1. 개요

인공지능과 머신러닝은 지난 수 십년간 대학과 연구소를 중심으로 한 학계가 주도를 해왔다. 하지만 이러한 패러다임은 최근 들어 완전히 바뀌게 되었으며, 우리가 잘 알고 있는 구글, 페이스북, 마이크로소프트 같은 거대 IT기업이나 첨단 기술을 보유한 소규모 스타트업 위주의 기업들이 주도하는 양상이다. 이는 기업이 보유한 대량의 데이터, 참신한 아이디어, 그리고 전례 없는 최신 컴퓨터 성능 덕택이라고 생각된다.

이러한 기업들 중에서, 우리가 잘 알고 있는 구글이 머신러닝 또는 딥러닝 기술을 자사의 프로토타입과 제품에 핵심 역할로 사용하면서, 가장 활발히 인공지능 기술을 연구하고 실제 제품에 활용하는 가장 큰 회사 중에 하나임에 틀림 없다. 기술적으로 보면 우리는 격변의 시기를 마주하고 있다. 여기에는 구글 뿐만이 아니라 마이크로소프트, 페이스북, 아마존, 애플 같은 최신 IT기술 기반의 회사들이 이 분야에 집중적으로 투자하고 있고, 한국에서도 전통적 대기업인 삼성, LG, SK 뿐만 아니라, 네이버, 카카오 등의 회사에서도 활발히 관련 연구를 진행하고 있다.

본 강의자료에서는 대학의 학부생 또는 대학원생을 대상으로, 인공지능(특히, 딥러닝에 중점을 둔 머신러닝 관련 내용) 관련 내용을 학습하기 위해 작성되었다. 기본적으로 인공지능 관련 교과목을 학습하기 위해서는 난해한 수학(특히 확률과 선형대수학)과 프로그래밍 관련 지식이 필수적으로 필요하다. 하지만 이러한 복잡하고 어려운 수학지식이 없는 학부학생일지라도, 기본적인 프로그램 지식만 있으면 쉽게 딥러닝에 접근 가능하도록 노력하였으며, 실제 예제 중심으로 구성하도록 하였다. 대학원생의 경우, 강의자료 각 주제에 관련 된 논문을 찾아보고 그 이론적인 내용을 습득하여, 좀 더 깊이 있는 학습이 이루어져야 한다.

실제 구현 관점에서 보면, 본 강의자료는 구글의 텐서플로우를 기반으로 각 예제가 구현되어 있다. 현재 딥러닝 구현을 위한 다양한 플랫폼들이 소개되어 있다. 그중 가장 대표적으로 많이 사용하는 것들은, 텐서플로우(Tensorflow), 케라스(Keras), 파이토치(Pytorch)등을 들 수 있다. 이중에서 가장 사용하기 편한 것은 케라스로 알려져 있으며, 최근 들어 페이스북에서 만든 파이토치도 그 사용성의 편리함으로 인해 점차 사용자를 늘려가고 있다. 하지만, 가장 많은 사용자를 확보하고 있으며 딥러닝 관련 각종 커뮤니티에서 제일 많은 예제들을 보유하고 있는 딥러닝 개발도구는 텐서플로우라고 할 수 있다. 이러한 이유로, 본 강의에서는 텐서플로우를 기반으로 코드를 구현하고, 그 내용을 설명하였다.

본 강의자료에서 다루고 있는 내용은, 기본적인 텐서플로우 내용 및 사용법, 기본 신경망, CNN(Convolutional Neural Networks), RNN(Recurrent Neural Networks), GAN, 강화학습등 널리 알려진 딥러닝 기술들을 소개하고 있으며, 이러한 딥러닝 망들을 활용하여, 대규모 영상 인식, 자연어처리뿐만 아니라, 현재 다양한 딥러닝 응용으로 소개되고 있는 여러 가지 예들을 구현해보고자 한다.

2. Tensorflow

텐서플로우는 원래 머신러닝과 딥러닝 연구를 수행하는 구글 브레인 팀에서 개발된 오픈 소스 라이브러리다. 그리고 현재는 딥러닝과 관련하여 가장 많이 사용되는 플랫폼이며, 알파고로 유명한 구글의 딥마인드에서도 텐서플로우를 기반으로 관련 연구를 진행하며 개발하고 있다. 파이썬(Python)을 사용, 프레임워크로 애플리케이션을 구축하기 위한 편리한 front-end API를 제공하며 성능이 우수한 C++로 애플리케이션을 실행한다.

가. tensorflow 소개

텐서플로우는 데이터 플로우 그래프(data flow graph)를 사용해서 수치 연산을 하는 라이브러리로 볼 수 있다. 그래프의 노드(node)는 수학적 연산을 나타내고 노드를 연결하는 그래프의 엣지(edge)는 다차원 데이터 배열(array)을 나타낸다. 텐서플로우는 수치연산을 기호로 표현한 그래프 구조를 만들고 처리한다는 기본 아이디어를 바탕으로 구현되었다. 또한, 텐서플로우는 CPU, GPU의 장점을 모두 이용할 수 있고 안드로이드나 iOS 같은 모바일 플랫폼은 물론 맥 OS X와 같은 64비트 리눅스에서 바로 사용될 수 있다.

텐서플로우의 또 하나의 강점은 알고리즘이 어떻게 돌아가고 있는지 알려주기 위해 많은 정보를 모니터링하고 디스플레이 해주는 텐서보드 모듈을 제공한다는 점이다. 더 좋은 모델을 만들기 위해서 알고리즘의 동작을 조사해서 디스플레이 하는 것이 매우 중요하며, 텐서플로우에서는 텐서보드를 이용하여 이러한 기능을 손쉽게 구현하고 눈으로 확인 할 수 있게 해준다. 데이터 플로우 그래프는 수학 계산과 데이터의 흐름을 노드(Node)와 엣지(Edge)를 사용한 방향 그래프(Directed Graph)로 표현한다.

