DL 스터디&프로젝트

[Euron 중급 세션 10주차] 딥러닝 2단계 4. 최적화 알고리즘

by 공부하자_ 2023. 11. 13.

딥러닝 2단계: 심층 신경망 성능 향상시키기

4. 최적화 알고리즘

📌미니 배치 경사 하강법(C2W2L01)

핵심어: 미니배치(Mini Batch)

신경망을 더 빠르게 학습하도록 하는 최적화 알고리즘에 대해 학습할 것. 머신러닝을 적용하는 것은 매우 실험적인 과정으로, 잘 작동되는 모델을 찾기 위해 많은 훈련을 거쳐야 하는 반복적인 과정이다. 따라서 모델을 빠르게 훈련시키는 것이 중요하다. 딥러닝은 빅데이터에서 가장 잘 작동된다는 것도 큰 데이터 세트에서 신경망을 훈련시킬 때는 이 어렵게 만든다. 큰 데이터 세트에서 훈련하는 것은 매우 느린 과정이다. 따라서 좋은 최적화 알고리즘을 찾는 것은 일의 효율성을 좋게 만들어줄 것이다. 미니배치 경사 하강법에 대해 학습해보자.

분류 전체보기

DL 스터디&프로젝트

Data Science 프로젝트

Github 스터디

Data Science 개인 공부

Backend 프로젝트

기타 공부

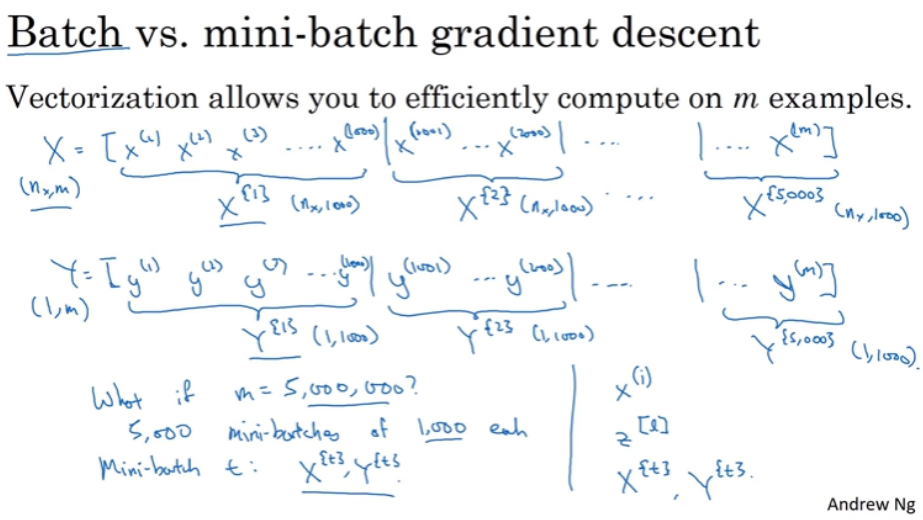
공지사항

최근글 인기글

[Euron 중급 세션 10주차]... 2023.11.13

백엔드 프로젝트 8주차 스... 2023.11.11

[Euron 중급 세션 9주차] 딥러닝 2단계 3…



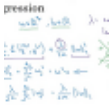
이전에 벡터화가 m개의 샘플에 대한 계산을 효율적으로 만들어 준다는 것에 대해 배웠는데, 이는 명시적인 반복문 없이도 훈련 세트를 진행할 수 있도록 하는 것이다. 이를 위해 훈련 샘플(x1~xm)을 받아서 큰 행렬(X)에 저장한다(Y에도 똑같은 작업). 이때 X의 차원은 (n_x, m)이고, Y는 (1, m)이다. 벡터화는 m개의 샘플을 상대적으로 빠르게 훈련시킬 수 있지만, m의 크기가 매우 크다면 여전히 느릴 수 있다. 예를 들어 m이 5000000이상일 경우를 생각해보자. 전체 훈련 세트에 대해 경사 하강법을 구현하면, 경사 하강법의 작은 한 단계를 밟기 저네 모든 훈련 세트를 처리해야 한다. 또 경사 하강법의 다음 단계를 밟기 전에 다시 오백만 개의 전체 훈련 샘플을 처리해야 한다. 따라서 오백만 개의 거대한 훈련 샘플을 모두 처리하기 전에 경사 하강법이 진행되도록 하면 더 빠른 알고리즘을 얻을 수 있다.

훈련 세트를 더 작은 훈련 세트들로 나누었다고 했을 때, 이러한 작은 훈련 세트는 미니배치라고 부른다. 각각의 미니배치가 1000개의 샘플을 갖는다고 하면, 첫 번째 작은 훈련세트(미니배치)에 x1~x1000까지 얻고 다음 1000개의 샘플인 x1001~x2000까지 얻고.. 이 작업을 계속 반복한다. 여기서 하나의 미니배치를 X{1}, X{2} 이런 식으로 표기할 수 있다(x(i)는 훈련샘플, z[l]는 신경망의 층). 총 오백만개의 훈련 샘플이 있고 각각의 미니배치는 천 개의 샘플이라면 오천 개의 미니배치가 있다는 것이 되며 X{5000}이 마지막이 된다. Y에 대해서도 비슷한 방식으로 훈련 데이터를 Y{1} ~ Y{5000}으로 나눈다. 여기서 미니배치의 개수 t는 X{t}, Y{t}로 나타내며 입력 출력 쌍에 대응하는 천 개의 훈련 샘플이다. 이때 X{t}와 Y{t}의 차원에 대해 말해보자면, X는 (n_x, m) 차원이며 X{1}이 천 개의 훈련 샘플이라면 X{1}는 (n_x, 1000)차원, X{2} 역시 (n_x, 1000) 차원이 되어야 한다. 이런 식으로 모든 미니배치의 차원은 (n_x, 1000)이 되어야 한다. Y{1}차원 마찬가지로 (1, 1000)이 되어야 한다.

이 알고리즘의 이름에 대해 설명하자면, 배치 경사 하강법은 우리가 전에 이야기해온 경사 하강법을 말하며 즉 모든 훈련 세트를 동시에 진행시키는 방법이다. 이 이름은 동시에 훈련 샘플의 모든 배치를 진행시킨다는 관점에서 나왔다. 이와 반대로 미

2023.11.06

백엔드 프로젝트 7주차 스...
2023.11.04

[Euron 중급 세션 8주차] ...
2023.10.30


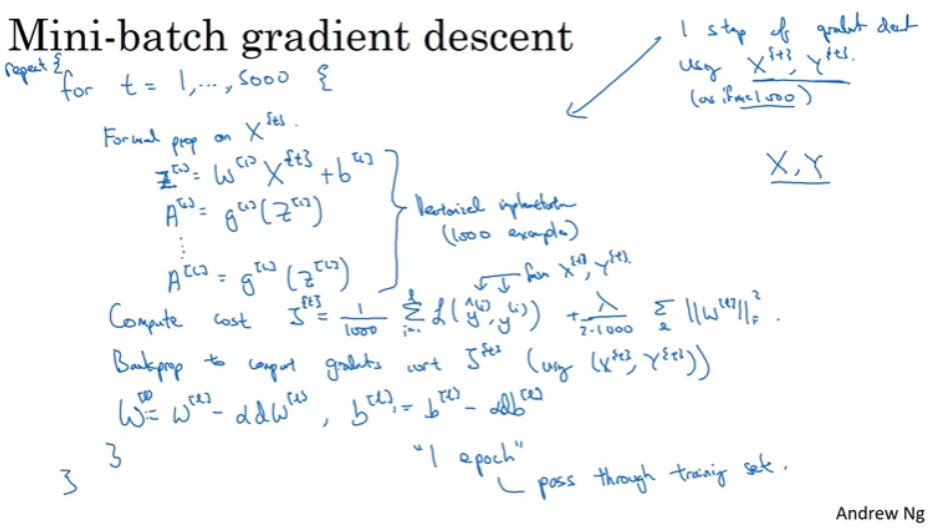
최근댓글

좋은 글 잘 보고 가요! 감사...
좋은 글 잘 보고 가요! 감사...
잘보고 갑니다!
포스팅 잘보고 갑니다! 응원...

태그

데이터분석, 데이터사이언스, 판다스입문, 판다스, bda, 딥러닝스터디, 이지스퍼블리싱, 딥러닝교과서, pandas, Doit

니 배치 경사하강법은, 이후에도 나오겠지만, 전체 훈련 세트 X, Y를 한 번에 진행시키지 않고 하나의 미니배치 X{t}, Y{t}를 동시에 진행시키는 알고리즘을 말한다.



미니배치 경사하강법이 어떻게 작동하는지 알아보자. 여러분의 훈련 세트에서 미니배치 경사 하강법을 실행하기 위해서는, t=1~5000 반복문을 돌려야 한다(각각의 크기가 1000인 미니배치가 5000개 있기 때문). 반복문 안에서 하려고 하는 것은 기본적으로 X{t}와 Y{t}를 사용하여 한 단계의 경사하강법을 구현하는 것이다. 1000개의 샘플이 있는 훈련 세트는, 이미 익숙한 알고리즘을 구현하는 것이지만, 단지 m이 1000인 작은 훈련 세트에서 구현하는 것일 뿐이다. 모든 1000개의 샘플에 대해 명시적인 반복문을 작성하는 것보다 벡터화를 사용해 모든 1000개의 샘플을 동시에 진행시키게 되는 것이다.

코드를 작성해보자면 첫째, 입력 X{t}에 대해 정방향 전파를 구현한다. 이를 위해 z[1]=W[1]x를 구현하는데 이전에는 x를 곱해주었으나 전체 훈련 세트를 진행하는 것이 아닌 첫 번째 미니배치를 진행하는 것이기 때문에 X{t}를 곱해주게 된다. 그리고 A[1]은 g[1](Z[1])(벡터화된 구현이므로 대문자 Z로 써줌)이다. 이런 식으로 A[L]=g[L](Z[L])까지 진행하며 이것이 예측이 된다. 여기에 벡터화된 구현을 사용해야 하는데, 이 벡터화된 구현이 동시에 오백만개의 샘플 대신에 1000개의 샘플을 진행하기 때문이다.

다음으로 비용함수 J를 계산해보자면, 작은 훈련 샘플의 크기가 1000이므로 1/1000으로 작성하고 여기에 i가 1부터 l까지 y(i)의 예측값과 y(i)를 매개변수로 하는 손실함수 L의 합을 곱해준다. 여기서 y(i)의 예측값과 y(i) 매개변수는 X{t}, Y{t}가 된다. 정규화를 사용한다면 λ/2*1000*(||w[l]||i~2) 정규화 항을 할 수도 있다. 이때 이는 하나의 미니배치에 대한 비용이기 때문에 J{t}라고 표기한다. 모든 것은 전에 했던 경사 하강법을 구현하는 것과 정확히 같지만, X, Y 대신에 X{t}, Y{t}를 사용한다는 점에서 명백히 다르다.

