머신러닝 스터디 발표

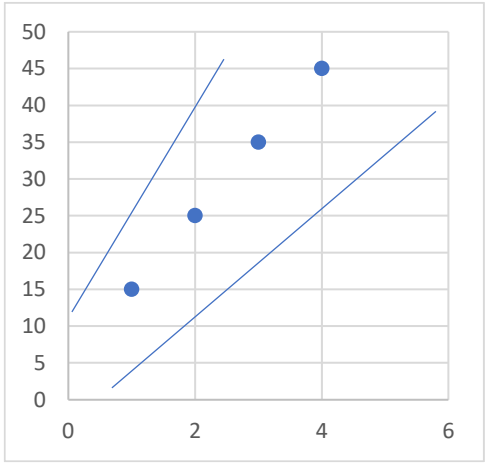
배운 목차

- 1. 선형회귀
- 2. 인공신경망 (ANN)
- 3. 합성곱 신경망 (CNN)
- 4. 순환신경망 (RNN)
- 5. 그래프 자료구조

1.선형회귀

- 가설 : H(x) = wx+b 이때 w는 가중치 b는 편향을 말함
- x와 y의 관계를 유추하기 위해서 수학적으로 식을 세워보게 되는데 머신러닝에서는 이러한 식을 가설(Hypothesis)라고 한다

시간	점수
2	15
4	25
6	35
8	45

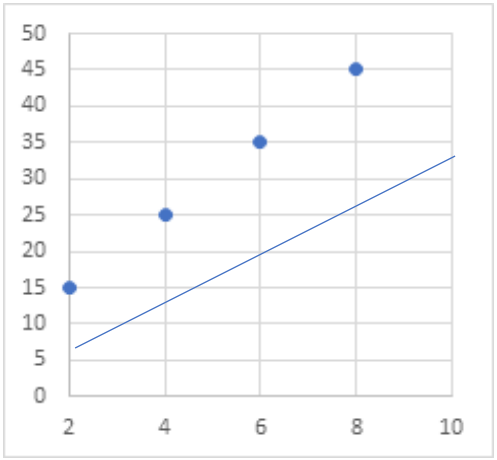


.

어떤 직선인지 결정하는 것은 결국 w와 b의 값이므로 선형 회귀에서 해야할 일은 결국 w와 b를 찾는 일이 된다.

1.선형회귀

- 비용함수: 실제값과 가설로부터 얻은 예측값의 오차를 계산해서 w와 b를 찾기 위한 함수
  평균 제곱 오차(Mean Squered Error, MSE)를 사용함
- 옵티마이저 : 비용함수를 최소화 하기 위한 작업을 수행하기 위해 쓰는 알고리즘을 말함
  경사하강법(Gradient Descent)을 사용 함



시간(X)	2	4	6	8
실제값	15	25	35	45
예측값	9	13	17	21
오차	6	12	18	24

Y=2X+5 직선이 예측한 예측값과 실제값으로 부터의 오차

1.선형회귀

```
Epoch    0/2000 Cost: 1087.454102
Epoch  100/2000 Cost: 1.814750
Epoch  200/2000 Cost: 0.947755
Epoch  300/2000 Cost: 0.494966
Epoch  400/2000 Cost: 0.258496
Epoch  500/2000 Cost: 0.135000
Epoch  600/2000 Cost: 0.070504
Epoch  700/2000 Cost: 0.036821
Epoch  800/2000 Cost: 0.019230
Epoch  900/2000 Cost: 0.010043
Epoch 1000/2000 Cost: 0.005245
Epoch 1100/2000 Cost: 0.002739
Epoch 1200/2000 Cost: 0.001431
Epoch 1300/2000 Cost: 0.000747
Epoch 1400/2000 Cost: 0.000390
Epoch 1500/2000 Cost: 0.000204
Epoch 1600/2000 Cost: 0.000106
Epoch 1700/2000 Cost: 0.000056
Epoch 1800/2000 Cost: 0.000029
Epoch 1900/2000 Cost: 0.000015
Epoch 2000/2000 Cost: 0.000008
```

```
[3]  class Linear(nn.Module):
        def __init__(self):
            super().__init__()
            self.linear = nn.Linear(1, 1)

        def forward(self, x):
            return self.linear(x)
    model = Linear()
```

```
# optimizer 설정. 경사 하강법 SGD를 사용하고 learning rate를 의미하는 lr은 0.01
optimizer = torch.optim.SGD(model.parameters(), lr=0.01)
```

```
# 전체 훈련 데이터에 대해 경사 하강법을 2,000회 반
nb_epochs = 2000
for epoch in range(nb_epochs+1):

    # H(x) 계산
    prediction = model(x_train)

    # cost 계산
    # 파이토치에서 제공하는 평균 제곱 오차 함수
    cost = F.mse_loss(prediction, y_train)

    # cost로 H(x) 개선하는 부분
    # gradient를 0으로 초기화
    optimizer.zero_grad()
    # 비용 함수를 미분하여 gradient 계산
    cost.backward()
    # W와 b를 업데이트
    optimizer.step()

    if epoch % 100 == 0:
    # 100번마다 로그 출력
      print('Epoch {:4d}/{} Cost: {:.6f}'.format(
          epoch, nb_epochs, cost.item()
```