그림 2.1 구글 텐서플로우(Tensorflow) 로고

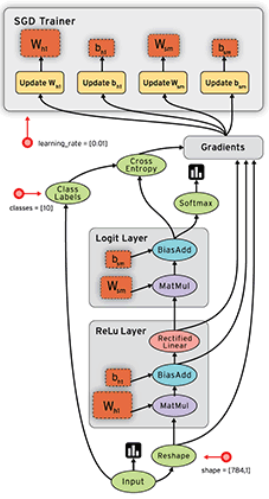


그림 2.2 텐서플로우(Tensorflow) 그래프의 데이터의 흐름

노드(node)는 수학적 계산, 데이터 입/출력, 그리고 데이터의 읽기/저장 등의 작업을 수행한다. 그리고 엣지(edge)는 노드들 간 데이터의 입출력 관계를 나타낸다. 엣지는 동적 사이즈의 다차원 데이터 배열(=텐서)을 실어나르는데, 여기에서 텐서플로우라는 이름이 지어지게 되었다.

> **텐서(Tensor)**는 과학과 공학 등 다양한 분야에서 이전부터 쓰이던 개념임. 수학에서는 임의의 기하 구조를 좌표 독립적으로 표현하기 위한 표기법으로 알려져 있지만, 여러 응용 분야마다 조금씩 다른 의미로 사용됨. 딥러닝에서는 학습 데이터가 저장되는 다차원 배열 정도로 이해하면 됨.

나. 텐서프롤우와 경쟁하는 다른 딥러닝 개발 프랫폼들

텐서플로우는 여러 머신러닝 프레임워크와 경쟁한다. 파이토치(PyTorch), CNTK, MXNet은 텐서플로우와 상당 부분 용도가 비슷한 주요 경쟁 프레임워크다. 필자가 생각한 텐서플로우와 비교한 각 프레임워크의 장단점은 다음과 같다.

- **파이토치(PyTorch)**: 파이썬으로 구축된다는 점 외에도 텐서플로우와 유사한 부분이 많다. 하드웨어 가속 구성 요소, 진행하면서 설계가 가능한 고도의 대화형 개발 모델, 그 외의 많은 유용한 구성 요소가 기본적으로 포함된다. 파이토치는 일반적으로 단시간 내에 실행해야 하는 빠른 프로젝트 개발에 더 유리하지만 큰 프로젝트와 복잡한 워크플로에서는 텐서플로우가 더 적합하다.

- **CNTK**: 마이크로소프트 코그니티브 툴킷(Cognitive Toolkit)은 텐서플로우와 마찬가지로 그래프 구조를 사용해 데이터 흐름을 기술하지만 딥 러닝 신경망을 만드는 데 초점을 둔다. CNTK는 여러 가지 신경망 작업을 더 빠르게 처리하며 폭넓은 API를 보유하고 있다(파이썬, C++, C#, 자바). 그러나 현재 CNTK는 텐서플로우만큼 배우고 배포하기가 쉽지는 않다.

- **아파치(Apache) MXNet**: 아마존이 AWS의 고급 딥 러닝 프레임워크로 채택했으며 복수의 GPU와 머신에 걸쳐 거의 선형적으로 확장이 가능하다. 또한 파이썬, C++, 스칼라, R, 자바스크립트, 줄리아, 펄, 고 등 폭넓은 언어 API를 지원한다. 다만 네이티브 API의 사용편의성은 텐서플로우에 비해 떨어진다.

다. 텐서플로우 기본 용어

1) 오퍼레이션(Operation)

그래프 상의 노드는 오퍼레이션(줄임말 op)으로 불린다. 오퍼레이션은 하나 이상의 텐서를 받을 수 있다. 오퍼레이션은 계산을 수행하고, 결과를 하나 이상의 텐서로 반환할 수 있다.

2) 텐서(Tensor)

내부적으로 모든 데이터는 텐서를 통해 표현된다. 텐서는 일종의 다차원 배열인데, 그래프 내의 오퍼레이션 간에는 텐서만이 전달된다.

3) 세션(Session)

그래프를 실행하기 위해서는 세션 객체가 필요하다. 세션은 오퍼레이션의 실행 환경을 캡슐화한 것이다.

4) 변수(Variables)

변수는 그래프의 실행시, 파라미터를 저장하고 갱신하는데 사용된다. 메모리 상에서 텐서를 저장하는 버퍼 역할을 한다.

< Example 2-1 > Tensorflow 맛보기 : 간단한 곱셈 프로그램

```
import tensorflow as tf       #STEP1

a = tf.placeholder("float")     #STEP2
b = tf.placeholder("float")

y = tf.multiply(a, b)           #STEP3

sess = tf.Session()             #STEP4

print sess.run(y, feed_dict={a: 3, b: 3})   #STEP5
```

위의 < Example 2-1 > 코드를 순차적으로 설명하면 다음과 같다.

STEP1. 텐서플로우 파이썬 모듈을 임포트

STEP2. 프로그램 실행 중에 값을 변경할 수 있는 placeholder라 부르는 심볼릭 변수들을 정의

STEP3. 텐서플로우에서 제공하는 곱셈 함수를 호출하여, 이 두 변수를 파라메타로 넘김.

* 여기에서 tf.multiply은 텐서(tensor)를 조작하기 위해 텐서플로우가 제공하는 많은 수학 연산 함수 중 하나이다. 여기서 텐서는 동적 사이즈를 갖는 다차원 데이터 배열이라고 생각하면 됨.

STEP3. 텐서플로우 세션(tf.Session)열고, 변수(sess)에 할당

STEP5. 변수(a와 b)에 값을 할당하고, 세션 실행(sess.run)

라. 텐서플로우를 이용한 프로그래밍

< Example 2-2 텐서와 그래프 실행 전체 프로그램 살펴보기 >

코드를 직접 입력해가면서 텐서플로우를 익혀본다.

1) 텐서플로우 라이브러리 임포트

```
import tensorflow as tf
```

2) tf.constant로 상수를 특정변수에 저장

```
hello = tf.constant('Hello World!!!')
print(hello)
```

결과 → Tensor("Const:0", shape=(), dtype=string): hello 변수의 값을 출력한 결과 hello가 텐서플로의 텐서(Tensor)라는 자료형이며, 상수이고, 데이터 타입은 스트링이라는 것을 알려주고 있음.

