기계학습이론 기말대비 2

CNN ~ Transformer

CNN

Image: H*W*3의 크기를 가진다.

Image를 처리하기 위해선 필요한 parameter의 개수가 많아진다.

- $W \in R^{D*(HWC)}$

따라서 MLP를 사용하기엔 부담이 있어 **MLP 대신 CNN**을 사용하고, **MLP의 Matrix multiplication 대신 CNN의 Convolution 연산을 사용**한다.

CNN 이전에 이미지를 처리하는 방법에 대해 먼저 알아보자.

이전에는 **Feature detector**를 이용하여 이미지에서 Feature를 뽑고 각 Feature의 개수를 세어 Histogram으로 만들고 이를 통해 Classification 하였다.

그러나 이 방법은 굉장히 비효율적이고 Feature extractor가 사람이 정한 Feature만을 탐지할 수 있었기 때문에, **뽑아낼 Feature 자체도 학습**하고자 하였다.

각 Filter가 특정 Feature를 탐지할 수 있도록 하고**, 이미지의 각 부분이 Feature와 얼마나 유사한 지를 Inner product로 측정**한다.

2D 이미지와 Filter를 1D vector로 Flatten하고 내적하면 Convolution을 하는 것과 동일하다.

- 따라서 Filter를 이용하여 각 부분을 **Convolution하는 것은 이미지의 각 부분과 Filter가 얼마나 유사한지 확인하는 과정**이다.

Convolution은 **Linear Operation**이다.

- $w * (x_1 + x_2) = w * x_1 + w * x_2$
- $w * (2x_1) = 2(w * x_1)$

Convolution이 linear transformation이기 때문에, **Matrix로 나타낼 수도 있다.**

- Matrix W의 $Columnvector_i$가 $e_i * w$가 되도록 하면 된다.
- Matrix의 차원은 **(Convolution 결과) * (Input dimension)**이 된다.

Convolution matrix의 특징으로 아래 두 가지가 있다.

1. **Sparse**: 0이 굉장히 많다
    - 한 Pixel에서 많은 수의 다른 Pixel을 보지 않는다.
2. **Weight sharing:** 중복되는 값이 많다.
    - **Translation Invarient:** 위치가 변해도 결과가 동일
        - 같은 값이 다른 위치에서 여러 번 나오기 때문이다.
    - W를 적은 Parameter (Filter-only)로도 저장이 가능하다.

위 **두 가지 특징이 강한 Inductive bias**가 된다.

따라서 **CNN은 MLP에서는 Parameter 개수가 많기 때문에 컸던 hypothesis space를 두 가지 강한 Inductive bias를 이용하여 작게 만든다.**

Inductive bias가 크기 때문에, **Data가 비교적 적어도 잘 학습**시킬 수 있다.

Convolution의 종류

1. **Valid convolution**: Padding이 없는 경우
2. **Same convolution**: $H_{in} = H_{out}$

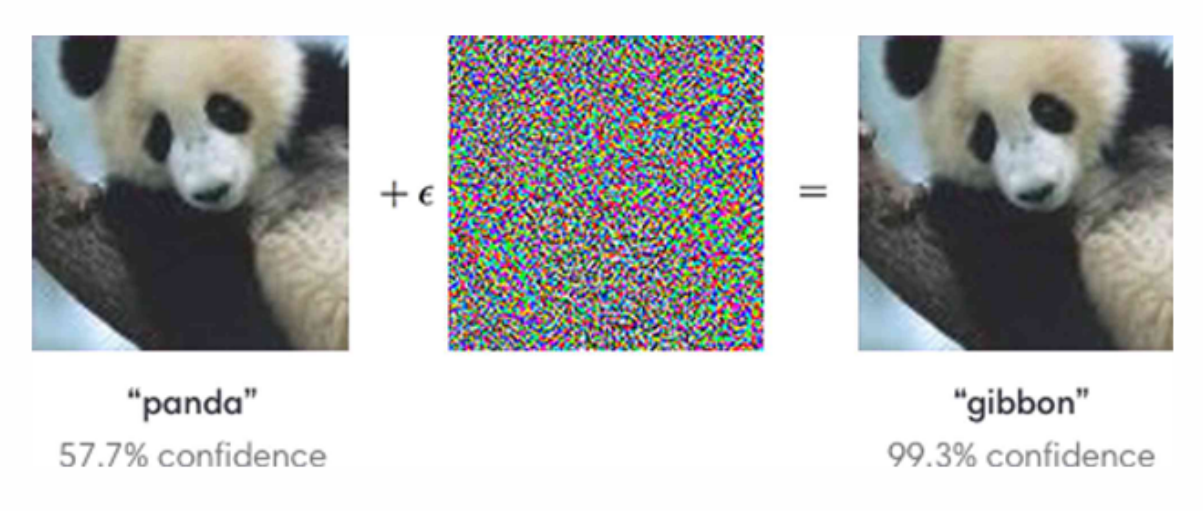
3. **Strided convolution**: s > 1

Pooling

1. **Max pooling**: Linear 하지 않다.
2. **Average pooling:** Linear하다.