2. 인공신경망

- 1. 퍼셉트론
- 2. XOR 게이트
- 3. 순전파, 역전파

2. 인공신경망

- 1. 퍼셉트론 : 인공신경망으로 실제 뇌를 구성하는 신경 세포 뉴런의 동작과 유사하게 짠 모델

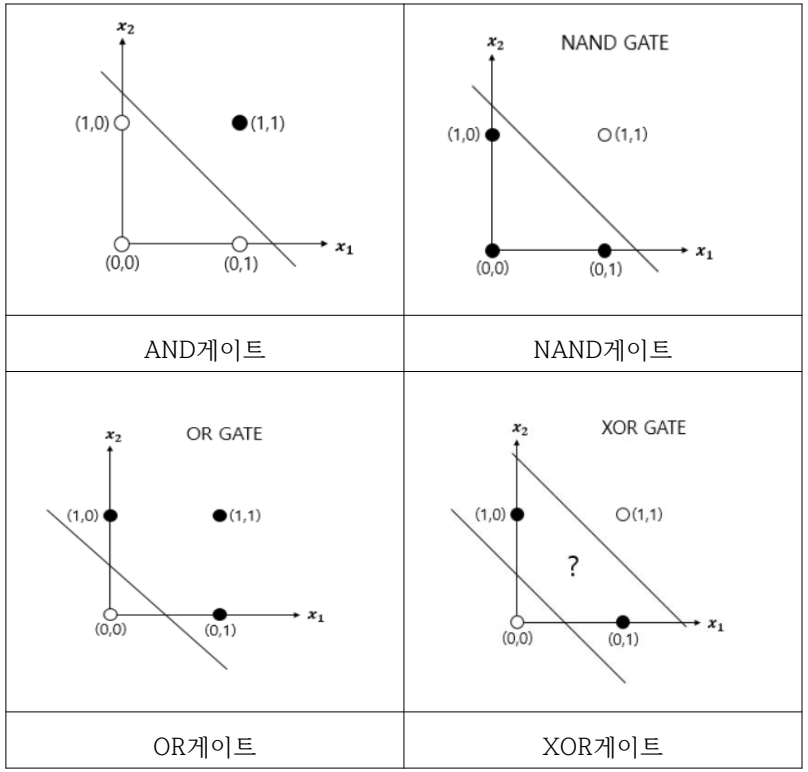
- 2. XOR Gate :

단층 퍼셉트론(Single Layer Perceptron) 으로는

AND, NAND, OR 게이트를 구현할 수 있다.

XOR게이트는 단층퍼셉트론을 조합한

다층 퍼셉트론(MultiLayer Perceptronm MLP)으로 만들 수 있다

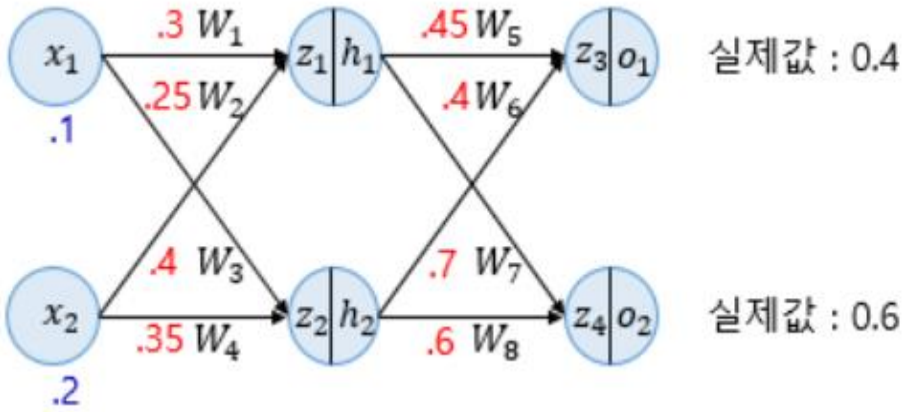


2. 인공신경망

•순전파(foward propagation)

