

과제명	머신러닝 기말고사 과제
-----	--------------



- 과 목 명 머신러닝
- 담당교수 김 용 운
- 학부(과) 컴퓨터소프트웨어공학과
- 학 년 4
- 분반번호 01
- 학 번 20173147
- 이 름 이 명 진
- 연 락 처 010-2999-7748
- 제 출 일 2020 년 07 월 03 일



원광대학교
WONKWANG UNIVERSITY

01-01 머신러닝 수업의 개요와 일정



세기의 대국이라고 불렸던 이세돌 9단과 구글의 인공지능 알파고와의 대국.

아쉽게도 이세돌 선수가 4대 1로 패배. 알파고는 총 전적 68승 1패로 이세돌 9단에게 유일하게 패배를 허용하고 은퇴를 하였음.

그 동안 오직 인간만이 할 수 있는 의사결정을 이제 컴퓨터도 할 수 있다라는 개념을 사람들에게 전파했던 경기.

머신러닝의 대가 중 한 명 앤드류 윤

“이런 시대에 살면서 머신러닝을 잘 이해하고 활용하는 것이 바로 **슈퍼파워**를 갖는 길이다.”

복잡한 망 데이터가 있을 때 머신러닝을 활용하여 잘 의사결정을 내린다면 이런 사람들이 **슈퍼파워**를 가지는 것이고 다른 이들보다 앞서나가는 사람이 된다.

강의 목표: 머신러닝 알고리즘을 이해하고 딥러닝을 이해한다.

Machine learning 알고리즘의 기본적인 이해

Neural networks, Convolutional Neural networks, Recurrent Neural networks
DNN, CNN, RNN을 대략적으로 이해하고 실습

LINEAR REGRESSION과 LOGISTIC REGRESSION만 제대로 이해하고 있더라도 머신러닝을 어느 정도 이해했다고 봐도 된다. 어떤 문제를 해결해야할 때의 사용도구는 다음과 같다.



=====

1- 02. 기본적인 Machine Learning의 용어와 개념

>> Machine Learning(ML)은 무엇인가?

소프트웨어 즉 ,프로그램: 입력을 기반으로 어떠한 데이터를 처리해서 보여줌

= explicit programming

> spam filter(걸러주는 것) --> many rules

> Automatic driving(자동 운전) --> too many rules

=> 이러한 문제점들을 해결하기 위해 꾸준한 연구가 이어졌음.

1959년 Artur Samuel

- 일일이 프로그래밍 X 자료, 현상에서 자동적으로 시스템이 배우게하는 것

- 프로그램 자체가 학습을 하는 것

Supervised learning / Unsupervised learning

> Supervised(감독관) learning

- 데이터(training set)

- ex) 고양이, 개 라는 레이블들이 달려있는 자료를 가지고 학습

=> 초반에 이해하기 쉽. 초반에는 supervised learning 위주로 설명할 예정

> Unsupervised learning

- Google news grouping, Word clustering와 같이

=> 레이블을 사람이 일일이 만들기 어려움. 아주 많은 데이터셋들이 필요

Training data set

주어지는 값 -> X: (3,6,9), (2,5,6)

보통 결과값 -> Y: (3),(2)

Xtest=[9,3,6] --> Y= 3 이라는 값 도출

Supervised learning의 종류

- 시험 성적 예상 (0~100점) ==> Regression
- Pass/Non-pass (둘 중에 하나 고르는 것) ==> Binary classification
- A,B,C,D & F based on time spent ==> Multi-label classification

Predicting exam score: Regression

- 무엇인가 학습된 이후에 x값에 7을 넣었다

x (hours)	y (score)
10	90
9	80
3	50
2	30

==> y 값은 약 72점 정도 나온다고 예상가능

Pass/non-pass based on time spent : Binary classification

x (hours)	y (pass/fail)
10	P
9	P
3	F
2	F

==>공부를 9.5시간 정도했으면 p라고 예측가능

Letter grade (A, B, C, E and F) : multi-label classification

==> 예측가능

x (hours)	y (grade)
10	A
9	B
3	D
2	F

=====

1- 03. TensorFlow의 기본적인 설명과 실습환경 구축

=> google에서 만든 open source library for Machine Intelligence

- tensorflow가 github에 올려져있는 라이브러리 중 단연 1등.
- 많은 사람들이 사용

TensorFlow

✓ TensorFlow™ is an open source software library for **numerical computation** using **data flow graphs**. (데이터 플로우 그래프등을 이용하여 수치계산을 하는 라이브러리)

✓ Python을 가지고 tensorflow 프로그래밍을 할 수 있음.

Data Flow Graph란?

- ✓ Nodes in the graph represent mathematical operations
- ✓ Edges represent the multidimensional data arrays (tensors) communicated between them.

=> 노드와 노드들을 구성하는 엣지로 구성되어있는 것 = graph

dataflow는 이러한 노드들과 엣지(data arrays or tensors)들이 연결이 되어있고 작업할 수 있는 것

우리의 실습 환경은 윈도우인데...

=> Anaconda를 이용하여 tensorflow 이용 가능.

이후 계속하여 매뉴얼에 따라 실습 환경을 구축 해 본다.

- Anaconda를 이용한 TensorFlow 설치

1. Anaconda prompt 실행 후 TensorFlow 구동을 위한 가상 환경 생성

» Python 3.6 버전 지정 필수

```
conda create -n 환경이름 pip python=3.6
```

2. 추가적으로 설치하게 될 패키지 목록들이 뜨고 “y”눌러 진행

3. 생성 완료 후 가상 환경 실행

» 가상 환경이 실행되면 (base) C:\~ 가 (환경이름) C:\~로 변경된다.

```
conda activate 환경이름
```

• Anaconda를 이용한 TensorFlow 설치

4. TensorFlow를 설치하기 위해 다음의 코드를 차례대로 실행

```
pip install --upgrade pip  
pip install --upgrade tensorflow-cpu==1.15
```

5. 설치가 끝나면 제대로 설치되어 있는지 확인 하기 위해 아래의 코드를 입력

```
python -c "import tensorflow as tf; print(tf.reduce_sum(tf.random.normal([1000, 1000])))"
```

6. 설치가 정상적으로 완료 되었다면 아래와 같은 결과가 화면에 나타난다.

```
tf.Tensor(숫자, shape=(), dtype=float32)
```

```
(TensorFlow) C:\Users\김YongWun>python -c "import tensorflow as tf; print(tf.reduce_sum(tf.random.normal([1000, 1000])))"  
Tensor("Sum:0", shape=(), dtype=float32)
```

이렇게 뜨면 설치 성공!

• Jupyter Notebook 설치 및 실행

1. Anaconda prompt에서 Jupyter Notebook을 설치한다.

» 꼭 가상 환경 상태에서 설치 해야 한다.

```
conda install jupyter notebook
```

2. 추가적으로 설치하게 될 패키지 목록들이 뜨고 “y”눌러 진행

3. 생성 완료 후 Jupyter Notebook을 실행 한다.

```
Jupyter notebook
```

• Jupyter Notebook 기본 경로 설정

4. "jupyter_notebook_config.py"에서 #c.NotebookApp.notebook_dir = "를 찾아 해당 문장을 c.NotebookApp.notebook_dir = 'C:\MachineLearning'로 수정 한다.

• Jupyter Notebook 기본 경로 설정

1. 기본 경로를 설정 하기 위해서는 "jupyter_notebook_config.py"파일의 위치를 알아야 함
그에 따른 경로는 아래와 같음

```
jupyter notebook --generate-config
```

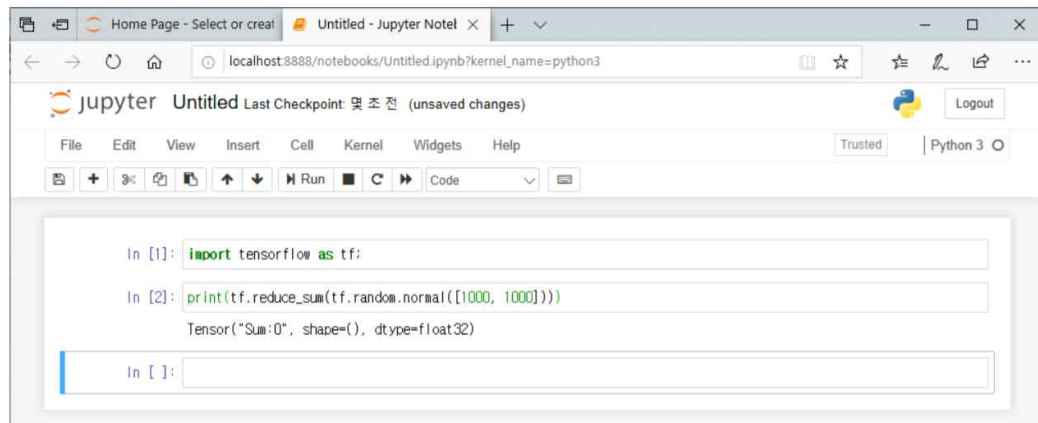
2. 명령어를 입력하면, 아래와 같이 경로가 나타나게 됨

```
(TensorFlow) C:\Users\KimYongWun>jupyter notebook --generate-config
Writing default config to: C:\Users\KimYongWun\jupyter\jupyter_notebook_config.py
(TensorFlow) C:\Users\KimYongWun>
```

3. 실습 환경 폴더를 새로 만든다.

» Ex) C드라이브에 "MachineLearning"폴더를 새로 생성한다.

• Jupyter Notebook에서 TensorFlow 코드 실행



02-01 Linear Regression_1(선형회귀)

01. 개요

- X변수(원인)와 Y변수(결과) 사이의 관계

✓확정적 관계와 확률적 관계가 있음

- 확정적 관계 : X변수만으로 Y를 100%로 표현 (오차항이 없음) $Y = f(X)$

-> 값이 100% 정해져있음

✓예) 힘 = $f(\text{질량, 가속도})$, 주행거리 = $f(\text{속도, 시간})$

- 확률적 관계 : X변수와 오차항이 Y를 표현 (오차항이 있음) $Y = f(X) + \epsilon$

- 예시 3가지)

반도체 수율 = $f(\text{온도, 습도, 설비 파라미터들의 상태}) + \epsilon$

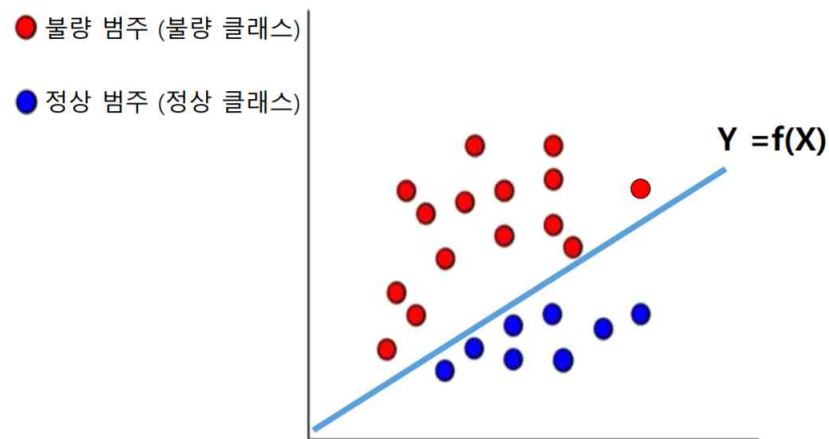
포도주 가격 = $f(\text{강우량, 온도, 포도품종}) + \epsilon$

위조 카드 여부 = $f(\text{사용 장소, 사용 패턴, 사용 시간}) + \epsilon$

- 수치 예측 VS 범주 예측

X , Y -----> 연속성: 수치 예측
 -----> 범주형: 범주 예측

- 수치 예측(Regression) VS 범주 예측(classification)



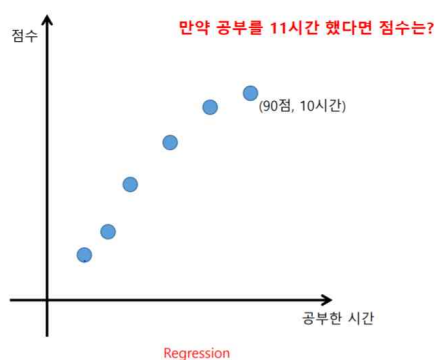
수치 예측: 함수 식을 찾고 함수식을 이용해서 Y값이 없는 X값이 주어졌을 때 Y값을 예측하는 것

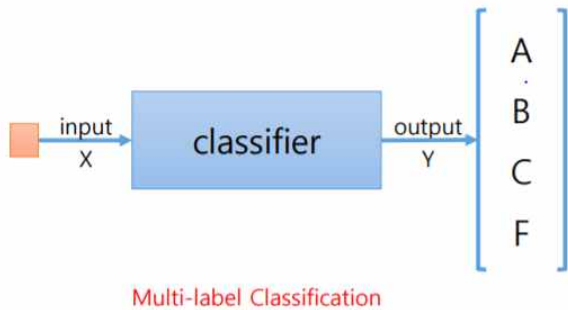
범주 예측: 어떠한 사이를 가장 최대화 할 수 있는 함수식을 찾고 새로운 데이터가 어떠한 범주 중에 속하는 지 예측 실제로 어떠한 수치를 예측하는 것이 아님

- Classification VS Regression

classification에는 classifier가 존재
 (classifier에 트레이닝을 시킨다)

Regression: 공부한 시간을 통해 성적을 예측하라!





만약 공부를 11시간 했다면 점수는?

• Regression(우리말= 후퇴, 퇴보?)

✓“Regression toward the mean(전체의 평균을 의미)”

Regression이란?

Sir Francis Galton (1822 ~ 1911)

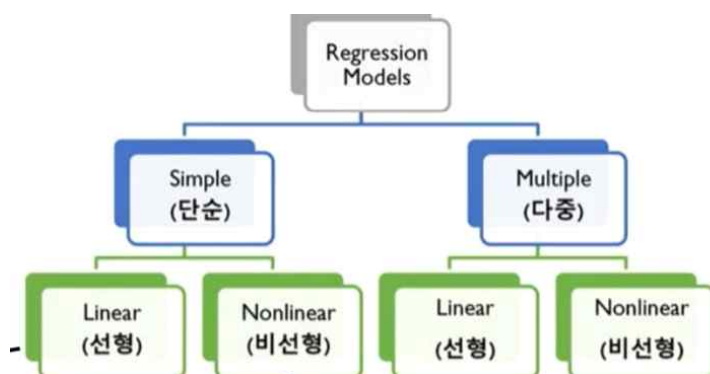
- 데이터들은 전체의 평균으로 회귀하려는 특징이 있다 ==> Regression의 의미

• Regression Models 분류(크게 4가지로 나눌 수 있음)

✓X변수의 수

✓X변수와 Y변수의

선형성 여부에 따라 구분



Simple :

선형: 직선, 비선형: 곡선

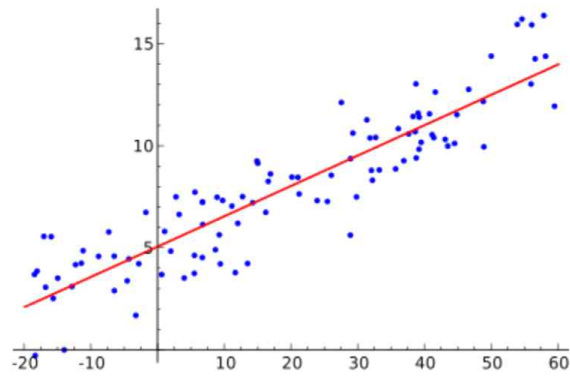
Multiple: x가 2개가 있는 것

선형: 면들도 x들의 선형결합으로 표현, 비선형: 면이 구부러진 경우

02-02 Chapter002 Linear Regression_2

02. Simple Linear Regression

$y = ax + b$ (a는 기울기, b는 y 절편)



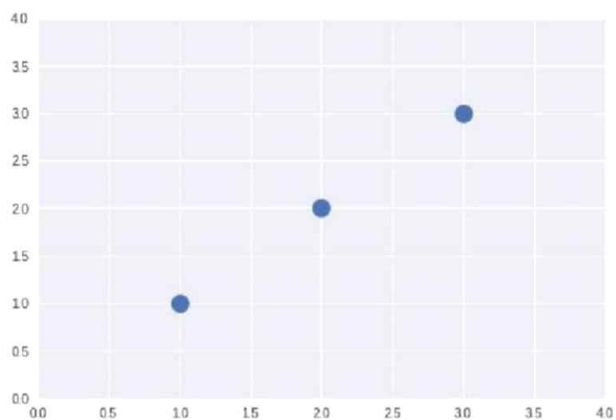
Simple Linear Regression

- Predicting exam score(시험 성적 예측)

x (hours) y (score)

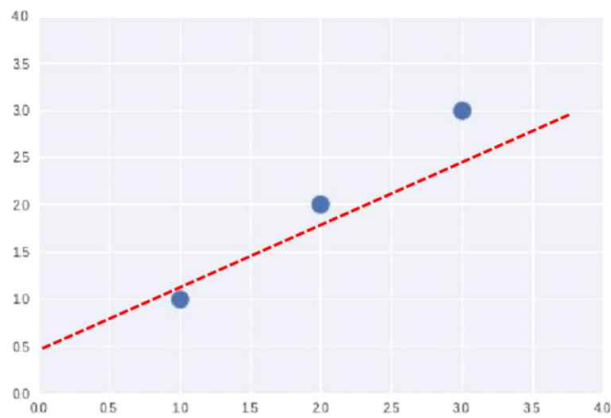
10	90
9	80
3	50
2	30

x	y
1	1
2	2
3	3



- Hypothesis

$$H(x) = Wx + b \quad (W = a)$$



Simple Linear Regression

- Hypothesis 중에 가장 최상은?

녹색선과 적색선 중 더 나은 모델은 무엇인가?

정답: 녹색선(mean square error의 크기가 녹색선이 더 작음, 예측값을 더 실제값과 가깝게 예측)

왼쪽 녹색선과 오른쪽 녹색선 중 더 나은 모델은 무엇인가?

(물론, 확정적 관계에 대해서는 머신러닝을 잘 이용하지 않음.)

정답: 왼쪽 (데이터들이 직선의 형태를 이루고 있음)

선 밖으로 나와있는 데이터들을 수치적으로 나타내어 보면, 에러를 이용하여 나타낼 수 있다.

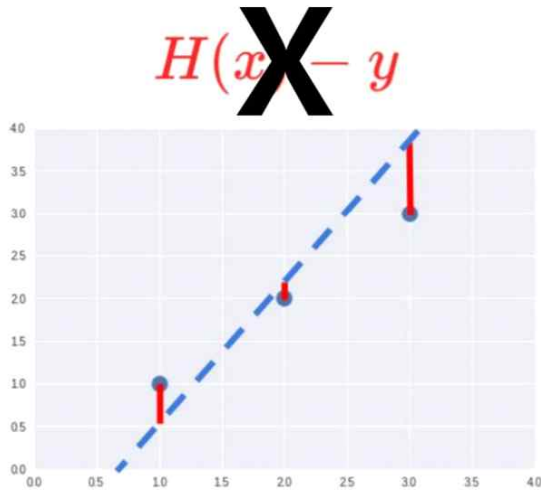
레드 사각형은 Square error들 이다. ==> **error에 제공한 값(넓이로 보자)**

- Cost(머신러닝에서는 square error라 하지 않고, cost라 한다)

$$H(x) = W(x) + b$$

$H(x) - y$: 우리의 가설(hypothesis)와 실제 데이터 간의 차이(빨간선)

- 빨간선의 길이가 짧을 수 록 파란색 식을 잘 대변하고 있다고 생각하면 됨.



