learning-AI : deep learning application (61357002)

summary_for_final : 기말고사를 위한 총정리 및 서술

- **임규연 (lky473736)**

---

(1) train, test로 나뉘는 이유

- train data는 모델 훈련을 위해, test data는 모델의 일반화 성능을 확인하기 위해
    - 만약 train score가 val score보다 높으면서 gap이 크다 -> overfitting
    - 만약 train score와 val score 둘 다 낮다 -> underfitting
    - cross-validation
        - train set을 폴드로 나누어 계속 번갈아가면서 모델의 일반화 성능을 check

(2) self-supervised learning

- label이 없는 데이터에서 label을 스스로 새롭게 생성함
- masked learning이 여기서 나옴
- 일종의 비지도 학습임

(3) masked learning

- 원본 데이터에 masking을 입힌다 -> masking된 데이터를 입력으로, 원본 데이터를 target으로 하여 학습 -> 복원하는 능력을 학습 가능
- 모델이 다양한 상황을 대처하게 하여 "robust"하게 모델 만듦
- BERT
    - Bidirectional Encoder Representations from Transformer(BERT)
    - 문장의 특정 단어를 가려서 그 단어가 무엇인지를 예측하게 하여 학습 (self-supervised)

(4) semi-supervised learning

- 반지도학습 == 지도학습 + 비지도학습
- 먼저 비지도학습하여 특정 데이터의 군집을 찾고, 레이블이 있는 데이터로 지도학습함
    - 일부분은 label을 알려주어 정확도를 높임
    - 데이터를 섞을 수도 있고, 모델을 섞을 수도 있고

(5) feature selection vs feature engineering

- feature selection : target과 상관관계 (corr) 및 유사도가 높은 feature만을 택하여 학습하는 방법 (모든 feature를 전부 학습에 동원하는 것이 아니라)
    - 모델의 성능이 좋아짐 + overfitting을 막을 수 있음
- feature engineering : feature들끼리 조합하여 유의미한 feature 생성하여 예측에 도움을 줌
    - feature extraction, feature selection ...

(6) overfitting, underfitting

- overfitting
    - 모델이 train data에만 너무 학습이 잘 되어서 새로운 데이터에 대하여 제대로 예측하지 못할 때
    - train score와 val score의 gap이 너무 큼 + train score의 값이 월등히 큼
    - 해결 방법
        - 모델 파라미터의 수를 줄인다 (모델의 복잡도를 낮춘다)
        - 규제를 적용한다
            - layer 내에서 적용 가능
            - l2, l1 규제
                - l2 : ridge : 제곱 규제 : feature을 가능하면 0에 가깝게 만드려고 함
                - l1 : lasso : 절댓값 규제 : feature을 가능하면 0으로 만드려고 함
        - early stopping, dropout
        - 데이터 추가 및 노이즈 제거
- underfitting
    - 모델이 train data에도 학습을 제대로 하지 못하여 전체적인 성능이 떨어질 때
    - train score와 val score가 현저히 낮은 성능을 보임
    - 해결 방법
        - data 수를 늘린다
        - 규제의 양을 줄인다
        - model parameter의 수를 늘리기 (모델을 복잡하게 만든다)
        - 노이즈, 결측치를 제거한다

(7) confusion matrix, 각종 지표, MSE, RMSE, MAE...

- confusion matrix : 혼동 행렬
    - 모델이 얼마나 분류를 잘 했는지 나타냄
    - 쓰는 이유 : 어느 클래스에서 분류가 잘 되었는지를 알 수 있음
TN	**FP**
FN	TP
        - TN(진음성), FP(가양성), FN(가음성), TP(진양성)
        - accuracy = (TN + TP) / (TN + TP + FN + FP)
        - precision = (TP) / (FP + TP)
        - recall = (TP) / (FN + TP)
        - F1-score = 2 * (precision + recall) / (precision * recall)
- F1-score을 사용하는 이유
100	**10**
10	10
    - 위와 같은 confusion matrix가 있다고 가정하였을 때, accuracy는 매우 높음. 하지만 accuracy가 높다고 꼭 분류가 잘 된거라고 생각할 수 있을까?
        - 대한민국에서 암에 걸린 환자는 암에 걸리지 않은 인간보다 매우 희소함
        - 따라서 만약 이 둘을 분류한다면, 암의 걸리지 않은 인간보다 암에 걸린 환자를 분류해내는 것이 main task가 될 것임

