DL 스터디&프로젝트

[Euron 중급 세션 15주차] 딥러닝 2단계 7. 다중 클래스 분류 ~ 8. 프로그래밍 프레임워크 소개

by 공부하자_ 2023. 12. 18.

딥러닝 2단계: 심층 신경망 성능 향상시키기

7. 다중 클래스 분류

📌 Softmax Regression(C2W3L08)

핵심어: Softmax

지금까지 우리가 봤던 분류 문제는 이진 분류였는데, 여기서는 0과 1 두 가지 선택이 있었다. 이러한 로지스틱 회귀를 일반화한 소프트맥스 회귀가 있다. 이는 클래스가 두 개인 경우 외에도 여러 클래스나 C 중 하나를 인식할 때 예측에 사용할 수 있다.

분류 전체보기

DL 스터디&프로젝트

Data Science 프로젝트

Github 스터디

Data Science 개인 공부

Backend 프로젝트

기타 공부

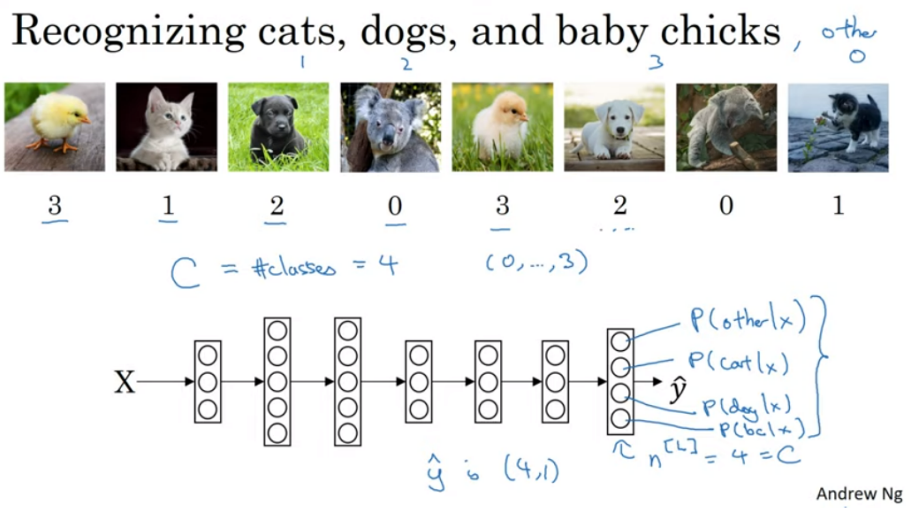
공지사항

최근글 인기글

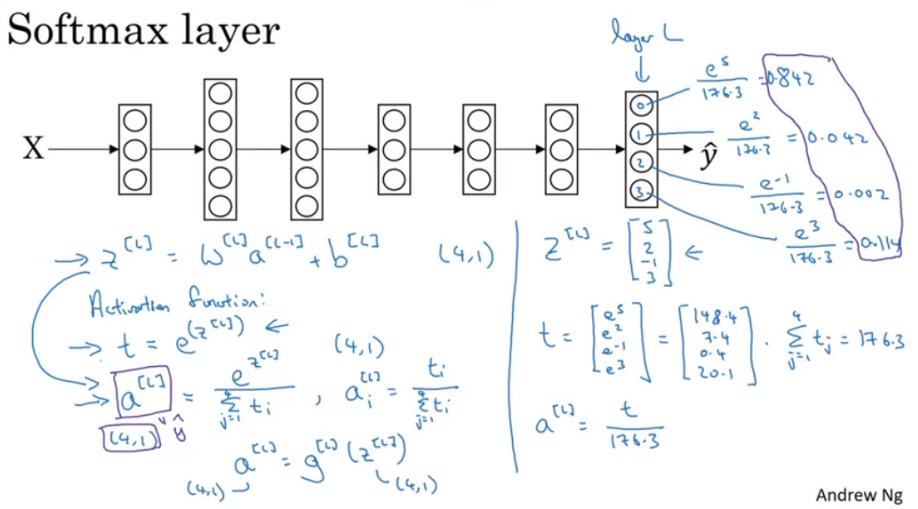
[Euron 중급 세션 15주차]… 2023.12.18

[Euron 중급 세션 12주차]… 2023.11.27

[Euron 중급 세션 11주차] 딥러닝 2단계 …



고양이를 인식하는 것에서 나아가 개, 고양이, 병아리를 인식해보자. 고양이는 클래스 1, 개는 클래스 2, 병아리는 클래스 3이다. 어느 것에도 속하지 않으면 클래스 0이다. 여기 이미지와 속해 있는 클래스를 적어두었다. 여기서 대문자 C는 클래스의 숫자를 나타내는 데 사용할 것이다. 입력값을 분류하는 것이다. 만약 클래스에 숫자를 붙인다면 0에서 C-1까지가 부여되는 것이며, 여기서는 0, 1, 2, 3이 된다. 이제 아래와 같은 신경망을 하나 만들었다고 하자. 출력층에는 C개 , 이 경우 4개의 출력 단위가 있는 신경망이다. 출력층 L의 단위 개수인 n은 4, 또는 일반적으로 C가 될 것이다. 우리는 각 단위와 상위층에서 각 클래스의 확률을 알려주었으면 한다. 첫 번째 단위에서는 우리가 원하는 출력값이 입력값 X가 주어졌을 때 기타 클래스가 나올 확률이 될 것이다. 여기는 X가 주어졌을 때 고양이의 확률(P(cat|X)), 그 아래는 X가 주어졌을 때 개일 확률(P(dog|X))이 되는 것이다. 따라서 출력값인 yhat (4,1)차원의 벡터가 되는데, 왜냐하면 출력값으로 네 개의 확률값이 주어지기 때문이다. 그리고 각 yhat의 값들의 합은 1이 되어야 한다.



