Batch Normalization

Reference Paper: Batch Normalization: Accelerating Deep Network Training by Reducing Internal Covariate Shift, Sergey Ioffe et al.

1. Introduction

딥러닝이 발전함에 따라 효율적인 gradient descent에 대한 연구가 이루어졌다. SGD, SGD의 변형인 momentum이나 Adagrad 등이 발표되었다. 기본적으로 SGD는 단계별로 훈련이 진행되며, 각 단계에서 크기가 m인 $mini-batch$가 사용된다. mini-batch는 true gradient을 근사하는데 사용되고 그 개수가 증가할수록 true gradient과 유사해진다. 또한 mini-batch를 이용하면 전체 데이터 N개를 한번에 계산하는 GD에 비해서 각 mini-batch의 gradient를 parallel하게 계산할 수 있기 때문에 훨씬 효율적이다.

SGD는 이러한 장점이 있지만 여러 단점도 있다. 최적화에 사용되는 learning rate를 아주 세심하게 tuning해야 하고 parameters의 모수에 영향을 많이 받는다. 훈련 과정에서 각 층(layer)의 입력 값은 선행 층의 모수에 영향을 받기 때문에 모수의 작은 변화도 네트워크 층이 깊어질수록 그 영향이 커질 수 있다.

모수가 업데이트됨에 따라서 층의 입력 값의 분포가 바뀐다는 점은 각 층이 지속적으로 새로운 분포를 adapt해야 하기 때문에 문제를 야기한다. SGD의 주요 문제인, 입력 값이 분포가 달라지는 현상을 **covariate shift**라고 부른다.

covariate shift는 모든 learning system의 문제로 볼 수 있는데, sub-network의 예시를 보자. 아래와 같이 F_1, F_2는 임의의 변환인 네트워크를 생각해보자.

$$\ell = F_2(F_1(u, \Theta_1), \Theta_2)$$

모수 Θ_1, Θ_2는 loss인 ℓ을 최소화하며 학습된다. 만약 $x = F_1(u, \Theta_1)$라고 둔다면 Θ_2을 학습하는 것은 x가 입력으로 주어지는 sub-network로 볼 수 있다.

$$\ell = F_2(x, \Theta_2)$$

예를 들어, gradient descent는

$$\Theta_2 \longleftarrow \Theta_2 - \frac{\alpha}{m}\sum_{i=1}^{m}\frac{\partial F_2(x_i, \Theta_2)}{\partial \Theta_2}$$

로 볼 수 있고 이는 하나의 네트워크 F_2와 입력 x의 gradient descent와 동일하다. 따라서 층의 입력 값이 동일한 분포를 가져야 한다는 점은 sub-network에서도 동일하게 적용된다.

sub-network의 입력 값의 분포을 고정하는 것은 sub-network의 밖에 있는 층에 좋은 결과를 가져올 수 있다. 예를 들어 sigmoid activation function인 $z = g(Wu + b)$를 생각하자. 여기서 u는 입력 층이고 W는 가중치 행렬, b는 편향, $g(x) = \dfrac{1}{1 + exp(-x)}$이다. sigmoid의 가장 큰 단점은 입력 값이 커지거나

작아진다면 gradient가 vanish한다는 점이다. 극단 값에 갈수록 함수 값의 변화가 거의 없기 때문에 여기서 gradient는 0에 가깝고 backpropagation시에 upstreaming gradient가 0에 가까워져서 이를 local gradient에 곱해도 결국 0에 가까워지는 문제점이 있다. 게다가 네트워크의 층이 깊어질수록 입, 출력 값의 범위가 다양해지고 이를 조절하지 않는다면 언제든 sigmoid에서 gradient가 vanish할 위험이 있다.
물론 sigmoid의 이러한 단점 때문에, 이를 activation function으로 잘 안 쓰기도 하고 대신 ReLU를 많이 사용한다. 하지만 그럼에도, 입력 값을 잘 조정해서 극단 값을 가지지 않도록 조절한다면 sigmoid의 saturated regime에 덜 stuck될 것이다.

본 연구에서는 이렇게 입력 값이 바뀌며 발생하는 문제점을 해결하기 위해 $Batch - Normalization$을 제안한다.