        - 결국엔 현재 위 상황은 암 환자에 대한 분류가 적절히 이루어진 것이 아니라고 판단할 수 있겠으며, acc 이 데이터 불균형을 제대로 고려하지 못한다는 것이 됨 -> 따라서 데이터 불균형 고려하는 F1-score을 사용함

- precision & recall 사용하는 이유
    - 일단 이 둘은 서로 trade-off
    - precision이 낮으려면 TP+FP가 낮아야함
    - recall이 낮으려면 FN+TP가 낮아야 함
    - 불균형 데이터셋(클래스 간 데이터 분포가 크게 차이 나는 경우)에 적합
        - precision : 스팸 필터링에 적합 (거짓 양성 최소화하고 싶으니깐)
        - recall : 암 진단 시스템에 적합 (거짓 음성 최소화하고 싶으니깐)

(8) embedding

- 하는 이유 : ai가 각각의 component를 이해하기 위하여 숫자로 변환시키는 것
- 종류
    - one-hot encoding : 각 label의 관계를 끊어주기 위하여 (데이터 편향성을 없애기 위하여 + label을 각각 공평하게 보기 위하여)
    - positional encoding : transformer에서 나옴
    - vector encoding : {0.1, 0.4, 1.0, 0.7}

(9) scaling

- 하는 이유 : 모든 데이터의 분포를 일정하게 하여 학습을 용이하게 하고 성능을 높이기 위해서
    - 만약에 거리 기반의 사례 기반 모델 (KNN)을 사용하여 학습한다고 가정 -> scaling을 안하면 값이 큰 feature 가 모델에 더 많은 영향을 미칠 것임 -> feature의 공평성이 없어짐

(10) ensemble

- 훈련세트로부터 랜덤으로 각기 다른 서브셋 만들어 결정 트리를 훈련 가능하다
    - 일련의 예측기로 예측 수집하면 가장 좋은 모델 하나보다 더 좋은 예측 얻을 수 있음
    - 랜덤 포레스트의 정의 : 결정 트리의 앙상블, 개별 트리의 예측을 모아 가장 많은 선택을 받은 클래스를 앙상블의 예측으로 결정 (일반적으로 배깅 혹은 페이스팅을 적용한 결정 트리 앙상블)
- sample을 boostraping한다 (중복을 허용하여 여러개의 샘플을 만든다 == 복원추출한다)
    - 복원추출하기 때문에 OOB 발생 (선택받지 못한 데이터를 검증으로 씀)
        - 배깅 : 복원추출
        - 페이스팅 : 비복원추출 (배깅이 복원추출이기 때문에 편향 높음)
            - 참고 : 분산이 크다 -> hyperparameter나 모델의 구조가 조금만 변경되어도 다른 모델이 탄생될 수 있음

(11) hyperparameter tuning

- hyperparameter : 사용자가 직접 조정 가능한 파라미터, 모델의 구조를 구성하거나 할 때 사용 (규제량 등)
- hyperparameter tuning의 이유 : 모델의 성능을 높이기 위함 + overfitting을 막기 위함
    - weight나 bias는 모델 내부의 파라미터지만 모델링은 개발자가 하게 됨
    - hyperparameter의 조합에 따라 모델은 각자 천차만별임 (최적의 조합을 찾아야함)
        - grid search : 가능한 모든 조합을 brute-force

