



데이터사이언스개론

Data

[Similarity](#)

[Correlation](#)

[Sampling](#)

Classification

[Decision Tree](#)

[K Nearest Neighbor](#)

[Bayesian](#)

[Rule-based](#)

2. Image Classification Pipeline

3. Loss Functions and Optimization

Loss Functions

[Support Vector Machine \(SVM\)](#)

[Softmax](#)

[Full loss](#)

[L1 \(MAE\)](#)

[L2 \(MSE\)](#)

Optimization

4. Backpropagation and NN

Backpropagation

[Activation functions](#)

5. CNN

[Stride](#)

[Padding](#)

[Pooling \(Subsampling\)](#)

6+7. Training NN

[Activation functions](#)

[Weight Init](#)

[Batch Regularization](#)

[Learning Rate](#)

[Dropout](#)

8. Deep Learning Software

9. CNN Architectures

[LeNet](#)

[AlexNet](#)

ZFNet
VGGNet
GoogleNet
ResNet
Survey
Others

10. RNN

11. Detection and Segmentation

Sematic segmentation
Classification + Localization

R-CNN
Fast R-CNN
Faster R-CNN
YOLO & SSD
Mask R-CNN

12. Visualizing and Understanding

13. Generative Models

Fully visible belief network
Pixel(R|C)NN & VAE & GANs

14. Reinforcement Learning

Markov decision process (MDP)
Value function & Q-value function
Solving for the optimal policy: Q-learning
Q-network Architecture
Policy Gradients

15. Adversarial Examples & Training

Ensemble Techniques
Imbalanced Class Problem

Confusion matrix
Cost matrix

Association Analysis Basic Concepts

Cluster Analysis

K-Means
Hierarchical Clustering
DBSCAN

Data

- 데이터는 attribute와 object의 집합
 - **attribute:** columns, variables, fields, characteristics, dimensions, features

- **object**: rows, recordds, points, cases, samples, enttities, instances
- attribute의 종류
 - nominal: 순서가 없는 카테고리컬 데이터, ID, 우편번호
 - ordinal: 순서가 있는 데이터, 순위, 성적, 키
 - interval: 구간 데이터, 캘린더 날짜, 온도
 - ratio: 비율 데이터, 캘빈 온도, 길이, 개수
- **curse of dimensionality**: attribute가 증가함에 따라 상대적으로 훈련 샘플이 줄어들어 모델 성능이 안 좋아지는 문제.
- 데이터 품질 문제
 - noise: 정상적이지 않은 값. 설문조사에 모두 0으로 응답하는 경우.
 - outliers: 평균치에서 크게 벗어나는 값. 소득 평균에 빌게이츠를 포함하는 경우.
 - noise는 없을수록 좋지만, outlier는 흥미로운 요소일 수 있음.
 - 누락, 중복, 오류(wrong), 가짜(fake) 등

Similarity

- euclidean distance: $d(x, y) = \sqrt{\sum_{k=1}^n (x_k - y_k)^2}$
- minkowski distance: $d(x, y) = (\sum_{k=1}^n |x_k - y_k|^r)^{1/r}$
 - 모든 거리의 차이가 아닌 최대 차이만 고려하는 것. 맨해튼 거리와 euclidean
 - $r = 1$: 맨해튼 거리 (그리드, 직각 거리)
 - $r = 2$: euclidean distance (직선 거리)
 - $r \rightarrow \infty$: supremum distance (최대 거리)
- 데이터 오브젝트가 binary attributes로 구성된 경우 유사도:
 - $f_{01}, f_{10}, f_{11}, f_{00}$ where f_{xy}
 - simple matching coefficient (SMC): $\frac{f_{11} + f_{00}}{f_{01} + f_{10} + f_{11} + f_{00}}$
 - jaccard coefficient (J): $\frac{f_{11}}{f_{01} + f_{10} + f_{11}}$

$\mathbf{x} = \begin{matrix} 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 \end{matrix}$

$f_{01} = 2$ (the number of attributes where \mathbf{x} was 0 and \mathbf{y} was 1)

$f_{10} = 1$ (the number of attributes where \mathbf{x} was 1 and \mathbf{y} was 0)

$f_{00} = 7$ (the number of attributes where \mathbf{x} was 0 and \mathbf{y} was 0)

$f_{11} = 0$ (the number of attributes where \mathbf{x} was 1 and \mathbf{y} was 1)

$$\begin{aligned} \text{SMC} &= (f_{11} + f_{00}) / (f_{01} + f_{10} + f_{11} + f_{00}) \\ &= (0+7) / (2+1+0+7) = 0.7 \end{aligned}$$

$$J = (f_{11}) / (f_{01} + f_{10} + f_{11}) = 0 / (2 + 1 + 0) = 0$$

Correlation

- -1이면 반비례, 0이면 상관관계 없음, 1이면 정비례.
- 상관계수: $\frac{\sum(x_i - \bar{x})(y_i - \bar{y})}{(n-1) \times \sigma_x \times \sigma_y}$

Sampling

- 전통적인 통계학에서는 모든 데이터를 조사하는 비용 때문에 샘플링.
- 빅데이터에서는 사용 가능한 데이터가 너무 많아서 샘플링.

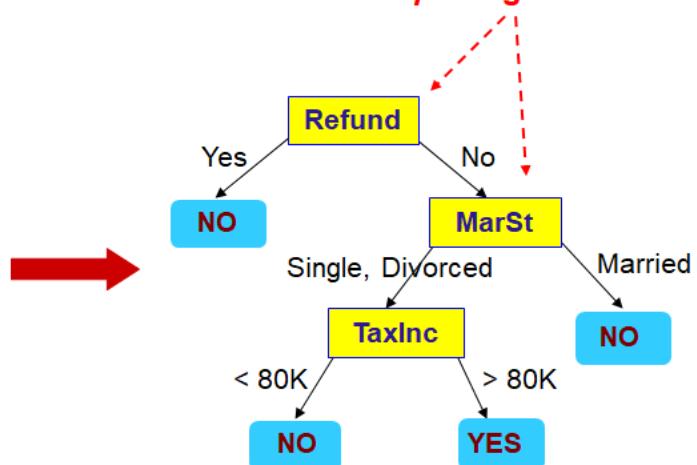
Classification

- 주어진 데이터 x 를 y 로 매핑하는 모델.

Decision Tree

Tid	Refund	Marital Status	Taxable Income	Cheat
1	Yes	Single	125K	No
2	No	Married	100K	No
3	No	Single	70K	No
4	Yes	Married	120K	No
5	No	Divorced	95K	Yes
6	No	Married	60K	No
7	Yes	Divorced	220K	No
8	No	Single	85K	Yes
9	No	Married	75K	No
10	No	Single	90K	Yes

Splitting Attributes



Training Data

Model: Decision Tree

- 좋은 트리는 결과를 한쪽으로 쏠리게 하는 트리.
- 불순도를 측정하고 불순도가 낮은 트리를 선택해야 한다.
- Gini Index: $1 - \sum_{i=0}^{c-1} p_i(t)^2$

C1	0
C2	6

$$P(C1) = 0/6 = 0 \quad P(C2) = 6/6 = 1$$

$$\text{Gini} = 1 - P(C1)^2 - P(C2)^2 = 1 - 0 - 1 = 0$$

C1	1
C2	5

$$P(C1) = \frac{1}{6} \quad P(C2) = \frac{5}{6}$$

$$\text{Gini} = 1 - (1/6)^2 - (5/6)^2 = 0.278$$

C1	2
C2	4

$$P(C1) = 2/6 \quad P(C2) = 4/6$$

$$\text{Gini} = 1 - (2/6)^2 - (4/6)^2 = 0.444$$

- Entropy: $-\sum_{i=0}^{c-1} p_i(t) \log_2 p_i(t)$

K Nearest Neighbor

- 비슷한 데이터를 한 클래스로 구분하는 모델.
- 중요한 attributes만 사용한다: PCA
- KNN 모델은 주변 K개 요소의 클래스를 보고 자신의 클래스를 결정한다.

Bayesian

- 베이즈 정리: $P(Y|X) = \frac{P(X|Y)P(Y)}{P(X)}$
- $P(X|Y)$ 를 알고 있다면 베이즈 정리를 이용해 $P(Y|X)$ 를 구할 수 있다.
- 어차피 분모는 같고, 분자 크기를 비교하는게 중요하므로 분모는 신경쓰지 않아도 됨.

Tid	Refund	Marital Status	Taxable Income	Evade
1	Yes	Single	125K	No
2	No	Married	100K	No
3	No	Single	70K	No
4	Yes	Married	120K	No
5	No	Divorced	95K	Yes
6	No	Married	60K	No
7	Yes	Divorced	220K	No
8	No	Single	85K	Yes
9	No	Married	75K	No
10	No	Single	90K	Yes

$$P(X | Yes) =$$

$$P(\text{Refund} = \text{No} | Yes) \times$$

$$P(\text{Divorced} | Yes) \times$$

$$P(\text{Income} = 120K | Yes)$$

여기서 멈추면 안 된다!
베이즈 정리를 사용하는 것이므로,
 $P(X | Yes)$ 에 $P(Y)$ 를 곱해줘야 함.

$$P(X | No) =$$

$$P(\text{Refund} = \text{No} | No) \times$$

$$P(\text{Divorced} | No) \times$$

$$P(\text{Income} = 120K | No)$$

- $P(X|Yes)$ 에서 $P(Y)$ 는 $P(Yes) = \frac{7}{10}$

Rule-based

- if ... then ... 규칙으로 데이터를 분류하는 방식.
- 아무것도 없는 상태에서 규칙을 만들 수도, 이미 특정된 규칙을 바탕으로 일반화된 규칙을 만들 수도 있음.

2. Image Classification Pipeline

- hyperparameters: 학습해서 얻는 게 아니라 우리가 조정하는 인자.
 - KNN에서 k 값이나 거리 계산 방식.

3. Loss Functions and Optimization

Loss Functions

Support Vector Machine (SVM)

$$L = \sum_{j \neq y_i} \max(0, s_j - s_{y_i} + 1)$$

Suppose: 3 training examples, 3 classes.

With some W the scores $f(x, W) = Wx$ are:



cat	3.2	1.3	2.2
car	5.1	4.9	2.5
frog	-1.7	2.0	-3.1
Losses:	2.9		

Multiclass SVM loss:

Given an example (x_i, y_i) where x_i is the image and where y_i is the (integer) label,

and using the shorthand for the scores vector: $s = f(x_i, W)$

☞ Multiclass SVM loss

the SVM loss has the form:

$$\begin{aligned} L_i &= \sum_{j \neq y_i} \max(0, s_j - s_{y_i} + 1) \\ &= \max(0, 5.1 - 3.2 + 1) \\ &\quad + \max(0, -1.7 - 3.2 + 1) \\ &= \max(0, 2.9) + \max(0, -3.9) \\ &= 2.9 + 0 \\ &= 2.9 \end{aligned}$$

- 위에서 이미지의 정답은 cat. 따라서 s_{y_i} 는 3.2다.
- 만약 제곱한다면 0에서 1 사이값은 더 작아지고, 1 이상의 값은 더 커짐. loss가 큰 요소에 최적화됨.

Softmax

$$L = -\log\left(\frac{e^{s_{y_i}}}{\sum_j e^{s_j}}\right)$$

- SVM보다 계산이 복잡하지만 정밀하다.

Full loss

$$L = \frac{1}{N} \sum_{i=1}^N L_i + R(W)$$

L1 (MAE)

$$L = \sum_{i=1}^n |y_i - f(x_i)|$$

- 실제 값과 예측 값 사이의 차이에 절대값을 취해 그 오차의 합을 loss로 사용.

L2 (MSE)

$$L = \sum_{i=1}^n (y_i - f(x_i))^2$$

- LSE(Least square error). L1에 비해 robustness가 적다. 그래서 outlier의 효과를 무시하고 싶다면 L1을, outlier에 주목할 필요가 있다면 L2를 사용한다.

Optimization

- Gradient descent: 예측값과 정답값의 오차를 최소로 만들기 위해 loss function의 최소값을 찾아가는 방법.
- **Stochastic Gradient Descent (SGD)**
 - 배치 크기가 1인 경사하강법.
 - 전체 배치를 보고 경사를 계산하는 것은 너무 비용이 큼.
 - 데이터셋에서 무작위로 uniform하게 선택한 minibatch에 의존해 각 단계의 예측 경사를 계산.
 - 아래와 같은 과정을 거친다.

Loop: ↗ 어떤가요!

1. **Sample a batch of data** ↗ 짐작을 대체하고 이를 위한 샘플링
2. **Forward prop it through the graph** (network), get loss
3. **Backprop to calculate the gradients**
4. **Update the parameters using the gradient**

1. 데이터 전체가 아니라 일부 배치를 샘플링한다.
2. 네트워크를 순전파하고 loss를 구한다.
3. 역전파하여 경사도를 계산한다.

- 4. 경사도를 이용해 파라미터를 업데이트한다.

4. Backpropagation and NN

Backpropagation

- 계산 결과와 정답의 오차를 구해 이 오차에 관여하는 값들의 가중치를 수정하여 오차가 작아지는 방향으로 일정 횟수를 반복해 수정하는 방법.

