**Lecture 0**

학습 목표

* ML algorithm에 대한 기본적인 이해
  + Deep learning을 이해하기 위한 기본적인 개념들: Linear regression(선형 회귀), logistic regression(로지스틱 회귀)
  + Deep learning: neural networks, convolutional neural network, recurrent neural network
* ML tool을 이용한 문제 해결
  + Tensorflow와 Python

**Lecture 1 – machine Learning Basics**

Machine Learning(ML)

* 소프트웨어의 일종이라고 할 수 있다.
* Limitations of explicit programming (명시적 프로그래밍의 한계점)
  + Spam filter: many rules
  + Automatic driving: too many rules
  + 직접 개발자가 지정해 줘야 하는 것들이 많다.
* Machine learning: 개발자가 일일이 정하지 않고 프로그램 자체가 데이터를 보고 학습하는 프로그램

Supervised / Unsupervised learning

* 학습 방법에 따라서 달라진다.
* Supervised learning
  + 하나의 정해진 데이터(labeled examples – training set)를 사용해 학습하는 것
  + Labeled = 정답이 정해져 있는 이라고 생각하면 편하다.
  + 예시: 개/고양이 그림을 보고 구별하는 프로그램에게 개 혹은 고양이라고 label이 주어진 data를 제공해 학습을 시킨다.
* Unsupervised learning
  + Un-labeled data를 사용
  + 예시
    - google news grouping: 스스로 유사한 news를 grouping을 진행한다.
    - Word clustering: 단어 중 비슷한 단어를 labeling이 진행되지 않은 data를 보고 스스로 학습해 알아내는 것

Supervised learning

* Most common problem type in ML
  + Image labeling: learning from tagged images
  + Email spam filter: learning from labeled email
  + Predicting exam score: learning from previous exam score and time spent

Training data set

* Supervised learning model에서 사용하는 경우, labeling이 이미 진행되어 있는 상황이다.
* 어떤 표가 존재해서 x에는 data가, y에는 해당 data의 label이 지정되어 있다고 생각하다. 이 data를 ML model에 넣어 학습을 시킨 뒤 xtest값을 학습이 진행된 model에 넣으면 해당 model이 자기가 생각하는(계산한) label을 반환하게 된다.
* 이 때 학습에 사용되는 data가 training data set이다.
* AlphaGo 역시 바둑 기보(training dataset)로 학습을 진행하고 특정 수에 대해 값을 반환하기 때문에 supervised learning의 한 예시라 할 수 있다.

Types of supervised learning

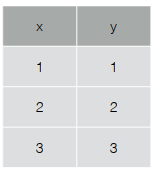
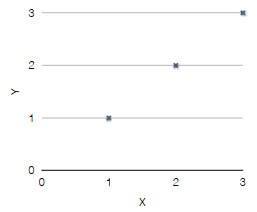
* Predicting final exam score based on time spent for studying
  + **Regression**: 시험 점수의 범위가 넓다(0~100점)
  + 10시간 – 90점 / 9시간 – 80점 / 3시간 – 50점 / 2시간 – 30점일 때, 7시간 공부했다면 몇 점을 받을 수 있을까?
* Pass/non-pass based on time spent for studying
  + **Binary classification**: 둘 중의 한 경우만을 예상
* Letter grade (A~F) based on time spent for studying
  + **Multi-label classification**: 3개 이상의 label에 대해 classification을 진행

**Lecture 2 – Linear Regression**

Predicting exam score: regression

* Supervised learning
* 공부를 진행한 시간과 시험 점수를 training data로 입력받아 학습을 진행한다.
* 결과적으로, 공부한 시간을 입력받아 시험 점수를 예측하는 model을 만들게 된다.

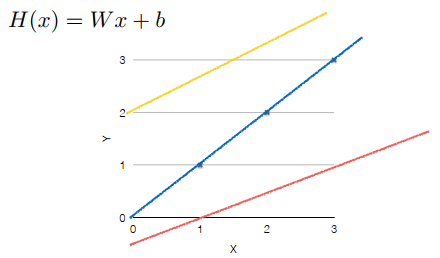
Regression

* X는 예측을 위한 자료(feature)라 한다.

Linear hypothesis

* 세상의 많은 data들이 실제로 linear한 경향을 따르게 된다.
* 입력 받은 데이터를 만족하는 직선을 찾아내는 과정을 regression이라 한다.



* 위처럼 H(x) = Wx + b 라는 직선 형태를 가질 것으로 예상한다.

Cost function(Loss function)

* 어떤 가설(직선)이 더 좋을지를 판단하는 함수이다.
* 예상(가설)과 실제 값이 얼마나 차이가 나는지를 판단한다.
* Linear regression의 경우, 직선과 실제 데이터(점) 사이의 거리를 이용하게 된다.
* H(x) – y
  + 이 경우는 차이가 양 또는 음이 모두 나오기 때문에 적합한 방법은 아니다.
* (H(x) – y)2
  + 이 경우가 일반적으로 사용된다. 항상 양수이며 값이 클수록 더 잘 드러나게 된다.
* , 여기서 m은 학습에 사용된 데이터의 수.
* 우리의 목표는 이 cost를 minimize하는 것이다.

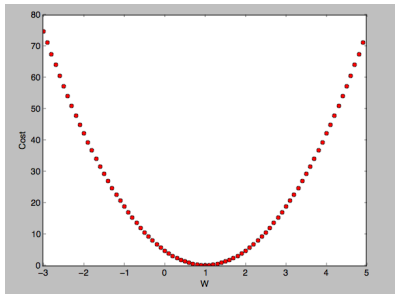
**Lecture 3 – How to minimize cost**

Hypothesis and Cost

* + Simplified hypothesis:

What cost(W) looks like for simplified hypothesis?

* Data: x = 1, y = 1 / x = 2, y = 2 / x = 3, y = 3
* W = 1, cost(W) = 0
* W = 0, cost(W) = 4.67
* W = 2, cost(W) = 4.67
* W에 대한 cost의 그래프를 그리게 되면 다음과 같다.