다음은 역전파를 구현해보도록 하자. J{t}에 대응하는 경사를 계산하기 위한 것이다. 여전히 X{t}와 Y{t}를 사용하며, 가중치를 모두 W[l]은 W[l] - αdW[l]로 업데이트 하며 b도 비슷한 방식으로 업데이트 한다. 따라서 이것은 미니 배치 경사 하강법을 사

용한 훈련 세트를 지나는 한 번의 반복이다. 위 코드는 훈련의 한 에포크를 거친다고도 말할 수 있다. 여기서 에포크란 훈련 세트를 거치는 한 반복을 의미한다. 따라서 배치 경사 하강법에서 훈련 세트를 거치는 한 반복은 오직 하나의 경사 하강 단계만을 할 수 있게 하며, 미니배치 경사 하강법의 경우 훈련 세트를 거치는 한 반복은 5000개의 경사 하강 단계를 거치도록 한다. 또 다른 반복문을 사용해서 훈련 세트를 여러 번 거치면, 원하는 만큼 계속 거의 수렴할 때까지 훈련 세트를 계속 반복시키게 된다.

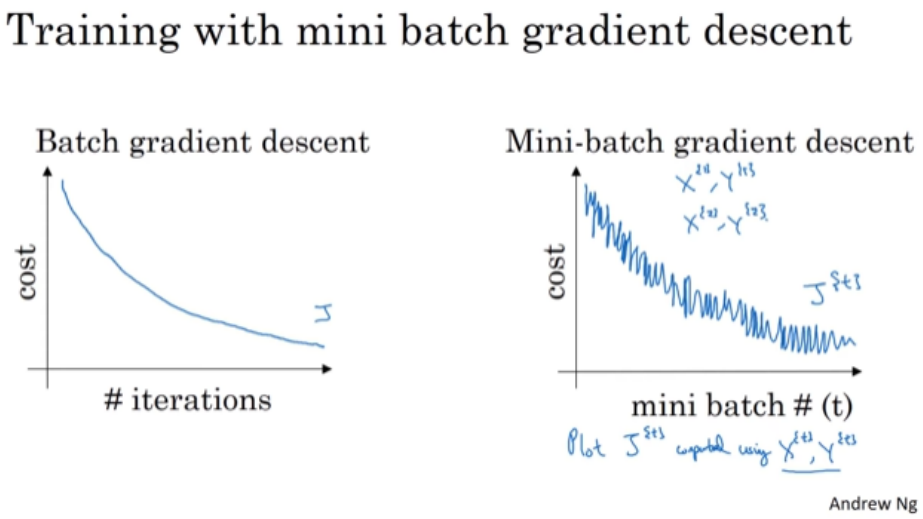
훈련 세트가 많다면 미니배치 경사하강법이 배치 경사하강법보다 훨씬 더 빠르게 실행되며 따라서 많은 데이터 세트를 훈련 시킬 때 거의 모든 사람들이 사용하는 방법이다.

- 배치 경사 하강법
- 전체 훈련 샘플에 대해 훈련 후 경사 하강 진행
- 미니배치 경사 하강법
- 전체 훈련 샘플을 작은 훈련 세트인 미니배치로 나눈 후, 미니배치 훈련 후 경사 하강 진행
- 배치 경사 하강법은 큰 데이터 세트를 훈련하는데 많은 시간이 들기에 결과적으로 경사 하강을 진행하기까지 오랜 시간이 걸립니다. 따라서 작은 훈련 세트인 미니배치로 나누어 훈련 후 경사 하강을 진행합니다.
- 예를 들어 전체 훈련 세트 크기가 5,000,000이라고 할 때 이를 사이즈가 1,000인 미니배치 5,000개로 나누어 훈련 및 경사 하강법을 진행합니다.
- 표기법
  - i번째 훈련 세트 : $x^{(i)}$
  - l번째 신경망의 z값 : $z^{[l]}$
  - t번째 미니배치 : $X^{\{t\}}, Y^{\{t\}}$

📌미니 배치 경사 하강법 이해하기(C2W2L02)

핵심어: 미니배치(Mini Batch)

이전 강의에서 처음 훈련 세트를 훈련하는 도중에라도 경사 하강의 단계를 진행시키기 위해 미니배치 경사 하강법을 어떻게 사용하는지에 대해 배웠다. 이번에는 경사 하강법에 대해 더 자세히 배우고, 이것이 무엇이고 왜 잘 작동되는지에 대한 이해를 도울 것이다.



배치 경사 하강법에서는 모든 반복에서 전체 훈련 세트를 진행하고 각각의 반복마다 비용이 감소하기를 기대한다. 비용함수 J를 서로 다른 반복에 대한 함수로 그렸을 때, 모든 반복마다 감소해야 하며 하나라도 올라간다면 무언가 잘못된 것이다. 반면 미니배치 경사 하강법에서 비용함수에 대한 진행을 그려보면, 모든 반복마다 감소하지는 않는다. 특히 모든 반복에서 어떤 X{t}와 Y{t}를 진행시키는데 비용함수 J{t}를 그려본다면 X{t}와 Y{t}만을 이용해 계산된 값이다. 그럼 모든 반복에서 다른 훈련 세트, 즉 다른 미니배치에서 훈련한다는 것이 된다. 따라서 비용함수 J를 그리면 전체적인 흐름은 감소하나 약간의 노이즈가 발생하는 모양이 된다. 미니배치 경사 하강법을 훈련하기 위한 J{t}를 그리면 여러 에포크를 거쳐서 위와 같은 곡선을 보게 될 것이며, 모든 반복에서 아래로 내려가지는 않는다. 그러나 큰 흐름은 아래로 내려가는 것이다. 약간의 노이즈가 발생하는 이유는 아마 X{1}과 Y{1}이 상대적으로 쉬운 미니배치라서 비용이 약간 낮은데, 우연적으로 X{2}와 Y{2}가 더 어려운 미니배치라서 비용이 약간 더 높아질 수 있다. 이러한 이유로 미니배치 경사 하강법을 실행시킬 때 진동이 발생할 수 있는 것이다.

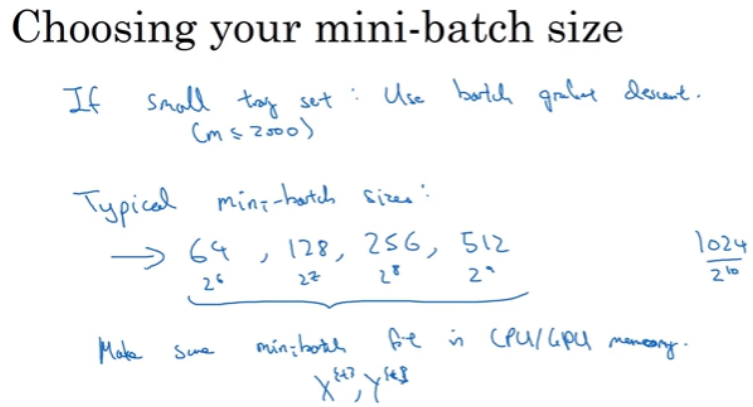
Understanding Mini-Batch Gradient Dexcent 원본보기

우리가 선택해야 할 매개변수 중 하나는 미니배치의 크기이다. m이 훈련 세트의 크기일 때 극단적인 경우 미니배치 크기가 m과 같은 경위다. 그럼 이것은 배치 경사 하강법이 된다. 이런 극단적인 경우, 하나의 미니배치 X{1}과 Y{1}만을 갖게되며 이 미니배치의 크기는 전체 훈련 세트와 같다. 따라서 미니배치 크기를 m으로 설정하는 것은 일반저긴 경사 하강법과 같다. 다른 극단적인 경우는 미니배치 크기가 1과 같은 경우이다. 이것은 확률적 경사 하강법이라고 불리는 알고리즘을 제공하며, 각각의 샘플은 하나의 미니배치이다. 이런 경우 첫 번째 미니배치인 X{1}과 Y{1}을 살펴보면, 미니배치 크기가 1일 때 이것은 첫 번째 훈련 샘플과 같다. 첫 번째 훈련 샘플로 경사 하강법을 하는 것이다. 그 다음 두 번째 미니배치를 살펴보면 이는 두 번째 훈련 샘플과 같고 이것에 대한 경사 하강 단계를 취한다. 이런 식으로 한 번에 하나의 훈련 샘플

만을 살펴보며 계속 진행한다. 이제 이 두 극단적인 경우가 비용함수를 최적화할 때 무엇을 하는지 살펴보자.

만약 왼쪽 아래 그림이 우리가 최소화하고자 하는 비용함수의 등고선이라면, 최솟값은 중앙에 위치한다. 그럼 배치 경사 하강버븐 어딘가에서 시작해 상대적으로 노이즈가 적고 상대적으로 큰 단계를 취해 계속 최솟값으로 나아간다. 그와 반대로 확률적 경사 하강법에서 어딘가에서 시작하면 모든 반복에서 하나의 훈련 샘플로 경사 하강법을 실행하게 된다. 대부분의 경우 전역 최솟값으로 가게 되지만 어떤 경우 이처럼 잘못된 방향을 가리켜 잘못된 곳으로 가기도 한다. 따라서 확률적 경사 하강법은 극단적으로 노이즈가 많을 수 있지만 평균적으로는 좋은 방향으로 가게 된다(잘못된 방향일수도 있지만). 따라서 확률적 경사 하강법은 절대 수렴하지 않을 것이다. 진동하면서 최솟값의 주변을 돌아다니지만 최솟값으로 곧장 가서 머물지는 않는다는 것이다.

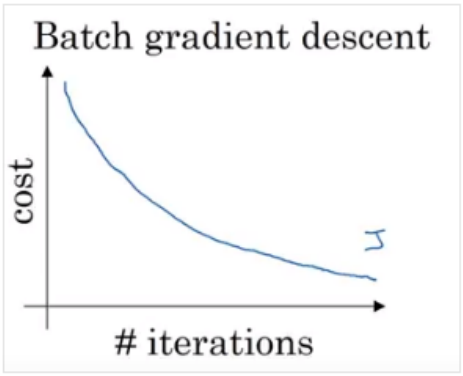
실제로 우리가 사용하는 미니배치의 크기는 1과 m 사이일 것이다. 1은 상대적으로 너무 작고 m은 상대적으로 너무 큰 값인데, 여기에 그 이유가 있다. 배치 경사 하강법을 사용한다면 미니배치의 크기는 m과 같다. 그럼 매우 큰 훈련 세트를 모든 반복에서 진행하게 된다. 이것의 주된 단점은 한 반복에서 너무 시간이 오래 걸린다는 점이다. 작은 훈련세트에서는 괜찮지만 큰 훈련 세트에서는 오랜 시간이 필요하다. 그와 반대로 확률적 경사 하강법을 사용한다면, 하나의 샘플만 처리한 뒤 계속 진행할 수 있어서 매우 간단하다. 노이즈도 작은 학습률을 사용해 줄일 수 있다. 그러나 확률적 경사 하강법의 큰 단점은 벡터화에서, 얻을 수 있는 속도 향상을 잃게된다는 것이다. 한 번에 하나의 훈련 세트를 진행하기 때문에 각 샘플을 진행하는 방식이 매우 비효율적이라는 것이다. 따라서 가장 잘 작동하는 것은 1과 m 사이에 있는 값이 된다. 미니배치 크기가 너무 크거나 작지 않을 때이다. 그리고 실제로 이것은 가장 빠른 학습을 제공한다. 이를 통한 두 가지 장점이 있는데, 하나는 많은 벡터화를 얻는다는 것이다. 이전에 들었던 예시를 생각해보면, 미니배치의 크기가 1000개인 샘플이라면 1000개의 샘플에 벡터화를 하게 될 것이며 그럼 한 번에 샘플을 진행하는 속도가 더 빨라지게 된다. 두 번째, 전체 훈련 세트가 진행되기를 기다리지 않고 진행을 할 수 있다. 또다시 이전에 들었던 예를 생각해보면, 각각의 훈련 세트의 에포크는 5000번의 경사 하강 단계를 허용한다. 실제로 사이에 있는 미니배치 크기가 잘 작동하며, 여기서 시작한 미니배치라면 한 번의 반복, 두 번의 반복마다 진행된다.항상 솟값으로 수렴한다고 볼 수는 없지만 더 일관되게 전역의 최솟값으로 향하는 경향이 있다. 그리고 매우 작은 영역역에서 항상 정확하게 숨어들고 수렴하거나 진동하게 된다. 그렇다면 미니배치 크기가 m이나 1이 아닌 그 사이의 값이어야 한다면, 이때 값을 선택하는 가이드라인이 존재한다.