Backpropagation: a simple example

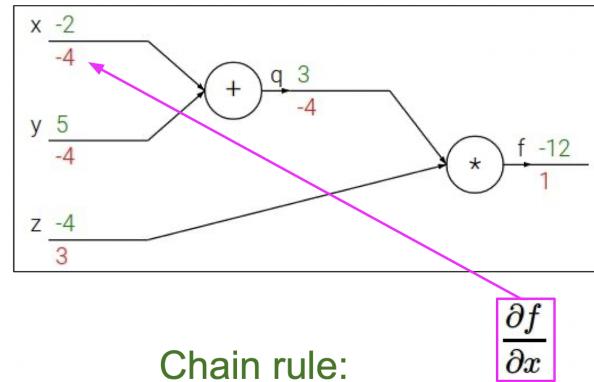
$$f(x, y, z) = (x + y)z$$

e.g. $x = -2, y = 5, z = -4$

$$q = x + y \quad \frac{\partial q}{\partial x} = 1, \frac{\partial q}{\partial y} = 1$$

$$f = qz \quad \frac{\partial f}{\partial q} = z, \frac{\partial f}{\partial z} = q$$

Want: $\frac{\partial f}{\partial x}, \frac{\partial f}{\partial y}, \frac{\partial f}{\partial z}$



Chain rule:

$$\frac{\partial f}{\partial x} = \frac{\partial f}{\partial q} \frac{\partial q}{\partial x}$$

Activation functions

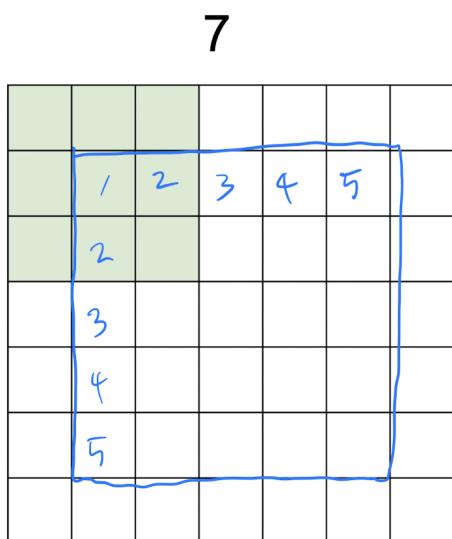
- 한 노드에 대해 입력값을 다음 노드에게 보낼지 결정하는 함수.

Summary so far...

- neural nets will be very large: impractical to write down gradient formula by hand for all parameters
- backpropagation** = recursive application of the chain rule along a computational graph to compute the gradients of all inputs/parameters/intermediates
- implementations maintain a graph structure, where the nodes implement the **forward()** / **backward()** API
- forward**: compute result of an operation and save any intermediates needed for gradient computation in memory \rightsquigarrow Score
- backward**: apply the chain rule to compute the gradient of the loss function with respect to the inputs \rightsquigarrow Score \rightarrow weight

5. CNN

- 이미지에 filter를 적용해 activation map을 얻는다.
- 6개의 filter를 적용한다면 6개의 activation map이 만들어짐.



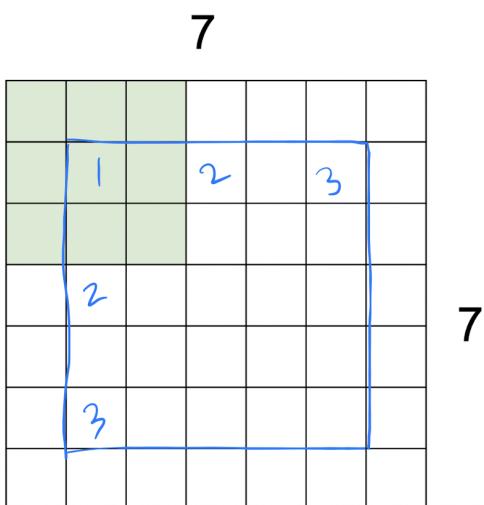
7x7 input (spatially)
assume 3x3 filter

7x7 image가 3x3 필터를 거치면
5x5 activation map이 된다.

- output activation map 크기: $N - F + 1$
- $7 - 3 + 1 = 5$

Stride

- 보폭을 조정하는 것.



7x7 input (spatially)
assume 3x3 filter
applied **with stride 2**

7x7 image가 3x3 필터를 stride 2로 거치면
3x3 activation map이 된다.

- stride 3로 적용하면? 불가능하다!
- output activation map 크기: $\frac{N-F}{\text{stride}} + 1$

- stride 1: $\frac{7-3}{1} + 1 = 5$
- stride 2: $\frac{7-3}{2} + 1 = 3$

Padding

- border를 0으로 채워 output 크기가 줄어드는 것을 방지.

0	0	0	0	0	0	0			
0	1	2	3	4	5	6	7		
0	2								
0	3								
0	4								
	5								
	6								
	7								

e.g. input 7x7

3x3 filter, applied with **stride 1**

pad with 1 pixel border => what is the output?

output 크기가 줄어드는 걸이 심각한

margin 영역은 0인 줄이기(padding)

(recall:)

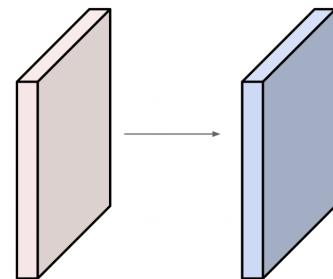
$$(N - F) / \text{stride} + 1$$

- output activation map 크기: $\frac{N+2P-F}{\text{stride}} + 1$
- 너무 중요한 예제: 필터 적용 후 output activation map의 크기

Examples time:

Input volume: **32x32x3**

10 5x5 filters with stride 1, pad 2



Output volume size:

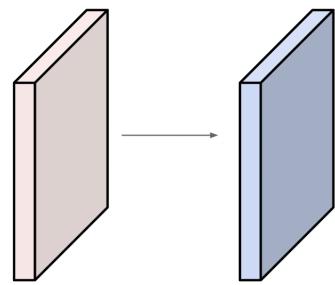
$$(32+2*2-5)/1+1 = 32 \text{ spatially, so}$$

32x32x10

필터가 10개인 10차원 output

- 너무 중요한 예제: 파라미터 개수

Examples time: 



Input volume: **32x32x3**

10 5x5 filters with stride 1, pad 2

Number of parameters in this layer?

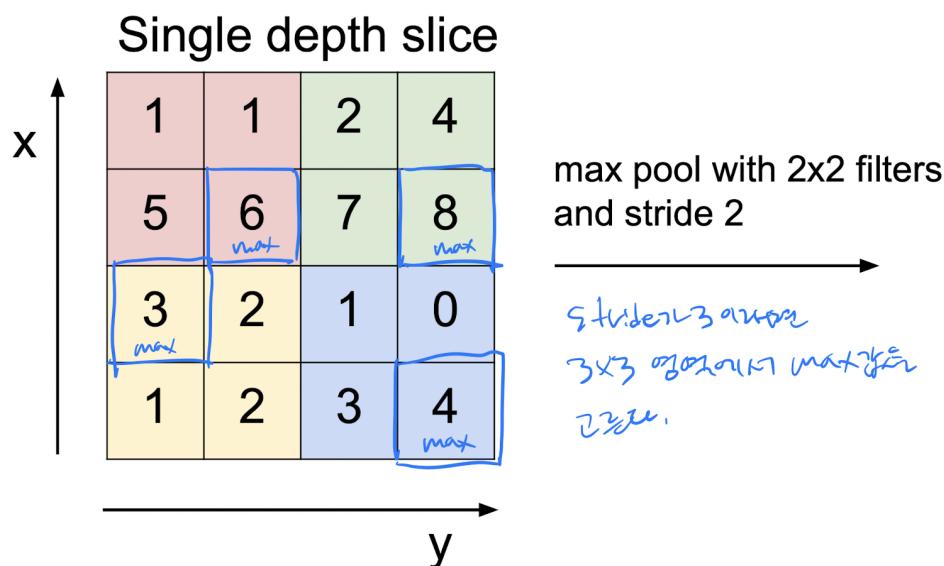
each filter has $5 \times 5 \times 3 + 1 = 76$ params (+1 for bias)

$$\Rightarrow 76 \times 10 = 760$$

- 각 필터의 파라미터 개수: $F^2 \times C$ (bias를 고려한다면 +1)
따라서 총 파라미터 개수는 위에 필터 개수를 곱한 값.

Pooling (Subsampling)

- 앞선 레이어들을 거치고 나온 데이터가 모두 필요한건 아님. 추론에는 적은 데이터만 있어도 충분할 수도.
- max pooling: 필터 영역에서 가장 큰 값만 골라 output을 만든다.



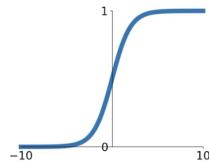
6	8
3	4

6+7. Training NN

Activation functions

Sigmoid

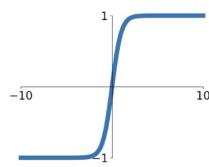
$$\sigma(x) = \frac{1}{1+e^{-x}}$$



tanh

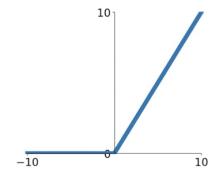
$$\tanh(x)$$

언어 학습에 사용되는 활성화 함수.
Sigmoid와 비슷한 특징.



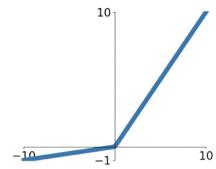
ReLU ~

$$\max(0, x)$$



Leaky ReLU

$$\max(0.1x, x)$$



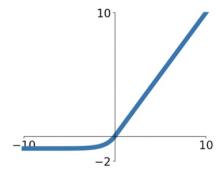
Maxout

$$\max(w_1^T x + b_1, w_2^T x + b_2)$$

성능을 좋지만 계산량이 많아.

ELU

$$\begin{cases} x & x \geq 0 \\ \alpha(e^x - 1) & x < 0 \end{cases}$$



- Sigmoid: 역사적으로 의미있는 함수. 요즘에는 잘 안 쓴다.
- tanh: sigmoid에 비해 saturated가 덜 되지만, 여전히 되긴 함.
- ReLU: 좋다. 근데 0 이하가 saturate됨.
- Leaky ReLU: saturate되지 않는다!
- Maxout: 성능은 좋은데 계산량이 너무 많다.
- ELU: ReLU의 모든 장점. exp 연산이 필요하다.

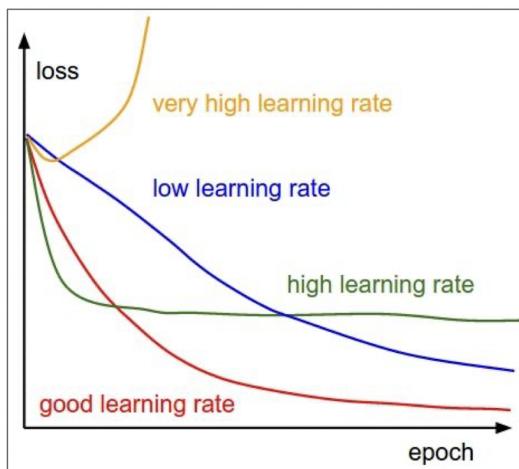
Weight Init

- 가중치를 어떻게 초기화할 것인가?
- too small: 1보다 작은 값으로 하면 0으로 수렴해서 학습이 안 됨.
- too big: activation이 saturate됨. 경사도가 0이 되어 학습이 안 됨.
- just right: 분포가 잘 되어 학습도 잘 됨.

Batch Regularization

- overfitting 방지를 위해 regularization을 한다. 가중치의 값이 커지지 않도록 제한하는 것.

Learning Rate



Q: Which one of these learning rates is best to use?

학습률은 high, 저로 가면 low가 좋다.
(high는 good은 좋지 않아서 좋지 않다.)

Dropout

- 랜덤하게 일정 노드를 dropout하면 오버피팅이 개선된다.

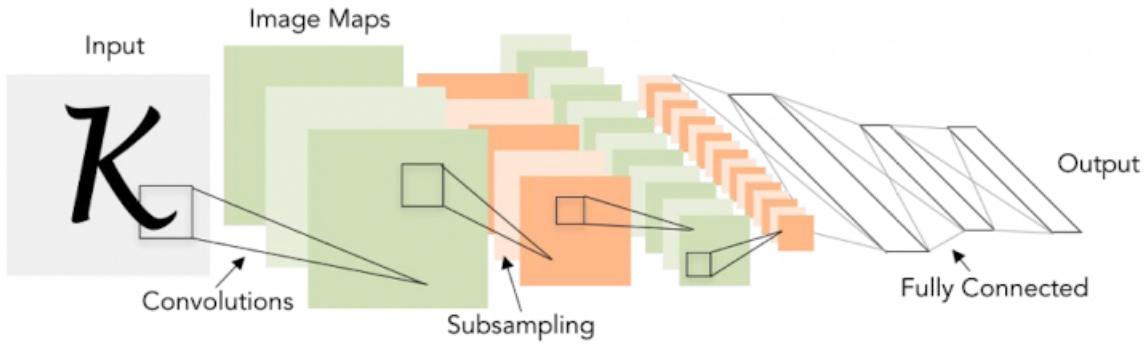
8. Deep Learning Software

- PyTorch, Keras, TensorFlow, Caffe 등...
- TF: 대부분의 프로젝트에서 안전함. 완벽하진 않지만 큰 커뮤니티, 광범위한 유즈케이스가 있음. Keras, Sonnet과 같은 하이 레벨 래퍼가 있음.
- PyTorch: 연구에 베스트. 아직은 새로움.
- Caffe: 코드 없이 json이랑 비슷한 prototxt로 모델을 정의. 코드가 방대함.

