20.05.11

주제 **Batch Normalization**

https://s3-us-west-2.amazonaws.com/secure.notion-static.com/2039da39-040d-42d5-9635-11e6c429bb23/SAI-I_DNN_Batch_Normalization.pptx

Batch Normalization

Idea

딥러닝에서 마주한 문제점

- vanishing gradient
- exploding gradient

Layer 수가 적은 경우는 그 문제가 심각하지 않지만, layer수가 많아지면 많아질수록 누적되어 나타나기 때문에 심각해진다.

그 이유는 활성함수로 sigmoid나 hyper-tangent와 같은 비선형 포화함수를 사용하게 되면, 입력의 절대값이 작은 일부 구간을 제외하면 미분값이 0 근처로 가기 때문에 역전파(back-propagation)을 통한 학습이 어려워지거나 느려지게 된다.

1차 해결책 - ReLU, Dropout, Ensemble

이 문제에 대한 해결책으로 ReLU(Rectifier Linear Unit)을 활성함수로 쓰는 방법이 소개되어 문제가 완화되기는 했지만, 이것은 간접적인 회피이지 본질적인 해결책이 아니라서 망이 깊어지면 여전히 문제가 된다. dropout이나 기타 regularization 방법들 역시 본질적인 해결책이 아니기 때문에 여전히 일정 layer 수를 넘어가게 되면 "training"을 성공시킨다는 것을 보장할 수 없게 된다.

2차 해결책 - Batch Normalization

그러다가 2015년에 획기적인 방법 두개가 발표가 되는데, 그것은 BN(Batch Normalization)과 Residual Network이다. 여기서는 BN에 관해 설명하고자 한다.

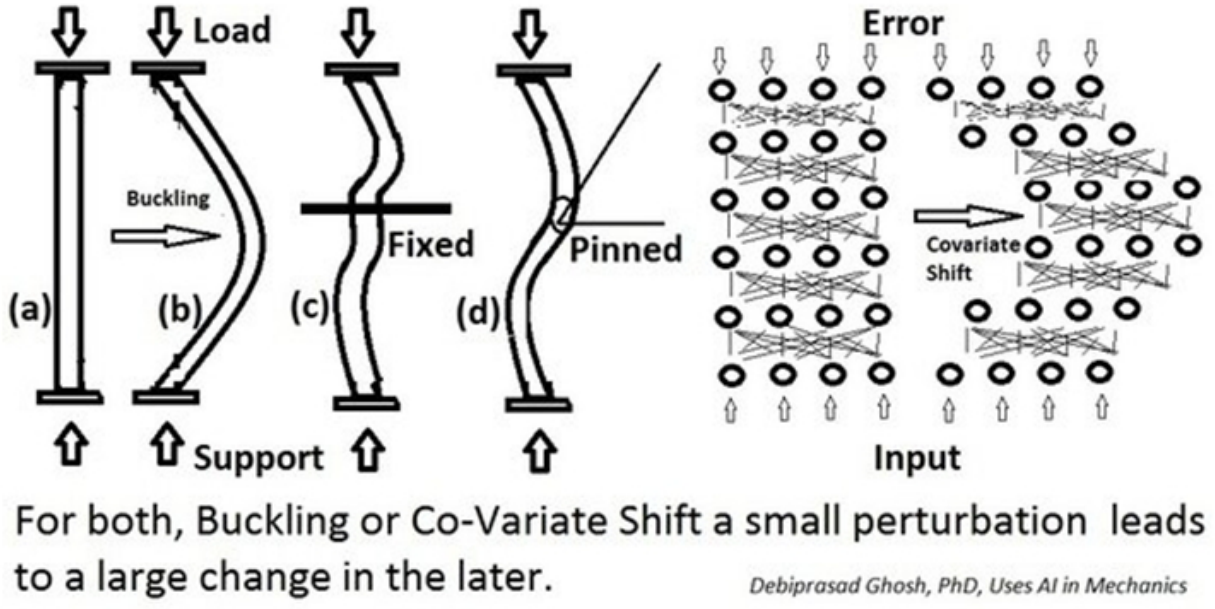
Internal Covariate Shift

최근 딥러닝에는 대부분 GPU가 사용되고 있으며, GPU를 효율적으로 사용할 수 있도록 보통은 32~256크기를 갖는 mini-batch SGD(stochastic gradient descent)방법을 많이 사용한다.

SGD 방식이 효율적이기는 하지만, 효과를 거두려면 hyper-parameter의 설정에 신경을 많이 써줘야 하며, 특히 초기값과 학습 진도율(learning rate)은 매우 중요한 요소가 된다. 학습 시 현재 layer의 입력은 모든 이전 layer의 파라미터의 변화에 영향을 받게 되며, 망이 깊어짐에 따라 이전 layer에서의 작은 파라미터 변화가 증폭되어 뒷단에 큰 영향을 끼치게 될 수도 있다.

이처럼, 학습하는 도중에 이전 layer의 파라미터 변화로 인해 현재 layer의 입력의 분포가 바뀌는 현상을 "Covariate Shift"라고 한다. 이것은 예전 TV 오락 프로에서 귀마개를 하고 상대방의 입모양을 보고 무슨 말인지 알아내는 게임과 비슷하다.

다른 비슷한 비유로, Covariate shift는 건축에서 하중에 의해 기둥이 휘어지는(buckling)과 비슷하다고 불 수 있다.



기둥이 휘어지는 것을 막으려면 위 그림에서 c나 d의 경우처럼 휘어짐을 방지하는 수단이 필요하며, 그 수단으로는 batch normalization이나 whitening 기법을 들 수가 있다.