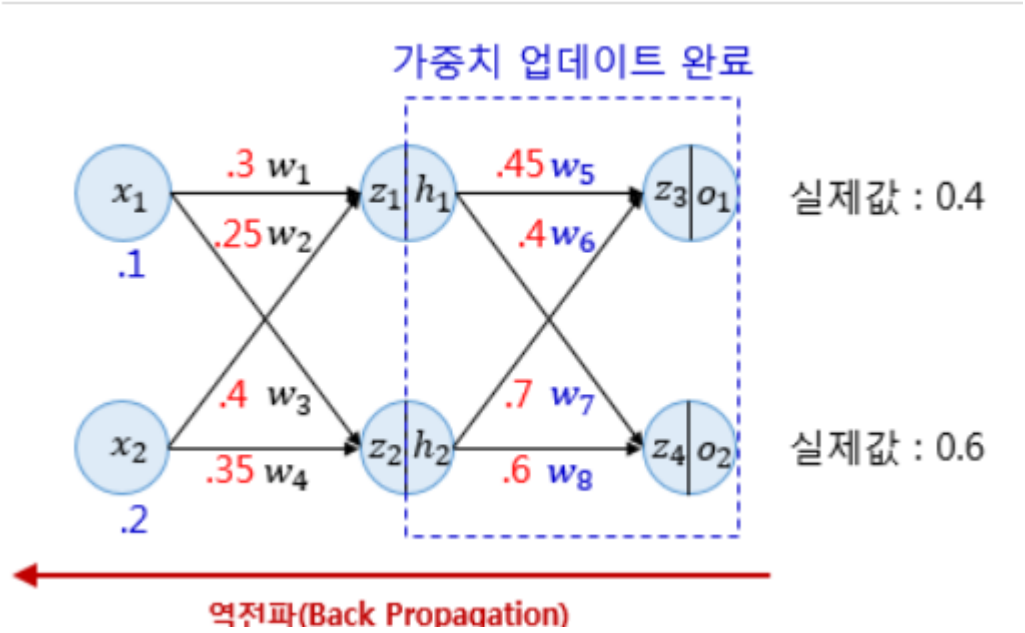
•입력받은 데이터를 각 가중치와 곱하여 출력을 뽑아내는 과정

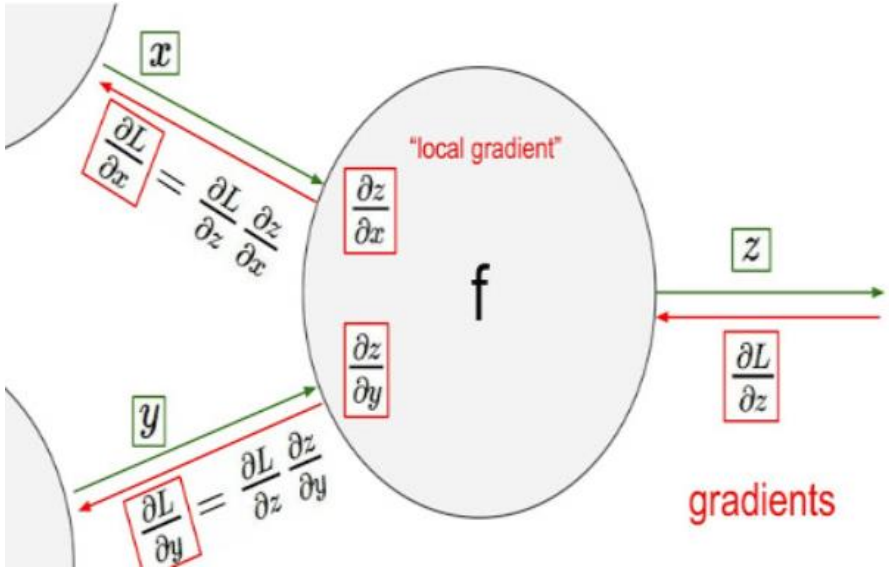
•이 때 은닉층에서 활성화 함수를 계산하는데 활성화 함수를 사용하는 이유는 활성화 함수를 사용하면 입력값에 대한 출력값이 선형하게 나오지 않으므로 선형분류기를 비선형 시스템으로 만들 수 있다



2. 인공신경망

- 역전파 :역전파 알고리즘은 출력값에 대한 입력값의 기울기(미분값)을 출력층 layer에서부터 계산하여 거꾸로 전파시키는 것이다.
- 이렇게 거꾸로 전파시켜서 최종적으로 출력층에서의 output값에 대한 입력층에서의 input data의 기울기 값을 구할 수 있다.
- 이 과정에는 중요한 개념인 chain rule이 이용된다.





2. 인공신경망

- Torch의 autograd
- backward() 메소드를 통해 가중치를 부분 업데이트 할 수 있다.

```
# a.grad: a->b->c->d 로 갈때 d에서 a의 미분값을 저장
#grad_fn:  이전에 연산을 무엇을 했는지 확인
# a->b->c->d
a=torch.ones(2,2, requires_grad=True)
b=a+4
c=b**2
d=c.sum()
d.backward()
```

```
#d.backward()를 통해 연산 처음 부분과 마지막 부분의 미분값을 저장
#이때 a의 grad_fn이 none인 이유는 a설정 이전에 연산이 없었기 때문
print("------a의값--------")
print('a.data: ',a.data)
print('a.grad: ',a.grad)
print('a.grad_fn :',a.grad_fn)
```

```
a.data:  tensor([[1., 1.],
        [1., 1.]])
a.grad:  tensor([[10., 10.],
        [10., 10.]])
a.grad_fn : None
```

```
#b로 넘어가기 위해 a에서 더하기 연산을 해서 addBackward
print("------b의값--------")
print('b.data:',b.data)
print('b.grad:',b.grad)
print('b.grad_fn :',b.grad_fn)
```

```
b.data: tensor([[5., 5.],
        [5., 5.]])
b.grad: None
b.grad_fn : <AddBackward0 object at 0x7fc76a720400>
```

3. 합성곱 신경망(CNN)