CNN은 Convolution layer을 깊게 쌓으며 **Low layer에서는 간단한 Feature, high layer에서는 복잡한 Feature를 배우도록 하며 이미지를 이해**한다.

그러나, Layer를 깊게 쌓아도 이미지를 완전히 이해하지 못 하는 경우가 있었다.



원본 이미지에 대한 분류는 잘했지만, **원본 이미지에 굉장히 작은 Noise만 추가해도 분류에 실패하였다.**

이후, CNN이 어떻게 동작하는지에 대한 연구가 많아지고 Safety에 대한 관심이 높아졌다.

NLP

Text를 처리하는 분야이다.

Task마다 Input과 Output의 형식이 달라지기 때문에 **각 Input, Output의 형식**부터 이해하는 것이 중요하다.

Text는 다루기 굉장히 까다롭다. 이런 Text를 어떻게 다룰 수 있을까?

Text를 **Vector로 Embedding하면 Text간의 연산이 가능**하다. 이 점을 이용하여 Text를 Vector로 Embedding하는 과정부터 알아보자.

1. Word2Vec

- 단어 단위로 수행한다.
- 유사한 단어는 Vector space 내에서 비슷한 위치에 위치하도록 한다.
- Embedding을 학습한 이후에 **Embedding으로의 변환은 고정**되고 L**STM/RNN 등에 plug-in으로 추가되어 사용**된다.
    - Context를 포함하지 못 한다.
- 주변 몇 개의 단어 (Local context)로부터 원하는 단어를 예측하는 방법을 사용한다.
    - **CBOW:** 주변 단어로부터 가운데 단어를 예측
    - **skipgram**: 중간으로부터 주변 단어를 예측

2. GPT-style embeddings

- Subword , Token-level이다.
- **Embedding 방법과 Casual langauge modeling (Next token prediction)이 동시에 Training**된다.
    - Context에 따라 동일한 단어여도 다른 Embedding으로 매핑될 수 있다.

Text sequence의 처리를 더 자세히 이해하기 위해 **PGM**부터 살펴보자.

Probabilistic Graphical Model (PGM)

- Joint distribution을 **Graph(DAG)**를 이용하여 표현한 모델
- DAG를 사용하는 이유는 **Cycle이 없어 순서라는 개념이 성립하게 되기 때문**이다.

PGM에서는 아래와 같은 **Conditionally indenpendent assumption**이 가능하다.

$$Y_i \perp Y_{pred(i)|pa(i)} | Y_{pa(i)}$$

- Parent가 주어지면 특정 한 Token과 Parent를 제외한 그 전의 모든 token은 Independent하기 때문에 볼 필요가 없다는 것이다.

결국, PGM에 따르면 Joint distribution은 아래와 같이 정의된다.

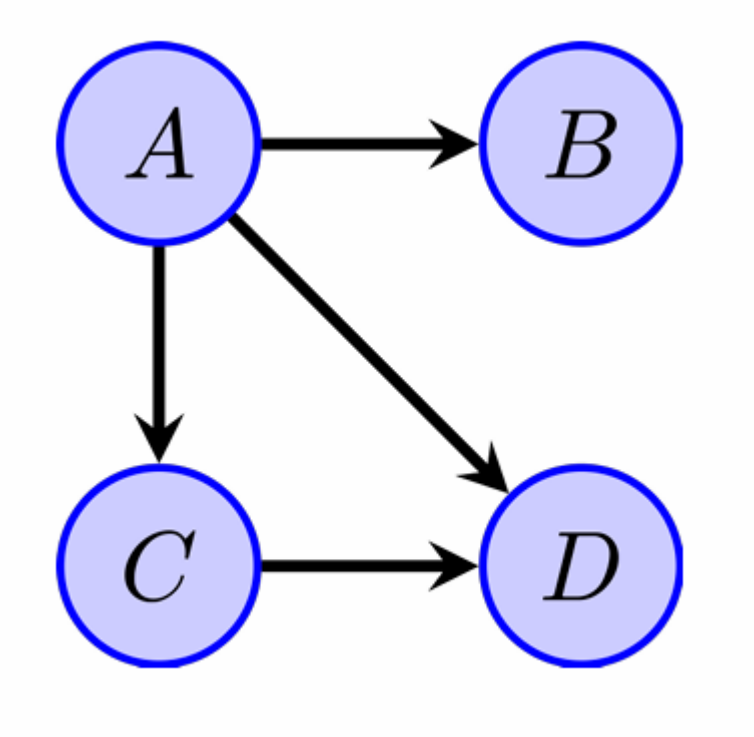
$$p(Y_{1:V}) = \prod_{i=1}^{V} p(Y_i | Y_{pa(i)})$$