빨간색 거리의 합을 구하는 것이 무의미 해질 수도 있음. 그렇기 때문에 **cost의 값을 제곱으로 취함.**

Simple Linear Regression

- Cost function

cost 함수를 다시 수식으로 나타내면 다음과 같다.

$$cost(W) = \frac{1}{m} \sum_{i=1}^m (Wx_i - y_i)^2$$

$$H(x) = Wx + b$$

$H(x)$ 를 대입하면 아래와 같은 식을 얻을 수 있음.

$$cost(W, b) = \frac{1}{m} \sum_{i=1}^m (H(x_i) - y_i)^2$$

가장 작은 W와 b를 구하는 것이 바로 Linear Regression의 학습이다.

03.

Simplified Hypothesis

Hypothesis $H(x) = Wx + b$

Cost $cost(W, b) = \frac{1}{m} \sum_{i=1}^m$

예측과 실제 데이터의 차이 즉 에러 or lost의 제곱의 합의 평균이 cost 함수!!

우리의 목표: W와 b를 작게 만드는 것.

간략화 하기 위해 b를 생략하면,

$$H(x) = Wx$$

$$cost(W) = \frac{1}{m} \sum_{i=1}^m (Wx_i - y_i)^2$$

- what cost(W) looks like?

-

$$cost(W) = \frac{1}{m} \sum_{i=1}^m (Wx_i - y_i)^2$$

x	y
1	1
2	2
3	3

$$\checkmark W = 0, cost(W) =$$

$$\gg \frac{1}{3}((0 \times 1 - 1)^2 + (0 \times 2 - 2)^2 + (0 \times 3 - 3)^2)$$

$$\checkmark W = 1, cost(W) = 0$$

$$\gg \frac{1}{3}((1 \times 1 - 1)^2 + (1 \times 2 - 2)^2 + (1 \times 3 - 3)^2)$$

$$\checkmark W = 2, cost(W) = 4.67$$

$$\gg \frac{1}{3}((2 \times 1 - 1)^2 + (2 \times 2 - 2)^2 + (2 \times 3 - 3)^2)$$

$$\checkmark W = 3, cost(W) = 18.67$$

$$\gg \frac{1}{3}((3 \times 1 - 1)^2 + (3 \times 2 - 2)^2 + (3 \times 3 - 3)^2)$$

cost의 값에 따라 함수의 값은 다음과 같이 변화 됨.

$$W = 0, cost(W) = 4.67$$

$$W = 1, cost(W) = 0$$

$$W = 2, cost(W) = 4.67$$

$$W = 3, cost(W) = 18.67$$

=> cost가 최소화 되는 때는 w 가 1일 때

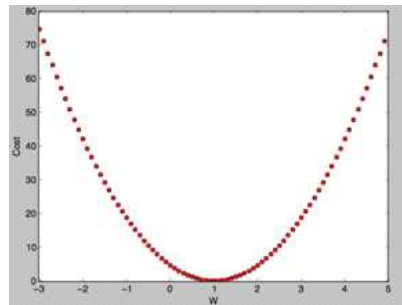
>> **Gradient descent algorithm**(경사하강 알고리즘: 경사를 따라 내려가면서 최적점을 찾는 알고리즘)

Minimize cost function

Gradient descent is used many minimization problems

For a given cost function, cost (W , b), it will find W , b to minimize cost

It can be applied to more general function : cost (w_1 , w_2 , ... w_n) : 변수가 하나 일 때 혹은 여러 개 일 때 도 사용할 수 있음.



>>Gradient descent algorithm의 동작 원리

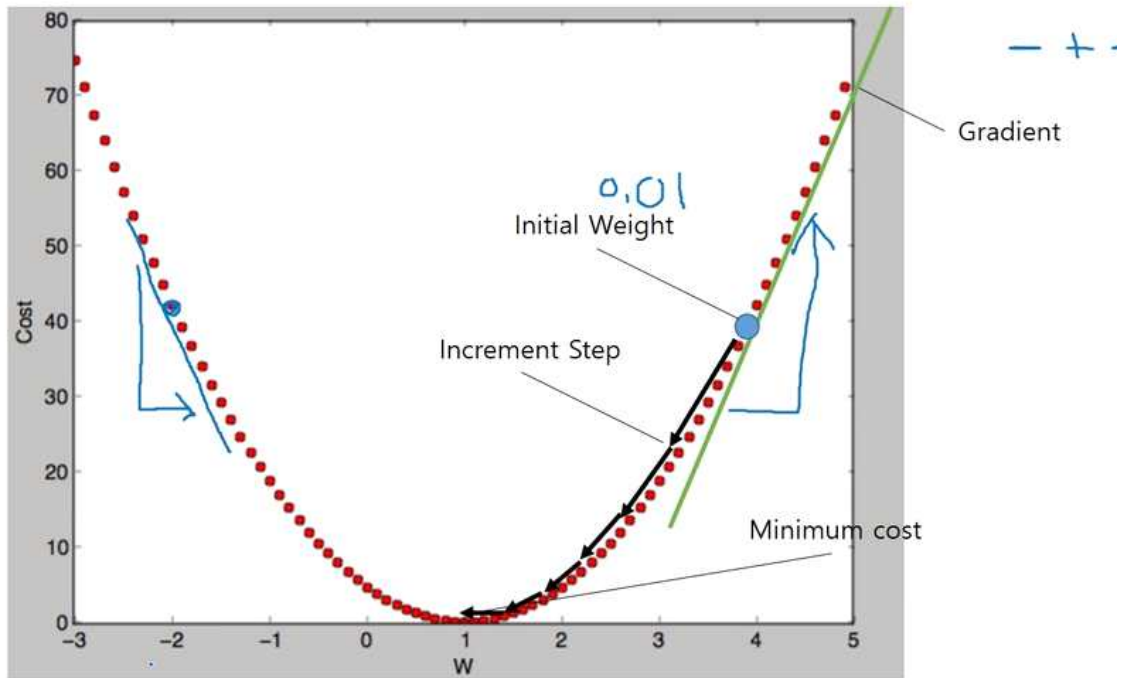
- Start with initial guesses
 - Start at 0,0 (or any other value)
 - Keeping changing W and b a little bit to try and reduce cost(W , b)
- Each time you change the parameters, you select the gradient which reduces cost(W , b) the most possible
- Repeat
- Do so until you converge to a local minimum
- Has an interesting property
- Where you start can determine which minimum you end up

그림에서 보이는 빨간점들은 cost 함수를 나타냄.

곡선 상에서 한 점을 정한다.

-> 웨이트 값을 초기화 하는 것.

웨이트는 가중치 값을 이야기 하는 것입니다.



보통 가중치는 매우 작은 값을 잡으며, 양수를 잡습니다.

=> 통상적으로 0.01정도로 잡는다

ex) W값의 초기값 4

기울기 즉 그레디언트 값을 구하고요.

이 W값에 웨이트값과 그레디언트 값을 곱해서 빼줍니다.

= $W - a(\text{weight의 값}) * \text{기울기 값}$

점점 W 값 감소하게 됨.

계속 동일하게 시행하다보면, 언젠가는 코스트가 최소가 되는 점에 도달 하게됨.

이 지점은 지금 기울기가 0.

우리가 웨이트에 기울기를 곱해서 W에 빼주기 때문에 이 지점이 되면 더 이상 진행이 되지 않게 될 것입니다.

이 지점이 바로 기울기가 최소가 되는 지점 즉 가장 낮은 지점이 된다.

아무곳에서나 시작을 해도 동일한 결과를 나타낸 다는 것입니다.

=>

ex) 반대방향에서 시작

이 기울기 값은 마이너스

즉 이 기울기 값을 웨이트에서 곱해서 W에 빼주면 이 기울기값 자체가 마이너스 이기 때문에...

마이너스 값을 빼주면 ??

증가하게 됩니다.

W값이 증가하는 방향으로 이동

또 이동하는 간격도 기울기를 곱한 값에 따라 움직이기 때문에 이 기울기 즉 경사가 급할수록 기울기 값 증가.

기울기 값이 더 크면 더 많이 이동

이 지점에 다가갈수록 매우 조금씩 이동을

이런 알고리즘이 바로 gradient descent algorithm:

즉, 경사로를 따라서 내려오면서 최저점을 찾는 그런 알고리즘

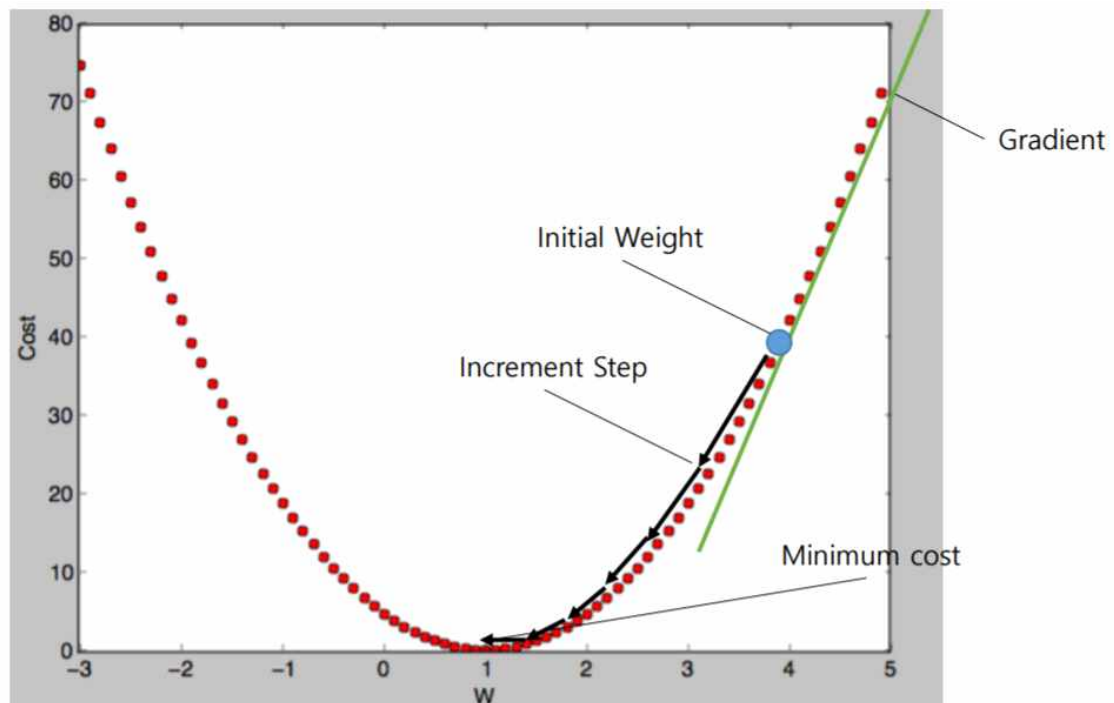
자 그럼 이 그레디언트 -> 미분을 통해 구함

이 곡선에 접하는 직선에 기울기를 바로 이 지점에서 미분값에 해당.

=====

03-01 Linear Regression _ 4

Gradient descent algorithm의 동작 원리



w 함수의 분포에 따라 cost가 분포되어있는 것을 나타낸 그래프

gradient descent algorithm을 적용하기 위해서는 임의의 한 점을 정해야됨.

그리고 경사값을 구해야함. -> 우리가 잡았던 W에 가중치값과 기울기를 곱한 것

빠기 -> cost가 최소가 될 때까지 반복 ㄱㄱ

이번 시간: 기울기를 어떻게 구할 것인가? (미분을 이용하자)

- Gradient descent algorithm Formal definition

$$W := W - \alpha \frac{\partial}{\partial W} \frac{1}{2m} \sum_{i=1}^m (Wx^{(i)} - y^{(i)})^2$$

$$W := W - \alpha \frac{1}{2m} \sum_{i=1}^m 2(Wx^{(i)} - y^{(i)})x^{(i)}$$

$$W := W - \alpha \frac{1}{m} \sum_{i=1}^m (Wx^{(i)} - y^{(i)})x^{(i)}$$

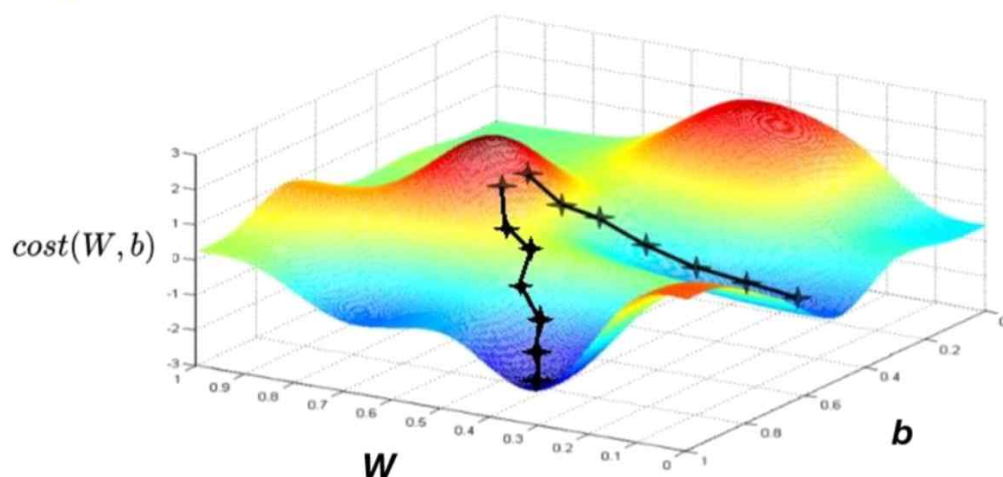
미분, 편미분 이용

최종결과 3번째 : Gradient descent algorithm 완성

미분 어려우면 Derivative Calculator or WolframAlpha 이용하여 계산 ㄱㄱ

이제 기계 적으로 적용시키면 cost 함수의 최소를 구할 수 있음.

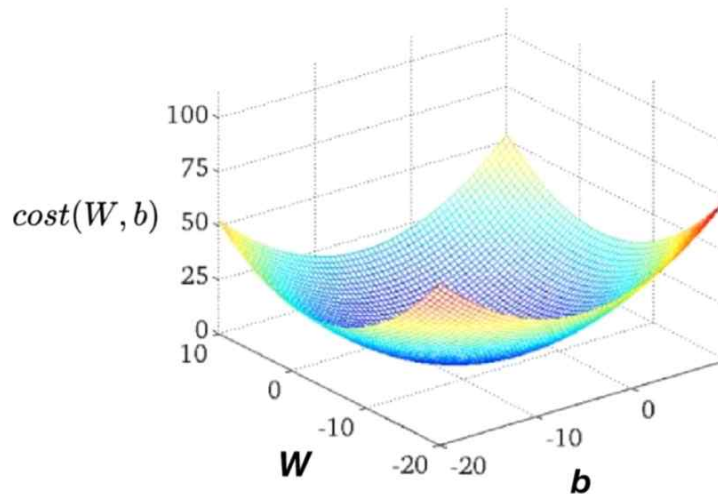
- Convex function



W 와 b 까지 적용시키면 3차원으로 표현 가능

- W 는 기울기이기 때문에 절편 값에 해당하는 값으로 돈다.
- b 값도 gradient를 이용하여 따로 구하면 최소값 구하기 가능
- W 그래프와 b 그래프를 합치면 위와 같이 3차원으로 구현가능.

$$cost(W, b) = \frac{1}{m} \sum_{i=1}^m (H(x_i) - y_i)^2$$



cost 함수를 정의할 때 모양이 convex 함수가 되는 지를 확인해야한다!

=====

03-02 Linear Regression_5

하나가 아닌 여러 개의 데이터가 들어왔을 때 어떻게 해야하나?

Predicting exam score : regression using one input (x)

one-variable
one-feature

x (hours)	y (score)
10	90
9	80
3	50
2	60
11	40

$H(x) = Wx + b$ 를 통해서 $Cost(W, b)$ 를 구하게 되면? ->

($H(x_i)$ 의 제곱) - y_i 의 제곱의 제곱의 $i=1$ 부터 m 까지의 합을 m 으로 나눈 값

데이터가 하나가 아닌 다음과 같이 3개가 있을 때의 y 값 예측

일단 hypothesis를 구해야함 3개의 값이 들어갔기 때문에 x_1, x_2, x_3

- Predicting exam score : regression using three input (x1, x2, x3)

multi-variable/feature			
x ₁ (quiz 1)	x ₂ (quiz 2)	x ₃ (midterm 1)	Y (final)
73	80	75	152
93	88	93	185
89	91	90	180
96	98	100	196
73	66	70	142

$$H(x_1, x_2, x_3) = w_1x_1 + w_2x_2 + w_3x_3 + b$$

만약 세 개 이상이라면 마찬가지로 x의 값을 더 늘리면 됨.

matrix (행렬)계산을 이용하여 더 편하게 계산 가능.

Hypothesis using matrix

$$w_1x_1 + w_2x_2 + w_3x_3 + \dots + w_nx_n$$

$$(x_1 \quad x_2 \quad x_3) \cdot \begin{pmatrix} w_1 \\ w_2 \\ w_3 \end{pmatrix} = (x_1w_1 + x_2w_2 + x_3w_3)$$

$$H(X) = XW$$

H(X)가 대문자로 표현된 것은 행렬로 표현된 것, 소문자는 행렬로 표현되지 않은 것이라고 생각하자.

행렬의 특성을 이용하여 W값 유추가능 ==> 첫 번째 행렬 열의 개수와 두 번째 행렬 행의 개수가 같아야함. -> 결과값의 열이 2이기 때문에

$$[?,?] = [3,2]$$

- Hypothesis using matrix

$$\begin{pmatrix} x_{11} & x_{12} & x_{13} \\ x_{21} & x_{22} & x_{23} \\ x_{31} & x_{32} & x_{33} \\ x_{41} & x_{42} & x_{43} \\ x_{51} & x_{52} & x_{53} \end{pmatrix} \cdot \text{?} = \begin{pmatrix} x_{11}w_{11} + x_{12}w_{21} + x_{13}w_{31} & x_{11}w_{12} + x_{12}w_{22} + x_{13}w_{32} \\ x_{21}w_{11} + x_{22}w_{21} + x_{23}w_{31} & x_{21}w_{12} + x_{22}w_{22} + x_{23}w_{32} \\ x_{31}w_{11} + x_{32}w_{21} + x_{33}w_{31} & x_{31}w_{12} + x_{32}w_{22} + x_{33}w_{32} \\ x_{41}w_{11} + x_{42}w_{21} + x_{43}w_{31} & x_{41}w_{12} + x_{42}w_{22} + x_{43}w_{32} \\ x_{51}w_{11} + x_{52}w_{21} + x_{53}w_{31} & x_{51}w_{12} + x_{52}w_{22} + x_{53}w_{32} \end{pmatrix}$$

[n, 3] [?, ?] [n, 2]

$$H(X) = XW$$

- WX vs XW

✓이론적인 설명

$$\text{» } H(x) = Wx + b \quad \begin{matrix} h_{\theta}(x) = \theta_1 x + \theta_0 \\ f(x) = ax + b \end{matrix}$$

✓구현 (TensorFlow)

$$\text{» } H(X) = XW$$

다양한 표현법들 but, 모두 1차 방정식으로 다 표현가능.

=====

03-03 Logistic Classification_1

굉장히 정확도가 높은 알고리즘.

Regression은 x값으로 y값을 예측하는 것 -> hypothesis 정의를 가지고 cost 함수를 이끌어낼 수 있음 -> cost 함수의 최소값은 gradient decent algorithm을 이용하여 구할 수 있음.

Classification (Binary는 0 -> False와 1 -> True로 분류를 하는 것)

- ✓이메일이 스팸인지 정상 메일인지 분류
- ✓제품이 불량인지 정상품인지 분류
- ✓카드 거래가 정상인지 사기 인지 분류
- ✓내원 고객이 질병이 있는지 없는지 분류
- ✓페이스북 피드에서 보이게 할지 숨길지

Regression과 Classification의 차이는?