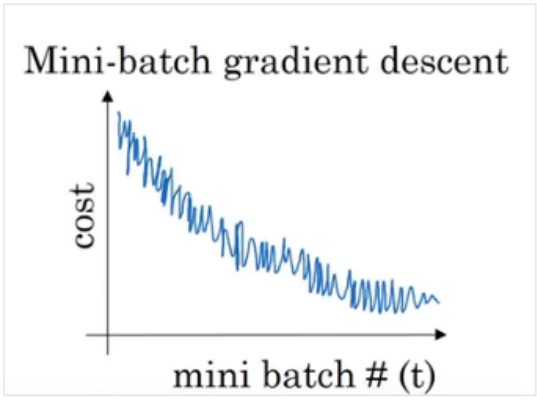
첫 번째로, 작은 훈련 세트라면 그냥 배치 경사 하강법을 사용하는 것이 좋다. 훈련 세트가 작다면, 미니배치 경사 하강법을 사용할 필요 없이 전체 훈련 세트를 빠르게 진행할 수 있다. 여기서 작은 훈련 세트라는 것은 2000개보다 적은 경우를 말한다. 그런 경우라면 배치 경사 하강법을 쓰는 것이 더 좋다.

이와 달리 더 큰 훈련 세트라면 전형적인 미니배치 크기는 64에서 512 사이가 가장 일반적이다. 이는 컴퓨터 메모리와 관련이 있으며 2의 거듭제곱으로 사용하는 것이 좋다. 미니배치 크기로 1024는 조금 드문 경우인데, 64~512 범위의 미니배치 크기가 더 일반적이라고 할 수 있다.

마지막 팁은 미니배치에서 모든 X{t}와 Y{t}가 CPU와 GPU 메모리에 맞는지 확인해야 하는 것이다. CPU나 GPU 메모리에 맞지 않는 미니배치를 진행시키면 성능이 갑자기 떨어지고 훨씬 나빠지게 된다. 실제로 미니배치의 크기는 빠른 탐색을 통해 찾아내야 하는 또 다른 하이퍼파라미터이다. 가장 효율적이면서 비용함수 J를 줄이는 값을 찾아야 하는 것이다. 좋은 방법은 몇 가지 다른 2의 제곱수를 시도해보고, 경사 하강법 최적화 알고리즘을 가능한 가장 효율적이게 만드는 값을 선택하는 것이다.



- 배치 경사 하강법에서는 한 번의 반복을 돌 때마다 비용 함수의 값은 계속 작아져야 합니다.

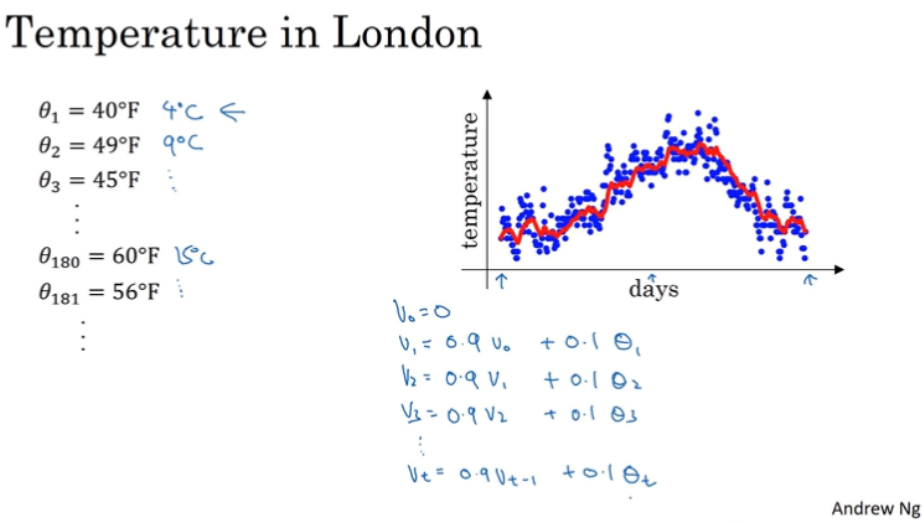


- 미니배치 경사 하강법에서는 전체적으로 봤을때는 비용 함수가 감소하는 경향을 보이지만 많은 노이즈가 발생합니다.
- 미니배치 사이즈를 어떻게 선택하는지에 따라 학습 속도의 차이가 나기에 최적의 값을 찾아내는 것이 중요합니다.
- 만약 훈련 세트가 작다면 (2,000개 이하) 모든 훈련 세트를 한 번에 학습시키는 배치 경사 하강을 진행합니다.
- 훈련 세트가 2,000개 보다 클 경우 전형적으로 선택하는 미니배치 사이즈는 64, 128, 256, 512와 같은 2의 제곱수입니다.

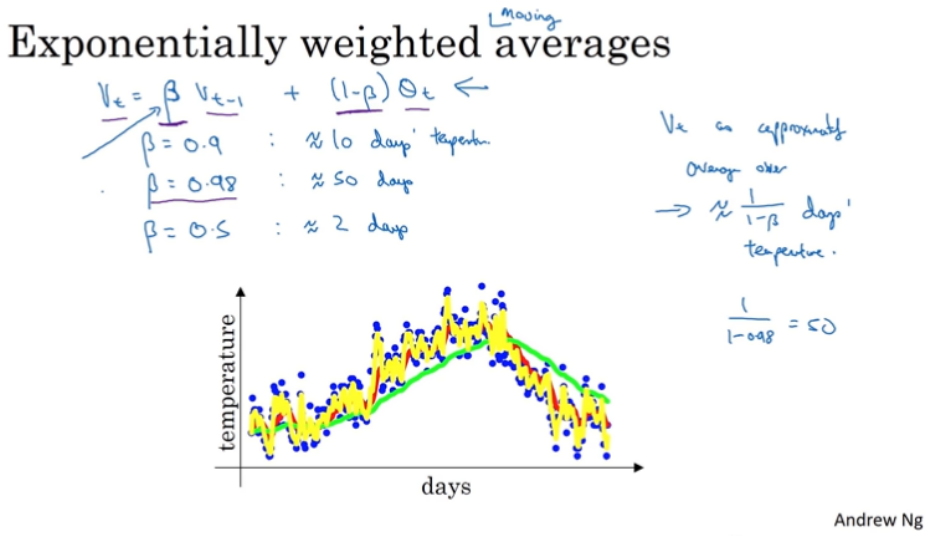
📌지수 가중 이동 평균(C2W2L03)

핵심어: 지수 가중 이동 평균(Exponentially Weighted Average)

경사하강법보다 빠른 몇 가지 최적화 알고리즘이 존재한다. 그 알고리즘들을 이해하기 위해서는 지수가중평균을 사용할 수 있어야 한다(통계학에서는 지수가중이동평균이라고도 불림). 먼저 이것에 대해 알아보고 가장 정교한 최적화 알고리즘을 구현하는 데 사용할 것이다.



런던의 일별 기온을 데이터셋으로 사용해보자. 데이터를 그려보면 오른쪽 그림과 같으며, 1월에 시작해서 12월에 끝난다. 이 데이터는 약간의 노이즈가 있는데, 여기서 지역 평균이나 이동 평균의 흐름을 계산하고 싶다면 이런 방법을 사용할 수 있다. v0를 0으로 초기화하고, 매일 이전 값에 0.9를 곱한 후 0.1*(해당 날의 기온)을 더해줄 것이다(θ1은 1월 1일의 기온) .이 값은 v1이 된다. 둘째 날의 가중 평균 역시 v2 = (이전 날의 값=v1)*0.9 + 0.1* θ2로 계산할 수 있다. 이를 반복하여 더 일반적인 식은 vt = 0.9*v(t-1) + 0.1θt. 이를 계산해 그림으로 나타내면 빨간색 그래프가 되며 이는 일별 기온의 지수가중균을 얻은 것이다.



이전 슬라이드의 공식을 다시 살펴보자. 이전에 쓰던 0.9를 β, 0.1은 1-β로 표현한다면, vt = β*v(t-1)+(1-β)θt가 된다. 여기서 vt를 계산한 값은 대략적으로 (1/(1-β))*(일별 기온의 평균)과 같다. 예를 들어 β가 0.9인 경우 이는 10일 동안 기온의 평균과 같은 것이다.

β가 1과 매우 가까운 0.98의 경우, 1/(1-0.98)은 50과 비슷하므로 이 값은 50일의 기온의 평균과 거의 같다. 이를 그리면 초록색 선과 같다. 여기서 알 수 있는 것은, β 값이 클수록 선이 더 부드러워진다는 것인데, 이는 더 많은 날짜의 기온의 평균을 이용하기 때문이다. 그러나 곡선이 올바른 값에서 더 멀어지는데, 이는 더 큰 범위에서

기온을 평균하기 때문이다. 그래서 기온이 바뀔 경우, 지수가중평균 공식은 더 느리게 적응하며 지연되는 시간이 더 크다. 그 이유는 예를 들어 β가 0.98이면 이전 값에 많은 가중치를 주고 현재 기온에 작은 가중치를 주게 되는데, 따라서 기온이 올라가거나 내려가면 이 지수가중평균은 β가 커서 더 느리게 적응하게 되는 것이다.

또 다른 극단적인 예시로 β가 0.5라면 위의 공식에 의해 2일의 기온만 평균하는 것과 같게 된다. 그럼 다음과 같은 노란색 그래프를 얻게 된다. 오직 2일의 기온만을 평균했기 때문에 더 노이즈가 많고 이상치에 더 민감하다. 그러나 기온 변화에 더 빠르게 적응한다. **vt = β*v(t-1)+(1-β)θt** 공식은 지수가중평균을 구현하기 위한 공식이며, β 매개변수 또는 학습 알고리즘의 하이퍼파라미터 값을 바꿈으로써 약간씩 다른 효과를 얻게 되고 이를 통해 가장 잘 작동하는 값을 찾게 된다. 예를 들어 위 그림에서, 빨간색 그래프는 초록색이나 노란색 곡선보다 더 나은 평균을 제공한다.