텐서(Tensor)는 랭크(rank)와 셰이프(shape)라는 개념을 가지고 있다.

- rank는 차원의 수를 나타냄
  rank가 0이면 스칼라, 1이면 벡터, 2면 행렬, 3이상이면 n 차원 텐서
- shape는 각 차원의 요소 개수를 나타내며, 텐서의 구조를 설명한다.

```
1. 10                                  #rank=0, shape=()
2. [1., 2., 3.]                        #rank=1, shape=(3)
3. [[1., 2., 3.], [4., 5., 6.]]        #rank=2, shape=(2,3)
4. [[[1., 2., 3.]], [[4., 5., 6.]]]    #rank=3, shape=(2,1,3)
```

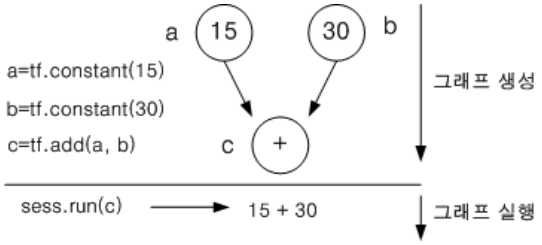
3) 텐서를 이용한 연산

```
a = tf.constant(15)
b = tf.constant(30)
c = tf.add(a, b)  # a + b 로도 쓸 수 있음
print(c)
```

결과 → Tensor("Add:0", shape=(), dtype=int32) : a+b의 결과인 45가 출력되지 않음

텐서플로 프로그램 구조

1. 그래프 생성
2. 그래프 실행



- 그래프 : 텐서들의 연산 모음임. 텐서와 텐서의 연산들을 먼저 정의하여 그래프를 만들고 난 다음, 연산 실행 코드를 원하는 곳에 넣어 프로그램 수행
  이러한 방식을 지연실행(lazy evaluation)이라고 한다.

4) 그래프의 실행

#위에서 변수와 수식들을 정의했지만, 실행이 정의한 시점에서 실행되는 것은 아님

#다음처럼 Session 객제와 run 메소드를 사용할 때 계산이 된다.

#모델을 구성하는 것과, 실행하는 것을 분리하여 프로그램을 작성.

```
# 그래프를 실행할 세션을 구성합니다.
sess = tf.Session()
# sess.run: 설정한 텐서 그래프(변수나 수식 등등)를 실행
print(sess.run(hello))
print(sess.run([a, b, c]))
# 세션을 닫는다.
sess.close()
```

결과 → b'Hello, World!!!'

[15, 30, 45]

< Example 2-2 > 텐서와 그래프 실행 전체 프로그램

```
import tensorflow as tf

# tf.constant: 말 그대로 상수
hello = tf.constant('Hello, World!!!')
print(hello)

a = tf.constant(15)
b = tf.constant(30)
c = tf.add(a, b)  # a + b 로도 쓸 수 있음
print(c)

# 그래프를 실행할 세션을 구성합니다.
sess = tf.Session()
# sess.run: 설정한 텐서 그래프(변수나 수식 등등)를 실행합니다.
print(sess.run(hello))
print(sess.run([a, b, c]))

# 세션을 닫습니다.
sess.close()
```

< Example 2-3 Placeholder와 변수(bariable) 살펴보기 >

5) Placeholder와 변수

- Placeholder: 그래프에 사용할 입력값을 나중에 받기 위해 사용하는 매개변수
- 변수: 그래프를 최적화하는 용도로 텐서플로가 학습한 결과를 갱신하기 위해 사용

```
# tf.placeholder: 계산을 실행할 때 입력값을 받는 변수로 사용합니다.
# None 은 크기가 정해지지 않았음을 의미합니다.
X = tf.placeholder(tf.float32, [None, 3])
print(X)
```
결과 → Tensor("Placeholder:0", shape=(?, 3), dtype=float32)
: placeholder라는 (?, 3) 모양의 float32 자료형을 가진 텐서가 생성

6) Placeholder에 넣을 자료 정의
```
# X 플레이스홀더에 넣을 값
# 플레이스홀더에서 설정한 것 처럼, 두번째 차원의 요소의 갯수는 3개
x_data = [[1, 2, 3], [4, 5, 6]]
```

7) 변수 정의
```
#tf.Variable: 그래프를 계산하면서 최적화 할 변수.
#이 값이 바로 신경망을 좌우하는 값
#tf.random_normal: 각 변수들의 초기값을 정규분포 랜덤 값으로 초기화.
W = tf.Variable(tf.random_normal([3, 2]))
b  = tf.Variable(tf.random_normal([2, 1]))
```

8) 입력값과 변수들을 이용해 계산
```
#입력값과 변수들을 계산할 수식을 작성
#X와 W가 행렬이기에 tf.matmul 사용
#향렬이 아닌경우에는 곱셈연산지(*) 또는 tf.mul 함수 사용
expr = tf.matmul(X, W) + b
```
X가 [2,3] 행열 이므로, W는 [3,2] 행렬로 정의

9) 연산을 실행하고, 그래프 결과 확인
```
sess = tf.Session()

sess.run(tf.global_variables_initializer())
print("=== x_data ===")
print(x_data)
print("=== W ===")
print(sess.run(W))
print("=== b ===")
print(sess.run(b))
print("=== expr ===")
print(sess.run(expr, feed_dict={X: x_data}))

sess.close()
```