0 ~ 100점까지 나타나는 것 -> Regression(연속성)

날씨를 온도로 표현 -> Regression(연속성)

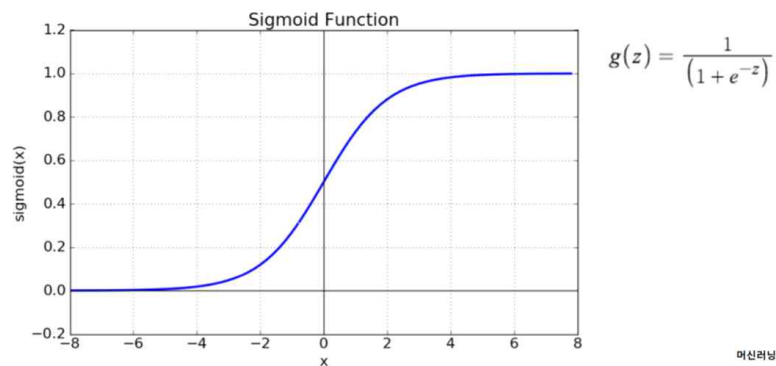
0이냐 1이냐 -> Classification(범주)

날씨가 추운지 더운지 -> Classification(범주)

주식 차트 -> 0 : 팔아야할 때, 1: 사야할 때 -> Logistic Classification

ex)골드만삭스 -> 이제 우리는 금융회사 X, IT 회사 O

classification/Regression 둘다 표현을 할 수 있으나 조건에 따라 표현의 한계가 있을 수 있음.



위의 시그모이드 함수 -> 0과 1로 무한하게 가까이 간다(단조 증가)

역행렬을 이용하여 Logistic Hypothesis를 다음과 같이 정의 내릴 수 있음.

- Logistic Hypothesis

$$H(x) = Wx + b$$



$$H(X) = \frac{1}{1 + e^{-W^T X}}$$

Linear Regression: 데이터들이 있으면 1차 방정식(ax+b 직선)이기 때문에 확실히 분류를 할 수 가 있음. (y 절편과 기울기 값만 안다면)
그러나,

Logistic Classification에서는 애매한 부분이 발생 -> 그럴 때는 어떻게 해야하나?
의미: 그래프의 절편이나 기울기를 조정을 할 수가 있을 것 같은데??

desmos.com : 그래프 그리는 사이트에서 직접 그래프를 그려보며 알아보면,

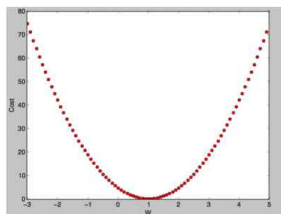
W(절편)와 b(기울기, 오른쪽)이 바뀔때 따라 그래프의 표현이 바뀐다.

04-01 Logistic Classification_2

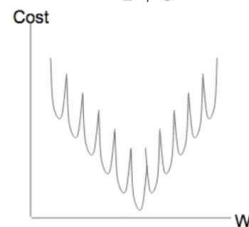
- Cost function

$$cost(W, b) = \frac{1}{m} \sum_{i=1}^m (H(x^{(i)}) - y^{(i)})^2$$

$$H(x) = Wx + b$$



$$H(X) = \frac{1}{1 + e^{-W^T X}}$$



그래프의 결과가 이러한 이유는 시그모이드 함수 때문이다.

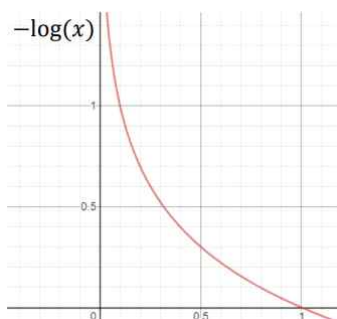
- New cost function for logistic

$$cost(W) = \frac{1}{m} \sum c(H(x), y)$$

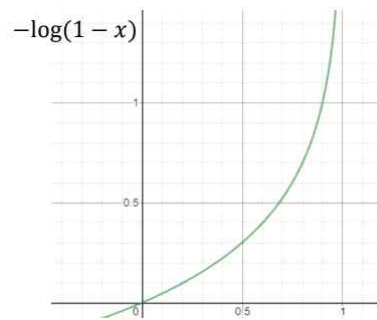
$$c(H(x), y) = \begin{cases} -\log(H(x)) & : y = 1 \\ -\log(1 - H(x)) & : y = 0 \end{cases}$$

로그함수 이용.(글로벌 미니멈 찾기 가능)

y=1 일 때 점점 0에 수렴, y=0 일 때 무한대로 수렴(에러값이 엄청나게 증가)



y=0 일 때 점점 0에 수렴, y=1 일 때 무한대로 수렴



이 친구를 단 한 번의 수식으로 사용(if 문 사용할 필요X)

$$cost(W) = \frac{1}{m} \sum c(H(x), y)$$

$$C(H(x), y) = \begin{cases} -\log(H(x)) & : y = 1 \\ -\log(1 - H(x)) & : y = 0 \end{cases}$$

$$C(H(x), y) = y \log(H(x)) - (1 - y) \log(1 - H(x))$$

이제 직접적으로 구현하려면 Gradient Descent 알고리즘 이용 (미분하겠다.)

ex) tensorflow 코드

```
# cost function
cost = tf.reduce_mean(-tf.reduce_sum(Y*tf.log(hypothesis) + (1-Y)*tf.log(1-hypothesis)))

# Minimize
a = tf.Variable(0.1) # Learning rate, alpha
optimizer = tf.train.GradientDescentOptimizer(a)
train = optimizer.minimize(cost)
```

=====

04-02 Softmax_1

Logistic regression.

$H(x) = Wx \rightarrow$ 값이 수치적으로 크게 나옴

0~1 사이의 값으로 표현할 수 있도록 sigmoid function을 이용하였음.

0이나 1로 수렴하는 함수.

cost 함수를 씌우면 우리가 원하는 글로벌 미니멈 값을 얻기위해 로그를 취해
완만하게 하여 표현하였음.

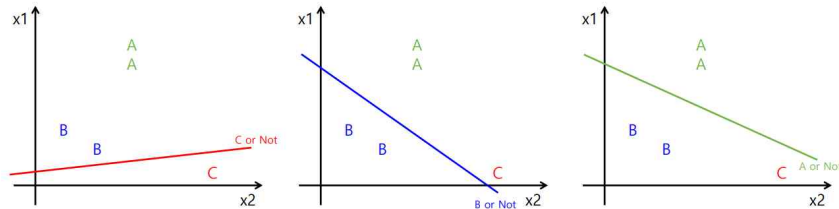
둘 중에 하나가 아닌 여러 개를 선택해야 할 때는 어떻게 해야하는가?

표현을 어떻게 해야 분리할 수 있을까?

- Multinomial classification

x1 (hours)	x2 (attendance)	y (grade)
10	5	A
9	5	A
3	2	B
2	4	B
11	1	C

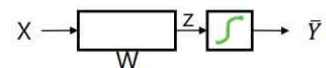
- Multinomial classification



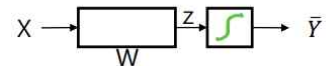
A인 것과 A가 아닌 것, B인 것과 B가 아닌 것, C인 것과 C가 아닌 것. 이런 식으로 분리할 수 있음

실제 X 값에는 A인 것, B인 것, C인 것 이렇게 3가지로 분류 한 후 시그모이드 함수를 이용해서 표현하면??

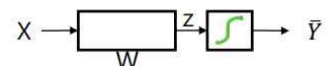
$$\begin{bmatrix} w_1 & w_2 & w_3 \end{bmatrix} \begin{bmatrix} x_1 \\ x_2 \\ x_3 \end{bmatrix} = [w_1x_1 \quad w_2x_2 \quad w_3x_3]$$



$$\begin{bmatrix} w_1 & w_2 & w_3 \end{bmatrix} \begin{bmatrix} x_1 \\ x_2 \\ x_3 \end{bmatrix} = [w_1x_1 \quad w_2x_2 \quad w_3x_3]$$



$$\begin{bmatrix} w_1 & w_2 & w_3 \end{bmatrix} \begin{bmatrix} x_1 \\ x_2 \\ x_3 \end{bmatrix} = [w_1x_1 \quad w_2x_2 \quad w_3x_3]$$



값 표현을 행렬로 하면 좋음. 행렬은 가로 곱하기 세로 위의 세가지 표현을 한 번에 묶어버리면?

$$\begin{bmatrix} w_{A1} & w_{A2} & w_{A3} \\ w_{B1} & w_{B2} & w_{B3} \\ w_{C1} & w_{C2} & w_{C3} \end{bmatrix} \begin{bmatrix} x_1 \\ x_2 \\ x_3 \end{bmatrix} = \begin{bmatrix} w_{A1}x_1 & w_{A2}x_2 & w_{A3}x_3 \\ w_{B1}x_1 & w_{B2}x_2 & w_{B3}x_3 \\ w_{C1}x_1 & w_{C2}x_2 & w_{C3}x_3 \end{bmatrix}$$

logistic classification으로도 충분히 multinomial로 다 구분 가능, 근데 why softmax가 나왔을까??

tensorflow에선 arg max이용. 0.9 , 0.8, 0.4, 0.2, 0.1

one-hot-encoding 방식을 이용하면 신뢰할 수 있는 데이터라고 판단하기 힘들. 학습하기도 난처.

때문에 확률 값으로 나오면 어떨까? 라는 생각을 하게됨. 그래서 시그모이드를 사용하지 않고 softmax를 사용함.

• Softmax Cost function

$$D(S, L) = - \sum_i L_i \log(S_i)$$

0.7
0.2
0.1

1.0
0.0
0.0

04-03 Softmax_2

행렬을 이용하여 x값이 하나가 들어가게됨.

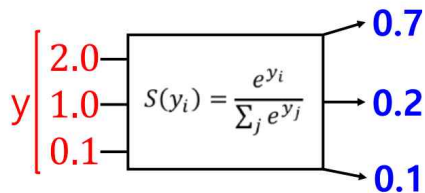
Logistic classifier 벡터 값으로 표현. 그러나 확률 값으로 표현되는 것이 가장 좋음.
ex) 고양이 = 0.7 , 강아지 = 0.2, 말= 0.1

$$WX = y \begin{bmatrix} 2.0 \\ 1.0 \\ 0.1 \end{bmatrix}$$

softmax란?

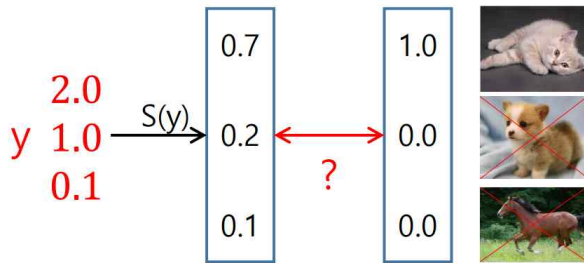
scores -> probabilities

벡터 값들이 들어가게 되면 확률 값으로 나오게됨.



아래와 같이 1.0, 0.0, 0.0 표현 ==> one-hot-encoding 방법
tensorflow에서는 argmax함수를 이용하여 표현.

로그함수를 이용하자.



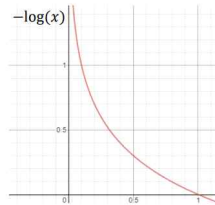
• Cross-entropy cost function

$$-\sum_i L_i \log(S_i) = -\sum_i L_i \log(\bar{y}_i) = \sum_i L_i * \log(\bar{y}_i)$$

$$L = \begin{bmatrix} 1 \\ 0 \end{bmatrix}$$

$$\bar{y} = \begin{bmatrix} 0 \\ 1 \end{bmatrix}$$

$$\bar{y} = \begin{bmatrix} 1 \\ 0 \end{bmatrix}$$



복잡하니깐 A와 B라고 한다면,

$$L = \begin{bmatrix} 0 \\ 1 \end{bmatrix} = B$$

$$\bar{y} = \begin{bmatrix} 0 \\ 1 \end{bmatrix} = B(x), \quad \begin{bmatrix} 0 \end{bmatrix} \odot -\log \begin{bmatrix} 0 \\ 1 \end{bmatrix} = \begin{bmatrix} 0 \end{bmatrix} \odot \begin{bmatrix} \infty \\ 0 \end{bmatrix} = \begin{bmatrix} 0 \end{bmatrix} \Rightarrow 0$$

$$\bar{y} = \begin{bmatrix} 1 \\ 0 \end{bmatrix} = A(x), \quad \begin{bmatrix} 1 \end{bmatrix} \odot -\log \begin{bmatrix} 0 \\ 1 \end{bmatrix} = \begin{bmatrix} 1 \end{bmatrix} \odot \begin{bmatrix} \infty \\ 0 \end{bmatrix} = \begin{bmatrix} \infty \end{bmatrix} \Rightarrow \infty$$

반대의 경우는 다음과 같다.

$$L = \begin{bmatrix} 1 \\ 0 \end{bmatrix} = A$$

$$\bar{y} = \begin{bmatrix} 0 \\ 1 \end{bmatrix} = B(x), \quad \begin{bmatrix} 0 \end{bmatrix} \odot -\log \begin{bmatrix} 1 \\ 0 \end{bmatrix} = \begin{bmatrix} 0 \end{bmatrix} \odot \begin{bmatrix} 0 \\ \infty \end{bmatrix} \Rightarrow 0$$

$$\bar{y} = \begin{bmatrix} 1 \\ 0 \end{bmatrix} = A(x), \quad \begin{bmatrix} 1 \end{bmatrix} \odot -\log \begin{bmatrix} 1 \\ 0 \end{bmatrix} = \begin{bmatrix} 1 \end{bmatrix} \odot \begin{bmatrix} 0 \\ \infty \end{bmatrix} \Rightarrow 0$$

시그모이드가 함수가 확률값으로 나오니깐.

위의 두 개의 식은 같은 식.

L1과 L2는 real 값.

이런 L의 값에 따라 확률값은 예측이 가능: $L2 = 1 - L1$

L2에 $1 - L1$ 을 대입, S2에 $1 - S1$ 을 대입해서 풀어서 계산해보면

두 식은 결국 같다.

- Logistic cost VS cross entropy

$$C(H(x), y) = y \log(H(x)) - (1 - y) \log(1 - H(x))$$

$$D(S, L) = - \sum_i L_i \log(S_i)$$

- Cost function

$$LOSS = \frac{1}{N} \sum_i D(S(WX_i + b), L_i)$$

평균.

gradient descent 함수를 이용하여 기울기를 구해서 cost를 구해 학습을 할 수 있다.

Lab_Chapter_002_04(20173147 이명진) ¶

1.NumPy

```
In [20]: import tensorflow as tf
import numpy as np
tf.set_random_seed(777) # for reproducibility
xy = np.loadtxt('Lab_Chapter002_01_Data(Test-Score).csv', delimiter=',', dtype=np.float32)
x_data = xy[:, 0:-1]
y_data = xy[:, -1]
# Make sure the shape and data are OK
print(x_data.shape, x_data, len(x_data))
print(y_data.shape, y_data)
# placeholders for a tensor that will be always fed.
X = tf.placeholder(tf.float32, shape=[None, 3])
Y = tf.placeholder(tf.float32, shape=[None, 1])
W = tf.Variable(tf.random_normal([3, 1]), name='weight')
b = tf.Variable(tf.random_normal([1]), name='bias')
# Hypothesis
hypothesis = tf.matmul(X, W) + b
# Simplified cost/loss function
cost = tf.reduce_mean(tf.square(hypothesis - Y))
# Minimize
optimizer = tf.train.GradientDescentOptimizer(learning_rate=1e-5)
train = optimizer.minimize(cost)

# Launch the graph in a session.
sess = tf.Session()
# Initializes global variables in the graph.
sess.run(tf.global_variables_initializer())
# Set up feed_dict variables inside the loop.
for step in range(10001):
    cost_val, hy_val, _ = sess.run(
        [cost, hypothesis, train],
        feed_dict={X: x_data, Y: y_data})
    if step % 10 == 0:
        print(step, "Cost: ", cost_val,
              "\nPrediction:\n", hy_val)
# Ask my score
print("Your score will be ", sess.run(hypothesis,
    feed_dict={X: [[100, 70, 101]]}))
print("Other scores will be ", sess.run(hypothesis,
    feed_dict={X: [[60, 70, 110], [90, 100, 80]]}))
```

```
(25, 3) [[ 73. 80. 75.]
 [ 93. 88. 93.]
 [ 89. 91. 90.]
 [ 96. 98. 100.]
 [ 73. 66. 70.]
 [ 53. 46. 55.]
 [ 69. 74. 77.]
 [ 47. 56. 60.]
 [ 87. 79. 90.]
 [ 79. 70. 88.]
 [ 69. 70. 73.]
 [ 70. 65. 74.]
 [ 93. 95. 91.]
 [ 79. 80. 73.]
 [ 70. 73. 78.]
 [ 93. 89. 96.]
 [ 78. 75. 68.]
 [ 81. 90. 93.]
 [ 88. 92. 86.]
 [ 78. 83. 77.]
 [ 82. 86. 90.]
 [ 86. 82. 89.]
 [ 78. 83. 85.]
 [ 76. 83. 71.]
 [ 96. 93. 95.]] 25
(25, 1) [[152.]
 [185.]
 [180.]
 [196.]
 [142.]
 [101.]
 [149.]
 [115.]
 [175.]
 [164.]
 [141.]
 [141.]
 [184.]
 [152.]
 [148.]
 [192.]
 [147.]
 [183.]
```

```
[[155.81144]]
970 Cost: 42.701973
Prediction:
[[155.86375]
 [180.46674]
 [181.10135]
 [201.1542 ]
 [132.54301]
 [102.67328]
 [158.10529]
 [128.01451]
 [171.70467]
 [166.8956 ]]
980 Cost: 42.514656
Prediction:
[[155.86128]
 [180.47633]
 [181.105 ]
 [201.14989]
 [132.56032]
 [102.67462]
 [158.0861 ]
 [127.97768]
 [171.70634]
 [166.87987]]
990 Cost: 42.328297
Prediction:
[[155.8588 ]
 [180.48589 ]
 [181.10864 ]
 [201.14554 ]
 [132.57756 ]
 [102.67599 ]
 [158.06699 ]
 [127.940926]
 [171.70804 ]
 [166.86421 ]]
1000 Cost: 42.142902
Prediction:
[[155.8563 ]
 [180.49544 ]
 [181.11226 ]
 [201.14122 ]
 [132.59479 ]
 [102.677376]
 [158.04793 ]
 [127.90429 ]
 [171.70978 ]
 [166.84865 ]]
```

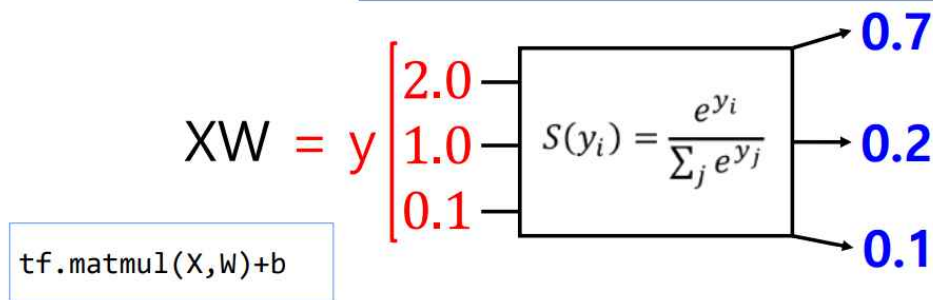
08-01 Lab_Softmax (1)

- Softmax function 여러 개의 클래스들을 예측할 때 매우 유용.

실생활에서는 1~2가지의 경우가 아니라 여러 경우들 즉, n개를 예측하는 경우가 많음.

