학습률에 따른 경사 하강법 알고리즘의 정확도와 속도 탐구
Research on Varying Accuracy and Speed of Gradient Descent Relative to Learning Rate

Jaewook Lee

Abstract

In artificial intelligence, one of the most popular technologies in modern society, a technology called Gradient Descent is a quintessential concept. Through gradient descent, it is possible to know whether Feature Extraction is activated and whether models have objectively learned well. This study was conducted to explore the accuracy and speed of gradient descent algorithms according to varying step sizes since some models did not perform well even though it took too long to actually learn when AI models were produced through Keras open-source deep learning models. On the other hand, some models showed high levels of recognition even with a relatively small learning rate.

서론

현대 사회에서 가장 각광받는 기술 중 하나인 인공지능, 그 중 딥러닝에서는 Gradient Descent라는 기술이 굉장히 중요하다. 이를 통해 Feature Extraction 등이 행해지고 사용자들이 객관적으로 학습이 잘 되었는지를 알 수 있다. Keras 오픈 소스 딥러닝 모델들을 제작하며 대입하는 step size에 따라 지나치게 오랜 시간이 걸렸음에도 학습이 잘 되지 않았던 경우도 있고, 비교적 작은 수치를 넣었음에도 높은 수준의 인식률을 보이는 모델들도 있어 step size와 경사하강법 알고리즘의 정확도와 속도에 대해 탐구하고자 이 연구를 진행하였다.

비용함수**(Cost Function)**

Cost function, 즉 비용함수는 원래의 값과 비교하였을 때 오차가 가장 적은 가설 함수를 도출해내기 위하여 만드는 함수이다. $f_{w,b}(x^{(i)}) = wx^{(i)} + b$라는 Linear Model이 있다고 가정한다. 여기서 w는 기울기 b는 상수, 그리고 $x^{(i)}$는 i번째 x값이다. 이러한 일차함수가 있다고 가정였을 때, 비용함수, $J(w, b)$는 아래와 같다.

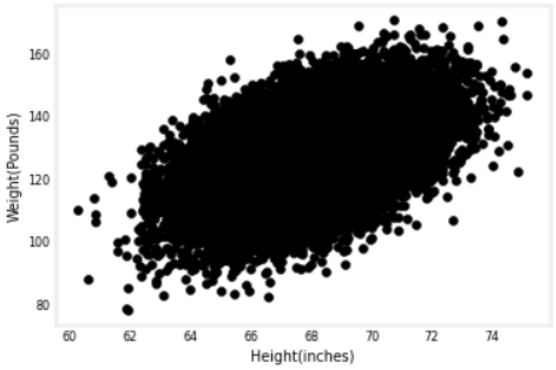
$$J(w,\ b) = \frac{1}{2m}\sum_{i=0}^{m-1}\left(f_{w,\ b}(x^{(i)}) - y^{(i)}\right)^2$$

일단 $f_{w,b}(x^{(i)})$는 가설함수에 $x^{(i)}$를 대입하였을 때 나오는 값이므로 Predicted Value이고, $y^{(i)}$는 실제 값이다. 이 두 변수의 차를 제곱한 후 i가 0일때부터 $m-1$일때까지의 값들을 모두 더한다. 여기서 m으로 나눠 평균을 구하지 않고 $2m$으로 나누는 이유는 연산을 편리하게 하기 위해서이다.

이해를 돕기 위하여 Kaggle에서 제공하는 Heights and Weights 데이터셋을 이용하여 Cost Function을 생성해보았다.

```
height_values = bmi_df["Height(Inches)"].values
weight_values = bmi_df["Weight(Pounds)"].values
plt.scatter(height_values,weight_values)
plt.xlabel('Height(inches)')
plt.ylabel('Weight(Pounds)')
```

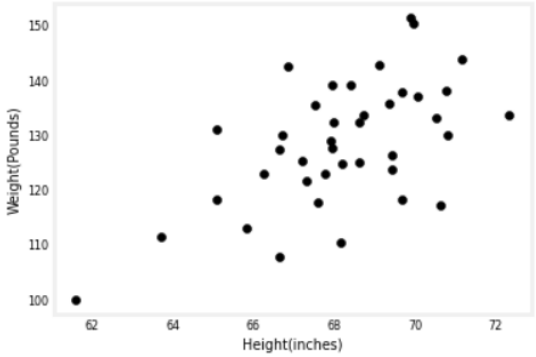
```
Text(0, 0.5, 'Weight(Pounds)')
```



[그림 1]

