VIII. 딥러닝(Deep Learning) 강의안

Contents(목차)

1. 딥 러닝 Intro

2. 딥 러닝 기본 개념

3. 퍼셉트론

4. 딥 러닝 활성화함수

5. 인공신경망 모형 종류

6. Random Initialization

7. Parameters and Hyperparameters

8. 딥 러닝의 역사

9. 딥 러닝 알고리즘의 발전과 활용

10. 딥 러닝 응용 분야

VIII. 딥러닝(Deep Learning)

1. 딥 러닝 Intro

인공지능, 머신 러닝, 딥 러닝 관계

인공지능

인간과 유사하게 사고하는 컴퓨터 지능을 일컫는 포괄적 개념

머신러닝

데이터를 통해 컴퓨터를 학습시키거나, 컴퓨터가 스스로 학습하여 인공지능의 성능(정확도, 속도, 응용 범위 등)을 향상시키는 방법

딥러닝

인공신경망 이론 기반으로 인간의 뉴런과 유사한 입/출력 계층 및 복수의 은닉 계층을 활용하는 학습 방식. 복잡한 비선형 문제를 비지도방식 학습으로 해결하는 데 효과적

강인공지능과 약인공지능

강인공지능

● 다양한 분야에서 보편적으로 활용

● 알고리즘을 설계하면 AI가 스스로 데이터를

찾아 학습

● 정해진 규칙을 벗어나 능동적으로 학습해 창조

가능

약인공지능

● 특정 분야에서만 활용 가능

● 알고리즘은 물론 기초 데이터, 규칙을 입력해야

● 이를 바탕으로 학습 가능, 규칙을 벗어난 창조는

불가

알파고 (AlphaGo)

● 알파고(AlphaGo)는 구글(Google)의 딥마인드(DeepMind Technologies Limited)가 개발한 인공지능(AI,

Artificial Intelligence) 바둑 프로그램

● 영국의 스타트업 기업이었던 딥마인드가 2014년 구글에 인수되면서 개발이 본격적으로 진행되었다.

● 2015~2017년 프로토타입 버전인 알파고 판, 알파고 리, 알파고 마스터가 공개되었고, 2017년 10월에

최종 버전인 알파고 제로를 발표하였다. 2018년 12월에는 바둑을 포함한 보드게임에 적용할 수 있는

범용 인공지능 알파 제로(Alpha Zero)를 발표하였다.

알파고 (AlphaGo)

● 2015년 10월 유럽바둑챔피언 판후 2단에 5:0 승리

● 2016년 3월 이세돌 9단에 4:1 승리

● Deep Learning

○ 일류 프로선수들의 기보를 학습(3000만수)

○ 알파고 프로그램끼리 대국을 시켜 승리한 판의 수들에 가중치

알파고 (AlphaGo)가 가져온 의미

● 알파고는 인공 지능 연구의 랜드마크적 성장으로 일컬어졌다.

● 2016년 3월에 개최된 알파고와 이세돌의 대결은 인공 지능 연구에 있어 획기적인

사건이었다. 대국 전에 대부분의 참관인들은 이세돌이 알파고를 이길 것으로

기대하였다. 하지만, 5차례의 대국에서 알파고는 이세돌을 4대 1로 이겼다.

● 체스와 함께 바둑에서도 컴퓨터가 사람을 이기면서 기존의 방식으로 인기 보드 게임에서

이기는 것은 더 이상 인공 지능의 중대 사건이 아니게 되었다. 딥 블루의 머리 캠벨은

알파고의 승리를 '한 시대의 끝 ... 보드 게임은 거의 마무리되었고 이제는 옮겨갈 때'라고

하였다.

● 딥 블루나 왓슨과 비교하여 알파고의 근원적인 알고리즘은 보다 다목적인 잠재성이

있으며, 과학계가 인공 일반 지능(AGI, artificial general intelligence)으로 진전하고 있다는

증거일 수 있다.

● 일부 해설자들은 알파고의 승리는 사회에 있어 인공 일반 지능을 지닌 기계가 가져올 수

있는 미래의 충격에 대한 대비의 논의를 시작하는 좋은 기회를 가져다 주었다고 평가한다.