Gradient decent algorithm

* 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 (w1, w2, …)
* Cost가 감소하는 방향(gradient의 방향)으로 W와 b를 변경한다.

How it works

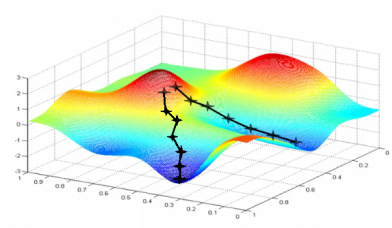
* Start with initial guesses
  + 예를 들어, (0,0)에서 시작한다 하자.
  + W와 b를 바꿔가며 cost(W, b)를 감소시키는 방향으로 조절한다.
* 매번 W와 b를 바꿀 때, cost(W, b)를 가장 많이 감소시킬 수 있는 gradient를 선택한다.
* Local minimum에 도달할 때까지 반복
* 어느 지점에서 시작하는지에 따라서 다른 local minimum에 도달하게 된다.

Formal definition

* 여기서 는 learning rate이다.
* 위의 방법을 적용해 W를 갱신해 나간다.

Convex function

* Convex function의 경우, 어느 지점에서 learning을 시작해도 항상 같은 local minimum에 도달하게 된다.
* 이는 convex function임을 확인할 수 있으면 위의 알고리즘을 사용 가능하다는 이야기이다.
* 함수에 따라서는 다음과 같이 유효하지 않을 수도 있다.

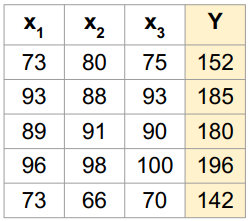


**Lecture 4 – Multivariable linear regression**

Hypothesis and Cost

Predicting exam score

* Regression using one input(x)
  + One variable – one feature
* Regression using three inputs(x1, x2, x3)
  + Multi-variable – one feature



Hypothesis & cost function for multiple inputs

Hypothesis using matrix

* 위의 hypothesis는 matrix를 이용한 표현이 가능하다.
* 이 때 위의 5개 instance에 대한 행렬을 만들면 다음과 같다.
  + [5, 3] \* [3, 1] = [5, 1]
* 여기서 행렬 X에 대해 5(nrow)는 instance의 수, 3(ncol)은 variable의 수이다.

**Lecture 5 – Logistic regression classification**

Regression(HCG)

* Hypothesis
* Cost
* Gradient decent

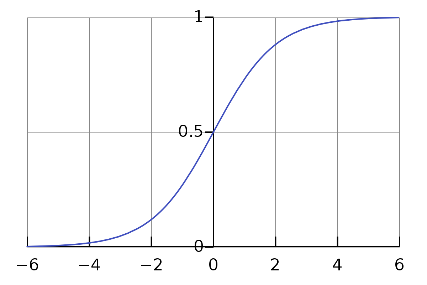
Classification

* Binary classification의 경우 둘 중 하나를 고르게 되는 classification이다.
* 예시는 spam detection(spam or ham), facebook feed(show or hide), credit card fradudlent transaction detection(legitimate or fraud)와 같이 둘 중 하나를 선택하는 알고리즘이다.
* 0, 1 encoding이 가능하다.
* radiology의 경우 tumor의 악성 여부 판단, 주식에서 사고 팔고의 결정 등을 한다.

Pass(1), fail(0) based on study hours

* 공부를 한 시간에 따라 합격 여부를 적은 데이터가 있다.
* Linear regression
  + Linear regression을 이용한다면, y가 0.5가 되는 지점을 기점으로 그 이하는 불합격, 그 이상은 합격이라고 판단할 것이다.
  + 만약 합격한 사람 중 공부를 한 시간이 아주 긴 사람이 있다면? Linear regression 시 기울기가 너무 낮아져 실제로 합격인 사람이 불합격인 것처럼 판단할 수 있다는 문제가 발생한다.
  + 우리는 y가 0 또는 1임을 알지만 1보다 크거나 0보다 작은 값을 반환할 수도 있다. 가령 1~10시간을 공부한 데이터로 학습을 시킨다면 이 범위 안에서는 0~1의 y값을 가진다. 그러다 50시간 공부한 데이터를 넣으면 1을 넘는 결과를 반환할 것이다.

Logistic hypothesis

* 기존의 linear regression의 hypothesis인 대신 다음의 가설을 사용한다.
* Hypothesis
  + 의 범위를 가지는 함수
* Logistic function(sigmoid function)
  + 
  + Z가 0이면 g(z) = 0.5, 양의 무한대로 발산 시 g(z)는 1에 수렴하며 음의 무한대로 발산 시 g(z)는 0에 수렴한다.
* Logistic hypothesis

Cost function

* Hypothesis
  + Linear:
  + Logistic:
* Logistic hypothesis의 경우, 위의 cost function을 사용하게 되면 여러 개의 local minimum이 발생하게 되어 부적합하다. 즉, cost function을 바꿔줘야 한다.

New cost function for logistic

* 실제 y = 1인 경우
  + H(x) = 1 -> cost(1) = 0
  + H(x) = 0 -> cost(0) = inf
* 실제 y = 0인 경우
  + H(x) = 0 -> cost(0) = 0
  + H(x) = 1 -> cost(1) = inf
* 즉, 틀리면 cost를 매우 크게, 맞으면 0으로 주는 방식이라고 생각할 수 있다.
* 로도 표현 가능하다.

Minimize cost – gradient decent algorithm

* 실제 코딩 시에는 TF에 이미 있는 GradientDecentAlgorithm을 사용한다.

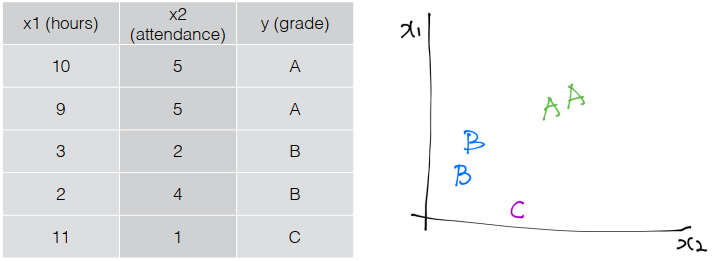
**Lecture 6 – Softmax classification: Multinomial classification**

Logistic regression

* X라는 입력이 W라는 weight에 대해 계산된다. 이 값을 z라 하고, z를 sigmoid function에 넣으면 의 값을 얻으며, 이 값은 0과 1 사이이다.