        - random search : 정해놓은 분포 안에서 균등분포 추출을 통한 조합 시도
        - bayesian optimization

(12) 경사하강법

- loss가 최소가 되는 weight와 bias를 찾는 최적화알고리즘
    - $W_{t+1} = r * W_{t} + b$
        - r : learning rate
            - 학습율이 높다 : 빠르게 적응하지만 예전 데이터 금방 잊음
            - 학습율이 낮다 : 학습 속도가 느려지지만 노이즈에 덜 민감
        - b : bias
- 종류
    - 배치경사하강법 : 모든 데이터를 이용하여 bias와 weight를 구함
        - 장점 : 특이성에 민감하지 않다
        - 단점 : 시간이 오래 걸린다 + local minima + epoch 안에 global minima에 수렴하지 못할 가능성 있음
    - 확률적경사하강법 : 매 스텝에서 한 샘플을 랜덤으로 선택하고 그에 대한 weight와 bias를 구함
        - 장점 : 시간이 적게 걸림
        - 단점 : global minima에 수렴할 거라는 보장을 할 수 없음
    - 미니배치경사하강법 : 데이터셋을 일정한 크기의 미니배치 단위로 나누고, 한 배치를 택하여 학습을 진행, online 학습이 가능함
        - 장점 : 적은 데이터를 사용하기 때문에 한정된 자원에서 매우 유리함, GPU를 사용하여 빠른 병렬 연산이 가능해짐

(13) SVM

- 각 class와의 거리가 최대가 되는 결정 경계를 찾는 알고리즘, margin이 최대가 되는 것이 목표
    - hyperplane : 결정 경계
    - support vector : hyperplane 주변에 있는 component를 의미
        - support vector가 hyperplane 안에 있는지 없는지에 따라 (즉, margin violation이 있는지에 따라) soft margin, hard margin으로 구분
            - soft margin : margin violation이 어느 정도 허용
            - hard margin : margin violation을 허용하지 않음
            - 위는 hyperparameter 값 중 C로 결정됨
                - C 크다 -> margin violation을 엄격히 제한 -> overfitting 야기
                - C 작다 -> margin violation을 완화 -> underfitting을 야기

(14) DT

- If-then 형식의 질문 노드를 통하여 예측을 수행, 이진 트리를 계속 만듦
- gini impurity (지니 불순도)
    - 현재 node에 들어온 각 component들의 class가 얼마나 섞여 있는지를 나타냄
    - 0.5에 가깝다 -> 불순도 낮음 / 0이다 -> 순수함 (한 클래스만 있음)
    - 결국에 불순도를 낮게 하는 것이 DT의 목적이 됨

(15) curse of dimensionality

- 차원의 저주 : 데이터의 차원이 높을 수록 계산이 복잡해지고 overfitting될 수 있다. (feature가 너무 많기 때문)

    - featuer가 많다 -> 훈련이 힘들다 -> 시간이 너무 오래 걸린다
    - 차원을 줄이는 방법
        - feature selection
        - PCA (주성분 분석)
        - regularization

(16) PCA

- 주성분을 찾아서 차원을 감소시키는 방법, 데이터의 분산이 최대가 되는 축을 찾음, n차원의 공간을 n-p차원으로 줄이기 위해서 p+1개의 과정이 필요함 (분산이 최대 -> hyperparameter나 모델 파라미터가 조금이라도 변해도 새로운 모델이 만들어질 수 있음)
- 과정
    - 학습 데이터셋에서 분산이 최대가 되는 축을 찾음
    - 그 축과 수직이면서 그 다음으로 분산이 큰 축을 찾음
    - 반복 ...

(17) manifold learning

- 고차원의 데이터를 잘 분석하여 dimension을 줄이면 더욱 쉽게 파악이 가능하다는 가설
- 실제 고차원 데이터가 저차원 데이터에 가깝게 놓여 있다고 가정하는 것임
- 고차원을 저차원으로 피면 학습 속도가 빨라지겠지만 모델의 성능이 더 나아지는 것이라고 장담할 수는 없음

(18) activation function

- 딥러닝 6요소
    - loss function
    - optimizer
    - activation function
    - back propagation
    - forward propagation
    - one-hot encoding
- activation function : 선형적인 데이터 흐름에 비선형성을 추가하는 것
    - 만약 한 layer를 f(x)라고 한다면, layer 3개를 선형적으로 놓았을 시 3f(x)가 된다. 계수를 무시한다고 가정하였을 때, 이는 출력값과 입력값이 동일해지므로 layer를 놓는 과정이 의미가 없어진다.
    - -> 따라서 각 layer에 activation, 즉 비선형성을 주어서 조금 더 복잡한 특징을 추출하기 위함으로 사용한다. (feature extraction을 더욱 잘 하기 위함이다.)
    - 종류
        - sigmoid
        - softmax
        - tanh
        - relu
        - selu
        - 계단 함수
        - ...

