

NLP팀

2팀

박상훈
곽동길
박윤아
김수진
신민서

INDEX

1. 가중치 초기화

4. 데이터 증강

2. 드롭아웃

5. 파인튜닝

3. 정규화

6. 임베딩

1

가중치 초기화

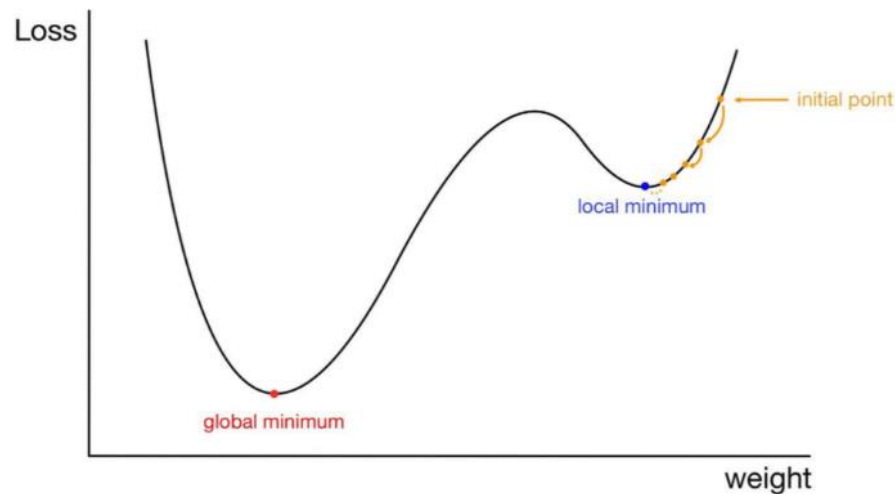
가중치 초기화의 중요성

초기 가중치 설정에 따라 최종 Loss가 달라지고

기울기 소실 및 폭발 문제가 발생할 수 있음



초기값은 모델 성능에 유의미한 차이를 줄 수 있음



1

가중치 초기화

가중치 초기화의 중요성

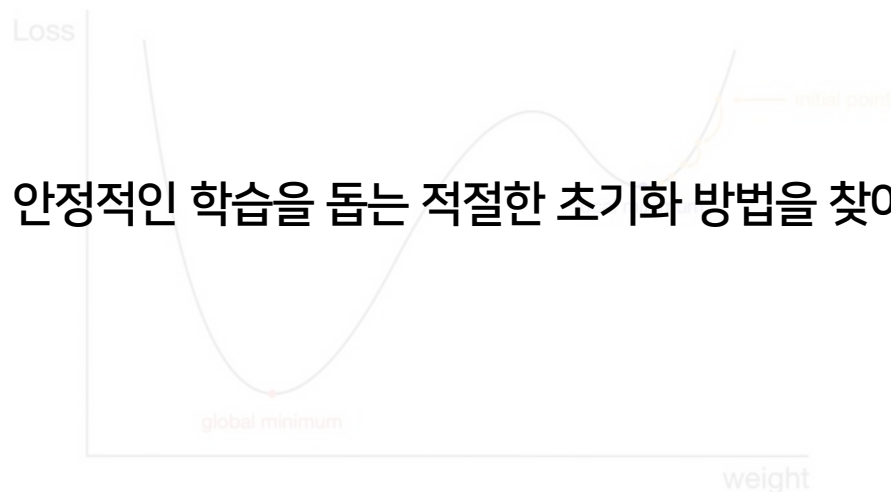
초기 가중치 설정에 따라 최종 Loss가 달라지고

기울기 소실 및 폭발 문제가 발생할 수 있음



초기값은 모델 성능에 유의미한 차이를 줄 수 있음

모델의 안정적인 학습을 돕는 적절한 초기화 방법을 찾아보자!