알파고 제로(AlphaGo Zero)

● 딥마인드는 새로운 알파고에 ’알파고 제로’란 명칭을 붙였다.

-> ‘완전히 맨 바탕에서 시작했다’는 점을 강조한 것!

● 알파고 제로는 아예 백지 상태에서 수많은 강화학습을 통해 바둑을 스스로 터득한다.

● 핵심은 학습 속도가 아니라 기존 기보나 이론을 공부하지 않은 채 혼자 바둑을 두면서

실력을 배양했다는 점이다. 즉, 여러 수를 둔 뒤 강화학습을 통해 최적의 규칙과 수를

익혀나간 것이다.

알파고 제로(AlphaGo Zero)

기존 AI

● 인간이 이미 잘 할 수 있는 것들을

자동화하는 수준

알파고 제로

● 방대한 데이터 분석과 강화학습 등을 통해

‘인간의 지혜’를 배우는 데 초점을 맞춤

알파고 제로(AlphaGo Zero)

● 알파고 제로는 이전 세대와 비교했을 때 월등한 성능을 자랑한다.

● 순위 산출에 사용되는 엘로 점수\*1를 기준으로 했을 때 알파고 제로는 5,185점을 보유하고 있다. [그림

1] 알파고 마스터(4,858점)는 327점, 알파고 리(3,739점)와는 1,446점, 알파고 판(3,144점)과는 2,041점의

격차가 있었다.

● 엘로 점수에서 800점 이상 차이 나면 승률이 100%라는 것을 고려할 때, 알파고 제로는 현존하는

인공지능 바둑 컴퓨터로서 최정상급이다.

[그림] 알파고 제로와 기존 알파고 버전 간의 바둑 실력 비교 그래프

VIII. 딥러닝(Deep Learning)

딥러닝 (Deep Learning) 이란?

● 딥 러닝(deep learning), 심층학습(深層學習)은 여러 비선형 변환기법의 조합을 통해 높은 수준의

추상화(abstractions, 다량의 데이터나 복잡한 자료들 속에서 핵심적인 내용 또는 기능을 요약하는

작업)를 시도하는 기계학습(machine learning) 알고리즘의 집합

● 즉, 사람의 사고방식을 컴퓨터에게 가르치는 기계학습의 한 분야

딥러닝 (Deep Learning) 이란?

쉘로우 러닝 (Shallow Learning)

● Hidden layer가 1개 일 경우

딥 러닝 (Deep Learning)

● Hidden layer가 2개 이상인 경우

뉴런(Neuron)이란?

● Neuron : 생물체의 신경계를 이루는 신경 세포

● Artificial neuron : 생물체의 neuron을 구성하는 dendrite, soma, axon의 기능을 추상화한

것

[그림] 생물체의 neuron (좌)과 artificial neuron (우)

뉴럴 네트워크(neural network)란?

● “뉴런”을 여러 개 묶어 연결시킨 것

뉴럴 네트워크(neural network)란?

Single Neural Network(단일 뉴럴 네트워크) Multiple Neural Network(다중 뉴럴 네트워크)

인공신경망(Artificial neural network; ANN)이란?

● 딥 러닝의 가장 핵심적인 기술로써, 신경 세포인 neuron을 추상화한 artificial neuron으로 구성된

네트워크

● 일반적으로 어떠한 형태의 function이든 근사할 수 있는 universal function approximator로도 알려져

있음

인공신경망(Artificial neural network; ANN)이란?

● 인공신경망은 크게 input layer (입력층), hidden layer(은닉층), output layer(출력층) 이 있다.

● 신경계와 비교를 하면 input layer (입력층)은 자극, hidden layer(은닉층)은 신경계, output

layer(출력층)은 반응이다.

● 그리고 여기서의 원은 node라고 하며 신경계에 비유하면 하나의 뉴런이라고 할 수 있다.

● 은닉층은 저희가 직접 볼 수 없는 부분이라서 분석을 하면 어떤 식으로 분석이 되는지 알 수 없다.

인공신경망(Artificial neural network; ANN)의 기본 모형

● f : 비선형 함수

● X : 연속형 그리고 혹은 이산형 변수들의 벡터로 이루어져 있으며 input nodes라고 한다.

