Lab 3: RNN, GAN

<기계학습 개론> 실습

이현도, 최원석, 김윤성

2019.11.07

Biointelligence Laboratory

School of Computer Science and Engineering

Seoul National University

BI

BIOINTELLIGENCE

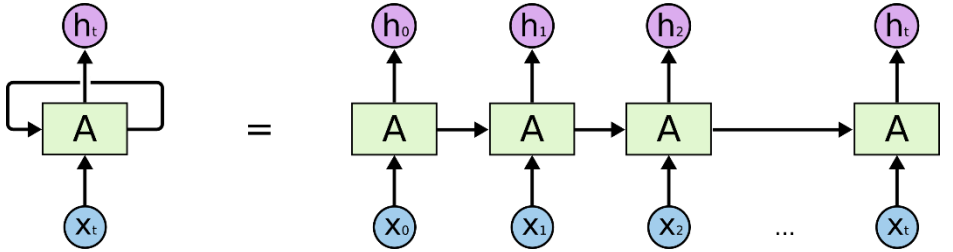
Contents

- RNN, LSTM
- NLP(자연어 처리)
- GAN, cGAN

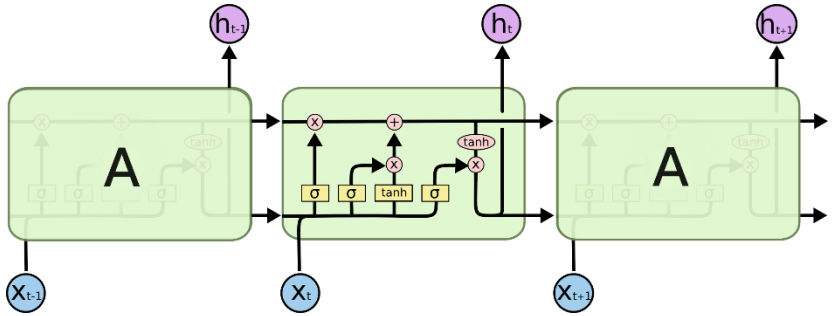
Recurrent Neural Network

RNN, LSTM

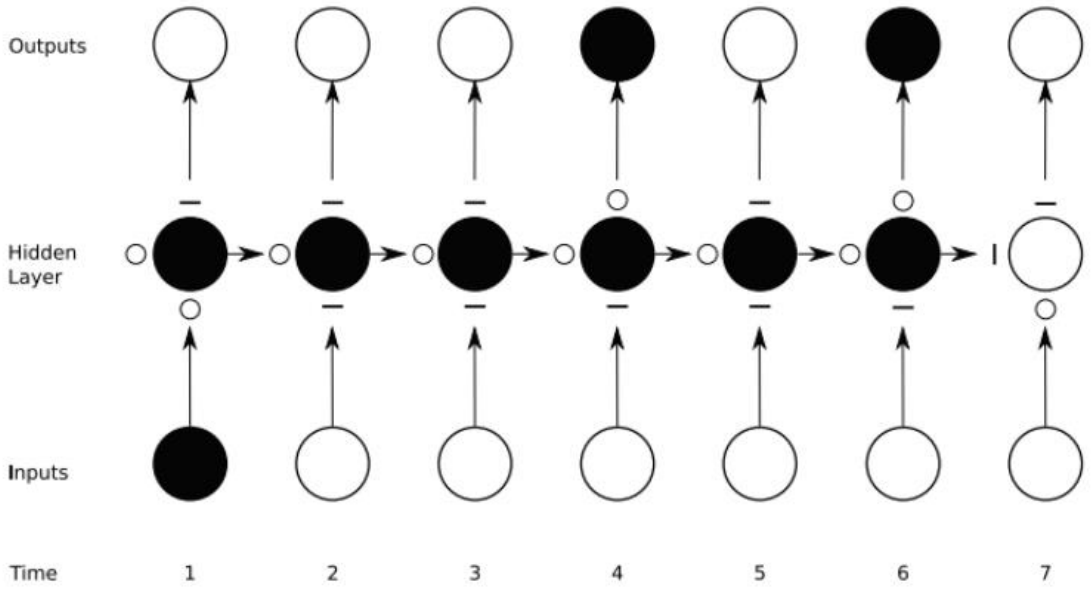
- RNN
  - 시퀀스 데이터의 모델링
  - 기억을 가지고 있음
  - 새로운 입력과 기존 기억을 조합하여 출력
  - 입력이 들어올 때마다 같은 구조를 반복