< Example 2-3 > Placeholder와 변수(bariable)

```
# 플레이스홀더와 변수의 개념을 익혀봅니다
import tensorflow as tf

# tf.placeholder: 계산을 실행할 때 입력값을 받는 변수로 사용
# None 은 크기가 정해지지 않았음을 의미
X = tf.placeholder(tf.float32, [None, 3])
print(X)

# X 플레이스홀더에 넣을 값
# 플레이스홀더에서 설정한 것 처럼, 두번째 차원의 요소의 갯수는 3개
x_data = [[1, 2, 3], [4, 5, 6]]

#tf.Variable: 그래프를 계산하면서 최적화 할 변수. 이 값이 바로 신경망을 좌우
# tf.random_normal: 각 변수들의 초기값을 정규분포 랜덤 값으로 초기화
W = tf.Variable(tf.random_normal([3, 2]))
b = tf.Variable(tf.random_normal([2, 1]))

# 입력값과 변수들을 계산할 수식을 작성
expr = tf.matmul(X, W) + b

sess = tf.Session()
# 위에서 설정한 Variable 들의 값들을 초기화 하기 위해
# 처음에 tf.global_variables_initializer 를 한 번 실행
sess.run(tf.global_variables_initializer())

print("=== x_data ===")
print(x_data)
print("=== W ===")
print(sess.run(W))
print("=== b ===")
print(sess.run(b))
print("=== expr ===")
# expr 수식에는 X 라는 입력값이 필요
# expr 실행시에는 이 변수에 대한 실제 입력값을 다음처럼 넣어줌
print(sess.run(expr, feed_dict={X: x_data}))

sess.close()
```

3. 선형회귀분석(Linear Regression Analysis)

선형회귀분석이란, 통계학에서 종속 변수 y와 한 개 이상의 독립 변수 x와의 선형 관계를 모델링하는 기법이다. 적절한 과정을 거쳐 x와 y의 관계를 알게 되면 새로운 X가 주어졌을 때, y의 값을 알 수 있게 된다(정확히는 y값을 예측 할 수 있게 된다.). 즉 입력에 대한 출력 값을 예측하는 것이 머신러닝이라며, 선형회귀분석은 머신러닝의 가장 기본이 되는 개념이다.

구체적인 예를 들어 살펴보자. 주택의 크기에 따른 주택 가격을 예측하는 모델을 가정해보자. 우리는 일반적으로 주택의 크기가 크면, 주택의 가격도 비싸진다는 것을 알수 있다.

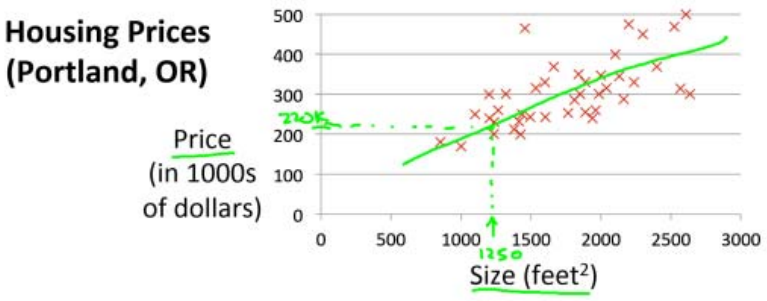


그림 3.1 주택의 크기와 가격 사이의 상관관계 그래프

주택의 크기를 x, 가격을 y변수로 하면, x와 y의 관계를 y = ax + b 형태의 1차원 직선의 방정식으로 표현할 수 있게 된다. 즉, 선형회귀분석에서 선형의 의미는 x와 y의 관계를 1차원 직선으로 표현한다는 의미이다.

$$y = ax + b$$

Price($) in 1000's Size in $feet^2$

이미 수집된 데이터를 학습하여, y값을 추정할 수 있는 예측모델을 만드는 작업이 머신러닝의 기본 개념이다. 이때 훈련에 사용하게 되는 데이터를 훈련 데이터(training data)라고 하며, 그림 3.2는 테이블에 주택가격 관련 훈련 데이터의 예를 보여주고 있다. 이때, 훈련 데이터의 개수를 N, x는 입력 또는 데이터의 특징을 나타내는 feature, 그리고 y는 출력 또는 target이라고 한다.

Training set of housing prices (Portland, OR)

Size in feet² (x)	Price ($) in 1000's (y)
2104	460
1416	232
1534	315
852	178
...	...

m = 47

그림 3.2 주택가격 관련 훈련 데이터의 예

수집된 훈련 데이터를 이용하여 주택의 가격을 추측하는 알고리즘을 학습한다고 하면, 학습시킬 알고리즘이 h, 즉 가설(hypothesis)이라고 한다. 이때 정확한 예측이 가능하려면, 훈련데이터의 분포와 직선의 방정식에 의해 만들어진 직선이 최대한 일치하여야 한다. 즉, 예측에 의해 만들어진 직선과 실제 훈련데이터사이의 오차(error)를 최소화해야 한다. 다른 말로 표현하며, 직선 y = ax + b가 실제 데이터와 최대한 일치 할 수 있도록 파라미터(parameter) a와 b의 값을 추정해야 한다. 또한, 오차를 측정하는 함수를 만들어야 하며, 이를 오차함수(cost function 또는 loss function)이라고 한다. 최종적으로 오차함수의 값이 최소가 되는 파라미터 a와 b의 값을 알아내는 것이다.

일반적으로, 선형회귀분석에서는 cost function으로 평균제곱오차(squared error) 함수를 사용한다. 평균제곱오차는 가설 h_θ에 훈련데이터의 입력 값을 넣었을 때, 실제 출력과 가설에 의한 출력 값의 차를 제곱하여 사용하는 방법이다.

위에서 설명한 내용을 정리하면 다음과 같다.

Hypothesis : $h_\theta(x) = ax + b$, (a 대신에 W를 많이 사용)

Parameter : $\theta : a, b$

Cost Function : $J(a,b) = \frac{1}{2N}\sum_{i=1}^{N}(h_\theta(x^{(i)}) - y^{(i)})^2$

Goal : $\underset{a,b}{\text{minimize}}\ J(a,b)$

최종 목표인 오차함수 $J(a,b)$를 최소화 하는a와 b를 구하는 방법은 산꼭대기에서 내려가는 방법으로 설명할 수 있으며, 아래의 그림 3.3과 같다. 밤중에 산꼭대기에서 랜턴을 쥐고 산 아래로 내려가는 길을 찾고자 한다면, 주변을 살펴보고 아래로 내려가는 길을 찾아 그 방향으로 일정거리를 내려가고, 다시 랜턴을 비추어 아래로 가는 길을 다시 찾아 내려가는 방법을 반복하면 된다. 이런 방법으로 파라미터 a와 b를 찾는 방법을 경사하강법(Gradient descent) 방법이라고 한다. 즉, 경사하강법은 임의의 a와 b를 정한 다음, 그 점으로부터 기울기가 감소하는 구간을 찾아 이동하는 것을 반복하여, 오차함수가 최소가 되는 지점을 찾는 방법이다.