Multinomial classification

* 
* 위와 같이 3개 이상의 classification을 진행하는 것을 말한다.
* Binary classification의 여러 조합이라고 생각할 수 있다. 가령, A인지 아닌지/B인지 아닌지/C인지 아닌지 등의 조합이라고 생각할 수도 있다.
* 즉, 3개의 독립된 classifier를 가지면서 각각 A/B/C인지 여부를 판단하는 것이라고 생각할 수 있다.

Softmax

* 위의 식에서 인 출력값은 vector로 나오게 된다. 다만 이 경우 등의 값이 0과 1사이의 범위로 나오지 않게 된다. 즉 score로서 출력이 되는 것이고, 우리는 이 값을 probability로서 바꿀 것이다.
* 출력된 score vector를 softmax에 넣게 되면 probability를 얻게 된다. 이 값들의 특성은 각 값의 범위가 0~1이고, 합이 1이 된다. 가령 [0.7, 0.2, 0.1]과 같은 값을 얻게 된다. 즉, 정규화 과정을 진행하는 것이다.
* TF에서는 가장 큰 값을 1로서 생각해 [1,0,0]으로 생각하게 argmax라는 함수를 적용 가능하다. 이러한 방법을 one-hot encoding이라 한다.

Cost function: cross entropy

* 예측값: / 예: [0.7, 0.2, 0.1]T
* 실제값: / 예: [1, 0, 0]T
* L = [1, 0]이라 하자. 두 가지 예측값 =[1, 0]과 =[0,1]을 생각하자.
  + 전자의 경우는 cost가 0이 된다.
  + 후자는 cost가 양의 무한대로 발산하게 된다.

Logistic cost vs. cross entropy

* 위의 두 식은 실제로 같은 표현이다. (i=2인 경우)

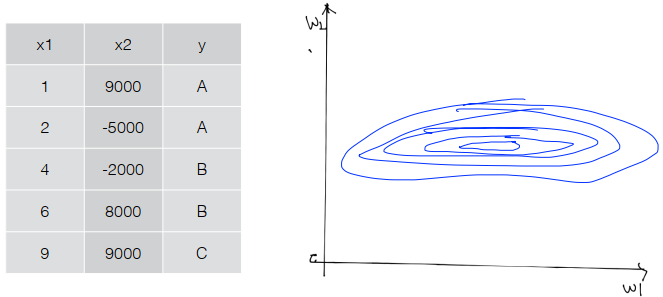
Gradient descent

**Lecture 7 - Application & Tips: Learning rate, data preprocessing, overfitting, learning & test data sets**

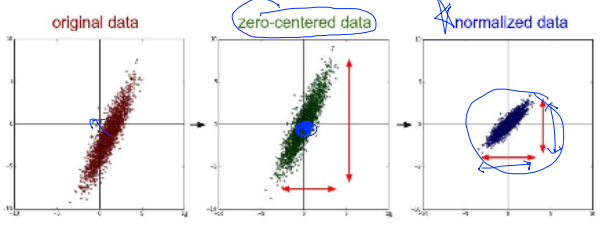
Gradient descent

* Gradient:
* 위 식에서 alpha 값을 learning rate라 정의하고, 기울기를 따라 내려가는 정도라고 생각할 수 있다.
* Learning rate가 너무 크다면, overshooting 현상이 일어날 수 있다. 이는 gradient를 따라 이동하는 거리가 너무 커 발산 진동을 해 cost가 오히려 커지는 현상을 말하는 것이다.
* Learning rate가 너무 작다면, 최저점에 도달하기까지 시간이 너무 오래 걸린다. 또한 local minimum에서 멈추는 잘못된 현상이 발생할 수 있다.
* 위와 같은 문제를 해결하기 위해서는 일반적으로 0.01로 시작해 여러 learning rate를 시도해 최적 값을 찾게 된다.

Data preprocessing for gradient descent



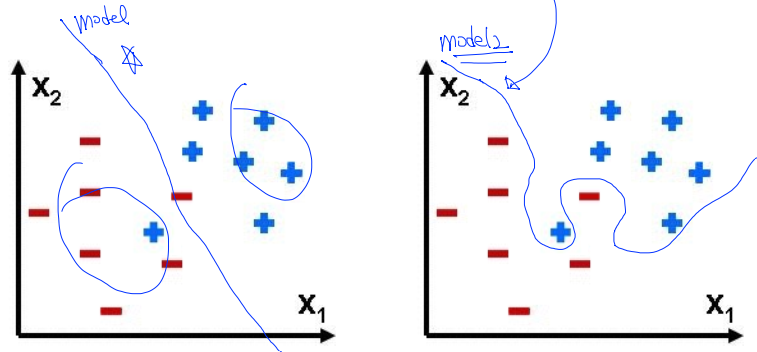
* 위의 예시처럼 x1과 x2 사이에 data scale의 차이가 많이 나는 경우를 생각해볼 수 있다.
* 우리가 원하는 이상적인 등고선은 타원이 아닌 원형에 가까운 등고선인데, 만약 위와 같은 타원형인 경우 앞서 언급한 overshooting 현상이 발생할 수 있다.



* 이러한 현상을 방지하기 위해서는 위와 같이 data의 zero-centering과 normalization같은 전처리를 진행하는 것이 좋다.
* Normalization은 변수의 범위를 0~1로 변형시키는 방법이다.
* Standardization은 특정 분포(정규분포 등)들의 mean, standard deviation 등을 이용해 해당 분포에서 속성값이 평균으로부터 얼마나 떨어져 있는지 확인하는 것이다. 일반적으로 mean = 0, SD = 1이 된다.
  + / X\_std[:,0] = (X[:,0] – X[:,0].mean()) / X[:,0].std()

Overfitting

* Overfitting이란 학습 모델에만 너무 잘 맞는 경우를 말한다. 만약 test data가 아닌 실제 데이터를 대입하게 되면 정확도가 떨어지는 경우가 발생한다.



