.Embedding(num_embeddings=vocabulary_size,
                                      embedding_dim=embedding_size,
                                      padding_idx=padding_idx)
        self.fc1 = nn.Linear(in_features=embedding_size,
                             out_features=vocabulary_size)

    def forward(self, x_in, apply_softmax=False):
        """분류기의 정방향 계산

        매개변수:
            x_in (torch.Tensor): 입력 데이터 텐서
                x_in.shape는 (batch, input_dim)입니다.
            apply_softmax (bool): 소프트맥스 활성화 함수를 위한 플래그
                크로스-엔트로피 손실을 사용하려면 False로 지정합니다
        반환값:
            결과 텐서. tensor.shape은 (batch, output_dim)입니다.
        """
        x_embedded_sum = F.dropout(self.embedding(x_in).sum(dim=1), 0.3)
        y_out = self.fc1(x_embedded_sum)

        if apply_softmax:
            y_out = F.softmax(y_out, dim=1)

        return y_out
```

| Word2Vec : CBOW 학습하기 - 평가

```
# 가장 좋은 모델을 사용해 테스트 세트의 손실과 정확도를 계산합니다
classifier.load_state_dict(torch.load(train_state['model_filename']))
classifier = classifier.to(args.device)
loss_func = nn.CrossEntropyLoss()

dataset.set_split('test')
batch_generator = generate_batches(dataset,
                                   batch_size=args.batch_size,
                                   device=args.device)
running_loss = 0.
running_acc = 0.
classifier.eval()

for batch_index, batch_dict in enumerate(batch_generator):
    # 출력을 계산합니다
    y_pred =  classifier(x_in=batch_dict['x_data'])

    # 손실을 계산합니다
    loss = loss_func(y_pred, batch_dict['y_target'])
    loss_t = loss.item()
    running_loss += (loss_t - running_loss) / (batch_index + 1)

    # 정확도를 계산합니다
    acc_t = compute_accuracy(y_pred, batch_dict['y_target'])
    running_acc += (acc_t - running_acc) / (batch_index + 1)

train_state['test_loss'] = running_loss
train_state['test_acc'] = running_acc
```

```
print("테스트 손실: {};".format(train_state['test_loss']))
print("테스트 정확도: {}".format(train_state['test_acc']))
```

```
테스트 손실: 8.193071616677674;
테스트 정확도: 11.411764705882353
```

Why? 약 7만개의 단어로는 규칙성을 감지하기에 충분하지 않음

➔ 일반적으로 *사전 훈련된 임베딩*을 미세 조정하여 사용

사전 훈련된 임베딩을 사용한 전이 학습 (문서분류)

전이 학습(transfer learning)

한 작업에서 훈련된 모델을 다른 작업의 초기 모델로 사용하는 방식

사전 훈련된 단어 임베딩(Glove)을 로드

뉴스 기사 분류 작업(**제목**으로 기사 **카테고리** 예측) → 사전 훈련 임베딩(Glove) 미세 조정

* Glove (Global Vectors for Word Representation)

  2014년에 미국 스탠포드대학에서 개발한 단어 임베딩 방법론 (카운트 기반과 예측 기반을 모두 사용)

사전 훈련된 임베딩을 사용한 전이 학습 - 데이터셋

AG 뉴스 데이터셋

데이터 마이닝과 정보 추출 방법을 연구할 목적으로 2005년에 수집한 뉴스 기사 모음.

(100만개 이상의 기사, 여기서는 12만개로 구성된 축소 버전 사용)

	category	title
0	Business	Wall St. Bears Claw Back Into the Black (Reuters)
1	Business	Carlyle Looks Toward Commercial Aerospace (Reu...
2	Business	Oil and Economy Cloud Stocks' Outlook (Reuters)
3	Business	Iraq Halts Oil Exports from Main Southern Pipe...
4	Business	Oil prices soar to all-time record, posing new...

	category	title	split
0	Business	jobs , tax cuts key issues for bush	train
1	Business	jarden buying mr . coffee s maker	train
2	Business	retail sales show festive fervour	train
3	Business	intervoice s customers come calling	train
4	Business	boeing expects air force contract	train

| 사전 훈련된 임베딩을 사용한 전이 학습 – Vocabulary

```
class SequenceVocabulary(Vocabulary):
    def __init__(self, token_to_idx=None, unk_token="<UNK>",
                 mask_token="<MASK>", begin_seq_token="<BEGIN>",
                 end_seq_token="<END>"):
```

SequenceVocabulary

```
{
'<MASK>': 0,
'<UNK>': 1,
'<BEGIN-OF-SEQUENCE>' : 2,
'<END-OF-SEQUENCE>' : 3,
'is': 4,
'happy': 5
}
```

"Jerry is happy"	0단계: 시퀀스 입력
1 4 5	1단계: 단어를 정수로 매핑합니다
2 1 4 5 3	2단계: 문장을 경계 토큰으로 감쌉니다
2 1 4 5 3 0 0	3단계: 모든 벡터의 길이가 같도록 0으로 패딩합니다

사전 훈련된 임베딩을 사용한 전이 학습 - 모델