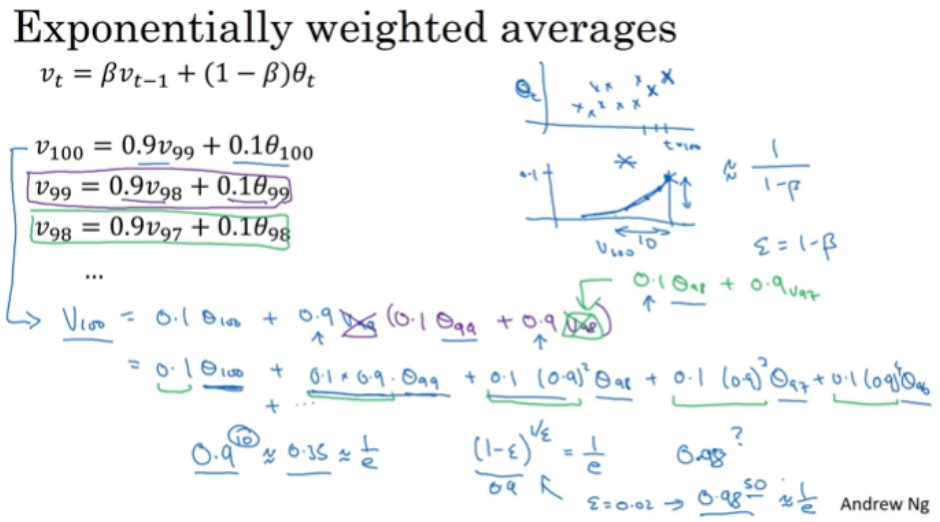
- 앞 강의에서 배운 경사 하강법 및 미니배치 경사 하강법보다 더 효율적인 알고리즘을 이해하기 위해 지수 가중 이동 평균을 먼저 이해해야 합니다.
- 최근의 데이터에 더 많은 영향을 받는 데이터들의 평균 흐름을 계산하기 위해 지수 가중 이동 평균을 구합니다. 지수 가중 이동 평균은 최근 데이터 지점에 더 높은 가중치를 줍니다.
- θ_t 를 t 번째 날의 기온이라고 했을 때, 지수 가중 이동 평균(v_t)의 식은 다음과 같습니다.
- $v_t = \beta v_{t-1} + (1-\beta)\theta_t$
- 이 때 β 값은 하이퍼 파라미터로 최적의 값을 찾아야 하는데, 보통 사용하는 값은 0.9 입니다. (이에 대한 이유는 향후 배우게 될 것입니다.)
- v_t 는 $\frac{1}{1-\beta}$ 기간 동안 기온의 평균을 의미합니다.
  - β = 0.9일 때 10일의 기온 평균
  - β = 0.5일 때 2일의 기온 평균

📌 지수 가중 이동 평균 이해하기(C2W2L04)

핵심어: 지수 가중 이동 평균(Exponentially Weighted Average)

지수 가중 평균은 신경망 훈련에 사용하는 몇 가지 최적화 알고리즘의 주요 구성요소가 될 것이다. 이번에는 이 알고리즘이 하는 일에 대해 더 깊게 알아보고자 한다.

vt = β*v(t-1)+(1-β)θt 공식이 지수가중평균을 구현하는데 주요한 공식이라는 것을 학습했었다. 위 그림에서 β가 0.9와 같은 경우 빨간색 곡선을 얻게 된다. 1에 가까운 0.98의 경우 초록색 곡선을, 0.5로 작은 값의 경우 노란색 곡선을 얻게 된다. 일일 기온의 평균을 계산하는 방법에 대해 수학적으로 더 알아보자.

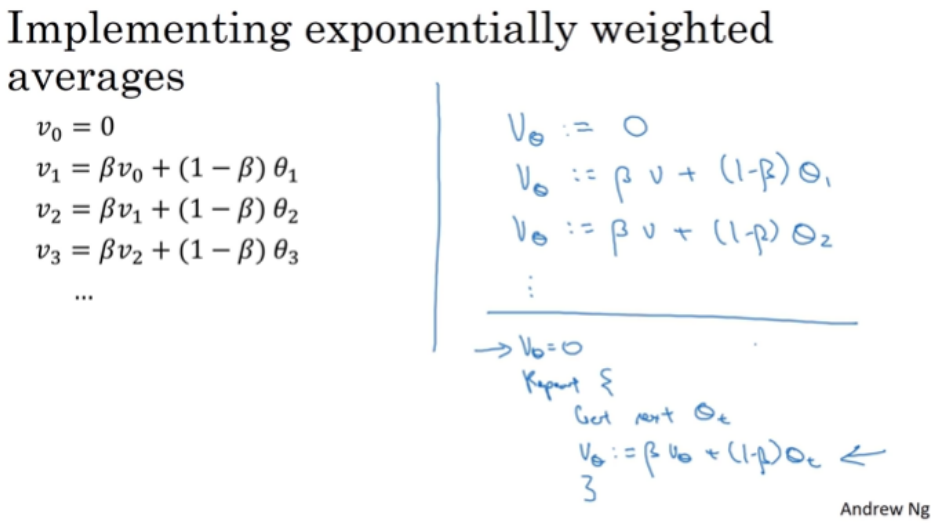


여기 공식에서 β를 0.9로 설정하고, 이를 이용해 몇 가지 식을 더 작성해보았다. 이를 구현할 때 t는 0부터 1, 2, 3, ...으로 증가하지만, 이번에는 분석을 위해 t가 감소하는 방향으로 식을 작성하였다. 첫 번째 식 v100 = 0.9*v99 + 0.1*θ100 (t=100)을 살펴보자. v100의 의미를 알기 위해 두 항의 위치를 바꿔 v100 = 0.1*θ100 + 0.9*v99 으로 식을 써준다. 마찬가지로 v99의 식은 두 항의 값을 바꾼 0.1*θ99 + 0.9*v98이 된다. v98역시 이 식을 대입할 수 있으며, 그 뒤로도 이런 식으로 계산된다. 이 모든 항을 곱하면 v100 = 0.1*θ100 + 0.1*0.9*θ99 + 0.1*0.9*0.9*θ98 + 0.1*0.9*0.9*0.9*θ97 ... 이런 식으로 확장되며 이것이 가중치의 합, 즉 θ100의 가중치의 평균이 된다. θ100은 여기서 현재 온도이고, 한 해의 100일째인 날의 v100을 계산하는 것이다. v100은 θ100, θ99, θ98, θ97, ...의 합으로 되어 있다.

이를 그림으로 나타내는 한 가지 방법은, 몇 일간의 온도가 주어졌다고 할 때, t축과 θ축의 그래프를 그리는 것이다. θ100는 어떤 값을 가지고 θ99, θ98...도 특정 값을 가지며 이는 t가 100, 99, 98, ... 일때의 θ를 표시한 것이다. 따라서 몇 일간의 온도를 표시한 것이 된다. 우리가 볼 수 있는 것은 감소하는 지수함수인데, 0.1에서 시작해 0.9*0.1, (0.9)^2*0.1, (0.9)^3*0.1 ... 이런식으로 감소하는 형태의 지수 함수가 되는 것이다. v100을 구하는 과정은 이 두 함수 사이의 요소별 곱셈을 해서 더하는 것이다. 이 값을 취해서 0.1*θ100, 0.1*0.9*θ99, 0.1*(0.9)^2*θ98, ... 이런 식으로 일일 온도에 지수 함수를 곱하고 모두 더한다. 이 앞에 곱해지는 계수들(0.1, 0.1*0.9, 0.1*(0.9)^2 ...)을 모두 더하면 1 또는 1에 가까운 값이 되는데 이는 추후 배우게 될 편향 보정이라고 불리는 값이며, 이들에 의해 지수가중평균이 되는 것이다.

우리는 얼마나 많은 날들이 평균적인 온도가 되는지 궁금해하는데, 0.9^10이 0.35와 대략적으로 같고 이 값은 1/e, 즉 많은 자연로그의 밑과 대략적으로 같다. 더 일반적으로 (1-ϵ)이 있다면(위 예시에서 ϵ는 0.1, (1-ϵ)은 0.9), (1-ϵ)^(1/ϵ)은 대략적으로 1/e와 같다(0.3 또는 0.35). 다시 말 온도가 감소하기까지, 즉 1/3이 될 때까지 약 10일이 걸린다는 것이다. 이러한 이유 때문에, β가 0.9와 같을 때 만약 지난 10일간의 온도에만 초점을 맞춰 가중평균을 계산한다면 10일 뒤에는 가중치가 현재 날짜의 가중치의 1/3으로 줄어드는 것이다. 그와 반대로 β가 0.98이라면, 이 값이 1/e와 매우

유사해지는 것은 0.98^50일 때이다. 여기 처음 50일 동안의 1/e보다 가중치는 더 커지며 감소는 가파르게 일어날 것이다. 따라서 직관적으로 50일의 온도의 평균은 더 급격히 빠르게 떨어지는데, 이 경우 ϵ은 0.02가 되기 때문이다($1/\epsilon = 50$). 이것은 1/(1-β)와도 대략적으로 같다. 여기서 ϵ=1 - β이다. 평균적인 온도가 몇 일 정도가 될지에 관한 상수를 알려주는 것이다. 그러나 이것은 관습적으로 쓰이는 것이지, 수학 공식은 아니다.



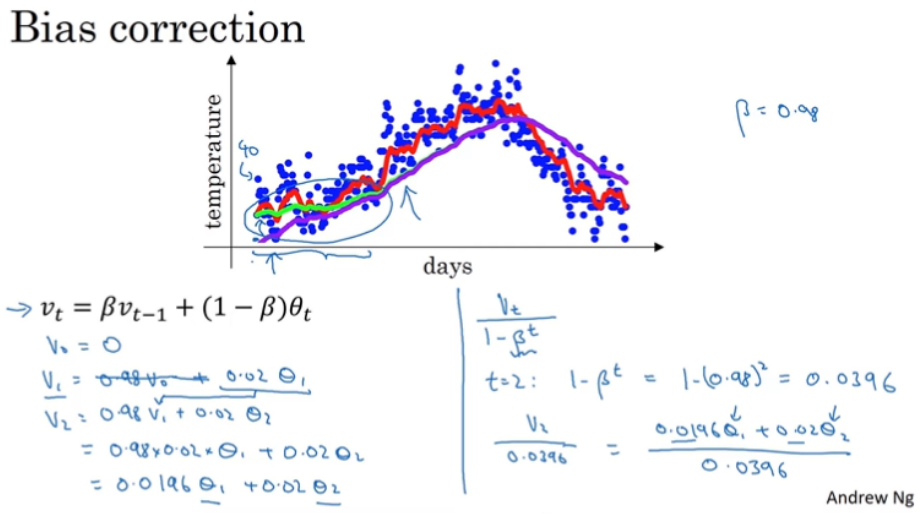
이제 실제로 어떻게 구현할지를 알아볼 것인데, v0을 0으로 초기화하고 첫째 날에는 v1, 둘째 날에는 v2 ...를 계속해서 계산한다. v0, v1, v2 ... 를 별개의 변수로 취급하며, 실제로 구현할 때는 v를 0으로 초기화하며 첫째 날에는 v를 v = β*v+(1-β)θ1으로 계산하고, 다음 날에는 v를 v = β*v+(1-β)θ2로 설정한다. 가끔 v에 아래 첨자 θ를 한 표기법을 사용하기도 하는데, 이는 v가 θ를 매개변수로 하는 지수가중평균을 계산한다는 것을 나타내기 위해서이다. 이것은 반복문으로 나타내면 vθ=0, 각각의 날짜마다 다음 θt를 얻고 vθ= β*vθ +(1-β)θt로 업데이트된다. 이렇게 지수평균을 얻는 식의 장점은 아주 적은 메모리를 사용한다는 것인데, vθ 실수 하나만을 컴퓨터 메모리에 저장하고 가장 최근에 얻은 값을 이 식에 기초해 덮어쓰기하면 되기 때문에 한 줄의 코드만 작성하면 되어서 효율적이다. 지수가중평균을 계산하기 위해 하나의 실수를 저장하는 메모리만 필요하다는 것이다. 이것이 평균을 계산하는 가장 정확하고 최선의 평균은 아니다. 명시적으로 지난 10일 또는 50일간의 온도를 더하고 10이나 50으로 나누는 것이 더 나은 추정치를 제공한다. 그러나 그렇게 하는 방법은 더 많은 메모리를 필요로 하며, 더 복잡한 구현이기 때문에 컴퓨터적으로 많은 비용이 필요하다. 따라서 다음에 다룰 몇 가지 예제에 나오는 많은 변수의 평균을 계산하는 경우, 위와 같은 반복문으로 계산하는 것이 컴퓨터 계산 비용과 메모리 효율 측면에서 더욱 뛰어나다고 할 수 있으며 이것이 머신러닝에서 해 방법을 많이 사용하는 이유이다.