9. CNN Architectures

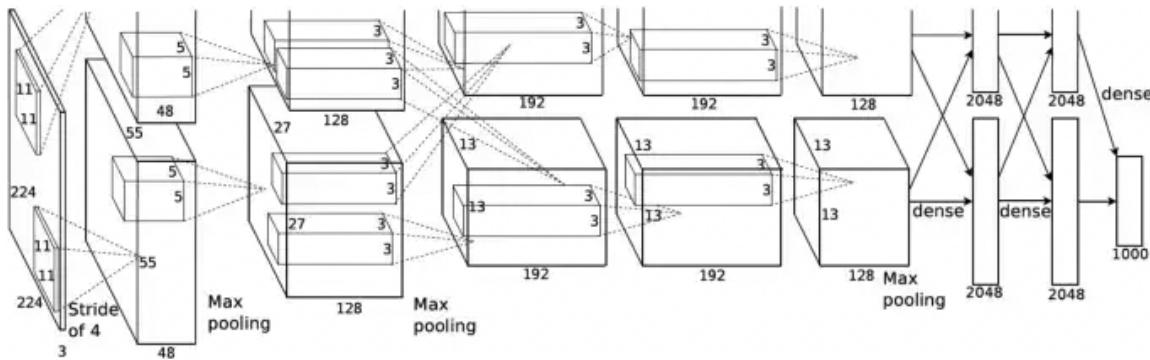
LeNet

- 산업에 성공적으로 적용된 최초의 ConvNet. 이미지를 입력으로 받아 stride = 1인 5x5 필터를 거치고, 몇 개의 conv layer와 pooling layer를 거친다. 마지막으로 FC layer를 거친다.



AlexNet

- 최초의 large scale CNN. conv-pool-normalization 구조를 두 번 반복.



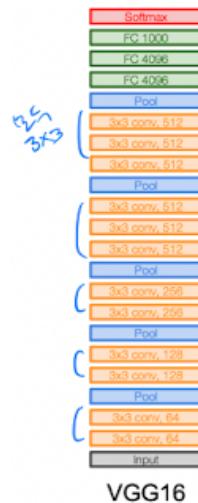
- CONV1 - MAX POOL1 - NORM1 - CONV2 - MAX POOL2 - NORM2 - CONV3 - CONV4 - CONV5 - MAX POOL3 - FC6 - FC7 - FC8
- 입력이 227x227x3 이미지일 때:
 - AlexNet의 첫 레이어 CONV1은 stride가 4인 96개의 11x11 필터. 이때 출력 사이즈는?
 - $(227 - 11)/4 + 1 = 55$
 - 따라서 출력 볼륨은 55x55x96.
 - 이때 파라미터 수는 $(11 * 11 * 3) * 96 = 34848 \sim 35K$
 - 두 번째 레이어 POOL1은 stride가 2인 3x3 필터. 이때 출력 사이즈는?
 - $(55 - 3)/2 + 1 = 27$
 - 따라서 출력 볼륨은 27x27x96.
 - 이때 파라미터 수는 0.
 - 학습되는 게 없기 때문. 애초에 POOL 레이어는 학습이 아니라 크기를 줄이는 용도.

ZFNet

- AlexNet의 하이퍼 파라미터를 조정해 개선한 아키텍처.
- CONV1을 7×7 stride 2로, CONV3,4,5를 521, 1024, 512개 필터로 변경.

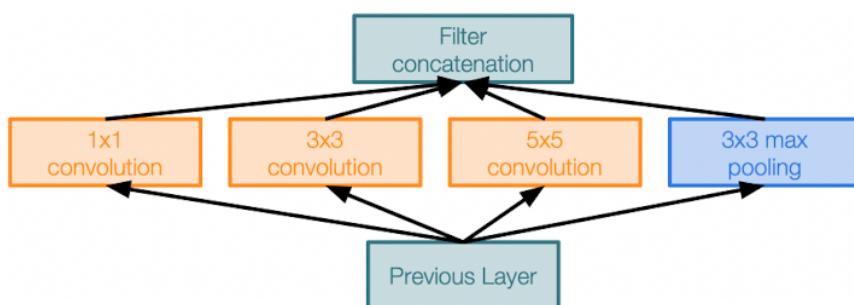
VGGNet

- 더욱 작고 깊은 네트워크. 16~19개 레이어.
- 3×3 CONV stride 1, pad 1 레이어와 2×2 MAX POOL stride 2 레이어.
- 작은 레이어를 사용하면 파라미터의 수를 줄일 수 있음.
 - 3×3 CONV를 여러 개 쌓으면 7×7 CONV와 같은 효율을 낼 수 있음.
- 더 깊은 아키텍처임에도 더 적은 파라미터를 사용함.
3개의 3×3 레이어이므로, $3 \times (3^2 C^2)$ 개. 이때 C는 레이어당 채널 수.

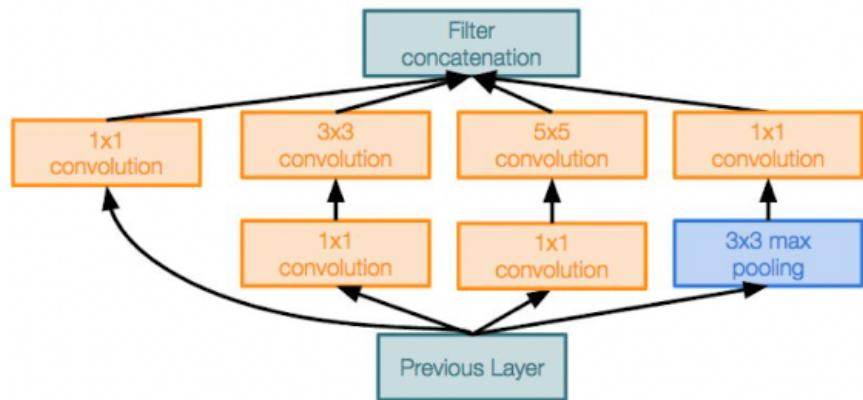


GoogleNet

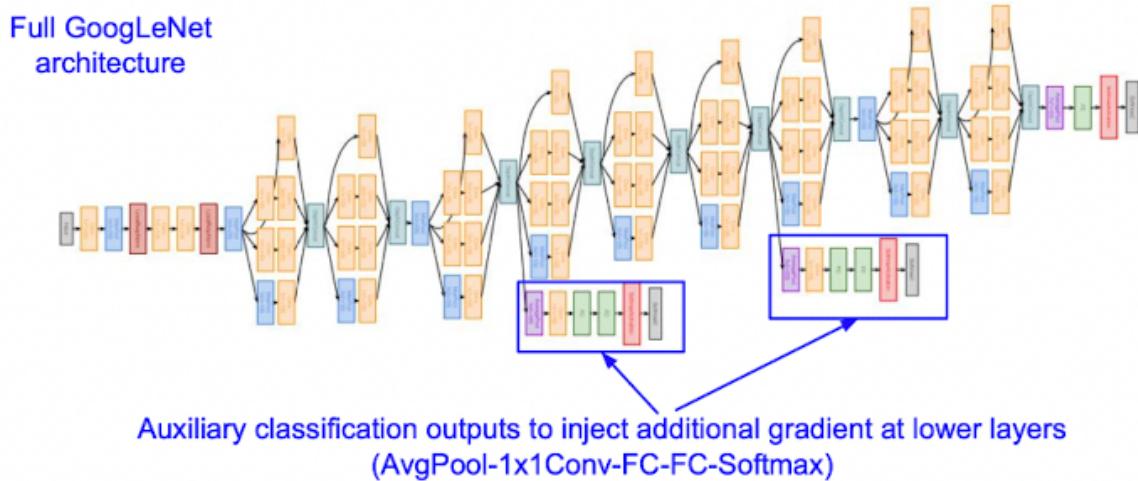
- 높은 계산량을 효율적으로 수행하는 22개 레이어의 깊은 네트워크.
- Inception module:



- 네트워크 안의 네트워크
- 동일한 입력을 받는 서로 다른 다양한 필터들이 병렬적으로 연결되어 있음.
- 다양한 연산을 수행하고 이를 하나로 합친다. 하지만 computational complexity가 커지는 문제가 있음. 레이어를 거칠수록 depth가 더 커짐.
- bottleneck layer: 연산 복잡도가 커지는 문제를 해결하기 위한 레이어.

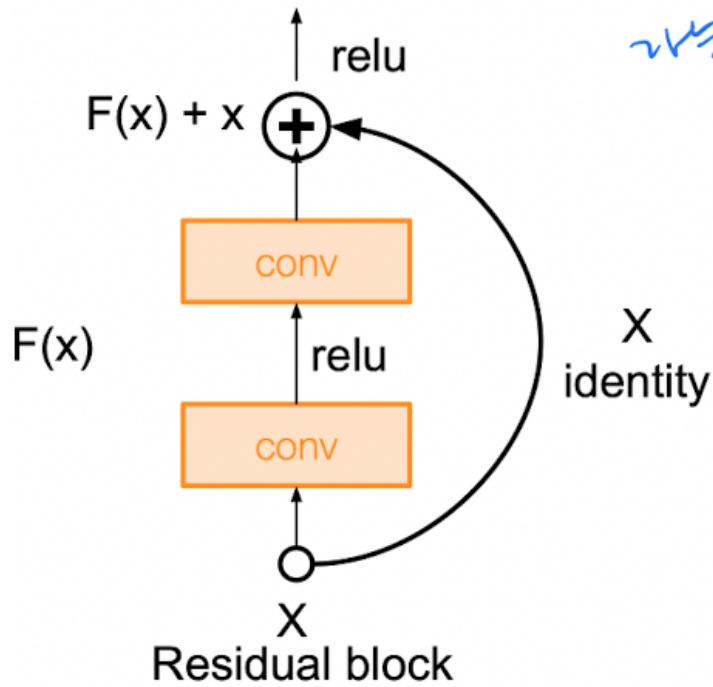


- conv 연산 전에 1x1 conv 레이어를 추가해 입력의 depth를 더 낮은 차원으로 투영 한다.
- 파라미터 수를 줄이기 위해 마지막에는 FC 레이어가 아닌 클래스 분류기가 있음. 또한 네트워크가 깊은 경우 그래디언트 신호가 점점 작아지기 때문에 중간중간 보조 분류기 를 배치해서 하위 레이어에 추가적인 그래디언트 신호를 주입한다.

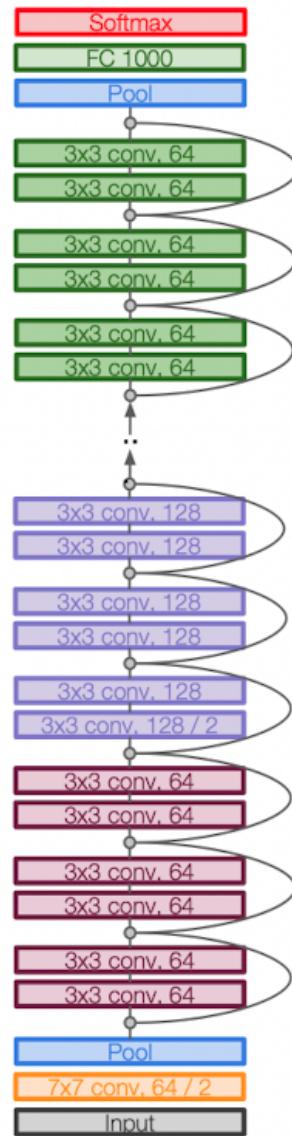


ResNet

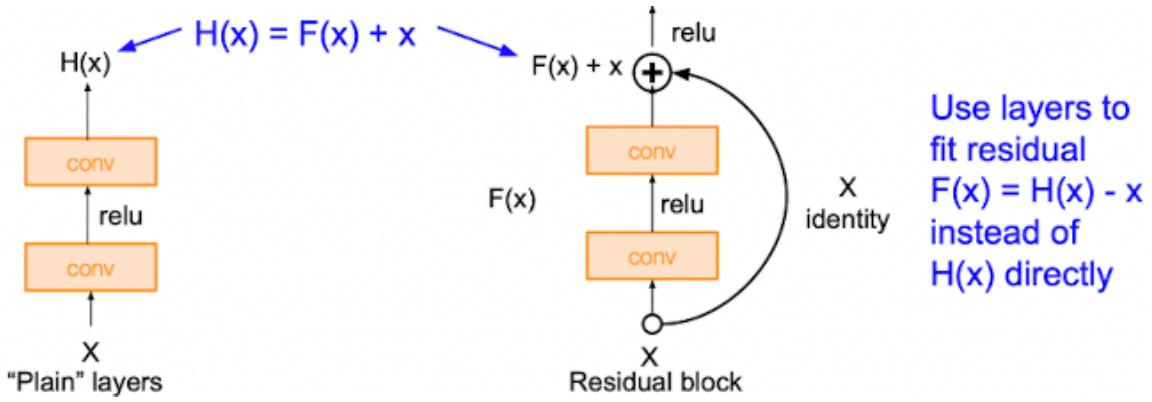
- 혁명적으로 깊이가 깊어진 아키텍처. (152 레이어)