2. Towards Reducing Internal Covariate Shift
앞서, **Internal Covariate Shift**을 훈련 중 네트워크 모수의 변화로 인해, 네트워크 activation의 분포가 변하는 것으로 정의했다. 입령 층 x의 분포를 고정함으로써, 이를 방지하고 훈련 속도를 높일 수 있음은 선행 연구로 밝혀진바 있다. 이를 또 다른 표현으로 입력 층을 whiten한다고 하는데, activations을 whiten 하는 것을 매 훈련마다 할 수도 있다. 하지만 이러한 변경이 최적화 과정에 배치되면 gradient descent step은 정규화 (normalization) 이 업데이트되도록 요구하는 방식으로 모수를 업데이트 할 수도 있고 이는 gradient step의 효과를 줄인다.
예를 들어, 입력으로 u을 받고 편향 b를 더하고 훈련 데이터로부터 계산된 평균을 뺌으로써 정규화하는 층을 생각해보자.

$$\widehat{x} = x - E[x]$$

$$x = u + b,\ \mathcal{X} = \{x_{1\cdots N}\}(training\ set),\ E[x] = \frac{1}{N}\sum_{i=1}^{N} x_i$$

만약 gradient descent step이 b에 대한 $E[x]$의 의존성을 무시한다면 $b \longleftarrow b + \Delta b,\ \Delta b \propto -\partial\ell/\partial\widehat{x}$로 업데이트할 것이다. 따라서,

$$u + (b + \Delta b) - E\left[u + (b + \Delta b)\right] = u + b - E\left[u + b\right]$$

따라서 b를 업데이트하고, 다시 정규화를 할텐데, 그에 따른 결과가 b를 업데이트 하기 이전과 동일하므로 gradient descent step을 한 의미가 없어지고 loss 또한 변화가 없다. 이러한 문제점은 위 네트워크처럼 centering만 하는 것이 아니라 scaling까지 한다면 심각해질 수 있다.
위의 이슈는 gradient descent 최적화 작업이 정규화가 일어나는 곳을 고려하지 않는다는 점이다. 이러한 문제점은 어떤 모수 값에 대해서도 네트워크가 항상 적절한 분포를 가지는 activations을 만들어낸다면 해결할 수 있다.
x를 layer input 벡터로, $\mathcal{X}$를 훈련 데이터 세트라고 하자. 정규화는 아래 변환으로 쓸 수 있다.

$$\widehat{x} = Norm(x, \mathcal{X})$$

이는 주어진 layer input 뿐만 아니라 모든 데이터, $\mathcal{X}$에 의존하는 형태이다. 정규화 과정에서 표본 평균이나 표본 분산을 사용하기 때문에, $\mathcal{X}$에도 의존하는 것이다. backpropagation을 위해, 아래 Jacobian matrix을 계산해야 한다.

$$\frac{\partial Norm(x, \mathcal{X})}{\partial x}, \frac{\partial Norm(x, \mathcal{X})}{\partial \mathcal{X}}$$

바로 두 번째 term을 무시하는 것이 위에서 gradient descent의 효과가 없어지는 현상을 유발하는 것이다.

이러한 틀에서 layer input을 whiten하는 것은 계산량이 엄청난데, 그 이유는 $x \in \mathcal{X}$에 대한 covariance matrix인 $Cov(x) = E[xx^T] - E[x]E[x]^T$와 $Cov(x)^{-1/2}(x - E[x])$을, 또한 backpropagation을 위해 이들의 derivatives을 구해야하기 때문이다. 이 점때문에 미분 가능하고, 전체 훈련 세트를 고려하지 않는 방법을 찾게 되었다.

3. Normalization via Mini-Batch Statistics

모든 층의 입력을 정규화하는 것은 비용이 많이 들고, 미분 가능하다는 보장도 없다. 따라서 아래와 같이 간략한 가정을 한다.