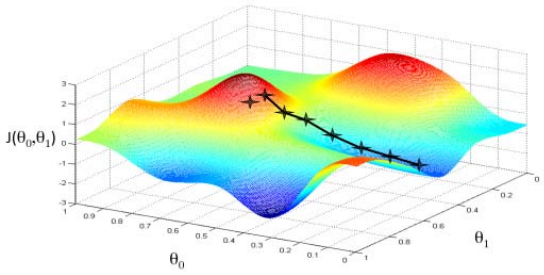


그림 3.3 오차함수 $J(\theta_0,\theta_1)$가 최소화 되는 θ_0와 θ_1을 찾는 과정을 산꼭대기에서 산 아래로 내려가는 과정으로 설명한 예

위에서 설명한 경사하강법을 그래프를 이용하여 다시 설명하면, 그림 3.4로 표현할 수 있다. 그림 3.4에서 “Random initial value”는 임의의 a와 b를 정하는 것이고, “Learning step”은 “Learning rate”라고도 하며, 한 번에 얼마만큼 내려 갈 것인지를 정하는 값이 된다.

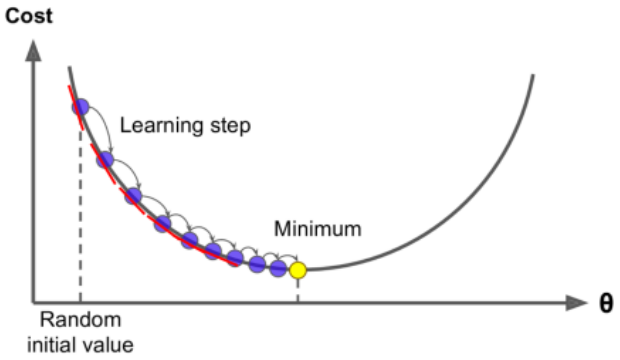


그림 3.4 경사하강법(Gradient descent)

$$J(W,b)=\frac{1}{N}\sum_{i=1}^{N}(h_\theta(x^{(i)})-y^{(i)})^2=\frac{1}{N}\sum_{i=1}^{N}((Wx^{(i)}+b)-y^{(i)})^2$$

$$\rightarrow \quad \frac{1}{2N}\sum_{i=1}^{N}(h_\theta(x^{(i)})-y^{(i)})^2=\frac{1}{2N}\sum_{i=1}^{N}((Wx^{(i)}+b)-y^{(i)})^2$$

: Cost function에서 기울기를 구하려면 미분을 해야 하므로 분모에 2를 곱해주었다. 1/N에서 최소화하나 1/2N에서 최소화하나 어차피 같은 결과가 나오므로 계산상 편하게 하기 위해 분모에 2를 곱해주었다.

경사하강법은 원래 값에서 기울기를 빼가면서(내려가면서) cost 값이 최소가 되는 지점을 찾는 것이다. 이를 수식으로 나타내면 아래와 같다.

$$W:=W-\alpha\frac{\partial}{\partial W}J(W,b),\ \text{where}\ \alpha\text{=학습률(learning rate)}$$

원래 W 값에서 cost의 기울기만큼 빼가면서 다시 W에 값을 업데이트해준다. 이렇게 반복해서 조정시켜주면 언젠가는 최소값을 찾게 돼있다. '언젠가'는 α(학습률)에 달려있다. 학습률은 얼마나 빠르게 내려갈 것인지 스텝 크기를 결정한다. 만약 스텝의 크기가 너무 작다면 알고리즘이 수렴하기 위해 반복을 많이 진행해야 하므로 시간이 오래 걸린다. 반대로 학습률이 너무 크다면 골짜기를 튀어나가 값이 발산하게 된다.

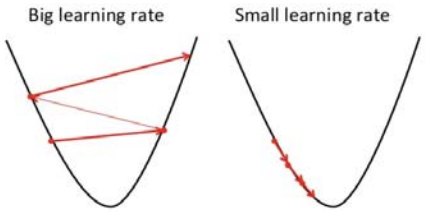


그림 3.5 학습률 α에 따른 오차함수 값의 변화

경사하강법 수식을 풀어서 써본다면,

$$W := W - \alpha \frac{\partial}{\partial W} J(W,b)$$

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

$$W := W - \alpha \frac{1}{2N} \sum_{i=1}^{N} 2((Wx^{(i)} + b) - y^{(i)})$$

$$W := W - \alpha \frac{1}{N} \sum_{i=1}^{N} ((Wx^{(i)} + b) - y^{(i)})$$

미분을 쉽게 하기 위해 분모에 2를 곱해 주었다. 이렇게 최종 경사하강법 수식까지 풀어보았다. 위에서 풀어보았듯이, 경사하강법 알고리즘의 수식은 아래와 같이 나온다.이 수식을 적용시키면 이제 cost 값을 최소화시키는 W와 b를 구해 낼 수 있고, 이러한 선형회귀의 학습과정을 통해 선형함수 모델을 만들 수 있다.

$$W := W - \alpha \frac{1}{N} \sum_{i=1}^{N} ((Wx^{(i)} + b) - y^{(i)}) \text{ : 경사하강법 알고리즘 수식}$$

하지만, 여기서 주의해야 할 점이 있다. 경사하강법은 gradient가 0에 가까워지는 지점을 찾는 것이다. 만일, 오차함수가 2차방정식이 아닌 고차 방정식일 경우, 여러 개의 국부 최소값이 존재하게 된다. 즉, 정확히 표현하면 극값을 찾는 것이지, 최소값을 찾는 것이 아니다. 그러므로, 극값이 여러 개라면 각 극값들을 찾아서 비교해서 최소값을 찾아야 한다. 또한 최소값이 아닌 국부적인 극점인 국부 최소값(local minimum)을 찾을 수도 있다.