● Y : 연속형 그리고 혹은 이산형 변수들의 벡터로 이루어져 있으며 output nodes라고 한다.

● G : x의 함수로 g(x)는 hidden nodes라고 한다.

● W : hidden node들의 가중치로 인공신경망의 핵심

● wi : 가중치는 조정할 수 없는 부분이기 때문에 인공신경망 분석 내에서 추정을 하여

사용한다.

● σ : (activation function) output Y를 만들기 위해 은닉된 부분을 가중하는 값

인공신경망(Artificial neural network; ANN)이란?

● 인공신경망(Artificial Neural Network)는 가장 처음 나온 Single-Layer Neural

Network(단일층 신경망) -> Multi-Layer Neural Network(다층 신경망) ->

Convolutional Neural Network(합성곱 신경망)->... 순서대로 발전했다.

인공신경망(Artificial neural network; ANN)의 종류

● Multi-Layer Neural Network(다층 신경망)은 hidden layer(은닉층)가 3개 이상 들어간

Neural Network(인공 신경망)을 말한다.

● Convolutional Neural Network(합성곱 신경망)은 3차원 데이터의 공간적 정보를

유지한 채 다음 레이어로 보낼 수 있는 Neural Network(인공 신경망)을 말한다.

대표적인 CNN으로는 LeNet(1998)과 AlexNet(2012)이 있으며, VGG, GoogLeNet, ResNet

등이 있으며, 이들은 층을 더 깊게 쌓은 CNN기반의 DNN(Depp Neural Network, 심층

신경망)이다.

● Recurrent Neural Network(순환 신경망)은 순차적인 정보를 이용하여 시퀀스의 모든

요소에 대해 동일한 작업을 수행하고 출력은 이전 계산에 의존하는 Neural

Network(인공 신경망)을 말한다. 구글의 번역기와 네이버의 파파고가 RNN을 응용한

모델이다.

합성곱 신경망(Convolutional Neural Network)

일반 신경망(Standard Neural Network) 합성곱 신경망(Convolutional Neural Network)

순환 신경망(Recurrent Neural Network)

일반 신경망(Standard Neural Network) 순환 신경망(Recurrent Neural Network)

순환 신경망(Recurrent Neural Network)

● 왼쪽 그림을 Unfold한 오른쪽 그림은 전체 네트워크로 전개되는 RNN을 보여준다.

Unfold(unrolling)란 단순히 전체 시퀀스에 대한 네트워크를 작성한다는 의미이다.

예를 들어 우리가 신경 써야하는 시퀀스가 5 단어의 문장이라면 네트워크는 5단어

신경망, 각 단어마다 하나의 레이어로 전개된다.

VIII. 딥러닝(Deep Learning)

3. 퍼셉트론

Perceptron(퍼셉트론)이란?

● Perceptron(퍼셉트론) :

다수의 신호(Input)을 입력받아서 하나의 신호(Output)을 출력한다.

각 입력신호에는 고유한 weight가 부여되며 weight가 클수록 해당 신호가 중요하다.

기계학습이 하는 일은 이 weight(parameter)의 값을 정하는 작업

출력 값은 앞에서 말했듯이 1 또는 0(or -1)이기 때문에 선형 분류(linear classifier) 모형

Perceptron(퍼셉트론) 학습 방법

1. 처음에는 임의로 설정된 weight로 시작한다.

2. 학습 데이터를 퍼셉트론 모형에 입력하며 분류가 잘못됐을 때 weight를 개선해 나간다.

3. 모든 학습 데이터를 정확히 분류시킬 때까지 학습 진행

-> 따라서 학습 데이터가 선형적으로 분리될 수 있을 때 적합한 알고리즘

Perceptron(퍼셉트론)의 한계점

● 선형으로 분류를 할 수 있지만 XOR와 같이 선형 분류만 가능하며 비선형 분류는

불가능하다

● 아래의 그림을 보면 XOR에서는 선형으로(직선 하나로) 분류가 불가능함을 알 수 있다.

-> 이를 극복한 것이 다층 퍼셉트론

Multi-Layer Perceptron(다층 퍼셉트론)이란?