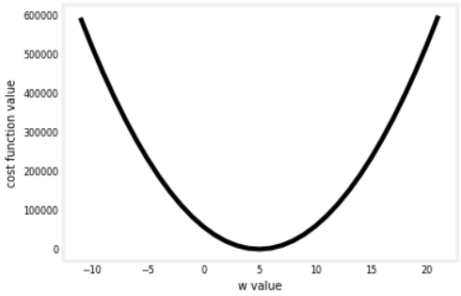
```
plt.scatter(height_value, weight_value)

plt.xlabel('Height(inches)')
plt.ylabel('Weight(Pounds)')
```

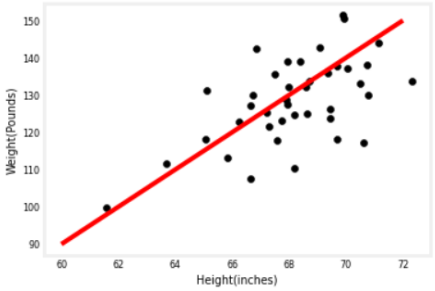
```
Text(0, 0.5, 'Weight(Pounds)')
```



[그림 2]

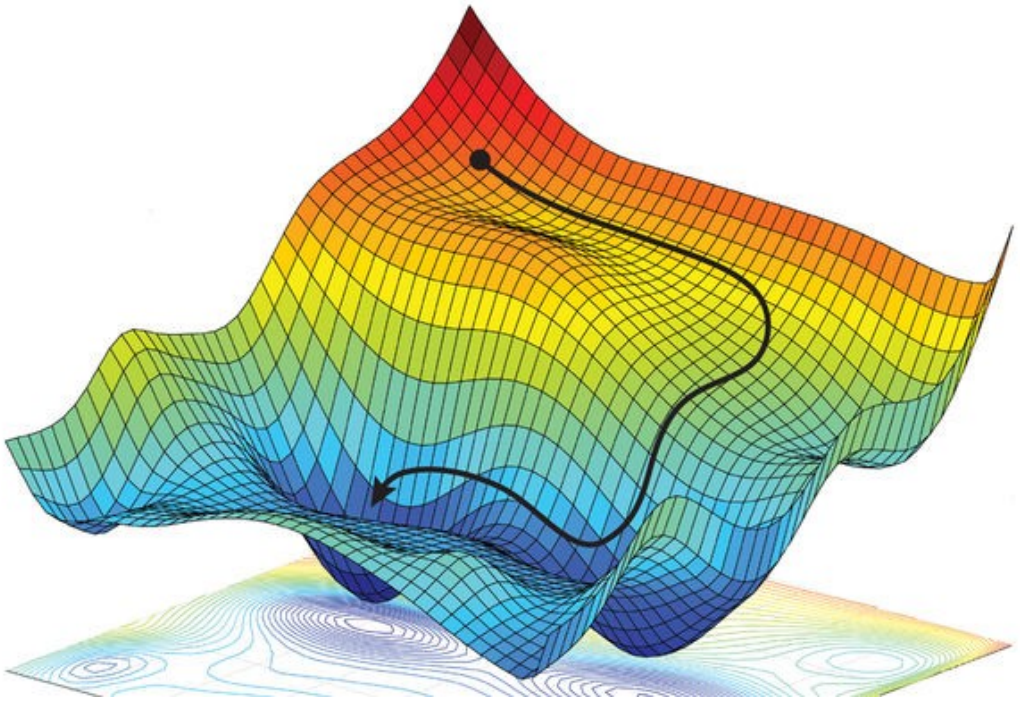
25000개의 키(인치)와 무게(파운드) 데이터셋을 로드하였지만[그림 1], 이로 Cost Function을 만들면 연산 처리가 비효율적이고 정확하지 않다는 점을 감안해 random모듈로 40개의 데이터를 뽑았다[그림 2]. 2차원 평면으로 구현하는 것이기에 $f_{w,b}(x^{(i)}) = wx^{(i)} + b$에서 b를 -210으로 잡았다. -11에서부터 22까지 w값이 대입되면 $J(w,\ b) = \frac{1}{2m}\sum_{i=0}^{m-1}\left(f_{w,\ b}(x^{(i)}) - y^{(i)}\right)^2$ m은 40, 그리고 $x^{(i)} - y^{(i)}$은 랜덤으로 생성한 수들을 집어넣었다. 그림 3과 같이 2차함수가 나왔고, 이 값들에 따르면 오차가 가장 적은 가설함수는 w가 5일 때 나온다. 이것을 그림 2에 대입하면 그림 4와 같다(빨간선).