(19) pre-trained model

- 사전 학습의 장점과 특징

    - pre-trained model : 기존에 비슷한 domain에서 문제를 해결하기 위해 모델링한 모델
    - 사전 학습 : 누군가가 비슷한 domain의 문제를 해결하기 위하여 만든 pre-trained model을 이용하여 모델링하자
        - 성능을 더욱 높일 수 있으며, 빠른 학습이 가능하다, 학습이 용이해진다
        - 적은 데이터셋으로 학습할 때 overfitting을 예방 가능
- 순서
    - pretrained-model freezing : 기존 모델의 weight와 bias를 다시 학습할 필요는 없다
    - unfreezing -> fine tuning : 기존 모델은 이미 비슷한 domain을 해결하기 위하여 만들어진 모델이기 때문에 모델 파라미터가 이미 근삿값에 도달되어 있다. 따라서 unfreezing 후에 미세 조정하여 빠르게 학습한다.

(20) gradient vanishing, exploding

- gradient vanishing
    - layer가 많이 구성될 수록 back-propagation할 시에 계속 미분되어 0이 될 가능성이 존재
    - 해결법
        - 모델의 layer를 줄이거나
        - 모델의 파라미터 수를 줄이거나
        - ResNet과 같은 skip-connection 도입
            - 학습을 용이
            - gradient vanishing problem 해결
- gradient exploding

> 그레이디언트 소실(gradient vanishing)이라고 합니다. 반면, 경우에 따라서는 그레이디언트가 점점 커져 여러 층에서 비정상적으로 큰 가중치가 갱신되어 알고리즘이 발산하게 되는 현상도 발생할 수 있습니다. 이를 그레이디언트 폭발(gradient exploding)이라고 부르며, 순환 신경망(RNN)에서 주로 나타나는 문제입니다. 일반적으로 불안정한 그레이디언트는 심층 신경망의 훈련을 어렵게 만들며, 각 층마다 학습 속도가 다르게 변할 수 있습니다.

- gradient descendant는 기울기가 소실되고 있는 상태이지만, 기울기가 폭발적으로 증가할 수도 있다.
    - ex) LSTM에서 optimizer를 relu를 사용하면 기울기 폭발 가능성 있음 (그래서 tanh를 쓰는 것)
- gradient clap을 이용하여 해결 가능

(21) CNN

- CNN을 사용하는 이유
    - DNN은 데이터를 일렬로 flatten하여 대입 -> local feature를 추출할 수 없게 된다
    - 따라서 이미지를 통째로 집어넣을 수 있는 모델을 구축
    - 사용 목적
        - 이미지와 같은 부분적인 정보추출 시에는 DNN 부적합 -> convolution layer 사용
        - 이미지 부분에 filter 곱하여 convolution 연산을 수행 -> 중요한 부분만 feature extraction -> regression/classification 용이
        - 전체 정보가 한 뉴런에 들어가면 학습율이 떨어짐 -> 부분적 학습으로 각 이미지를 영역으로 seperate하여 학습하는 것이 중요
- stride : filter를 움직임
- filter값은 초기에 random으로 정해지고, 학습이 되면서 최적값으로 update됨 (back-propagation 통하여 각 filter의 component가 결정)

- pooling의 의미
  - 정보 요약
  - 평행 이동 불변
- pooling의 장점 : 이동에 대하여 둔감하게 하는 것

(22) ResNet

- 목적
  - 잔차 학습 (residual learning) 도입하여 학습 용이하게 함
  - gradient vanishing problem 해결
- skip connection
  - gradient vanishing problem을 해결하기 위해 이전의 정보를 반영한다 -> 입력과 이전 것의 차이를 구하여 (잔차) 사전 정보가 있기 때문에 학습이 용이해진다
  - -> layer를 더욱 많이 늘릴 수 있게 된다