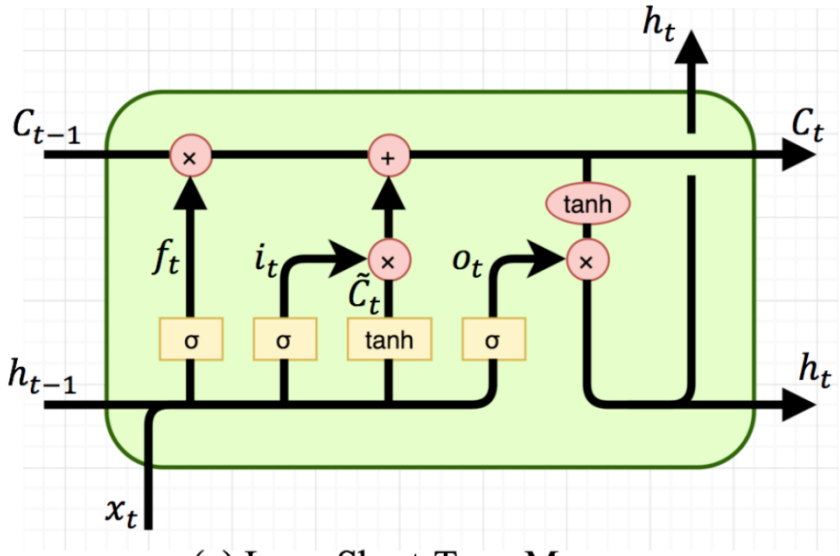
- LSTM
  - RNN의 long-term dependencies 문제 해결
  - Cell state의 도입
    - Gate를 도입하여 cell state를 제어



LSTM의 구조



O는 각 게이트가 열려 있음을,
-는 각 게이트가 닫혀 있음을 의미



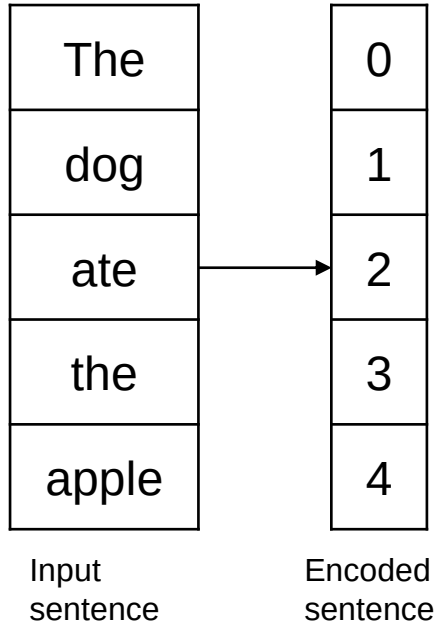
(a) Long Short-Term Memory

$$
\begin{aligned}
f_t &= \sigma_g(W_f x_t + U_f h_{t-1} + b_f) \\
i_t &= \sigma_g(W_i x_t + U_i h_{t-1} + b_i) \\
o_t &= \sigma_g(W_o x_t + U_o h_{t-1} + b_o) \\
c_t &= f_t \circ c_{t-1} + i_t \circ \sigma_c(W_c x_t + U_c h_{t-1} + b_c) \\
h_t &= o_t \circ \sigma_h(c_t)
\end{aligned}
$$

품사 추론 LSTM 모델

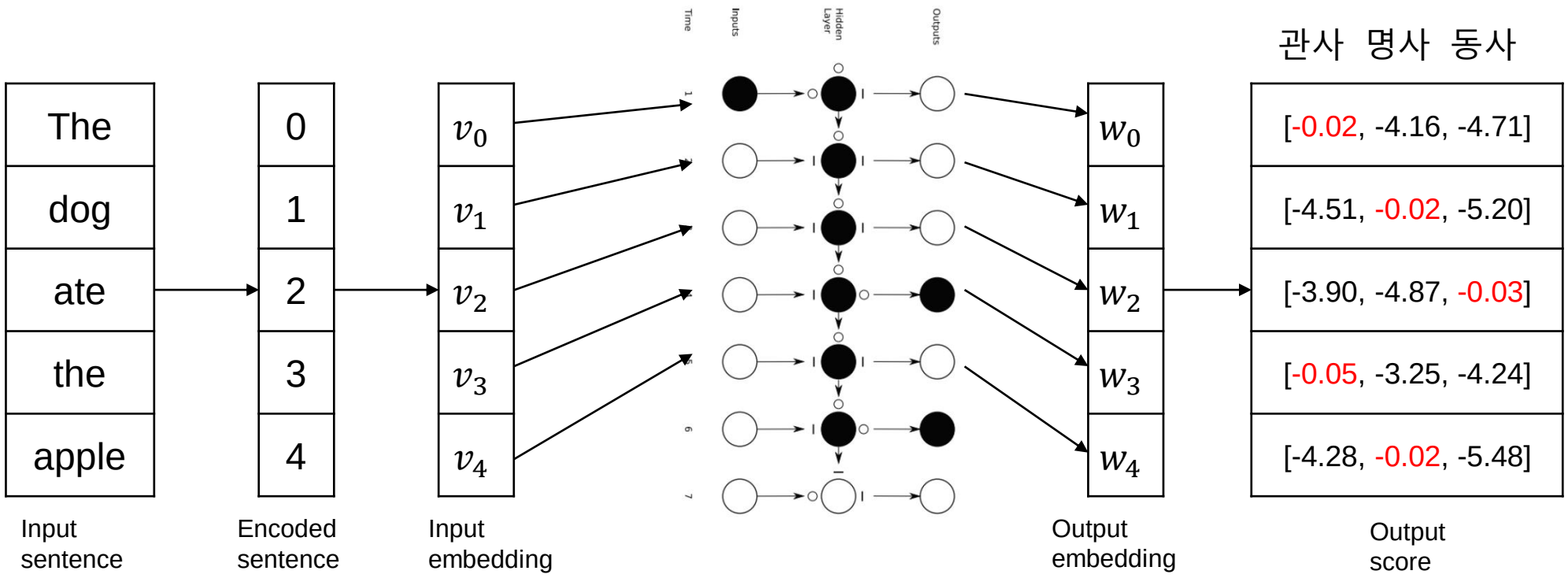
- 목표
  - 문장이 주어지면, 각 단어의 품사를 추론하는 모델 설계
  - 이전까지 나온 단어들을 알아야 현재 단어의 품사를 추론하기 용이
    - 이전 단어들을 기억할 수 있는 LSTM 모델이 적합
  - input sequence: 문장에 있는 단어들의 sequence
  - output sequence: 각 단어의 품사 sequence