이런 신경망을 얻기 위한 가장 표준적인 모델은, 출력층에 이러한 출력값을 만들 수 있도록 소프트맥스층을 사용하는 것이다. 큰 그림을 그린 뒤에 다시 돌아가서 소프트맥스가 무엇을 하는 것인지 살펴보도록 하자. 신경망의 최종층에서, 평소처럼 층의 선형적인 부분인 z[L]을 계산할 것이다. 이는 최종층인 L의 z값이다. 평소에는 z[L]을

2023.11.20

[Euron 중급 세션 10주차]…
2023.11.20

[Euron 중급 세션 10주차]…
2023.11.13

최근댓글

오늘 하루 고생 많으셨습니다.

좋은 글 잘 보고 가요! 감사…

좋은 글 잘 보고 가요! 감사…

잘보고 갑니다!

태그

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

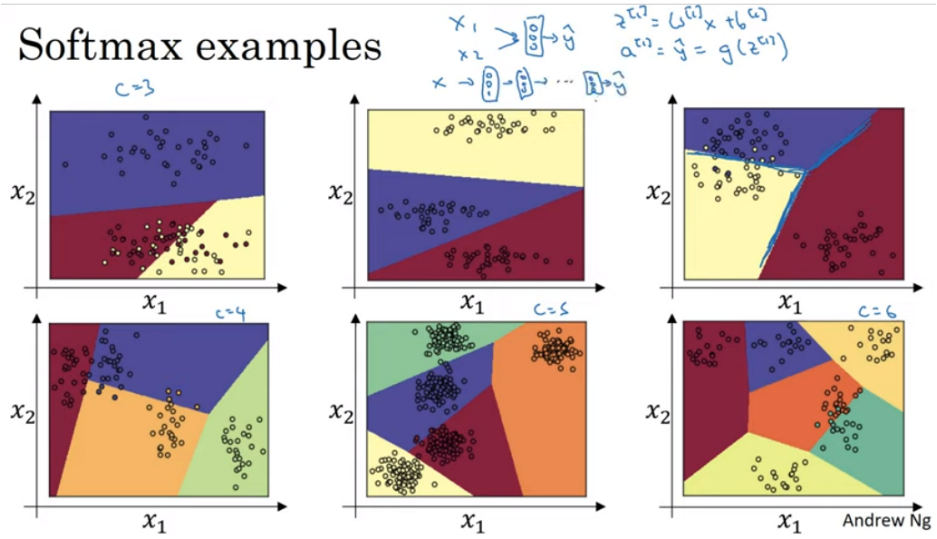
전체 방문자

615

Today : 1
Yesterday : 1

w[L]과 이전 층의 활성화 함수를 곱한 뒤 그 최종층의 편향을 더하여 계산하였다. 이제 z값을 계산하기 위해 소프트맥스 활성화 함수라는 것을 사용해야 한다. 소프트맥스 층의 활성 함수는 조금 다른데, 우선 t=e^(z[L])이라는 임시 변수를 사용한다. 이를 각 원소에 대해 계산하는데, z[L]이 (4, 1) 차원의 벡터이며 t는 z[L[의 각 원소에 e를 취한 것이므로 결과도 (4,1)이 될 것이다. 출력갑 a[L]은 벡터 t와 같게 되는 것이다. 다만 합이 1 되도록 정규화해야 한다. 즉 a[L]은 z[L]을 j가 1부터 4까지(원소가 4개이기 때문) t_j를 모두 더한 값으로 나누는 것이다. 즉 a[L]은 (4, 1) 벡터이고, 이 벡터의 i번째 원소인 a[L]_i는 t_i를 t_ 값들의 합으로 나눈 것과 같게 되는 것이다. 아직 감이 잘 안 잡힌다면, 예시를 하나 들어보자.

예를 들어 (4, 1)벡터인 z[L]이 [5, -2, 1, 3]이라고 하자. 우선 원소별로 e를 취해 t를 구하자. 그럼 t=[e^5, e^2, e^-1, e^3]이 될 것이다. 이를 계산기에서 계산하면 [e^5 = 148.4, e^2 = 7.4, e^-1=0.4, e^3=20.1]이 된다. 이제 t에서 a[L]으로 합이 1이 되도록 정규화시키기 위해서 t의 네 개 값의 합을 구하면 176.3이 된다. 이제 a[L]의 값은 t를 176.3으로 나눈 것이 된다. 즉 예시에서 첫 번째 노드의 값은 e의 5승을 176.3으로 나눈 것이며 이는 0.842이다. 만약 이런 z값을 얻었다면 클래스 0이 될 확률은 84.2%가 되는 것이다. 마찬가지로 클래스 1, 2, 3이 될 확률을 구할 수 있다. 이 신경망의 출력값 yhat과도 같은 a[L]은 (4, 1)벡터가 되고 그 안에는 계산한 숫자들(0.842, 0.042, 0.002, 0.114)가 들어있게 된다. 이 알고리즘은 z[L]이라는 벡터를 취해서 합이 1이 되는 네 개의 확률 값을 내놓는다. z[L]에서 a[L]으로 되는 과정을 요약하자면 e를 취해서 임시 변수 t를 얻어 정규화하는 것인데, 해당 과정을 소프트맥스 활성화 함수로도 요약할 수 있다. 즉 a[L]은 z[L] 벡터에 활성함수 g[L]을 적용한 것이다. 이 활성화 함수 g의 특이한 점은 (4,1) 벡터를 받아서 (4,1)벡터를 내놓는다는 것이다. 이전에는 활성함수가 하나의 실수값을 받았는데, 예를 들어 시그모이드나 Relu 활성화 함수 등은 실수를 받아서 실수를 내놓았다. 소프트맥스 활성화 함수의 특이한 점은 정규화를 하기 위해서 입력값과 출력값이 모두 벡터라는 것이다.