[그림 3] [그림 4]

비록 구현은 2차원에서 되었지만, [그림 5]와 같이 b까지 구현한다면 더욱 고차원적인 그래프가 될 수 있다.

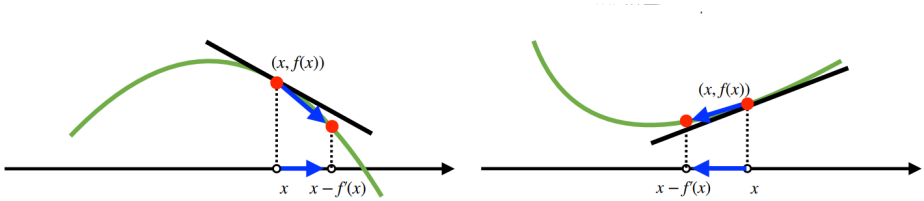


[그림 5]

경사하강법**(Gradient Descent)**

경사하강법은 비용함수에서 최저점(오차가 가장 적은 가설함수)를 찾기 위해 사용되는 기법이다. 주로 딥러닝에서는 단순 식을 풀어 계산하기에는 [그림 9]와 같이 복잡한 상황이 많이게 어떤 상황에서도 최저점을 찾을 수 있는 경사하강법을 주로 사용한다. 기본적으로 Gradient Descent에서는 미분값을 응용하여 극소값을 구할 수 있다. 기울기 부호의

반대방향으로 움직이게 된다면 최솟값을 찾아낼 수 있다. 기울기가 양수라는 것은 x값과 같이 함수의 값이 커진다는 것이기에 x가 작아지는 방향으로 움직이고, 기울기가 음수라는 것은 x값이 커질수록 f(x)값은 반대로 작아진다는 것이기에 x값이 커지는 방향으로 지속적으로 움직인다면 최솟값을 찾아낼 수 있다. 만약 기울기가 0이라면 그것은 local maximum일 것이다.



[그림 6]

수렴할때까지 아래의 공식을 반복한다:

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

이 상황에서는 w와 b가 지속적으로 업데이트되며 알파는 학습률이다, 기울기는 다음과 같다:

$$\frac{\partial J(w,b)}{\partial w} = \frac{1}{m}\sum_{i=0}^{m-1}(f_{w,b}(x^{(i)}) - y^{(i)})x^{(i)}$$

$$\frac{\partial J(w,b)}{\partial b} = \frac{1}{m}\sum_{i=0}^{m-1}(f_{w,b}(x^{(i)}) - y^{(i)})$$

학습률(Learning rate, Step size)는 가장 중요한 요소중 하나로 여겨진다. 만약 학습률이 지나치게 작을 경우 정확도는 높아질 수 있지만, 알고리즘이 반복해야 하는 값이 굉장히 많아져야 수렴할 수 있어 느린 속도로 인해 시간이 많이 걸린다. 반면에 학습률이 너무 크면 학습 시간 자체는 줄어들 수 있으나 정확도가 떨어지거나, 0으로부터 오차 범위가 적게 설정해놨다면 반대편으로 가 오히려 최소값에서 멀어질 수 있다.