- 1. Convolutoin(합성곱) Layer
- 2. pooling Layer
- 3. 파라미터

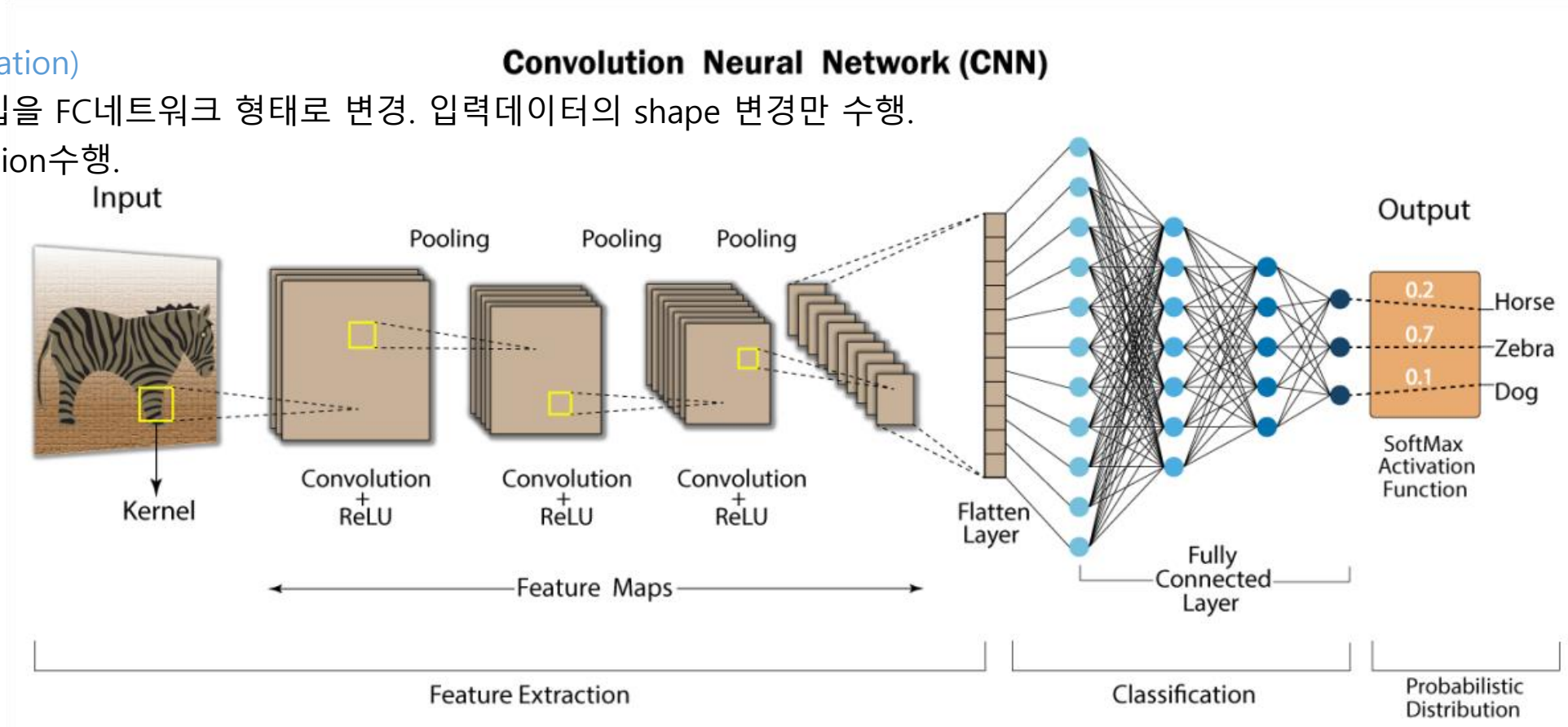
- Cnn은 이미지를 날 것 그대로 받음으로써 공간적/지역적 정보를 유지한 채 특성들의 계층을 빌드업 한다
- Cnn의 중요 포인트는 이미지 전체보다는 부분을 보는 것 , 그리고 이미지의 한 픽셀과 주변 픽셀들의 연관성을 살리는 것이다

3. 합성곱 신경망(CNN)

- 특징 추출 단계(Feature Extraction)
- Convolution Layer : 필터를 통해 이미지의 특징을 추출. 높이x너비x채널의 3차원 tensor로 표현
- Pooling Layer : 특징을 강화시키고 이미지의 크기를 줄임.
- (Convolution과 Pooling을 반복하면서 이미지의 feature를 추출)

- 이미지 분류 단계(Classification)
- Flatten Layer : 데이터 타입을 FC네트워크 형태로 변경. 입력데이터의 shape 변경만 수행.
- Softmax Layer : Classification수행.
- Output : 인식결과

- CNN의 파라미터
- Convolution Filter의 개수
- Filter의 사이즈
- Padding여부
- Stride



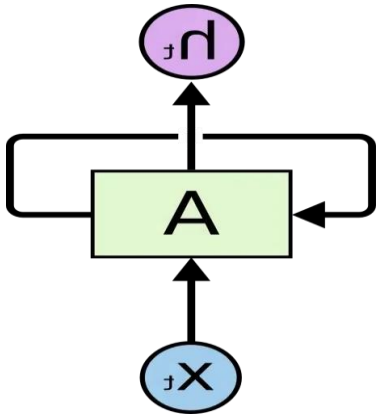
4. 순환 신경망(RNN)

- 입력된 데이터가 입력층에서 은닉층으로, 은닉층의 결과가 다시 은닉층의 입력으로 반복한다.
- 이전 상태에 대한 정보를 메모리 형태로 저장해서 그다음 상태의 입력으로 사용하며 Sequence Data를 다루는 데 도움이 된다

- 학습방법 : Rnn은 기존의 인공신경망과 같이 각층의 뉴런이

연결되어 있는 구조이고 추가로 과거 자신의 정보(가중치)를 기억하고 학습에 반영한다.

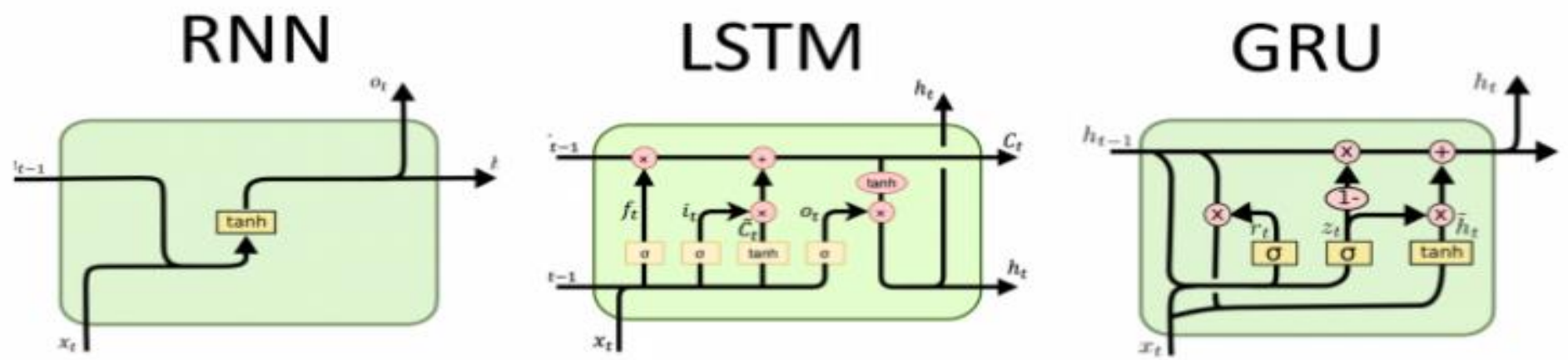
그로 인해 이전 작업을 현재 작업과 연결 할 수 있다는 의미를 가진다



4. 순환 신경망(RNN)