● Multi-Layer Perceptron(다층 퍼셉트론) : 하나의 퍼셉트론에 또 다른 퍼셉트론을 덧붙인다는 의미

단층 퍼셉트론이 비선형 영역을 분리할 수 없다는 것이 문제이며 다층으로 할 경우 비선형으로

이를 해결할 수 있다.

이런식으로 층을 겹겹이 쌓아나가면서 선형 분류만으로는 풀지 못했던 문제를 비선형적으로 풀 수 있게

된다.

VIII. 딥러닝(Deep Learning)

4. 딥 러닝 활성화함수

Activation Function(활성화 함수)란?

● Activation Function(활성화 함수) : 어떠한 신호를 입력받아 이를 적절한 처리를 하여

출력해주는 함수로 이를 통해 출력된 신호가 다음 단계에서 활성화 되는지를 결정

● 어떠한 활성화 함수를 사용하느냐에 따라 그 출력 값이 달라지기 때문에, 적절한 활성화

함수를 사용하는 것이 매우 중요

Activation Function(활성화 함수)의 종류

● Step Function

● Sigmoid Function

● tanh 함수

● ReLU function

Step Function

● Step Function : 그래프 모양이 계단과 같이 생긴 함수로,

임계값을 기준으로 활성화 되거나 혹은 비활성화 되는 형태를 보이는 함수

Sigmoid Function

● Sigmoid Function : 항상 0과 1사이의 값만 가질 수 있도록하는 비선형 함수

step function은 0, 1이라는 출력 값만 가졌지만, 시그모이드는 이 사이에도 연속적인

출력값이 있다는 것으로 보며, 미분이 가능하도록 1과 0 사이를 부드럽게 이어준다.

Sigmoid Function의 특징

1. 함수값이 (0, 1)로 제한된다.

2. 중간 값은 1/2이다.

3. 매우 큰 값을 가지면 함수값은 거의 1이며, 매우 작은 값을 가지면 거의 0이다

왜 비선형 함수를 사용해야 하는가?

● 선형함수를 사용했을 때는 은닉층을 사용하는 이점이 없기 때문이다.

● 다시 말해 선형함수를 여러층으로 구성한다 하더라도 이는 선형함수를 세번 연속

반복한 것에 지나지 않는다는 의미와 같기 때문이다.

● 예를 들어, y = ax라는 선형함수가 있다고 한다면 이것을 3층으로 구성하면 y =

a(a(a(x))) 와 동일한 것으로 이는 y = a3(x)와 같다. 이는 곧, 굳이 은닉층 없이

선형함수로 네트워크를 구성하는 것은 의미가 없다는 뜻이다.

Sigmoid Function의 문제점

● Sigmoid Function의 Gradient Vanishing 문제 :

미분함수에 대해 x=0에서 최대값 1/4 을 가지고, input값이 일정이상 올라가면

미분값이 거의 0에 수렴하게된다. 이는 |x|값이 커질 수록 Gradient Backpropagation시

미분값이 소실 될 가능성이 크다.

-> 이를 해결하기 위해 생겨난 활성화 함수가 바로 ReLU Function

● 함수값 중심이 0이 아니다 :

함수값 중심이 0이 아니라 학습이 느려질 수 있다. 이는 학습을 zigzag 형태로 만들어

느리게 만드는 원인이 된다.

● exp 함수 사용시 비용이 크다.

tanh 함수 (Hyperbolic tangent function)

● tanh 함수 (Hyperbolic tangent function) : 시그모이드 함수를 transformation해서 얻을

수 있는 활성화함수

tanh 함수 (Hyperbolic tangent function)의 특징

● tanh 함수는 함수의 중심값을 0으로 옮겨 sigmoid의 최적화 과정이 느려지는 문제를

해결했다.

● 하지만 미분함수에 대해 일정값 이상 커질시 미분값이 소실되는 gradient vanishing

문제는 여전히 남아있다.

ReLU function

● ReLU function (Rectified Linear Unit) : 최근 가장 많이 사용되는 활성화 함수

Gradient Vanishing문제를 해결할 수 있으며, 미분이 아주 간단하게 된다.

ReLU function의 특징

1. x>0 이면 기울기가 1인 직선이고, x<0이면 함수값이 0이된다.

2. sigmoid, tanh 함수와 비교시 학습이 훨씬 빨라진다.