* 위의 그림에서 오른쪽 model이 overfitting이 발생한 경우이다.

Solutions for overfitting

* Training data를 최대한 많이 확보한다.
* Number of feature를 감소시킨다.
* **Regularization**을 적용한다.

Regularization

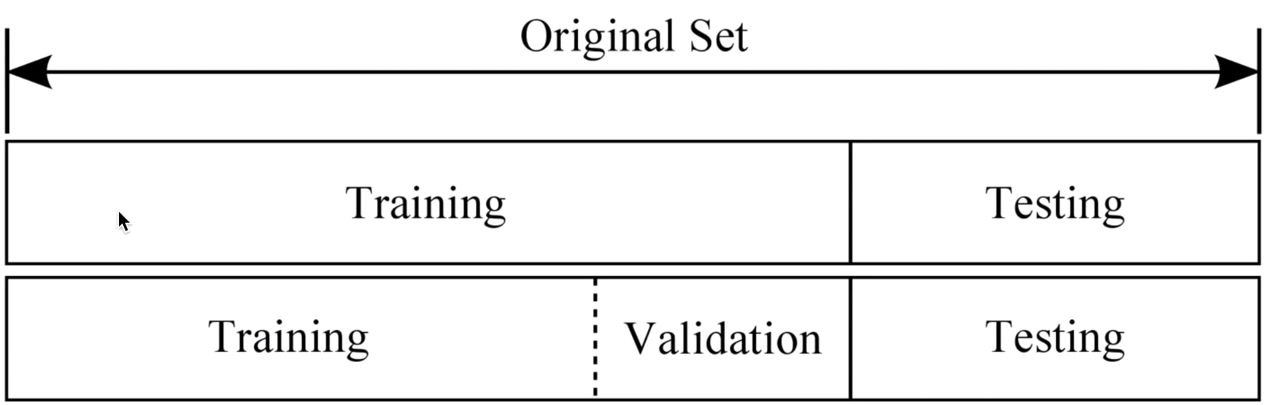
* Weight에 너무 큰 값을 부여하지 않는 것을 말한다.
* 또는, 학습된 model에서 많이 ‘구부러진’ 곡선을 좀 ‘펴 주는’ 방법이라고 직관적으로 생각할 수 있다.
* 기존의 우리 목표는 cost를 최소화하는 것이었지만 cost function에 다음과 같이 weight에 대한 항을 하나 추가하게 된다.
* 위의 식에서 는 regularization strength라고 하며, 이 값이 클수록 regularization이 강하게 이루어진다.
* TF에서는 에 해당하는 변수를 하나 만든 뒤 추가하면 된다.

Performance evaluation

* 우리가 학습을 시킨 model이 얼마나 잘 작동하는지 평가할 수 있을까?
* Training set을 이용해 evaluation을 진행하면?
  + 100% accuracy
  + Training set을 ‘기억’하기 때문에 굉장히 좋지 않은 방법이다.

Training and test sets

* Training set과 test set을 나눠 test set을 이용해 evaluation을 진행해야 한다.



* 위의 그림에서 training은 실제 학습이 이루어지는 dataset, validation은 learning rate와 regularization strength를 조절하기 위한 dataset이다. Testing은 실제로 평가를 진행하는 test set이다.

Online learning

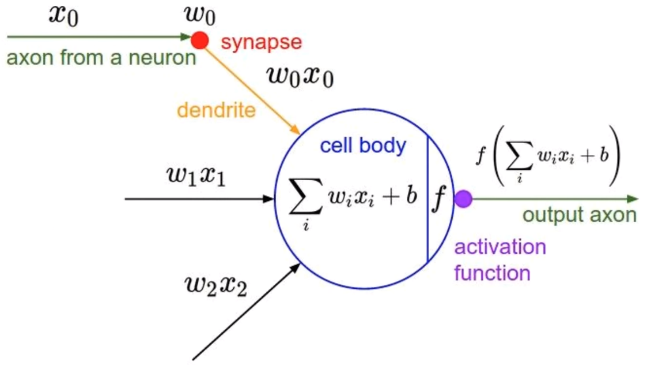
* 100만개의 dataset을 가지고 있다고 하자. 너무 많아서 10만개씩 10개로 쪼개 학습을 시키려 한다.
* 이 경우 online learning 방법을 사용하게 되면 처음 10만개를 learning한 정보가 남아있는 상태에서 다음의 10만개 data를 받아 학습을 시킨다. 즉, 누적 학습을 시킨다는 의미이다.
* 대표적으로 MINIST dataset이 있다. 글씨로 적은 숫자를 인식하는 것이다.

Accuracy

* 최근 이미지 판독과 같은 경우 정확도가 95% 이상을 보인다.

**Lecture 8 – Deep Neural Nets for Everyone**

Activation functions



* 실제 신경과 비슷한 형태의 model을 만든다.
* Cell body에서의 합이 특정 값을 넘으면 1이라는 신호를, 그렇지 않으면 0이라는 신호를 보내게 된다.

Simple AND/OR problem: linearly separable?

* 0과 1로 구성된 입력에 대해 or와 and 연산의 학습이 가능할까?
* 그 결과는 linear regression으로 간단히 구현 가능했다.

Simple XOR problem: linearly separable?

* XOR의 경우는 linear하게 separation이 불가능해 linear regression을 사용하는 경우 정확도가 떨어지게 된다.
* 수학적으로 linear regression을 이용하면 XOR 문제를 풀 수 없다는 것이 증명되었다.
* 이 XOR 문제를 풀기 위해서는 MLP(multilayer perceptions, multilayer neural nets)이 필요하다.

Backpropagation

* 기본적으로 forward하게 학습을 진행해 틀린 결과를 얻으면 다시 앞에서 weight와 bias를 조절해야 하는데, 대신 backward하게 조절해 나가는 것을 의미한다.