문장 전체의 확률을 나타내는 **Joint distribution을 표현할 수 있다면 좋지만, 그렇게 된다면 경우의 수가 굉장히 많아져 어렵다.**

이를 해결하기 위해 **Joint distribution을 Factorization한 위 공식을 사용하여 대체**한다.

아래와 같은 DAG를 보자.

P(A, B, C, D) = P(A)P(B|A)P(C|A)P(D|A, C)



위 그래프에서 **Edge(P(B|A), P(C|A), P(D| A, C))를 Nerual network가 각각 예측하는 것**이라고 생각할 수 있다.

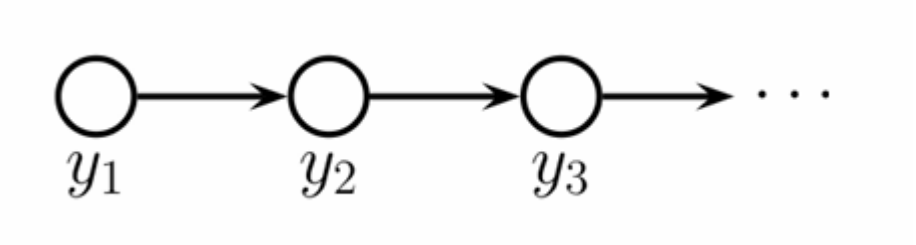
- 즉, **Joint distribution을 여러 Conditional distirubution으로 Factorization하여 각각을 예측하도록 하는 방법**이다.
- Text sequence의 처리를 DAG를 통해서 이해할 수 있다.

이 구조의 특징은 **Conditionally Independent Assumption으로 인하여 Parent Token을 제외한 이전 Token들의 구조를 반영할 수 없다는 것이다.**

- PGM의 각 Node는 Random variable이다.
- 이 가정 때문에 **Inductive bias가 크다.**
- CNN과 마찬가지로 데이터가 적을 때 유리한 이유이다.

Parent가 하나만 존재하는 **가장 극단적인 경우**를 보자.

Marcov model은 **future state가 직전 하나의 상태에만 의존한다는 아주 강한 Inductive bias**를 부여한다.



Marcov assumption 하에선 **Parent가 하나라는 강한 Bias로 인해 Model의 자유도가 떨어진다.**

RNN

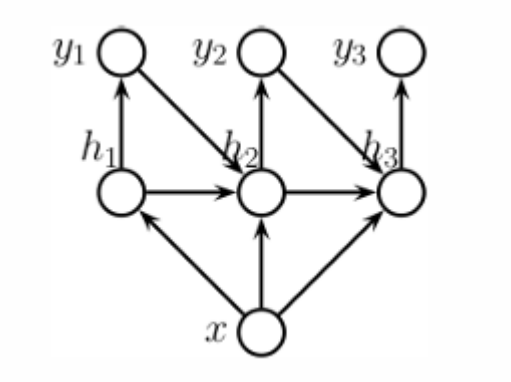
위에서 본 Text를 처리하는 방법과 Task의 종류에 따라 여러 RNN이 정의될 수 있다.

RNN의 기본 구조는 PGM을 따르기 때문에 Joint distribution을 Conditional distribution의 곱으로 나타낼 수 있다.

크게 3가지 구조로 나뉜다.

1. **Vec2Seq** ($x \rightarrow y_{1:T}$)
    - Language modeling (x is None), Image captioning
2. **Seq2Vec** ($x_{1:T} \rightarrow y$)
    - Text classifcitaion
3. **Seq2Seq (**$x_{1:T} \rightarrow y_{1:T}$)
    - Document summurization

이 중, **Vec2Seq**의 구조를 자세히 살펴보자.