3. 연산 비용이 크지않고, 구현이 매우 간단하다.

4. x<0인 값들에 대해서는 기울기가 0이기 때문에 뉴런이 죽을 수 있는 단점이 존재한다.

-> 이를 해결한 활성화 함수가 Leakly ReLU

Leakly ReLU function

● Leakly ReLU function (Leakly Rectified Linear Unit) : ReLU의 뉴런이 죽는(“Dying

ReLu”)현상을 해결하기위해 나온 함수

● 위의 식에서 0.01대신 다른 매우 작은 값 사용 가능하다.

● Leakly ReLU는 음수의 x값에 대해 미분값이 0되지 않는다는 점을 제외하면 ReLU와

같은 특성을 가진다.

PReLU

● Leakly ReLU (Leakly Rectified Linear Unit) function : Leakly ReLU와 거의 유사하지만

새로운 파라미터 α 를 추가하여 x<0에서 기울기를 학습할 수 있게 하였다.

Exponential Linear Unit(ELU) function

● Exponential Linear Unit(ELU) function : 비교적 가장 최근에 나온 함수

○ (Clevert et al. ,2015)

Exponential Linear Unit(ELU) function의 특징

1. ReLU의 모든 장점을 포함한다.

2. “Dying ReLU” 문제를 해결했다.

3. 출력값이 거의 zero-centered에 가깝다

4. 일반적인 ReLU와 달리 exp함수를 계산하는 비용이 발생한다.

Maxout function

● Maxout function : ReLU가 가지는 모든 장점을 가졌으며, dying ReLU문제 또한 해결

하지만 계산량이 복잡하다는 단점이 있다.

Summary

● 가장 많이 사용되는 함수는 ReLU이다. 간단하고 사용이 쉽기 때문에 우선적으로

ReLU를 사용한다.

● ReLU를 사용한 이후 Leakly ReLU등 ReLU계열의 다른 함수도 사용 해본다.

● sigmoid의 경우에는 사용하지 않도록 한다.

● tanh의 경우도 큰 성능은 나오지 않는다.

VIII. 딥러닝(Deep Learning)

5. 인공신경망 모형 종류

인공신경망(Artificial neural network; ANN)의 모형 종류

● Feed-Forward Network (FFN) (순전파)

● Backpropagation (역전파)

● Recurrent Neural Network (RNN) (순환 신경망)

● Hidden Markov Model (은닉 마르코프 모델)

● Gradient Descent (경사 하강법)

Feed-Forward Network (FFN) (순전파)

● Feed-Forward Network (FFN) : 정보들이 input에서 hidden으로 hidden에서 output으로

쭉 한 방향으로 흐르는 신경망

Feed-Forward Network (FFN) (순전파)

Error Backpropagation (오차 역전파)

● Backpropagation (역전파) = Error Backpropagation (오차 역전파) :

결과 값을 통해서 다시 역으로 input 방향으로 오차를 다시 보내며 가중치를

재업데이트 하며 가중치의 에러를 줄이며 이 과정을 계속 반복하는 신경망

FFN에 비해 가중치가 잘 나오는 편이고 또 상대적으로 빠른 편이어서 인공신경망에서

자주 사용하는 종류

Error Backpropagation (오차 역전파)

한 번 돌리는 것을 1 epoch 주기라고 하며 epoch를 늘릴 수록 가중치가 계속 업데이트(학습)되면서 점점 오차가 줄어나가는 방법

Error Backpropagation (오차 역전파)

● 위의 그림을 보면 Output layer에서 나온 결과 값이 가진 오차가 0.6이라고 되어 있다.

이전 노드(뉴런에서) Output layer에 각각 3, 2라는 값을 전달하였기 때문에 Ouput의

Error에 위 노드는 60%, 아래 노드는 40% 영향을 주었다고 볼 수 있다. error 0.6을 0.6,

0.4를 곱해 위 노드에는 error가 0.36이, 아래 노드에는 0.24가 전달된다. 이처럼 오차

역전파는 말 그대로 이렇게 오차를 점점 거슬러 올라가면서 다시 전파하는 것을

의미한다.