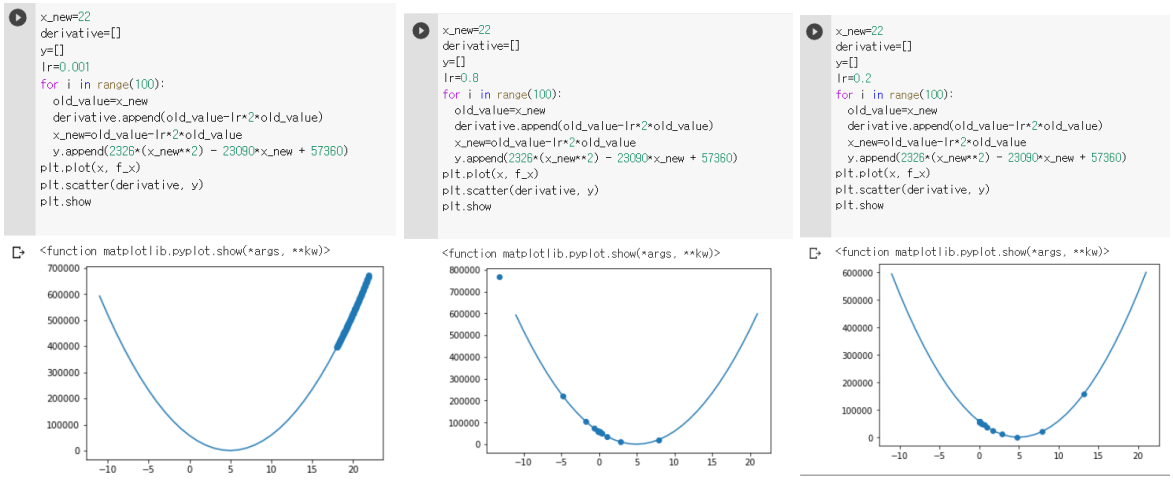
경사하강법의 문제점으로 Local Optima 문제가 있는데, 주변의 낮은 점을 따라 지속적으로 움직인다면 Local Minimum에 도달하지만 Local Maximum에는 도달하지 못할 수도 있다는 단점이 있지만, 위치가 다양한 곳에서 시작되기에 이 문제를 맞닥뜨릴 확률은 굉장히 낮다.

경사하강법**(Gradient Descent)**

위에서 임의의 값으로 고정시킨 b의 그래프의 공식을 구했을 때 다음과 같았다.

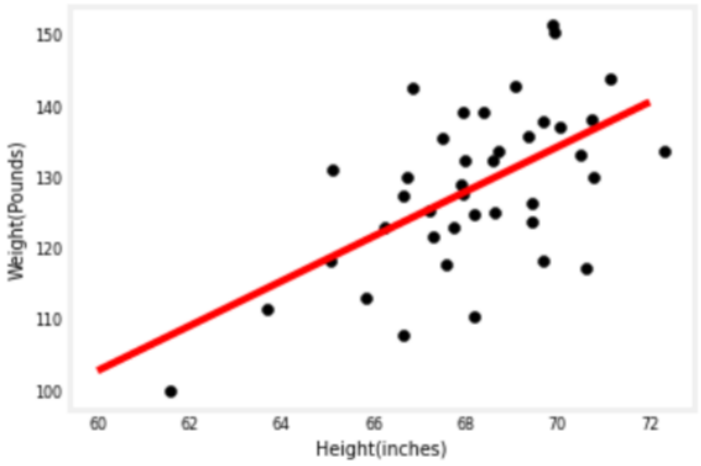
$$y = 2326x^2 - 2.309e^{+}04 * x + 5.736e^{+}04$$

이 그래프의 반복 횟수를 100으로 고정 시켜 놓고, 시작 점은 x=22, learning rate(코드에서는 lr)를 바꿔보며 확인하였을 때 그림7과 같이 learning rate가 0.001등 굉장히 작았을 때는 찾아내지 못하는 것을 보아 학습 시간이 굉장히 오래 걸리는 것을 알 수 있다. 반면에 learning rate를 0.8까지 올렸을 때는[그림 7] 오른쪽인 22에서 시작했음에도 불구하고 x=-10을 넘은 위치까지 점이 있는 것을 보아 발산하고 있는 것을 확인할 수 있다. 마지막으로 learning rate가 0.2였을 때는 적당히 움직이다 정확한 지점에 수렴하는 것을 알 수 있다.