기존의 input을 모두 합친다음에 예상한,
이 예상치와 차이를 차이(Residual)라고 한다



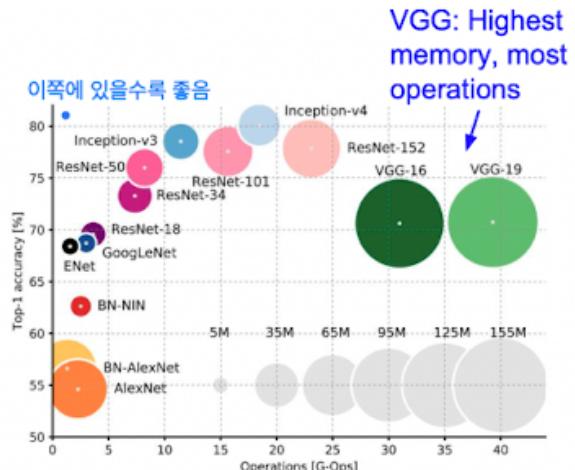
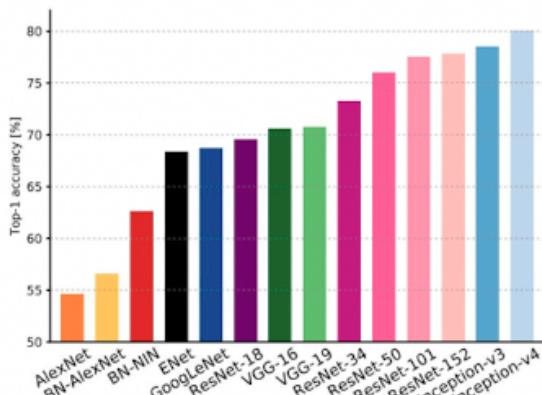
- 레이어가 깊을 수록 오버피팅이 일어날 것이라고 예상. 근데 실제로 해보니 트레이닝 에러가 높게 나옴.
 - 깊은 모델을 학습하는 경우 최적화에 문제가 생긴다.
 - 그래서 레이어를 그냥 잇는 direct mapping이 아닌 residual mapping을 한다.



- 전 레이어에서 흘러온 입력을 그대로 레이어의 출력에 더하는 skip connection이 필요.
- 레이어는 $H(x)$ 가 아닌 $H(x) - x$ (그림에서 $F(x)$)만 학습하면 됨.
따라서 최종 출력 값은 입력 $X + \text{잔차(residual)} F(x)$ 가 된다.
- 정확도가 중요하다면 더 깊게 구성할 수도 있음. 하지만 더 많은 리소스가 필요.

Survey

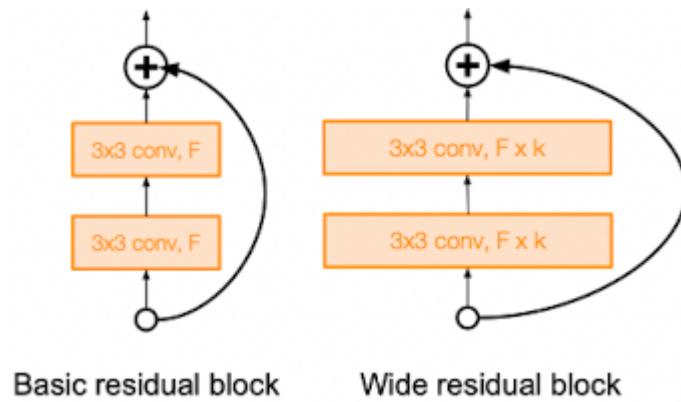
Comparing complexity...



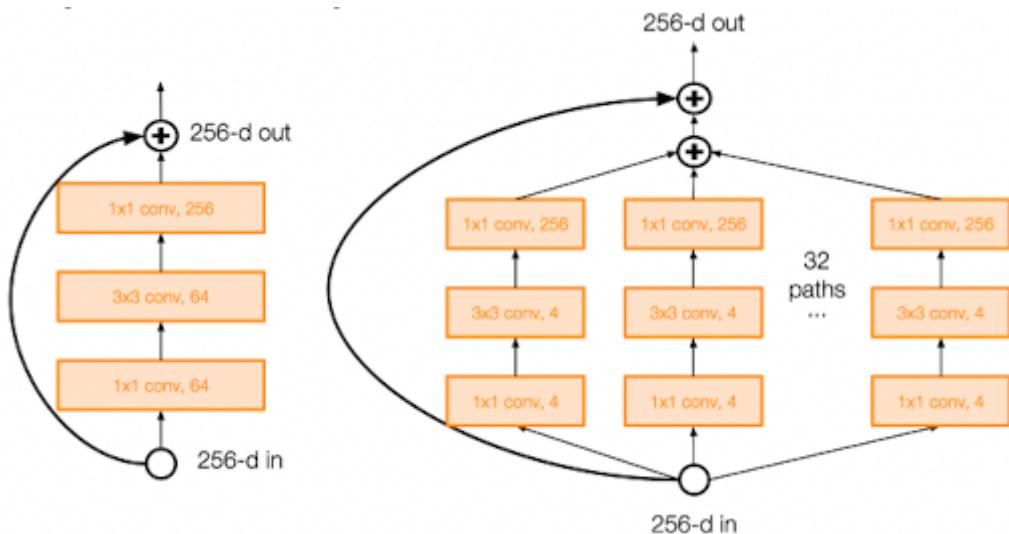
An Analysis of Deep Neural Network Models for Practical Applications, 2017.

Others

- NiN(Network in Network)
- Identity Mappings in Deep Residual Networks
- Wide Residual Networks: 레이어가 넓어지면 네트워크가 깊지 않아도 된다.



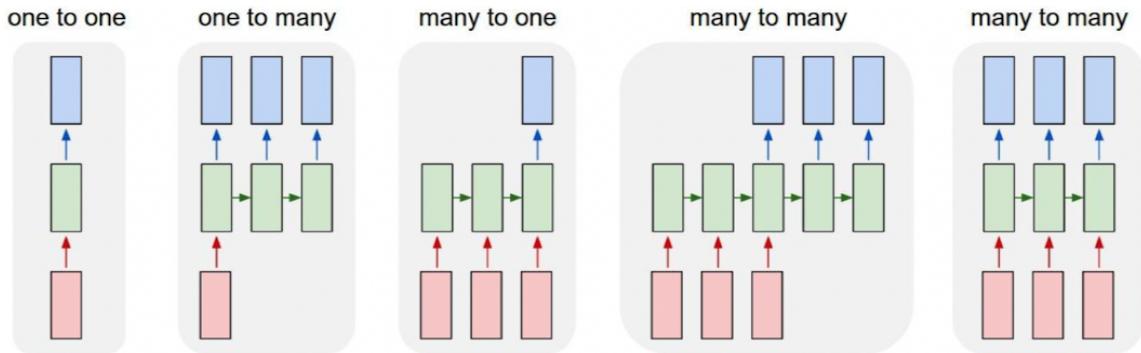
- ResNeXt: Wide ResNet의 width와 GoogleNet의 inception module을 합친 느낌.



- Deep Networks with Stochastic Depth: 네트워크가 짧으면 트레이닝이 더 잘 될 수 있음. 그래서 랜덤하게 일부 레이어를 drop out하는 ResNet이라고 생각하면 됨.
- FractalNet: ResNet과 달리 residual connection이 전혀 없음.
- DenseNet
- SqueezeNet

10. RNN

- 지금까지 본 네트워크는 하나의 입력으로 하나의 출력을 내는 one-to-one 구조.
- RNN은 다양한 I/O가 가능.

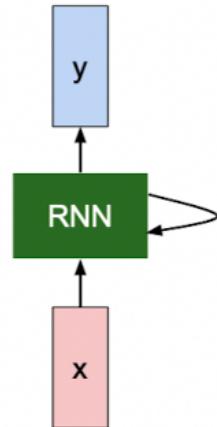


- 입력 x 가 들어올 때마다 RNN 내부의 hidden state가 업데이트된다.

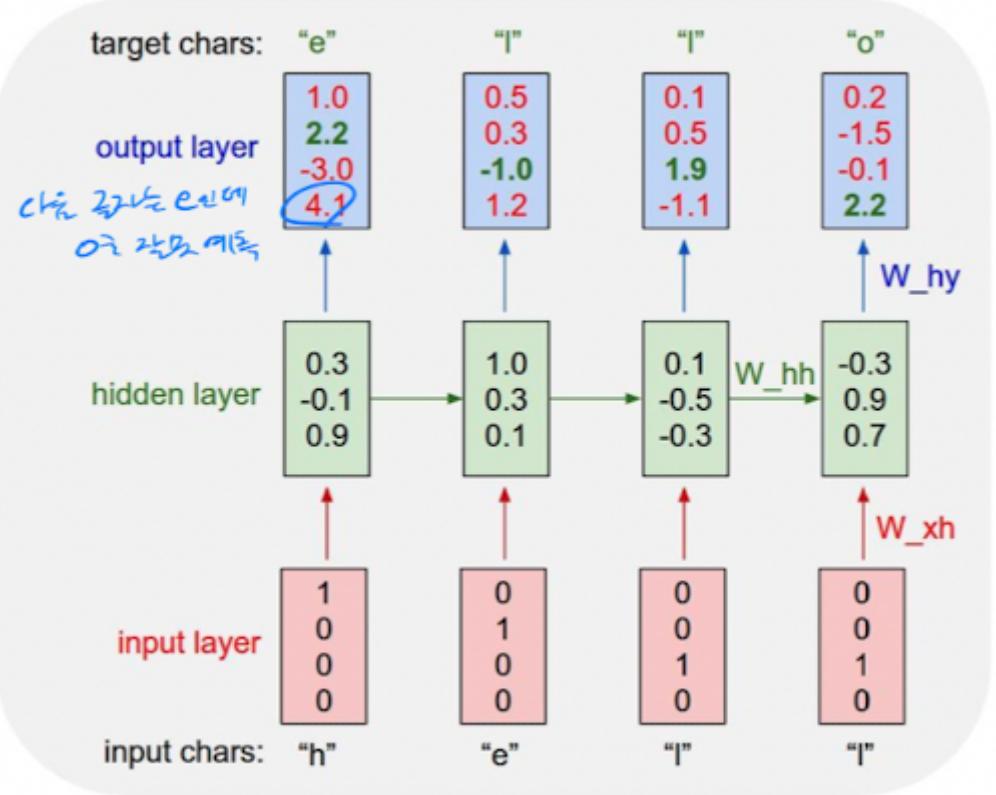
We can process a sequence of vectors \mathbf{x} by applying a **recurrence formula** at every time step:

$$h_t = f_W(h_{t-1}, x_t)$$

new state / old state input vector at
 some function \ some time step
 with parameters W



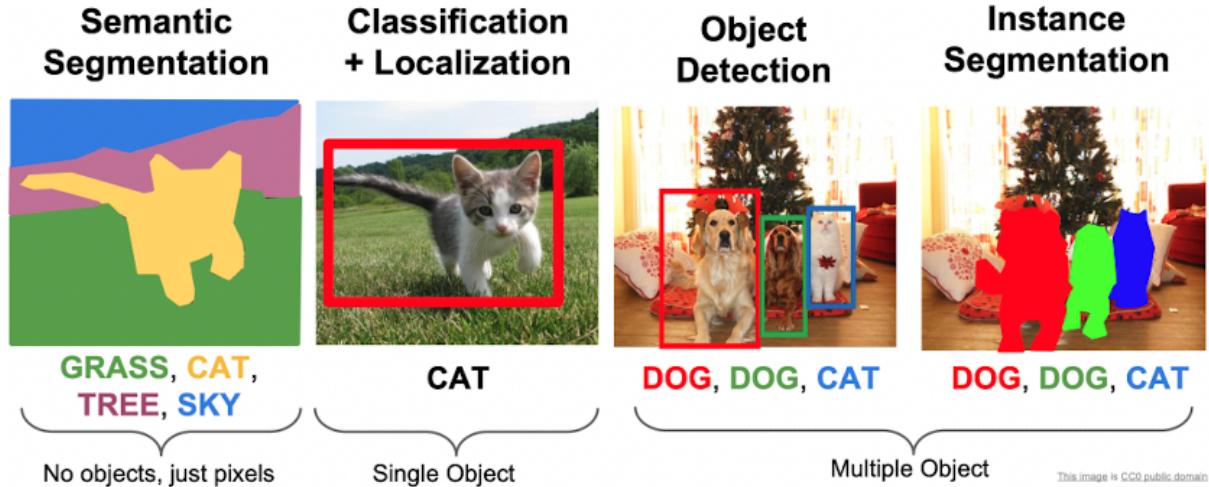
- RNN은 특정 time step t 에서 결과 y 를 예측. 여기서 f_W 는 매 스텝에서 동일하다.
- [h, e, l, o] 알파벳으로 'hell'이 입력됐을 때 'hello'가 출력되는 것을 목표로 하는 네트워크:



- 각 스텝에서의 출력이 다음 스텝의 입력이 된다.