- LSTM , GRU
- RNN의 경우 시간의 흐름에 따른 작업이기에 시간을 거슬러올라가서 역전파가 적용된다.
- 학습 도중 데이터의 시퀀스가 길어지면 Vanishing Gradients Problem이 생긴다

- 셀 상태(Cell state)라는 구조를 만들어 게이트를 추가하여 입력(input), 망각(Forget),출력(Output) 게이트등을 통해 상태 값을 메모리 공간 셀에 저장하고, 데이터를 접하는 게이트 부분을 조정하여 불필요한 연산, 오차 등을 줄이고 연산의 경우 곱셈이 아닌 더하기 연산으로 장기 의존성 문제를 일정 부분 해결한다.



http://dprogrammer.org/rnn-lstm-gru

4. 순환 신경망(RNN)

- GRU로 만든 쇼핑몰 리뷰분석

```
import
import
import
import
import
import
from co
from ko
from sk
from te
from te
```

```
from tensorflow.keras.layers import Embedding, Dense, GRU
from tensorflow.keras.models import Sequential
from tensorflow.keras.models import load_model
from tensorflow.keras.callbacks import EarlyStopping, ModelCheckpoint
```

0	5	
1	2	택배가 엉망이네용 저희집
2	5	아주좋아요 바지 정말 좋아서2개 더 구매했어요 이가격
3	2	선물용으로 빨리 받아서 전달했어야 하는 상품이었는데 머
4	5	민트색상 예뻐요. 옆 손잡이는

```
    ng_dim = 100 # 임베딩층
    units = 128 # 은닉상태

    Sequential()
dd(Embedding(vocab_size, embedding_dim))
model.add(GRU(hidden_units))
model.add(Dense(1, activation='sigmoid'))

# 데이터 손실이 4회증가하면 조기 종료
es = EarlyStopping(monitor='val_loss', mode='min', verbose=1, patience=4)
mc = ModelCheckpoint('best_model.h5', monitor='val_acc', mode='max', verbose=1, save_best_only=True)
model.compile(optimizer='rmsprop', loss='binary_crossentropy', metrics=['acc'])
history = model.fit(X_train, y_train, epochs=15, callbacks=[es, mc], batch_size=64, validation_split=0.2)
```

```
def sentiment_predict():
  new_sentence = input('리뷰를 써주세요 : ')
  new_sentence = re.sub(r'[^ㄱ-ㅎㅏ-ㅣ가-힣 ]','', new_sentence)
  new_sentence = mecab.morphs(new_sentence)
  new_sentence = [word for word in new_sentence if word not in stopwords]
  encoded = tokenizer.texts_to_sequences([new_sentence])
  pad_new = pad_sequences(encoded, maxlen = max_len)
  # print(pad_new)
  score = float(loaded_model.predict(pad_new))
  if(score > 0.5):
    print("{:.2f}% 확률로 긍정 리뷰입니다.".format(score * 100))
  else:
    print("{:.2f}% 확률로 부정 리뷰입니다.".format((1 - score) * 100))
```

```
, 'reviews'])
```

```
리뷰를 써주세요 : 이거 별로지만 쓸만하다
1/1 [==============================] - 0s 307ms/step
64.36% 확률로 긍정 리뷰입니다.

리뷰를 써주세요 : 이거 쓸만하지만 별로다
1/1 [==============================] - 0s 16ms/step
97.27% 확률로 부정 리뷰입니다.
```

Word2vec

- 영화 리뷰로 만든 word2vec

```
import pandas as pd
import pickle
import matplotlib.pyplot as plt
import urllib.request
from tqdm import tqdm
from gensim.models.word2vec import Word2Vec
from konlpy.tag import Okt
```

	id	document	label
0	8112052	어릴때보고 지금다시봐도 재밌어요ㅋㅋ	1
1	8132799	디자인을 배우는 학생으로, 외국디자이너와 그들이 일군 전통을 통해 발전해가는 문화산...	1
2	4655635	폴리스스토리 시리즈는 1부터 뉴까지 버릴께 하나도 없음.. 최고.	1
3	9251303	와.. 연기가 진짜 개쩔구나.. 지루할거라고 생각했는데 몰입해서 봤다.. 그래 이런...	1
4	10067386	안개 자욱한 밤하늘에 떠 있는 초승달 같은 영화.	1

```
# 리뷰 개수 출력
print(len(train_data))
```

```
200000
```

```
from gensim.models import Word2Vec

model = Word2Vec(sentences = tokenized_data, size = 100, window = 5, min_count = 5, workers = 4, sg = 0)
```

```
print(model.wv.most_similar("이정재"))
```

```
[('최민수', 0.890080988407135), ('박신양', 0.8856963515281677), ('톰크루즈', 0.8721780776977539), ('박중훈', 0.865345299243927), ('설경구', 0.858391523361206)
```

```
print(model.wv.most_similar("전쟁"))
```

