Lecture 6-1

Title	Training Neural Networks I
slide	http://cs231n.stanford.edu/slides/2017/cs231n_2017_lecture6.pdf

이번 시간에는 nerural network를 학습시키는 방법을심도깊게 살펴보겠습니다

복습

우리는 함수를 computational graph로 표현하는 방법을 배웠고, 이를 통해 어떤 함수든 표현할 수 있게 되었다. 또한 Linear Layer들을 여러 층으로 쌓은 Neural network에 대해서 자세히 배웠다. "Spatial structure" 를 사용하기 위해 Conv Layer를 사용하는 NN의 특수한 형태인 CNN에 대해서도 배웠다. CNN에서는 다수의 Conv 필터가 입력 이미지를 슬라이딩 하면서 계산한 값들이 모여 출력 Activation map을 만들게 된다. Activation map은 필터의 개수만큼 존재하며 각 map은 입력의 공간적인 정보를 보존하고 있다. 그 다음으로 하고 싶은 것은 가중치, 즉 파라미터를 업데이트 하는 것이다. Optimization을 통해 네트워크의 파라미터를 학습할 수 있으며, Mini-batch Stochastic Gradient Descent을 통해 파라미터를 업데이트 할 수 있다. Mini-batch SGD는 우선 데이터의 일부만 가지고 Forword pass를 수행한 후에 Loss를 계산하고, gradient를 계산하기 위해서 backprop를 수행하여 파라미터를 업데이트한다.

Activation Function(활성함수)

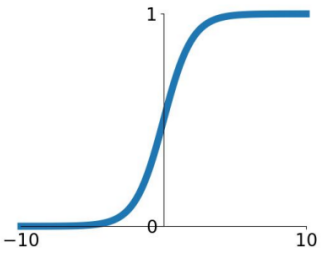
다양한 종류의 활성함수와 그것들 간의 Trade-off에 대해 다뤄보도록 한다.

Sigmoid

Activation Functions

$$\sigma(x) = 1/(1 + e^{-x})$$

- Squashes numbers to range [0,1]
- Historically popular since they have nice interpretation as a saturating “firing rate” of a neuron