- β = 0.9 일때, 어떤 시점에서 앞서 나온 지수 가중 이동 평균 식을 하나의 값으로 정리하여 표현하게 되면 아래와 같습니다.
  - $v_100 = 0.1\theta_{100} + 0.1 \times 0.9\theta_{99} + 0.1 \times (0.9)^2\theta_{98} + \cdots$
- 이를 그림으로 표현하면 지수 적으로 감소하는 그래프 입니다. (v_{100} 을 기준으로 보았을 때), 그 이유는 v_{100} 은 각각의 요소에 지수적으로 감소하는 요소($0.1 \times (0.9)^n$)를 곱해서 더한 것이기 때문입니다.
- 얼마의 기간이 이동하면서 평균이 구해졌는가? 는 아래의 식으로 대략적으로 구할 수 있습니다.
  - $\beta = (1 - \varepsilon)$ 라고 정의 하면
  - $(1 - \varepsilon)^n = \frac{1}{e}$ 를 만족하는 n 이 그 기간이 되는데, 보통 $\frac{1}{\varepsilon}$ 으로 구할 수 있습니다.
- 지수 가중 이동 평균의 장점은 구현시 아주 적은 메모리를 사용한다는 것입니다.

📌지수 가중 이동 평균의 편향 보정(C2W2L05)

핵심어: 편향 보정(Bias Correction)

이전에 지수가중평균을 어떻게 구현하는지 학습하였다. 편향 보정이라고 불리는 기술적인 세부 사항으로 평균을 더 정확하게 계산할 수 있는데, 어떻게 작동하는지 살펴보자.



지난 강의에서 β가 0.9일 때의 그래프와 0.98일 때의 그래프를 살펴보았다. 그러나 vt = β*v(t-1)+(1-β)θt 공식을 구현하면 β가 0.98일 때 초록색 곡선을 얻지 못할 것이며 대신 보라색 곡선을 얻게 될 것이다. 보라색 곡선은 매우 낮은 곳에서 시작하고 있다는 문제가 있는데, 이를 고쳐보도록 하자. 이동평균을 구할 때 v0은 0으로 초기화하고, v1 = 0.98*v0 + 0.02*θ1이다. 그러나 v0=0이기 때문에 첫 항은 사라지고 v1은 0.02*θ1이 된다. 따라서 첫 번째 날의 온도가 화씨 40도라면 v1의 값은 0.02*40인 8이 될 것이다. 값이 훨씬 더 낮아져서 첫 번째 날의 온도를 잘 추정할 수 없게 된다. v2 = 0.98*v1 + 0.02*θ2가 될 것이며 v1값을 여기 대입하면 v2의 값은 0.98*0.02*θ1 + 0.02*θ2가 된다. θ1과 θ2가 양수라고 가정하면 v2를 계산한 값은 θ1과 θ2보다 훨씬 더 작아질 것이다. 한 해의 첫 두 날짜를 추정한 값이 좋지 않은 추정이 되는 것이다.

이 추정값이 더 나은 값이 될 수 있도록 수정하는 방법이 있는데, 특히 추정의 초기 단계에서 더 정확하게 보정할 수 있다. vt를 취하는 대신 vt/(1-β^t)를 취하는 것이다 (t는 현재 온도). 예를 들어 t가 2일 때 1-β^t는 1-(0.98)^2와 같다. 이 값을 계산하면 0.0396이며, 따라서 둘째 날 온도를 추정한 값은 v2를 0.0396으로 나눈 값과 같으며 이는 다시 0.0196*θ1 + 0.02*θ2를 0.0396으로 나눈 값과 같다. 따라서 이것은 θ1과 θ2의 가중 평균에 편향을 없앤 것과 같다. t가 더 커질수록 β^t는 0에 더 가까워지며 따라서 t가 충분히 커지면 편향 보정은 그 효과가 거의 없어진다. 이것이 t가 커질 때 보라색 곡선과 초록색 곡선이 거의 겹치는 이유이다. 그러나 초기 단계의 학습에서 편향 보정은 더 나은 온도의 추정값을 얻을 수 있도록 도와주며 보라색 선에서 초록색 선으로 갈 수 있도록 해준다.

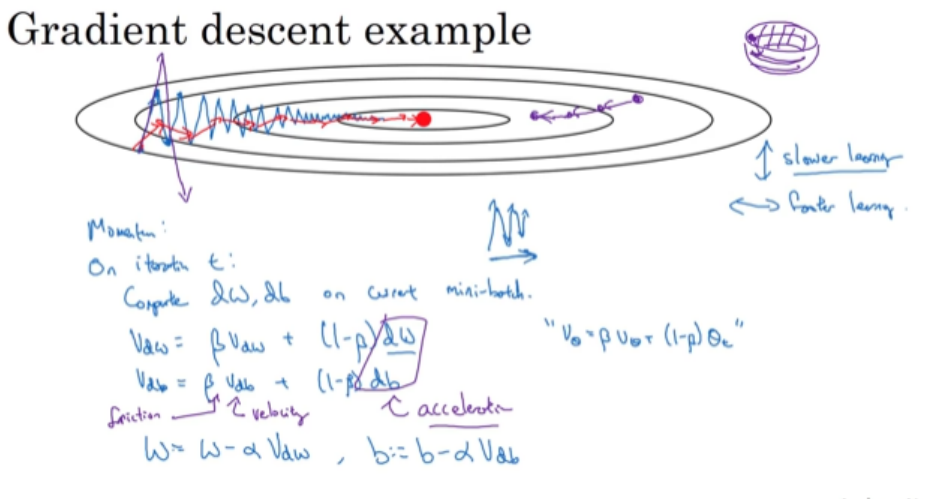
머신러닝에서 지수가중평균을 구현하는 대부분의 경우, 사람들은 편향 보정을 거의 구현하지 않는데 이는 초기 단계를 그냥 기다리고 편향된 추정이 지나간 후부터 시작하기 때문이다. 그러나 초기 단계의 편향이 신경쓰인다면, 편향 보저은 초기에 더 나은 추정값을 얻는데 도움이 될 것이다.

- 편향 보정으로 평균을 더 정확하게 계산할 수 있습니다
- 저번 시간에 따른 지수평균식대로라면 t=1 일때 $(1-\beta)$ 를 곱한 값이 첫번째 값이 되는데, 이는 우리가 원하는 실제 v_1 값과 차이가 나게 된다.
- 따라서 $v_t/(1-\beta^t)$ 를 취해서 초기 값에서 실제값과 비슷해지게 합니다.
- 보통 머신러닝에서 구현하지는 않습니다. 시간이 지남에 $(1-\beta^t)$ 는 1에 가까워 져서 저희가 원하는 값과 일치하게 되기 때문입니다.
- 그렇지만 신경이 쓰인다면 구현하는게 옳습니다.

📌 Momentum 최적화 알고리즘(C2W2L06)

핵심어: Momentum, 최적화(Optimization)

모멘텀 알고리즘 혹은 모멘텀이 있는 경사하강법은 일반적인 경사 하강법보다 거의 항상 더 빠르게 동작한다. 기본적인 아이디어는 경사에 대한 지수가중평균을 계산하는 것이며, 그 값으로 가중치를 업데이트 하는 것이다. 이번 강의에서는 이 한 문장에 대한 설명을 풀어서 어떻게 구현할 수 있을지 알아보고자 한다.

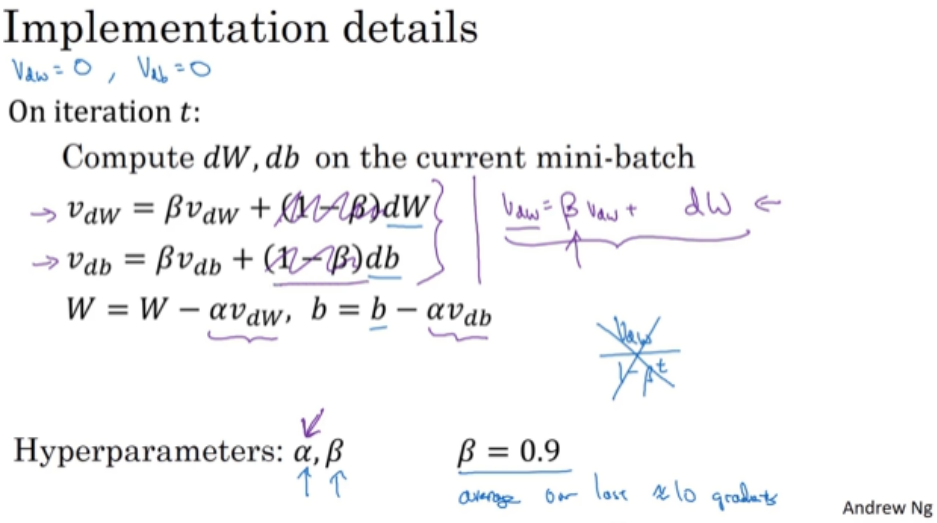


Andrew Ng

대부분의 예제에서 비용함수를 최적화한다고 가정해보면, 등고선은 다음과 같으며 빨간 점은 최솟값의 위치를 나타낸다. 경사 하강법을 가장자리에서 시작하여 경사 하강법 혹은 미니배치 경사 하강법의 한 반복을 취하면 최솟값 쪽으로 향한다.타원의 반대쪽에서 경사 하강법의 한 단계를 취하면 반대로 돌아오고, 계속 한 단계씩 갈 때마다 이런 식으로 나아가며 많은 단계를 취하면 최솟값으로 나아가면서 천천히 진동한다. 위 아래로 일어나는 이러한 진동은 경사 하강법의 속도를 느리게 하고 더 큰 학습률을 사용하는 것을 막는다. 왜냐하면 오버슈팅하게 되어 발산(보라색의 경우)할 수도 있기 때문이다. 따라서 학습률이 너무 크지 않아야 진동이 커지는 것을 막을 수 있다. 이 문제를 보는 또 다른 관점은, 수직축에서는 진동을 막기 위해 학습이 더 느리게 일어나기를 바라지만, 수평축에서는 더 빠른 학습을 원한다. 최솟값을 향해 왼쪽에서 오른쪽으로 이동하는 것을 처리하고 싶기 때문이다. 따라서 모멘텀을 이용한 경사 하강법에서는 구현할 때 다음과 같이 한다.

각각의 반복에서, 더 정확히 말해서, 반복 t에서 보편적인 도함수 dw와 db를 계산하게 되는데, 여기서는 현재의 미니배치에 대한 dw와 db를 계산하게 된다. 배치 경사 하강법을 사용하는 경우 현재의 미니배치는 전체 배치와 같으며 현재의 미니배치가 전체 훈련 세트와 같은 경우에도 잘 작동한다. 그 다음에 하는 것은 v_dw = β*v_dw + (1-β)dw를 계산하는 것이다. 이는 전에 계산했던 v_θ = β*v_θ + (1-β)θ_t와 비슷하다. 이동평균을 w에 대한 도함수로 계산하는 것이다. v_db도 비슷한 방식으로 v_db = β*v_db + (1-β)db를 계산한다. 그럼 이제 w를 사용해 가중치를 업데이트하는데, w = w - α*v_dw, b = b - α*v_db로 업데이트 된다. 이것은 경사 하강법의 단계를 부드럽게 만들어준다.