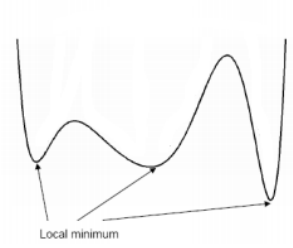


그림 3.6 국부 최소값의 예

가. 텐서플로를 이용한 선형회귀분석 프로그램

먼저 선형회귀모델을 만들 데이터를 생각해보자. 간단하게 다음의 데이터를 고려해 보자.

x_data = [1, 2, 3], y_data = [1, 2, 3]

1) x와 y의 상관관계를 표현할 직선의 방정식 $Wx + b$를 만들기 위해, W와 b를 -1.0에서 1.0 사이의 랜덤한 값을 가지는 값으로 임의로 초기화 한다.

```
W = tf.Variable(tf.random_uniform([1], -1.0, 1.0))
b = tf.Variable(tf.random_uniform([1], -1.0, 1.0))
```

2) 데이터를 입력받을 Placeholder를 설정해 준다.

```
# name: 텐서보드 등으로 값의 변화를 살펴보기 쉽게 이름을 붙여줍니다.
X = tf.placeholder(tf.float32, name="X")
Y = tf.placeholder(tf.float32, name="Y")
```

3) X 와 Y 의 상관 관계를 분석하기 위한 가설 수식을 작성합니다.

```
# y = W*x + b, W와 X가 행렬이 아니므로 tf.matmul이 아니라 기본 곱셈 기호사용
hypothesis = W * X + b
```

4) 손실 함수를 작성합니다.

```
# mean(h - Y)^2 : 예측값과 실제값의 거리를 비용(손실) 함수로 정합니다.
cost = tf.reduce_mean(tf.square(hypothesis - Y))
```

5) 경사하강법 적용

```
# 텐서플로우에 기본적으로 포함되어 있는 함수를 이용해 경사 하강법 최적화를 수행
optimizer = tf.train.GradientDescentOptimizer(learning_rate=0.1)
# 비용을 최소화 하는 것이 최종 목표
train_op = optimizer.minimize(cost)
```

6) 세션을 생성하고 초기화

```
with tf.Session() as sess:
    sess.run(tf.global_variables_initializer())
 for step in range(100):       # 최적화를 100번 수행합니다.
     # sess.run 을 통해 train_op 와 cost 그래프를 계산
     # 이 때, 가설 수식에 넣어야 할 실제 값을 feed_dict 을 통해 전달
     _, cost_val = sess.run([train_op, cost], feed_dict={X: x_data, Y: y_data})
     print(step, cost_val, sess.run(W), sess.run(b))
```

7) 결과 확인

```
print("\n=== Test ===")
print("X: 5, Y:", sess.run(hypothesis, feed_dict={X: 5}))
print("X: 2.5, Y:", sess.run(hypothesis, feed_dict={X: 2.5}))
```

< Example 3-1 > 선형회귀분석 프로그램

```
# X 와 Y 의 상관관계를 분석하는 기초적인 선형 회귀 모델
import tensorflow as tf

x_data = [1, 2, 3]
y_data = [1, 2, 3]

W = tf.Variable(tf.random_uniform([1], -1.0, 1.0))
b = tf.Variable(tf.random_uniform([1], -1.0, 1.0))

# name: 나중에 텐서보드등으로 값의 변화를 추적하거나 살펴보기 쉽게 하기 위해 이름을 붙여줍니다.
X = tf.placeholder(tf.float32, name="X")
Y = tf.placeholder(tf.float32, name="Y")
print(X)
print(Y)

# X 와 Y 의 상관 관계를 분석하기 위한 가설 수식을 작성합니다.
# y = W * x + b
# W 와 X 가 행렬이 아니므로 tf.matmul 이 아니라 기본 곱셈 기호를 사용했습니다.
hypothesis = W * X + b

# 손실 함수를 작성합니다.
# mean(h - Y)^2 : 예측값과 실제값의 거리를 비용(손실) 함수로 정합니다.
cost = tf.reduce_mean(tf.square(hypothesis - Y))
# 텐서플로우에 기본적으로 포함되어 있는 함수를 이용해 경사 하강법 최적화를 수행합니다.
optimizer = tf.train.GradientDescentOptimizer(learning_rate=0.1)
# 비용을 최소화 하는 것이 최종 목표
train_op = optimizer.minimize(cost)

# 세션을 생성하고 초기화합니다.
with tf.Session() as sess:
    sess.run(tf.global_variables_initializer())

    for step in range(100):  # 최적화를 100번 수행합니다.
        # sess.run 을 통해 train_op 와 cost 그래프를 계산합니다.
        # 이 때, 가설 수식에 넣어야 할 실제값을 feed_dict 을 통해 전달합니다.
        _, cost_val = sess.run([train_op, cost], feed_dict={X: x_data, Y: y_data})
        print(step, cost_val, sess.run(W), sess.run(b))

    # 최적화가 완료된 모델에 테스트 값을 넣고 결과가 잘 나오는지 확인해봅니다.
    print("\n=== Test ===")
    print("X: 5, Y:", sess.run(hypothesis, feed_dict={X: 5}))
    print("X: 2.5, Y:", sess.run(hypothesis, feed_dict={X: 2.5}))
```

4. 신경회로망의 기초

실제 뉴런의 모양 및 구성요소는 우리가 구현하게 될 모델과는 많이 다르고, 우리는 길제 뉴런 모델을 간략화 하여 사용하게 된다 신경망을 구성하는 각 노드를 유닛(unit)이라고 부르고 각 유닛은 서로 연결되어 있다.

뉴런은 뇌를 구성하는 기본단위로, 사람의 뇌는 약 1,000억 개의 뉴런을 가지고 있다. 실제로 좁쌀만 한 크기의 작은 뇌 조각 안에 1만 개 이상의 뉴런이 있으며, 각 뉴런은 다른 뉴런과 평균 6,000개의 연결을 형성한다. 그 연결을 통해, 여러 뉴런으로부터 얻은 정보를 처리하고 그 결과를 다른 세포로 전송하게 된다. 한마디로 뇌는 거대한 생물학적 네트워크라고 볼 수 있으며, 사람은 이 신경망을 통해 감정을 느끼고 생각할 수 있게 된다. 인공 신경 회로망(ANN:Artificial Neural Networks)은 인간의 뇌를 구성하는 신경세포인 뉴런(Neuron)의 동작원리를 기초로 하여 구성되었다.