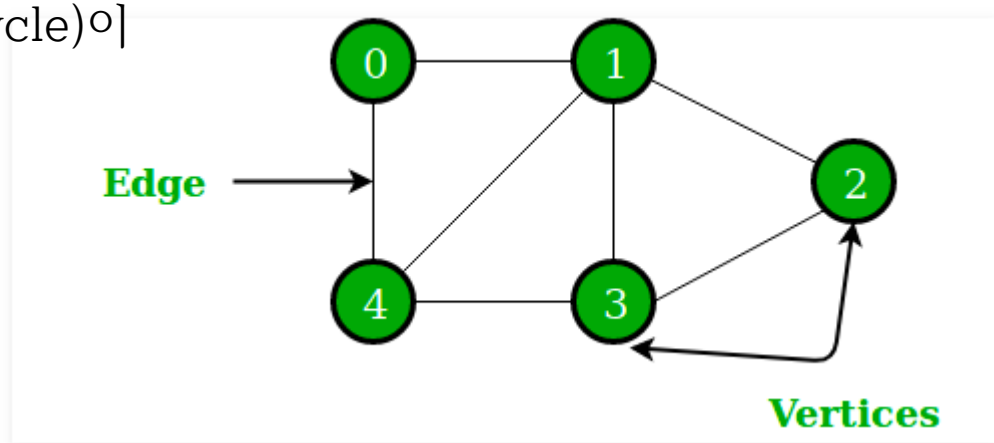
```
[('폭력', 0.8113541007041931), ('영웅', 0.796877384185791), ('정의', 0.7879595756530762), ('인권', 0.7534334063529968), ('학살', 0.7513887882232666),
```

5.그래프

•그래프는 연결되어 있는 원소 사이의 다대다 관계를 표현하는 자료구조

•그래프 G는 객체를 나타내는 정점 V(vertex)와 객체를 연결하는 E(edge)의 집합이다

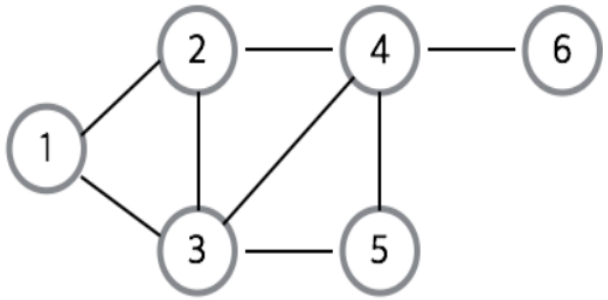
•트리도 그래프의 한 종류이며, 그 중 사이클(cycle)이 허용되지 않는 그래프를 말한다



5.그래프(구현방법)

- 1. 인접 행렬(Adjacency Materix)

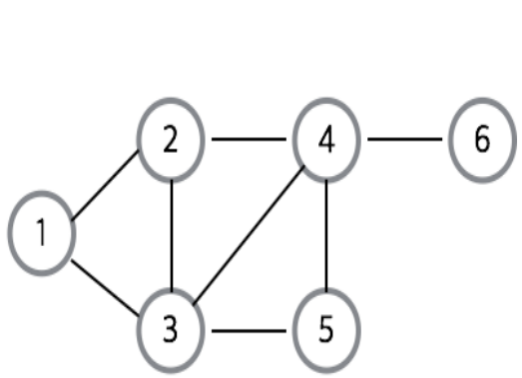
인접 행렬은 2차원 배열로 그래프를 구현하는 방식
간선이 존재하는 두 정점 칸은 1로 없는 칸은 0으로 채워주고 만약 가중치가 다른 그래프라면 해당 가중치 값을 넣어준다

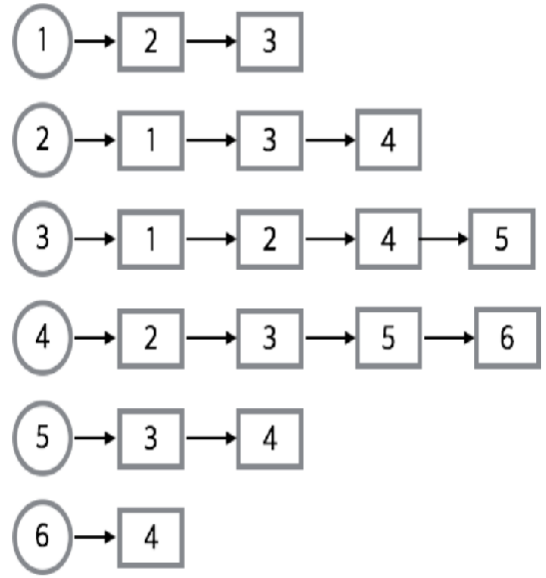


	1	2	3	4	5	6
1	0	1	1	0	0	0
2	1	0	1	1	0	0
3	1	1	0	1	1	0
4	0	1	1	0	1	1
5	0	0	1	1	0	0
6	0	0	0	1	0	0

- 2. 인접 리스트(Adjacency List)