(23) inception

- GoogLeNet (inception model)
  - 여러 크기의 특성맵을 출력하는 합성곱 층을 구성
  - ```
    1x1 커널의 합성곱 층은 인셉션 모듈에서 중요한 역할을 합니다. 겉보기에는 한 번에 하나의 픽셀만 처리하기 때문에 공간적인 패턴을 잡지 못할 것처럼 보일 수 있지만, 실제로는 세 가지 주요 목적을 가지고 있습니다:

    1. **깊이 차원의 패턴 감지**: 1x1 합성곱층은 공간상의 패턴을 잡을 수 없지만, 채널(깊이) 차원을 따라 놓인 패턴을 잡을 수 있습니다. 즉, 채널 간 상관관계를 학습하는 데 유용합니다.

    2. **차원 축소**: 이 층은 입력보다 더 적은 특성 맵을 출력하므로, 병목층(bottleneck layer) 역할을 합니다. 차원을 줄이는 동시에 연산 비용과 파라미터 개수를 줄여 훈련 속도를 높이고, 과적합을 줄여 일반화 성능을 향상시키는 데 기여합니다.

    3. **복잡한 패턴 감지**: 1x1 합성곱층은 더 복잡한 패턴을 감지하기 위한 도구로 사용됩니다. 1x1 합성곱층과 3x3 또는 5x5 합성곱층의 쌍은 더 복잡한 패턴을 감지할 수 있는 하나의 강력한 합성곱층처럼 작동합니다. 이는 두 개의 층을 가진 신경망이 이미지를 훑는 것과 같은 방식으로, 더욱 정교한 특성을 학습할 수 있게 해줍니다.

    따라서, 1x1 합성곱층은 단순해 보일 수 있지만, 깊이 차원의 패턴을 감지하고, 차원을 축소하며, 더 복잡한 패턴을 감지하는 데 중요한 역할을 합니다.
    ```

(24) SENet

- SENet : 채널에 대한 중요도 매기고 학습 용이 (기존 CNN에서의 채널의 중요도는 모두 동일하였다 -> SENet으로 중요도 매기고 중요한 채널만 온전히 학습하자 -> 그래서 light-weighted) - 채널 중요도 추출은 pooling을 이용 - channel attention - **그러면 중요한 것만 집중하자 -> attention이 등장!**
- Variant CNN의 목적
  - layer을 쌓으면서 gradient vanishing problem 해결
  - layer을 쌓으면 parameter 갯수가 많아지는데 이걸 light-weighted (경량화)
    ```

(25) Xception, depthwise convolution

- Xception
    - GoogLeNet + 깊이별 분리 합성곱층
- 채널별로 다른 필터를 사용하자
    - ```
      **깊이별 분리 합성곱층(depthwise separable convolution)**이란, 일반 합성곱 연산을 두 단계로 나누는 방식을 의미합니다:
      1. **Depthwise Convolution**: 채널별로 개별적인 합성곱을 적용하여 공간 차원에서 패턴을 감지합니다.
      2. **Pointwise Convolution (1x1 Convolution)**: 1x1 합성곱을 사용하여 채널 간의 상호작용을 학습합니다.

      이 방식을 통해 **연산량을 줄이고** **모델의 효율성을 극대화**할 수 있습니다. 즉, 더 적은 계산으로도 복잡한 패턴을 감지할 수 있게 됩니다. 이러한 구조는 ResNet의 잔차 연결(residual connections)과 인셉션 모듈의 특징을 결합하여 **성능과 효율성**을 모두 강화시킨 모델입니다.

      Inception-v4에서도 이러한 분리 합성곱층을 사용하여 더 나은 성능을 달성하며, 동시에 모델의 복잡도를 조절하고 연산 비용을 줄일 수 있습니다.
      ```

(26) RNN, LSTM, GRU

- DNN, CNN은 데이터의 추세성을 반영한 것이 아니다. 이전의 시점을 반영하는 것도 아니다.
- 전에 있었던 데이터를 통하여 다음 데이터를 예측하는 모델 (이전 시점을 반영)
    - 시계열 데이터, 주식 데이터, EMG, HARTH...
    - 어제 시장가를 확인한 후에 오늘 매각할지 매수할지를 결정하는 것처럼
- 순환 데이터
    - 순환 데이터를 고려할 때, regression인 경우엔 window 수 + 다음 수로 데이터를 만들면 됨.
    - classification이면 class의 갯수가 많은 것 or 맨 끝의 것
        - 0 1 1 1
        - 많은 것이 1, 끝의 것이 1
- RNN의 단점
    - 미분값의 손실 (기울기 소실) -> 문장이 길어지면 기울기 소실이 일어남 -> LSTM의 등장
- LSTM
    - short term뿐만이 아닌 long term까지 학습
        - short term은 현재 학습하는 시계열의 시간 t
        - long term은 여태껏 학습해온 시계열들의 정보 (cell state / 중요한 정보만)