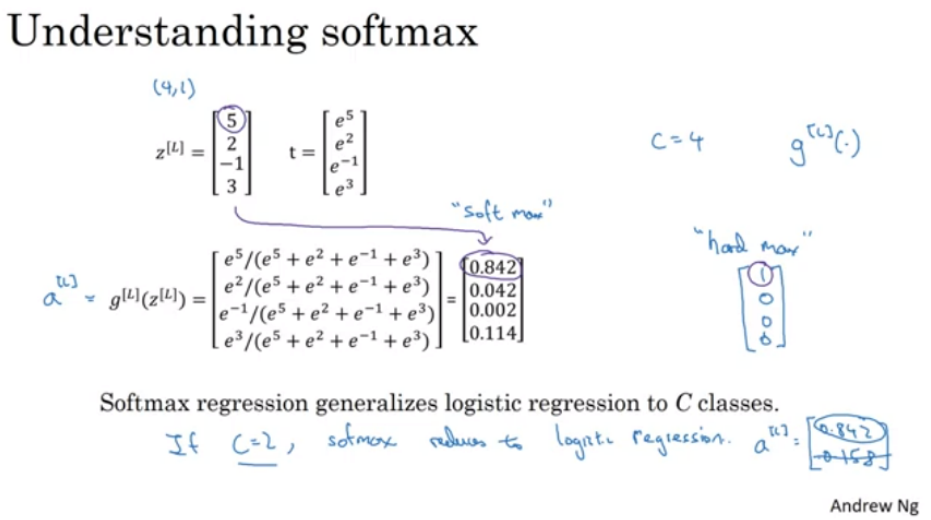
소프트맥스 분류로 할 수 있는 것을 보이기 위해, 예를 들어 x1과 x2의 입력값이 있으며 이 값들이 곧바로 소프트맥스 층에 들어간다고 가정하자. 안에는 서너개의 노드가 있고 출력값은 yhat이다. 이렇게 은닉층이 없는 신경망이다. 여기서 하는 일은 z[1]을 w[1]과 x를 곱한 뒤 b[1]을 더하여 계산하고 출력값 yhat이자 a[1]을 z[1]에 소프트맥스 활성화 함수를 적용시켜 얻는 것이다. 은닉층이 없는 신경망에서 소프트맥스 함수가 무엇을 하는지 감을 잡아보자. 이 예시에서는 입력값 x1과 x2에 대하여 결정 기준을 나타내는 c=3의 클래스를 가진 소프트맥스 층을 사용했다. 여기서 선형적인 기준에 따라 데이터가 세 개의 클래스로 나뉘는데, 우리가 했던 것은 이 그림처럼 학습 세트를 가져와서 비용 함수와 세 개의 선택지에 따라 분류하는 소프트맥스 함수를 학습시키는 것이다. 색깔은 소프트맥스 분류 함수에 따라 출력값을 나타낸 것이고, 입력값은 가장 높은 확률의 출력값에 따라 색을 입혔다. 선형 기준을 갖고 있는 로지스틱 회귀의 일반적인 형태이다. 하지만 클래스는 0,1 또는 0,1,2가 될 수 있다. 그 옆은 소프트맥스 분류 함수가 나타낸 또 다른 경우인데, 세 개의 클래스에 따라 데이터를 학습시켰다. 그 옆은 또 다른 경우이다. 여기서 얻을 수 있는 직관은 두 클래스 사이의 경계가 선형이라는 것이다. 따라서 예시를 보면 노란색과 빨간색 사이에도 선형 경계가 그려져 있고, 보라색과 빨간색, 빨간색과 노란색 사이에도 선형 결정 경계가 그려져 있다. 하지만 다른 선형 함수를 사용해서 공간을 세 개의 클래스로 나눌 수도 있다. 더 많은 클래스를 다룬 예시도 살펴보자. 이 예시는 c=4인 경우인데, 여기서 소프트맥스가 선형 경계를 그리고 있다. 그 뒤로 c=5, c=6인 예시들도 있다. 이렇게 은닉층이 없을 때 소프트맥스 분류 함수가 하는 일을 살펴보았다. 만약 은닉 유닛이 여러 개인, 더 깊은 신경망을 다룬다면 여러 클래스를 분류하기 위해 더 복잡하고 비선형의 경계도 볼 수 있을 것이다.

- Softmax 는 여러개의 클래스 분류시 사용됩니다.
- 마지막 층의 출력값이 주어졌을 때 해당 클래스에 속할 확률을 Softmax 층을 통해서 구할 수 있습니다. 마지막 선형 출력값(z)들을 각각 지수화시켜 임시변수 $t = e^z$ 를 만듭니다. 그후 모든 값들의 합이 1이 될 수 있도록 모든 임시 변수값들의 합을 나눠서 정규화시킵니다.
- $a_i = \frac{e^{z_i}}{\sum_{j=1}^{C} e^{z_j}}$

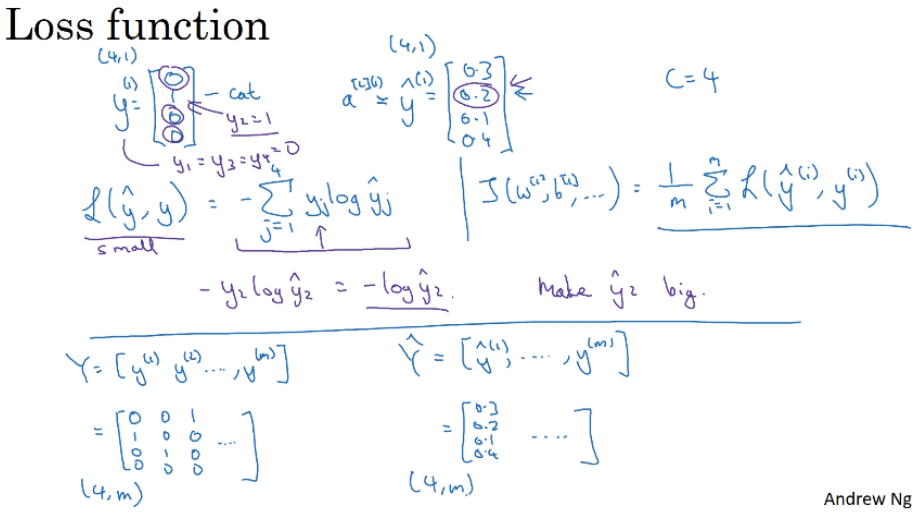
📌 Softmax 분류기 훈련시키기(C2W3L09)

핵심어: Softmax, 손실함수(Loss Function)

지난 강의에서 소프트맥스 층과 소프트맥스 활성화 함수에 대해 배웠다. 이번에는 소프트맥스 분류에 대해 더 깊이 이해하고 소프트맥스 층을 써서 모델을 학습시키는 법을 배워보자.