Recurrent Neural Network (RNN) (순환 신경망)

● Recurrent Neural Network (RNN) : context unit이라는 부분으로 hidden node들과의

교류하여 output nodes를 만들어내는 과정, 여기서 context unit은 input과 output

node들에는 영향을 미치지 않음.

Recurrent Neural Network (RNN) (순환 신경망)

은닉 마르코프 모델(Hidden Markov Model) 이란?

● Markov Model : 어떠한 날씨, 주식가격 등과 같은 어떠한 현상의 변화를 확률 모델로

표현한 것

● Hidden Markov Model : 통계적 마르코프 모델의 하나로, 시스템이 은닉된 상태와

관찰가능한 결과의 두 가지 요소로 이루어졌다고 보는 모델

● Hidden Markov Model은 observation을 이용하여 간접적으로 은닉된 상태를 추론하기

위한 문제를 풀기 위해 사용된다.

은닉 마르코프 모델(Hidden Markov Model) 이란?

● 아래의 그림은 은닉된 상태(Hidden states)와 그에 따른 observation의 개념을 나타낸다.

HMM을 이용해 우리가 풀고자 하는 문제는 관측 가능한 것은 오직 yt 뿐이며, yt는 qt에

종속적으로 발생한다고 할 떄, yt의 sequence를 통해 qt의 sequence를 추론하는 것이다.

Gradient란?

● Gradient : 어떤 다변수 함수 f(x1,x2,...,xn)이 있을 때, f의 그레디언트(gradient)는

다음과 같이 정의된다.

● 즉, 그레디언트(gradient)는 위 식과 같이 각 변수로의 일차 편미분 값으로 구성되는

벡터이다.

● 그리고 이 벡터는 f의 값이 가장 가파르게 증가하는 방향을 나타낸다.

● 또한 벡터의 크기는 그 증가의 가파른 정도(기울기)를 나타낸다.

Gradient란?

● 예를 들어, f(x,y) = x2 + y2의 그레디언트(gradient)를 구해보면 다음과 같다.

● 이므로, (1,1)에서 f값이 최대로 증가하는 방향은 (2,2), 그 기울기는 ∥(2,2)∥= sqrt(8)

이다.

[그림] f(x,y) = x2 + y2 그래프

Gradient Descent(경사 하강법)란?

● Gradient Descent : Gradient(경사) descent(내려감)

● Gradient Descent Algorithm(Steepest descent Algorithm) :

이러한 그레디언트의 특성을 이용하여 어떤 비용함수의 값을 최소화시키기 위한

파라미터 값을 아래와 같이 점진적으로 찾는 방법

즉, 어떤 초기값 x0 = (x10,...,xn0)부터 시작하여 위 식에 따라 gradient 반대 방향으로

x를 조금씩 이동시키면 f(x)가 극소가 되는 x를 찾을 수 있다는 방법

● Gradient ascent Algorithm :

함수의 극소점이 아니라 극대점을 찾는 것이 목적일 경우 다음 식을 이용하여 x를

업데이트

Gradient Descent(경사 하강법)란?

Gradient Descent(경사 하강법)의 직관적 이해

● 자신이 한치앞도 잘 안보이는 울창한 밀림에 있을 때 산 정상으로 가기 위한 방법은

간단하다. 비록 실제 산 정상이 어디에 있는지는 모르지만 현재 위치에서 가장 경사가

가파른 방향으로 산을 오르다 보면 언젠가는 산 정상에 다다르게 될 것이다.

● 또는 이와 반대로 깊은 골짜기를 찾고 싶을 때에는 가장 가파른 내리막 방향으로 산을

내려가면 될 것입니다.

● 이와 같이 어떤 함수의 극대점을 찾기 위해 현재 위치에서의 gradient 방향으로 이동해

가는 방법을 gradient ascent 방법, 극소점을 찾기 위해 gradient 반대 방향으로 이동해

가는 방법을 gradient descent 방법이라 부른다.

Gradient Descent Optimization Algorithms이란?

● Gradient Descent Algorithm은 Neural network의 weight을 조절하는 과정에 보통

사용된다.