11. Detection and Segmentation

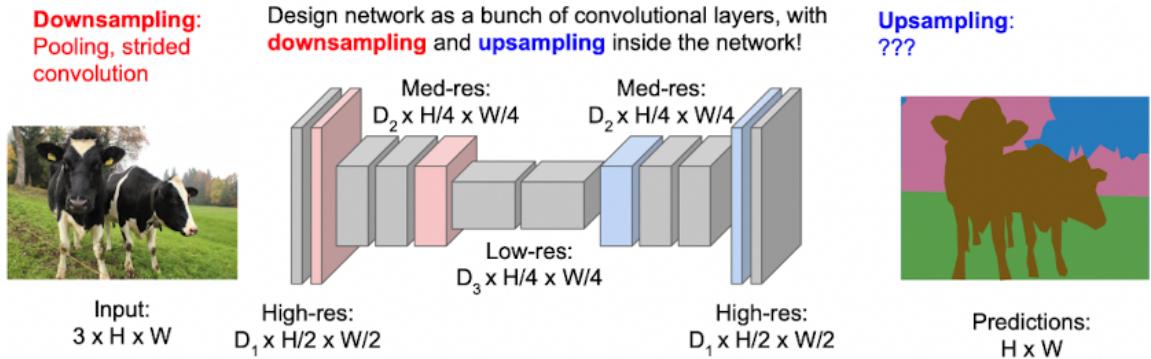
- segmentation: 이미지 내에서 영역을 분리하는 것.



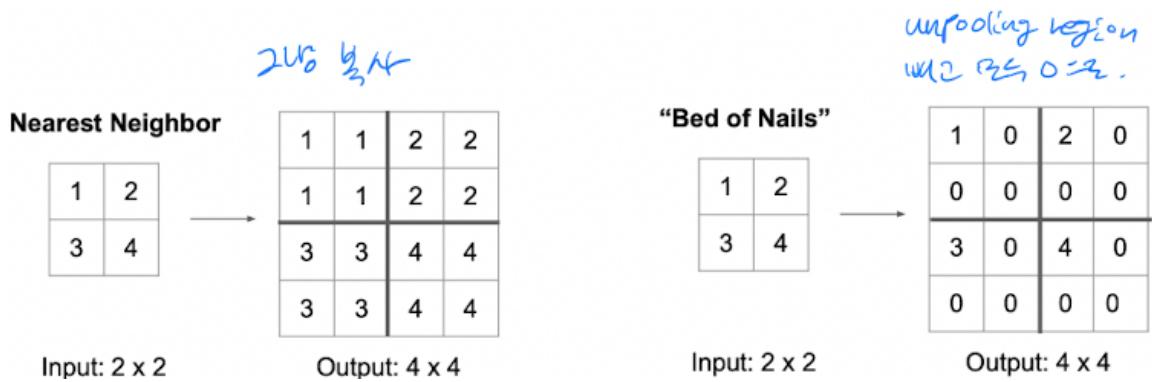
Sementic segmentation

- 입력은 이미지, 출력은 이미지의 모든 픽셀에 카테고리를 정한 맵.

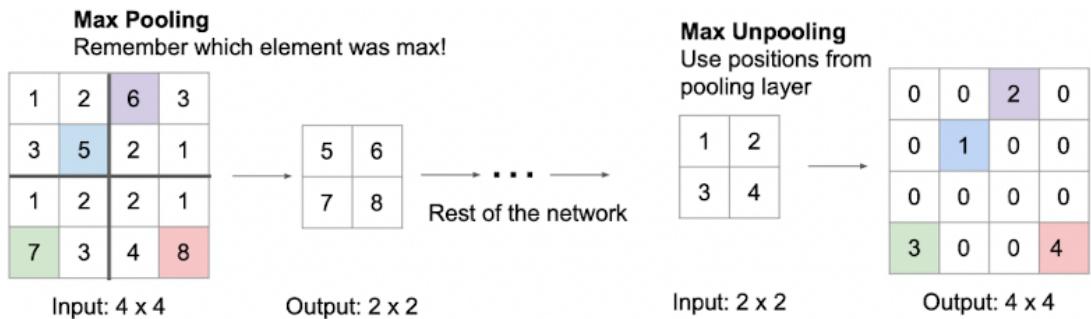
- 모든 픽셀에 대해 그 픽셀이 고양이인지, 하늘인지, 나무인지 분류하는 것.
- 개별 개체를 구분하는게 아니라 픽셀의 카테고리만 구분. 따라서 소가 두 마리라는 것은 알 수 없음.



- 특징맵을 downsampling, upsampling한다. 여기에는 FC layer가 없다.
- upsampling 전략 중 하나는 unpooling.

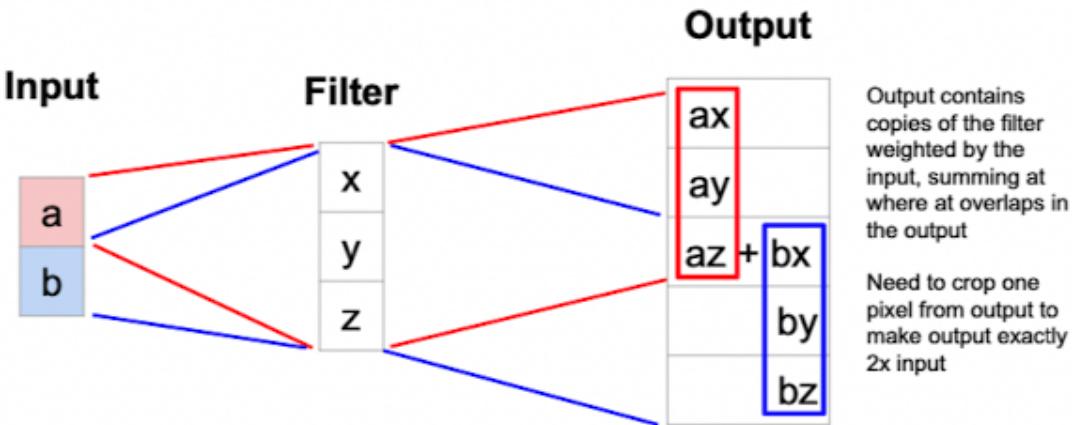


- NN: 해당하는 receptive field로 그냥 값을 복사.
- Bed of Nails: unpooling region에만 값을 복사하고, 다른 곳은 모두 0으로 채움.
- max unpooling**: max를 가져온 위치를 기억해뒀다가 다시 돌려놓음.



- **bed of nails**와의 차이: max unpooling은 각 영역의 max 위치를 기억해두고 그 위치에 복구한다. **bed of nails**는 모든 영역에서 같은 위치에 복구한다.
- Transpose Convolution: convolution을 반대로 하는 것.

Transpose Convolution: 1D Example

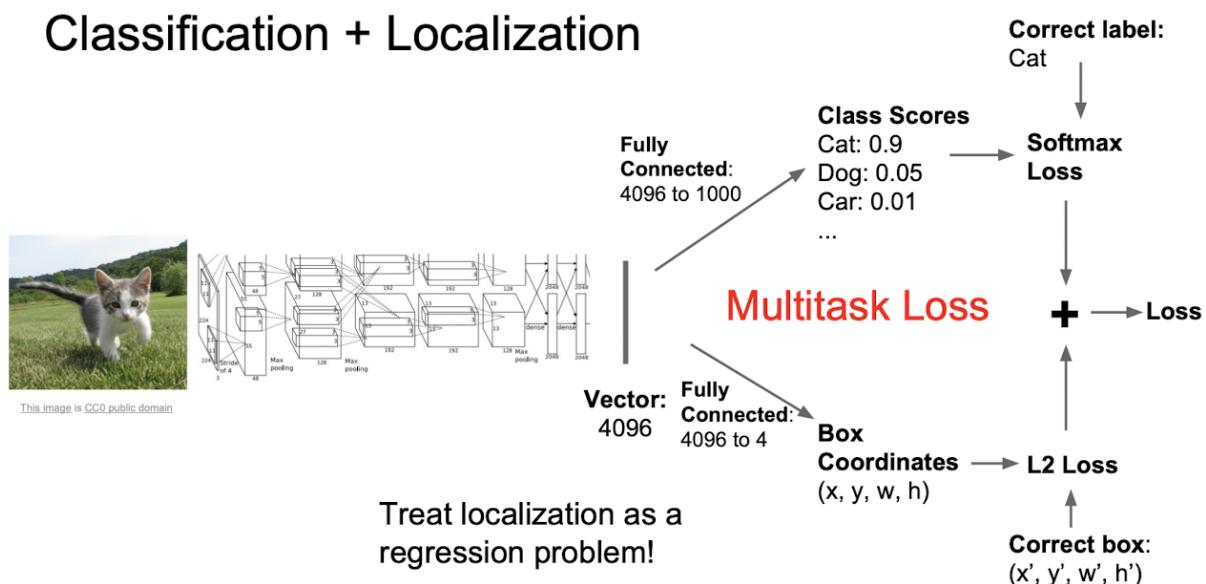


- filter가 input matrix에서 움직이는게 아니라 output matrix에서 움직임.

Classification + Localization

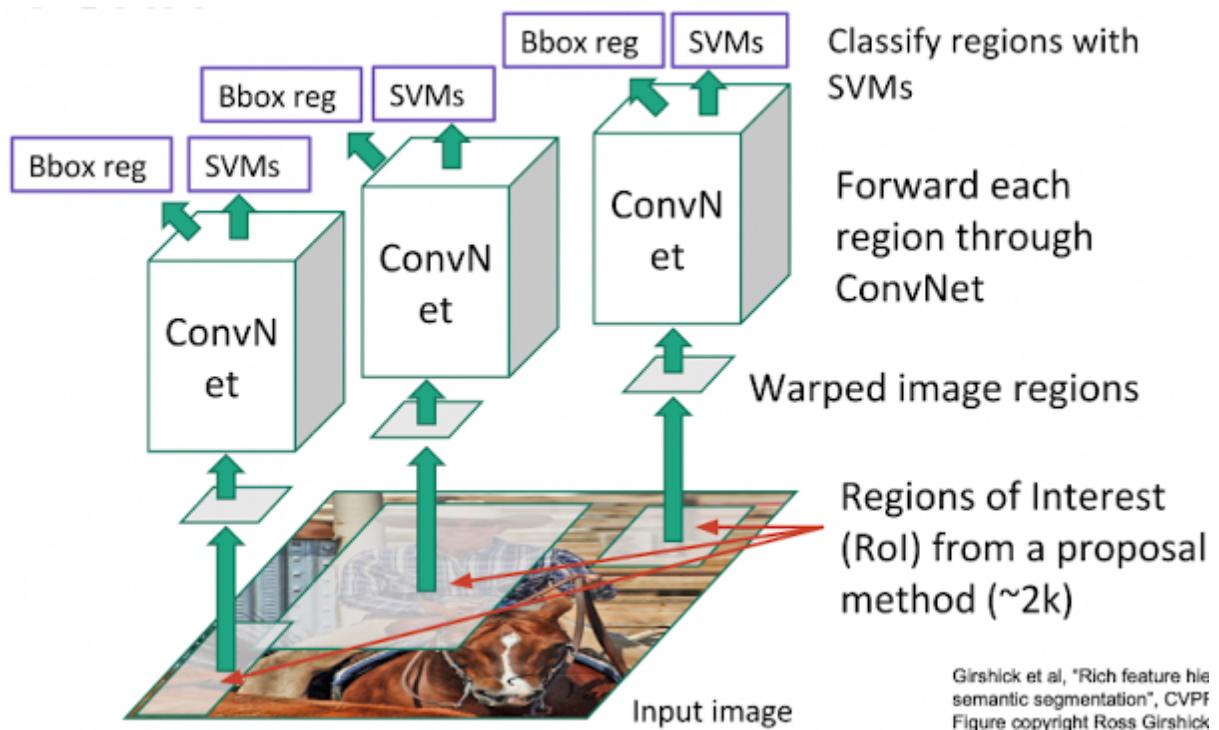
- classification은 이미지의 레이블을 매기는 것. localization은 단일 개체의 위치를 판별하는 것.
- class score를 예측하기 위한 softmax, ground truth와의 차이를 측정하는 L2가 사용.