지난 예시에서는 위쪽 층에서 z[L]을 계산했었는데, 여기서 c=4 개의 클래스를 다루니 z[L]은 (4, 1)개의 벡터이다. 그리고 원소 단위로 e를 취해 임시 변수 t를 얻었다. 끝으로 활성화 함수 g[L]은 소프트맥스 활성화 함수였는데, 이 함수는 t를 합이 1이 되도록 정규화시켰다. 이것은 결국 a[L]과 같은 것이다. 여기서 z[L]의 가장 큰 원소가 5였고, 가장 큰 확률도 첫 번째 확률이다. 소프트맥스라는 이름은 하드맥스와 반대되는 뜻을 가지는데, 하드맥스는 z벡터를 받아와서 이러한 벡터와 대응시킨다. 하드맥스는 z의 원소를 살펴보고 가장 큰 값이 있는 곳에 1을, 나머지에는 0을 대입시키는 방식이다. 가장 큰 원소만 1이고 나머지는 0인 것이다. 소프트맥스는 반면에 부드러운 느낌으로 z를 이런 확률들로 대응시킨다. 여기서 중요한 것은, 소프트맥스 회귀나 활성화 함수가 두 클래스만 다루는 로지스틱 회귀를 일반화했다는 사실을 알아야 한다. 만약 c=2라고 한다면, 즉 소프트맥스에서 c=2라면 결국 로지스틱 회귀와 같아진다.

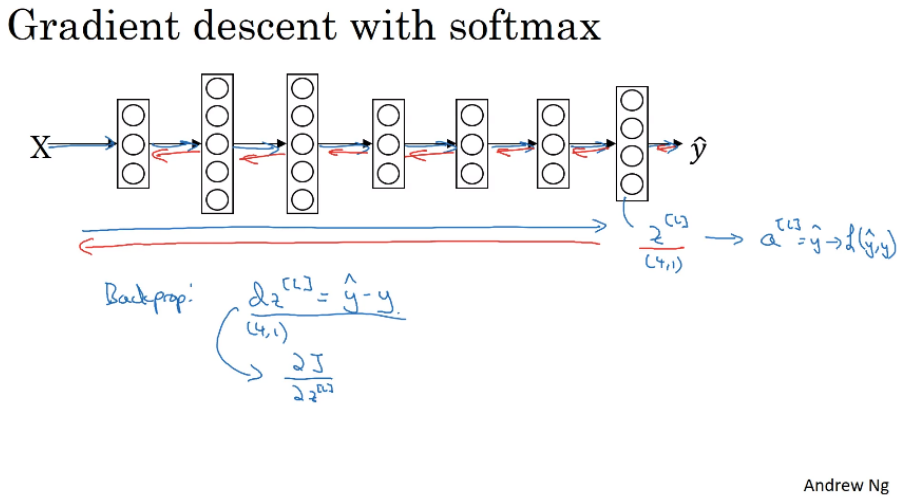


이제 소프트맥스 출력층을 이용해 신경망을 학습하는 방법을 알아보자. 우선 신경망을 학습하기 위해 사용했던 손실 함수를 정의해보자. 예를 들어 보자. 한 샘플이 목표로 하는 출력값이 관측을 기반으로 0, 1, 0, 0이라고 하자. 지난 강의를 생각하면 이 벡터는 고양이를 뜻한다(클래스 1이기 때문). 그리고 우리 신경망의 출력값 yhat은

다음과 같다고 하자. yhat은 합이 1인 확률로 구성된 벡터인데, 이는 곧 a[L]과 같다. 여기서 고양이는 20%에 불과하기 때문에 신경망이 잘 작동한다고 할 수는 없다. 이 때 신경망을 학습시키기 위한 손실함수, 즉 소프트맥스 분류에서 주로 사용하는 손실 함수는 j=1부터 4까지, 일반적으로는 부터 C까지, y_j*log(y_j)의 합의 음수값이다. 위 샘플에서 무슨 일이 일어난 것인지 살펴보자. 여기서 y_1=y_3=y_4=0이며 유일하게 y_2만 1이다. 이 합을 구할 때 y_j가 0이면 고려해주지 않기 때문에 유일하게 남은 항은 -y_2*log(y_2)가 될 것이다.여기서 y_2=1이니 결국 -log(y_2)가 된다.

이제 학습알고리즘이 경사하강법을 이용해서 이 손실 함수의 값을 작게 만들려고 할 것이다. 결국 -log(y_2)값을 작게 만드는 것이다. 그러면 결국 yhat_2의 값을 가능한 한 크게 만들어야 한다(이 값은 확률이므로 1보다 커질 수는 없다). 여기서 말이 되는 것이 입력값 x가 고양이의 사진이었으니 그에 대응하는 출력값인 확률을 최대로 키워야하는 것이다. 즉 일반적으로 손실 함수는 훈련 세트에서 관측에 따른 클래스가 뭐든 간에 그 클래스에 대응하는 확률을 가능한 한 크게 만드는 것이다.