Convolutional Neural Networks

* 고양이에게 다양한 그림을 보여준 뒤 시신경의 활성도를 확인한다. 그 결과, 그림에 따라서 활성화되는 시신경의 종류가 달라졌다.
* 그래서 이 현상에 착안해 이미지의 경우 전체 이미지를 학습에 투입시키는 것이 아닌 일부분을 잘라서 학습에 투입시키게 된다.

A big problem

* Backpropagation은 10여개 이상의 많은 layer를 가지는 학습에 대해서는 잘 진행되지 않았다.
* 이 외에도 SVM, RandomForest와 같은 다른 ML 알고리즘이 나오게 되었다.

CIFAR(Canadian Institute For Advanced Research)

* 직접적인 적용이 없는 basic research를 장려했고, Hinton이 캐나다로 이주하는 데에 영향을 미쳤으며 이주 후에도 지원을 해 주었다.
* Neural net ML의 성장에 주된 기여를 한 단체이다.

Breakthrough – 2006/2007 ‘Deep’의 출현

* 2006, 2007년의 Hinton와 Bengio가 논문을 투고하게 된다.
* 많은 layer를 가지는 neural network는 처음 weight가 좋게 initialized 된 경우 잘 training이 진행될 수 있다. 즉, 초기값을 잘 주는 것이 중요하다. (Hinton, Simon Osindero, and Yee-Whye Teh, “A fast learning algorithm for deep belief nets”, 2006)
* 깊게 신경망을 구축하게 되면 더 어려운 문제를 풀 수 있다. (Yoshua Bengio et al., “Greedy Layer-Wise Training of Deep Networks”, 2007)
* 이러한 방법을 Deep Nets, Deep Learning으로 이름 붙이게 된다.

ImageNet Classification

* 주어진 image에 대해 해당 image 안에 어떤 object가 있는지 식별해 내는 것이다.
* 사람의 경우 5% 정도의 오답률을 보이며, 2010년대 초반에는 30%대의 오답률을 보였으나 2015년도에는 3%대까지 감소하게 된다.
* 뿐만 아니라, Vision Deep CNN을 적용한 뒤 Language Generating RNN을 적용하게 되면 해당 그림을 설명할 수 있기도 하다.

기타 응용 분야

* Deep API Learning
* Deep speech
  + 바이두에서 주변 소음이 많은 곳에서 사람의 말을 알아들을 수 있도록 학습한 모델
* DeepMind
  + 게임의 자동 실행
* AlphaGo
  + 바둑 인공지능
* YouTube 자동 자막
* Facebook 사용자 맞춤 피드
* Google의 사용자 최적 검색 시스템
* Netflix, Amazon의 맞춤 서비스

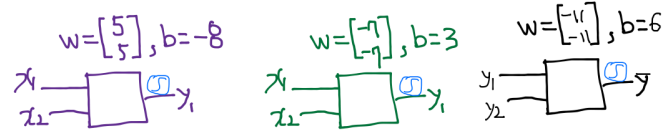
**Lecture 9 – Neural Nets(NN) for XOR**

XOR problem

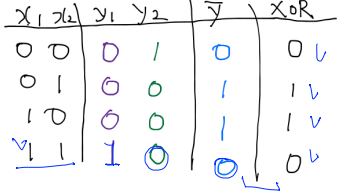
* 하나의 logistic regression으로는 XOR을 분리할 수 없다. 이는 수학적으로도 증명되었다.
* 만약 2개 이상을 사용하는 경우, 해결 가능하다.

XOR using NN

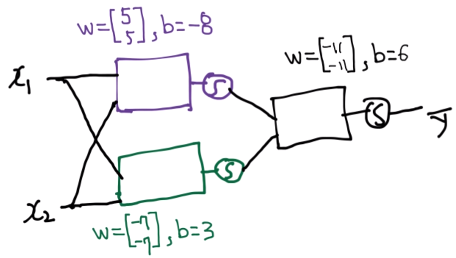
* Linear하게 구분하는 것은 불가능하다. 오른쪽 그림을 참고하자. 



* 위와 같은 두 개의 sigmoid function을 적용한 linear model을 합친 model이 있다고 생각하자. 위 경우 0과 1을 각각 입력한 진리표는 아래와 같아진다.

Sigmoid 함수 특성 상 0.5 이상은 1, 이하는 0으로 변환

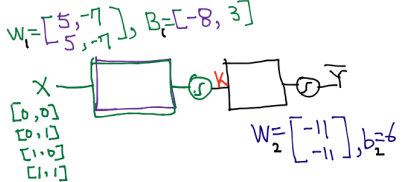
Forward propagation



* 동일한 과정을 위와 같이 표현 가능하다.
* 여기서 우리가 XOR이 가능하게 하는 다른 조합의 W와 b를 추가로 찾을 수 있을까? 또는 존재할까? 하는 의문을 가질 수 있다.

NN(Neural Network)

* 사실 위의 그래프는 multinomial classification처럼 weight와 bias를 합칠 수 있다.

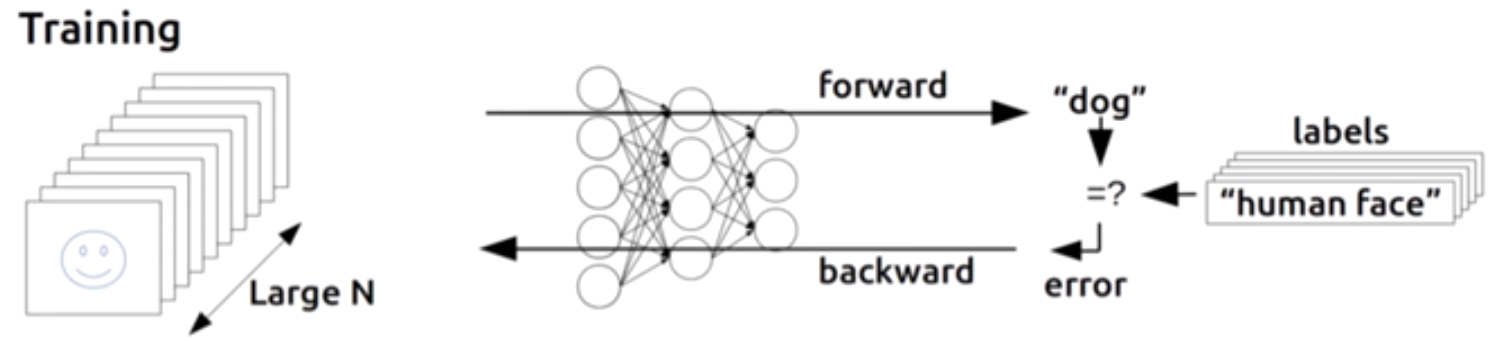


* 위 그림은 다음과 같은 수식으로 표현 가능하다.
* 그렇다면 어떻게 W1, W2, B1, b2를 학습시킬 수 있을까?