● 이는 네트워크의 parameter들을 θ라고 했을 때, 네트워크에서 내놓는 결과값과 실제

결과값 사이의 차이를 정의하는 함수 Loss function J(θ)의 값을 최소화하기 위해

기울기(gradient) ∇θJ(θ)를 이용하는 방법이다. Gradient Descent에서는 θ에 대해

gradient의 반대 방향으로 일정 크기만큼 이동해내는 것을 반복하여 Loss function J(θ)

의 값을 최소화하는 θ 의 값을 찾는다. 한 iteration에서의 변화 식은 다음과 같다.

● 이 때 η 는 미리 정해진 걸음의 크기 ‘step size’ 로서, 보통 0.01~0.001 정도의 적당한

크기를 사용한다.

Gradient Descent Optimization Algorithms이란?

● 이 때 Loss Function을 계산할 때 전체 train set을 사용하는 것을 Batch Gradient Descent

라고 한다. 그러나 이렇게 계산을 할 경우 한번 step을 내딛을 때 전체 데이터에 대해

Loss Function을 계산해야 하므로 너무 많은 계산량이 필요하다.

● 이를 방지하기 위해 보통은 Stochastic Gradient Descent (SGD) 라는 방법을 사용한다.

이 방법에서는 loss function을 계산할 때 전체 데이터(batch) 대신 일부 조그마한

데이터의 모음(mini-batch)에 대해서만 loss function을 계산한다.

● 이 방법은 batch gradient descent 보다 다소 부정확할 수는 있지만, 훨씬 계산 속도가

빠르기 때문에 같은 시간에 더 많은 step을 갈 수 있으며 여러 번 반복할 경우 보통

batch의 결과와 유사한 결과로 수렴한다. 또한, SGD를 사용할 경우 Batch Gradient

Descent에서 빠질 local minima에 빠지지 않고 더 좋은 방향으로 수렴할 가능성도 있다.

Stochastic Gradient Descent (SGD)의 변형 알고리즘 종류 (1)

● Momentum : Gradient Descent를 통해 이동하는 과정에 일종의 ‘관성’을 주는 것

● Nesterov Accelerated Gradient (NAG) : momentum step을 먼저 고려하여, momentum

step을 먼저 이동했다고 생각한 후 그 자리에서의 gradient를 구해서 gradient step을 이동

● Adagrad : 변수들을 update할 때 각각의 변수마다 step size를 다르게 설정해서

이동하는 방식

Stochastic Gradient Descent (SGD)의 변형 알고리즘 종류 (2)

● RMSProp : Adagrad의 식에서 gradient의 제곱값을 더해나가면서 구한 Gt 부분을 합이

아니라 지수평균으로 바꾸어서 대체한 방법

● AdaDelta : RMSProp과 동일하게 G를 구할 때 합을 구하는 대신 지수평균을 구하는

방법

● Adam : RMSProp과 Momentum 방식을 합친 것. Momentum 방식과 유사하게 지금까지

계산해온 기울기의 지수평균을 저장하며, RMSProp과 유사하게 기울기의 제곱값의

지수평균을 저장

Gradient Descent Optimization Algorithms이란?

● 보통 Neural Network를 트레이닝할 때는 이 Stochastic Gradient Descent (SGD)를 이용한다.

● 그러나 단순한 SGD를 이용하여 네트워크를 학습시키는 것에는 한계가 있다.

● 다음 그림들은 SGD 및 SGD의 변형 알고리즘들이 최적값을 찾는 과정을 시각화한 것이다.

● 빨간색의 SGD가 우리가 알고 있는 Naive Stochastic Gradient Descent 알고리즘이고, Momentum, NAG, Adagrad, AdaDelta, RMSprop 등은 SGD의 변형이다. 보다시피 모든 경우에서 SGD는 다른 알고리즘들에 비해 성능이 월등하게 낮다.

[그림] Gradient Descent Optimization Algorithms at Long Valley

Gradient Descent Optimization Algorithms이란?

[그림] Gradient Descent Optimization

Algorithms at Beale's Function

[그림] Gradient Descent Optimization Algorithms at Saddle Point

VIII. 딥러닝(Deep Learning)

6. Random Initialization

Random Initialization (신경망 모델의 초기값 설정)의 필요성

● Neural Netwrok을 구현하기 위해서는 맨 처음에 theta를 초기화 하는 코드를 작성

optTheta = fminunc (@costFunction, initialTheta, options)

● advanced optimization을 실행할 때 initialTheta값을 아래와 같이 모두 0으로 설정하면

어떻게 될까?