가중치 초기화 방법

Zero Initialization

모든 가중치를 0으로 초기화하는 방법

Random Initialization

정규분포 혹은 균일분포를 이용하여 랜덤으로 가중치를 초기화하는 방법

Xavier / He Initialization

정규분포 혹은 균일분포를 이용해 가중치를 랜덤으로 초기화되
인접 레이어의 노드 수에 따른 **유동적인 분산을 사용하는 방법**

Zero Initialization

Zero Initialization

모든 가중치를 0으로 초기화하는 방법

각 층의 모든 가중치 값이 동일하게 업데이트 되기 때문에

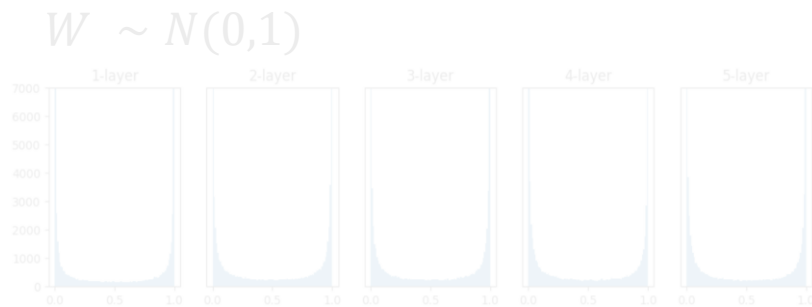
딥러닝 학습에서 좋은 전략이 아님

Random Initialization

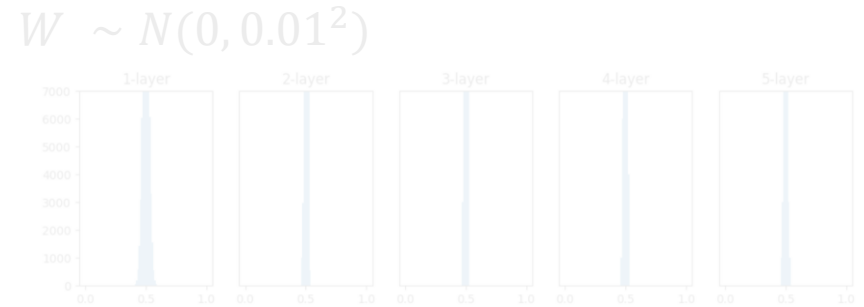
Random Initialization

정규분포 혹은 균일분포를 이용하여 랜덤으로 가중치를 초기화하는 방법

<Sigmoid를 활성화 함수로 쓰는 신경망에 적용한 경우>



레이어를 통과할수록 출력값이 0과 1에 치우침



출력값이 0.5에 몰리는 양상

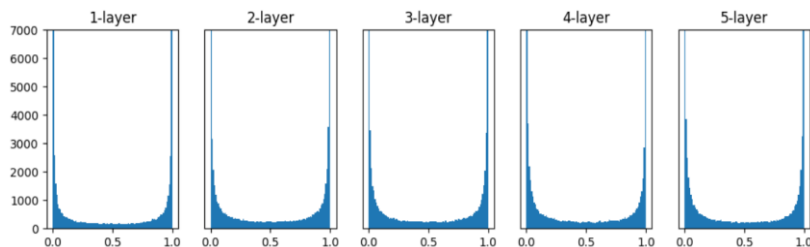
Random Initialization

Random Initialization

정규분포 혹은 균일분포를 이용하여 랜덤으로 가중치를 초기화하는 방법

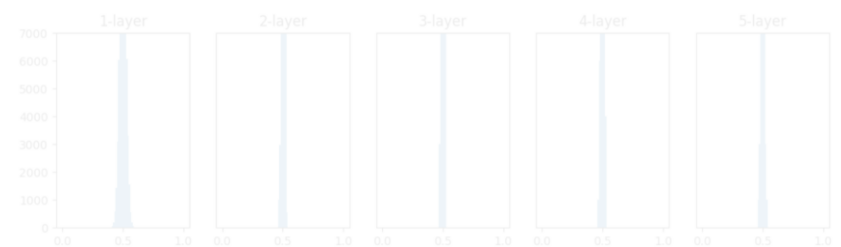
<Sigmoid를 활성화 함수로 쓰는 신경망에 적용한 경우>

$$W \sim N(0,1)$$



레이어를 통과할수록 출력값이 0과 1에 치우침

$$W \sim N(0, 0.01^2)$$



출력값이 0.5에 몰리는 양상

1

가중치 초기화

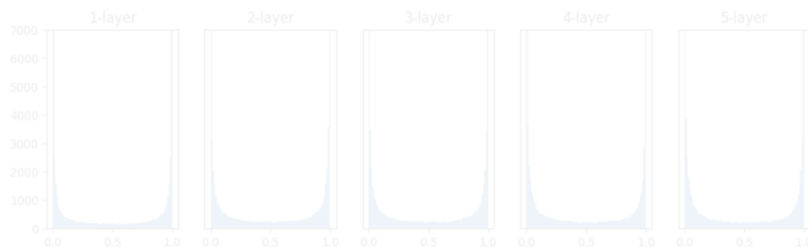
Random Initialization

Random Initialization

정규분포 혹은 균일분포를 이용하여 랜덤으로 가중치를 초기화하는 방법

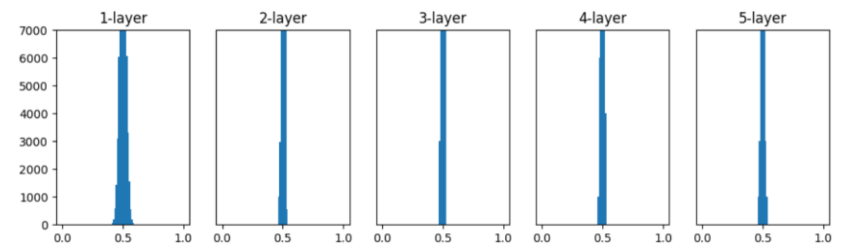
<Sigmoid를 활성화 함수로 쓰는 신경망에 적용한 경우>

$$W \sim N(0,1)$$



레이어를 통과할수록 출력값이 0과 1에 치우침

$$W \sim N(0, 0.01^2)$$



출력값이 0.5에 몰리는 양상

1

가중치 초기화

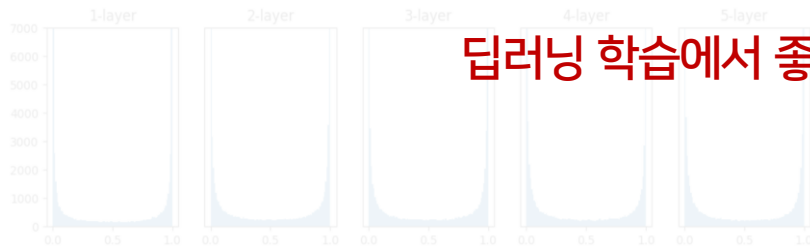
Random Initialization

Random Initialization

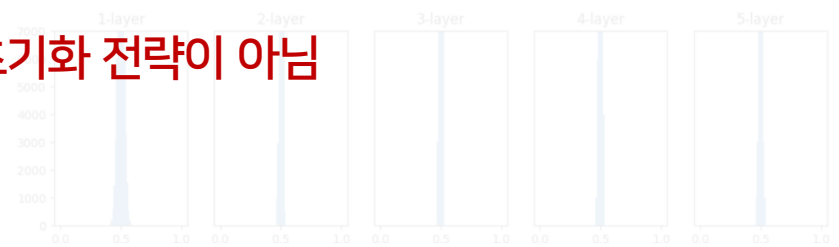
정규분포 혹은 균일분포를 이용하여 랜덤으로 가중치를 초기화하는 방법

$W \sim N(0,1)$ 각 층의 모든 Output 값이 고르게 유지되지 않기 때문에

딥러닝 학습에서 좋은 초기화 전략이 아님



레이어를 통과할수록 출력값이 0과 1에 치우침



출력값이 0.5에 몰리는 양상

Xavier Initialization

Xavier Initialization

정규분포 혹은 균일분포를 이용해 가중치를 랜덤으로 초기화하되
인접 레이어의 노드 수(n_l)에 따른 **유동적인 분산을 사용하는 방법**

Xavier Initialization

Xavier Initialization

정규분포 혹은 균일분포를 이용해 가중치를 랜덤으로 초기화하되
인접 레이어의 노드 수(n_l)에 따른 **유동적인 분산을 사용하는 방법**

연구진들은 각 레이어의 Output과 가중치 Gradient의 분산 안정화를 위한
레이어별 가중치 분산의 2가지 조건을 여러 수학적 가정으로부터 도출해냄



$$\forall l : Var(W_l) = \frac{1}{n_{l-1}}, Var(W_l) = \frac{1}{n_l}$$

Xavier Initialization

Xavier Initialization

정규분포 혹은 균일분포를 이용해 가중치를 랜덤으로 초기화하되
인접 레이어의 노드 수(n_l)에 따른 **유동적인 분산을 사용하는 방법**

두 인접 레이어의 노드 수가 다르다면 분산이 하나의 값으로 결정될 수 없음
이를 위해 Xavier는 **노드 수의 평균값을 사용하는** 방식을 절충안으로 제시



$$\text{Var}(W_l) = \frac{2}{n_{l-1} + n_l}$$

Xavier Initialization

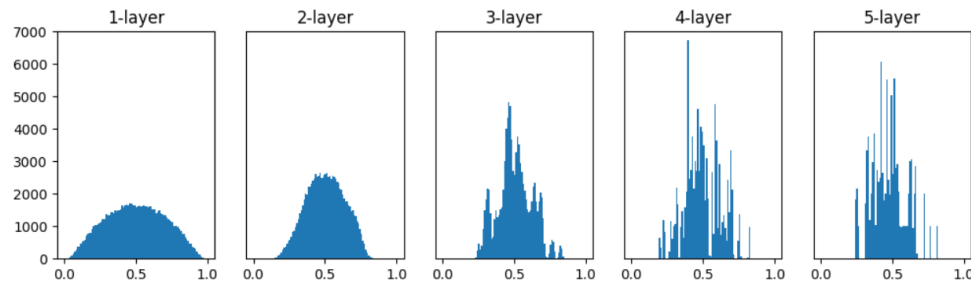
Xavier Initialization

< W_l 는 l 번째 레이어와 $l - 1$ 번째 레이어를 잇는 가중치 집합>

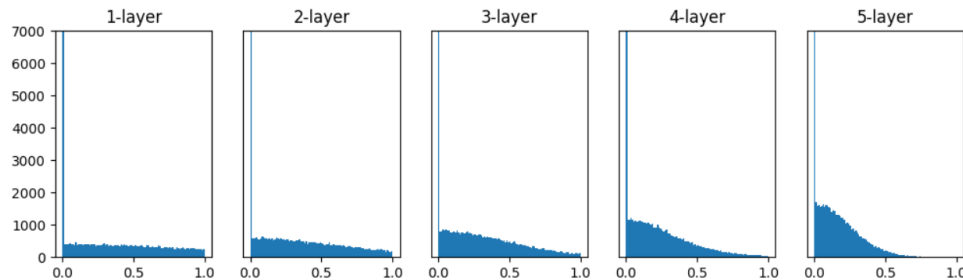
[Xavier Normal init] $W_l \sim N(0, Var(W_l))$, $Var(W_l) = \frac{2}{n_{l-1} + n_l}$

[Xavier Uniform init] $W_l \sim U\left(-\sqrt{\frac{6}{n_{l-1} + n_l}}, \sqrt{\frac{6}{n_{l-1} + n_l}}\right)$

Xavier Initialization



Sigmoid, tanh 활성화함수를 사용하는 신경망에는 효과적



ReLU 활성화함수를 쓸 때는
출력값이 0으로 쏠리는 문제 발생

He Initialization

He Initialization

정규분포 혹은 균일분포를 이용해 가중치를 랜덤으로 초기화하되
인접 레이어의 노드 수(n_l)에 따른 **유동적인 분산을 사용하는 방법**

<Xaiver>

$$\forall l : Var(W_l) = \frac{1}{n_{l-1}}, Var(W_l) = \frac{1}{n_l}$$

<He>

$$\forall l : Var(W_l) = \frac{2}{n_{l-1}}, Var(W_l) = \frac{2}{n_l}$$

Kalming He는 ReLU의 특성에 맞게 가중치 분산의 조건을 수정

He Initialization

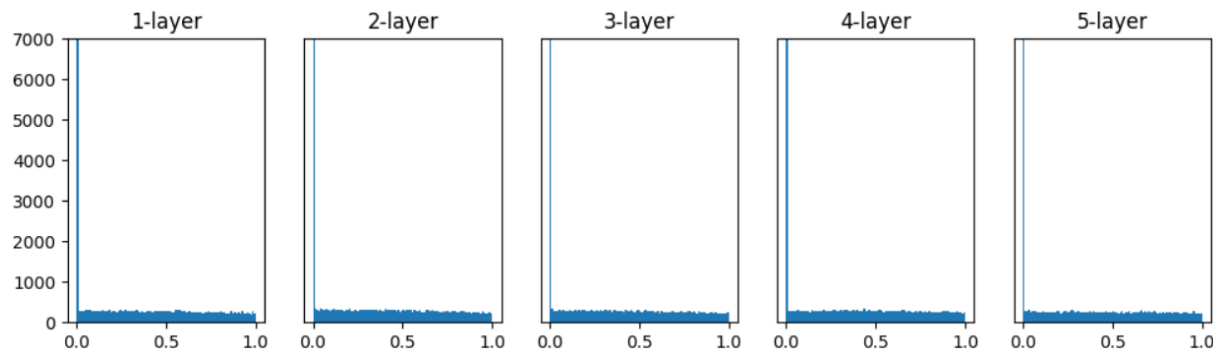
He Initialization

[He Normal init] $W_l \sim N(0, Var(W_l)) , Var(W_l) = \frac{2}{n_{l-1}}$

[He Uniform init] $W_l \sim U\left(-\sqrt{\frac{6}{n_{l-1}}}, \sqrt{\frac{6}{n_{l-1}}}\right)$