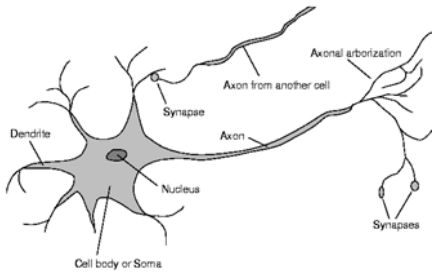


그림 4.1 실제 뉴런 및 정보전달

인공 신경망에 대한 최초의 발상은 1943년, 신경 생리학자 워렌 맥컬로치(Warren McCulloch)와 논리학자 월터 피츠(Walter Pitts)가 함께 제안한 'McCulloch-Pitts 뉴런' 모델이다. 이들은 인간의 두뇌를 이진 원소들의 집합으로 표현했는데, 이 발상을 기초로 하여 1957년, 코넬 항공 연구소(Cornell Aeronautical Laboratory)의 과학자 프랭크 로젠블래트(Frank Rosenblatt)가 '단층 퍼셉트론(SLP, Single-Layer Perceptron)' 모형을 제안한다.

*** 신경회로망의 역사**

- 1943년 : McCulloch & Pitts show that neurons can be combined to construct a **Turing machine** (using ANDs, Ors, & NOTs)
- 1958년 : Rosenblatt shows that perceptrons will converge if what they are trying to learn can be represented
- 1969년 : Minsky & Papert showed the limitations of perceptrons, killing research for a decade
- 1985년 : backpropagation algorithm revitalizes the field
- 2006 년 : Hinton proposed a method that the network which has deep structure can be trained.

가. 퍼셉트론(Perceptron)

단층 퍼셉트론은 다수의 신호(Input)을 받아서 하나의 신호(Output)를 출력한다. 이 때문에 그 동작은 뉴런과 굉장히 유사하며, 다수의 입력을 받았을 때, 퍼셉트론은 각 입력 신호의 세기에 따라 다른 가중치를 부여합니다. 그 결과를 고유한 방식으로 처리한 후, 입력 신호의 합이 일정 값을 초과한다면 그 결과를 다른 뉴런으로 전달한다.

그림 4-2는 입력이 2개인 퍼셉트론의 예를 보여주고 있다. 그림 4-2에서 x1, x2는 입력 신호, y는 출력 신호, w1과 w2는 가중치(weight), 그리고 원을 뉴런 또는 노드라고 한다. 입력 신호가 뉴런에 보내질 때 각각의 가중치가 곱해진다. 뉴런에서 보내온 신호의 총합이 정해진 한계를 넘어설 때에만 1을 출력하고, 이 때 한계를 임계값이라고 하며 θ라고 한다.

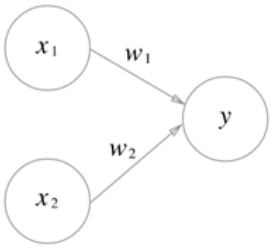


그림 4.2 입력이 2개인 퍼셉트론 모델의 예

$$y = \begin{cases} 0, & (w_1 x_1 + w_2 x_2 \le \theta) \\ 1, & (w_1 x_1 + w_2 x_2 > \theta) \end{cases} \tag{4-1}$$

아래는 θ를 -b(편향 : bias)로 치환한 것임. 즉, 입력 신호 × 가중치 + 편향 > 0 이면 1의 값을 가지게 된다.

$$y = \begin{cases} 0, & (b + w_1 x_1 + w_2 x_2 \le 0) \\ 1, & (b + w_1 x_1 + w_2 x_2 > 0) \end{cases} \tag{4-2}$$

나. 퍼셉트론과 논리회로

퍼셉트론으로 논리회로를 구현할 수 있다. 논리회로의 기본 게이트인 AND, NAND, OR 게이트를 구성하기 위해서는 아래와 같이 적절한 (w_1, w_2, b)를 설정하면 된다.

- AND : if (w_1, w_2, b) = (0.5, 0.5, -0.7)
  x_1 = 1, x_2 = 1일 때에만 $w_1 x_1 + w_2 x_2 + b > 0$를 만족
- NAND : if (w_1, w_2, b) = (-0.5, -0.5, 0.7), 조건식 만족
- OR : if (w_1, w_2, b) = (0.5, 0.5, -0.2), 조건식 만족

AND, NAND, OR 각각의 경우에 따라 달라지는 것은 매개변수의 값뿐이다. 즉, 똑같은 구조에서 매개변수만 바꾸는 것이라고 할 수 있다. 기계학습 문제는 이 매개변수의 값을 정하는 작업을 컴퓨터가 자동으로 하도록 하는 것이다. 기계학습에서 신경망이 적절한 매개변수 값을 정하는 작업을 **학습(training)**이라고 한다. 즉, AND, NAND, OR의 기능이 구현되도록 각각의 논리게이트에 해당하는 매개변수 값을 정해주어야 하는 것을 학습이라고 한다.

w는 입력신호(x)가 결과에 주는 영향력을 조절하는 변수이고, 편향은 뉴런이 얼마나 쉽게 활성화 (결과로 1을 출력) 하느냐를 조정하는 매개변수 이다. 예를 들어 편향의 값이 -10 이면 sum(입력 신호 * 가중치) 값이 10이 넘어야 활성화 된다는 의미이다.

x_1	x_2	y
0	0	0
1	0	0
0	1	0
1	1	1

x_1	x_2	y
0	0	1
1	0	1
0	1	1
1	1	0

x_1	x_2	y
0	0	0
1	0	1
0	1	1
1	1	1

그림 4.3 AND, NAND, OR 게이트 진리표

```
import numpy as np

def AND(x1, x2, w1 = 0.5, w2 = 0.5, b = -0.7):
    x = np.array([x1, x2])
    w = np.array([w1, w2])
    return 1 if np.sum(x*w) + b >=0 else 0

def NAND(x1, x2, w1 = -0.5, w2 = -0.5, b = 0.7):
    x = np.array([x1, x2])
    w = np.array([w1, w2])
    return 1 if np.sum(x*w) + b >=0 else 0

def OR(x1, x2, w1 = 0.5, w2 = 0.5, b = -0.2):
    x = np.array([x1, x2])
    w = np.array([w1, w2])
    return 1 if np.sum(x*w) + b >=0 else 0

result = AND(0, 0)
result = NAND(0, 0)
result = OR(0, 0)
print(result)
```

