Before – 주어진 데이터를 통해서 출력값을 가지는 함수를 모사하자!

주어진 입력(x)에 대해서 원하는 출력(y)을 반환하도록 손실함수를 최소화하는 파라미터(θ)를 찾자!

Gradient descent를 수행하기 위해 back-propagation을 수행한다.

After – 주어진 데이터를 통해서 출력값을 가지는 확률분포 함수를 모사하자!

확률 분포 P(x)와 p(y|x)로부터 데이터를 수집하여, 해당 데이터를 잘 설명하는 확률 분포 함수의 파라미터(θ) 찾자: logP(y|x; θ) – MLE, NLL, Gradient Descent using backpropagation

정보이론 관점에서 두 분포를 비슷하게 만들자! Minimize Cross Entropy, KL-Divergence

Geometric perspective

데이터는 저차원의 manifold에 분포하고 있으며, 여기에 노이즈가 추가된다.

Manifold: 고차원상의 데이터가 존재하고 있는 모양. 이것을 저차원 공간으로 매핑

데이터는 고차원 공간에 존재하고 있는데, 고차원 공간에 균등하게 있는 것이 아니라 저차원의 manifold에 분포하고 있다. Manifold를 배울 수 있다면, 더 낮은 차원으로 효율적인 매핑이 가능하다.

고차원 공간의 샘플들이 저차원 다양체(manifold)의 형태로 분포해 있다. 고차원 공간에서의 두점 사이의 거리는 저차원 공간으로의 맵핑 후 거리와 다르다. (증명은 되지 않아 가설이지만, 기정사실)

BPTT(Back-Propagation with Multiple Forwarding Path)는 강의를 보고 확인하자. 그냥 이해하기는 어렵다.

CNN RNN LSTM

# CNN

Convolution filter가 입력(x)에 대해서 돌아가며 동작한다.

도표, 사각형, 직사각형, 라인이(가) 표시된 사진

자동 생성된 설명

위 그림에서는 input size=4, kernel size = 3, pad가1, 이며 output size = 4가 된다. Kernel size가 3이고 pad가 1이면 output size가 그대로 유지되므로 많이 사용된다.

CNN은 kernel(filter)가 돌아다니기 때문에 input size나 output size가 상관없다. Kernel의 size가 주어졌을 떄, input의 사이즈에 따라서 output size가 변한다.

도표, 직사각형, 라인, 평행이(가) 표시된 사진

자동 생성된 설명

Input channels가 1일때는 gray, 3일때는 RGB다.

스크린샷, 텍스트, 디자인이(가) 표시된 사진

자동 생성된 설명

N : 배치사이즈, C : 채널의 수(RGB, gray), H : 높이, W : 넓이

텍스트, 스크린샷, 폰트, 문서이(가) 표시된 사진

자동 생성된 설명

# Dimension Reduction 방법 : Polling, stride

Max Pooling, Avg Pooling이 있다.

텍스트, 스크린샷이(가) 표시된 사진

자동 생성된 설명

Avg pooling은 주어진 크기 내에서, ㅋ값들의 합을 평균으로 나타낸 것이다.

Stride는 kernel size가 이동하는 칸수를 말한다. 만약 stride가 2라고 하면,

도표, 직사각형, 사각형, 라인이(가) 표시된 사진

자동 생성된 설명도표, 직사각형, 라인, 스크린샷이(가) 표시된 사진

자동 생성된 설명

위와 같은 방식으로 두칸씩 이동한다.

CNN은 병렬 계산 구성이 쉬우므로, GPU에서의 연산이 매우 빠르다. FC Layer에 비해 입출력 크기가 까다로워 네트워크 구성이 쉽지 않다.

영상처리 분야에서는 convolutional filter를 이전부터 활용해왔다. CNN은 데이터셋의 x와 y의 구성에 따라 자동으로 탐지해야 할 패턴을 추출한다. FC Layer에 비해 매우 빠르고 적은 weight parameters를 가진다. + 이미지 특화 패턴 추출을 한다.

있기있는 backbone으로는 VGG,Resnet등이 있다!! 이것을 Transfer learning을 통해 학습을 한다!

\*Transfer Learning(전이학습): 이미지넷을 통해서 학습된 네트워크를 가지고 우리의 데이터 셋에다가 학습을 조금 더 시키면 high-level feature을 추출할 수 있을 것이다(적어도 edge나 low level feature들을 추출할 수 있다). Zero-base로 학습시키는 것보다와 pretraining된 θ를 가져와서 학습하는게 수렴도 빠르고 정확도가 높다. 특히 NLP에선 BERT, GPT가있다!

# RNN

RNN은 순환 신경망으로, 시퀀스 데이터를 처리하고 모델링하는 데 사용된다. 이 네트워크는 이

전 단계의 출력을 현재 단계의 입력으로 사용하여 순환적인 구조를 갖추고 있다.