또한 가중치 분산 조건 2가지 중 하나만 만족시켜도 된다고 판단하여
한 쪽 레이어의 노드 수를 사용하는 방안을 제시

He Initialization



ReLU 활성화함수를 사용하는 경우에도 층별 Output의 분산이
일정하게 유지되는 것을 확인



ReLU를 사용하는 심층신경망의 안정적인 학습에 기여

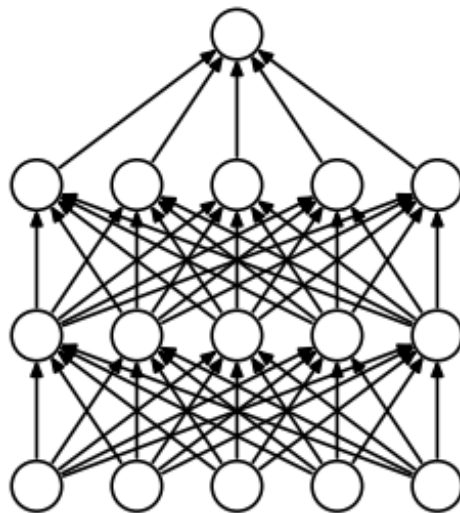
2

드롭아웃

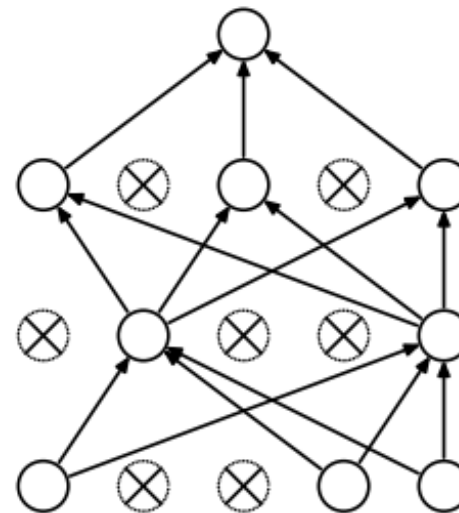
Dropout

Dropout

학습 과정에서 신경망의 뉴런을 확률적으로 제외시켜 과적합을 방지하는 기법



(a) Standard Neural Net



(b) After applying dropout.

Dropout

학습 : 순전파 과정

Output 뉴런을 제외한 모든 뉴런들에 대해

$D \sim \text{Bernoulli}(p)$ 에서 표본 추출

$D=1$ 이면 해당 뉴런의 output 보존

$D=0$ 이면 0으로 고정

$$y_i^{(k)} = D \times f \left(\sum_j w_{ji}^{(k)} y_j^{(k-1)} + b_i^{(k)} \right) \quad P(D = d) = \begin{cases} p & (d = 1) \\ 1 - p & (d = 0) \end{cases}$$

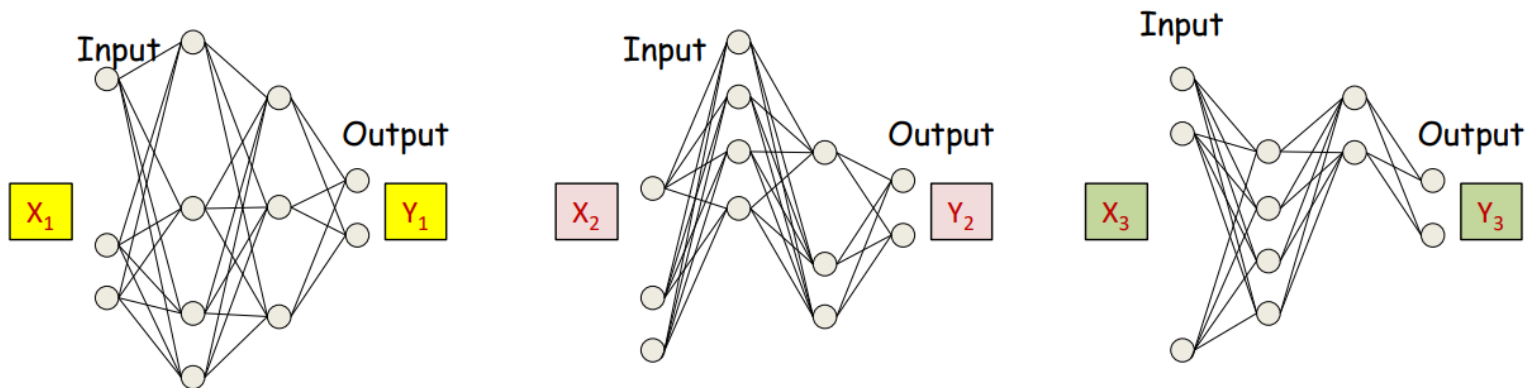


Sample 하나가 신경망에 들어올 때마다 Dropout을 독립적으로 수행

Dropout

학습 : 역전파 과정

비활성화된 뉴런과 연결된 가중치들의 Gradient는 역전파 과정에서 0으로 계산됨



하나의 sample에 대한 역전파 계산 진행 시 활성화된 가중치의 Gradient만 반영!

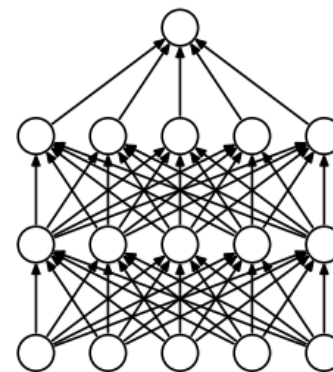
Dropout

추론 : 예측값 계산 과정

학습된 가중치 전체에 확률값 p 를 곱하는 **보정 작업** 후
새로운 데이터에 대한 예측 진행

$$W_{test} = p \times W_{train}$$

추론 과정에서는 Dropout 없이 모든 뉴런을 활용함



Dropout

과적합 방지 원리 : Approximate averaging

모든 sub-network로부터 얻은 output의 평균

≈ 모든 hidden unit들이 기댓값을 출력할 때의 신경망 output

= 가중치 보정이 된 신경망의 output

Dropout은 수많은 신경망을 앙상블하는 효과로 과적합 방지에 기여

Dropout의 효과는 평균값에 대한 근사의 정확도에 따라 달라짐

3

정규화

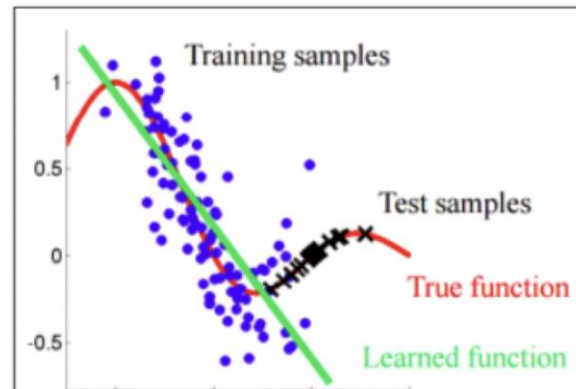
3

정규화 (Normalization)

공변량 변화 (Covariate Shift)

공변량 변화

입력 데이터의 분포가 **학습할 때와 테스트할 때 다르게 나타나는 상황**



공변량 변화 발생시 모델이 좋은 성능을 보이기 어려움

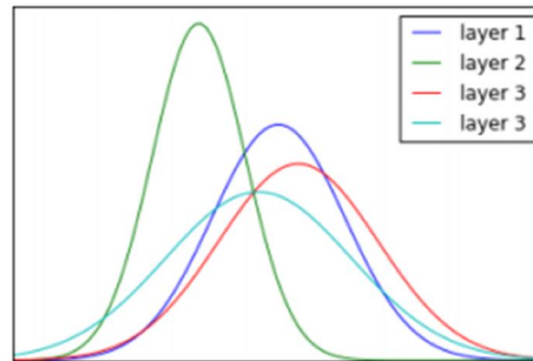
3

정규화 (Normalization)

내부 공변량 변화 (Internal Covariate Shift, ICS)

내부 공변량 변화

입력 데이터 분포의 변화가 **신경망 내부 각 층마다** 발생하는 상황



(b) Internal covariate shift



내부 공변량 변화는 층이 깊어질수록 심화되며
심층 신경망의 안정적인 학습을 위해 해결해야 할 문제!