이렇게 하나의 훈련 샘플에서 손실 함수를 보았고, 전체 훈련 세트에 대한 비용 함수 J는 무엇일까? 편향 등의 매개변수를 설정할 때 비용 함수는 우리가 생각하는 것처럼 전체 훈련 세트에서 학습 알고리즘의 예측에 대한 손실 함수를 합하는 것이다. 그리고 이 비용 함수를 최소화하기 위해 경사 하강법을 사용해야 한다. 마지막으로 구현에 관해 살펴보자면 C=4이고 y는 (4, 1)벡터인 상황에서 yhat도 (4,1)벡터이다. 그럼 Y라는 행렬은y[1], y[2]부터 y[m]이 된다. 위에 있는 샘플이 첫 번째 훈련 샘플이라 하면 첫 번째 열은 0100이 되고, 두 번째 샘플은 개, 세 번째 심플은 아무것도 아닌 것으로 계속 진행한다. 그럼 Y는 (4,m)차원의 행렬이 되는 것이다. 비슷하게 Yhat은 yhat(1)부터 yhat(m)을 수평하게 쌓은 것이다. 즉 이것이 yhat(1)이 되고, 그러면 Y는 0.3, 0.2, 0.1, 0.4를 갖게되는 것이다. 그럼 Yhat도 (4,m)차원이 될 것이다.



끝으로 소프트맥스 출력층이 있는 경우 경사하강법을 어떻게 구현할지 살펴보자. 이 출력층이 (4, 1)차원인 z[L]을 계산하며 여기에다 소프트맥스 활성화 함수를 취해서 a[L]또는 yhat을 얻는 것이다. 그러면 그 값을 이용해서 손실 함수를 계산할 수 있다. 이전에 신경망의 정방향 전파를 다룬 적이 있는데, 손실 함수를 구하는 데 썼었다.

역방향 전파나 경사하강법은 어떨까? 역방향 전파에서 초기화를 위한 핵심 단계, 핵심이 되는 식은 마지막 층에서 z[L]의 미분이 (4,1) 벡터인 yhat에서 (4,1)벡터인 y를 뺀 것과 같다는 것이다. 즉 클래스가 4개일 때 모두 (4,1)벡터가 되는 것이다. 일반적으로는 (C,1)차원이 된다. 우리의 일반적인 정의를 빌리자면 여기서 dz[L]은 비용함수를 z[L]에 대해 편미분한 것이다. 이렇게 시작해서 신경망 전체에 미분을 역방향 전파로 구하는 것이다.

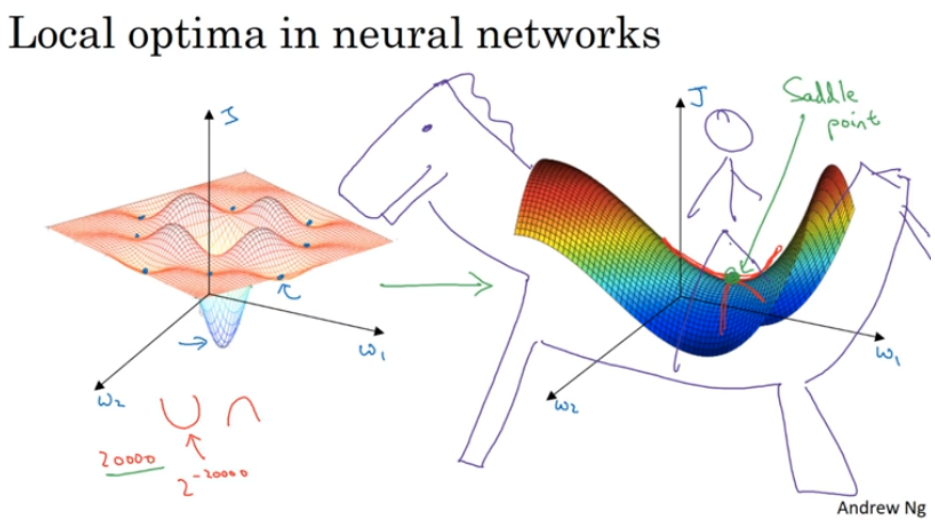
- 학습을 위한 손실함수는 다음과 같습니다.
  - $L(\hat{y}, y) = -\sum_{j=1}^{4} y_j \log y_j$
  - $y = [0, 1, 0, 0]$
  - $\hat{y} = [0.3, 0.2, 0.1, 0.4]$
  - 두번째를 제외한 나머지 y_j 값은 0 이기 때문에 $-log(y_2)$ 값만 남을 것입니다.
  - 즉 이 값을 최소화 하여 클래스2 이 될 확률을 최대화 시키는 것입니다.
- Softmax 와 손실함수를 결합한 역전파의 값은 아래와 같습니다.
  - $dz^{[L]} = \hat{y} - y$

8. 프로그래밍 프레임워크 소개

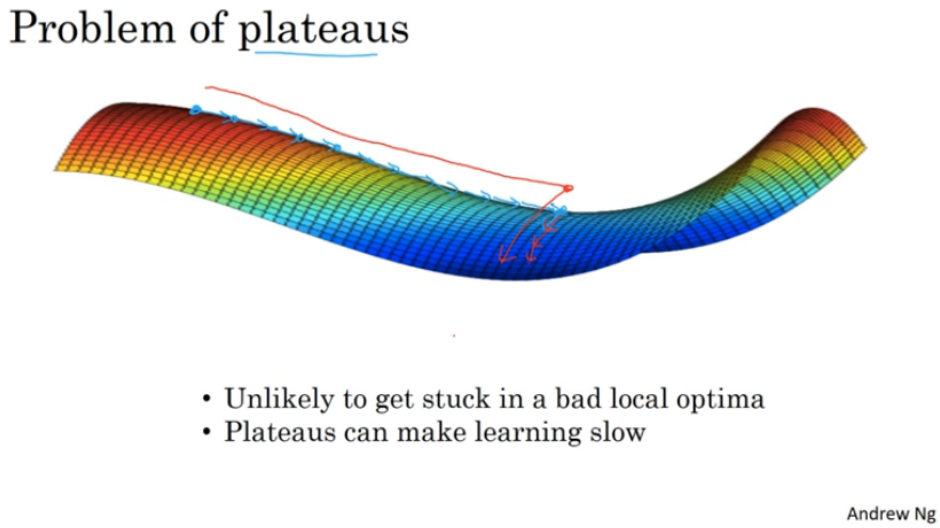
📌지역 최적값 문제(C2W3L10)

핵심어: 지역 최적값(local optima), 최적화 문제(optimiza tion problem), 안장점(saddle point), 안정지대(plateaus)

딥러닝 학문 초기에는 나쁜 지역 최적값에 갇히는 최적화 알고리즘을 많이 사용하곤 했다. 하지만 딥러닝 이론이 계속 발전하면서 지역 최적값에 대한 이해도 바뀌었다. 이제 지역 최적값에 대해 어떻게 생각하고 딥러닝 문제 안의 최적화 문제를 알아보도록 하자.