텍스트, 도표, 스크린샷, 폰트이(가) 표시된 사진

자동 생성된 설명

는 이전 time step의 output과 현재 time step의 input을 동시에 받는다

Sequential Data vs Time Series

Sequential Data: 데이터의 순서 정보가 매우 중요 (텍스트, 샘플링 주기가 일정한 영상/음성)

Time Series: 해당 데이터가 발생한 순서정보+시각정보가 매우 중요 (주식 데이터, 센서 데이터)

RNN에는 시간개념이 추가되지만, Generation task가 아니라면 시간개념을 구현해서 넣을 필요가 없다! 입/출력 텐서의 모양만 알면 된다. Back-propagation도 기본원리만 기억하면 된다!

**RNN은 이전 time step의 output을 hidden state에 저장하여 사용된다. 이 hidden state와 현재 time step의 input을 동시에 입력으려 받는다.**

## RNN의 표현방법

텍스트, 폰트, 도표, 라인이(가) 표시된 사진

자동 생성된 설명

이전 time step의 output size + 현재 time step의 input

RNN은 이전 time step의 출력(:hidden state), w: weight, b: biase,

RNN의 non-linear activation unction은 tanh, 두번째 줄은 2개의 linear layer가 있는것처럼 동작.

도표, 텍스트, 디자인이(가) 표시된 사진

자동 생성된 설명

단지, 말려있는 모양을 펴준 모양이다. 첫번째부터 time step이며 점점 진행된다.

텍스트, 도표, 라인, 그래프이(가) 표시된 사진

자동 생성된 설명

**이것은 single-layer RNN에만 해당된다. Multi-layer RNN에서는 마지막 time step의 가 이다.** 입력이 되면 한 번에 출력이 한 번에 팡! 하고 튀어오른다! 입력값을 신경쓸 필요가 없다. 시간개념없이 그냥 tensor를 넣어주면 tensor가 출력되도록 pytorch가 구현되어 있다. Loss가 minimizer 되도록 설계되어있다.

## RNN의 Input Tensor shape

각 time step에서 입력을 받는다.

텍스트, 스크린샷, 도표, 폰트이(가) 표시된 사진

자동 생성된 설명

한 단어는 vector로 이루어져 있다. 문장의 경우 n개의 단어이다. NLP에선 1개의 Time step이 아닌, 전체 Time step이 들어올 것이다.

NLP: (Batch size, Length, input size) 이미지와 다르게 3차원이다.

첫번째 모양은 batch\_ size, Length는 단어의 수, input size는 입력으로 들어가는 feature의 크기로, 단어나 토큰을 나타내는 임베딩 벡터의 차원이다.

## RNN의 Hidden Tensor shape

텍스트, 스크린샷, 도표, 디자인이(가) 표시된 사진

자동 생성된 설명

Hidden\_state = (layers, batch\_size, hidden\_size): 이전 time step의 학습정보이며, hidden state는 batch first를 적용할 수 없다. 따라서 batch size가 중간에 온다

Hidden size는 hidden state 벡터의 차원이다. 모델의 용량과 학습가능한 매개변수를 결정하며, 모델의 표현력과 더 많은 정보를 학습할 수 있지만, 모델의 복잡성이 증가하고 오버피팅을 주의해야한다.

## Multi-layerd RNN

도표, 라인, 평면도이(가) 표시된 사진

자동 생성된 설명

밑에서 나오는 이 input이며, time step이 된다. 왼쪽에서 받는 것들이 hidden state이며 노란색이 1층, 연두색이 2층, 초록색이 3층이 된다.

Hidden state는 모든 층을 말한다. 마지막 layer의 hidden state들이 이 된다.

각 층마다 다른 파라미터(θ)를 가진다. 하지만시간에 대해서는 같은 파라미터(θ)를 가진다.

직접 층을 어떻게 쌓을지는 고민할 필요가 없다! RNN을 선언할 때 num\_layers 라는 argument가 있다. 그대로 사용하면 된다.

## Multi-layer Output Tensor

도표, 스크린샷, 텍스트, 라인이(가) 표시된 사진

자동 생성된 설명



1층의 출력은 2층의 입력 + hidden state, 2층의 출력 = 3층의 입력 + hidden state… 마지막 층 출력이 multi-layer RNN의 출력이다. 출력값은 마지막 layer에 대해서 hidden state다.

즉, (출력값)는 마지막 layer의 다. 빨간 점선이 output이 되겠다.

## Multi-layer Hidden State Tensor

도표, 텍스트, 스크린샷, 그래프이(가) 표시된 사진

자동 생성된 설명



마지막 layer의 Hidden state가 출력이다. 빨간색 점선이 hidden state다.

한 time step만 보기 때문에 time step에 대한 정보는 필요없다.

## Multi-layer Input Tensor

텍스트, 도표, 스크린샷, 폰트이(가) 표시된 사진

자동 생성된 설명

Sigle-layer와 multi-layer의 input 형태는 같다.