- 전처리
  - Input을 단어 그대로 넣는다면? (The, dog, ate, the, apple)
    - 단어마다 길이가 다름 -> 벡터화하기 어려움
  - 따라서 단어마다 하나의 숫자를 대응시켜 encoding



품사 추론 LSTM 모델

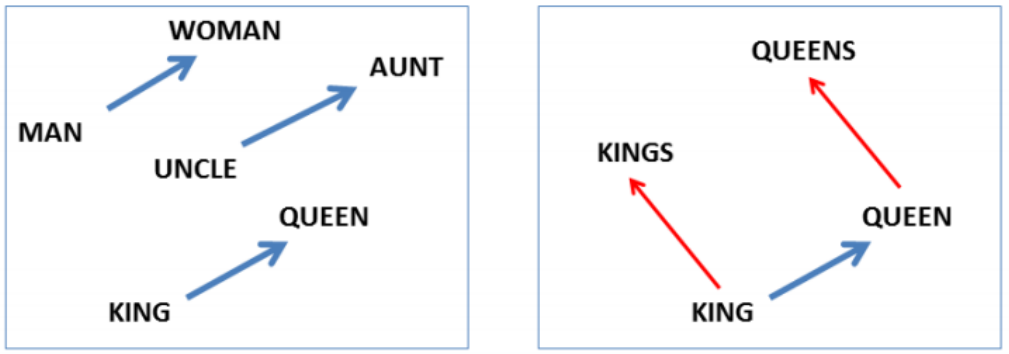
- Encoding 과정을 거친 숫자를 n차원 공간에 embedding하여 LSTM의 input으로 사용
- LSTM의 output에 fully connected layer를 적용하여 각 단어의 품사 score를 계산



Preprocessing

- 전처리 과정 개선
  - 각 단어를 단순히 서로 다른 숫자에 대응시키기만 했는데, 단어의 연관성에 따라 '더 잘' 대응시킬 수는 없을까?
  - 즉, 서로 비슷한 단어를 가까운 vector에, 서로 관련 없는 단어를 먼 vector에 대응시키는 방법은 없을까?
    - Word vectorization

- Word2Vec, GloVe
- ELMo, **BERT**



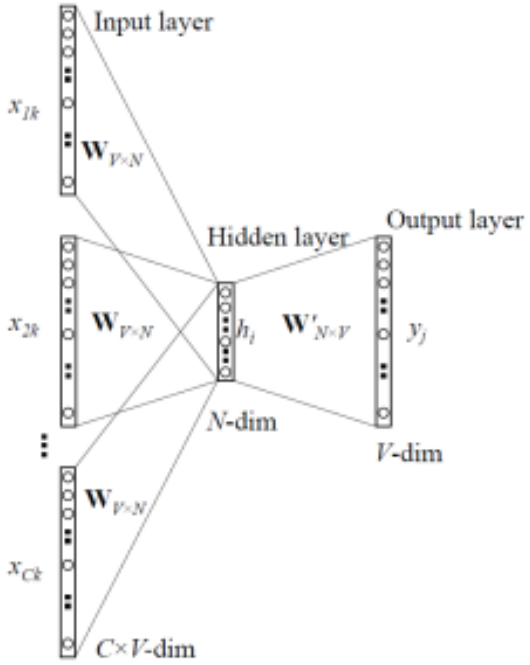
(Mikolov et al., NAACL HLT, 2013)