지난 번에 계산한 몇 가지 도함수가 진동이 심하다고 하면, 이 경사의 평균을 구하면 수직 방향의 진동이 0에 가까운 값으로 평균이 만들어진다. 진행을 늦추고 싶은 수직 방향에서는 양수와 음수를 평균하기 때문에 평균이 0이 된다. 반면 수평 뱡햐에서 모든 도함수는 오른쪽을 가리키고 있기 때문에 수평 방향의 평균은 꽤 큰 값을 가진다. 따라서 몇 번의 반복이 있는 이 알고리즘에서 경사 하강법은 결국 수직 방향에서는 훨씬 더 작은 진동이 있고, 수평 방향에서는 더 빠르게 움직인다는 것을 찾을 수 있으며 이 알고리즘은 더 직선의 길을 가거나 진동을 줄일 수 있게 한다는 점을 알 수 있다. 이 모멘텀에서 얻을 수 있는 직관은, 밥그릇 모양의 함수를 최소화하려고 하면 dw, db 도함수의 항들은 아래로 내려갈 때 가속을 제공한다고 볼 수 있으며, v_dw, v_db와 같은 모멘텀 항들은 속도를 나타낸다고 볼 수 있다. 따라서 작은 공이 이 그릇의 경사를 내려갈 때 도함수는 여기에 가속을 부여하고, 더 빠르게 내려가도록 만든다. β의 값은 1보다 조금 작기 때문에 마찰을 제공해서 공이 제한 없이 빨라지는 것을 막는다. 경사 하강법이 모든 이전 단계를 독립적으로 취하는 대신, 그릇을 내려가는 공에 가속을 주고 모멘텀을 제공할 수 있다.



이제는 어떻게 구현할 것인지에 대한 세부사항을 살펴볼 것이다. 여기 학습률 α와 지수가중평균을 제어하는 β라는 두 가지 하이퍼파라미터가 있다. β의 가장 일반적인 값은 0.9이며 이는 지난 10일 간의 온도를 평균한 것이다. 실제로 β가 0.9인 경우 매우 잘 작동한다. 편향 보정의 경우, v_dw를 (1 - β^t)로 나눠주면 되는데 많은 사람들은 이를 잘 사용하지 않는다. 그 이유는 10번의 반복 뒤에 이동평균이 충분히 진행되어서 편향 추정이 더 이상 일어나지 않기 때문이다. 이전에도 말했듯 경사 하강법이나 모멘텀을 구현할 때 편향 보정을 하는 사람은 거의 없다. v_dw를 0으로 초기화하는 과정은 dw와 같은 차원의 0으로 이루어진 행렬이며 w와도 같은 차원이다. v_db 또한 0에 대한 벡터로 초기화된다(db, b와 같은 차원).

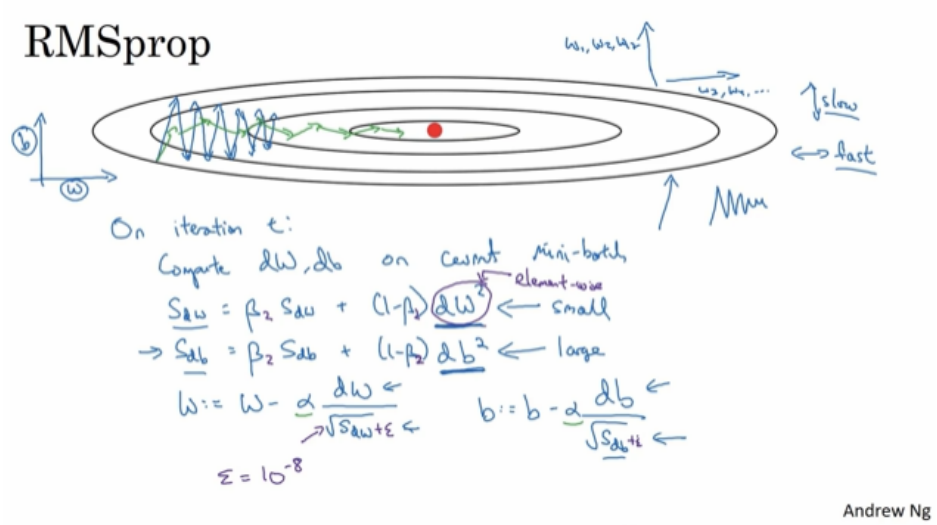
모멘텀이 있는 경사하강법에 대한 논문을 읽어보면, (1 - β)에 관한 항이 자주 삭제되어 있는 것을 발견할 수 있다. 따라서 v_dw = β*v_dw + dw으로도 쓸 수 있는 것이다. 보라색으로 표시한 버전의 효과는 v_dw가 1/(1-β)에 대한 계수로 스케일링 되는 것이다. 경사하강법의 업데이트를 실행할 때는 α가 1/(1- β)에 대응되는 값으로 바뀔 필요가 있다. 실제로는 두 가지 모두 잘 작동할 것이며 학습률 α에 대한 가장 최적의 값에만 영향을 미치게 된다. 그러나 이 특정한 식 v_dw = β*v_dw + dw이 덜 직관적으로 느껴질 수 있는데, 이것의 한 가지 효과는 하이퍼파라미터 β의 값을 보정하는 것인데, 이로 인해 v_dw와 v_db의 스케일링에 영향을 주게 되고 학습률도 다시 보정해야 한다. 따라서 왼쪽 수식((1-β)가 살아있는 항)이 더 많이 선호된다. β를 0.9로 하는 것은 두 설정 모두 하이퍼파리미터의 일반적인 선택이며 학습률 α가 다르게 보정된다는 것이 이 두 버전의 차이점이다. 모멘텀이 있는 경사 하강법은 모멘텀이 없는 경사 하강법보다 거의 항상 더 잘 작동한다.

- 알고리즘은 아래와 같습니다.
  - $V_{dW} = \beta_1 V_{dW} + (1-\beta_1)dW$
  - $w := w - \alpha V_{dW}$
- Momentum 의 장점은 매 단계의 경사 하강 정도를 부드럽게 만들어줍니다
- Momentum 알고리즘에서는 보통 평향 추정을 실행하지 않습니다. 이유는 step 이 10 단계정도 넘어가면 이동평균은 준비가 돼서 편향 추정이 더 이상 일어나지 않기 때문입니다

📌RMSProp 최적화 알고리즘(C2W2L07)

핵심어: RMSProp, 최적화(Optimization)

모멘텀을 사용하는 것이 경사 하강법을 빠르게 할 수 있다는 것을 학습하였다. 또 다른 알고리즘으로 root mean square prop, 줄여서 RMSprop이 있다. 이 알고리즘 역시 경사 하강법을 빠르게 하는데, 작동 방식을 살펴보자.



Andrew Ng

지난 번에 했던 예제를 살펴보면 경사 하강법에서 수평 방향으로 진행을 시도해도 수직 방향으로 큰 진동이 있다는 것을 알 수 있다. 이 예제에 대한 직관을 위해 수직축은 매개변수 b, 수평축은 매개변수 w라고 가정하자. b 방향 또는 수직 방향의 학습 속도를 낮추고, 수평 방향의 속드롤 빠르게 하기 위한 것이다. RMSprop 알고리즘이 하는 일은 다음과 같다. 반복 t에서 현재의 미니배치에 대한 보통의 도함수 dw와 db를 계산할 것이다. 지수 가중 평균을 유지하기 위해 새로운 표기법인 s_dw를 사용하며, 이 값은 β*s_dw + (1-β)*dw^2이다. 여기서 제곱 표시는 명확히 말하자면 요소별 제곱을 나타낸다. 이것은 도함수의 제곱을 지수가중평균하는 것이며, 비슷하게 s_db도 β*s_db + (1-β)*db^2와 같다. 다음으로, RMSprop은 매개변수를 w = w - α*(dw/sqrt(s_dw)), b = b - α*(db/sqrt(s_db)) 와 같이 업데이트 한다. 수평 방향(이 예제에서의 w 방향)에서는 학습률이 꽤 빠르게 가기를 원하는 반면 수직 방향(이 예제에서의 b 방향)에서는 느리게, 혹은 수직 방향의 진동을 줄이고 싶어한다. 따라서 이 두 항, s_dw와 s_db에서 우리가 원하는 것은 w에서 s_dw가 상대적으로 작고(상대적으로 작은 숫자로 나눠주고), b에서 s_db가 상대적으로 큰(상대적으로 큰 숫자로 나눠주는 것) 것이다(수직 방향에서의 업데이트를 줄이기 위해). 실제로 수직 방향에서의 도함수가 수평 방향의 것보다 훨씬 크다. 즉 b 방향에서의 경사가 매우 크다. 도함수 db는 매우 크고, dw는 상대적으로 작다. 왜냐하면 수직 방향 b에서 기울기가 수평 방향 w에서의 기울기보다 가파르기 때문이다. 따라서 db^2는 상대적으로 크고, dw가 작기 때문에 dw^2는 상대적으로 더 작다.

다음에 오는 효과는 더 큰 숫자로 나눠서 수직 방향에서 업데이트 하기 때문에 진동을 줄이는 데 도움을 준다. 반면 수평 방향에서는 작은 숫자로 나눠서 업데이트 하기

때문에 RMSprop를 사용한 업데이트는 초록색 화살표 같다. 수직 방향에서의 업데이트는 감소하지만 수평 방향은 계속 나아가는 것이다. 이것의 효과는 큰 학습률을 사용해 빠르게 학습하고 수직 방향으로 발산하지 않는다. 더 명확히 말하자면 수직과 수평 방향을 b와 w로 나타냈는데, 실제로는 매우 고차원의 공간에 있기 때문에 진동을 줄이려는 수직 차원은 w1, w2, ... , w17의 매개변수 집합이고 수평 방향의 차원은 w3, w4, ... 처럼 나타날 것이다. 따라서 w와 b의 분리는 표현을 위한 것이고, 실제로 dw와 db는 매우 고차원의 매개변수 벡터이다. 그러나 직관적으로 이런 진동을 얻는 차원에서 더 큰 합 또는 가중평균 도함수의 제곱을 계산하기 때문에, 결국에는 이러한 진동이 있는 방향을 감쇠시키게 된다. 이것이 제곱 평균 제곱근, 줄여서 RMSprop에 대한 설명이며, 도함수를 제곱해서 결국 제곱근을 얻기 때문이다.

마지막으로 넘어가기 전에 몇 가지 세부사항을 말하자면, 다음에는 RMSprop을 모멘텀과 결합할 것이다. 따라서 모멘텀을 위해 사용했던 하이퍼파라미터 β를 쓰는 대신, 충돌을 피하기 위해 여기 있는 하이퍼파라미터를 β2라고 나타내고자 한다. 또한 알고리즘이 0으로 나눠지지 않도록 주의해야 하며, s_dw의 제곱근이 0에 매우 가깝다면 이 값은 폭발할 위험이 있다. 실제로 구현할 때는 이러한 수학적 안정성이 보장되어야 한다. 이럴 때는 s_dw, s_db에 매우 작은 수 ϵ더해주는데, ϵ가 10^(-8)이면 합리적인 기본값이지만 이보다 약간 더 큰 값도 수학적 안정성을 보장할 수 있다. 너무 작은 숫자로 나누지 않도록 주의하는 것이 중요하다. RMSprop는 진동을 줄이는 효과가 있다는 점에서 모멘텀과 비슷하며, 더 큰 학습률을 사용할 수 있게 해서 속도를 올려준다. RMSprop와 모멘텀을 함께 사용하면 더 나은 최적화 알고리즘을 얻을 수 있다.