사람들이 지역 최적값에 대해 고민할 때 위와 같은 그림을 생각한다. w_1과 w_2라는 매개변수 최적화한다고 할 때 이 면적의 높이가 비용 함수가 될 것이다. 이 그림에서 지역 최적값이 많아 보이는데, 이런 곳들에서 경사하강법 등의 알고리즘이 전역 최적값에 도달하기 전 지역 최적값에 갇혀버리기 쉽다. 이렇게 2차원에서 그림을 그린다면 서로 다른 지역 최적값이 많은 그림을 쉽게 접할 수 있다. 이렇게 낮은 차원의 그림을 통해서 직관을 얻곤 한다. 하지만 직관이 항상 옳지만은 않은데, 경사가 0인 점은 대부분 지역 최적값이 아니라 비용 함수의 경사가 0인 경우 대게 안장점이다. 역시 경사가 0인 점 중에 하나이다. 마찬가지로 w1과 w2일테고 높이는 비용 함수 J이다. 사실 고차원의 함수에서 경사가 0이면 각 방향에서 볼로 함수나 오목 함수가 되기 마련인데, 예를 들어 20000차원의 공간에서 지역 최적값이 되기 위해서는 20000개의 방향이 모두 위와 같이 생겨야 하며 그러할 확률은 매우매우 낮다. 대신 어떤 방향에서는 위로 굽어져있고 어떤 방향에서는 아래로 굽어져있는 형태가 주로 발생할 것이다(모두 위로 굽어있는 것보다는). 따라서 고차원 공간에서는 오른쪽 그림처럼 지역 최적값보다 안장점이 되기 쉽다. 여기서 왜 안장점이라고 불리냐하면 이 모양이 말에 얹는 안장과 비슷하기 때문이다. 따라서 경사가 0인 이 점을 안장점이라고 부르는 것이다. 딥러닝의 역사에서 배울 수 있는 것은 왼쪽 그림처럼 낮은 차원의 공간에서 얻었던 직관이 학습 알고리즘이 높은 차원에서 돌아갈 때 적용되지 않을 수 있다는 것이다. 왜냐하면 20000개의 매개변수가 있을 때 J는 20000차원의 벡터에 대한 함수일 것이고, 지역 최적값보다 안장점을 훨씬 많이 볼 수 있을 것이다.



그럼 지역 최적값이 문제는 아니고 무엇이 진짜 문제인 것일까? 바로 안정지대가 학습을 아주 지연시킬 수 있다는 것이다. 안정 지대는 미분값이 아주 오랫동안 0에 가깝게 유지되는 지역을 말한다. 만약 위쪽에서 출발한다면 경사하강법에 의해 면을 따라서 아래로 움직일 것이다. 여기서 경사가 0이거나 0에 가까울테니 면이 거의 평평할 것이고 안정지대에서 여기까지 이르는 데 아마 아주 오랜 시간이 걸릴 것이다. 그리고 왼쪽이나 오른쪽에 무작위로 작은 변화가 주어지면 알고리즘이 안정지대를 벗어날 수 있다. 하지만 이 점에 도달하기 전에 아주 긴 시간동안 경사를 탄 후에야 안정지대를 벗어날 수 있다.

여기서 알아야 할 것은, 충분히 큰 신경망을 학습시킨다면 지역 최적값에 갇힐 일이 잘 없다는 점이다. 여러 매개변수와 비용함수 J가 상대적으로 고차원에서 정의되어야 한다는 것이다. 하지만 둘째로 안정지대가 문제이다. 학습속도가 매우 느려지기 때문이다. 여기서는 모멘텀이나 RMSprop, Adam 등의 알고리즘의 도움을 받을 수 있다. 이런 경우 Adam과 같은 최적화 알고리즘이 안정지대 내에서 움직이거나 벗어나는 속도를 올릴 수 있다.

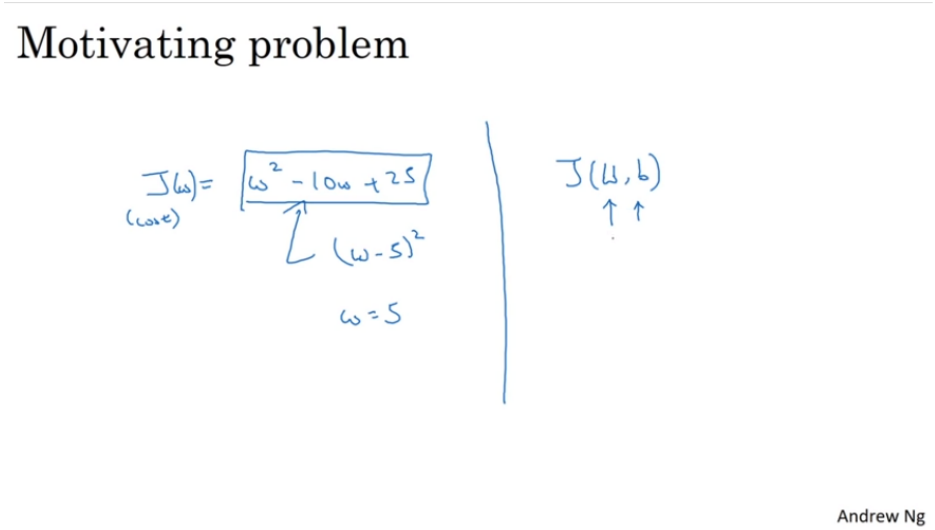
신경망이 일반적으로 고차원에서 최적화 문제를 해결할 때 사실 어떤 사람도 이 공간이 어떻게 생겼는지 잘 모를 것이다. 이 공간에 대한 이해도 계속 발전하고 있고, 이 강의를 통해 최적화 알고리즘이 맞닥뜨릴 법한 문제에 대해 직관을 얻는 것이 중요하다.