Vectorization - Word2Vec

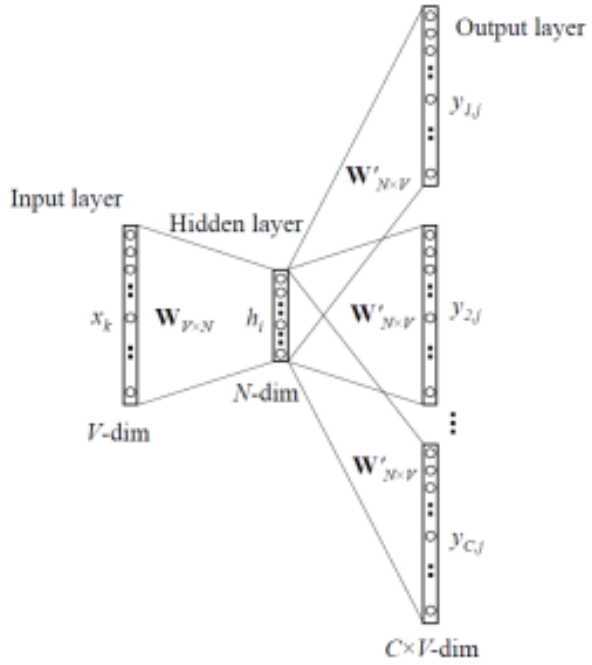
- Word2Vec
  - Word Embedding : {set of words} -> R^n (feature space)
  - Shallow two-layer NNLM
    - CBOW, Skip-gram
  - 한 문장에 있는 단어들이 **similar context**라 간주
    - Distributional Hypothesis - 비슷한 분포를 가진 단어는 비슷한 의미를 가짐
  - 관련 있는 단어일수록 embedding 결과의 cosine similarity를 작게

  - Library - Python Gensim
    - https://pypi.org/project/gensim/

Vectorization - Word2Vec



CBOW



Skip-gram

```
# Word2Vec embedding
from gensim.models import Word2Vec
embedding_model = Word2Vec(tokenized_contents, size=100, window = 2, min_c
ount=50, workers=4, iter=100, sg=1)
```

Vectorization - Word2Vec

- 학습을 위해서 필요한 요소들
  - 학습 데이터베이스
    - 학습하고자 하는 DB 사용 (충분히 클 경우)
    - 한국어 - 한국어 위키피디아 DB, KorQuAD, 나무위키 DB, …
  - 형태소 분석(Pos-tagging)
    - 원시말뭉치를 형태소(어근, 접두사/접미사, 품사 등) 단위로 쪼개고 각 형태소에 품사 정보를 부착하는 작업
    - Library – KoNLPy
      - https://konlpy-ko.readthedocs.io/ko/v0.4.3/

```
가방에 들어가신다 -> 가방/NNG + 에/JKM + 들어가/VV + 시/EPH + ㄴ다/EFN
```

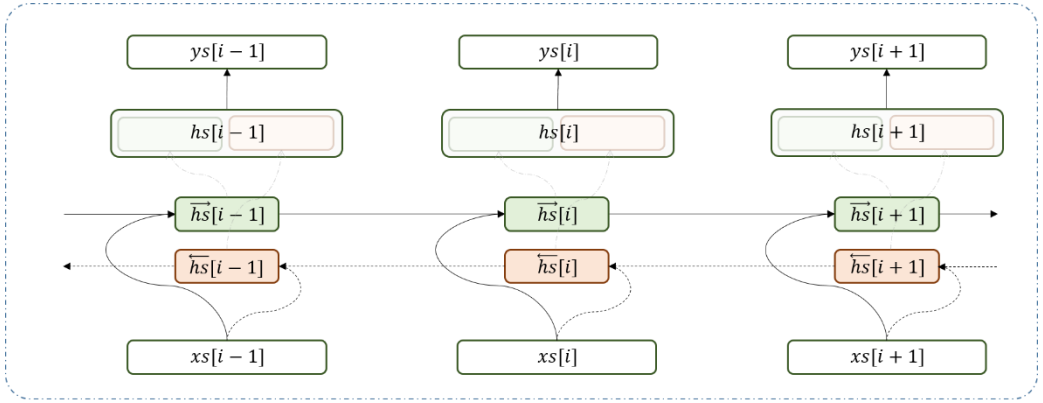
Word Embedding sources

- 직접 하고 싶은 경우
  - 아래 링크를 참조할 것을 강하게 추천
  - https://ratsgo.github.io/embedding/

- Pretrained된 데이터 사용
  - **BERT - https://pypi.org/project/bert-embedding/**
  - GloVe - https://nlp.stanford.edu/projects/glove/
  - 구글에 ***pre-trained word embedding***을 키워드로 적절히 검색