: Zero Initializatoin

initialTheta = zeros (n,1)

-> 문제 발생

Random Initialization (신경망 모델의 초기값 설정)의 필요성

● 0으로 초기화(Zero Initialization) 후 Gradient Descent 알고리즘을 수행하면 모든 Hidden

Layer들이 정확히 같이 값이 변화하기 때문에 그저 중복적인 Hidden Layer에 지나지

않게 된다.

● 이처럼 Zero initialization은 Logistic regression을 작성할 때는 별 다른 문제가 되지

않지만 Neural Network이라면 문제가 된다.

Symmetry breaking (대칭깨짐)

● 이를 해결하기 위한 방법이 바로 Symmetry braking (대칭깨짐) 이다.

즉, 초기 theta 값을 -ε과 ε의 사이의 임의적인 값으로 설정함으로써 문제를 해결할 수

있다.

VIII. 딥러닝(Deep Learning)

7. Parameters and Hyperparameters

Parameters and Hyperparameters

Parameters (=weight)

● 모델 내부에서 확인이 가능한 변수 즉, 데이터를 통해서 산출이 가능한 값

● 모델의 능력을 결정

● 측정되거나 데이터로부터 학습되어짐

● 모델의 Parameters는 최적화 과정에서 결정됨

Hyperparameters (=meta-parameters)

● 모델에서 외적인 요소 즉, 데이터 분석을 통해 얻어지는 값이 아님

● 학습 진도율이나 일반화 변수처럼 사람들이 선험적 지식(priori)으로 설정하거나 외부 모델 메커니즘을 통해 자동으로 설정이 되는 변수

Parameters and Hyperparameters의 예시

Parameters (=weight)

1. 인공신경망에서의 가중치

2. SVM(Support Vector Machine)에서의 Support

Vector

3. 선형회귀나 Logistic 회귀분석에서의

결정계수

Hyperparameters (=meta-parameters)

1. 신경망 학습에서의 Learning Rate(학습률)

2. SVM(Support Vector Machine)에서의 Cost

Value인 C

3. KNN에서의 K의 개수

Parameters의 예시

1. 인공신경망에서의 가중치

2. SVM(Support Vector Machine)에서의 Support Vector

3. 선형회귀나 Logistic 회귀분석에서의 결정계수

Optimization with Hyperparameters

● 신경망 모델의 학습과 그 결과에 따른 손실함수의 값을 최소화하는 방향으로

Hyperparameters(하이퍼파라미터)의 값을 찾는 것이 최적화의 목표

● 하이퍼파라미터 매개변수의 수 n 만큼 가능한 매개변수의 조합이 n x n 만큼

증가하므로 이중에서 최적의 조합을 찾는 것은 쉽지 않다. 다양한 방법으로 자동으로

찾고자 하지만 여전히 이는 연구자의 주관이나 직관, 경험 등에 의존하고 있다.

● 최적의 가중치 값을 구하기 위해서 앞에서는 미분을 통해 기울기를 구하여 가중치

값을 갱신하는 방법인 확률적 경사하강법(Stochastic Gradient Descent; SGD) 방법을

사용하였다. 이는 무작정 가중치 값을 랜덤으로 콕콕콕 찍어서 찾는 방식보다는 훨씬

스마트한 방법이다.

● 확률적 경사하강법 이외에도 다양한 최적화 기법을 통해 최적 하이퍼파라미터를 찾을

수 있다.

Optimization with Hyperparameters

1. Learning Rate (학습 진도율)

: 학습 진도율을 학습하고자 하는 대상이나 망에 따라 적절하게 조절해야 한다.

2. Cost Function (비용 함수)

: 일반적으로 최소자승법을 사용하거나, Cross-Entropy 함수를 사용할 수 있다.

3. Regularization Parameter (정규화 매개변수)

: Overfitting 문제를 피하기 위해 L1 또는 L2 regularization 방법을 사용할 수 있다.

4. Mini-batch 크기

: Mini-batch의 크기가 큰 경우는 병렬연산 구조를 사용할 때 효과적일 수 있으며,

크기가 작으면 더 많은 Update를 할 수 있다.