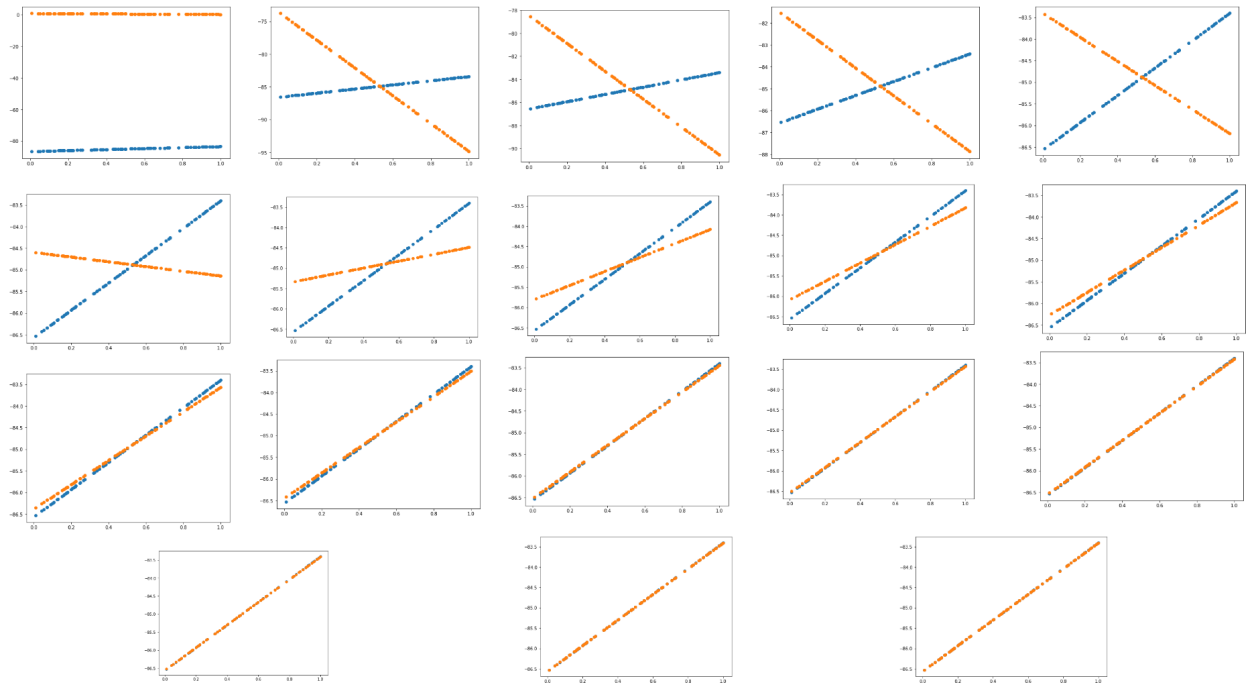
[그림 7]

위의 경우에서는 정확하게 수치화할 수 없다고 판단하였다. 그 이유는 애초에 b를 고정시켜놓고 w만 움직이는 것이기에 초반에 언급한 height와 weight사이의 cost function을 정확히 학습할 수 없기에 보다 정확하게 수치화하고자 하였다. [그림 4]를 들고 와서 이번에는 임의로 b값을 지정하는 대신, 파이썬 scikit-learn에서 제공하는 LinearRegression모델을 들고와 정확한 b와 w 값을 찾았다. 일차식은 $y = 3.15491183x - 86.558972529651 56$이고, 시각화 하면 [그림 8]과 같다.



[그림 8]

b와 w를 모두 다 Gradient Descent를 이용하여 구현하고자 하였고, 아래의 파란색 그래프가 일차식, $y = 3.15491183x - 86.558972529651 56$이고, 노란색 그래프가 경사하강법을 통해 그래프를 따라가는 것을 구현하였다(위 모델과 달리 전체적인 경사하강법 흐름은 3d로 구현해야 하지만 노트북 성능상 불가능하였다).