Bidirectional RNN(LSTM)

- 필요성
  - 텍스트 데이터는 정방향(시점을 기준으로 과거에서 미래 방향) 추론 못지 않게 역방향(시점을 기준으로 미래에서 과거 방향) 추론도 유의미한 결과를 냄
    - *나는 ____를 뒤집어 쓰고 펑펑 울었다.*
  - 하지만 일반적인 RNN 구조는 오로지 정방향으로 데이터를 처리



$$ys[i] = W_{hy}\, hs[i] + by$$
$$(Y \times 1) \quad (Y \times H)\ (H \times 1) \quad (Y \times 1)$$

$$\overleftarrow{hs}[i] = \tanh(\overleftarrow{W_{xh}} xs[i] + \overleftarrow{W_{hh}}\ \overleftarrow{hs}[i+1] + \overleftarrow{bh})$$
$$\left(\frac{H}{2} \times 1\right) \quad \left(\frac{H}{2} \times V\right)\ (V \times 1) \quad \left(\frac{H}{2} \times \frac{H}{2}\right)\ \left(\frac{H}{2} \times 1\right) \quad \left(\frac{H}{2} \times 1\right)$$

$$\overrightarrow{hs}[i] = \tanh(\overrightarrow{W_{xh}}\ xs[i] + \overrightarrow{W_{hh}}\ \overrightarrow{hs}[i-1] + \overrightarrow{bh})$$
$$\left(\frac{H}{2} \times 1\right) \quad \left(\frac{H}{2} \times V\right)\ (V \times 1) \quad \left(\frac{H}{2} \times \frac{H}{2}\right)\ \left(\frac{H}{2} \times 1\right) \quad \left(\frac{H}{2} \times 1\right)$$

$$hs[i] = \begin{bmatrix} \overrightarrow{hs}[i] \\ \overleftarrow{hs}[i] \end{bmatrix} \quad \text{stacked!}$$

$Output\ Size = Y$

$Hidden\ Size = H$

$Vocabulary\ size = V$

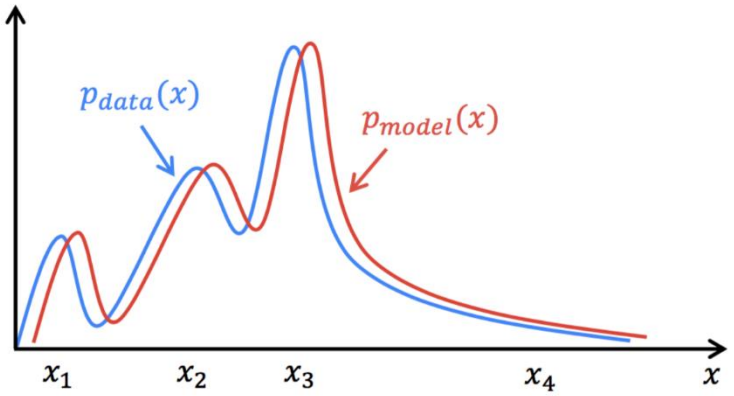
- 코드
  - LSTM(…, *bidirectional=True*)

Generative Adversarial Network

Generative Models(생성 모델)

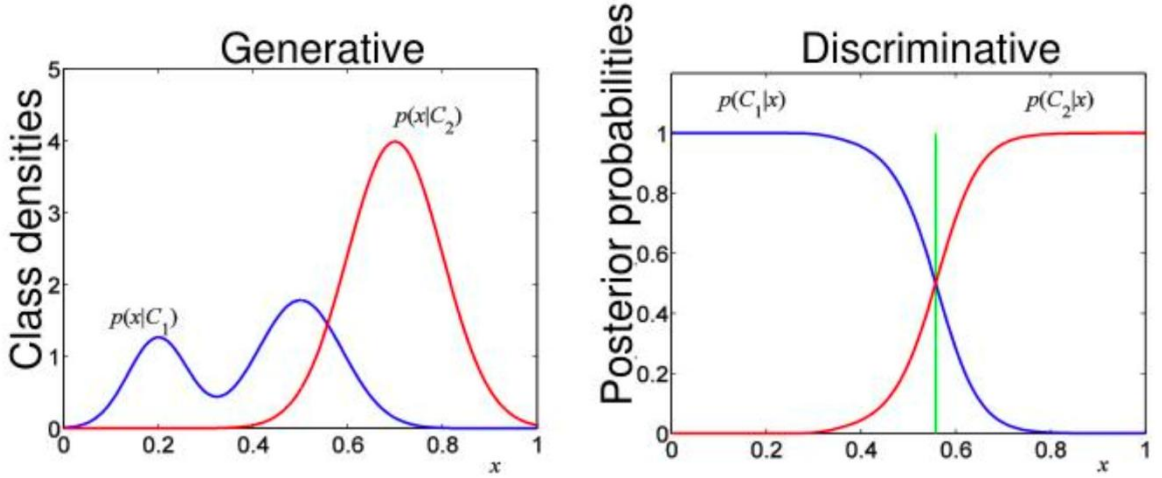
- Discriminative Models
  - 데이터 X, 레이블 Y → 데이터를 받아 레이블을 추측, $P(Y|X)$
  - 입력에 대한 레이블의 경계선만 파악하면 충분
  - 레이블로부터 다시 데이터를 복구하는 문제 $P(X|Y)$ – 해결이 어려움