위 구조에 따른 Joint distribution은 다음과 같다.

$$
\begin{aligned}
p(x, y_1, y_2, y_3, h_1, h_2, h_3) = p(x) \\
&\times p(h_1 \mid x) \\
&\times p(h_2 \mid x, h_1, y_1) \\
&\times p(h_3 \mid x, h_2, y_2) \\
&\times p(y_1 \mid h_1) \\
&\times p(y_2 \mid h_2) \\
&\times p(y_3 \mid h_3).
\end{aligned}
$$

여기서 $p(y_i|h_i)$는 **Categorical distribution**이고, $p(h_i|x)$는 **Neural network의 결과**라고 생각할 수 있다.

- $p(y_i|h_i) = Cat(y_t|softmax(W'h_t))$이기 때문에 확률적이다.
  - Softmax 분포에서 Sampling된다.
  - Stochastic
- $p(h_i|x)$는 보통 이미 결정되어져 있다.
  - 확률적 요소가 없고 정해진 W에 대해 값이 나온다.
  - Deterministic

위 구조의 RNN은 **Hidden state 덕분에 이전의 여러 상태를 확인**할 수 있다.

- 위 구조가 DAG이지만, 이 때 **DAG의 Node는 deterministic value이기 때문에 지난 계산 값이 누적되어 이전 정보가 압축되어 표현된 것처럼 사용**된다.

- 따라서 **메모리가 모든 hidden state의 값을 저장할 수 있을 정도로 충분히 크다면, Inductive bias를 크게 줄일 수 있다.**
- 그러나 여전히, **PGM 구조 자체가 큰 Inductive bias**이다.
  - Hidden state를 통해 정보가 전달된다고 하더라도, **직접 보지 못 하고 다른 Node를 건너서 들어야 한다는 것 자체가 큰 Bias**이다.
  - **구조에 따라 Bias가 증가하거나 감소**한다.
- 전체 Sequence의 정보를 담기에 **하나의 Hidden state만 사용하는 것이 적합하지 않거나 메모리 부족** 등의 문제가 생길 수 있다.

Training

1. 우리에게 주어진 Data는 $\{X, Y\}$ pair일 것이다.

2. Hidden state인 H는 데이터에 주어지지 않으므로 Joint distribution을 H에 대해 적분하여 $P(X, Y)$를 얻는다.

3. $P(X, Y)$에 대해 NLL을 하여 Loss를 계산한다.

RNN의 성능을 좋게 하기 위해 여러 Layer를 쌓을 수 있다.

$$h_t^l = \phi(W^l[h_t^{l-1}; h_{t-1}^l])$$

- 직전 Layer의 Hidden state와 직전 Time step의 Hidden state의 영향을 받는다.

BackPropagation Throung Time (BPTT)

Text는 Sequence로 이루어져 있기 때문에, **텍스트가 길거나 Time이 길수록 Loss와 Input간의 거리가 커지게 된다.**

이는 **BPTT의 소요 시간을 늘리고, Gradient exploding / vanishing problem에 빠지게 만든다.**

Gradient vanishing problem을 해결하기 위해 Residual net의 아이디어를 빌린 **LSTM과 GRU**를 사용하기도 한다.

Transformer

RNN이 Hidden state를 통해 이전 Token의 정보를 간접적으로 전달할 수 있지만, **여전히 t번째 token을 결정하는 데에는 최근 Token의 영향**이 크다.

- **Inductive (Sequential bias)가 크고, Locality & Recency**

이를 해결하고 **Inductive bias를 줄이기 위해 Transformer**를 사용한다.

가장 큰 차이점은 Transformer는 **특정 Token이 전체 Token을 확인**할 수 있도록 하는 것이다.

- 이 방법은 **Attention**이라고 한다.
- **Inductive (Sequential bias)가 작다.**

$$h^l = A(h^{l-1})h^{l-1}$$

- $A(h^{l-1})$: Attention weights
- h^{l-1}: Value
- 모든 Token의 값의 Weighted sum이다.

Position encoding이 없다면, Input의 순서가 변경되더라도 Output의 순서만 변경되고 값은 달라지지 않는다!

Inductive bias 차이 때문에, 성능에서 **Data가 작을수록 CNN > Transformer, Data가 많을수록 Transformer > CNN인 경향을** 보인다.