- Softmax function

```
hypothesis = tf.nn.softmax(tf.matmul(X,W)+b)
```



소프트맥스에 적용하게 되면 위와 같이 확률 값으로 나오게된다.

(score -> probability: 모든 확률값들의 합은 항상 1)

- Cost function : cross entropy

$$LOSS = \frac{1}{N} \sum_i D(S(WX_i + b), L_i)$$

loss function은 y에 로그를 취한다.

Cross entropy cost/loss

```
cost = tf.reduce_mean(-tf.reduce_sum(Y * tf.log(hypothesis), axis=1))
```

```
optimizer = tf.train.GradientDescentOptimizer(learning_rate=0.1).minimize(cost)
```

hypothesis에 로그를 취하고 평균을 구하는 것 = cost

gradient descent 알고리즘을 사용하여 optimizer를 구한다.

실제 학습에 들어가는 함수는 arg_max함수를 이용

- Test & one-hot encoding

```
hypothesis = tf.nn.softmax(tf.matmul(X,W)+b)
```

```
# Testing & One-hot encoding
a = sess.run(hypothesis, feed_dict={X: [[1, 11, 7, 9]]})
print(a, sess.run(tf.argmax(a, 1)))
[[ 1.38904958e-03  9.98601854e-01  9.06129117e-06]] [1]
```

score -> probability -> 확률을 통해 물체 판단

여러개의 데이터를 삽입할 때는 feed_dict를 이용하여 삽입

```
all = sess.run(hypothesis, feed_dict={X: [[1, 11, 7, 9],
[1, 3, 4, 3],
[1, 1, 0, 1]]})
print(all, sess.run(tf.argmax(all, 1)))
```

=====

08-02 Lab_Softmax (2)

- softmax_cross_entropy_with_logits

```
logits = tf.matmul(X, W) + b
hypothesis = tf.nn.softmax(logits)
```

주어진 X에 대해서 우리가 학습할 W를 matmul로 곱한다음에 b를 더하는 것이 logits 이를 통해 어떤 내용의 확률이 나올지에 대한 계산값을 얻을 수 있음.

```
tf.matmul(X,W)+b
```

이게 바로 logits.

```
# Cross entropy cost/loss
cost = tf.reduce_mean(-tf.reduce_sum(Y * tf.log(hypothesis), axis=1))
```

Y는 one-hot으로 주어지고 hypothesis에 로그를 취한 값을 전체적인 평균을 내어보면 cost가 나옴 -> 이 과정이 어렵기 때문에 좀더 쉽게 할 수 있도록 나온 것이 tensorflow에서 softmax_cross_entropy with logits이다.

```
# Cross entropy cost/loss
cost_i = tf.nn.softmax_cross_entropy_with_logits(logits=logits,
labels=Y_one_hot)
cost = tf.reduce_mean(cost_i)
```

그 동안에는 hypothesis 값을 넣었었는데 여기서 logits 값을 넣는다는 것을 알고있자.

- Animal classification

✓with softmax_cross_entropy_with_logits

```
Y = tf.placeholder(tf.int32, [None, 1]) # 0 ~ 6, shape=(?, 1)
Y_one_hot = tf.one_hot(Y, nb_classes) # one hot shape=(?, 1, 7)
```

이 상태에서 실행시키면 100% 에러가 나오게 된다.

=> why?? one-hot을 이용할 때 rank를 N개를 이용하게 되면 output이 N+1이 나오게 됨. 즉, 다른 차원이 output으로 나올 수 있음.

```
Y_one_hot = tf.reshape(Y_one_hot, [-1, nb_classes]) # shape=(?, 7)
```

X의 데이터 개수 -> 16개 , Y의 데이터 개수 -> 1개
one-hot을 이용할 때는 Y의 개수를 그대로 집어넣자.

```
# Predicting animal type based on various features
xy = np.loadtxt('Lab_Chapter004_01_Data(zoo).csv', delimiter=',',
dtype=np.float32)
x_data = xy[:, 0:-1]
y_data = xy[:, [-1]]
nb_classes = 7 # 0 ~ 6
X = tf.placeholder(tf.float32, [None, 16])
Y = tf.placeholder(tf.int32, [None, 1]) # 0 ~ 6
Y_one_hot = tf.one_hot(Y, nb_classes) # one hot
Y_one_hot = tf.reshape(Y_one_hot, [-1, nb_classes])
W = tf.Variable(tf.random_normal([16, nb_classes]), name='weight')
b = tf.Variable(tf.random_normal([nb_classes]), name='bias')
# tf.nn.softmax computes softmax activations
# softmax = exp(logits) / reduce_sum(exp(logits), dim)
logits = tf.matmul(X, W) + b
hypothesis = tf.nn.softmax(logits)
# Cross entropy cost/loss
cost_i = tf.nn.softmax_cross_entropy_with_logits(logits=logits,
labels=Y_one_hot)
cost = tf.reduce_mean(cost_i)
optimizer =
```

```

tf.train.GradientDescentOptimizer(learning_rate=0.1).minimize(cost)
값 나오게 하는 출력
cost = tf.reduce_mean(cost_i)
optimizer =
tf.train.GradientDescentOptimizer(learning_rate=0.1).minimize(cost)
prediction = tf.argmax(hypothesis, 1)
correct_prediction = tf.equal(prediction, tf.argmax(Y_one_hot, 1))
accuracy = tf.reduce_mean(tf.cast(correct_prediction, tf.float32))
# Launch graph
with tf.Session() as sess:
    sess.run(tf.global_variables_initializer())
    for step in range(2000):
        sess.run(optimizer, feed_dict={X: x_data, Y: y_data})
        if step % 100 == 0:
            loss, acc = sess.run([cost, accuracy], feed_dict={
                X: x_data, Y: y_data})
            print("Step: {5}\tLoss: {:.3f}\tAcc: {:.2%}".format(
                step, loss, acc))
            # Let's see if we can predict
            pred = sess.run(prediction, feed_dict={X: x_data})
            # y_data: (N,1) = flatten => (N, ) matches pred.shape
            for p, y in zip(pred, y_data.flatten()): # Y 데이터를 일차원으로 하나로
                # 평평하게 집어넣겠다는 의미
                print("{} Prediction: {} True Y: {}".format(p == int(y), p, int(y)))

```

물론 정확도가 완벽히 100%는 아님. 그러나 매우 뛰어남

=====

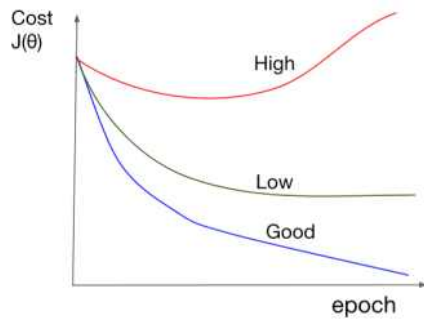
08-03 Application And Tip (1)

지금까지 Linear Regression, Soft-Max, Logistic Regression 등 다양한 기법들을 배워봤는데 이를 실제 현장에 적용하려면 많이 다른 경우가 존재함. 때문에 실제 현장에서 사용할 수 있는 팁들을 배워보자

Learning rate 값을 학습을 할 때 어떻게 조정을 해야하나?

1. Gradient Descent Algorithm

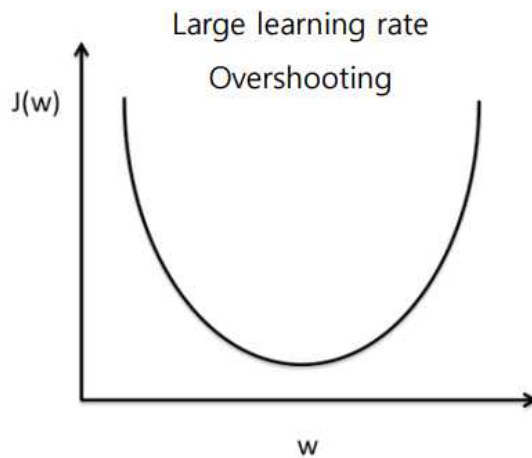
- Learning rate : Good and Bad -> Learning rate 값에 따라 천차만별.



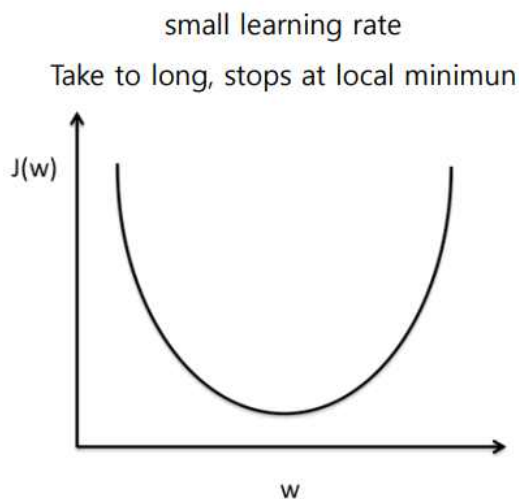
[Train Log]

Iter: 0, Loss: 6.0257, Learning Rate: 0.1000
 Iter: 1000, Loss: 0.3723, Learning Rate: 0.0960
 Iter: 2000, Loss: 0.2779, Learning Rate: 0.0922
 Iter: 3000, Loss: 0.2293, Learning Rate: 0.0885
 Iter: 4000, Loss: 0.1977, Learning Rate: 0.0849
 Iter: 5000, Loss: 0.1750, Learning Rate: 0.0815

Overshooting 값은 보통 Learning rate 값을 크게 설정했을 때 나타남.
 -> Learning rate 값에 따라 기울기가 크게 움직이느냐 아니냐를 보여줌.



-> bad에 대한 상황



최소 6만~10만 정도를 돌리는데 중간에 cost 값이 떨어지게 되면 트레이닝을 멈추게 됨. 그럼 결국 우리가 원하는 형태로 가지 못함.

-> 이를 방지하기 위해 계속해서 cost 값을 볼 수 밖에 없음.

- Try several learning rates
- ✓Observe the cost function
- ✓Check it goes down in a reasonable rate

데이터 양이 많은 학습을 돌리면 시간이 오래걸림. -> 때문에 그때 그때 중간에 Learning rate 값을 조절해줄 수 있는 기능이 있음.

```
[Tensorflow Code]
learning_rate = tf.train.exponential_decay(starter_learning_rate,
                                           global_step, 1000, 0.96, staircase=True)
# tf.train.exponential_decay / tf.train.inverse_time_decay
# tf.train.natural_exp_decay / tf.train.piecewise_constant
# tf.train.polynomial_decay

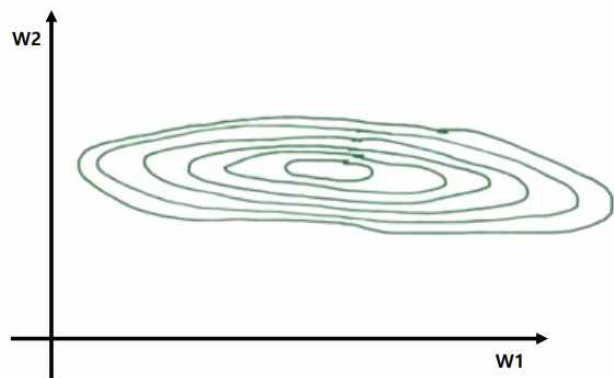
def exponential_decay(epoch):
    starter_rate = 0.01
    k = 0.96
    exp_rate = starter_rate * exp(-k*t)
    return exp_rate
```

decay를 이용하면 된다. 시간 날 때 더 자세히 알아보자.

- Data (X) preprocessing for gradient descent
- 전처리: 데이터를 더 쉽게 학습할 수 있도록 가공을 하는 것.

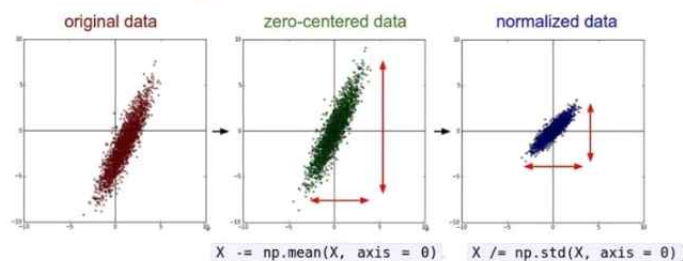
- Data (X) preprocessing for gradient descent

x1	x2	y
1	9000	A
2	-5000	A
4	-2000	B
6	8000	B
9	9000	C



전처리 값이 커져서 위와 같은 결과가 나올 수 있음.
overshooting이 발생할 수 도 있음.

- Data (X) preprocessing for gradient descent



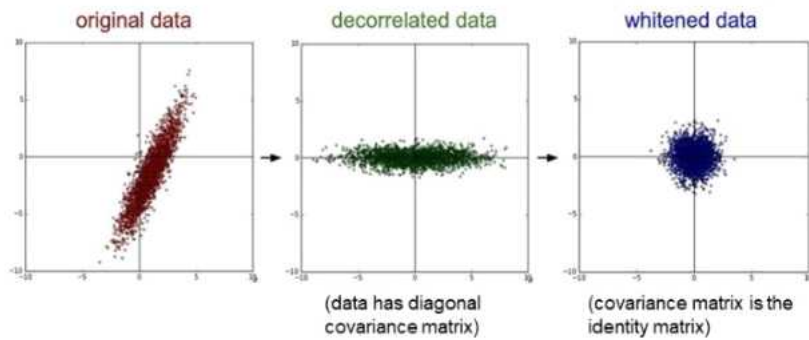
Standardization (Mean Distance)

$$x_{new} = \frac{x - \mu}{\sigma}$$

[Python Code(numpy)]
Standardization = (data - np.mean(data)) / sqrt(np.sum((data - np.mean(data))^2) / np.count(data))

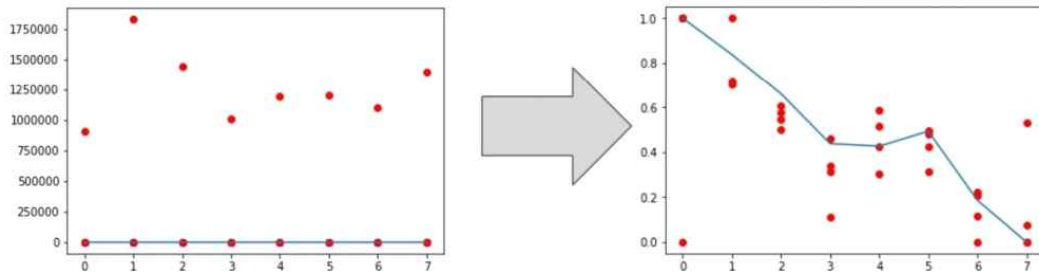
이를 위해, 표준편차를 나누어 일반화 시키는 방법으로 막을 수 있음.

- Data (X) preprocessing for gradient descent



또는 모아서 하나의 원의 형태로 모으는 방법이 있음.
물론 이미지와 같은 것들은 위의 두 가지 방법을 이용하지 않음.

- Feature Scaling

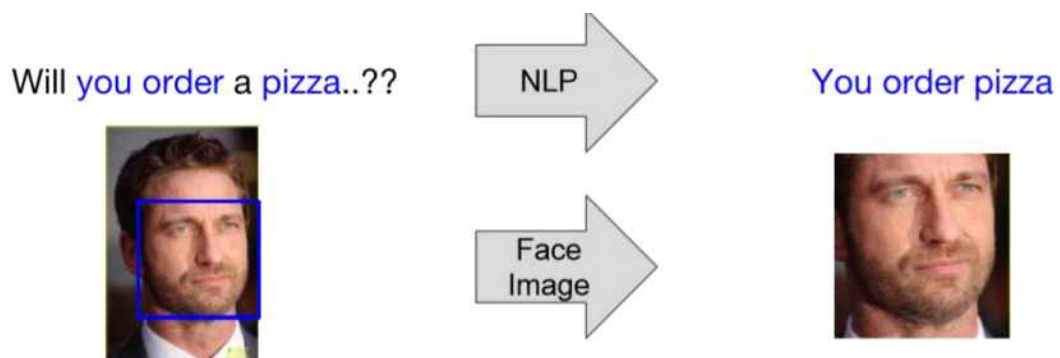


Normalization (0~1)

$$x_{new} = \frac{x - x_{min}}{x_{max} - x_{min}}$$

Normalization = (data - np.min(data, 0)) / (np.max(data, 0) - np.min(data, 0))

엄청나게 큰 값들로 생뚱맞게 존재하는 경우
-> 이러한 데이터들은 과감히 버려버리고 학습하는 경우도 있음.



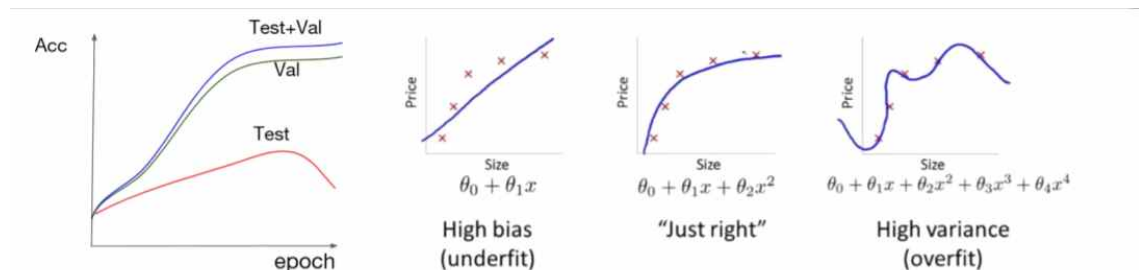
NLP 자연어 처리 방법에서 많이 사용하는 noisy data를 삭제하기도 하는 방식으로 전처리를 하기도한다.

얼굴인식 또한 눈 코 입과 같이 중요한 부분만 잘라 전처리

09-01 Application And Tip (2)

지난 시간,

어플리케이션의 팁 : learning rate, 데이터 전처리



이번 시간 주제: Overfitting -> 머신러닝에서 가장 큰 문제점.

의미 해석: over->과하게 , fitting -> 맞춰져 있다.

학습이 반복되면 되면 평가를 진행하면 녹색선과 같이 모델의 정확도가 증가하게됨.

그러나, 데이터가 적기 때문에 그 데이터에 맞게 실제 모델이 만들어진 경우.

물론 평가나 테스트를 한다고 하면 그렇지 않음. 즉, 모델에서 사용하지 않은

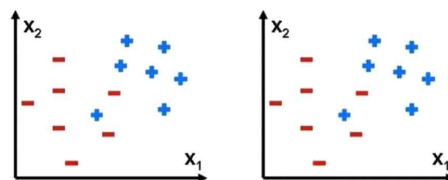
데이터가 들어오게되면 빨간 선과 같이 정확도가 떨어지게됨.

학습을 많이 하게되면 -> 정확도가 증가할 거야 라는 생각을 갖게됨.

bias(무언가 편향되어있다) 그래프를 살펴보면 학습이 이상적으로 됐다고 보는 건 just right, 학습이 덜 이루어진 것은 underfit, 학습이 이상적으로 되지 않음 -> overfit(variance 학습을 많이해서 변화량이 높은 것).

학습을 많이하게되면 -> 학습한 데이터만 인식을 하게되고 새로운 데이터가 들어왔을 때 학습을 하지 못함. 이를 overfitting이라고 한다.

어떠한 모델을 학습한다고 했을 때 가장 이상적인 모델은 무엇인가?



기울기가 -1인 직선을 x_2 와 x_1 점을 기준으로 그려보면 이상적인 모델이라고 볼 수 있음. 그러나 과도하게 학습을 하다보면 + 데이터 하나를 인식하지 못함.

이를 해결하기 위한 방법은 다음과 같다.

- Overfitting을 해결하기 위한 방법

✓Set a features

» Get more training data -> 더 많은 데이터를 집어넣으면 overfitting 방지 가능.