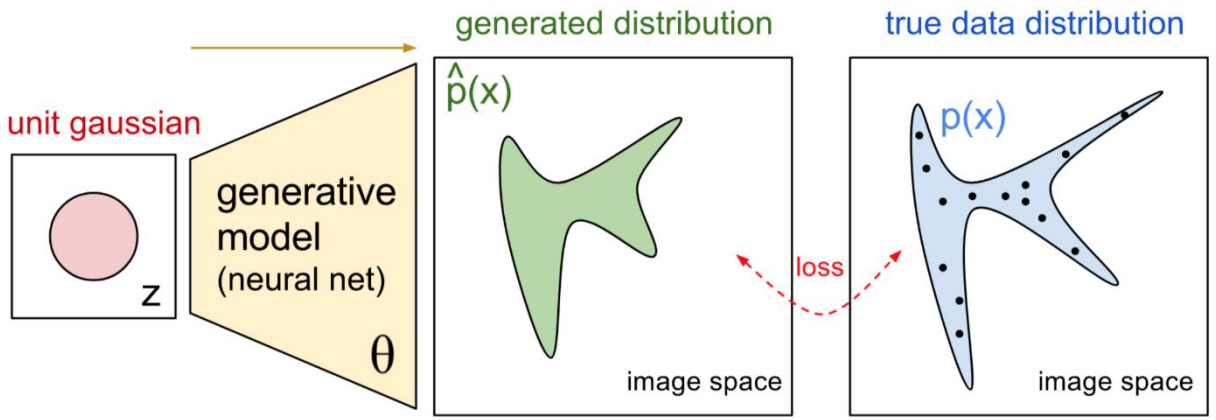
- Generative Models
  - 데이터 생성의 과정: $P(Y) \rightarrow P(X|Y)$
  - 실제 데이터의 분포를 토대로 데이터 생성
    - 실제 데이터의 분포를 학습
  - 비교적 복잡하고, 실제 현상에 대한 가정이 필요



Generative Models(생성 모델)

- 생성 모델: 몇가지 가정 하에 실제 데이터의 분포를 학습
  - 학습한 데이터의 확률분포로부터 데이터를 생성





Adversarial Networks

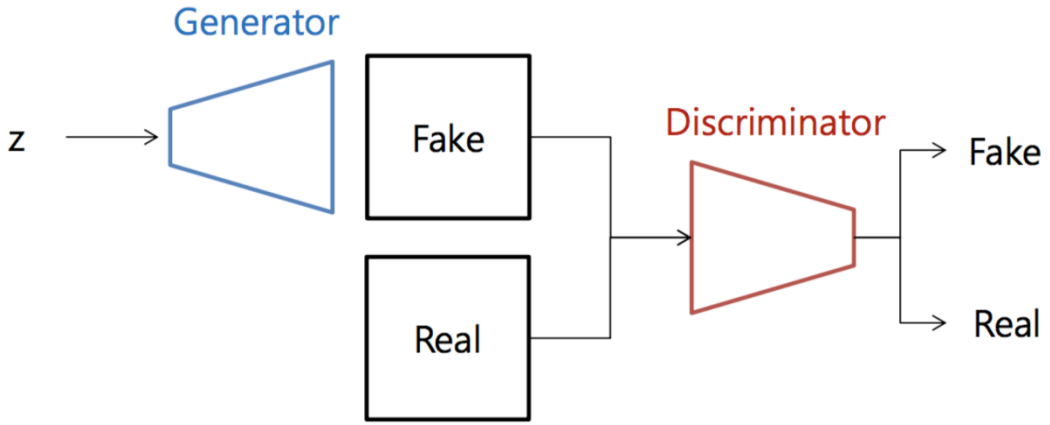
- 대립 신경망(Adversarial Networks)
  - 서로 다른 다수의 신경망이 경쟁하면서(적대적으로) 학습하는 방식

- “Generative” Adversarial Network(GAN)
  - 생성 모델(generator)
    - 실제 데이터와 비슷한 데이터를 생성하는 모델
    - 실제 데이터의 분포를 근사 -> 분포에 따른 샘플 반환
  - 판별 모델(discriminator)
    - 입력이 실제 데이터인지, generator가 생성한 데이터인지를 판별