3

정규화 (Normalization)

입력 데이터의 분포를 조정하는 방법

정규화 (Normalization)

모든 데이터의 **스케일을 동일**하게 만드는 작업

신경망 내부에서 적용되는 정규화 기법

Batch Normalization

입력(Batch) 단위로 정규화

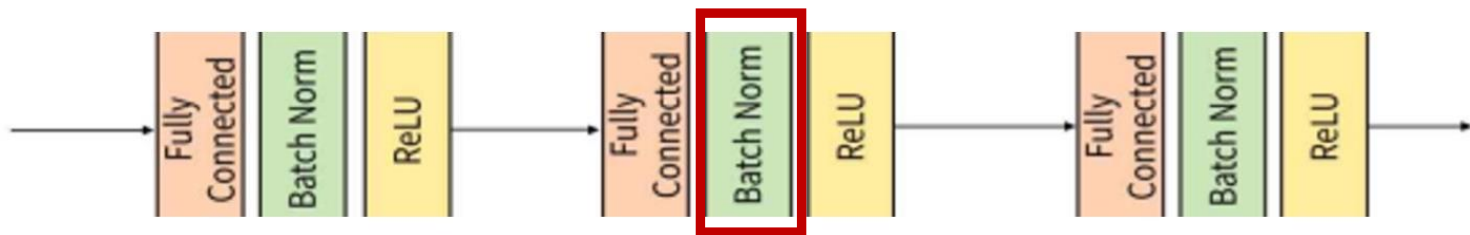
Layer Normalization

데이터 샘플 단위로 정규화

Batch Normalization

배치 정규화 (Batch Normalization)

신경망의 각 층에서 입력 데이터를 **Batch 단위 평균과 표준편차**로 정규화



입력값의 가중합에 편향을 더한 후 활성화 함수를 통과하기 전에 시행!

Batch Normalization

배치 정규화 (Batch Normalization)

신경망의 각 층에서 입력 데이터를 **Batch 단위 평균과 표준편차**로 정규화

하나의 뉴런에서 진행되는 배치 정규화 과정

$$\mu_B \leftarrow \frac{1}{m} \sum_{i=1}^m z_i \quad \sigma_B^2 \leftarrow \frac{1}{m} \sum_{i=1}^m (z_i - \mu_B)^2 \quad [1] \text{ Batch별 통계량 계산}$$

$$\hat{z}_i \leftarrow \frac{z_i - \mu_B}{\sqrt{\sigma_B^2 + \epsilon}} \quad u_i \leftarrow \gamma \hat{z}_i + \beta \quad [2] \text{ 정규화 후 분포 조정}$$

3

정규화(Normalization)



Batch Normalization의 장점

배치 정규화 (Batch Normalization)

- 👉 정확한 가중치 초기화의 중요성을 낮춤
- 👉 Gradient Vanishing을 완화해 학습 속도 향상
- 👉 Regularization 효과로 Dropout에 대한 의존성을 줄임



Batch Normalization의 단점

학습 과정에서 입력 데이터에 대한 평균과 분산을 구하기 쉽지만
예측 과정에서는 통계량을 정확히 계산할 수 없음

3

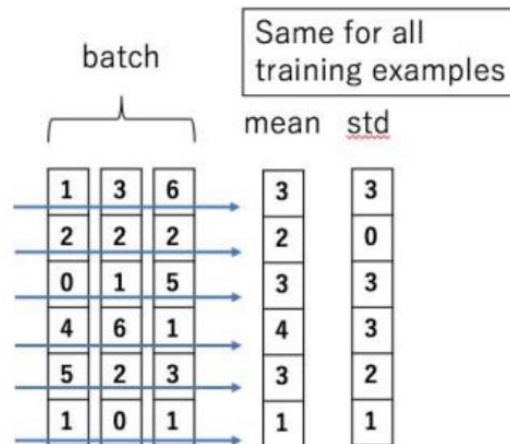
정규화 (Normalization)

Layer Normalization

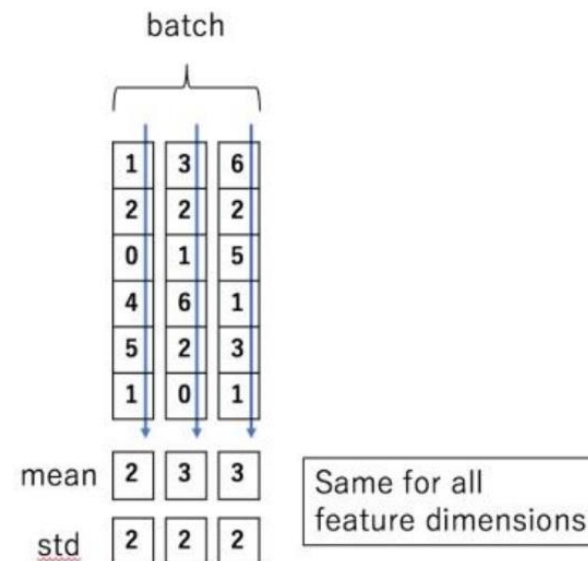
레이어 정규화 (Layer Normalization)

데이터 샘플 단위 표본평균과 표준편차로 정규화

Batch Normalization



Layer Normalization



Layer Normalization

레이어 정규화 (Layer Normalization)

데이터 샘플 단위 표본평균과 표준편차로 정규화

시퀀셜 데이터의 길이는 보통 일정하지 않음



Batch 내 input의 길이는 동일해야 하므로 Batch 크기가 1이 될 수 밖에 없음



일반적으로 Attention, RNN 기반 모델에서는 레이어 정규화가 더 많이 사용됨

4

데이터 증강

데이터 증강

데이터 증강

기존의 데이터를 이용하여 **새롭게 데이터를 창조**하는 기법

데이터 증강의 장점



모델 예측력 향상



데이터 수집 비용 감소



희귀한 사건에 대한 예측력 향상

데이터 증강은 일반적으로 컴퓨터 비전에서 많이 쓰임

자연어의 경우 순차 데이터이므로 의미 변형에 유의 해서 사용

데이터 증강 in 컴퓨터 비전

기하학적 변환

기존 이미지를 **Crop, Rotate, Contrast, Flip** 등의 변환을 통해 새로운 이미지를 생성하는 방법

색 변환

기존의 **RGB 색상**에 **다양한 변화**를 주어 새로운 이미지를 생성하는 방법

이미지 섞기 & 랜덤하게 자르기

여러 개의 **이미지**를 섞거나
랜덤하게 **픽셀을 지워** 새로운 이미지를 생성하는 방법

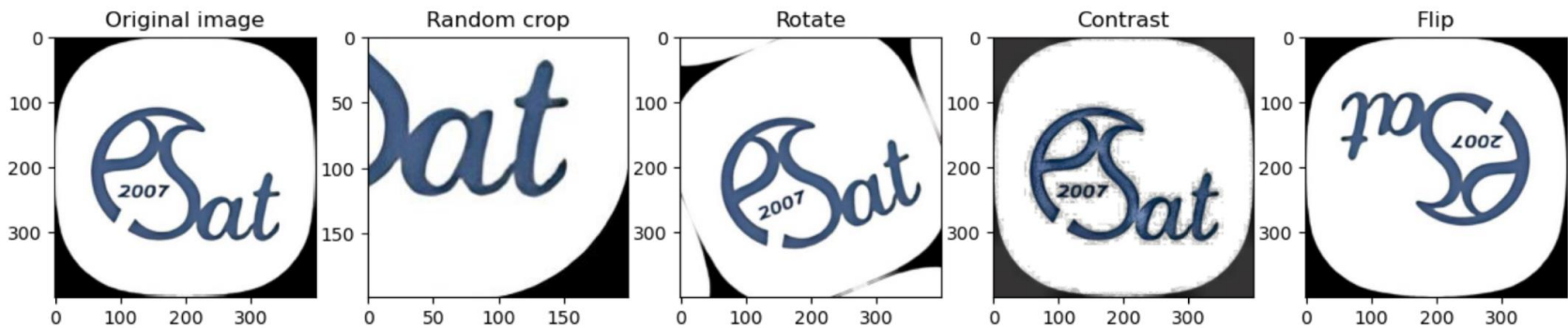
4

데이터 증강 (Data Augmentation)

데이터 증강 in 컴퓨터 비전

기하학적 변환

기존 이미지를 **Crop, Rotate, Contrast, Flip** 등의 변환을 통해 새로운 이미지를 생성하는 방법



변환 후 결과가 기존 데이터의 Label과 호환되어야 함!