» Smaller set of features -> feature들의 개수를 줄이자(차수를 줄이면 인식을 증가)

» Add additional features -> 오히려 feature의 개수를 더 늘리자(많아지면 많아질수록 학습이 덜 되었을 때 즉, 의미있는 데이터가 별로 없을 때) 물론, 어느 특성의 의미있는 부분을 찾은 이후에는 2번째 방법과 같이 에러율이 증가한다.

✓Regularization (Add term to loss) -> 람다 값들은 더 줘서 하는 방법

추후에 Neural Network에 대해 배운 이후에 설명 예정.

✓Dropout(0.5 is common)

✓Batch Normalization

Overfitting 너무 학습이 많아져 변화량이 많으니 내가 학습한 모델이 대해서만 정확도가 높은 것이지 새로운 데이터가 들어오면 학습이 힘든 상황 -> 학습이 잘 되는 적정 수준을 찾아야함.

=====

09-02 Application And Tip (3)

overfitting -> 머신러닝에서 가장 골치 아픈 문제.

학습을 할 때 테스트 데이터 셋들은 어떻게 구성해야하는지와 어떻게 했을 때 학습이 가장 잘 되는 것인지에 대해 알아보자.

- Performance evaluation: is this good?

data -> MODEL -> 우리가 예측하는 가설값이 나오게 됨.

학습을 할 때 트레이닝 셋은 어떻게 구성해야하는가?

Size	Price
2104	400
1600	330
2400	369
1416	232
3000	540
1985	300
1534	315
1427	199
1380	121
1494	243

매칭을 시켜서 컴퓨터보고 외우라고 하면 ex) if (x== 2014)

y= 400

처럼 트레이닝 셋을 구성을 하면 과연 좋을까? => NO!!

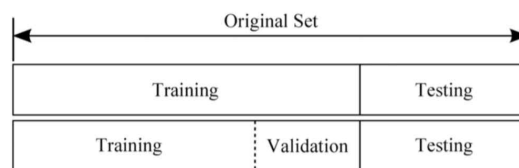
데이터 셋 중에서 트레이닝 셋 70%, 테스트 셋 30%으로 이용.

테스트 셋은 푹푹 숨겨놓는다. 이후에 학습을 시킨다. 이후에 예측값과 테스트 셋의 데이터들을 비교했을 때 인식률이 좋게 나타나면 학습이 잘 된 것.

즉, 처음부터 전체를 이용하는 것이 아님.

다른 경우, validation을 해야하는 경우.

- Training, validation and test sets



트레이닝 셋 70%, 테스트 셋 30%

위와 같이 구성되어있는 데이터들을 가지고 학습을 진행하면 된다.

데이터는 많으면 많을수록 좋다.

online learning -> 데이터가 천 만개 있다고 했을 때 모델이 학습을 하는데에 속도가 엄청나게 느릴 것은 뻔함. 온라인 러닝을 사용하면 천 만개의 데이터를 쪼개서 10만개 정도를 먼저 학습하고 마무리가 되면 그 다음 10만개를 넣어서 학습. 물론 이전에 학습된 데이터들은 온전히 저장. 천 만개를 한 번에 돌리는 것보다 속도가 훨씬 빠름. 또 다른 장점은 10만개의 데이터가 추후에 100만개로 늘었다면 이전에 저장되었는 10만개 때문에 90만개만 학습을 시키면됨 -> 인식률 또한 증가. 정적이 아닌 동적으로 활용. -> ex) 구글의 한국어 인식률, 과거보다 크게 인식률 향상. 처음에 만들 때는 물론 오프라인으로 할 수 밖에 없음. 어느정도 학습이 되었다고 생각되면 온라인 러닝으로 변경.

비교해보면 다음과 같다.

- Online learning vs Batch learning

	Online Learning	Batch(Offline) Learning
Data	Fresh	Static
Network	connected	disconnected
Model	Updating	Static
Weight	Tunning	initialize
Infra(GPU)	Always	Per call
Application	Realtime Process	Stopping
Priority	Speed	Correctness

온라인 러닝의 가장 우선순위는 속도! 배치는 초기에 먼저 이루어짐.

=====

Lab_chapter_005_02(20173147: 이명진)

Google Translate

MNIST dataset 실습

```
In [2]: import tensorflow as tf
import matplotlib.pyplot as plt
import random

tf.set_random_seed(777) # for reproducibility

from tensorflow.examples.tutorials.mnist import input_data
# Check out https://www.tensorflow.org/get_started/mnist/beginners for
# more information about the mnist dataset
mnist = input_data.read_data_sets("MNIST_data/", one_hot=True)
nb_classes = 10
# MNIST data image of shape 28 * 28 = 784
X = tf.placeholder(tf.float32, [None, 784])
# 0 - 9 digits recognition = 10 classes
Y = tf.placeholder(tf.float32, [None, nb_classes])
W = tf.Variable(tf.random_normal([784, nb_classes]))
b = tf.Variable(tf.random_normal([nb_classes]))

# Hypothesis (using softmax)
hypothesis = tf.nn.softmax(tf.matmul(X, W) + b)

cost = tf.reduce_mean(-tf.reduce_sum(Y * tf.log(hypothesis), axis=1))
optimizer = tf.train.GradientDescentOptimizer(learning_rate=0.1).minimize(cost)

# Test model
is_correct = tf.equal(tf.argmax(hypothesis, 1), tf.argmax(Y, 1))

# Calculate accuracy
accuracy = tf.reduce_mean(tf.cast(is_correct, tf.float32))

# parameters
training_epochs = 15
batch_size = 100
```

```
with tf.Session() as sess:
    # Initialize TensorFlow variables
    sess.run(tf.global_variables_initializer())
    # Training cycle
    for epoch in range(training_epochs):
        avg_cost = 0
        total_batch = int(mnist.train.num_examples / batch_size)

        for i in range(total_batch):
            batch_xs, batch_ys = mnist.train.next_batch(batch_size)
            c, _ = sess.run([cost, optimizer], feed_dict={X: batch_xs, Y: batch_ys})
            avg_cost += c / total_batch
            print('Epoch:', '%04d' % (epoch + 1), 'cost =', '{:.9f}'.format(avg_cost))

        print("Learning finished")

    # Test the model using test sets
    print("Accuracy: ", accuracy.eval(session=sess, feed_dict={X: mnist.test.images, Y: mnist.test.labels}))

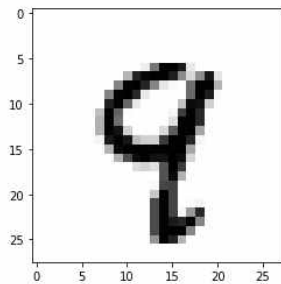
    # Get one and predict
    r = random.randint(0, mnist.test.num_examples - 1)
    print("Label:", sess.run(tf.argmax(mnist.test.labels[r:r+1], 1)))
    print("Prediction:", sess.run(tf.argmax(hypothesis, 1), feed_dict={X: mnist.test.images[r:r + 1]}))

    plt.imshow(
        mnist.test.images[r:r + 1].reshape(28, 28),
        cmap='Greys',
        interpolation='nearest')
    plt.show()
```

```

Epoch: 0001 cost = 2.826302786
Epoch: 0002 cost = 1.061668988
Epoch: 0003 cost = 0.838061343
Epoch: 0004 cost = 0.733232757
Epoch: 0005 cost = 0.669279900
Epoch: 0006 cost = 0.624611841
Epoch: 0007 cost = 0.591160355
Epoch: 0008 cost = 0.563868996
Epoch: 0009 cost = 0.541745182
Epoch: 0010 cost = 0.522673586
Epoch: 0011 cost = 0.506782334
Epoch: 0012 cost = 0.492447645
Epoch: 0013 cost = 0.479955842
Epoch: 0014 cost = 0.468893676
Epoch: 0015 cost = 0.458703490
Learning finished
Accuracy: 0.8951
Label: [9]
Prediction: [9]

```



11-01 딥러닝 (1)

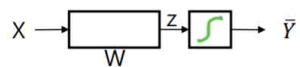
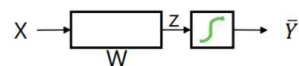
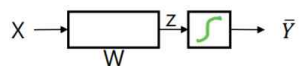
큰 골칫거리인 Xor이 어떻게 네트워크로 학습이 되는지 알아보자.

One logistic regression unit cannot separate XOR

-> 단일로 하면 xor을 표현할 수가 없음.

Multiple logistic regression units

-> 여러 개로 하면 표현 가능.



Neural Network (NN)

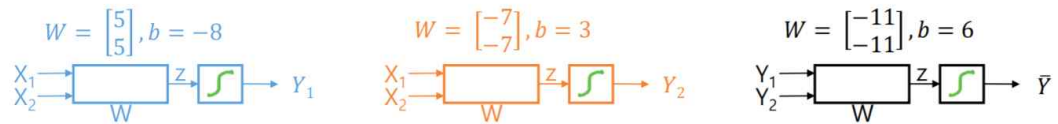
✓“No one on earth had found a viable way to train*”

-Marvin Minsky 안된다는 책만 안냈으면 참 좋았을 텐데.. 암흑기가 안왔을 텐데..

w와 b 값으로 계산을 할 수가 없음. -> 암흑기 도래.

그렇다면 여러 개의 네트워크를 사용하면 가능할까?

- Neural Net

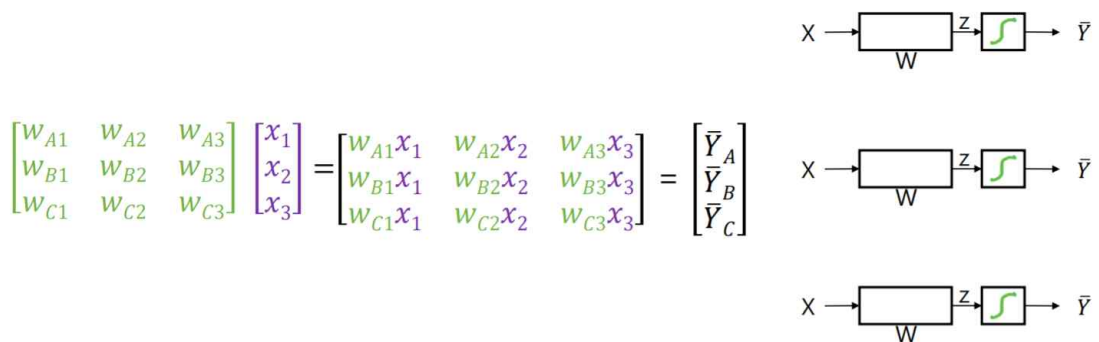


X_1	X_2	Y_1	Y_2	\bar{Y}	XOR
0	0	0	1	0	0
0	1	0	0	1	1
1	0	0	0	1	1
1	1				0

마지막 3개의 값은 각각, $5+5-8 = 3 \rightarrow$ sigmoid 하면 1
 $-7-7+3 = -11 \rightarrow$ sigmoid 하면 0
 $-11 +0+6 = -5 \rightarrow$ sigmoid 하면 0

위와 같이 세 개씩 있는 것들은 벡터 즉, 행렬 형태로 표현가능하다.

- Chapter004 : Multinomial classification



최종코드

=====

$$K(x) = \text{sigmoid}(xW_1 + b_1)$$

$$\bar{Y} = H(X) = \text{sigmoid}(K(x)W_2 + b_2)$$

```
# NN
K = tf.sigmoid(tf.matmul(X, W1) + b1)
hypothesis = tf.sigmoid(tf.matmul(K, W2) + b2)
```

gradient descent 값을 이용하여 최적의 값을 찾아 W값을 예측할 수 있음.

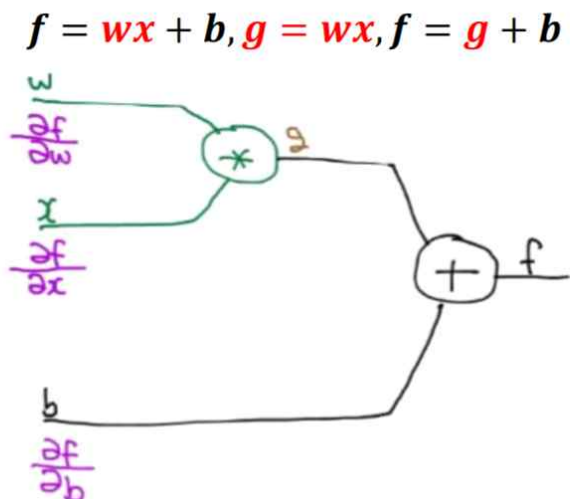
미스키 교수 -> 계산을 할 수 없다고 판단.

BackPropagation 방식을 이용하여 해결할 수 있다는 새로운 논문 발표.

chain rule 방식을 이용하여 해결하자 -> 뒤에서부터 찾아가는 것

기울기 -> 미분으로 구해 영향을 주는 것을 구할 수 있음. tensorflow에서 물론 그냥 계산가능하긴 하지만, 어느정도 미분에 대한 개념을 알고 있어야함.

- Back propagation (chain rule)



미분에 대해 알아보자.

미분? -> 값의 변화량을 나타내는 것. x의 변화량이 어떻게 되는지 살펴보는 것.

- Basic derivative

$$\frac{d}{dx}f(x) = \lim_{\Delta x \rightarrow 0} \frac{f(x + \Delta x) - f(x)}{\Delta x}$$

$$f(x) = 3$$

$$f(x) = x$$

$$f(x) = 2x$$

상수가 같이 존재한다면 따로따로 -> 그러나 상수는 미분해도 0

$$f(x, y) = x + y, \frac{\partial f}{\partial x}$$

$$f(x, y) = x + y, \frac{\partial f}{\partial y}$$

첫 번째: x에 대해서 미분한다. -> 즉 y는 상수 취급

두 번째: y에 대해서 미분한다. -> 즉, x는 상수 취급.

chain rule이란?? 연쇄법칙.

함수 g 가 x_0 에서 미분 가능하며, 함수 f 가 $g(x_0)$ 에서 미분 가능하다고 하자. 그렇다면, $f \circ g$ 는 x_0 에서 미분 가능하며, 그 미분은 다음과 같다.

$$(f \circ g)'(x_0) = f'(g(x_0))g'(x_0)$$

특히, 만약 g 가 구간 I 에서, f 가 $g(I)$ 에서 미분 가능하다면, $f \circ g$ 는 I 에서 미분 가능하며, 그 미분은 다음과 같다.

$$(f \circ g)' = (f' \circ g) \cdot g'$$

이를 **라이프니츠 표기법** 및 표기 $y = f(u)$, $u = g(x)$ 를 사용하여 다시 쓰면 다음과 같다.

$$\frac{dy}{dx} = \frac{dy}{du} \cdot \frac{du}{dx}$$

=====

지난 시간에 배운 미분만 이해하면 back propagation 충분히 이해가능.

미분을 해보면 다음과 같다.

• Back propagation (chain rule)

$f = wx + b, g = wx, f = g + b$

1. forward ($w = -2, x = 5, b = 3$)

2. backward

$$\frac{\partial f}{\partial w} = \frac{\partial f}{\partial g} \cdot \frac{\partial g}{\partial w} = 1 \cdot \frac{x}{5} = \frac{1}{5}$$

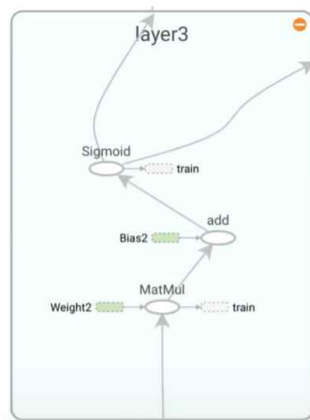
$$\frac{\partial f}{\partial x} = \frac{\partial f}{\partial g} \cdot \frac{\partial g}{\partial x} = 1 \cdot w = -2$$

YangWun Kim

$\frac{\partial f}{\partial g} = 1, \frac{\partial f}{\partial b} = 1$

$\frac{\partial g}{\partial w} = x, \frac{\partial g}{\partial x} = w$

• Back propagation in TensorFlow



`hypothesis = tf.sigmoid(tf.matmul(L2, W2) + b2)`

tensorflow가 알아서 해준다. tensor들이 미분을 이용하면 minsky 교수가 제기한 문제를 해결할 수 있음.

=====


12-01 Lab_딥러닝 (1)

sigmoid는 layer를 3장,4장 까지 쌓는 데는 문제가 없지만 그 이상(100개,200개)으로 증가하면 ~~ gradient 현상이 발생하게된다.

xor => A= x1 , B=x2

xor은 array 형태로 들어가게 될 것임.

- XOR data set

Boolean Expression	Logic Diagram Symbol	Truth Table															
$X = A \oplus B$		<table> <tr> <th>A</th><th>B</th><th>X</th></tr> <tr> <td>0</td><td>0</td><td>0</td></tr> <tr> <td>0</td><td>1</td><td>1</td></tr> <tr> <td>1</td><td>0</td><td>1</td></tr> <tr> <td>1</td><td>1</td><td>0</td></tr> </table>	A	B	X	0	0	0	0	1	1	1	0	1	1	1	0
A	B	X															
0	0	0															
0	1	1															
1	0	1															
1	1	0															

```

x_data = np.array([[0, 0], [0, 1], [1, 0], [1, 1]], dtype=np.float32)
y_data = np.array([[0], [1], [1], [0]], dtype=np.float32)

x_data = np.array([[0, 0], [0, 1], [1, 0], [1, 1]], dtype=np.float32)
y_data = np.array([[0], [1], [1], [0]], dtype=np.float32)

X = tf.placeholder(tf.float32)
Y = tf.placeholder(tf.float32)
W = tf.Variable(tf.random_normal([2, 1]), name='weight')
b = tf.Variable(tf.random_normal([1]), name='bias')

# Hypothesis using sigmoid: tf.div(1. - 1. + tf.exp(tf.matmul(X, W)))
hypothesis = tf.sigmoid(tf.matmul(X, W) + b)

# cost/loss function
cost = -tf.reduce_mean(Y * tf.log(hypothesis) + (1 - Y) * tf.log(1 - hypothesis))
train = tf.train.GradientDescentOptimizer(learning_rate=0.1).minimize(cost)

# Accuracy computation
# True if hypothesis>0.5 else False
predicted = tf.cast(hypothesis > 0.5, dtype=tf.float32)
accuracy = tf.reduce_mean(tf.cast(tf.equal(predicted, Y), dtype=tf.float32))

# Launch graph
with tf.Session() as sess:
    # Initialize TensorFlow variables
    sess.run(tf.global_variables_initializer())

    for step in range(10001):
        sess.run(train, feed_dict={X: x_data, Y: y_data})
        if step % 100 == 0:
            print(step, sess.run(cost, feed_dict={X: x_data, Y: y_data}), sess.run(W))

    # Accuracy report
    h, c, a = sess.run([hypothesis, predicted, accuracy], feed_dict={X: x_data, Y: y_data})
    print("\nHypothesis: ", h, "\nCorrect: ", c, "\nAccuracy: ", a)

```

XOR with
logistic regression?
But
it doesn't work!

Hypothesis:
[[0.5]
[0.5]
[0.5]
[0.5]]
Correct:
[[0.1]
[0.1]
[0.1]
[0.1]]
Accuracy: 0.5

예측값도 0.5정도로 나옴.

- Wide NN for XOR => 네트워크를 깊게 쌓는 것.

```

W1 = tf.Variable(tf.random_normal([2, 10]), name='weight1')
b1 = tf.Variable(tf.random_normal([10]), name='bias1')
layer1 = tf.sigmoid(tf.matmul(X, W1) + b1)
W2 = tf.Variable(tf.random_normal([10, 1]), name='weight2')
b2 = tf.Variable(tf.random_normal([1]), name='bias2')
hypothesis = tf.sigmoid(tf.matmul(layer1, W2) + b2)

```

10개 펼치기만 했는데 미묘하게 그래도 정확도가 좋아짐.