각 신경망의 목표

- Discriminator
  - Generator의 생성결과(fake)와 실제 데이터(real)를 제대로 분류하는 것
  - $\mathcal{L}_D = -(\log(real) + \log(1 - fake))$
    - $real = D(x) \rightarrow 1,\ fake = D(G(z)) \rightarrow 0$

- Generator
  - Discriminator를 속이는 것
    - $fake = D(G(z)) \rightarrow 1$



- Optimization
  - $\min_G \max_D \left(E_{x \sim p_{real}} [\log(D(x))] + E_{z \sim N(0,1)} [\log(1 - D(G(z)))] \right)$

[Ian J. Goodfellow et al., NIPS, 2014]

알고리즘

Algorithm 1 Minibatch stochastic gradient descent training of generative adversarial nets. The number of steps to apply to the discriminator, k, is a hyperparameter. We used $k = 1$, the least expensive option, in our experiments.

for number of training iterations **do**

for k steps **do**

- Sample minibatch of m noise samples $\{\boldsymbol{z}^{(1)}, \ldots, \boldsymbol{z}^{(m)}\}$ from noise prior $p_g(\boldsymbol{z})$.
- Sample minibatch of m examples $\{\boldsymbol{x}^{(1)}, \ldots, \boldsymbol{x}^{(m)}\}$ from data generating distribution $p_{\text{data}}(\boldsymbol{x})$.
- Update the discriminator by ascending its stochastic gradient:

$$\nabla_{\theta_d} \frac{1}{m} \sum_{i=1}^{m} \left[\log D\left(\boldsymbol{x}^{(i)}\right) + \log\left(1 - D\left(G\left(\boldsymbol{z}^{(i)}\right)\right)\right) \right].$$

end for

Discriminator

- Sample minibatch of m noise samples $\{\boldsymbol{z}^{(1)}, \ldots, \boldsymbol{z}^{(m)}\}$ from noise prior $p_g(\boldsymbol{z})$.
- Update the generator by descending its stochastic gradient:

$$\nabla_{\theta_g} \frac{1}{m} \sum_{i=1}^{m} \log\left(1 - D\left(G\left(\boldsymbol{z}^{(i)}\right)\right)\right).$$

Generator

end for

The gradient-based updates can use any standard gradient-based learning rule. We used momentum in our experiments.

Conditional GAN (cGAN)

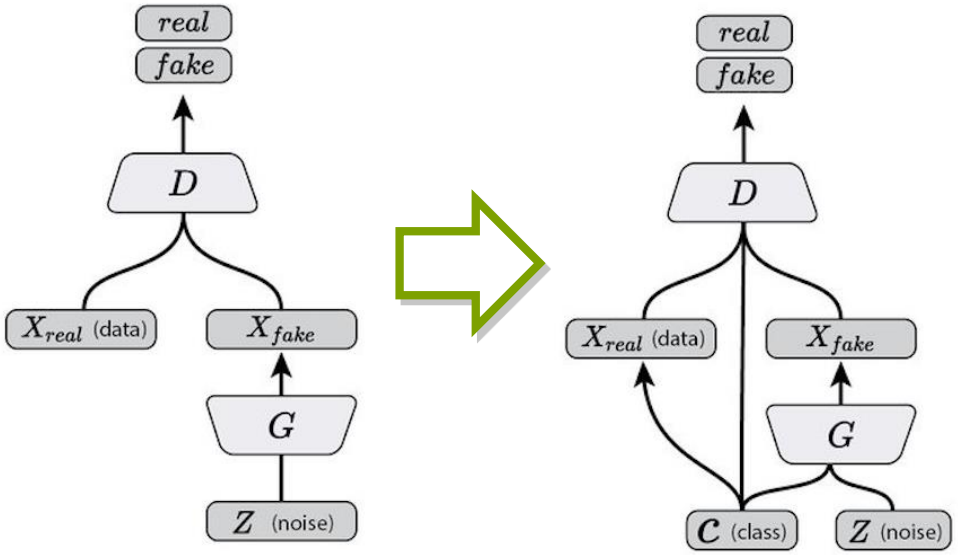
- 기본적으로 GAN은 class 정보에 대한 처리 과정이 없음
  - 숫자 6을 생성하고 싶어도, 명시적으로 6을 생성할 수 없음