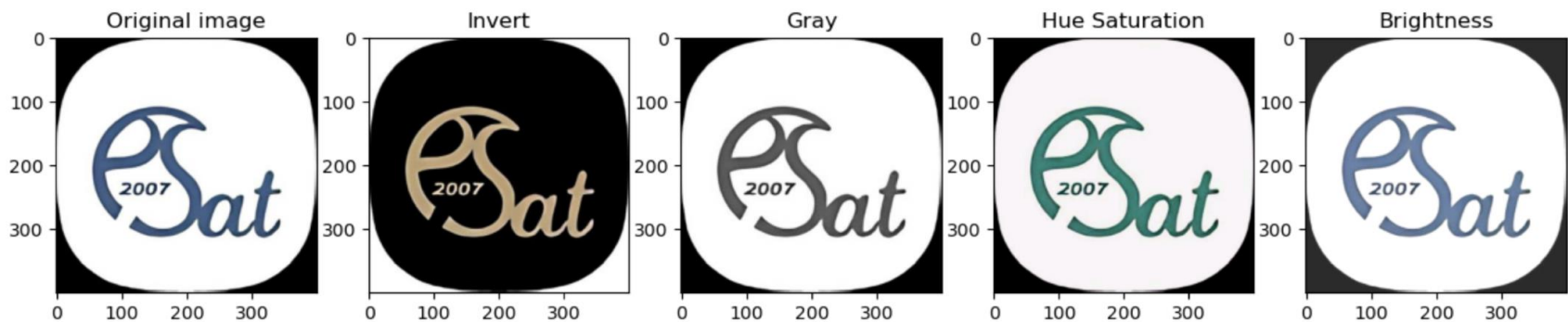
4

데이터 증강 (Data Augmentation)

데이터 증강 in 컴퓨터 비전

색 변환

기존의 **RGB 색상**에 **다양한 변화**를 주어
새로운 이미지를 생성하는 방법



4

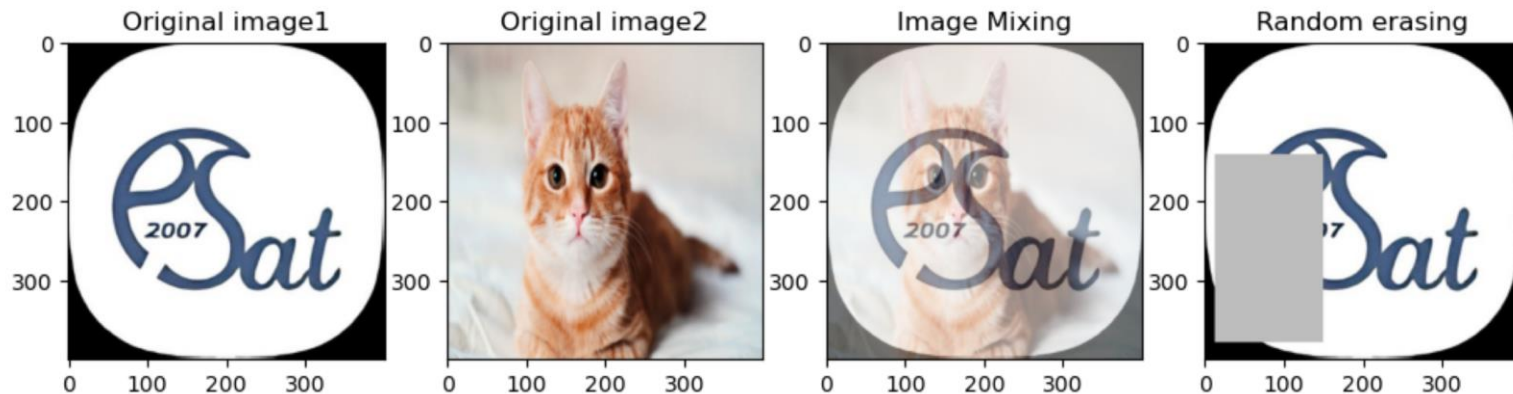
데이터 증강 (Data Augmentation)

데이터 증강 in 컴퓨터 비전

이미지 섞기 & 랜덤하게 자르기

여러 개의 이미지를 섞거나

랜덤하게 픽셀을 지워 새로운 이미지를 생성하는 방법



투명도를 의미하는 차원 'A'(alpha)를 추가하여 구현

데이터 증강 in 자연어 처리

역번역 (Back translation)

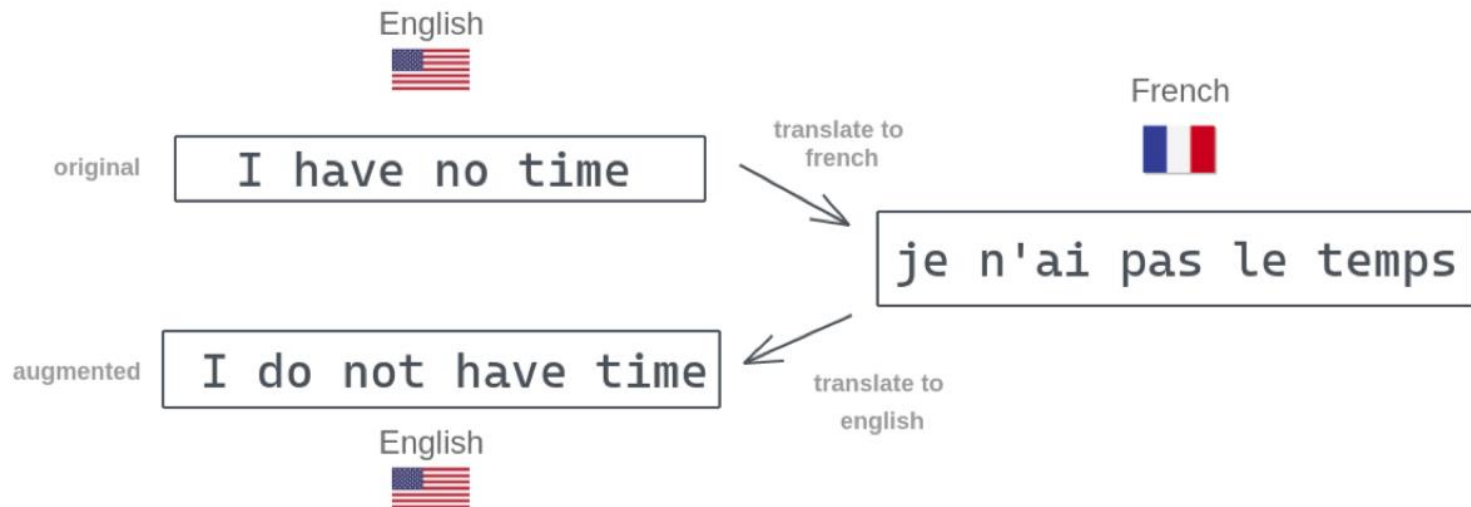
원본 문장을 **다른 언어로 번역** 후,
다시 원본 문장의 언어로 번역하여 새로운 데이터를 얻는 방법

EDA (Easy Data Augmentation)

텍스트 분류 문제를 위한 데이터 증강 방법
SR, RI, RS, RD 방법을 통해 새로운 데이터 생성

데이터 증강 in 자연어 처리

역번역(Back translation)

원본 문장을 **다른 언어로 번역** 후,**다시 원본 문장의 언어로 번역**하여 새로운 데이터를 얻는 방법

역번역을 통해 의미가 보존된 Augmented data 생성

데이터 증강 in 자연어 처리

EDA (Easy Data Augmentation)

텍스트 분류 문제를 위한 데이터 증강 방법

SR, RI, RS, RD 방법을 통해 새로운 데이터 생성

SR (Synonym Replacement)

단어를 무작위로 선택하여
동의어로 대체하는 방법

RI (Random Insertion)

무작위의 동의어를 임의의
위치에 삽입하는 방법

RS (Random Swap)

무작위로 두 단어를 골라
위치를 바꾸는 방법

RD (Random Deletion)

무작위로 단어를
삭제하는 방법

데이터 증강 in 자연어 처리

SR (Synonym Replacement)

Original sentence: *This **article** will focus on summarizing data augmentation **techniques** in NLP.*

Augmented sentence: *This **write-up** will focus on summarizing data augmentation **methods** in NLP.*

RI (Random Insertion)

Original sentence: *This **article** will focus on summarizing data augmentation **techniques** in NLP.*

Augmented sentence: *This article will focus on **write-up** summarizing data augmentation techniques in NLP **methods**.*

데이터 증강 in 자연어 처리



RS (Random Swap)

Original sentence: *This **article** will focus on summarizing data augmentation **techniques** in NLP.*

Augmented sentence: *This **techniques** will focus on summarizing data augmentation **article** in NLP.*

RD (Random Deletion)

Original sentence: *This article **will** focus on summarizing data augmentation **techniques** in NLP.*

Augmented sentence: *This article  focus on summarizing data augmentation  in NLP.*

5

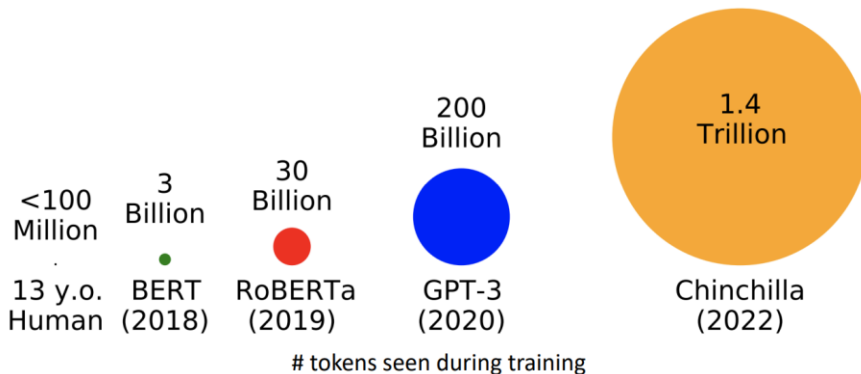
파인 튜닝

파인 튜닝 (Fine Tuning)



딥러닝 모델을 처음부터 학습시키고 문제에 적용하기는
비용이 많이 들고 복잡함

복잡한 모델은 매우 많은 수의 파라미터를 가지고 있음!