Deep 또한 마찬가지

- Deep NN for XOR

```

W1 = tf.Variable(tf.random_normal([2, 10]), name='weight1')

```

[2,10], [10,1]	[2,2], [2,1]
Hypothesis:	Hypothesis:
[[0.00358802]	[[0.01338218]
[0.99366933]	[0.98166394]
[0.99204296]	[0.98809403]
[0.0095663]]	[0.01135799]]
Correct:	Correct:
[[0.]	[[0.]
[1.]	[1.]
[1.]	[1.]
[0.]]	[0.]]
Accuracy: 1.0	Accuracy: 1.0

```

b1 = tf.Variable(tf.random_normal([10]), name='bias1')
layer1 = tf.sigmoid(tf.matmul(X, W1) + b1)
W2 = tf.Variable(tf.random_normal([10, 10]), name='weight2')
b2 = tf.Variable(tf.random_normal([10]), name='bias2')
layer2 = tf.sigmoid(tf.matmul(layer1, W2) + b2)
W3 = tf.Variable(tf.random_normal([10, 10]), name='weight3')
b3 = tf.Variable(tf.random_normal([10]), name='bias3')
layer3 = tf.sigmoid(tf.matmul(layer2, W3) + b3)
W4 = tf.Variable(tf.random_normal([10, 1]), name='weight4')
b4 = tf.Variable(tf.random_normal([1]), name='bias4')
hypothesis = tf.sigmoid(tf.matmul(layer3, W4) + b4)

```

4 layers	2 layers
Hypothesis:	Hypothesis:
[[7.80e-04]	[[0.01338218]
[9.99e-01]	[0.98166394]
[9.98e-01]	[0.98809403]
[1.55e-03]]	[0.01135799]]
Correct:	Correct:
[[0.]	[[0.]
[1.]	[1.]
[1.]	[1.]
[0.]]	[0.]]
Accuracy: 1.0	Accuracy: 1.0

네트워크를 어떠한 모양으로 쌓느냐에 따라 달라지는 것이 딥러닝의 핵심!!

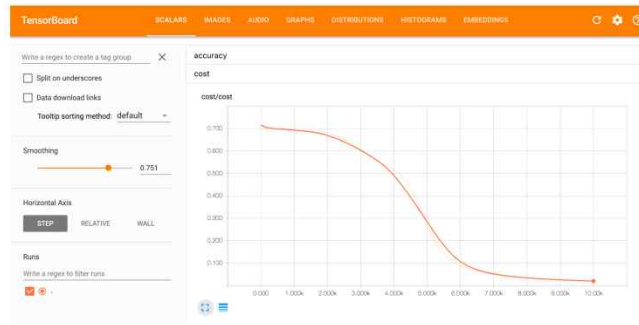
=====

12-02 Lab_딥러닝 (2)

Tensorboard 사용법을 실습해 본다.

- Old fashion : print, print, print => 가시성이 너무 안좋다. tensorboard를 이용하자

- TensorBoard: TF logging/debugging tool
- ✓Visualize your TF graph
- ✓Plot quantitative metrics
- ✓Show additional data



- 5 steps of using TensorBoard

5가지 방법으로 진행된다. ctrl c,v 말고 슬래쉬 주의하자!

1. From TF graph, decide which tensors you want to log

```
w2_hist = tf.summary.histogram("weights2", W2)
cost_summ = tf.summary.scalar("cost", cost)
```

2. Merge all summaries

```
summary = tf.summary.merge_all()
```

3. Create writer and add graph

```
# Create summary writer
writer = tf.summary.FileWriter("C:/MachineLearning/logs")
writer.add_graph(sess.graph)
```

4. Run summary merge and add_summary

```
s, _ = sess.run([summary, optimizer], feed_dict=feed_dict)
writer.add_summary(s, global_step=global_step)
```

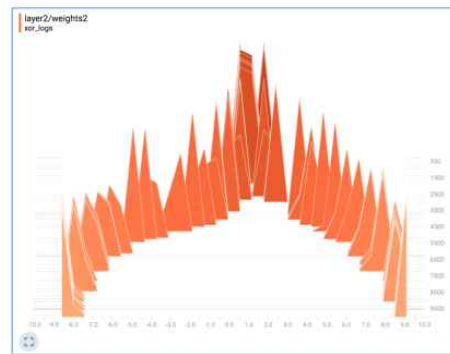
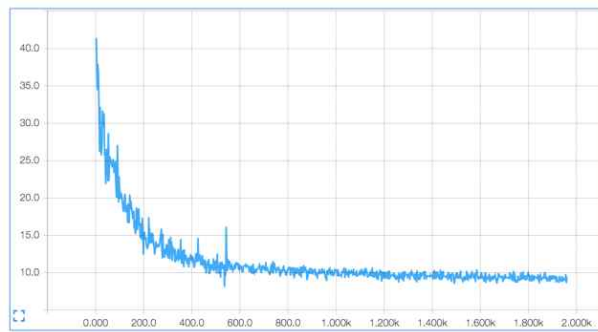
5. Launch TensorBoard

```
tensorboard --logdir=C:/MachineLearning/logs
```

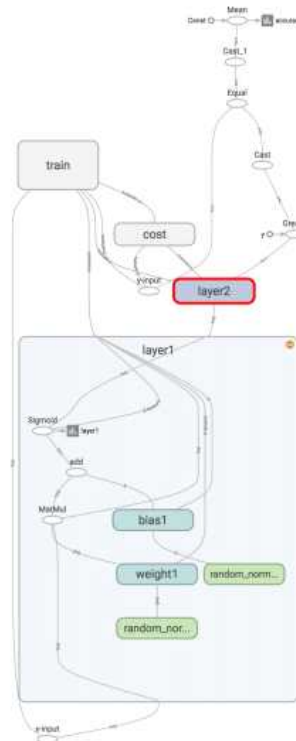
- Scalar tensors

```
cost_summ = tf.summary.scalar("cost", cost)
```

- Histogram (multi-dimensional tensors)



- Add scope for better graph hierarchy



- Multiple runs

```

tensorboard --logdir=./logs/xor_logs
train = tf.train.GradientDescentOptimizer(learning_rate=0.1).minimize(cost)
...
writer = tf.summary.FileWriter("./logs/xor_logs")
tensorboard --logdir=./logs/xor_logs_r0_01

```

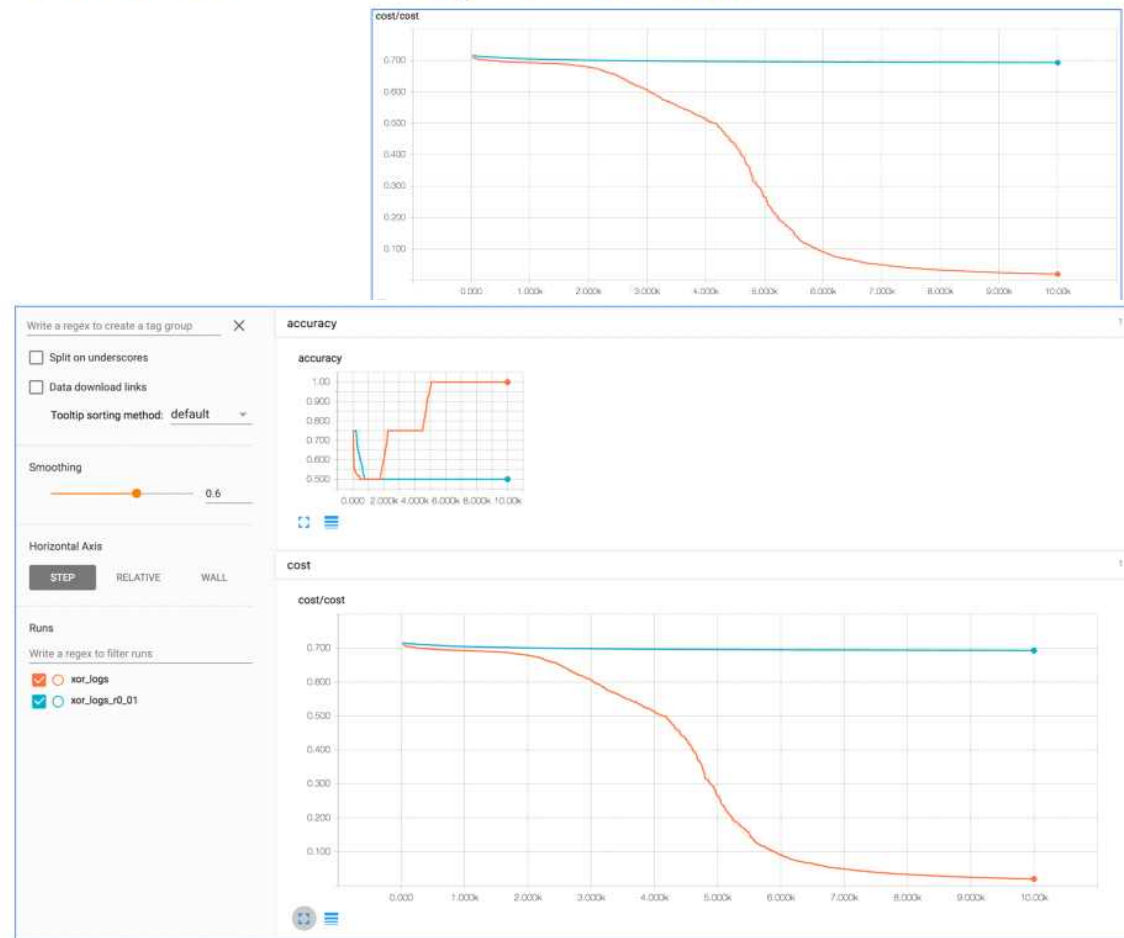


```
train = tf.train.GradientDescentOptimizer(learning_rate=0.01).minimize(cost)
```

```
...
```

```
writer = tf.summary.FileWriter("./logs/xor_logs_r0_01")
```

tensorboard --logdir=./logs

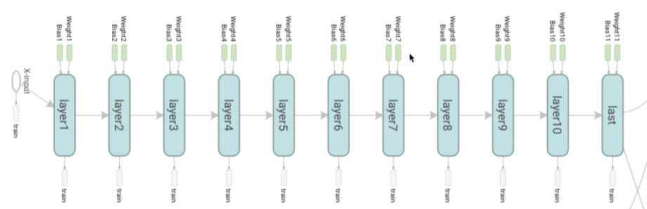


=====

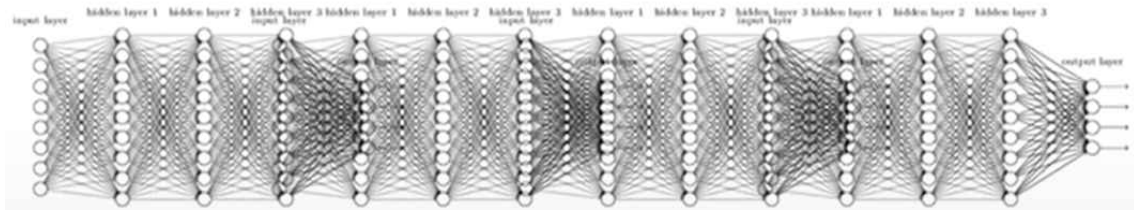
13-03 Lab_답러닝 (3)

ReLU

Tensorboard visualization

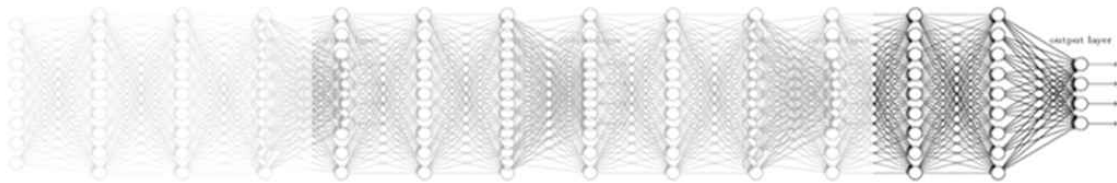


layer에 구성된 부분이 답하게 ㄱㄱ



back propagation이 등장하면서 암흑기에서 벗어날 수 있었다.

Vanishing gradient (NN winter2: 1986-2006) => 20년동안 또 암흑기 시작
경사 기울기가 사라진다.



Geoffrey Hinton's summary of findings up to today

Our labeled datasets were thousands of times too small.
Our computers were millions of times too slow.
We initialized the weights in a stupid way.
We used the wrong type of non-linearity.



곱하고곱하고 곱하면서 vanishing gradient 현상이 발생하게된다.

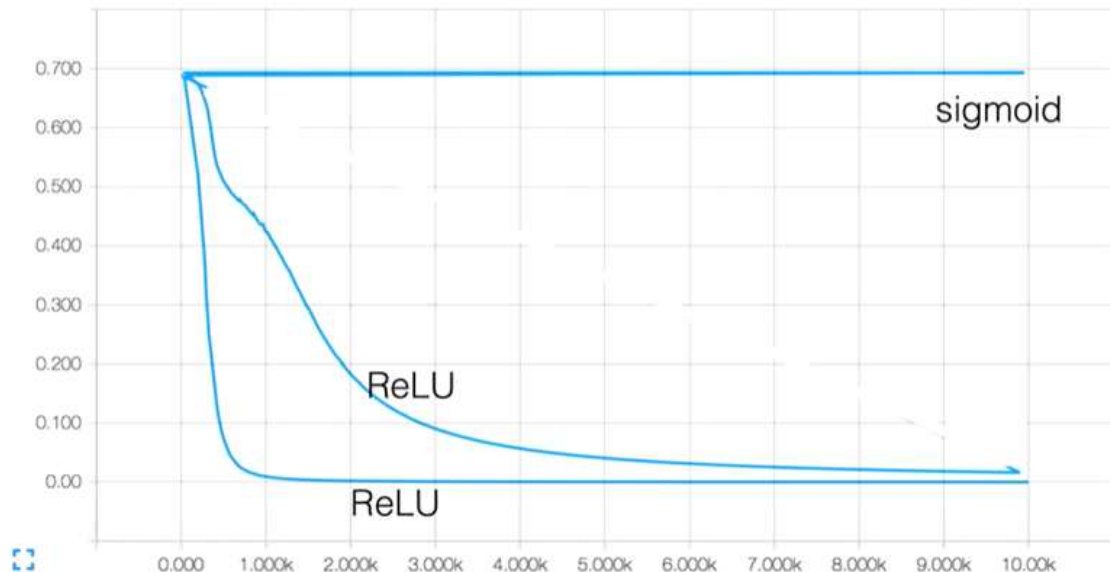
$$L1 = \text{tf.sigmoid}(\text{tf.matmul}(X, W1) + b1)$$

$$L1 = \text{tf.nn.relu}(\text{tf.matmul}(X, W1) + b1)$$

결국 시그모이드에서 렐루 함수로 바뀌게됨. 그러나 맨 마지막에는 시그모이드를 사용해야한다!

activation function

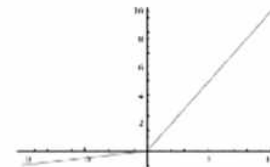
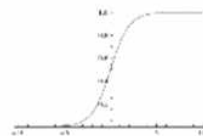
cost



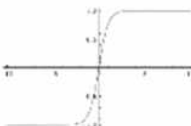
Leaky ReLU
 $\max(0.1x, x)$

Sigmoid

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



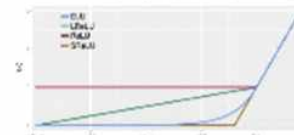
tanh $\tanh(x)$



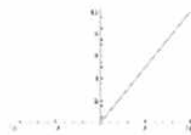
Maxout $\max(w_1^T x + b_1, w_2^T x + b_2)$

ELU

$$f(x) = \begin{cases} x & \text{if } x > 0 \\ \alpha (\exp(x) - 1) & \text{if } x \leq 0 \end{cases}$$



ReLU $\max(0, x)$



```
In [*]: # MNIST Softmax
import tensorflow as tf
import random
# import matplotlib.pyplot as plt
from tensorflow.examples.tutorials.mnist import input_data
tf.set_random_seed(777) # reproducibility

mnist = input_data.read_data_sets("MNIST_data/", one_hot=True)
# Check out https://www.tensorflow.org/get_started/mnist/beginners for
# more information about the mnist dataset

# parameters
learning_rate = 0.001
training_epochs = 15
batch_size = 100

# input place holders
X = tf.placeholder(tf.float32, [None, 784])
Y = tf.placeholder(tf.float32, [None, 10])

# weights & bias for nn layers
W = tf.Variable(tf.random_normal([784, 10]))
b = tf.Variable(tf.random_normal([10]))

hypothesis = tf.matmul(X, W) + b

# define cost/loss & optimizer
cost = tf.reduce_mean(tf.nn.softmax_cross_entropy_with_logits(
    logits=hypothesis, labels=Y))
optimizer = tf.train.AdamOptimizer(learning_rate=learning_rate).minimize(cost)

# initialize
sess = tf.Session()
sess.run(tf.global_variables_initializer())

# train my model
for epoch in range(training_epochs):
    avg_cost = 0
    total_batch = int(mnist.train.num_examples / batch_size)
```

```

    for i in range(total_batch):
        batch_xs, batch_ys = mnist.train.next_batch(batch_size)
        feed_dict = {X: batch_xs, Y: batch_ys}
        c, _ = sess.run([cost, optimizer], feed_dict=feed_dict)
        avg_cost += c / total_batch

    print('Epoch:', '%04d' % (epoch + 1), 'cost =', '{:.9f}'.format(avg_cost))

print('Learning Finished!')

# Test model and check accuracy
correct_prediction = tf.equal(tf.argmax(hypothesis, 1), tf.argmax(Y, 1))
accuracy = tf.reduce_mean(tf.cast(correct_prediction, tf.float32))
print('Accuracy:', sess.run(accuracy, feed_dict={
    X: mnist.test.images, Y: mnist.test.labels}))

# Get one and predict
r = random.randint(0, mnist.test.num_examples - 1)
print("Label: ", sess.run(tf.argmax(mnist.test.labels[r:r + 1], 1)))
print("Prediction: ", sess.run(
    tf.argmax(hypothesis, 1), feed_dict={X: mnist.test.images[r:r + 1]}))

# plt.imshow(mnist.test.images[r:r + 1]).
# reshape(28, 28), cmap='Greys', interpolation='nearest')
# plt.show()

# MNIST NN
import tensorflow as tf
import random
# import matplotlib.pyplot as plt

from tensorflow.examples.tutorials.mnist import input_data

tf.set_random_seed(777) # reproducibility

mnist = input_data.read_data_sets("MNIST_data/", one_hot=True)
# Check out https://www.tensorflow.org/get_started/mnist/beginners for
# more information about the mnist dataset

# parameters
learning_rate = 0.001
training_epochs = 15
batch_size = 100

# input place holders
X = tf.placeholder(tf.float32, [None, 784])
Y = tf.placeholder(tf.float32, [None, 10])