Classification + Localization



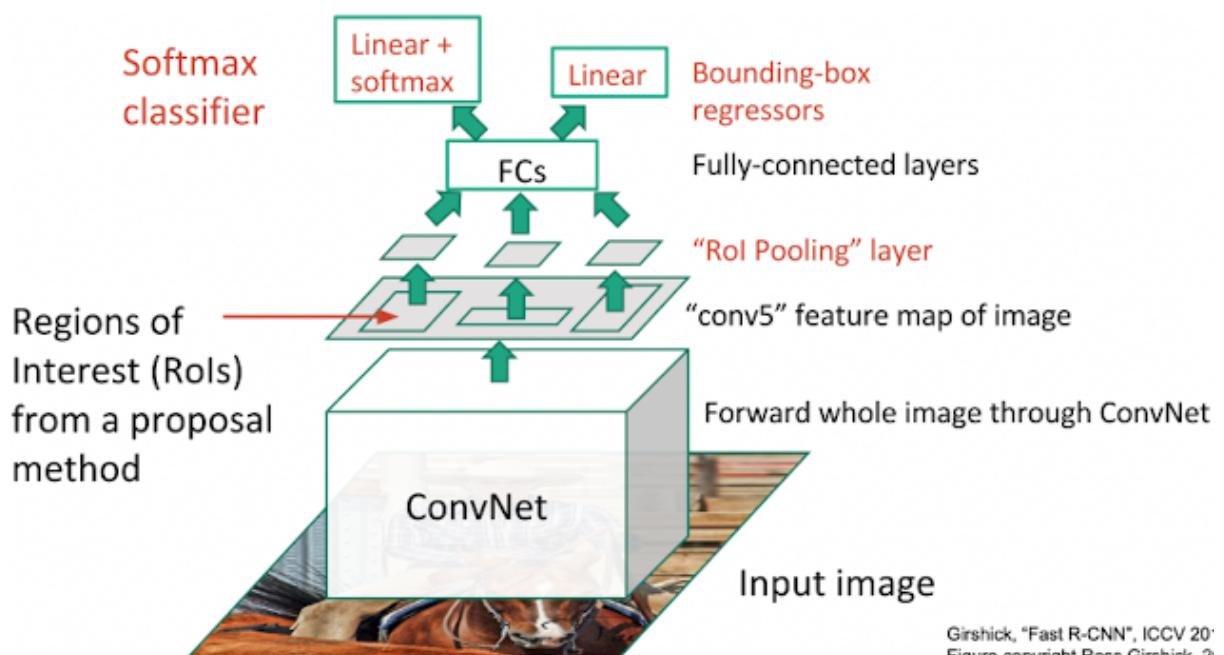
- classification과 regression의 차이점은 결과가 categorical한지, continuous한지.

R-CNN



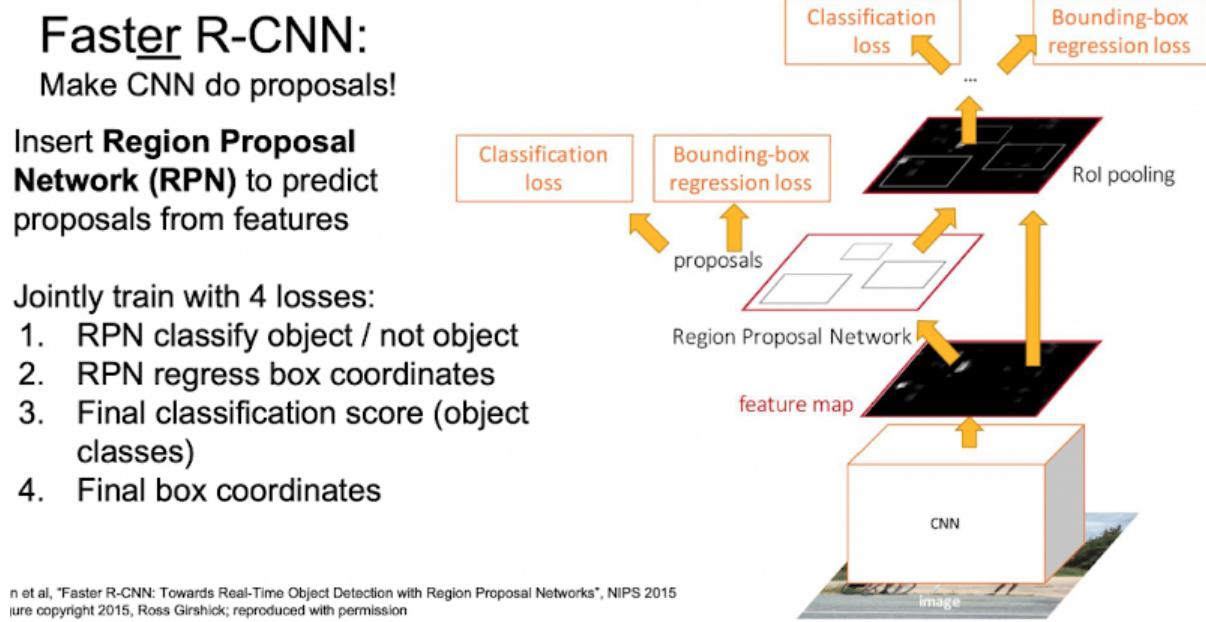
- Regions of interest(ROI) 크기가 제각각이면 문제가 되므로 region proposal로 통일하게 변환한다.
- 학습과 detection이 느린 문제.

Fast R-CNN



- FC layer가 고정된 크기의 입력을 받음. 크기를 조정해야 하므로 이때 ROI pooling layer를 사용.
- 가장 큰 개선은 feature map을 region 단위에서 pooling해 각 feature를 ROI가 공유할 수 있게 했다는 점. 중복 계산이 사라진다.
- 마지막에 softmax와 linear의 합으로 손실을 구해 역전파한다.
- region proposal을 구하는 부분이 가장 큰 병목.

Faster R-CNN



- Faster R-CNN은 region proposal을 개선해서 빠르다.
- region proposal 계산 후 CNN을 거치는 과정에서 모든 proposals가 feature map을 공유한다.

YOLO & SSD

- YOLO(You Only Look Once): convolution을 거친 feature map이 아닌 그리드를 이용한다.
- SSD(Single Shot Detection): object detection과 region proposal을 한 스테이지에 수행한다.
- region-based method인 R-CNN 계열 네트워크와 달리 YOLO와 SSD는 별도의 region proposal 과정을 거치지 않아서 빠르다.
- 하지만 정확도는 떨어진다.

Mask R-CNN

- Faster R-CNN과 비슷.

12. Visualizing and Understanding

- CNN 내부는 어떻게 생겼는가?
- 첫 레이어는 옛지와 보색을 찾아냄.
- PCA: 주성분분석. 원 데이터의 분포를 최대한 보존하면서 고차원 데이터를 저차원으로 환원하는 기법.
- Gradient Ascent:
 - 지금까지는 loss를 최소화해 모델을 개선하기 위해서 gradient descent 했음.
 - gradient ascent는 네트워크의 가중치를 모두 고정하고 중간 뉴런의 loss가 최대화 될 수 있도록 하는 픽셀을 찾는 것. 특정 필터의 response를 극대화해 무슨 일이 일어나는지 시각화하기 위한 목적.
 - $\text{argmax}_I S_c(I) - \lambda ||I||_2^2$
- feature inversion: 필터를 거쳐 나온 feature를 원상 복구해 입력 이미지를 어떻게 다루고 있는지 확인해보는 것. 주어진 feature vector와 새로운 이미지의 feature vector 사이 거리를 최소화하는 방식.

13. Generative Models

- **비지도학습**: 데이터는 있지만 레이블은 없는 상태에서 데이터의 숨겨진 구조를 학습하는게 목표.
 - 예: K-means 같은 클러스터링이 있음.

Fully visible belief network

- pixelRNN / CNN은 fully visible belief networks의 일종.
- **argmax**: y 가 최대가 되게 만드는 x 값을 구하는 함수.
- 이미지 데이터 x 가 주어졌을 때 $p(x)$ 는 x 에 대한 우도.
$$p(x) = \prod_{i=1}^n p(x_i | x_1, \dots, x_{i-1})$$
 - $p(x_i | x_1, \dots, x_{i-1})$ 은 과거의 픽셀들이 주어졌을 때 i 번째 픽셀 값의 확률.

Pixel(R|C)NN & VAE & GANs

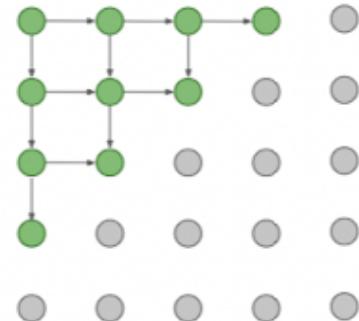
- PixelRNN and PixelCNN:

PixelRNN [van der Oord et al. 2016]

Generate image pixels starting from corner

Dependency on previous pixels modeled using an RNN (LSTM)

Drawback: sequential generation is slow!

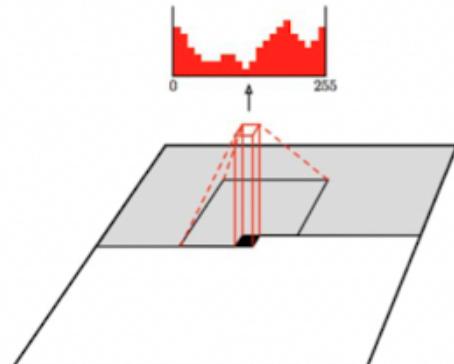


PixelCNN [van der Oord et al. 2016]

Still generate image pixels starting from corner

Dependency on previous pixels now modeled using a CNN over context region

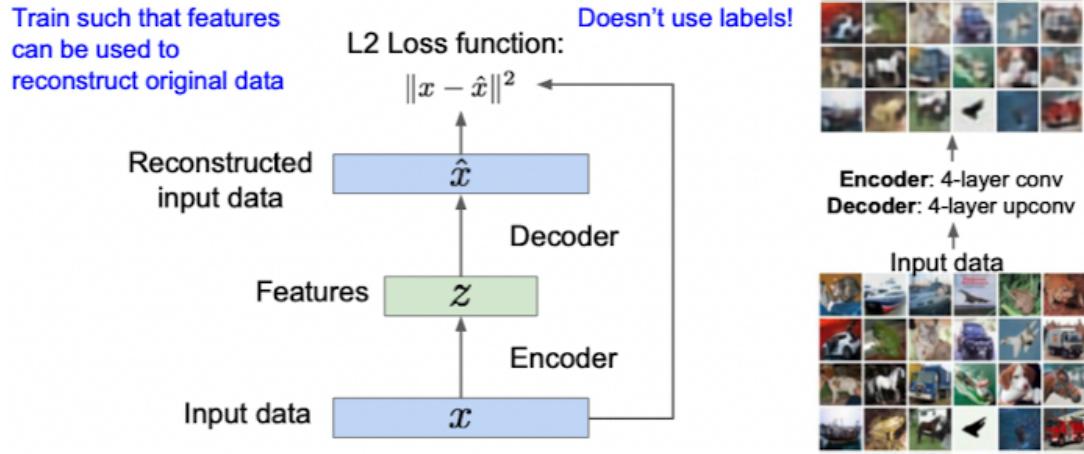
Training is faster than PixelRNN
(can parallelize convolutions since context region values known from training images)



Generation must still proceed sequentially
=> still slow

- 명시적 density model, 좋은 샘플, 특정 경향을 최적화. 하지만 비효율적인 순차 생성.
- PixelCNN은 tractable density function을 정의한다:
$$p_{\theta}(x) = \prod_{i=1}^n p_{\theta}(x_i | x_1, \dots, x_{i-1})$$
- Variational Autoencoders (VAE):
 - autoencoder는 생성 모델이 아니라 학습 데이터에 대해 낮은 차원의 feature를 학습하는 모델.

Some background first: Autoencoders



- 경향성의 variational lower bound를 최적화. 현재 샘플 품질로 최선의 선택은 아님.

Variational Autoencoders

Now equipped with our encoder and decoder networks, let's work out the (log) data likelihood:

$$\begin{aligned}
 \log p_\theta(x^{(i)}) &= \mathbf{E}_{z \sim q_\phi(z|x^{(i)})} [\log p_\theta(x^{(i)})] \quad (p_\theta(x^{(i)}) \text{ Does not depend on } z) \\
 &= \mathbf{E}_z \left[\log \frac{p_\theta(x^{(i)} | z)p_\theta(z)}{p_\theta(z | x^{(i)})} \right] \quad (\text{Bayes' Rule}) \\
 &= \mathbf{E}_z \left[\log \frac{p_\theta(x^{(i)} | z)p_\theta(z)}{p_\theta(z | x^{(i)})} \frac{q_\phi(z | x^{(i)})}{q_\phi(z | x^{(i)})} \right] \quad (\text{Multiply by constant}) \\
 &= \mathbf{E}_z [\log p_\theta(x^{(i)} | z)] - \mathbf{E}_z \left[\log \frac{q_\phi(z | x^{(i)})}{p_\theta(z)} \right] + \mathbf{E}_z \left[\log \frac{q_\phi(z | x^{(i)})}{p_\theta(z | x^{(i)})} \right] \quad (\text{Logarithms}) \\
 &= \underbrace{\mathbf{E}_z [\log p_\theta(x^{(i)} | z)]}_{\mathcal{L}(x^{(i)}, \theta, \phi)} - D_{KL}(q_\phi(z | x^{(i)}) \| p_\theta(z)) + \underbrace{D_{KL}(q_\phi(z | x^{(i)}) \| p_\theta(z | x^{(i)}))}_{\geq 0}
 \end{aligned}$$

Tractable lower bound which we can take gradient of and optimize! ($p_\theta(x|z)$ differentiable, KL term differentiable)

↳ 04-04-27(월) 2020
tractable lower bound는
모든 경우에.

- latent z와 intractable density function을 정의한다. 직접 최적화를 할 수 없음.