이러한 모델들은 이미 일반적인 문제에
적용하기 충분할 정도로 학습이 된 상태

각 대형언어모델(LLM)들이 학습한 토큰 수

파인 튜닝 (Fine Tuning)

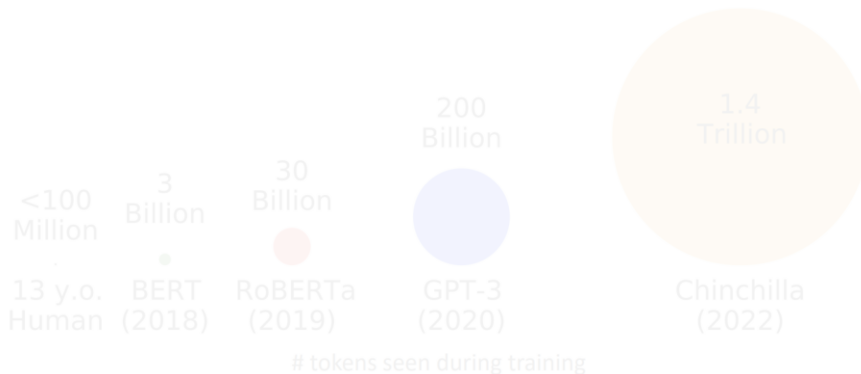


딥러닝 모델을 처음부터 학습시키고 문제에 적용하기는
비용이 많이 들고 복잡함



잘 훈련된 모델들을 빌려서 사용할 수는 없을까?

만약 사용할 수 있다면 어떻게 사용해야 할까?



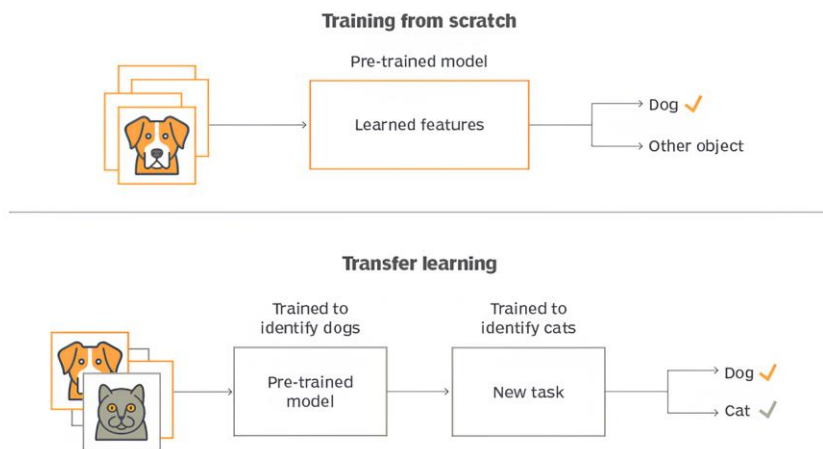
이러한 모델들은 이미 일반적인 문제에
적용하기 충분할 정도로 학습이 된 상태

각 대형언어모델(LLM)들이 학습함 토큰 수

파인 튜닝 (Fine Tuning) | 전이학습

전이학습 *Transfer Learning*

한 문제를 해결하면서 얻은 지식을 다른 관련 문제에 적용하는 것



처음부터 학습하는 이미지 처리 모델 (위),
전이 학습을 활용하는 모델 (아래)

(예시) 강아지를 분류하도록 사전 학습된
모델을 전이 학습하여 고양이 분류에 적용



처음부터 학습하는 것보다
더 적은 데이터와 비용으로 높은 성능을 얻음



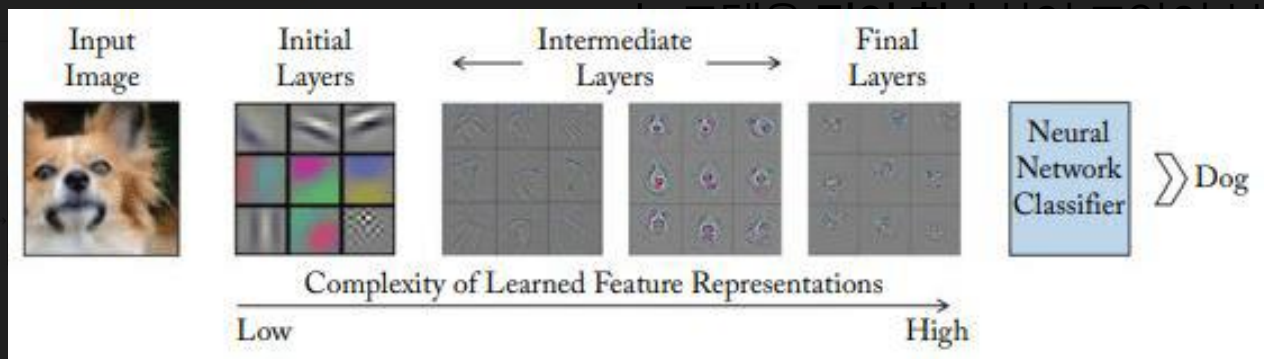
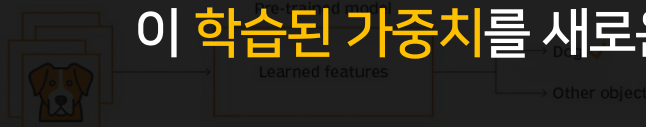
파인 튜닝 (Fine Tuning) 전이 학습이 효과적인 이유

전이학습 Transfer Learning 사전 학습 모델은 대규모 데이터로부터

이미 다양한 수준의 특징(feature)을 학습한 상태

신경망의 깊은 층에 있는 뉴런들은 복잡한 특징을 학습할 수 있으며,

이 학습된 가중치를 새로운 문제에 적용하여 사용할 수 있음



처음부터 학습한 것보다 적은 데이터와 비용으로 높은 성능을 얻음

전이 학습을 활용하는 모델 (아래)

더 적은 데이터와 비용으로 높은 성능을 얻음

파인 튜닝 (Fine Tuning) | 전이학습



Domain은 학습 데이터 feature의 분포,
Task는 Label의 종류와 관련 있는 개념

source domain	사전 학습 시 학습한 데이터 feature의 분포
target domain	새롭게 학습할 데이터 feature의 분포
source task	사전 학습 단계에서 해결한 문제
target task	새롭게 해결할 문제

파인 튜닝 (Fine Tuning) | 전이학습



Domain은 학습 데이터 feature의 분포,

Task는 Label의 종류와 관련 있는 개념

사전 학습 모델을 Target domain과 task에 맞게

변형해 좀 더 좋은 성능을 내게 할 수는 없을까?

source domain	사전 학습 시 학습한 데이터 feature의 분포
target domain	새롭게 학습할 데이터 feature의 분포
source task	사전 학습 단계에서 해결한 문제
target task	새롭게 해결할 문제

파인 튜닝 (Fine Tuning)

파인 튜닝 *Fine Tuning*

사전 학습된 모델의 가중치를 초기값으로 삼아,
새로운 데이터와 task에 맞게 모델을 다시 최적화하는 방법



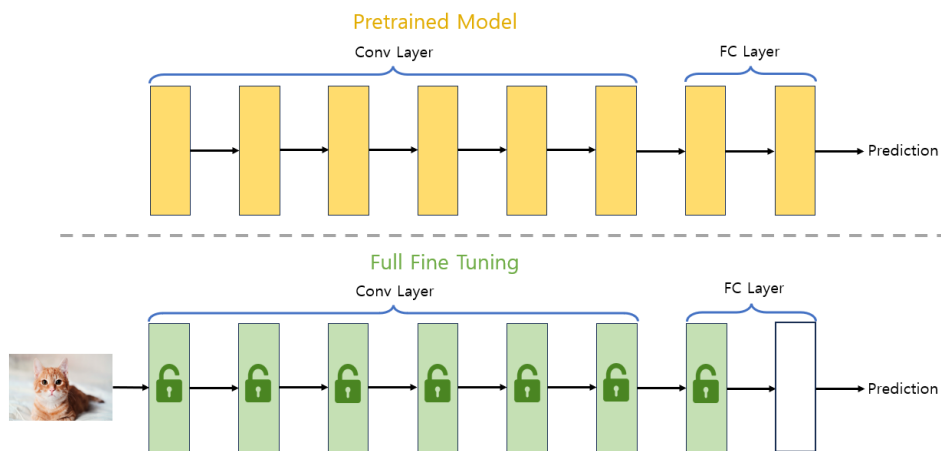
컴퓨터 비전에서 이미지를 분류하는 모델을 예시로 알아보자!



파인 튜닝 (Fine Tuning) | Full Fine Tuning

신경망 전체에 적용하기 (Full Fine Tuning)

전체 파라미터에 대해서 파인튜닝을 적용하는 방법



모든 가중치를 활용해 학습하기 때문에

Target task를 위해

가장 잘 최적화할 수 있는 방법

★ ★ 비교적 더 적은 데이터로 진행한다는 점에서 사전 학습과는 다름!