```

```

# weights & bias for nn layers
W1 = tf.Variable(tf.random_normal([784, 256]))
b1 = tf.Variable(tf.random_normal([256]))
L1 = tf.nn.relu(tf.matmul(X, W1) + b1)

W2 = tf.Variable(tf.random_normal([256, 256]))
b2 = tf.Variable(tf.random_normal([256]))
L2 = tf.nn.relu(tf.matmul(L1, W2) + b2)

W3 = tf.Variable(tf.random_normal([256, 10]))
b3 = tf.Variable(tf.random_normal([10]))
hypothesis = tf.matmul(L2, W3) + b3

# define cost/loss & optimizer
cost = tf.reduce_mean(tf.nn.softmax_cross_entropy_with_logits(
    logits=hypothesis, labels=Y))
optimizer = tf.train.AdamOptimizer(learning_rate=learning_rate).minimize(cost)

# initialize
sess = tf.Session()
sess.run(tf.global_variables_initializer())

# train my model
for epoch in range(training_epochs):
    avg_cost = 0
    total_batch = int(mnist.train.num_examples / batch_size)

    for i in range(total_batch):
        batch_xs, batch_ys = mnist.train.next_batch(batch_size)
        feed_dict = {X: batch_xs, Y: batch_ys}
        c, _ = sess.run([cost, optimizer], feed_dict=feed_dict)
        avg_cost += c / total_batch

    print('Epoch:', '%04d' % (epoch + 1), 'cost =', '{:.9f}'.format(avg_cost))

print('Learning Finished!')

# Test model and check accuracy
correct_prediction = tf.equal(tf.argmax(hypothesis, 1), tf.argmax(Y, 1))
accuracy = tf.reduce_mean(tf.cast(correct_prediction, tf.float32))
print('Accuracy:', sess.run(accuracy, feed_dict={
    X: mnist.test.images, Y: mnist.test.labels}))

```

```

# Get one and predict
r = random.randint(0, mnist.test.num_examples - 1)
print("Label: ", sess.run(tf.argmax(mnist.test.labels[r:r + 1], 1)))
print("Prediction: ", sess.run(
    tf.argmax(hypothesis, 1), feed_dict={X: mnist.test.images[r:r + 1]}))

# plt.imshow(mnist.test.images[r:r + 1].
#             reshape(28, 28), cmap='Greys', interpolation='nearest')
# plt.show()

# MNIST NN Xavier
import tensorflow as tf
import random
# import matplotlib.pyplot as plt

from tensorflow.examples.tutorials.mnist import input_data

tf.set_random_seed(777) # reproducibility

mnist = input_data.read_data_sets("MNIST_data/", one_hot=True)
# Check out https://www.tensorflow.org/get_started/mnist/beginners for
# more information about the mnist dataset

# parameters
learning_rate = 0.001
training_epochs = 15
batch_size = 100

# input place holders
X = tf.placeholder(tf.float32, [None, 784])
Y = tf.placeholder(tf.float32, [None, 10])

# weights & bias for nn layers
# http://stackoverflow.com/questions/38640581/how-to-do-xavier-initialization-on-tensorflow
W1 = tf.get_variable("W1", shape=[784, 256],
                    initializer=tf.contrib.layers.xavier_initializer())
b1 = tf.Variable(tf.random_normal([256]))
L1 = tf.nn.relu(tf.matmul(X, W1) + b1)

W2 = tf.get_variable("W2", shape=[256, 256],
                    initializer=tf.contrib.layers.xavier_initializer())
b2 = tf.Variable(tf.random_normal([256]))
L2 = tf.nn.relu(tf.matmul(L1, W2) + b2)

W3 = tf.get_variable("W3", shape=[256, 10],
                    initializer=tf.contrib.layers.xavier_initializer())
b3 = tf.Variable(tf.random_normal([10]))
hypothesis = tf.matmul(L2, W3) + b3

```

```

# define cost/loss & optimizer
cost = tf.reduce_mean(tf.nn.softmax_cross_entropy_with_logits(
    logits=hypothesis, labels=Y))
optimizer = tf.train.AdamOptimizer(learning_rate=learning_rate).minimize(cost)

# initialize
sess = tf.Session()
sess.run(tf.global_variables_initializer())

# train my model
for epoch in range(training_epochs):
    avg_cost = 0
    total_batch = int(mnist.train.num_examples / batch_size)

    for i in range(total_batch):
        batch_xs, batch_ys = mnist.train.next_batch(batch_size)
        feed_dict = {X: batch_xs, Y: batch_ys}
        c, _ = sess.run([cost, optimizer], feed_dict=feed_dict)
        avg_cost += c / total_batch

    print('Epoch:', '%04d' % (epoch + 1), 'cost =', '{:.9f}'.format(avg_cost))

print('Learning Finished!')

# Test model and check accuracy
correct_prediction = tf.equal(tf.argmax(hypothesis, 1), tf.argmax(Y, 1))
accuracy = tf.reduce_mean(tf.cast(correct_prediction, tf.float32))
print('Accuracy:', sess.run(accuracy, feed_dict={
    X: mnist.test.images, Y: mnist.test.labels}))

# Get one and predict
r = random.randint(0, mnist.test.num_examples - 1)
print("Label: ", sess.run(tf.argmax(mnist.test.labels[r:r + 1], 1)))
print("Prediction: ", sess.run(
    tf.argmax(hypothesis, 1), feed_dict={X: mnist.test.images[r:r + 1]}))

# plt.imshow(mnist.test.images[r:r + 1].
#             reshape(28, 28), cmap='Greys', interpolation='nearest')
# plt.show()

# MNIST NN Deep

```



```

# MNIST NN Deep
import tensorflow as tf
import random
# import matplotlib.pyplot as plt

from tensorflow.examples.tutorials.mnist import input_data

tf.set_random_seed(777) # reproducibility

mnist = input_data.read_data_sets("MNIST_data/", one_hot=True)
# Check out https://www.tensorflow.org/get_started/mnist/beginners for
# more information about the mnist dataset

# parameters
learning_rate = 0.001
training_epochs = 15
batch_size = 100

# input place holders
X = tf.placeholder(tf.float32, [None, 784])
Y = tf.placeholder(tf.float32, [None, 10])

# weights & bias for nn layers
# http://stackoverflow.com/questions/33640581/how-to-do-xavier-initialization-on-tensorflow
W1 = tf.get_variable("W1", shape=[784, 512],
                    initializer=tf.contrib.layers.xavier_initializer())
b1 = tf.Variable(tf.random_normal([512]))
L1 = tf.nn.relu(tf.matmul(X, W1) + b1)

W2 = tf.get_variable("W2", shape=[512, 512],
                    initializer=tf.contrib.layers.xavier_initializer())
b2 = tf.Variable(tf.random_normal([512]))
L2 = tf.nn.relu(tf.matmul(L1, W2) + b2)

W3 = tf.get_variable("W3", shape=[512, 512],
                    initializer=tf.contrib.layers.xavier_initializer())
b3 = tf.Variable(tf.random_normal([512]))
L3 = tf.nn.relu(tf.matmul(L2, W3) + b3)

W4 = tf.get_variable("W4", shape=[512, 512],
                    initializer=tf.contrib.layers.xavier_initializer())
b4 = tf.Variable(tf.random_normal([512]))
L4 = tf.nn.relu(tf.matmul(L3, W4) + b4)

W5 = tf.get_variable("W5", shape=[512, 10],
                    initializer=tf.contrib.layers.xavier_initializer())
b5 = tf.Variable(tf.random_normal([10]))
hypothesis = tf.matmul(L4, W5) + b5

```

```

L4 = tf.nn.dropout(L4, keep_prob=keep_prob)

W5 = tf.get_variable("W5", shape=[512, 10],
                    initializer=tf.contrib.layers.xavier_initializer())
b5 = tf.Variable(tf.random_normal([10]))
hypothesis = tf.matmul(L4, W5) + b5

# define cost/loss & optimizer
cost = tf.reduce_mean(tf.nn.softmax_cross_entropy_with_logits(
    logits=hypothesis, labels=Y))
optimizer = tf.train.AdamOptimizer(learning_rate=learning_rate).minimize(cost)

# initialize
sess = tf.Session()
sess.run(tf.global_variables_initializer())

# train my model
for epoch in range(training_epochs):
    avg_cost = 0
    total_batch = int(mnist.train.num_examples / batch_size)

    for i in range(total_batch):
        batch_xs, batch_ys = mnist.train.next_batch(batch_size)
        feed_dict = {X: batch_xs, Y: batch_ys, keep_prob: 0.7}
        c, _ = sess.run([cost, optimizer], feed_dict=feed_dict)
        avg_cost += c / total_batch

    print('Epoch:', '%04d' % (epoch + 1), 'cost =', '{:.9f}'.format(avg_cost))

print('Learning Finished!')

# Test model and check accuracy
correct_prediction = tf.equal(tf.argmax(hypothesis, 1), tf.argmax(Y, 1))
accuracy = tf.reduce_mean(tf.cast(correct_prediction, tf.float32))
print('Accuracy:', sess.run(accuracy, feed_dict={
    X: mnist.test.images, Y: mnist.test.labels, keep_prob: 1}))

# Get one and predict
r = random.randint(0, mnist.test.num_examples - 1)
print("Label: ", sess.run(tf.argmax(mnist.test.labels[r:r + 1], 1)))
print("Prediction: ", sess.run(
    tf.argmax(hypothesis, 1), feed_dict={X: mnist.test.images[r:r + 1], keep_prob: 1}))

# plt.imshow(mnist.test.images[r:r + 1].
#            reshape(28, 28), cmap='Greys', interpolation='nearest')
# plt.show()

```

Epoch: 0001 cost = 0.447322626
Epoch: 0002 cost = 0.157285590
Epoch: 0003 cost = 0.121884535
Epoch: 0004 cost = 0.098128681
Epoch: 0005 cost = 0.082901778
Epoch: 0006 cost = 0.075337573
Epoch: 0007 cost = 0.069752543
Epoch: 0008 cost = 0.060884363
Epoch: 0009 cost = 0.055276413
Epoch: 0010 cost = 0.054631256
Epoch: 0011 cost = 0.049675195
Epoch: 0012 cost = 0.049125314
Epoch: 0013 cost = 0.047231930
Epoch: 0014 cost = 0.041290121
Epoch: 0015 cost = 0.043621063
Learning Finished!

Accuracy: **0.9804!!**

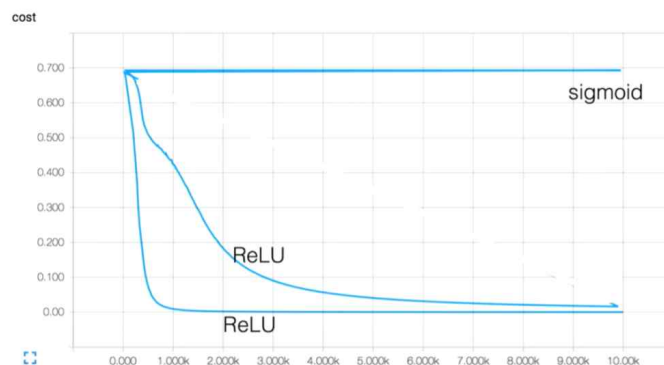
딥러닝(5)

weight를 어떻게 초기화 할 것인지에 대해 알아보자.

• Geoffrey Hinton's summary of findings up to today

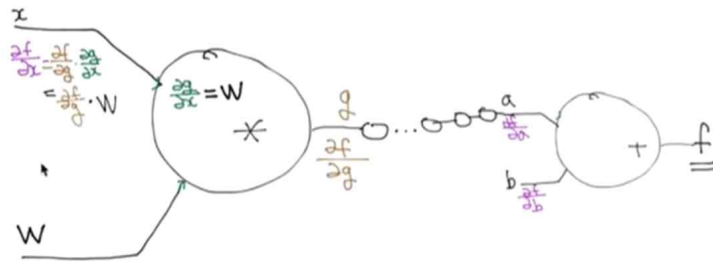
✓We initialized the weights in a stupid way.: 우리가 멍청하게 weight 값을 초기화 했구나~

지금까지 실습하는 중 랜덤하게 weight 값을 초기화 하였음.



-> 시그모이드 함수는 당연히 아니겠지만 ReLU함수 같은 경우에는 초기에 어떻게 값을 지정하느냐에 따라 많이 달라짐. 많이 돌리는 것 자체가 돈이 드는 행위, 추가 시간 또한 요구됨. 때문에 weight를 잘 초기화하는 것이 좋음.

그렇다면 깔끔하게 0으로 준다면?



sigmoid 함수를 쓰면 기울기가 사라지기 때문에 쓰면 안됨. 같은 논리로 0은 절대로 사용해서는 안된다!

- Need to set the initial weight values wisely
- ✓Not all 0's
- ✓Challenging issue
- ✓Hinton et al. (2006) "A Fast Learning Algorithm for Deep Belief Nets"
- » Restricted Boltzmann Machine (RBM) -> weight 값을 rbm을 이용해서 멍청하게 weight 값 초기화 하지말고 잘 초기화해보자!

forward 방식과 backward 방식을 이용하여 값을 조정. -> 값이 평균적으로 비슷해질 때까지 forward와 backward 반복.

How can we use RBM to initialize weights?

- ✓Apply the RBM idea on adjacent two layers as a pre-training step
- ✓Continue the first process to all layers
- ✓This will set weights
- ✓Example : Deep Belief Network - Weight initialized by RBM

pre-training: 두 개씩만 보겠다! 왔다갔다 계속 반복 온전히 완료가 되면 다음 단계로 넘어가서 조정 반복. 그럼 최종적으로 weight 값이 맞춰짐.

fine tuning이라고 불리기도 함.

=> tensorflow에서 잘 코딩이 되어있음.

그러나, 요즘에는 RBM을 잘 사용하지 않음. 더 좋은 방법들이 생겨났음.

Good news

- ✓No need to use complicated RBM for weight initializations
- ✓Simple methods are OK
- » Xavier initialization: X. Glorot and Y. Bengio, "Understanding the difficulty of training deep feedforward neural networks," in International conference on artificial intelligence and statistics, 2010
- » He's initialization: K. He, X. Zhang, S. Ren, and J. Sun, "Delving Deep into Rectifiers: Surpassing

-> ✓Using number of input (fan_in) and output (fan_out) -> /2 한게 더
좋다라는 주장 -> 이후 계속해서 다양한 초기화 방법들이 등장하였음.

=====

딥러닝(6)

마지막, 딥러닝 수업..

드롭아웃에 대해 설명하기 전에 Overfitting에 대해 다시 알아보자.

-> 학습을 너무 많이 하게 되면 실제와 잘 맞지 않게 된다.

- Am I overfitting?

- ✓Very high accuracy on the training dataset (eg: 0.99)

- ✓ Poor accuracy on the test data set (0.85)

ex) 교수님이 나올거라고 한 문제만 엄청 풀면? -> 그 문제에만 최적화 되어있음.

즉, 트레이닝을 많이 하면 할 수 록 에러율이 증가하는 원리와 같음.

overfitting 방지 방법:

- Solutions for overfitting -> training data가 많아지면 많아질 수 록 일어날
확률 감소.

- ✓More training data!

- ✓Reduce the number of features

- ✓Regularization

또 다른 방법.

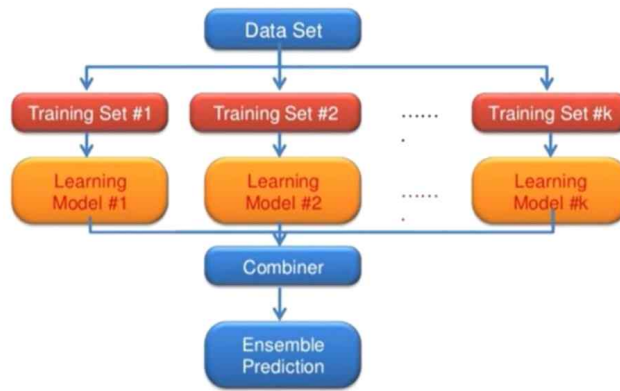
- Dropout : A Simple Way to Prevent Neural Networks from Overfitting
[Srivastava et al. 2014]

노드들을 전문가라고 할 때 잠깐 쉬어, 넌 일해 하는 식으로 끊어버리겠다! ->
dropout의 의미.

Ensemble은 무엇인가?

데이터가 적으면 앙상블을 이용하진 않지만, 음성인식을 한다던지와 같은 엄청난
양의 데이터를 이용한다면 앙상블 기법을 통해 나눠서 돌려보면 적게는 2%에서
많게는 4~5%정도까지 증가. -> 엄청난 것.

Feedforward neural network: 레고를 쌓는 것과 비슷.



- Fast forward
- Split & merge -> 병합을 이용, CNN 구조가 split & merge 방식
- Recurrent network -> 횡으로 옆으로 펼쳐는 것. RNN의 네트워크 기본 기법

결론: 네트워크를 어떻게 구성하느냐에 따라 결과값들은 엄청나게 달라질 수 있다.

네트워크를 잘 구성해서 더 좋은 알고리즘을 만들 수 있다.

“The only limit is your imagination”

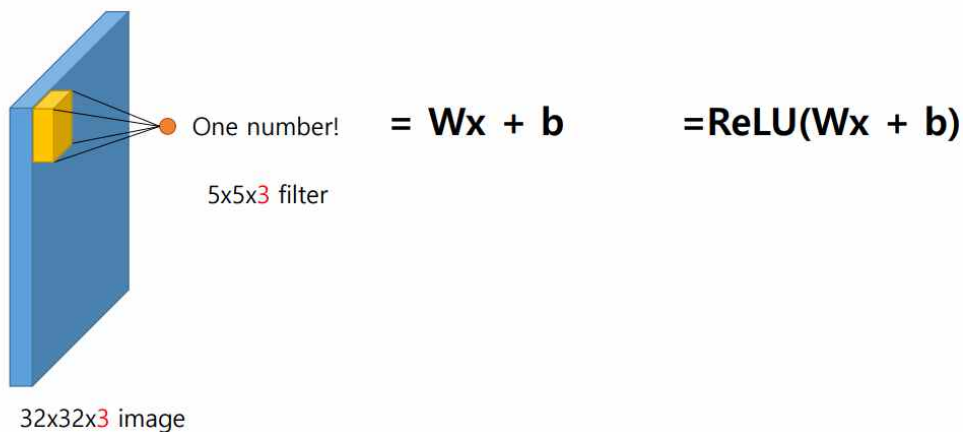
CNN(1)

ConvNet의 Conv 레이어 만들기

- 고양이 실험에서 시작
- 일정하게 잘라진 자료를 가지고 분석

Start with an image (width x height x depth)

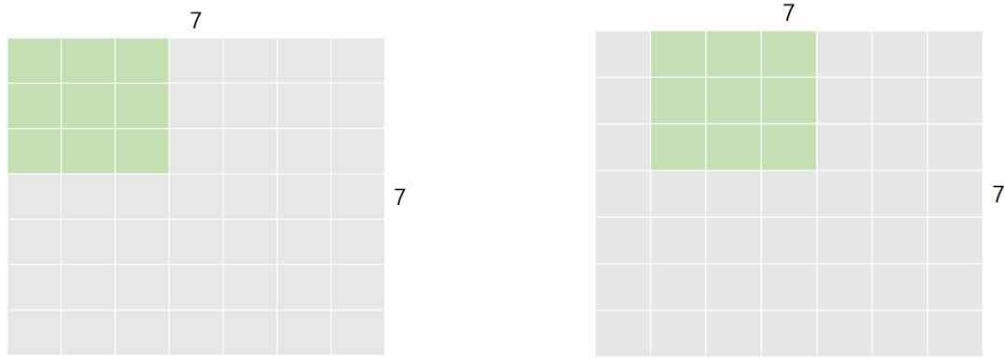
- Get one number using the filter



이것을 이용하여 하나의 숫자로 나타낼 수 있다. ReLU 함수도 사용하고 싶으면 사용가능 -> 이러한 과정을 반복하며 순차적으로 지역 전역을 훑는다.

-> TF가 다 해주지 않는다. 계산 과정 알아야함!(수학이 아닌 산수의 개념정도)

stride = 1 or 2 등으로 한, 두 칸 정도씩 이동하는 과정을 거친다.



위와 같은 방식으로 진행한다.