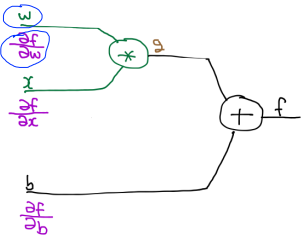
How can we learn W1, W2, B1, b2 from learning data?

* 기존 방법은 우리가 cost function을 만들어 이를 미분해 기울기를 구한 뒤, cost가 가장 작은 값을 가지는 방향으로 학습을 시키는 방법이었다.
* NN의 경우는 hidden layer(sigmoid 등)가 여러 개 생기게 되면서 각 hidden layer가 output에 영향을 미쳤기 때문에 이 hidden layer의 가중치 등을 조절하는 게 어렵다.

Backpropagation



* 우리가 예측으로 만든 출력값을 실제 값과 비교해서 여기서 나오는 cost를 뒤에서부터 앞으로 돌리며 미분값 혹은 무엇을 수정해야 하는지 확인하는 방법이다.
* 위 식에서 각 가중치가 영향을 미치는 것을 layer로 표현하면 다음과 같다.

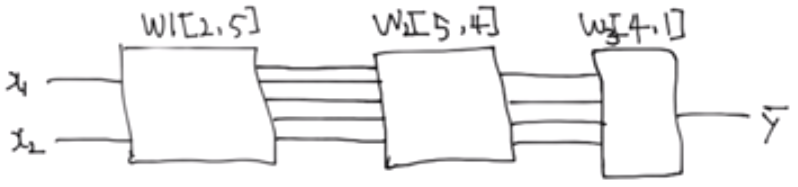
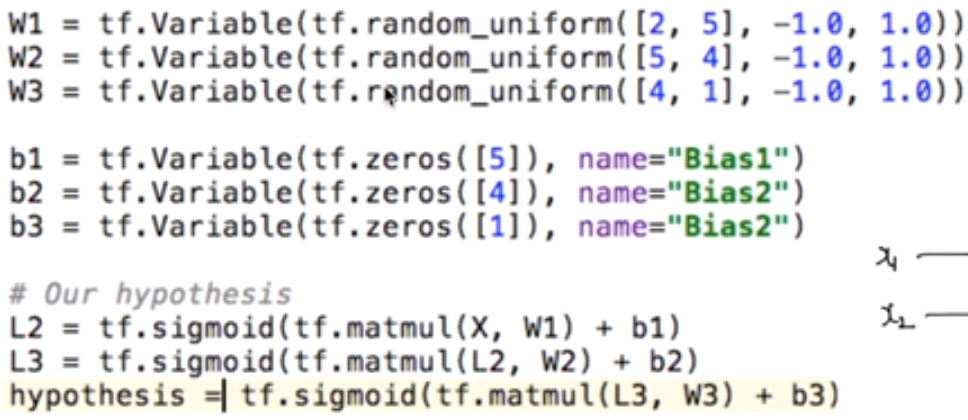


* 위 식에서 우리가 처음에 이라 하자.
* (위 값을 대입한 경우)
* Backpropagation은 미분의 chain rule을 이용하는 것과 비슷하다고 생각하면 된다. Chain rule을 사용하면 가장 안쪽의 함수부터 미분값을 계산하게 되는데, 이와 원리가 비슷하다고 생각하면 된다.
* 가장 안쪽(오른쪽)에서부터 쭉 곱해져 온 값을 왼쪽에 곱하게 되면 결국 전체 함수에 미치는 영향이 어느 정도인지 알 수 있다.
* Backpropagation은 TF에서는 TensorBoard라는 기능을 이용해 사용 가능하다.

**Lecture 10 – ReLU: Better non-linearity**

NN for XOR

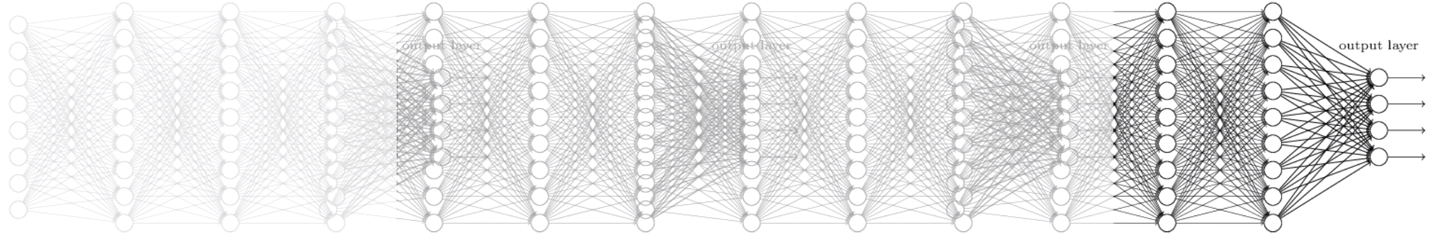
* Activation function: layer 사이를 넘어갈 때 특정 값 이상이면 activate를 시킨다. Sigmoid가 대표적인 예이다.
* 가령, 다음과 같은 3단 layer model을 사용한다고 생각하자.

* 처음의 layer는 input layer, 마지막 layer는 output layer, 중앙의 layer는 hidden layer라고 한다.