다. 퍼셉트론의 한계(XOR 문제)

과거에는 지금과 같은 성능의 컴퓨터는 존재하지 않았기 때문에, 퍼셉트론을 하드웨어를 이용하여 구현하였다. 하드웨어만으로도 당시에 중요하게 생각했던 AND와 OR 문제를 해결할 수 있었고, 그 때문에 퍼셉트론을 제안했던 프랭크 로젠블라트(Frank Rosenblatt) 박사는 1958년, 뉴욕 타임스에 위와 같은 이야기를 게재하게 된다. 하지만, 이러한 퍼셉트론에 기대감에 찬물을 끼얹은 문제가 등장했으니, 그것은 바로 XOR 문제였다.

The New York Times

NEW NAVY DEVICE LEARNS BY DOING

July 8, 1958

"The Navy revealed the embryo of an electronic computer today that it expects will be able to walk, talk, see, write, reproduce itself and be conscious of its existence... Dr. Frank Rosenblatt, a research psychologist at the Cornell Aeronautical Laboratory, Buffalo, said Perceptrons might be fired to the planets as mechanical space explorers"

"AND, NAND, OR 게이트 진리표우리가 개발한 인공지능은 스스로 학습해서 걷고 말하며, 보고 쓸 수 있게 될 것이다. 그리고 종국에는 자신의 존재를 인지할 것으로 기대한다."

그림 4.4 Frank Rosenblatt 박사의 뉴욕 타임즈 기사

AND와 OR 문제는 적절한 경계선만 찾으면 입력에 대한 정답을 알 수 있었다. 즉, 그럼 4-5에서 볼 수 있듯이, 출력 값 0 또는 1을 구분할 수 있는 직선을 적절히 찾으면 문제는 해결된다는 것이다. 하지만 XOR 문제는 어떤 경계선을 찾든, 문제를 완벽하게 해결할 수 없었다. 많은 사람이 문제에 도전했지만, 모두 실패로 끝이 나고 말았다. 실제로 1969년, MIT 대학의 인공지능 연구실 마빈 민스키(Marvin Minsky) 교수는 '퍼셉트론즈(Perceptrons)'라는 책을 통해 XOR 문제 해결이 불가능하다는 걸 수학적으로 증명하였고, 단층 퍼셉트론 모델(SLP)의 한계점을 지적하였다.

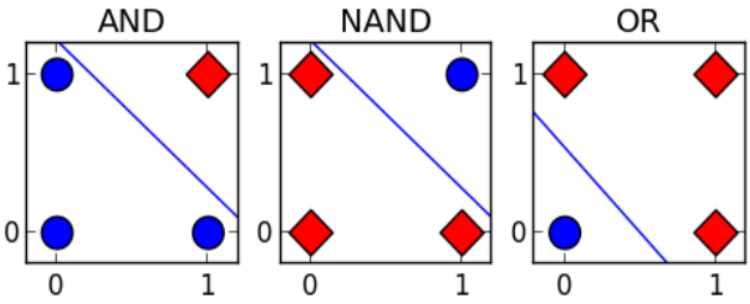


그림 4.5 선형직선으로 분리 가능한 AND, NAND, OR 문제

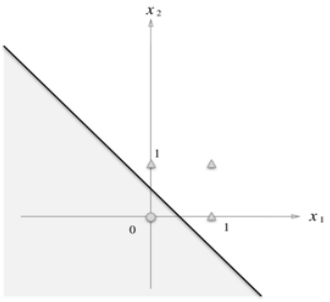
XOR 게이트는 배타적 논리합이라고 알려져 있다. 즉, 입력 2개중 하나만 1일 때 1을 출력한다. XOR 게이트는 AND, NAND, OR 과 같이 단일 퍼셉트론을 이용하여 구현할 수 없

다. 예를 들어 (w_1, w_2, b) = (1, 1, 0.5) 인 경우, 식 4-3과 같이 표현되며, 하나의 직선을 이용하여 구분할 수 없게 된다. 즉, 그림 4.6에서 볼 수 있듯이 XOR는 하나의 직선으로 영역을 둘로 나눌 수가 없다. 이 문제를 해결하기 위해서는, XOR의 경우 그림 4-7과 같이 비선형으로 영역을 나누어야 문제가 해결된다.

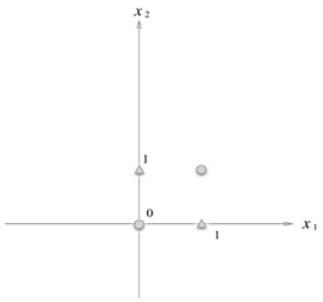
$$y = \begin{cases} 0, & (-0.5 + x_1 + x_2 \le 0) \\ 1, & (-0.5 + x_1 + x_2 > 0) \end{cases} \tag{4-3}$$

x_1	x_2	y
0	0	0
1	0	1
0	1	1
1	1	0

(a) XOR 진리표



(b) OR



(c) XOR

그림 4.6 선형직선으로 분리가 불가능한 XOR

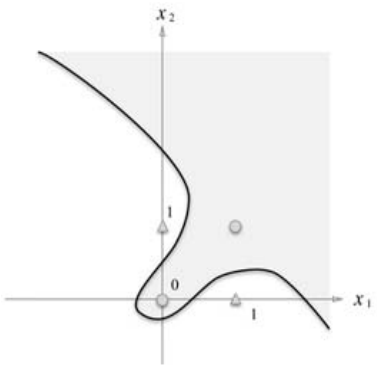


그림 4.6 XOR 문제를 선형이 아닌 비선형으로 영역을 나눈 경우의 예