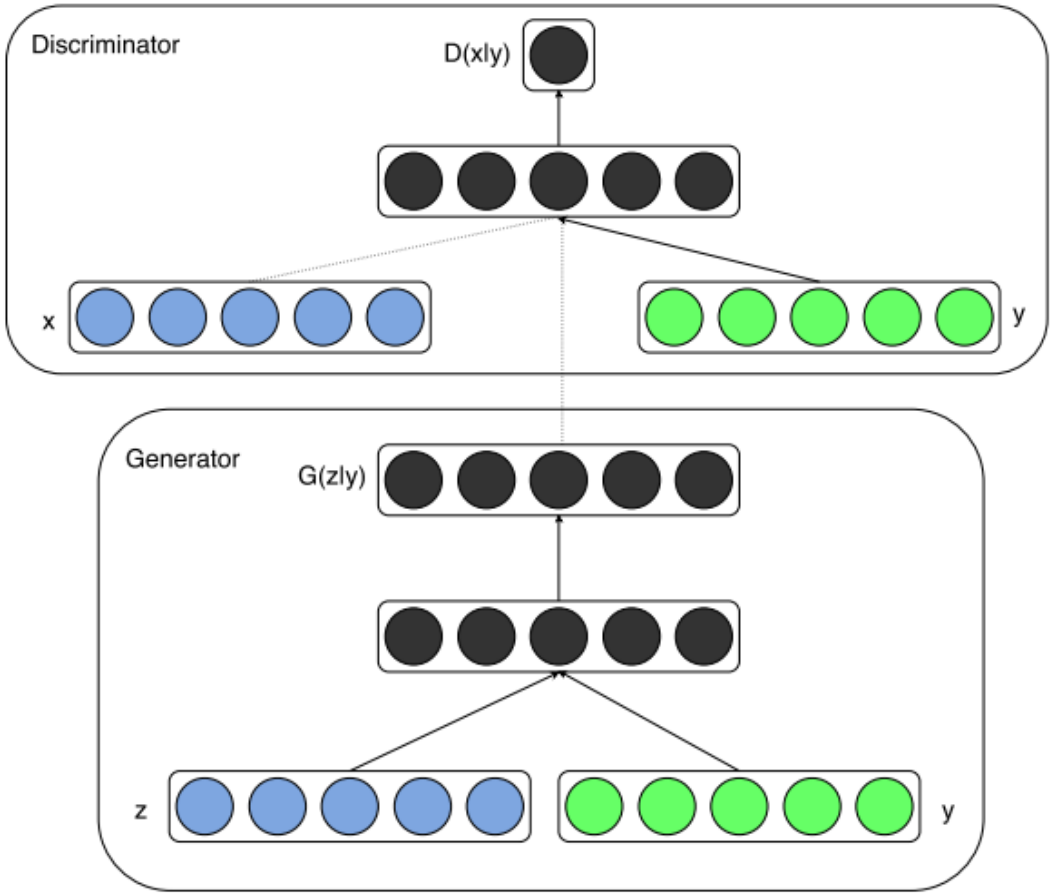
- cGAN – 추가적인 제약 정보를 GAN에 적용
  - Generator, discriminator가 데이터를 생성하고 분류할 때 참고할 정보를 부가적으로 입력
  - 레이블, 특성, feature 등

  - $\min_G \max_D V(D,G) = E_{x \sim p_{data}(x)}[\log D(x,y)] + E_{z \sim p_z(z)}[\log\{1 - D(G(z,y),y)\}]$

[Mehdi Mirza et al., 2014]

Conditional GAN (cGAN)





Vanilla GAN의 학습 불안정성

- **Non-convergence**
  - Generator, discriminator가 번갈아 학습하면서, 파라메터가 수렴하지 않음
- **Mode collapse**
  - Generator가 모든 data를 생성하는 것이 아닌, 특정 데이터만 생성
- **Diminished gradient**
  - Discriminator가 generator에 비해 성능이 너무 좋아, generator를 학습시키는 gradient가 0으로 사라짐
  - Loss function에 사용되는 distance metric이 부적절하여 의미 있는 gradient를 생성하지 못함
- Highly sensitive to the hyperparameter selections

GAN zoo

- GAN의 불안정성을 해결하거나 각종 기능을 추가한 모델들의 리스트
  - https://deephunt.in/the-gan-zoo-79597dc8c347

  - DCGAN - convolution layer 적용, 가장 기본이 되는 모델
  - Stabilized training – Unrolled GAN, LSGAN, WGAN(-GP), SN-GAN, …
  - Image translation – Pix2Pix, CycleGAN, StarGAN, …
  - Latent disentanglement – cGAN, InfoGAN, …
  - Quality - Progressively Growing GANs, BigGAN, …

  - 등등..

Codes

- 실습 코드 Github 주소
  - https://github.com/illhyhl1111/SNU_ML2019

- Colab 링크
  - https://colab.research.google.com/github/illhyhl1111/SNU_ML2019/blob/master/Lab3_LSTM.ipynb
    https://colab.research.google.com/github/illhyhl1111/SNU_ML2019/blob/master/Lab3_MNIST_cGAN.ipynb