      ```

    - RNN은 layer가 많이 중첩될 수록 (cell이 많아질 수록) gradient vanishing 혹은 gradient exploding 문제가 발생한다.
        - 당연히 그럴만도 한게, 학습이 진행되면서 이전의 정보는 점차 희석되기 마련이다.
        - 따라서 가장 최근에 들어온 데이터셋에 대한 정보는 또렷히 학습된다.
            - -> **LSTM과 GRU가 도입된다.**
        - Long Short term memory : memory cell (cell state)의 도입으로 인하여 이전 문제점을 해결
        - LSTM의 구조
            - (1) forget gate : 이전 hidden state의 일부를 까먹게 함
                - 중요한 정보만 남긴다
                - sigmoid랑 연산하여 특정 확률을 cell state에 남길 것인가, 남기지 않을 것인가
            - (2) input gate : 새로운 정보가 cell state에 더해짐 (현재 입력 정보를 반영)
            - (3) output gate
                - 결과를 출력
                - 다음 cell에 hidden state, cell state를 넘겨줌
    - GRU
        - LSTM과 비슷한데, gate를 하나 덜 사용한다. 하지만 LSTM과 비슷한 성능을 보이기에 효율적이다.

- DNN, CNN, RNN, LSTM 흐름 정리
    - DNN의 단점 : local feature인 공간적 특성을 추출하지 못한다
    - CNN의 단점 : 오직 현재 입력만 고려한다. (current state / 앞뒤 문맥을 파악할 수가 없다)
    - RNN의 단점 : 문장이 길어지면 gradient vanishing (깊어질수록 손실이 일어난다)
    - LSTM이 등장
        - LSTM도 길면 gradient descent가 발생할 수 있다. -> attention 등장

(27) transformer

- "Attention is all you need"
    - 모델은 중요한 부분에 더 집중하고, 덜 중요한 부분은 무시하여, 효율적으로 정보를 처리
    - 중요 요소
        - attention vs transformer
            - attention은 정보가 sequence하게 입력됨
            - transformer는 한 뭉텅이가 하나로 입력됨 -> 순서가 없음
            - 예시) 예를 들어서 사과는 맛있는 과일이다 이러면
                - 사과는이라는 단어와 맛있는, 과일이다 이것의 상호관계에 대한 attention score를 계산하게 됨
                - 연관성이 얼마나 밀접한가를 중점
                - 각 단어마다 당연히 시간이 오래 걸림 -> 한꺼번에 병렬적으로 구하자 : multi-head attention
        - 임베딩 하는 이유

            - 모델에서 관계를 끊어주기 위함
            - 고차원 벡터 -> 저차원 벡터

        - positional embedding

            - 나 너 돈줘, 너 나 돈줘 <- 나, 너의 위치가 달라지기만 해도 의미가 달라짐
            - 위치에 따라 의미가 달라진다 -> 각 단어를 숫자로 변환하는 임베딩이 필요하며, 그 단어의 위치까지 기억해야하는 상황이 일어난 것임

        - transformer에서 중요한 4가지

            - embedding
            - positional embedding
            - multi-head attention
            - transformer

    - RNN과 LSTM의 입력과 출력이 고정되어 있어서 출력 sequence의 길이가 유동적이고 미리 알 수 없는 경우엔 적합하지 않음 (예를 들어서 GPT. 단어의 길이가 고정적이지 않음) -> seq2seq

        - decoder-encoder 구조로 되어있음

(28) scaled dot-product attention

- 이유 : attention score가 하나가 너무 커져서 scaling하기 위함
- 유사도 점수를 scale로 조정한다

(29) AE

- PCA vs AE (auto-encoder)

    - 입력과 출력이 유사하게 되도록
    - 둘 다 encoder임 -> 결국엔 encoder로 latent space를 만들고 DNN 붙여서 출력해낼 수 있음