먼저 층의 입력과 출력을 joint하게 정규화하는 것이 아니라 각 feature을 독립적으로 정규화한다. 즉, d차원의 입력 층인 $x = (x^{(1)}, \cdots, x^{(d)})$에 대해서, d개의 각 feature에 대해

$$\widehat{x}^{(k)} = \frac{x^{(k)} - E[x^{(k)}]}{\sqrt{Var(x^{(k)})}}$$

로 정규화를 한다. 여기서 평균과 분산은 k번재 차원에 해당하는 전체 훈련 데이터를 이용한다.

그런데 이렇게 단순히 정규화하는 것은 층이 나타내는 것을 바꿀 수도 있다. 예를 들어서 sigmoid 입력을 정규화하는 것은 비선형 정체성을 해칠 수 있다. 따라서 네트워크의 identity 변환을 나타낼 수 있는 정규화를 고려한다. 각 activation $x^{(k)}$에 대해서 한 쌍의 모수 $\gamma^{(k)}, \beta^{(k)}$을 고려하고, 이는 각각 scale, shift의 역할을 담당한다.

$$y^{(k)} = \gamma^{(k)}\widehat{x}^{(k)} + \beta^{(k)}$$

$\gamma^{(k)}, \beta^{(k)}$는 원래 모수와 함께 업데이트되고 사실, 변환이 필요없는 네트워크라면 $\gamma^{(k)} = \sqrt{Var(x^{(k)})}$, $\beta^{(k)} = E[x^{(k)}]$와 가깝게 되어 원래의 activations으로 회복될 것이다.

mini-batch가 아니라 batch 세팅에서는 activation을 정규화하기 위해서 모든 훈련 데이터 세트가 사용된다. 하지만 SGD에서는 mini-batch가 사용되고 따라서 다음과 같은 두 번째 simplification을 한다: 각 SGD에서 사용되는 mini-batch는 각 activation에 대해 표본 평균과 표본 분산을 만든다. 이렇게 함으로써 정규화에 사용되는 statistics가 gradient backpropagtaion에 완전히 참여한다.

사이즈가 m인 하나의 mini-batch $\mathcal{B}$를 생각해보자. 정규화가 각 activation에 독립적으로 적용되므로 하나의 특정한 activation $x^{(k)}$에 주목하고 k를 생략한다. mini-batch에서 이 activation은 다음 m개의 값을 가진다.

$$\mathcal{B} = \{x_{1,\cdots,m}\}$$

정규화된 값을 $\widehat{x}_{1,\cdots,m}$ 이라고 하고 이들의 선형 변환을 $y_{1,\cdots,m}$ 이라고 하자. 아래의 변환을 $Batch\ Normalizing\ Transform$ 이라고 한다.

$$BN_{\gamma,\beta}:\ x_{1,\cdots,m} \longrightarrow y_{1,\cdots,m}$$

BN 알고리즘은 아래와 같다. ϵ은 numerical stability을 위해서 더해지는 값이다.

Input: Values of x over a mini-batch: $\mathcal{B} = \{x_{1\ldots m}\}$;
Parameters to be learned: γ, β
Output: $\{y_i = \mathrm{BN}_{\gamma,\beta}(x_i)\}$

$$\mu_{\mathcal{B}} \leftarrow \frac{1}{m}\sum_{i=1}^{m} x_i \qquad \text{// mini-batch mean}$$

$$\sigma^2_{\mathcal{B}} \leftarrow \frac{1}{m}\sum_{i=1}^{m}(x_i - \mu_{\mathcal{B}})^2 \qquad \text{// mini-batch variance}$$

$$\widehat{x}_i \leftarrow \frac{x_i - \mu_{\mathcal{B}}}{\sqrt{\sigma^2_{\mathcal{B}} + \epsilon}} \qquad \text{// normalize}$$

$$y_i \leftarrow \gamma\widehat{x}_i + \beta \equiv \mathrm{BN}_{\gamma,\beta}(x_i) \qquad \text{// scale and shift}$$

Algorithm 1: Batch Normalizing Transform, applied to activation x over a mini-batch.

훈련을 하는 동안, loss의 gradient을 backpropagate해야 하고 그 대상은 원래 모수 뿐만 아니라 BN 변환의 모수도 포함된다. chain rule을 이용해 아래와 같이 유도할 수 있다.[1]

[1] https://kevinzakka.github.io/2016/09/14/batch_normalization/