인접 리스트는 정점에 연결되어 있는 정점들만 리스트로 나타내는 그래프 표현 방식이다
그래프의 노드를 리스트로 표현한 것





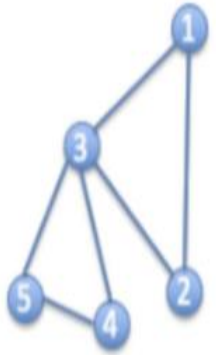
5.그래프

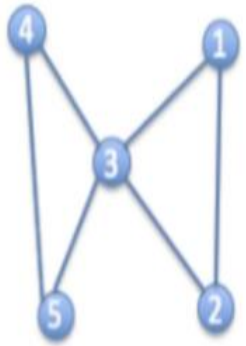
- 그래프 분석이 힘든 이유

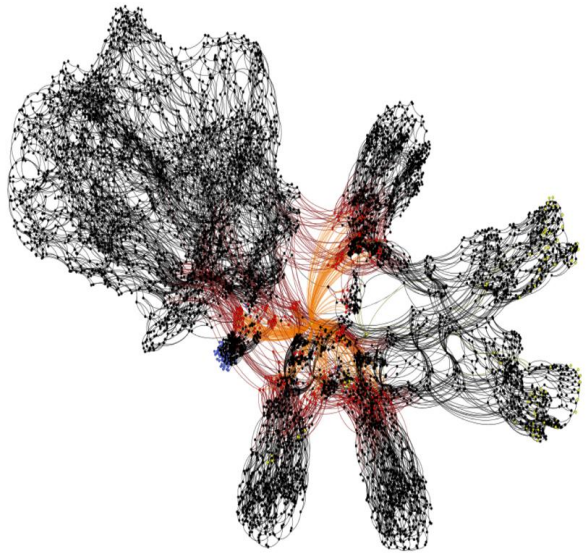
1. 그래프는 유클리드 공간에 있지 않기 때문에우리에게 익숙한 좌표계로 표현할 수 없다는 것을 의미한다.
   시계열 데이터, 음성, 이미지 같은 데이터는 2차원, 3차원 유클리드 공간에 쉽게 매핑할 수 있 는데 그래프 데이터의 해석은 비교적 어렵다

2. 그래프는 고정된 형태가 아니다.

3. 그래프는 사람이 해석할 수 있도록 시각화 하는 것이 어렵다







5. 그래프

Graph Neural Network(GNN)

- GNN은 그래프에 직접 적용할 수 있는 신경망 점 레벨, 선 레벨 ,그래프 레벨에서 예측 작업에 사용 됨
- GNN의 핵심은 점이 이웃과의 연결에 의해 정의된다.
- 만약 어떤 점의 이웃과 연결을 다끊으면 그 점은 고립되고 아무 의미를 갖지 않는다.

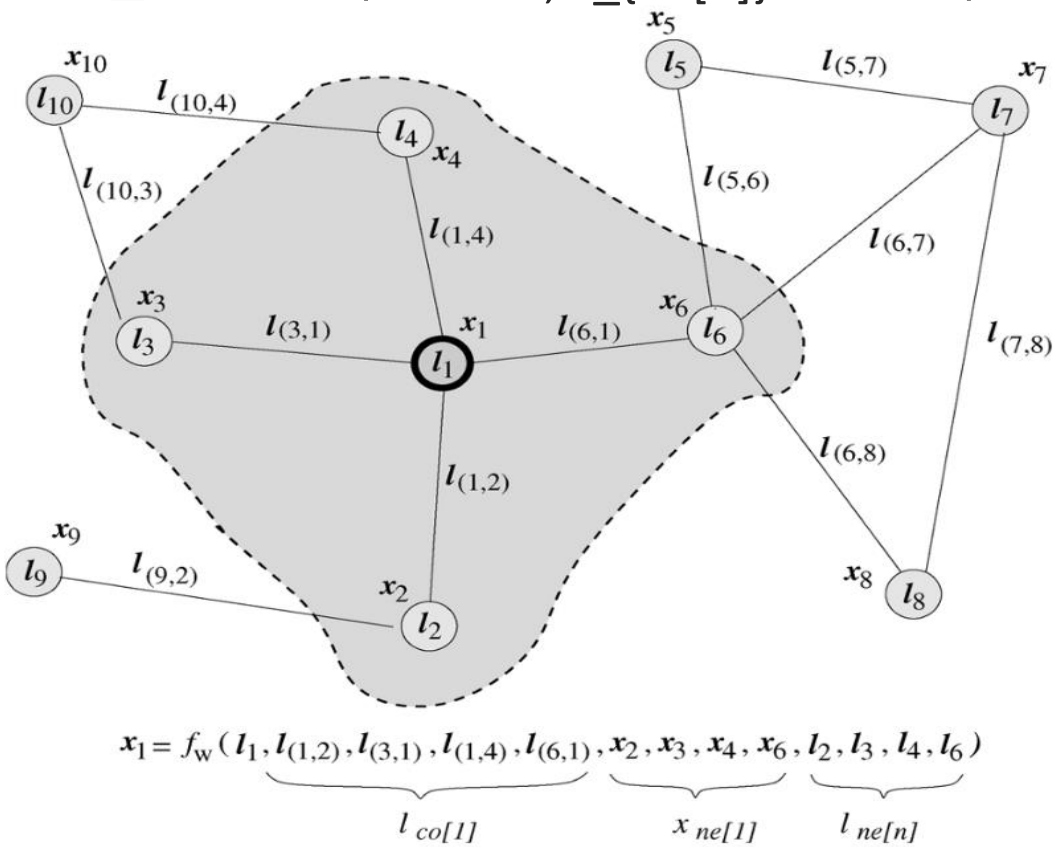
- 1. Recurrent Graph Neural Network
- 2. Spatial Convolutional Network
- 3. Spectral Convolutional Network

Recurrent GNN

- Recurrent GNN은 입력과 출력이 아래와 같은 함수 f_w를 정의하여 점의 상태를 업데이트 한다.

$$\boldsymbol{x}_n = f_{\boldsymbol{w}}(\boldsymbol{l}_n, \boldsymbol{l}_{\mathrm{co}[n]}, \boldsymbol{x}_{\mathrm{ne}[n]}, \boldsymbol{l}_{\mathrm{ne}[n]})$$

- 여기에서, l_{n}는 점 n의 feature, l_{co[n]}은 점 n과 연결된 선들의 feature, l_{ne[n]}은 점 n과 연결된 점들의 feature, x_{ne[n]}은 점 n과 연결된 점들의 상태를 의미한다.