## Bideirectinal Multi-layer RNN

도표, 라인, 텍스트이(가) 표시된 사진

자동 생성된 설명

입력과 출력이 여러 개다! 양방향이라 역방향도 존재한다!

## Output State Tensor

도표, 텍스트, 스크린샷, 라인이(가) 표시된 사진

자동 생성된 설명

마지막 layer의 각 time step별 정방향 역방향 hidden state들이 존재한다. 양방향이면, hidden size에 \*2를 해준다.

## Hidden State Tensor

도표, 스크린샷, 종이접기, 디자인이(가) 표시된 사진

자동 생성된 설명

RNN: = f(, ;θ) 이전 time step의 을 위해 보통 사용한다.

Bi-directional RNN: 전체 time step의 입력을 받고, 전체 time step의 출력이 나가는 형태가 되어, 다음 입력을 받기 위한 어떤 작업을 하지 않는 경우가 대부분이다. Bi-RNN은 두개의 RNN을 사용하여 각각 앞뒤 방향으로 데이터를 처리한다. 따라서 Bi-RNN은 양방향 정보를 결합하여 time step에서의 출력을 생성한다.

* Summary

- Single layer RNN에서 hidden state는 곧 output이다.

- Multi-layer RNN에서 output은 마지막 layer의 모든 time step의 hidden state다. Hidden state는 마지막 time step의 모든 layer의 hidden state다. 마지막 layer의 hidden state가 output이 된다.

- Bi-directional RNN에서 Output은 hidden state가 2배가 된다. Hidden state는 layer의 개수가 2배가 된다(역방향도 있기 때문이다)

# LSTM

RNN내부에는 tanh가 있으므로, time step이 길어짐에 따라서 gradient vanishing이 발생한다. 따라서 긴 sequence는 학습이 어렵다.

tanh대신 gate에 sigmoid를 곱하면 마치 문을 열고 닫는 듯한 효과를 낼 수 있다!

y = σ(x)\*f(x)

따라서 LSTM은 sigmoid 특성을 고려해 만든 gate라고 할 수 있다.

**Cell state는 LSTM에만 존재하며, 셀 내부의 정보를 유지하고 전달하는 역할을 한다. 이것은 Gradient Descent를 해결하는데 도움을 준다.**

텍스트, 도표, 평면도, 기술 도면이(가) 표시된 사진

자동 생성된 설명

: input gate로, 이전 time step의 output인 와 현재 입력값인 를 조합해 sigmoid를 통과시켜 만들어진다. 그리고 cell state에 정보를 얼마나 반영할지 결정하는 역할을 한다.

: forget gate로, 이전 cell gate에서 어떤 정보를 잊을지 결정한다.

: 입력 게이트의 출력값에 따라 후보값인 가 계산되고, 이는 tanh 함수를 통해 생성된다. 는 새로운 정보를 담은 후보 값으로 이후 cell state를 업데이트하는 데 사용한다.

: output gate로, 와 weight(가중치)를 통해서 구해지며, 최종 출력값을 결정한다.

: Cell state로, 셀 내부에서 정해진 정보를 유지하고 전달한다(메모리 관리). 메모리 저장을 위한 또다른 hidden gate라고 생각하면 된다. 장기적인 의존성을 학습하기 때문에, Gradient descent를 해결하는데 도움이 된다. LSTM에만 존재한다.

: Hidden state로, 각 time step에서 이전정보를 출력하는 역할을 한다.

LSTM의경우 복잡하고 무겁다. 보완하기 위해 GRU라는 알고리즘이 나왔다.

# GRU

텍스트, 도표, 라인, 스크린샷이(가) 표시된 사진

자동 생성된 설명

Hidden state: LSTM의 hidden stae + Cell state 개념이다.

Reset gate: 새로운 정보와 이전 상태를 얼마나 혼합할지 결정한다. 즉, 이전 상태의 정보를 얼마나 reset할지 σ(시그모이드)를 통해 결정한다. (얼마나 버릴지)

Update gate: 이전 정보를 얼마나 유지할지 σ(시그모이드)를 통해 결정한다. (얼마나 유지할지)

LSTM은 vanilla RNN에 비해 훨씬 더 많은 파라미터를 가진다. GRU를 통해 어느정도 해결이 되었다. 하지만 Gradient Vanishing문제가 완전 해결된 것은 아니다. -> Attention을 통해 해결

텍스트, 스크린샷, 도표, 폰트이(가) 표시된 사진

자동 생성된 설명

None Auto-regressive: Many to one, Many to Many: 현재 상태가 앞/뒤 상태를 통해 정해지는 경우, Bi-directional RNN을 사용한다. (pos tagging(형태소 분석), Text Classification)

Auto-regressive: One-to-Many: 현재 상태가 과거 상태에 의존하는 경우, Bi-directional 사용 불가능하다. (NLG, Machine Translation)