Poor results? – vanishing gradient problem

* 9개의 layer로 XOR 문제를 학습시켰다고 하자. Hidden layer는 5\*5의 weight를 가진다.
* Backpropagation의 경우 2~3단 정도의 layer는 학습이 비교적 잘 이루어지나, 그 이상의 많은 layer를 가지는 경우 학습이 잘 되지 않았다.
* 이는 sigmoid 함수의 특성 상 1보다 작은 값들이 chain rule에 따라 연속해서 곱해지게 되고 input layer에 도달하게 되면 gradient의 크기가 매우 작아지게 된다.



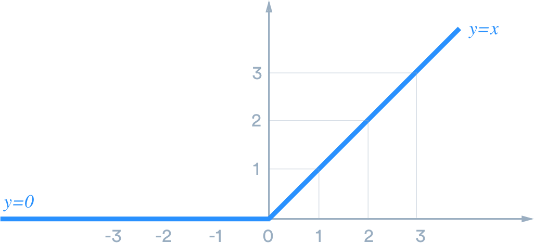
* 이는 두 번째 NN winter(1986~2006)의 원인이 되기도 한다.
* 사실, 이 문제는 non-linearity function을 잘못 사용했기 때문이다.

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.

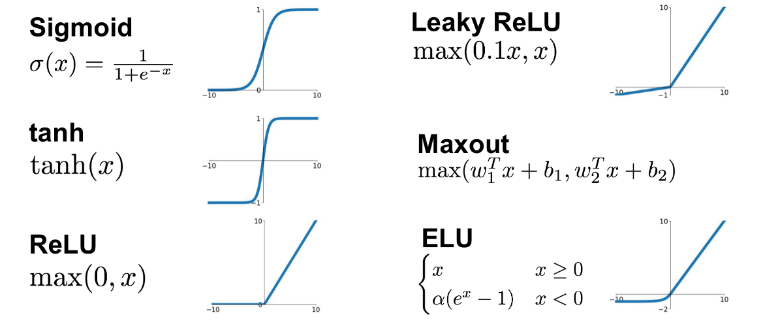
ReLU: Rectified Linear Unit

* NN에서는 sigmoid를 사용하지 않는 것이 좋다. Squishing function이기 때문이다.
* ReLU function은 다음과 같은 모양을 가진다.



* TF에서 사용시는 tf.nn.relu()로 사용한다.
* NN에서 마지막 단에서는 0~1 사이의 값이 필요하기 때문에 sigmoid를 사용해야 한다.

Activation functions

* Sigmoid
* Tanh
* ReLU
* Leaky ReLU
* ELU
* Maxout

Initializing weights

* 같은 ReLU 함수를 사용해도 weight에 따라서 학습 효율 등이 달라지게 된다.
* 즉, initial weight를 설정할 때에는 굉장히 유의해야 한다.
* Weight는 모두 0이 되면 안 된다. 학습이 전혀 진행되지 않는다.
* Weight의 개수가 많아 어떻게 설정할지가 굉장히 어렵다.
* 이와 관련된 해결책으로, Restricted Boatman Machine(RBM)이 존재하며 이를 이용해 initializing을 진행한 network를 Deep Belief Nets(DBM)이라 한다.

RBM

* 먼저 우리가 입력에 대해 weight를 곱한 결과를 얻는다.
* 이후 이 weight를 동일하게 적용해 ‘역방향’으로 input을 recreate한다.
* 이 때, 처음 input과 재생성된 input의 차이가 최소가 되는 방향으로 weight를 조절한다.
* 이 과정은 두 layer 사이에서 진행되는 과정이라고 생각할 수 있다.

RBM을 어떻게 사용할 것인가? – DBM, pre-training

* 여러 layer가 있겠지만, 입력에 대해 앞의 두 layer에 대해 RBM을 적용하고, 이후 이 weight가 적용된 입력값을 이용해 다음의 layer에 대해 이 과정을 점진적으로 반복한다.
* 이렇게 얻어낸 weight가 초기화가 된 weight이다.

Other initializations

* Weight initialization을 위해 복잡한 RBM을 사용할 필요가 없다. 간단한 다른 방법을 사용해도 무관하다.
* Xavier initialization / He’s initialization

Xavier/He initialization

* 너무 크지도, 작지도 않은 값을 사용하고 싶다.
* Input(fan\_in)의 수와 output(fan\_out)의 수를 이용해 random한 값을 부여한다.
  + W = np.random.randn(fan\_in, fan\_out) / np.sqrt(fan\_in) # Glorot et.al 2010
  + W = np.random.randn(fan\_in, fan\_out) / np.sqrt(fan\_in / 2) # He et.al 2015

Overfitting problem

* Overfitting은 실제보다 주어진 dataset에만 학습이 과하게 최적화되는 것을 말한다.
* 우리가 overfitting을 하고 있는지 알 수 있을까?
* Training dataset에 대해 매우 높은 정확도를 보인다. (0.99) 하지만 실제 데이터를 적용하는 경우 좋지 않은 효율을 보인다. (0.85)

Solutions

* Training data를 최대한 많이 확보한다.
* Regularization을 적용한다.

Regularization

* weight에 너무 큰 수를 부여하지 않는다.
* 일반적으로 L2reg를 많이 사용한다.

Dropout

* Dropout: A Simple Way to Prevent Neural Networks from Overfitting [Srivastava et al. 2014]
* Neural net을 구성한 뒤, random하게 몇 개를 끊어버린다.
* TF에서는 layer를 보내기 전 dropout rate가 있는 다른 layer를 하나 만들어 구현한다.
* 조심할 점은, dropout은 학습 시에만 적용된다는 점이다.

Ensemble

* Dataset에 대해, 하나의 learning model을 사용하는 것이 아닌 여러 개의 model을 사용해 합치는 것을 의미한다.