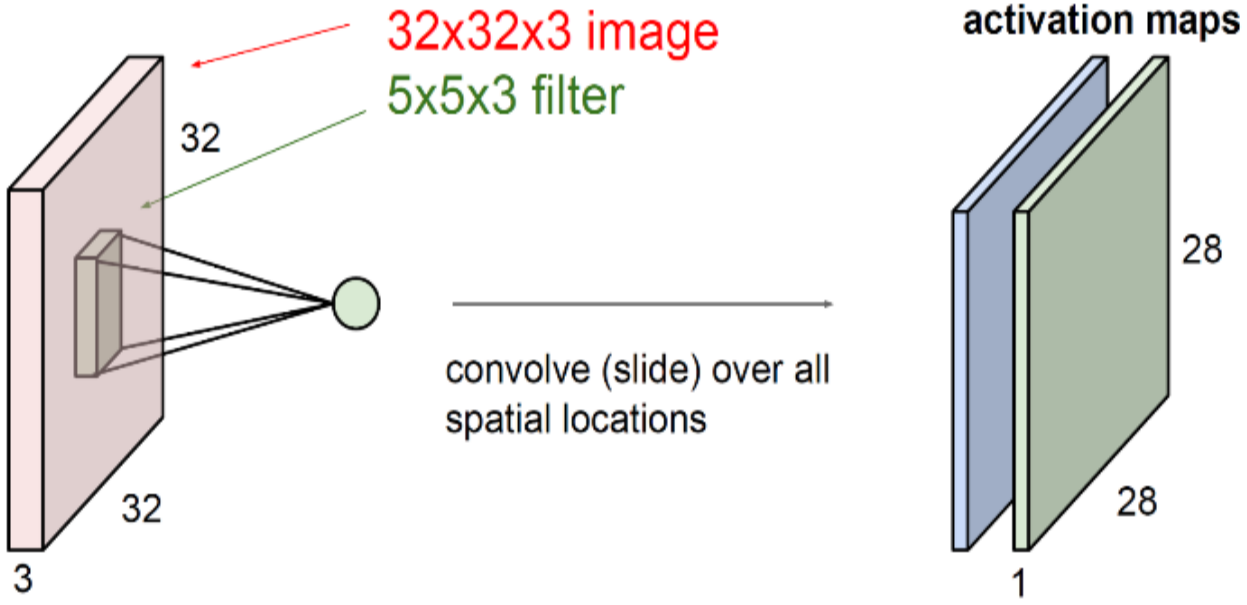
$$\boldsymbol{o}_n = g_{\boldsymbol{w}}(\boldsymbol{x}_n, \boldsymbol{l}_n)$$

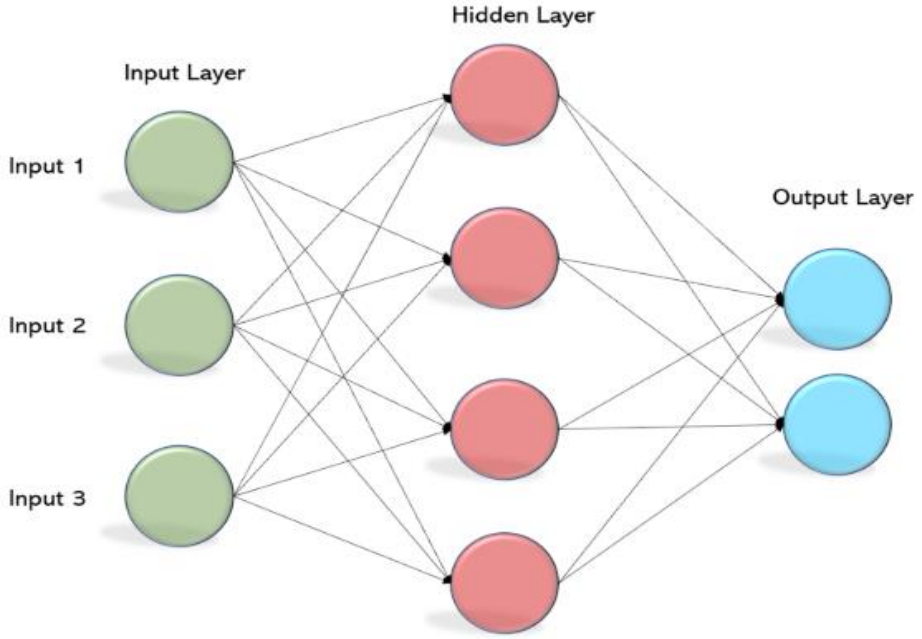
K번 반복을 통한 업데이트 후 마지막 상태(X_n)와 특징(l_n)을 사용하여 결과값 O_n을 출력

Graph Convolution Network

- 1. Weight Sharing
- CNN에서는 input data(ex. image)에 동일한 weight를 갖는 동일 filter를 이동하면서 연산을 진행한다.
- 이때 보통 stride가 filter크기보다 작게하여 이동시키므로 중복되어 연산되는 pixel이 다수 존재한다.

- 2. Learn local features
- 일반적인 MLP에서는 hidden node 들이 모든 input node와 연결지어 있어서 global feature들에 의해 학습된다.
- 그러나 CNN에서는 filter가 이동하면서 local feature들이랑만 연산하고 activation map을 생성하므로
- 각 activation map의 pixel들은 global이 아닌 local feature들에 의해서만 학습된다고 볼 수 있다.





GNN 활용 분야

- 1. Node Classification
- Node embedding 을 통해 점들을 분류하는 문제이다.
  그래프의 일부만 레이블링 된 상태에서 semi-supervised learning을 한다.
  Youtube 동영상 , 게시물 분류 등에 사용할 수 있다.

- 2. Link Prediction
- 그래프 점들 사이의 관계를 파악하고 그 연관성을 예측하는 문제이다.
  페이스북 친구추천 등의 추천알고리즘에 사용될 수 있다.

- 3. Graph Classification
- 그래프 전체를 여러 카테고리로 분류하는 문제이다.
  그래프의 노드들을 분류하는 문제로 주로 그래프는 분자구조 그래프 형태로 제공되며
  주로 화학 생의학 등에서 연구한다.