파인 튜닝 (Fine Tuning) | Full Fine Tuning



특정 도메인, task 관련 데이터로 모델을 직접 학습시키기 때문에
이전 모델보다 더 좋은 성능을 보임



여전히 매우 큰 모델의 모든 가중치를 업데이트하기 때문에
학습 비용이 높음



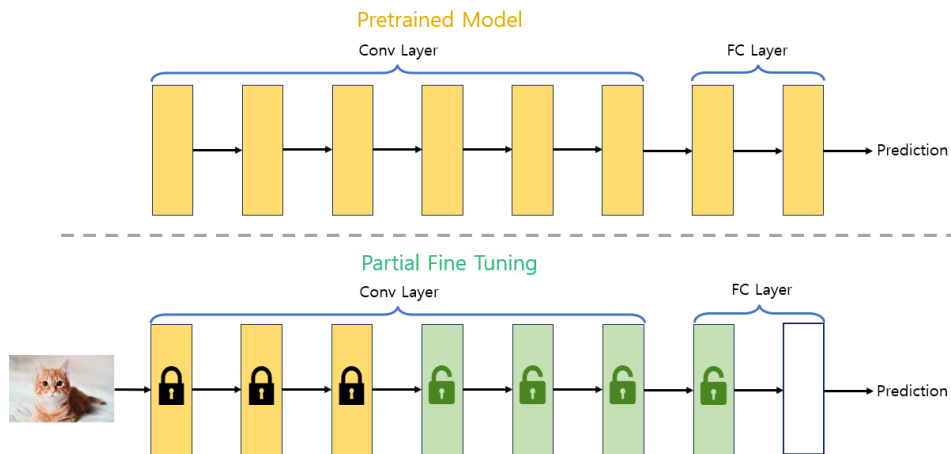
높은 수준의 GPU나 TPU가 요구됨
일반적인 컴퓨팅 수준으로는 제약이 있을 수 있음

파인 튜닝 (Fine Tuning) | Partial Fine Tuning

일부 레이어에 적용하기 (Partial Fine Tuning)

파라미터 일부는 고정하고 (Freezing)

나머지 파라미터 일부에 대해서 파인 튜닝을 적용하는 방법



더 적은 수의 파라미터를 업데이트하기
때문에 학습 비용이 더 적음

두 데이터 사이의 도메인이 다르더라도
부분적인 파라미터 업데이트를 통해
새로운 데이터에 어느 정도 적응할 수 있고,
결과적으로 정확도의 향상을 불러옴

파인 튜닝 (Fine Tuning) | Partial Fine Tuning



Full Fine Tuning에 비해서는 모델이 덜 유연하지만,
파라미터를 업데이트하기 때문에 새로운 도메인, task에 잘 적응할 수 있음



일부 파라미터만 업데이트를 진행하기 때문에
Full Fine Tuning에 비해 학습 비용이 저렴함

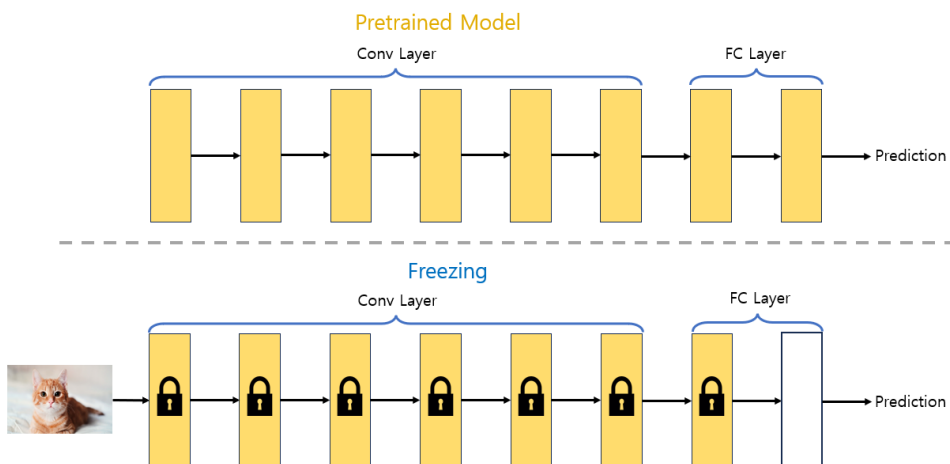


여전히 커다란 모델을 학습시키는 일이기 때문에
높은 수준의 하드웨어를 요구함

파인 튜닝 (Fine Tuning) | 모델 얼리기 (Freezing)

모델 얼리기 (Freezing)

마지막 완전 연결층을 제외한 모든 가중치를 고정하는 방법



소수의 파라미터만 다시 최적화하기 때문에
Target 데이터셋의 규모가 작은 경우
과적합을 피할 수 있음

마지막 완전 연결층은 완전히 새로 학습됨
가중치는 랜덤으로 초기화

새로운 문제로 넘어갈 때 클래스의 종류와 수가 바뀌어
기존 가중치가 더 이상 유효하지 않기 때문

파인 튜닝 (Fine Tuning) | 모델 얼리기 (Freezing)



앞선 두 방법에 비해서는 덜 유연하지만,
비슷한 도메인, task에 대해서는 사전 학습 모델보다 좋은 성능을 보임



사실상 MLP층 1개에 대해서만 학습하는 방법
앞선 두 방법에 비해 학습 비용이 매우 적음



마지막 완전 연결층만 학습하기 때문에
하드웨어의 부담이 적음

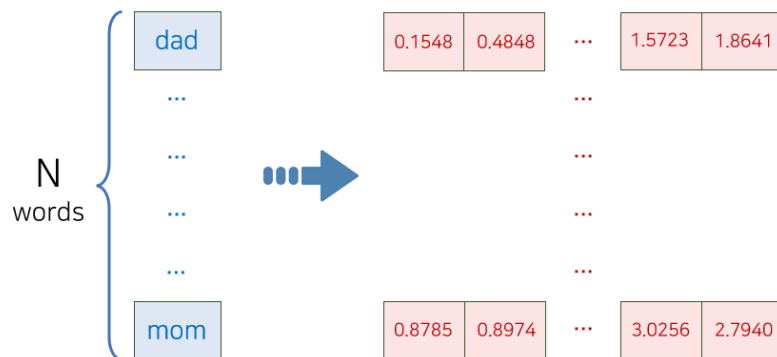
6

임베딩

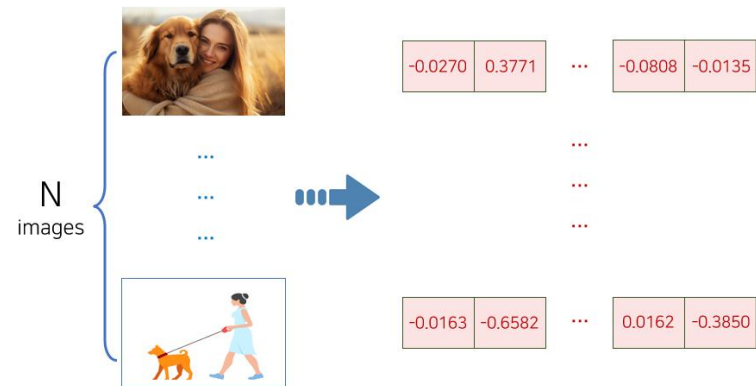
임베딩 (Embedding)

임베딩 *Embedding*

Text, Image, User, Item 같은 이산적인 객체들을
저차원의 연속적인 벡터로 변환하는 과정 (또는 결과물)



N개의 단어를 임베딩한 결과

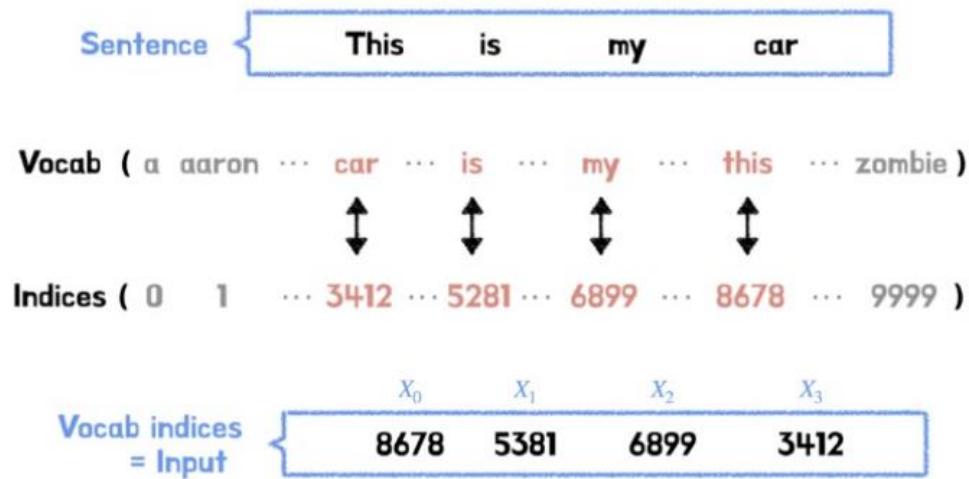


N개의 이미지를 임베딩한 결과

임베딩은 객체를 실수 벡터에 대응시키는 함수(규칙) 같은 개념!