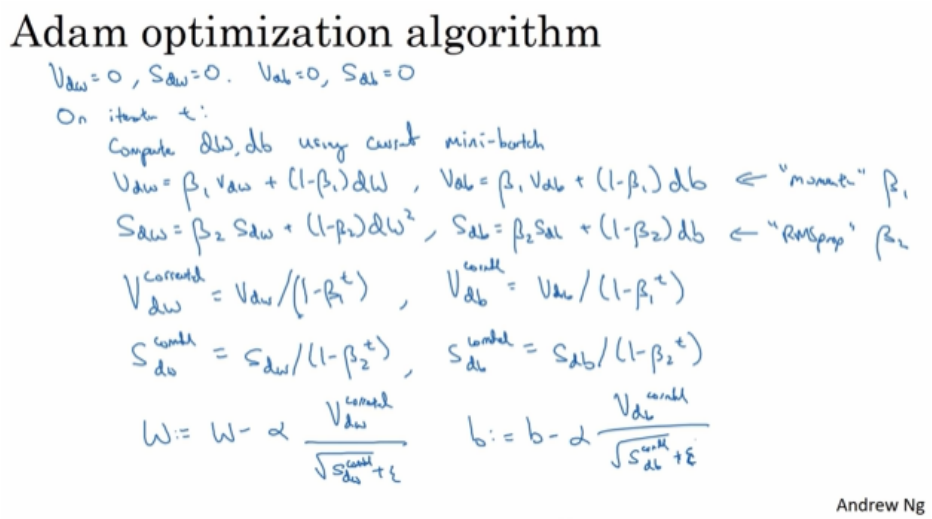
- 알고리즘은 아래와 같습니다.
  - $S_{dW} = \beta_2 S_{dW} + (1-\beta_2) dW^2$
  - 업데이트: $w := w - \alpha \frac{dW}{\sqrt{S_{dW} + \epsilon}}$
  - dW^2 은 요소별 제곱을 뜻합니다.
- RMSProp 의 장점은 미분값이 큰 곳에서는 업데이트 시 큰 값으로 나눠주기 때문에 기존 학습률 보다 작은 값으로 업데이트 됩니다. 따라서 진동을 줄이는데 도움이 됩니다. 반면 미분값이 작은 곳에서는 업데이트시 작은 값으로 나눠주기 때문에 기존 학습률 보다 큰 값으로 업데이트 됩니다. 이는 더 빠르게 수렴하는 효과를 불러옵니다.

📌 Adam 최적화 알고리즘(C2W2L08)

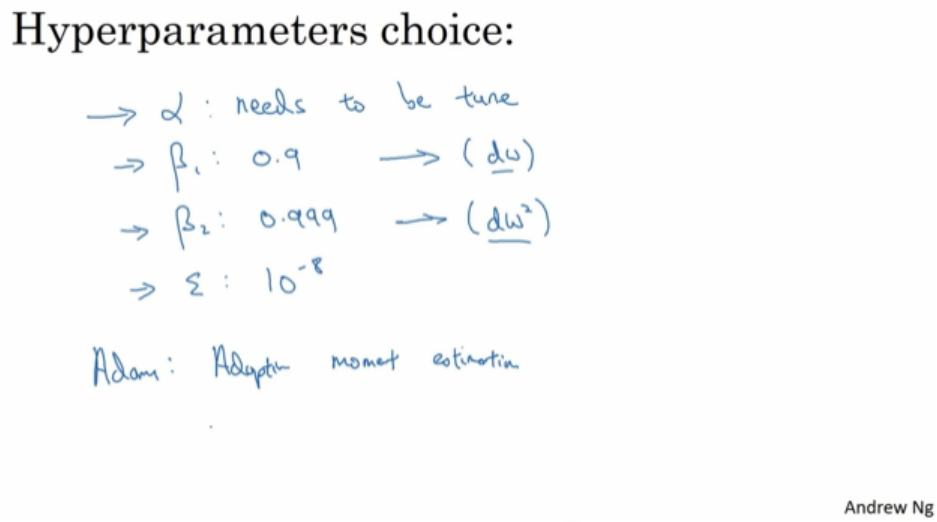
핵심어: Adam, 최적화(Optimization)

딥러닝의 역사에서 잘 알려진 사람들을 포함한 딥러닝 연구원들은, 가끔 최적화 알고리즘을 제안하고 몇 가지 문제에 잘 작동하는지를 증명한다. 그러나 그런 최적화 알고리즘이 훈련을 원하는 넓은 범위의 신경망에 일반적으로 잘 작동하지 않는다는 것을 보여주었다. 이에 대해 딥러닝 커뮤니티는 새로운 최적화 알고리즘에 약간의 의심을 갖게 되었다. 또한 모멘텀이 있는 경사 하강법이 아주 잘 작동하기 때문에 더 잘 작동하는 알고리즘을 제안하기 어려운 것도 있다. 이버네 다룰 RMSprop과 Adam

최적화 알고리즘은 넓은 범위의 딥러닝 아키텍처에서 잘 작동하는 알고리즘으로 우뚝 섰다. 이들은 시도를 망설이지 않아도 되는 알고리즘인데, 많은 사람들이 시도했고 많은 문제에 잘 작동한다는 것을 보았기 때문이다. Adam 최적화 알고리즘은 RMSprop과 모멘텀을 합친 알고리즘이다. 어떻게 작동하는지 살펴보도록 하자.



Adam을 구현하기 위해서 v_dw와 s_dw를 0으로 초기화한다. 그리고 반복 t에서 도함수 dw와 db를 현재의 미니배치를 써서 계산한다(주로 미니배치 경사 하강법을 사용). 그리고 모멘텀 지수가중평균을 계산해준다. 여기서 RMSprop의 하이퍼파라미터와 구분하기 위해 β1이라고 부른다고 했을 때, v_dw = β1*v_dw + (1-β1)*dw가 되며 이는 모멘텀을 구현할 때 사용하는 식이다. β 대신 하이퍼파라미터로 β1을 사용한 점만 다른 것이다. 비슷하게 v_db = β1*v_db + (1-β1)*db이다. RMSprop의 업데이트에서는 다른 하이퍼파라미터 β2를 사용하여, s_dw = β2*s_dw + (1-β2)*dw^2이며 여기서 제곱은 dw의 요소별 제곱을 의미한다. 마찬가지로 s_db = β2*s_db + (1-β2)*db^2라고 할 수 있다. v_dw, v_db는 하이퍼파라미터 ㅊ을 사용한 모멘텀 업데이터고, s_dw, s_db는 하이퍼파라미터 β2를 사용한 RMSprop 업데이트이다. 전형적인 Adam의 구현에서는 편향 보정을 하는데, v_dw^corrected는 편향 보정을 의미하며 이 값은 v_dw/(1- β1^t)와 같다. 비슷하게 v_db^corrected는 v_db/(1- β1^t)와 같고, s_dw에 대한 편향 보정 s_dw^corrected 역시 s_dw/(1- β2^t)와 같으며, s_db에 대한 편향 보정 s_db^corrected 또한 s_db/(1- β2^t)와 같다. 최종적으로 업데이트를 실행하면 w = w - α*(v_dw^corrected / sqrt(s_dw^corrected + ϵ)) 이다(모멘텀을 구현하고 있고 거기다 RMSprop을 부분적으로 추가하기 때문). b도 비슷한 방식으로 w = w - α*(v_db^corrected / sqrt(s_db^corrected + ϵ)) 이라 쓸 수 있다. 따라서 이 알고리즘은 모멘텀이 있는 경사 하강법의 효과와 RMSprop이 있는 경사 하강법의 효과를 합친 결과가 나온다. 이것은 매우 넓은 범위의 아키텍처를 가진 서로 다른 신경망에 잘 작동한다는 것이 증명된, 일반적으로 많이 쓰이는 학습 알고리즘이다.



이 Adam 알고리즘은 많은 하이퍼파라미터들을 가지고 있는데, 학습률 하이퍼파라미터 α는 매우 중요하고 보정될 필요가 있으므로 다양한 값을 시도해서 잘 맞는 것을 찾아야 한다. β1은 기본값으로 0.9를 보통 선택하며 이는 dw의 이동 평균, 가중 평균으로 모멘텀에 관한 항이다. β2에 대한 하이퍼파라미터는 Adam 논문에서 저자가 추천하는 값이 0.999이며 이것은 dw^2와 db^2의 이동가중평균을 계산한 것이다. 그리고 ϵ의 값은 크게 상관 없지만 Adam 논문의 저자에 따르면 10^(-8)을 추천한다고 한다. 이 값을 설정하지 않아도 전체 성능에 영향은 없지만, Adam을 구현할 때 보통 사람들은 β1과 β2, 그리고 ϵ도 기본값을 사용한다(ϵ의 값을 보정하는 사람은 드물다). 보통 α에 여러 값을 시도해 가장 잘 작동되는 값을 찾으며, β1과 β2도 보정할 수 있지만 자주 하지는 않는다.

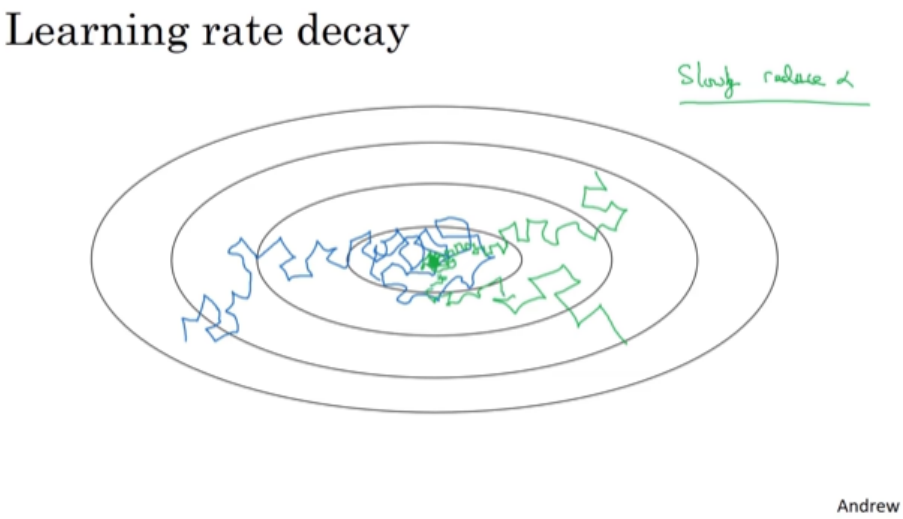
Adam이라는 용어는 Adaptive moment estimation에서 온 것이다. β1이 도함수의 평균을 계산하므로 이것이 첫 번째 모멘트이고, β2가 지수가중평균의 제곱을 계산하므로 두 번째 모멘트이다. 대부분의 사람들은 이를 그냥 Adam 최적화 알고리즘이라고 부른다. 이 알고리즘을 이용하여 신경망을 더 빠르게 훈련시킬 수 있을 것이다.