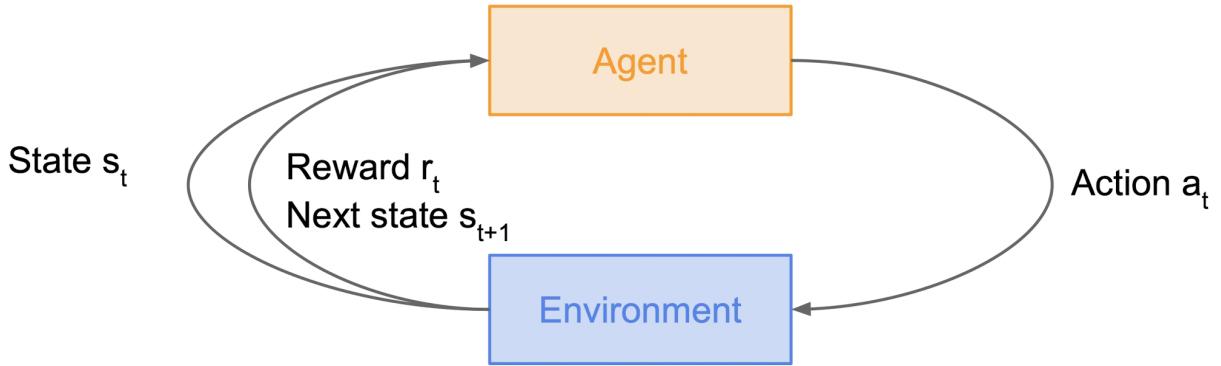
$$p_\theta(x) = \int p_\theta(z)p_\theta(x|z)dz$$

- Generative Adversarial Networks (GANs):

- 게임 이론 접근, 최선의 샘플. 안정적이지 않은 학습. 추론 쿼리 없음.
- explicit density function이 아니라 2-player 게임을 통한 학습 분포로부터 생성을 학습.

14. Reinforcement Learning

- **지도학습**: 데이터 x 와 레이블 y 를 바탕으로 x 를 y 에 매핑하는 함수를 학습하는게 목표.
 - 예: K-NN과 같은 분류, 회귀 문제. (회귀는 예측 타겟 값이 숫자인 경우)
- **강화학습**: 에이전트와 환경이 주어졌을 때 에이전트가 보상을 최대화하는 방향으로 행동을 결정하는게 목표.



Markov decision process (MDP)

- MDP는 의사결정 과정을 모델링하는 수학적 프레임워크. 강화학습을 수식화할 수 있음.
- **markov property**: 현재 상태만으로 전체 상태를 나타내는 속성.
 - \mathcal{S} : 가능한 상태의 집합.
 - \mathcal{A} : 가능한 행동의 집합.
 - \mathcal{R} : (state, action) 페어가 주어졌을 때 받는 보상의 분포. 즉, (state, action)를 보상으로 매핑하는 함수. 보상함수는 기대값.
 - \mathbb{P} : 전이 확률(transition probability). 가령 (state, action) 페어가 주어졌을 때 전이 될 다음 상태에 대한 분포. 실제 세계는 불규칙적이므로 상태 전이는 확률적임.
 - γ : 디스카운트 팩터(discount factor). 보상을 받는 시간에 대해 얼마나 중요하게 취급할 것인지 의미하는 인자. 에이전트가 현재에 가까운 보상일수록 더 큰 가치를 갖게 만든다.
- 에이전트가 종료될 때까지 아래와 같은 과정이 반복됨.
 1. 타임 스텝 $t = 0$ 에서, 환경은 초기 상태 분포인 $s_0 \sim p(s_0)$ 를 샘플링.
 2. 에이전트가 행동 a_t 를 선택.
 3. 환경은 분포 $r_t \sim R(.|s_t, a_t)$ 로부터 보상을 샘플링. 이때 보상은 상태와 행동일 주어졌을 때의 보상.
 4. 환경은 분포 $s_{t+1} \sim P(.|s_t, a_t)$ 에서 상태 s_{t+1} 을 샘플링.

5. 에이전트는 보상 r_t 와 다음 상태 s_{t+1} 을 받음.

- **policy(π)**: 각 상태에서 에이전트가 어떤 행동을 취할지에 대한 정책.
 - 결정론적일수도, 확률적일수도 있음.
 - 우리의 목표는 최적의 정책 π^* 를 찾는 것. 즉, cumulative discounted reward를 최대화하는 것.
 - **최적의 정책 π^*** : 정책 π 에 대한 미래 보상들에 대한 합의 기댓값을 최대화하는 정책.

$$\pi^* = \operatorname{argmax}_{\pi} \mathbb{E} \left[\sum_{t \geq 0} \gamma^t r_t | \pi \right]$$

- 보상 r_t 에는 미래에 얻을 보상이 포함. 이 보상은 discount factor γ 에 의해 할인됨.
- γ 가 0에 가까울수록 근시안적으로 선택하고, 1에 가까울수록 멀리보게 될 것.
- 여기서 \mathbb{E} 는 기대값이라는 의미.
 - **기대값(expected value)**: 각 사건이 일어났을 때의 이득과 그 사건이 벌어질 확률을 곱한 것을 전체 사건에 대해 합한 값. 어떤 확률적 사건에 대한 평균의 의미로 생각할 수 있음.
 - $E(X) = \sum xP(x)$
 - 주사위의 기대값은 $(1 \times \frac{1}{6}) + (2 \times \frac{1}{6}) + \dots + (6 \times \frac{1}{6}) = 3.5$
즉, 주사위를 던졌을 때 평균적으로 3.5가 나올 것을 기대할 수 있다.

Value function & Q-value function

- policy에 따라 t마다의 state, action, reward를 만들어낼 수 있음. → 일종의 ‘경로’가 됨.
- **value function**: state가 추가로 주어짐.
 - $V^\pi(s) = \mathbb{E} \left[\sum_{t \geq 0} \gamma^t r_t | s_0 = s, \pi \right]$
 - 앞서 본 것과 달리 π 외에 state s 도 주어졌음.
 - 주어진 상태 s 가 얼마나 좋은지 정의하는 함수. (π 가 주어졌을 때 누적 보상의 기대값.)
- **Q-value function**: action이 추가로 주어짐.
 - $Q^\pi(s, a) = \mathbb{E} \left[\sum_{t \geq 0} \gamma^t r_t | s_0 = s, a_0 = a, \pi \right]$

- action a 가 주어졌음.
- 주어진 상태 s 에서 어떤 액션 a 를 취하는게 좋은지 정의하는 함수.
- **Bellman equation**
 - Q^* 는 (상태, 행동) 페어에서 얻을 수 있는 누적 보상의 기대값을 최대화한다. 즉, 어떤 행동을 취했을 때 미래에 받을 보상의 최대치. 이를 통해 특정 상태에서 최상의 행동을 선택할 수 있는 최적의 정책을 구할 수 있음.

$$Q^*(s, a) = \max_{\pi} \mathbb{E} \left[\sum_{t \geq 0} \gamma^t r_t | s_0 = s, a_0 = a, \pi \right]$$

- optimal policy로부터 나온 q-value function Q^* 가 있다고 가정할 때, Q^* 는 벨만 방정식을 만족.

$$Q^*(s, a) = \mathbb{E}_{s' \sim \epsilon} [r + \gamma \max_{a'} Q^*(s', a') | s, a]$$

- 벨만 방정식은 현재 상태 가치와 다음 스텝의 상태 가치의 관계는 나타내는 방정식. s' 은 에피소드가 끝나는 시점의 상태. 따라서 $Q^*(s, a)$ 는 리워드와 에피소드가 끝나는 시점까지의 보상을 합한 값의 기대값. 여기서는 최적의 정책을 알 수 있으므로 $\max_{a'} Q^*$ 를 취함.
- 현재 페어에서 받을 수 있는 리워드와 에피소드가 종료된 s' 까지의 보상을 더한 값.
- s' 에서의 Q^* 는 현재 상태에서 할 수 있는 모든 액션 중 $Q^*(s', a')$ 를 최대화하는 것을 취함.

Solving for the optimal policy: Q-learning

- **value iteration:** 벨만 방정식을 iterative update로 사용하는 방법. 이터레이션을 반복하며 각 스텝마다 Q^* 를 조금씩 최적화한다.

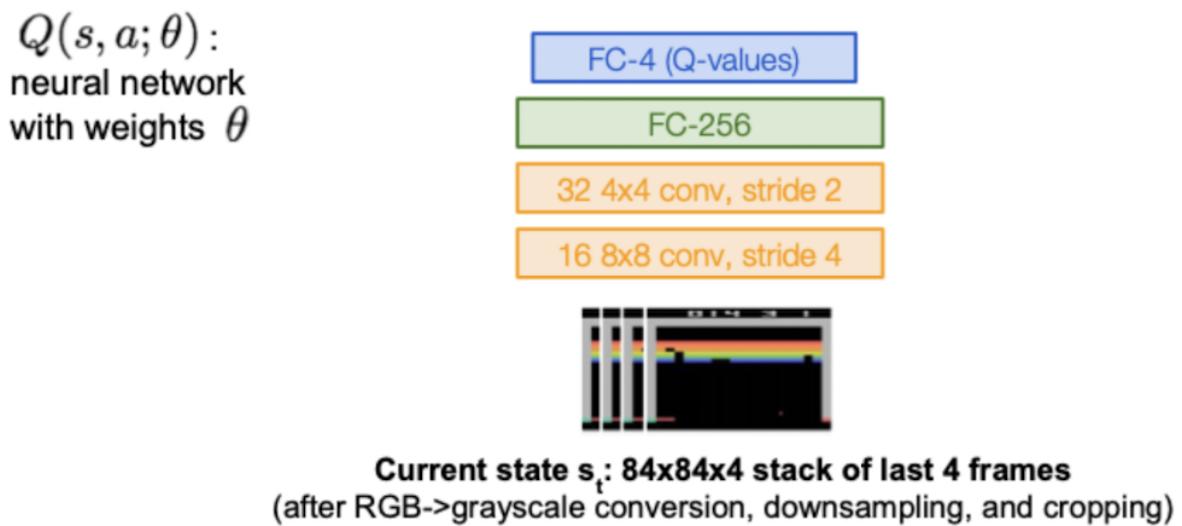
$$Q_{i+1}(s, a) = \mathbb{E}[r + \gamma \max_{a'} Q_i(s', a') | s, a]$$

- 각 스텝마다 Q_i 의 i 가 무한대로 갈 때 최적의 Q^* 로 수렴하게 된다.
- 하지만 모든 상태, 행동 페어에 대해 $Q(s)$ 를 계산해야 하기 때문에 시간이 너무 오래 걸림.
- **Q-learning:** action-value 함수를 추정하기 위한 근사 함수 Q 를 사용하는 방법.

$$Q(s, a; \theta) \sim Q^*(s, a)$$

- 이를 deep q-learning이라고 부른다.
- forward pass
 - neural network로 근사한 q-function을 학습시켜서 벨만 방정식의 에러가 최소가 되도록 해야.
- backward pass
 - forward pass에서 계산한 손실을 기반으로 파라미터 θ 를 업데이트.

Q-network Architecture



- 여기서 상태 s : 게임의 픽셀
 - 이미지가 너무 크면 곤란하니 최대한 84x84 크기로 줄임. RGB도 흑백으로 바꿈. 네트워크는 최근 4개 프레임을 본다.
- FC-256: 크기가 256인 full connected 계층. 이전 계층에서 추출된 특징을 받아서, 이를 최종적인 출력 값을 생성하는 데 사용.
- 마지막 FC 레이어(FC-4)는 4개 action(상하좌우) 중 하나를 고르게 된다.
- **Experience replay**
 - 게임을 진행하며 시간적으로 연속적인 샘플을 이용해 학습하면 모습 샘플이 상관 관계를 갖게 됨. 하지만, 샘플에 상관관계가 있으면 학습이 비효율적.
 - Q-network 파라미터를 생각하면, 정책을 결정함에 따라 다음 샘플도 결정하게 됨.
 - 현재 상태에서 '왼쪽'이 보상을 최대화하는 액션이라면 결국 다음 샘플도 전부 '왼쪽'에서 발생할 수 있는 것들로만 편향됨.