임베딩 접근하기 (1) Embedding Layer

모델에 임베딩을 학습할 Embedding Layer를 추가할 수 있음
 해당 Layer의 입력과 출력은 데이터 객체, 임베딩 벡터가 됨



객체를 객체 집합에서의
인덱스와 대응시킬 수 있음



인덱스 값을 입력값으로 대신 사용

임베딩 접근하기 (1) Embedding Layer

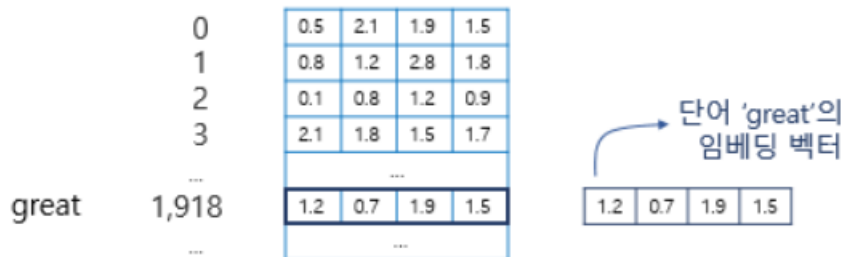


입출력 구조 설계하기

Embedding layer의 가중치 행렬 W_{Em} 를 행의 수가 단어 집합의 크기, 열의 수가 임베딩 벡터의 차원과 같은 실수 행렬로 정의

W_{Em} 의 i 번째 행벡터에 접근하여 얻어지는 벡터를 단어 집합 i 번째 원소의 임베딩 벡터라고 정의

Word → Integer → lookup Table → Embedding vector



훈련 과정에서 학습된다.

Pytorch - nn.Embedding()의 구조

임베딩 벡터에 접근하는 과정



W_{Em} 이라는 Lookup table에서
행을 조회하는 것

임베딩 접근하기 (1) Embedding Layer



입출력 구조 설계하기

Embedding layer의 가중치 행렬 W_{Em} 를 행의 수가 단어 집합의 크기, 열의 수가 임베딩 벡터의 차원과 같은 실수 행렬로 정의

W_{Em} 의 i 번째 행벡터에 접근하여 얻어지는 벡터를 단어 집합 i 번째 원소의 임베딩 벡터라고 정의

Word → Integer → lookup Table → Embedding vector



Pytorch - nn.Embedding()의 구조

임베딩의 학습

행렬 W_{Em} 의 값을 랜덤으로 초기화,
모델 학습 시 역전파를 통해
가중치 행렬 W_{Em} 도 통째로 학습시킴

임베딩 접근하기 (2) 사전 학습 모델

매우 많은 데이터로 사전 학습된 모델을 사용

텍스트	Word2Vec, Glove, Bert 등
이미지	CNN, ResNet, ViT 등

사전 학습된 모델이 제공하는 임베딩 결과를 사용해,
이를 직접 모델의 Embedding Layer에 입력하고,
모델 학습 시 Embedding Layer를 고정된 상태로 유지

임베딩 접근하기 (2) 사전 학습 모델

매우 많은 데이터로 사전 학습된 모델을 사용

임베딩 레이어의 가중치를 직접 학습시키면
해당 task에 더 적합한 임베딩 벡터를 만들 수 있지만,
소규모 데이터셋을 사용하는 경우 임베딩이 제대로 학습되지 않을 수 있음



사전 학습된 모델을 사용하는 것이 더 적절함!

데이터의 일치 여부에 따라 0 또는 1로 표시하는
One-Hot Encoding과 비교하여
임베딩을 사용하는 이유를 알아보자!

임베딩을 사용하는 이유

1,000,000

	-	apples	sour	monkeys	eat
1,000,000	apples	1	0	0	0
	sour	0	1	0	0

	monkeys	0	0	1	0
	eat	0	0	0	1

1,000,000

One-Hot Encoding 방법은
처리해야 할 단어가 늘어남에 따라
차원이 매우 커짐



심각한 공간 낭비
모두 표현하기 어려움

임베딩을 사용하는 이유

1,000,000

	-	apples	sour	monkeys	eat
1,000,000	apples	1	0	0	0
	sour	0	1	0	0

	monkeys	0	0	1	0
	eat	0	0	0	1

1,000,000

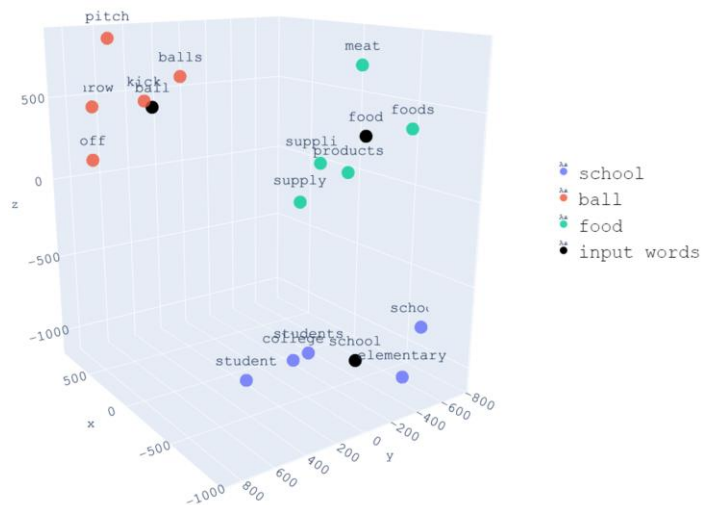
One-Hot 처리된 벡터들은
벡터 공간에서 서로 직교하기 때문에
모든 단어가 완벽히 독립적이라고
가정한 것과 같음



단어의 의미를 표현할 수 없음

임베딩을 사용하는 이유 | 의미 표현

객체의 의미를 반영하도록 설계된 모델을 학습하여 얻은
임베딩 벡터는 의미를 표현할 수 있음



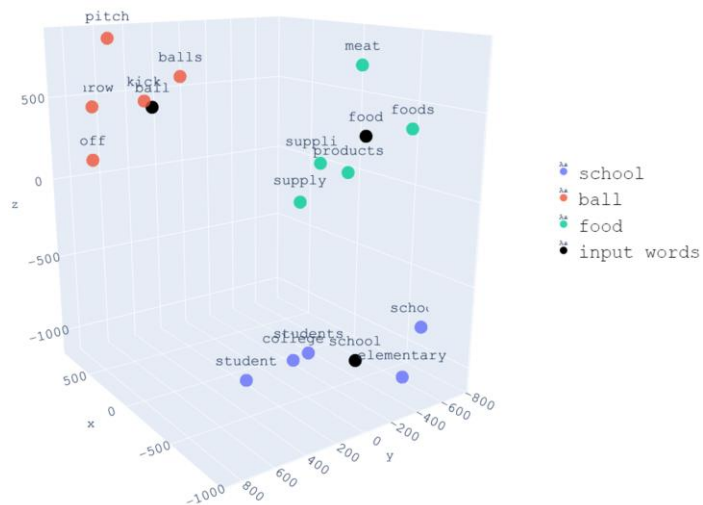
pitch, balls, kick 등 ball과 관련된
단어들은 비교적 가까운 거리에 위치

관련이 적은 meat, supply 등
food과 관련된 단어들은 먼 거리에 위치

Word2Vec을 이용해 school, ball, food의 임베딩
벡터를 3차원 공간에 표시한 예시

임베딩을 사용하는 이유 | 의미 표현

객체의 의미를 반영하도록 설계된 모델을 학습하여 얻은
임베딩 벡터는 의미를 표현할 수 있음



pitch, balls, kick 등 ball과 관련된
단어들은 비교적 가까운 거리에 위치

단어들의 의미와 관계가 보존됨!

관련이 적은 meat, supply 등
food과 관련된 단어들은 먼 거리에 위치

Word2Vec을 이용해 school, ball, food의 임베딩
벡터를 3차원 공간에 표시한 예시

임베딩을 사용하는 이유 | 유사도 계산

첫 번째 특징인 “의미 표현”에 근거하여
벡터들 사이의 유사도를 의미 있는 지표로 활용할 수 있음



유사도의 활용 - 추천 시스템

사용자들 간의 임베딩 벡터 유사도를 계산하여,
비슷한 취향을 가진 사용자들을 그룹화
이를 통해 특정 사용자 그룹에 맞춤형 추천을 제공할 수 있음

임베딩을 사용하는 이유 | 차원 축소

임베딩은 고차원의 데이터를
저차원의 밀집 벡터로 표현하는 방법

One-Hot Encoding - 수십, 수백만 차원의 공간에 객체를 표현
임베딩(GPT) - 256 ~ 3,096 차원을 사용

계산을 훨씬 효율적으로 할 수 있게 해주고, 메모리를 절약할 수 있음



THANK YOU