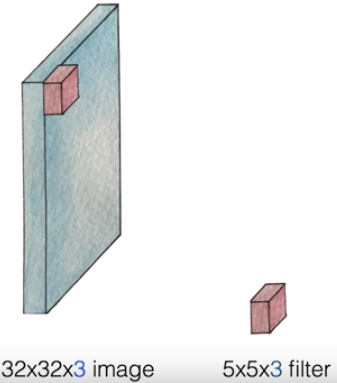
다양한 형태의 model

* Feedforward network: 우리가 지금까지 만든 형태의, 레고처럼 layer를 쌓아 올린 형태의 model
* Fast forward model: 어느 layer에서 나온 출력값을 두 layer를 건너뛰고 입력시키는 방법
* Split & merge: 입력을 나누거나 중간에 쪼갠 뒤 합치는 형태의 model (Convolutional NN)
* Recurrent network: 격자와 같이 한 줄이 아닌 옆으로 이동이 가능한 NN

**Lecture 11 – CNN: Convolutional Neural Network**

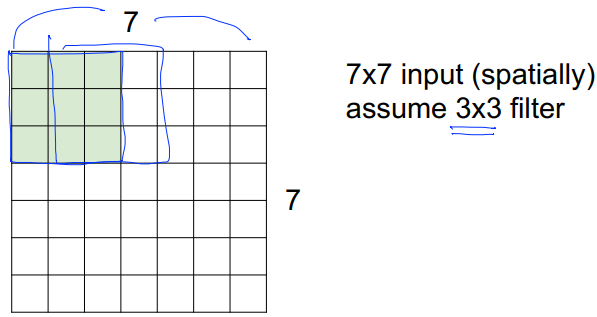
Convolutional Neural Networks

* 입력을 여러 개로 나누어서 투입하는 방법.
* 고양이에게 막대기를 보여줬더니, 방향에 따라 반응하는 뉴런의 종류가 달랐다. 입력을 나누어 받았다는 의미로 해석이 가능하다.



* 실제 이미지의 경우도 위와 같이 이미지를 잘라서 입력하는 방법으로 구현이 가능하다.
* 우리는 위의 5\*5\*3짜리 값을 하나의 단일 숫자로 만들 것이다.
  + Filter = Wx + b 또는 ReLU(Wx + b)

How many numbers?



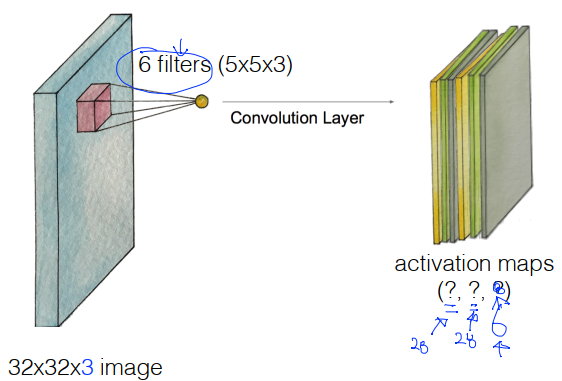
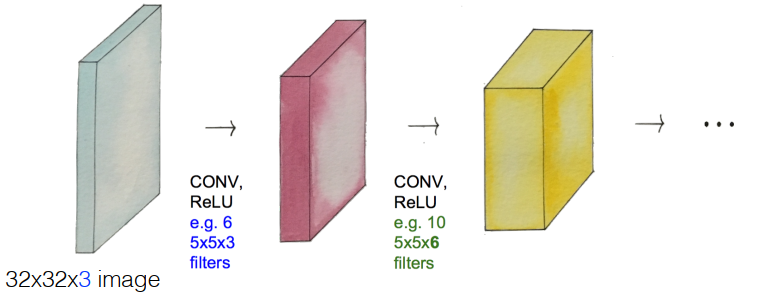
* 위와 같이 7\*7 격자에 3\*3 filter가 있다 하자. 이동하는 칸(stride)를 1로 하면 우리는 5\*5개의 값을 얻을 수 있고, stride를 2로 하면 3\*3의 값을 얻을 수 있다.
* 전체 input의 한 변의 길이가 N, filter의 한 변의 길이가 F라 하면 output size는

(N - F) / stride + 1이다.

* Stride가 크면 output의 크기가 작아지고, 이는 곧 데이터의 손실을 의미한다.
* 이러한 현상을 최소화하기 위해 zero pad를 이미지의 가장자리에 추가한다. 이는 이미지의 테두리를 표현하기 위함도 있다. 1픽셀을 추가했을 경우, stride가 1이라면 output은 원래의 크기와 같게 된다.
* 일반적으로 zero-padding은 (F-1)/2만큼 해준다.

Swiping the entire image

* Filter를 하나만 사용하지 않고, 여러 종류를 사용해 다양한 activation map을 만들게 된다.
* 이들을 겹쳐 새로운 convolution layer를 만들고, 만든 부분에 대해 같은 과정을 반복할 수 있다.

* 그렇다면 여기서 사용되는 weight의 수가 얼마나 될까?
  + 분홍색을 만드는 과정에서는 75개의 weight를 사용하고, initialization 방법은 앞서 공부한 방법과 동일하게 진행한다.

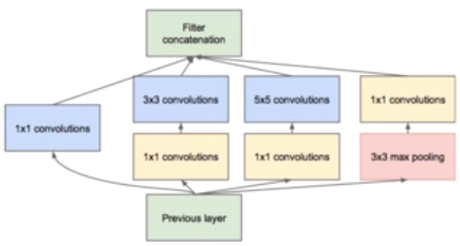
Pooling layer

* 제작한 convolution layer에서 하나의 layer만을 가져온다. 이 layer를 작게 resize라고 다시 합치는 과정을 pooling이라 한다.
* Max pooling: filter 내에서 가장 큰 값만을 취하는 방법

Fully Connected Layer(FC Layer)

* Convolutional layer, ReLU, pooling 등을 반복해 얻어낸 데이터를 기존의 NN 방식으로 학습시키는 것을 의미한다.

CNN의 활용 예시

* LeNet-5
* AlexNet
  + Conv, max pool을 반복하며 중간에 normalization layer도 넣는다. 단 이 Norm layer는 현재는 잘 사용되지 않는다.
* GoogLeNet
  + Inception module
* ResNet
  + 152개의 layer를 사용하게 된다.
  + 굉장히 많은 layer를 사용하는데, 중간중간 layer를 건너뛰는 방법을 사용했다.
  + 3%의 굉장히 낮은 오답률을 보였다.
* Convolutional NN for Sentence Classification