- Adam 은 Momentum 과 RMSProp 을 섞은 알고리즘입니다.
- 알고리즘은 아래와 같습니다.
  - $V_{dW} = 0, S_{dW} = 0$ 로 초기화 시킵니다.
  - Momentum 항: $V_{dW} = \beta_1 V_{dW} + (1 - \beta_1) dW$
  - RMSProp 항: $S_{dW} = \beta_2 S_{dW} + (1 - \beta_2) dW^2$
  - Bias correction: $V_{dW}^{correct} = \frac{V_{dW}}{1 - \beta_1^t}, S_{dW}^{correct} = \frac{S_{dW}}{1 - \beta_2^t}$
  - 업데이트: $w := w - \alpha \frac{V_{dW}^{correct}}{\sqrt{S_{dW}^{correct}} + \epsilon}$
- Adam 은 Adaptive moment estimation 의 약자입니다.

📌 학습률 감쇠(C2W2L09)

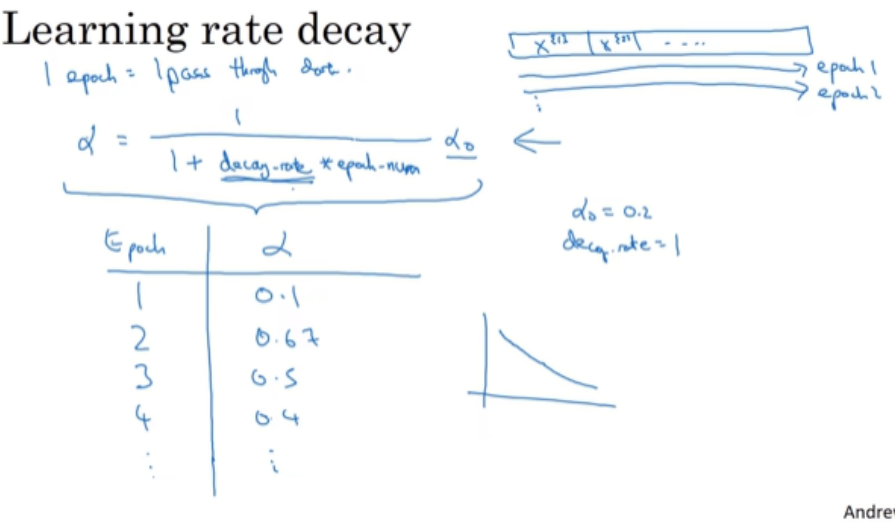
핵심어: 학습률 감쇠(Learning Rate Decay)

학습 알고리즘의 속도를 높이는 한 가지 방법은, 시간에 따라 학습률을 천천히 줄이는 것이다. 이것을 학습률 감쇠라고 하는데, 어떻게 구현할 수 있을지 살펴보도록 하자.



Andrew Ng

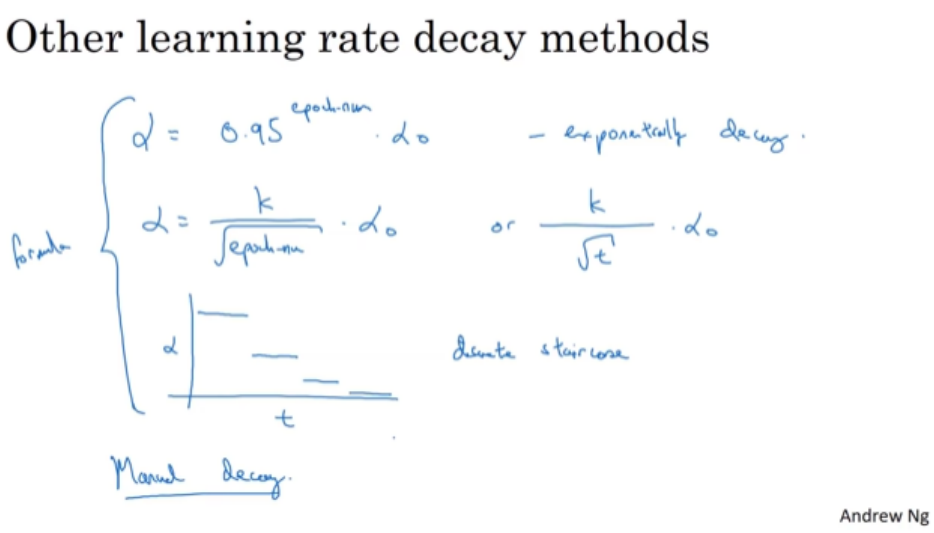
왜 학습률 감쇠가 필요한지 예시를 들고자 한다. 상당히 작은 미니배치에 대해 미니배치 경사 하강법을 구현한다고 가정하자(미니배치가 64나 128같은 경우). 단계를 거치면서 약간의 노이즈가 있지만 최솟값으로 향하는 경향을 보일 것이다. 그러나 정확하게 수렴하지는 않고, 주변을 돌아다니게 될 것이다. 이는 어떤 고정된 값인 α를 사용했고, 서로 다른 미니배치에 노이즈가 있기 때문이다. 그러나 천천히 학습률 α를 줄이는 경우, α가 여전히 큰 초기 단계에서는 여전히 상대적으로 빠른 학습이 가능하지만 α가 작아지면 단계마다 진행 정도가 작아지고, 최솟값 주변의 밀집된 영역에서 진동하게 될 것이다(훈련이 계속되더라도 최솟값 주변을 배회하는 대신). 즉 α를 천천히 줄이는 것의 의미는 학습의 초기 단계에서는 훨씬 더 큰 스탭으로 진행하고, 학습이 수렴할수록 학습률이 느려져 작은 스탭으로 진행하는 것이다.



Andrew Ng

이제 학습률 감쇠를 구현하는 방법에 대해 알아보자. 하나의 에포크는 데이터를 지나는 하나의 패스이다. 따라서 훈련 세트가 다음(슬라이드 오른쪽 위)과 같으면 서로 다른 미니배치로 나눠서 훈련 세트를 지나는 첫 번째 패스를 첫 번째 에포크, 두 번째 지나는 것을 두 번째 에포크라고 부른다. 우리가 할 수 있는 것은 학습률 α를 (1/(1+decay_rate*epoch_num))* α0(초기학습률)으로 설정하는 것이다. 여기 있는 감쇠율(decay_rate)은 조정이 필요한 또 다른 하이퍼파라미터인데, 구체적인 예시를 들어보자.

여러 번의 에포크를 거치면, α0가 0.2이고 감쇠율이 1일 때 첫 번째 에포크를 하는 동안 α는 (1/(1+1))*α0이며 이 값, 즉 학습률의 값은 α=0.1이 된다. 두 번째 에포크에서 학습률은 0.67로 감소하고, 세 번째 에포크에서는 0.5로, 네 번째는 0.4로 감소한다. 다음 에포크도 계산해보면 감이 잡힐 것이다. 에포크 수에 대한 함수에서 학습률은 점차적으로 감소하게 되는 것이다. 학습률 감쇠를 사용하고 싶다면, 하이퍼파라미터 α0과 감쇠율 decay_rate에 대해 다양한 값들을 대입하고, 잘 작동하는 값을 찾으면 된다. 학습률 감쇠에 대한 이 식 말고도 사람들이 사용하는 또 다른 방법들이 있다.



예를 들면 지수적 감쇠라고 불리는 것은, α가 1보다 작은 값을 가진다. 예를 들면 α = (0.95^epoch_num)*α0에서 α는 기하급수적으로 빠르게 학습률을 감소시킨다. 사람들이 사용하는 또 다른 식은 α = (k/sqrt(epoch_num))*α0 이다(또는 미니배치의 개수 t의 제곱근으로 나눠주는 다음과 같은 (k/sqrt(t))*α0). 또 어떤 사람들은 이산적 단계로 감소하는 학습률을 사용하기도 한다. 어떤 단계에서는 어떤 학습률 값을 가지고, 그 뒤에는 학습률이 반으로 줄어들고 일정 시간이 지날 때마다 계속 반씩 줄어드는 모습이다. 이런 것을 이산 계단이라고 부른다. 지금까지 시간에 따라 학습률이 어떻게 바뀌는지를 통제하는 여러 가지 식들을 살펴보았는데, 사람들이 사용하는 또 다른 방법은 직접 조작하는 감쇠이다.

한 번에 하나의 모델을 훈련하는 데 몇 시간 혹은 며칠이 걸린다면, 어떤 사람들은 훈련을 거치면서 모델을 정리해 나갈 것이다. 학습률이 느려지고 있는 것처럼 느껴서 데이터의 크기를 줄이는 것이다. 이런 식으로 α의 값을 시간이 날마다 직접 보정하는

것은 훈련이 작은 수의 모델로만 이루어진 경우에만 가능하지만, 사람들은 가끔 이 방법을 쓰기도 한다.

이제 우리에게는 학습률 α를 조작하는 몇 가지 선택지가 있다. 만약 하이퍼파라미터의 개수가 너무 많아서 어떤 선택을 해야 할지 잘 모르겠다면 이후에 배우게 될 시스템적으로 하이퍼파라미터를 선택하는 방법이 도움이 될 것이다. 학습률 감쇠는 α의 값만을 바꿔서 큰 영향을 얻을 수 있다는 점에서 학습률 감쇠는 도움이 되고 훈련 속도를 빠르게 해줄 수 있다.

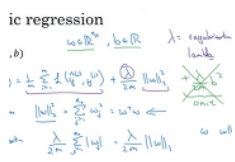
- 작은 미니배치 일수록 잡음이 심해서 일정한 학습률이라면 최적값에 수렴하기 어려운 현상을 볼 수 있습니다.
- 학습률 감쇠 기법을 사용하는 이유는 점점 학습률을 작게 줘서 최적값을 더 빨리 찾도록 만드는 것입니다.
- 다양한 학습률 감쇠 기법들이 있습니다.
  - 1 epoch = 전체 데이터를 1번 훑고 지나가는 횟수 입니다.
  - $\alpha = \frac{1}{1 + decay\ rate\ \times epoch\ num}\alpha_0$
  - $\alpha = 0.95^{epoch\ num}\alpha_0$ (exponential decay 라고 부릅니다.)
  - $\alpha = \frac{k}{\sqrt{epoch\ num}}\alpha_0$
  - $\alpha = \frac{k}{\sqrt{batch\ num}}\alpha_0$
  - step 별로 α 다르게 설정

'DL 스터디&프로젝트' 카테고리의 다른 글

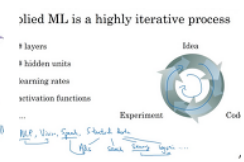
[Euron 중급 세션 9주차] 딥러닝 2단계 3. 최적화 문제 설정 (2) 2023.11.06
[Euron 중급 세션 8주차] 딥러닝 2단계 2. 신경망 네트워크의 정규화 (1) 2023.10.30
[Euron 중급 세션 5주차] 딥러닝 2단계 1. 머신러닝 어플리케이션 설정하기 (0) 2023.10.09
[Euron 중급 세션 4주차] 5. 심층 신경망 네트워크 (1) 2023.10.02
[Euron 중급 세션 3주차] 4. 얕은 신경망 네트워크 (0) 2023.09.25

관련글

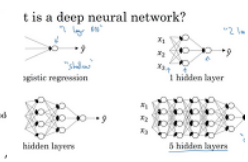
[Euron 중급 세션...



[Euron 중급 세션...



[Euron 중급 세션...



[Euron 중급 세션...