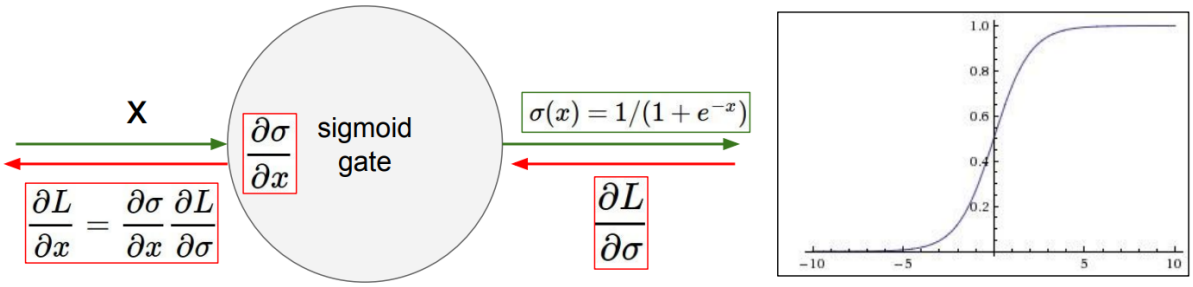


Sigmoid

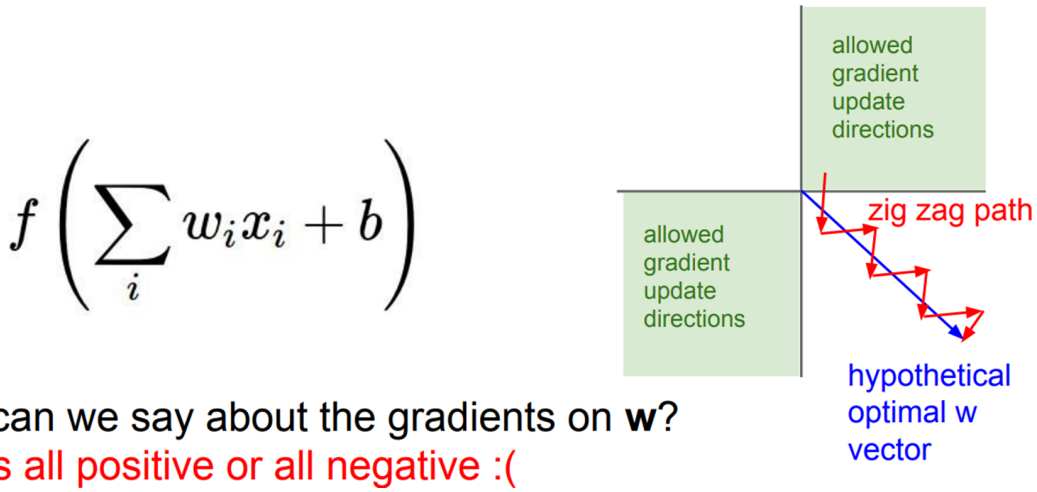
우선 Sigmoid부터 살펴보자. **Sigmoid는 1/(1+e^-x)로 표현**할 수 있다. 이것은 각 입력을 받아 그 입력을 [0, 1] 사이의 값이 되도록 해준다. **입력의 값이 크면 Sigmoid의 출력은 1에 가까울 것이고, 입력의 값이 작으면 0에 가까울 것이다.** 근처 구간(rigime)을 보면 "/ 모양"으로 선형 함수 같아 보인다. Sigmoid가 일종의 뉴런의 firing rate를 saturation 시키는 것으로 해석할 수 있기 때문에 Sigmoid는 역사적으로 아주 유명했다. 후에 배울 ReLU 같은 것이 실제로 생물학적 타당성이 더 크다는 것이 밝혀졌지만, Sigmoid 또한 이런 식으로 해석할 수 있다.

문제점

Sigmoid를 자세히 살펴보면 문제점이 몇 가지 있다.



첫 번째는 Saturation되는 것이 gradient를 없앤다는 것이다. 데이터 X가 있고 출력이 있을 때, Backprop에서 Gradient는 어떻게 될까? local sigmoid function의 gradient는 (dSigma/dX)*(dL/dSigma)가 되고, 이런 값들이 계속 연쇄적으로 Backporp될 것이다. X가 0이면 0이 아닌 gradient를 얻게 되고, 이 구간에서 잘 동작하겠지만, X가 양의 큰 값, 혹은 음의 큰 값일 경우에는 문제가 된다. 만약 X가 -10, 즉 음의 큰 값이면 sigmoid가 flat 하게 되어 gradient는 0이 되고(dX=0), 거의 0에 가까운 값이 backprob 될 것이다. 따라서 이 부분에서 gradient가 죽어버리게 되고 밑으로 0이 계속 전달되게 된다. 이는 X가 10일 때, 즉 양의 큰 값일 때에도 마찬가지의 현상이 발생한다.

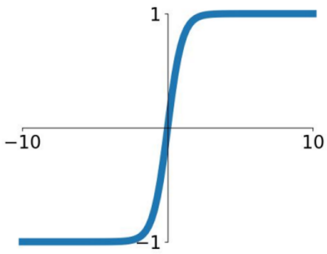


두 번째는 sigmoid 의 출력이 zero centered 하지 않다는 것이다. 만약 뉴런의 입력이 항상 양수인 경우(모든 x가 양수)를 생각해보자. 이때 W에 대한 gradient를 한번 구해보면, 우선 dL/d(활성함수) 를 계산하여 전달한다. Local gradient는 X가 되기 때문에 모든 X가 양수라면, gradient는 전부 "양수" 또는 "전부 음수"가 된다. 다시 말해서, dL/d(활성함수)가 넘어온 값이 음수 또는 양수가 될 것이고, 이 값과 local gradient (X)를 곱한 값은 그냥 부호만 바뀐 X가 될 것이다. 이는 W가 모두 같은 방향으로만 움직일 것임을 의미한다. 파라미터를 업데이트 할 때 다 같이 증가하거나 다 같이 감소하거나 할 수 밖에 없게 되는데, 이런 gradient 업데이트는 아주 비효율적이다.