```
class NewsClassifier(nn.Module):
    def __init__(self, embedding_size, num_embeddings, num_channels,
                 hidden_dim, num_classes, dropout_p,
                 pretrained_embeddings=None, padding_idx=0):
        """
        매개변수:
            embedding_size (int): 임베딩 벡터의 크기
            num_embeddings (int): 임베딩 벡터의 개수
            num_channels (int): 합성곱 커널 개수
            hidden_dim (int): 은닉 차원 크기
            num_classes (int): 클래스 개수
            dropout_p (float): 드롭아웃 확률
            pretrained_embeddings (numpy.array): 사전에 훈련된 단어 임베딩
                기본값은 None
            padding_idx (int): 패딩 인덱스
        """
        super(NewsClassifier, self).__init__()

        if pretrained_embeddings is None:

            self.emb = nn.Embedding(embedding_dim=embedding_size,
                                    num_embeddings=num_embeddings,
                                    padding_idx=padding_idx)
        else:
            pretrained_embeddings = torch.from_numpy(pretrained_embeddings).float()
            self.emb = nn.Embedding(embedding_dim=embedding_size,
                                    num_embeddings=num_embeddings,
                                    padding_idx=padding_idx,
                                    _weight=pretrained_embeddings)
```

→ nn.Embedding 층의 가중치 행렬을
사전 훈련된 임베딩으로 지정

사전 훈련된 임베딩을 사용한 전이 학습 – Glove 적용

디스크에서 사전 훈련된 임베딩 로드

```
# GloVe 데이터를 다운로드합니다.
!wget http://nlp.stanford.edu/data/glove.6B.zip
!unzip glove.6B.zip
!mkdir -p data/glove
!mv glove.6B.100d.txt data/glove
```

```
if args.use_glove:
    words = vectorizer.title_vocab._token_to_idx.keys()
    embeddings = make_embedding_matrix(glove_filepath=args.glove_filepath,
                                       words=words)
    print("사전 훈련된 임베딩을 사용합니다")
```

```
classifier = NewsClassifier(embedding_size=args.embedding_size,
                            num_embeddings=len(vectorizer.title_vocab)
                            num_channels=args.num_channels,
                            hidden_dim=args.hidden_dim,
                            num_classes=len(vectorizer.category_vocab)
                            dropout_p=args.dropout_p,
                            pretrained_embeddings=embeddings,
                            padding_idx=0)
```

```
def make_embedding_matrix(glove_filepath, words):
    """
    특정 단어 집합에 대한 임베딩 행렬을 만듭니다.

    매개변수:
        glove_filepath (str): 임베딩 파일 경로
        words (list): 단어 리스트
    """
    word_to_idx, glove_embeddings = load_glove_from_file(glove_filepath)
    embedding_size = glove_embeddings.shape[1]

    final_embeddings = np.zeros((len(words), embedding_size))

    for i, word in enumerate(words):
        if word in word_to_idx:
            final_embeddings[i, :] = glove_embeddings[word_to_idx[word]]
        else:
            embedding_i = torch.ones(1, embedding_size)
            torch.nn.init.xavier_uniform_(embedding_i)
            final_embeddings[i, :] = embedding_i

    return final_embeddings
```

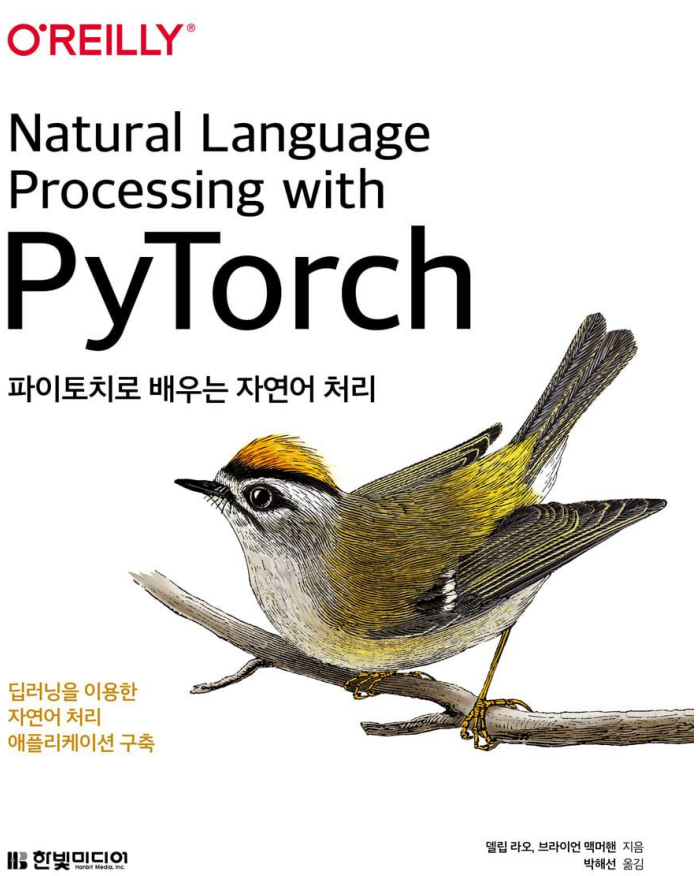
| 사전 훈련된 임베딩을 사용한 전이 학습 - 평가

```
print("테스트 손실: {};".format(train_state['test_loss']))
print("테스트 정확도: {}".format(train_state['test_acc']))
```

```
테스트 손실: 0.6050473968897546;
테스트 정확도: 82.61718750000003
```

```
True Category: Business
==============================
예측: Business (p=0.77)
        + 샘플: AZ suspends marketing of cancer drug
예측: Business (p=0.99)
        + 샘플: Business world has mixed reaction to Perez move
예측: Sports (p=0.64)
        + 샘플: Betting Against Bombay
예측: Sports (p=0.34)
        + 샘플: Malpractice Insurers Face a Tough Market
예측: Sports (p=0.65)
        + 샘플: NVIDIA Is Vindicated
------------------------------
```

| What's next?



6장 자연어 처리를 위한 시퀀스 모델링 - 초급

- 순환 신경망
- 문자 RNN으로 성씨 국적 분류하기

7장 자연어 처리를 위한 시퀀스 모델링 - 중급

- RNN으로 성씨 생성하기