    - AE를 사용하는 목적 : dimensionality reduction + 정보의 압축
        - PCA : 주성분 분석, eigenvalue를 이용하여 축소시킴 (PCA 전과 PCA 후를 비교하였을 때 무조건 PCA하여서 성능이 좋지 않을 수 있다)
        - AE : encoder-decoder 방식
            - 입력 x -> encoder로 latent space를 만듦 (데이터를 압축한 버전)
            - -> latent space를 보고 decoder로 다시 복호화 -> 출력 $x hat$
            - reconstruction error : $x hat - x$ (MSE, MAE, RMSE, RMAE)
            - 만약에 강아지 사진으로 AE를 학습시켰을 때 고양이 사진이 들어오면 이상치라고 반응 -> AE로 anomaly detection할 수 있게 됨
                - 차이가 크다 -> 에러가 난다
                - MSE, MAE를 reconstruction error로 많이 사용
            - encoder가 제일 중요함 (latent space를 만드는게 목적이니깐)

- linear AE : encoder와 decoder의 node가 대칭 (8, 4, 2, 2, 4, 8)

- non-linear AE : encoder와 decoder의 node가 비대칭 (16, 8, 4, 6, 12, 16)

- 신기한건 non-linear AE가 성능이 더 좋을 수도 있음

(30) VAE

- encoder의 목적 : 어떤 특징들을 압축하고 정보 추출 (핵심적, latent space)
    - 입력 데이터를 고차원 공간에서 저차원 벡터로 변환하는 역할을 수행
    - 텍스트, 이미지, 음성과 같은 데이터를 압축하여 핵심적인 정보를 저차원으로 표현
- decoder의 목적 : Encoder가 추출한 저차원 벡터(잠재 공간)를 다시 원래의 고차원 데이터 형식으로 복원
- VAE vs AE (차이점)
    - AE : latent vector를 추출함,
    - VAE : latent space를 추출함,
    - 결국엔 생성형 AI은 입력 데이터로부터 latent vector나 latent space를 추출하여, 그 latent space로부터 새로운 데이터를 생성
- likelihood (https://data-scientist-brian-kim.tistory.com/91)
    - 확률 : 가능성. 어떤 event가 일어날 가능성
        - 분포로부터 데이터
    - likelihood는 데이터로부터 분포를!
    - MLE : Maximum Likelihood
        - 데이터는 확률분포로 변할 수 있다. 데이터를 통해 확률분포를 예측할 수 있다. -> likelihood
- VAE의 loss == reconstruction error + KL Divergence (두개의 분포의 차이)
    - 두 분포를 최소화하자
        - 실제 값이랑 예측 값이랑 최대한 가깝게 하자
    - 원래 데이터를 가우시안 분포로 가정하는 것. (N(0, 1)) 이러면 평균과 분산을 알면 샘플링 쉬움. 원본 데이터로부터 샘플링.
    - 결국엔 정규분포와 input의 분포를 이용하여 variational inference (변분추론)하여 근삿값을 찾는 것
    - 재파라미터 트릭
    - ELBO

(31) GAN

- GAN, VAE의 목적 : 원본 데이터로부터 새로운 데이터의 생성
    - 코의 길이, 눈의 모양 등의 다변수 확률분포를 찾아서 새로운 걸 만들자 (결국에 아래 사진에서 generative가 확률분포를 찾는다는 것)
    - 기존 확률분포와 생성된 확률분포를 가능하면 비슷하게 하자
        - 결국에 모델은 분포를 학습하는 것
        - 실제 분포와 노이즈 분포를 계속 학습하면서 실제와 비슷하게 하자
- 아래는 GAN의 component structure
    - generator : noise를 진짜처럼
    - discriminator : 실제 이미지, 가짜 이미지를 구별
- mode collapse
    - 반대로 Generator가 지나치게 잘 학습되면, Discriminator가 생성된 이미지를 효과적으로 구분하지 못하게 되고, Generator는 특정한 종류의 이미지만 계속 생성하게 됨
        - 사과와 자두
    - 원인 1) gradient vanishing problem
    - 원인 2) data의 bias
    - 원인 3) 너무 민감함