오른쪽 그림은 W가 2차원인 예제이다. 전부 양수 또는 음수로 업데이트 된다는 것은 gradient가 이동할 수 있는 방향은 4분면 중 1, 3사분면 밖에 안 될 것이다. 그래서 이론적으로 가장 최적의 W 업데이트가 파란색 화살표라고 한다면, 저 방향으로 움직일 수 없기 때문에 gradient는 파란색 방향으로 내려갈 수 없다. 그렇기 때문에 여러 번 gradient 업데이트를 수행해줘야 한다. 가령 빨간 화살표 방향과 같이, gradient가 이동 가능한 방향으로만 이동을 할 수 있게 된다. 이것이 우리가 일반적으로 zero-mean data을 원하는 이유이다. 입력 X가 양수/음수를 모두 가지고 있으면 전부 같은 방향으로 움직이는 일은 발생하지는 않을 것이다.

세 번째는 exp()로 인하여 계산 비용이 크다는 것이다. 그러나 이것은 앞선 두 문제에 비해 그렇게 큰 문제는 아니다.

tanh

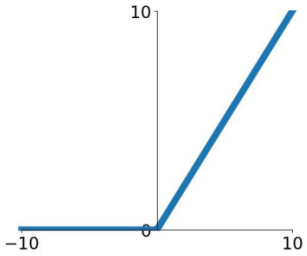


tanh(x)

- Squashes numbers to range [-1,1]
- zero centered (nice)
- still kills gradients when saturated :(

tanh는 sigmoid 랑 유사하다. 차이점은 범위가 [-1, 1]이라는 점이다. 이로 인해 발생한 가장 큰 차이점은 zero-centered 라는 것이다. 하지만 gradient가 평평해 지는 구간이 있기 때문에 saturation로 인하여 여전히 Gradient가 죽는 문제가 발생한다. tanh는 sigmoid보다는 조금 낫지만 그래도 여전히 문제점이 있다.

ReLU

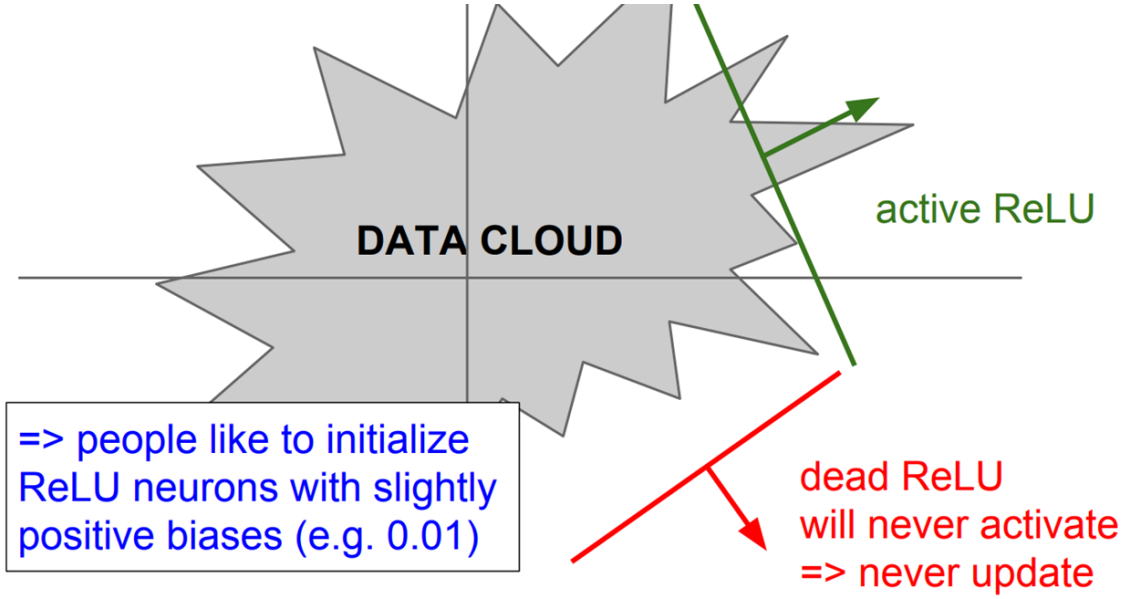


- Does not saturate (in +region)
- Very computationally efficient
- Converges much faster than sigmoid/tanh in practice (e.g. 6x)
- Actually more biologically plausible than sigmoid