- 고차원 비용함수에서 경사가 0 인경우는 대부분 지역 최적값이 아니라 대개 안장점 입니다.
- 안장점으로 향하는 구간인 안정지대는 미분값이 아주 오랫동안 0에 가깝게 유지되는 지역을 말합니다.
- 대개 충분히 큰 Network 학습시 지역 최적값에 갇이는 일은 거의 없습니다.
- 안정지대의 문제점은 경사가 거의 0에 가깝기 때문에 학습속도가 느려집니다. 또한, 다른 쪽으로 방향변환이 없다면 안정지대에서 벗어나기 어렵습니다. 이는 Adam, RMSprop 등 알고리즘이 해결해줍니다.

📌 Tensorflow(C2W3L11)

핵심어: Tensorflow

좋은 딥러닝 프레임워크가 많이 있는데, 그 중 하나는 텐서플로우이다.

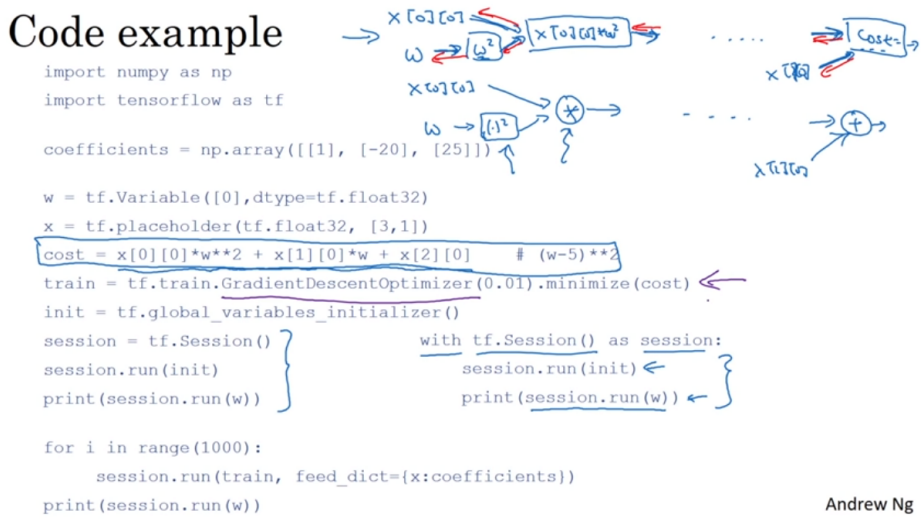


문제를 하나 다뤄보자. 최소화하고자 하는 비용함수 J가 있고, 아주 간단한 비용 함수인 J(w)=w^2-10w+25라는 비용함수를 사용하고자 한다. 비용함수가 이렇게 주어졌으며, 이 함수는 (w-5)^2와 같다. 이 제곱식을 풀면 이 비용함수 식을 얻을 수 있다. 따라서 이 식을 최소로 하는 w는 5이다. 이를 모른다고가정하고 비용함수를 보자. 텐서플로우에서 이 식을 어떻게 최소화하는지 보자. 프로그램이 모든 매개변수와 관련이 있어서 다소 복잡한 비용 함수 J(w, b)를 학습시키는 신경망과 구조가 비슷하므로 비슷한 방법으로 텐서플로우를 이용해서 자동으로 이 비용 함수를 최소화하는 w와 b의 값을 찾을 수 있다. 하지만 왼쪽의 단순한 경우에서 시작해보도록 하자.

```
In [1]: import numpy as np
        import tensorflow as tf

In [11]: coefficients = np.array([[1.], [-20.], [100.]])

         w = tf.Variable(0,dtype=tf.float32)
         x = tf.placeholder(tf.float32, [3,1])
         #cost = tf.add(tf.add(w**2,tf.multiply(-10.,w)),25)
         #cost = w**2 - 10*w + 25
         cost = x[0][0]*w**2 + x[1][0]*w + x[2][0]
         train = tf.train.GradientDescentOptimizer(0.01).minimize(cost)

         init = tf.global_variables_initializer()
         session = tf.Session()
         session.run(init)
         print(session.run(w))
         0.0

In [12]: session.run(train, feed_dict={x:coefficients})
         print(session.run(w))
         0.2
```

tf로 텐서플로우를 불러오고, 매개변수 w를 정의한다.(tf.Variable 이용). 그리고 비용함수 w^2-10w+25를 정의하자. 또한 train을 tf.train.GradientDescentOptimizer로 정의한다. 학습속도는 0.01로 하고 목표는 cost를 최소화하는 것이다. 이제 관용적인 식들을 쓰도록 하자. init = tf.global_variables_initializer(), session=tf.Session(), session.run(inti)으로 전역변수를 초기화한다. 아직 학습 알고리즘을 실행하지 않았기 때문에 w는0이다. session.run(train)을 실행하면 한 단계 수행이 된다. 한 단계의 경사하강법 후 w 값이 0.1이 되었으며, 1000 단계 실행하면 w값이 4.99999로 계산되었다. w의 최적값은 5이기 때문에 이에 아주 가까워진 모양이다. w는 최적화하고 싶은 변수이니 변수

로 선언했고, add와 muliply 등으로 비용 함수를 정의할 수 있다. 그리고 텐서플로우는 add와 multiply 등 여러 함수의 미분을 어떻게 계산할지 알게 되는 것이다. 따라서 정방향 전파만 잘 구현하면 경사계산법에서 역방향 전파를 잘 계산해낸다. 그것들은 이미 add와 multiply, 그리고 square 함수에 구현되어 있다. 텐서플로우는 또한 +, -처럼 일반적인 연산자도 지원한다.