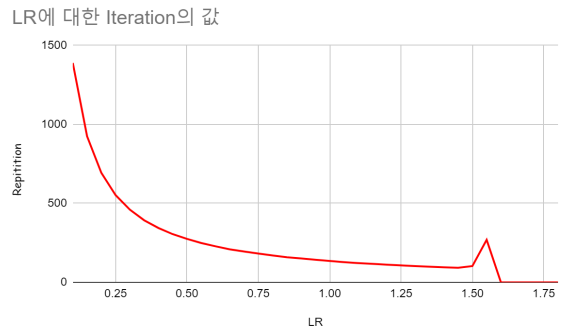
[그림 9]

[그림 10]에 보이는 것과 같이 각 학습률 마다 반복해야하는 step의 개수를 세었다. 위의 연구는 정확도에 관한 것이었다면, 이는 시간 효율성을 보다 직관적으로 보이기 위하여 정확도는 아예 배제하였다. Minimum에 도착하면 slope가 0이 되어야 하는데, 여기서 오차범위가 0.1 이내면 반복문이 끝나도록 하여 오차범위 안에 영원히 들어가지 않으면 None으로 표시되게 하였다. 각 반복문마다 사용되는 시간은 똑같기에 사실상 반복 횟수가 시간을 나타내는 것이다. Step Size가 0.1이었을 때 반복 횟수는 1389번이었던 반면에 Step size가 늘어날수록 점차 줄어들다 1.4일때는 92까지 찍는다. 하지만 이 이후에 조금 증가하다가 완벽히 발산한다[그림 11]. 일단 step size가 늘어날수록 시간 소요도는 점차 줄어드는 것은 전형적인 경사하강법의 원리이다. 하지만 끝나고 잠시 증가하는 이유는 지나치게 Learning Rate가 커져 그래프의 양 끝을 미세한 차이로 반복운동을 하여 시간이 늘어나는 것이다. 그리고 1.6 이후에는 지나치게 커져 그래프 밖을 나가게 되고, 이는 곧 발산으로 이어진다.

LR	Repetition	LR	Repitition
0.1	1389	1	135

0.15	925	1.05	128
0.2	693	1.1	122
0.25	552	1.15	117
0.3	460	1.2	112
0.35	393	1.25	107
0.4	344	1.3	103
0.45	305	1.35	99
0.5	275	1.4	95
0.55	249	1.45	92
0.6	228	1.5	103
0.65	209	1.55	269
0.7	195	1.6	0
0.75	182	1.65	0
0.8	170	1.7	0
0.85	159	1.75	0
0.9	151	1.8	0
0.95	143		

[그림 10]



[그림 11]

한계 및 의의

이 연구를 통해 경사하강법의 학습률과 학습 정확도, 그리고 학습 속도에 대한 명확한 상관관계에 대해 알 수 있었을 뿐만 아니라 그를 수치화하여 보다 높은 이해를 할 수 있었다. 적절한 learning rate와 step size를 섞어가며 지나치게 적은 수치를 넣어 시간이 오래 걸리거나

지나치게 높은 수치를 넣어 오히려 시간이 증가하거나 발산하는 것을 막기 위하여 여러번의 시행착오를 통해 정확한 learning rate를 찾아야할 것이다.

이 연구 자체가 저성능의 노트북에서 행해져 높은 수준의 시각화나 오랫동안 반복문을 사용하지 못하는 점과 같이 한계는 여전히 존재하였다. 뿐만 아니라 미래 연구에서는 공식 등을 통해 정확한 correlation를 밝혀낼 수 있을 것 같다.

출처

Ruder, Sebastian. (2016). An overview of gradient descent optimization algorithms.

강민제.(2020).딥러닝을 위한 경사하강법 비교.한국산학기술학회 논문지,21(2),189-194.

Chatzimichailidis, Avraam, etal. "Gradvis: Visualization and second order analysis of optimization surfaces during the training of deep neural networks." 2019 IEEE/ACM Workshop on Machine Learning in High Performance Computing Environments (MLHPC). IEEE, 2019.

Becker, Martin & Lippel, Jens & Zielke, Thomas. (2019). Gradient Descent Analysis -On Visualizing the Training of Deep Neural Networks. 338- 345. 10.5220/0007583403380345.

https://towardsdatascience.com/coding-deep-learning-for-beginners-linear-regression-gradient-descent-fcd5e0fc077d