ReLU
(Rectified Linear Unit)

ReLU의 함수는 f(x)=max(0, x)이다. 이 함수는 element-wise 연산을 수행하며 입력이 음수면 값이 0이 되고 양수면 입력 값 그대로 출력한다. ReLU는 상당히 자주 쓰이며, 기존 sigmoid와 tanh한테 있던 문제점들을 약간 해소한다. 우선 ReLU는 양의 값에서는 saturetion되지 않기 때문에 적어도 입력 스페이스의 절반은 saturation 되지 않는다. 또한 ReLU는 계산 효율이 아주 뛰어나다. 기존의 sigmoid는 함수 안에 지수 항이 있어 연산 속도가 느리지만, ReLU는 단순히 max 연산이므로 계산이 매우 빠르다. ReLU를 사용하면 실제로 sigmoid나 tanh보다 수렴 속도가 훨씬 빠르다. 그리고 생물학적 타당성도 ReLU가 sigmoid보다 크다. 실제 신경과학적인 실험을 통해 뉴런을 관찰하여 입/출력 값을 확인해보면, sigmoid보다 ReLU에 더 가깝다는 것을 알 수 있다.

ReLU에 문제가 하나 있다면, ReLU는 또 다시 zero-centered가 아니라는 것이다. tanh가 이 문제는 해결했지만, ReLU는 다시 이 문제를 가지고 있게 된다. ReLU가 가진 문제점 또 한 가지는 양의 수에서는 saturation 되지 않지만 음의 수에서는 saturation 된다는 것이다. 만약 X가 -10이면 gradient는 0이 되고, x가 0일 때에도 마찬가지이다. x가 10일 때는 gradient가 선형 영역(linear regime)에 속한다. 따라서 기본적으로 ReLU는 gradient의 절반(음~0)을 죽이게 된다. 이러한 현상을 dead ReLU 라고 한다.



만약 traing data를 DATA CLOUD라고 생각해보자. 각 평면(초록 빨강 직선)이 각 ReLU를 의미한다. 이를 통해 ReLU에서는 평면의 절반만 activate 되게 된다는 것을 알 수 있다. 그리고 ReLU가 data cloud에서 떨어져 있는 경우에는 "dead ReLU" 가 발생할 수 있다. dead ReLU에서는 activate 가 일어나지 않고 update되지 않는다. active ReLU에서는 일부는 active되고 일부는 active하지 않을 수 있다. 첫 번째 경우는 초기화를 잘 못한 경우이다. 위의 dead ReLU처럼 생긴 경우인데, 가중치 평면이 data cloud에서 멀리 떨어져 있는 경우이다. 이런 경우에는 어떤 데이터 입력에서도 activate 되는 경우가 존재하지 않을 것이고 backporp이 일어나지 않을 것이다. 더 흔한 경우는 Leraning rate가 지나치게 높은 경우이다. 처음에 "적절한 ReLU" 로 시작하더라도 만약 update를 지나치게 크게 해버려서 가중치가 날뛴다면 ReLU가 데이터의 manifold를 벗어나게 된다. 실제로 학습을 다 시켜 놓은 네트워크를 살펴보면 10~20% 가량은 dead ReLU가 되어 있으며, 이 정도의 수치는 네트워크 학습에 크게 지장이 있지는 않다. Update시에 active ReLU가 될 가능성을 조금이라도 더 높혀 주기 위해서 ReLU를 초기화할 때 positive biases를 추가해 주는 경우도 있지만 이것이 도움이 된다는 의견도 있고, 그렇지 않다는 의견도 있다.