- experience replay는 이 두 문제를 해결.
- replay memory에 (상태, 액션, 보상, 다음상태)로 구성된 전이 테이블을 만들어두고, 게임을 진행함에 따라 전이 테이블을 지속적으로 업데이트.
- 연속적인 샘플을 사용하는 대신 전이 테이블에서 임의로 샘플링된 샘플을 사용하도록 한다.
- 챌린지는 exploration

Policy Gradients

- 정책 자체를 학습시켜서 여러 정책 중 최고의 정책을 찾아낼 수 있다.

$$J(\theta) = \mathbb{E} \left[\sum_{t \geq 0} \gamma^t r_t | \pi_\theta \right]$$

- 따라서 우리가 찾고자하는 optimal policy $\theta^* = \operatorname{argmax}_\theta J(\theta)$
- 모든 상태에 대해서 전이확률과 정책 π 로부터 얻은 행동에 대한 확률을 모두 곱하면 경로에 대한 확률을 얻을 수 있음:

$$p(\tau; \theta) = \prod_{t \geq 0} p(s_{t+1} | s_t, a_t) \pi_\theta(a_t | s_t)$$
- gradient를 계산하기 위해 모든 리워드에 대한 모든 기대값을 합한다.

$$\nabla_\theta J(\theta) = \int_\tau r(\tau) \nabla_\theta p(\tau; \theta) d\tau$$

- 특정 policy θ 가 주어졌을 때 목적 함수 J (찾고자 하는 최적의 정책)를 trajectory에 대한 reward의 총합으로 표현할 수 있음. 이때 tau가 trajectory. 이를 편미분하여 gradient ascent를 하는 것.
- 위 식은 p가 theta에 의존하기 때문에 interactable하다. 아래와 같이 로그를 취해 준다.

$$\nabla_\theta J(\theta) = \int_\tau (r(\tau) \nabla_\theta \log p(\tau; \theta)) p(\tau; \theta) d\tau$$

- 인테그랄때문에 intractable하다. 로그를 씌우면 파이를 시그마로 바꿀 수 있다. 따라서:

$$\log p(\tau; \theta) = \sum_{t \geq 0} \log p(s_{t+1} | s_t, a_t) + \log \pi_\theta(a_t | s_t)$$
- 이때 Gradient estimator는:

$$\nabla_{\theta} J(\theta) \sim \sum_{t \geq 0} r(\tau) \nabla_{\theta} \log \pi_{\theta}(a_t | s_t)$$

- policy gradients의 문제는 분산이 커진다는 것
 - **baseline**: 경로에서 계산한 값들 모두가 의미있는 건 아님. 정말 중요한 것은 얻은 보상이 예상보다 좋은지, 아닌지를 판단하는 것. 이때 baseline function을 사용할 수 있음.
 - q-function과 value-function을 이용해서 baseline을 정할 수 있음:

$$Q^{\pi}(s_t, a_t) - V^{\pi}(s_t)$$
- 챌린지는 sample efficiency.

15. Adversarial Examples & Training

- 이미지에 사람 눈에 보이지 않는 노이즈 레이어를 덮어 인공지능 모델을 기만할 수 있음.
- 이러한 공격법이 **FGSM**(Fast Gradient Sign Method):
 - 모델이 선형적이라면 공격이 쉽다.
- 적대적 훈련: 애초에 모델을 트레이닝할 때 적대적인 예시에 대해서도 트레이닝해야 한다.

Ensemble Techniques

- 한 데이터를 여러 개로 쪼개서 분석하는 기법.

Imbalanced Class Problem

Confusion matrix

- 불균형에 대한 척도. 분류 성능에 대한 평가 지표인 셈.

		PREDICTED CLASS	
		Class=Yes	Class>No
ACTUAL CLASS	Class=Yes	a	b
	Class>No	c	d

		PREDICTED CLASS	
ACTUAL CLASS	Yes	No	
	Yes	TP	FN
	No	FP	TN

α is the probability that we reject the null hypothesis when it is true. This is a Type I error or a false positive (FP).

β is the probability that we accept the null hypothesis when it is false. This is a Type II error or a false negative (FN).

$$\text{Accuracy} = \frac{TP + TN}{TP + FN + FP + TN}$$

$$\text{ErrorRate} = 1 - \text{accuracy}$$

$$\text{Precision} = \text{Positive Predictive Value} = \frac{TP}{TP + FP}$$

$$\text{Recall} = \text{Sensitivity} = \text{TP Rate} = \frac{TP}{TP + FN}$$

$$\text{Specificity} = \text{TN Rate} = \frac{TN}{TN + FP}$$

$$\text{FP Rate} = \alpha = \frac{FP}{TN + FP} = 1 - \text{specificity}$$

$$\text{FN Rate} = \beta = \frac{FN}{FN + TP} = 1 - \text{sensitivity}$$

$$\text{Power} = \text{sensitivity} = 1 - \beta$$

- confusion matrix를 그려볼 수 있음.
- a는 true positive, b는 false negative, c는 false positive, d는 true negative.
- 따라서 정확도 $\text{accuracy} = \frac{a+d}{a+b+c+d}$. 전체 중 실제와 동일하게 분류한 것.
- 에러율 error rate = $1 - \text{accuracy}$
- 정밀도 precision(p) = $\frac{a}{a+c}$. true라고 분류한 것 중 실제로 true인 비율.
- 재현율 recall(r) = $\frac{a}{a+b}$. 실제로 true인 것 중 true라고 분류한 비율. sensitivity, TP Rate, hit rate라고도 한다.
- F-measure(F) = $\frac{2a}{2a+b+c}$.
- TN Rate = $\frac{d}{c+d}$. specificity라고도 한다.
- FP Rate = $1 - \text{specificity}$. 알파라고도 한다.
- FN Rate = $1 - \text{sensitivity}$. 베타라고도 한다.

		PREDICTED CLASS	
ACTUAL CLASS		Class=Yes	Class>No
	Class=Yes	10	0
	Class>No	10	980

$$\text{Precision (p)} = \frac{10}{10+10} = 0.5$$

$$\text{Recall (r)} = \frac{10}{10+0} = 1$$

$$\text{F - measure (F)} = \frac{2 * 1 * 0.5}{1 + 0.5} = 0.62$$

$$\text{Accuracy} = \frac{990}{1000} = 0.99$$

- 기계가 모든 걸 NO로 분류해도 정확도가 0이 나오지는 않음. 잘못된 판단으로 이어질 수도.

Cost matrix

- 예측에 따른 비용을 지표로 표현한다.

		PREDICTED CLASS		
		C(i,j)	+	-
ACTUAL CLASS	Model M ₁	+1	-1	100
	Model M ₂		+	-
	ACTUAL CLASS	+2	250	45

Accuracy = 80%
Cost = 3910

Accuracy = 90%
Cost = 4255

		PREDICTED CLASS	
ACTUAL CLASS		Class=Yes	Class>No
	Class=Yes	f(Yes, Yes)	f(Yes, No)
	Class>No	f(No, Yes)	f(No, No)

		PREDICTED CLASS		
ACTUAL CLASS	Cost Matrix	C(i, j)	Class=Yes	Class>No
	Class=Yes	C(Yes, Yes)	C(Yes, No)	
	Class>No	C(No, Yes)	C(No, No)	

- 비용은 $\sum C(i, j) \times f(i, j)$.

Association Analysis Basic Concepts

- association rule: 우유와 기저귀를 사면 → 맥주를 산다.
- support count(σ):
아이템셋의 출현 빈도.
- support(s):
해당 아이템셋을 가진 트랜잭션의 비율.

- confidence(c):
X를 가진 트랜잭션에서의 Y에 대한 비율.
- $\forall X, Y : (X \subseteq Y) \Rightarrow s(X) \geq s(Y)$
- 최소한의 서포트를 **minsup**이라고 한다.
minsup이 0.5라면 50% 미만의 아이템은 유의미하지 않은 것으로 간주하고 소거할 수 있다.

TID	Items
1	Bread, Milk
2	Bread, Diaper, Beer, Eggs
3	Milk, Diaper, Beer, Coke
4	Bread, Milk, Diaper, Beer
5	Bread, Milk, Diaper, Coke

Example:

$$\{Milk, Diaper\} \Rightarrow \{Beer\}$$

$$s = \frac{\sigma(Milk, Diaper, Beer)}{|T|} = \frac{2}{5} = 0.4$$

$$c = \frac{\sigma(Milk, Diaper, Beer)}{\sigma(Milk, Diaper)} = \frac{2}{3} = 0.67$$

- 차를 샀을 때 커피도 산 비율.

Contingency table

	Y	\bar{Y}	
X	f_{11}	f_{10}	f_{1+}
\bar{X}	f_{01}	f_{00}	f_{0+}
	f_{+1}	f_{+0}	N

f_{11} : support of X and Y
 f_{10} : support of X and \bar{Y}
 f_{01} : support of \bar{X} and Y
 f_{00} : support of \bar{X} and \bar{Y}

Customers	Tea	Coffee	...
C1	0	1	...
C2	1	0	...
C3	1	1	...
C4	1	0	...
...			

	Coffee	\bar{Coffee}	
Tea	15	5	20
\bar{Tea}	75	5	80
	90	10	100

Association Rule: Tea \rightarrow Coffee

$$\text{Confidence} \approx P(\text{Coffee}|\text{Tea}) = 15/20 = 0.75$$

- 그런데 이때 $P(\text{Coffee})$ 는 0.9임. 티를 샀을 때 커피사는 것보다는 그냥 커피만 마실 활률이 큰 거임. 사실 차를 구입하는게 커피 구입을 방해하는거임.

Note that $P(\text{Coffee}|\overline{\text{Tea}}) = 75/80 = 0.9375$

Cluster Analysis

- 비슷한 것들끼리 같은 군집으로 묶는 것.

K-Means

- 초기에는 랜덤하게 정해진 centroid를 설정하고, 개별 데이터와 각 centroid 사이의 거리를 계산해 가장 가까운 centroid로 클러스터링. 새로운 데이터가 추가되면 군집에 속한 데이터들의 평균을 바탕으로 centroid가 다시 계산된다.
- K-NN과는 다르다!
- Sum of Squared Error (SSE):
 - 거리 편차를 이용해 오차를 판단.
 - 클러스터 버전의 L2 distance (euclidean distance).
- K-means에서 발생할 수 있는 문제들:
 - 크기, 밀도, 원형이 아닌 군집.
 - outlier 하나가 SSE를 증가시킬 수 있음.

Hierarchical Clustering

DBSCAN

- K-means는 군집이 원형이 아닌 경우 문제가 생김. 군집의 개수도 사용자가 지정해야.
- DBSCAN은 데이터의 밀도를 이용해 군집이 불규칙해도 잘 동작.
- epsilon: 이웃을 정의하기 위한 거리.

$$Q^*(s, a) = E_{s \sim \epsilon}[r + \gamma \max_{a'} Q^*(s', a') | s, a]$$

$$\nabla_{\theta} J(\theta) = \int_{\tau} (r(\tau) \nabla_{\theta} \log p(\tau; \theta)) p(\tau; \theta) d\tau$$



기말고사는 100문제. 학기 초에 했던 introduction to data mining은 물론, 스텐포드 챕터 3부터 챕터 16까지. 챕터 15는 제외. 스텐포드 강의 비중이 더 높으니까 참고.

OX문제: 말그대로 OX.

다지선다: 10개의 단어가 있고 10개의 선택지가 있다고 봅시다. 이 단어는 7번째 설명에 대한거야, 하면 7이라고 쓰시면 됩니다.

단답형: 설명을 쭉 하고나서 뭐에 대한 설명이냐, 하면 q-learning이다 쓰면 됩니다.

주관식: 설명을 하고 계산을 해야 할 수도 있어요. CNN 같은 거 보면 stride 계산을 시킬 수 있어요. 아니면 뭐뭐에 대해 설명하시오 같은 식으로 낼 수도 있어요.

스탠포드 강의에서 챕터 1, 2에서 직접적으로 문제를 내지는 않았어요. 하지만 뒤에 있는 내용에 녹아 있으니까 공부하세요. 마찬가지로 introduction to data mining도 같아요.



챕터 11부터 15에서 많이 나옴. 지금 수학 계산 중에서 여러분이 이해 못하는 부분은, 이해가 안 가시면 산수 문제 중 이해가 되는 거라도 공부해주세요.

precision, recall, accuracy, optimization 계산, zero padding 계산 파악해주세요. 근데 이게 이해가 안 가면 용어에 대해서 이해해 보세요. 제일 어려운 문제는 뭐뭐에 대해 설명하라는 거예요.

챕터 15가 강화학습이죠. 거기서 낼 두 문제를 알려 드릴게요.

- **policy에 대한, policy gradient에 대한 수식을 쓰고 설명하라.**

$$\rightarrow \nabla_{\theta} J(\theta) = \int_{\tau} (r(\tau) \nabla_{\theta} \log p(\tau; \theta)) p(\tau; \theta) d\tau$$

최적의 정책 $J(\theta)$ 를 trajectory tau에 대한 cumulative reward로 표현할 수 있음. 여기서 J 를 편미분하여 gradient ascent를 하는 것이다. 다만 그렇게만 하면 p 가 θ 에 의존하기 때문에 \log 를 취해준다.

- **벨만 방정식을 쓰고 설명하라.**

$$\rightarrow Q^*(s, a) = \mathbb{E}_{s' \sim \epsilon}[r + \gamma \max_{a'} Q^*(s', a') | s, a]$$

벨만 방정식은 현재 상태 가치와 다음 스텝의 상태 가치의 관계를 나타내는 방정식. s' 은 에피소드가 끝나는 시점의 상태이며, 따라서 $Q^*(s, a)$ 는 리워드와 에피소드가 끝나는 시점까지의 보상을 합한 값의 기대값. 이때 최적의 상태, 행동 페어를 알고 있으므로 $\max Q^*$ 를 취하는 것이다.

- **decision tree와 rule-based의 차이.**

\rightarrow decision tree는 데이터의 패턴을 파악해 이진 트리의 조건을 따라 데이터를 분류한다. 반면, rule-based는 정의해둔 조건을 바탕으로 if-then-else 규칙을 따라 데이터를 분류한다.

- **regression의 개념에 대해서도 물어볼거예요.**

\rightarrow regression은 데이터에 대해 변수의 대표값을 구하여 상관성이나 적합성을 분석하는 것. classification이 categorical한 데이터를 출력함으로써 입력 데이터를 분류하는데 쓰인다면, regression은 continuous한 데이터를 다룰 때 쓰인다.