Output size :

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

e.g. $N = 7, F = 3$:

stride 1 $\rightarrow (7 - 3)/1 + 1 = 5$

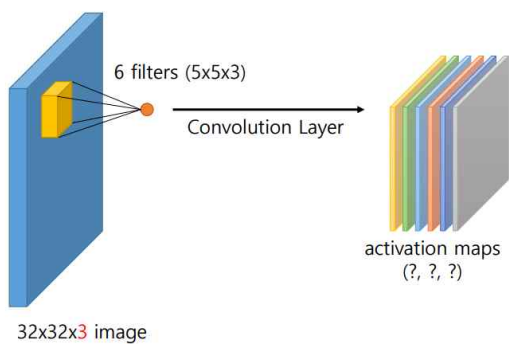
stride 2 $\rightarrow (7 - 3)/2 + 1 = 3$

stride 3 $\rightarrow (7 - 3)/3 + 1 = 2.33 \therefore$ 마지막 필터가 소수가 나오도록 하면 안됨.
정수로 떨어지게 계산을 해야한다.

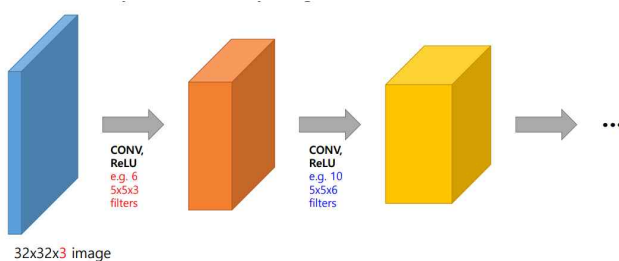
zero pad의 개념을 이용하여 크기가 변하지 않도록 사용을 한다.

input 7x7, 7x7 output! \Rightarrow input과 output의 값이 동일할 수 있도록 한다.

- Swiping the entire image



여러 필터를 이용하여 쌓는다.

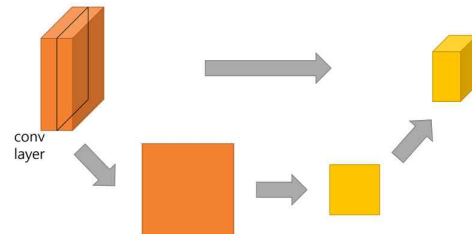


weight 값의 초기화는 지난 시간에 배웠던 방법을 이용하여 초기화 한다.

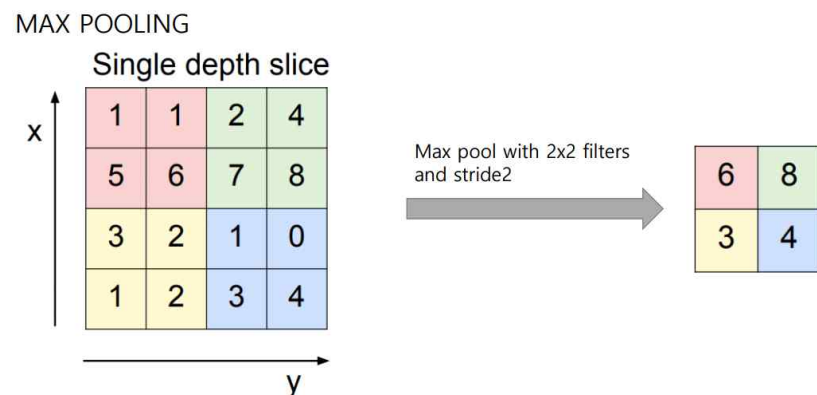
=====

CNN(2)

- Pooling layer (sampling)



Pooling의 개념은 sampling과 같다. 그러나 pooling은 하나의 layer만 뽑아서 보겠다라는 의미!



4 * 4 가 2 * 2 로 바뀌게 된다!

- Case Study: LeNet-5 =? 32 * 32의 사이즈
 - Case Study: AlexNet => 조금 더 복잡 227*227*3
- 즉 쌓는 방법은 다양하게 존재.

- Case Study: GoogLeNet

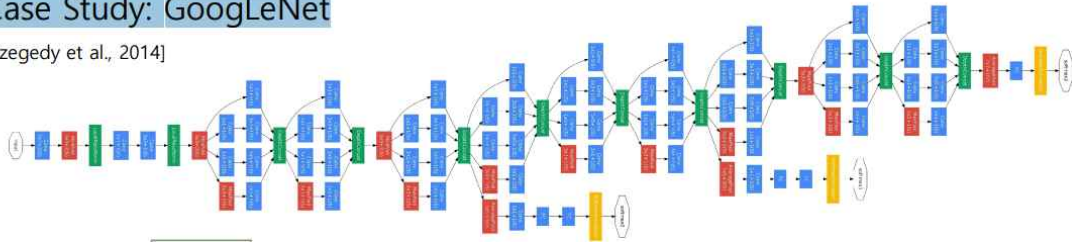
왜 이렇게 까지 쌓아서 인식률이 좋아졌는 지에 대한 명확한 답은 아직 알 수 가 없다.

- Case Study: ResNet

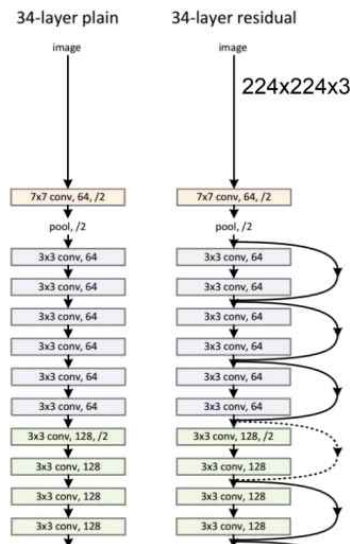
ILSVRC 2015 winner (3.6% top 5 error) ==> 엄청난 방법! 더 이상의 획기적인 방법은 안나올 거라 예상.

• Case Study: GoogLeNet

[Szegedy et al., 2014]



왜 이리 인식률이 좋을 까?



하나하나 따지지 않고 스킵하는 과정을 거침

어떻게 네트워크를 쌓느냐에 따라 무궁무진하다. 알파고도 CNN을 이용한 것!

=====

RNN

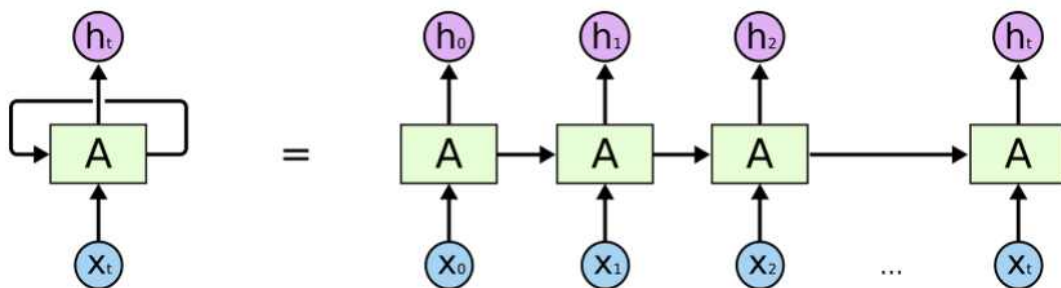
-> neural network의 꽃

• Sequence data을 이용한다. 순서에 따라 데이터가 입력되고 해야지 파악가능!

✓We don't understand one word only

✓We understand based on the previous words + this word. (time series)

✓NN/CNN cannot do this

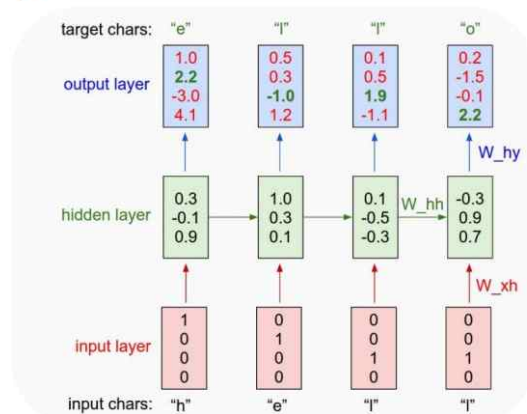


다시 되풀이 = Recurrent

- Character-level language model example

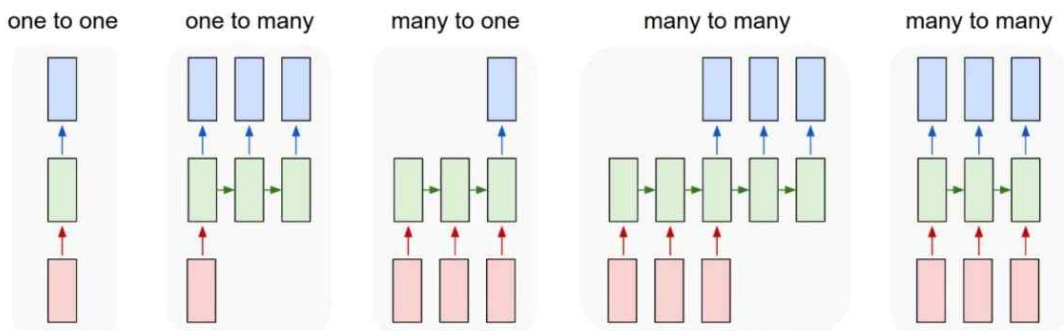
Vocabulary:
[h,e,l,o]

Example training
sequence:
"hello"



- RNN applications 사용되는 곳.
 - ✓Language Modeling
 - ✓Speech Recognition
 - ✓Machine Translation
 - ✓Conversation Modeling/Question Answering
 - ✓Image/Video Captioning
 - ✓Image/Music/Dance Generation

- Recurrent Networks offer a lot of flexibility:



모델을 다양하게 함에 따라 적용 분야가 다르다!

- Training RNNs is challenging
 - ✓Several advanced models
 - » Long Short Term Memory (LSTM) -> 한국인이 소개
 - » GRU by Cho et al. 2014

```

In [4]: # MNIST and Convolutional Neural Network
import tensorflow as tf
import random
# import matplotlib.pyplot as plt

from tensorflow.examples.tutorials.mnist import input_data

tf.set_random_seed(777) # reproducibility

mnist = input_data.read_data_sets("MNIST_data/", one_hot=True)

# hyper parameters
learning_rate = 0.001
training_epochs = 15
batch_size = 100

# input place holders
X = tf.placeholder(tf.float32, [None, 784])
X_img = tf.reshape(X, [-1, 28, 28, 1]) # img 28x28x1 (black/white)
Y = tf.placeholder(tf.float32, [None, 10])

# L1 imgIn shape=(?, 28, 28, 1)
W1 = tf.Variable(tf.random_normal([3, 3, 1, 32], stddev=0.01))
# Conv -> (?, 28, 28, 32)
# Pool -> (?, 14, 14, 32)
L1 = tf.nn.conv2d(X_img, W1, strides=[1, 1, 1, 1], padding='SAME')
L1 = tf.nn.relu(L1)
L1 = tf.nn.max_pool(L1, ksize=[1, 2, 2, 1],
                    strides=[1, 2, 2, 1], padding='SAME')
...
Tensor("Conv2D:0", shape=(?, 28, 28, 32), dtype=float32)
Tensor("Relu:0", shape=(?, 28, 28, 32), dtype=float32)
Tensor("MaxPool:0", shape=(?, 14, 14, 32), dtype=float32)
...

# L2 imgIn shape=(?, 14, 14, 32)
W2 = tf.Variable(tf.random_normal([3, 3, 32, 64], stddev=0.01))
# Conv -> (?, 14, 14, 64)
# Pool -> (?, 7, 7, 64)
L2 = tf.nn.conv2d(L1, W2, strides=[1, 1, 1, 1], padding='SAME')
L2 = tf.nn.relu(L2)
L2 = tf.nn.max_pool(L2, ksize=[1, 2, 2, 1],
                    strides=[1, 2, 2, 1], padding='SAME')
L2_flat = tf.reshape(L2, [-1, 7 * 7 * 64])

# Final FC 7x7x64 inputs -> 10 outputs
W3 = tf.get_variable("W3", shape=[7 * 7 * 64, 10],
                    initializer=tf.contrib.layers.xavier_initializer())
b = tf.Variable(tf.random_normal([10]))
logits = tf.matmul(L2_flat, W3) + b

# define cost/loss & optimizer
cost = tf.reduce_mean(tf.nn.softmax_cross_entropy_with_logits(
    logits=logits, labels=Y))
optimizer = tf.train.AdamOptimizer(learning_rate=learning_rate).minimize(cost)

# initialize
sess = tf.Session()
sess.run(tf.global_variables_initializer())

# train my model
print('Learning started. It takes sometime.')
for epoch in range(training_epochs):
    avg_cost = 0
    total_batch = int(mnist.train.num_examples / batch_size)

    for i in range(total_batch):
        batch_xs, batch_ys = mnist.train.next_batch(batch_size)
        feed_dict = {X: batch_xs, Y: batch_ys}
        c, _ = sess.run([cost, optimizer], feed_dict=feed_dict)
        avg_cost += c / total_batch

    print('Epoch:', '%04d' % (epoch + 1), 'cost =', '{:.9f}'.format(avg_cost))

print('Learning Finished!')

# Test model and check accuracy
correct_prediction = tf.equal(tf.argmax(logits, 1), tf.argmax(Y, 1))
accuracy = tf.reduce_mean(tf.cast(correct_prediction, tf.float32))
print('Accuracy:', sess.run(accuracy, feed_dict={
    X: mnist.test.images, Y: mnist.test.labels}))

# Get one and predict
r = random.randint(0, mnist.test.num_examples - 1)
print("Label: ", sess.run(tf.argmax(mnist.test.labels[r:r + 1], 1)))
print("Prediction: ", sess.run(
    tf.argmax(logits, 1), feed_dict={X: mnist.test.images[r:r + 1]}))

# plt.imshow(mnist.test.images[r:r + 1].
#             reshape(28, 28), cmap='Greys', interpolation='nearest')
# plt.show()

```



```

# MNIST and Deep learning CNN
import tensorflow as tf
import random
# import matplotlib.pyplot as plt

from tensorflow.examples.tutorials.mnist import input_data

tf.set_random_seed(777) # reproducibility

mnist = input_data.read_data_sets("MNIST_data/", one_hot=True)

# hyper parameters
learning_rate = 0.001
training_epochs = 15
batch_size = 100

# dropout (keep_prob) rate 0.7~0.5 on training, but should be 1 for testing
keep_prob = tf.placeholder(tf.float32)

# input place holders
X = tf.placeholder(tf.float32, [None, 784])
X_img = tf.reshape(X, [-1, 28, 28, 1]) # img 28x28x1 (black/white)
Y = tf.placeholder(tf.float32, [None, 10])

# L1 imgIn shape=(?, 28, 28, 1)
W1 = tf.Variable(tf.random_normal([3, 3, 1, 32], stddev=0.01))
# Conv -> (?, 28, 28, 32)
# Pool -> (?, 14, 14, 32)
L1 = tf.nn.conv2d(X_img, W1, strides=[1, 1, 1, 1], padding='SAME')
L1 = tf.nn.relu(L1)
L1 = tf.nn.max_pool(L1, ksize=[1, 2, 2, 1],
                    strides=[1, 2, 2, 1], padding='SAME')
L1 = tf.nn.dropout(L1, keep_prob=keep_prob)

```

```

# L2 imgIn shape=(?, 14, 14, 32)
W2 = tf.Variable(tf.random_normal([3, 3, 32, 64], stddev=0.01))
# Conv -> (?, 14, 14, 64)
# Pool -> (?, 7, 7, 64)
L2 = tf.nn.conv2d(L1, W2, strides=[1, 1, 1, 1], padding='SAME')
L2 = tf.nn.relu(L2)
L2 = tf.nn.max_pool(L2, ksize=[1, 2, 2, 1],
                    strides=[1, 2, 2, 1], padding='SAME')
L2 = tf.nn.dropout(L2, keep_prob=keep_prob)

```

```

# L3 imgIn shape=(?, 7, 7, 64)
W3 = tf.Variable(tf.random_normal([3, 3, 64, 128], stddev=0.01))
# Conv -> (?, 7, 7, 128)
# Pool -> (?, 4, 4, 128)
# Reshape -> (?, 4 * 4 * 128) # Flatten them for FC
L3 = tf.nn.conv2d(L2, W3, strides=[1, 1, 1, 1], padding='SAME')
L3 = tf.nn.relu(L3)
L3 = tf.nn.max_pool(L3, ksize=[1, 2, 2, 1], strides=[
    1, 2, 2, 1], padding='SAME')
L3 = tf.nn.dropout(L3, keep_prob=keep_prob)
L3_flat = tf.reshape(L3, [-1, 128 * 4 * 4])

```

```

# L4 FC 4x4x128 inputs -> 625 outputs
W4 = tf.get_variable("W4", shape=[128 * 4 * 4, 625],
                    initializer=tf.contrib.layers.xavier_initializer())
b4 = tf.Variable(tf.random_normal([625]))
L4 = tf.nn.relu(tf.matmul(L3_flat, W4) + b4)
L4 = tf.nn.dropout(L4, keep_prob=keep_prob)

```

```

# L5 Final FC 625 inputs -> 10 outputs
W5 = tf.get_variable("W5", shape=[625, 10],
                    initializer=tf.contrib.layers.xavier_initializer())
b5 = tf.Variable(tf.random_normal([10]))
logits = tf.matmul(L4, W5) + b5

```



```

# define cost/loss & optimizer
cost = tf.reduce_mean(tf.nn.softmax_cross_entropy_with_logits(
    logits=logits, labels=Y))
optimizer = tf.train.AdamOptimizer(learning_rate=learning_rate).minimize(cost)

# initialize
sess = tf.Session()
sess.run(tf.global_variables_initializer())

# train my model
print('Learning started. It takes sometime.')
for epoch in range(training_epochs):
    avg_cost = 0
    total_batch = int(mnist.train.num_examples / batch_size)

    for i in range(total_batch):
        batch_xs, batch_ys = mnist.train.next_batch(batch_size)
        feed_dict = {X: batch_xs, Y: batch_ys, keep_prob: 0.7}
        c, _ = sess.run([cost, optimizer], feed_dict=feed_dict)
        avg_cost += c / total_batch

    print('Epoch:', '%04d' % (epoch + 1), 'cost =', '{:.9f}'.format(avg_cost))

print('Learning Finished!')

# Test model and check accuracy

# if you have a OOM error, please refer to lab-11-X-mnist_deep_cnn_low_memory.py

correct_prediction = tf.equal(tf.argmax(logits, 1), tf.argmax(Y, 1))
accuracy = tf.reduce_mean(tf.cast(correct_prediction, tf.float32))
print('Accuracy:', sess.run(accuracy, feed_dict={
    X: mnist.test.images, Y: mnist.test.labels, keep_prob: 1}))

# Get one and predict
r = random.randint(0, mnist.test.num_examples - 1)
print("Label: ", sess.run(tf.argmax(mnist.test.labels[r:r + 1], 1)))
print("Prediction: ", sess.run(
    tf.argmax(logits, 1), feed_dict={X: mnist.test.images[r:r + 1], keep_prob: 1}))

# plt.imshow(mnist.test.images[r:r + 1].
#             reshape(28, 28), cmap='Greys', interpolation='nearest')
# plt.show()

```

```

Learning started. It takes sometime.
Epoch: 0001 cost = 0.385748474
Epoch: 0002 cost = 0.092017397
Epoch: 0003 cost = 0.065854684
Epoch: 0004 cost = 0.055604566
Epoch: 0005 cost = 0.045996377
Epoch: 0006 cost = 0.040913645
Epoch: 0007 cost = 0.036924479
Epoch: 0008 cost = 0.032808939
Epoch: 0009 cost = 0.031791007
Epoch: 0010 cost = 0.030224456
Epoch: 0011 cost = 0.026849916
Epoch: 0012 cost = 0.026826763
Epoch: 0013 cost = 0.027188021
Epoch: 0014 cost = 0.023604777
Epoch: 0015 cost = 0.024607201
Learning Finished!
Accuracy: 0.9938

```