텐서플로우의 또다른 특징을 하나 더 살펴보자. 위의 경우에서는 w에 관해 고정된 함수를 최소화한다. 최소화하고싶은 함수는 학습 세트와 관련한 함수일 수도 있는데, 따라서 어떤 학습 데이터 x를 갖고있든지 간에 신경망을 학습시키면 데이터 x가 바뀔 수 있다. 그러면 학습 데이터를 텐서플로우에서 어떻게 사용할 수 있을까? 학습 데이터의 역할을 하는 x를 찾아보자. 또는 실제로 학습 데이터가 x, y일 때 예시에서 x만 쓴다고 생각할 수도 있다. 비용 함수에서 계수는 고정되어 있으며 cost를 x[0][0] * w^2 + x[1][0]*w + x[2][0]로 대체하는 것이다. 그럼 이제 x는 이 이차 함수의 계수를 조절하는 데이터가 된다. 이 플레이스홀더 함수는 텐서플로우에게 x에 값을 나중에 줄 것이라고 말한다.

이제 다른 배열을 정의한 후 이 데이터를 x에 집어넣는다. 이제 이 coefficient 배열을 변수 x에 집어넣을 방법이 필요한데, 학습 중에 x를 제공하는 문법이 필요하그래서 feed_dict=()를 사용하는 것이다. 그럼 이전과 비슷한 결과를 얻게 된다. 그리고 이 이차함수의 계수를 바꾸고 싶다면, 이를 최소화하는 w값이 또 다르게 변할 것이다(10과 아주 가까움). 텐서플로우에서 이 플레이스홀더는 값을 나중에 넣는 변수이다. 이 방법을 사용하면 학습 데이터를 비용 함수에서 쉽게 얻을 수 있다. 우리가 비용 함수에 데이터를 불러오는 문법이 이것인데, 학습을 반복할 때 feed_dict에서 x를 coefficient로 두면 되는 것이다. 만약 각 학습에서 미니 배치 경사하강법을 쓴다면 서로 다른 미니 배치를 집어넣어야 한다. 그럼 각 학습마다 feed_dict에 학습 세트의 다른 부분 집합을 넣어야 할 것이다. 데이터를 기다리고 있느 비용 함수에 서로 다른 미니 배치를 집어넣는 것이다.

텐서플로우의 강점은 우리가 비용함수를 정의해주기만 하면 된다는 것이다. 그럼 미분을 해서 경사하강법이나 Adam 등 최적화 기법을 쓰는 것을 한 두줄의 코드로 구

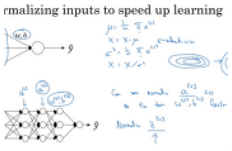
현할 수 있다.

session = tf.Session(), session.run(init), print(session.run(w))이 가장 중요하고도 많이 쓰이는 코드인데, 세션을 시작하기 위해 session을 tf.session으로 두고 session을 이용해 init을 run하며 마지막으로 session에서 w를 계산해서 그 결과를 출력하는 것이다. 여기서 with은 수 많은 텐서플로우 프로그램에서 사용된다. 하지만 파이썬에서 with문을 실행하고 있을 때는 오류나 예외의 경우에 더 깔끔하다. 텐서플로우 프로그램의 핵심은, 비용함수만 명시하면 자동으로 미분을 계산하고 비용함수를 최소화하는 데 있다. cost~ 코드가 하는 일은 텐서플로우가 계산 그래프를 그리도록 하며, 계산 그래프는 x[0][0]와 w를 가져온 뒤 w를 제곱하고 x[0][0]와 w^2를 곱해서 x[0][0]*w^2를 얻는 식이다. 이런 과정을 통해서 x[0][0]*x^2+x[1][0]*w+x[2][0]라는 비용함수를 만들어내는 것이다. 텐서플로우의 좋은 점은 비용 함수를 계산해낸 계산 그래프를 통해 정방향 전파를 구현하면 필요한 역방향 함수는 이미 구현되어 있다는 것이다. 심층신경망을 학습시키기 위해서는 저방향과 역방향 함수가 필요한데, 텐서플로우 같은 프레임워크에서는 필요한 역방향 함수가 이미 구현되어 있다. 그래서 정방향 함수를 구현하기 위해 내장 함수를 사용하는 것이다. 그러면 우리 입장에서 아무리 복잡한 함수나 미분이라도 자동으로 역방향 함수를 구해서 역방향 전파를 구현하게 되는 것이다. 따라서 따로 역방향 전파를 구현할 필요가 없는 것이다. 이것이 프로그래밍 프레임워크를 사용했을 때 얻을 수 있는 효율성 중 하나이다. 게다가 한 줄의 코드 내에서 아주 빠르게 더 나은 최적화 알고리즘으로 바꿀 수 있다(경사하강법 대신 Adam을 사용하는 등). 최근의 딥러닝 프로그래밍 프레임워크들은 이러한 기능을 지원하고 있어서 복잡한 신경망도 쉽게 코딩할 수 있다.

'DL 스터디&프로젝트' 카테고리의 다른 글

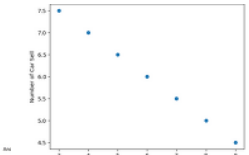
[Euron 중급 세션 12주차] 딥러닝 2단계 6. 배치 정규화 (1)	2023.11.27
[Euron 중급 세션 11주차] 딥러닝 2단계 5. 하이퍼파라미터 튜닝 (1)	2023.11.20
[Euron 중급 세션 10주차] 캐글 필사 과제 - Pytorch Tutorial for Deep Learning Lovers (2)	2023.11.20
[Euron 중급 세션 10주차] 딥러닝 2단계 4. 최적화 알고리즘 (1)	2023.11.13
[Euron 중급 세션 9주차] 딥러닝 2단계 3. 최적화 문제 설정 (2)	2023.11.06

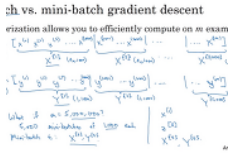
관련글



[Euron 중급 세션…

[Euron 중급 세션…


[Euron 중급 세션…



[Euron 중급 세션…