Covariate Shift를 줄이는 방법

Internal Covariate Shift를 줄이는 대표적인 방법 중 하나는 각 layer로 들어가는 입력을 whitening 시키는 것이다. 여기서 whitening을 시킨다는 의미는 입력을 평균 0, 분산 1로 바꿔준다는 것이다.

하지만 단순하게 whitening만을 시킨다면 whitening 과정과 parameter를 계산하기 위한 최적화 과정(backpropagation)이 무관하게 진행되기 때문에 특정 파라미터가 계속 커지는 상태로 whitening이 진행 될 수 있다. whitening을 통해 loss(cost function)이 변하지 않게 되면, 최적화 과정을 거치면서 특정 변수가 계속 커지는 현상이 발생할 수 있는 것이다.

그러므로 단순하게 whiteninig을 통해 평균과 분산을 조정하는는 것보다는 좀 더 확실한 방법이 필요하며, 그것이 바로 BN(batch normalization)이다.

BN은 평균과 분산을 조정하는 과정이 별도의 process로 있는 것이 아니라, 신경망 앙ㄴ에 포함이 되어 training 시에 평균과 분산을 조정하는 과정 역시 같이 조절된다는 점이 단순 whitening과 구별되는 차이점이다.

Batch Normalization(BN)

Normalization은 원래 training 전체 집합에 대하여 실시를 하는 것이 최고의 효과를 거둘 수 있겠지만, mini-batch SGD 방식을 사용하게 되면, 파라미터의 update가 mini-batch 단위로 일어나기 때문에, mini-batch 단위로 BN을 실시한다. 단 mini-batch 집합의 선정은 가급적으면 correlation이 적어 mini-batch가 전체 집합을 대표하는 것이라고 생각해도 무방하도록 해줘야 한다. (강화학습에서 데이터는 시간의 흐름에 따라 순차적으로 수집되고, 순차적 데이터는 근접한 것들끼리 높은 correlation을 띄게 됨. 만약 이 순차적 데이터를 그대로 입력으로 활용하면 입력 데이터들간의 높은 correlation에 의해 학습이 불안정해질 것임)

학습 시 BN 방법은 아래와 같다.

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

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

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

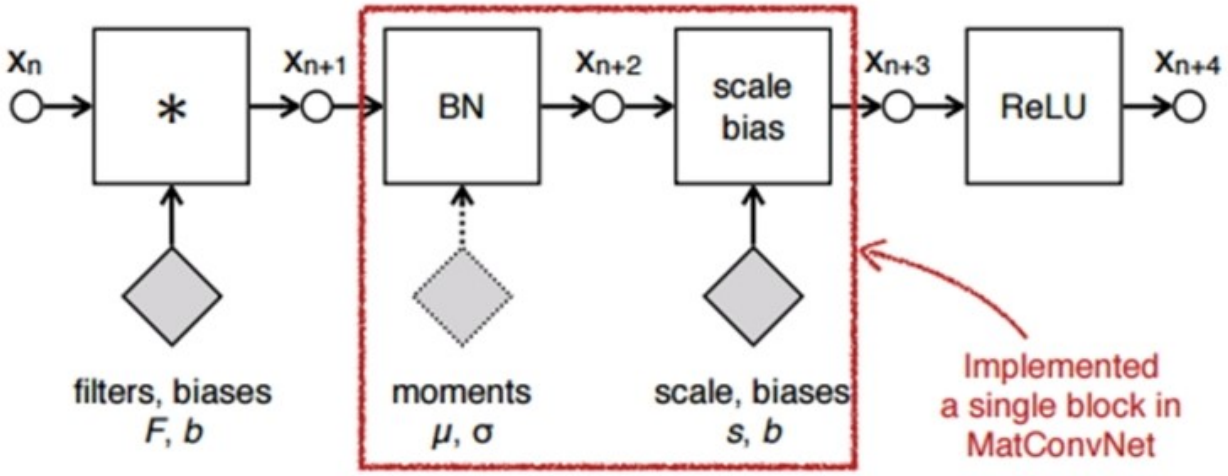
$$\widehat{x}_i \leftarrow \frac{x_i - \mu_{\mathcal{B}}}{\sqrt{\sigma^2_{\mathcal{B}} + \epsilon}} \qquad \text{// normalize}$$

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

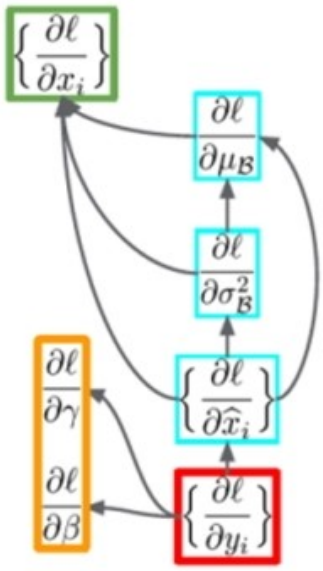
평균과 분산을 구하게 되면 입력을 정규화시킨다. 정규와 과정에서 평균을 빼주고 그것을 분산으로 나눠주게 되면 그 분포는 -1~1의 범위가 된다.

BN이 whitening과 다른 부분은 **평균과 분산을 구한 후 정규화 시키고 다시 scale과 shift연산을 위해 γ감마와 β베타가 추가되었으며, γ감마와 β베타가 추가됨으로써 정규화 시켰던 부분을 원래대로 돌리는 identity mapping도 가능하고, 학습을 통해 γ와 β를 정할 수 있기 때문에 단순하게 정규화만을 할 때보다 훨씬 더 강력해진다.**

BN은 보통 non-linear **활성 함수 앞쪽**에 배치가 되며 아래 그림과 같은 형태가 된다.



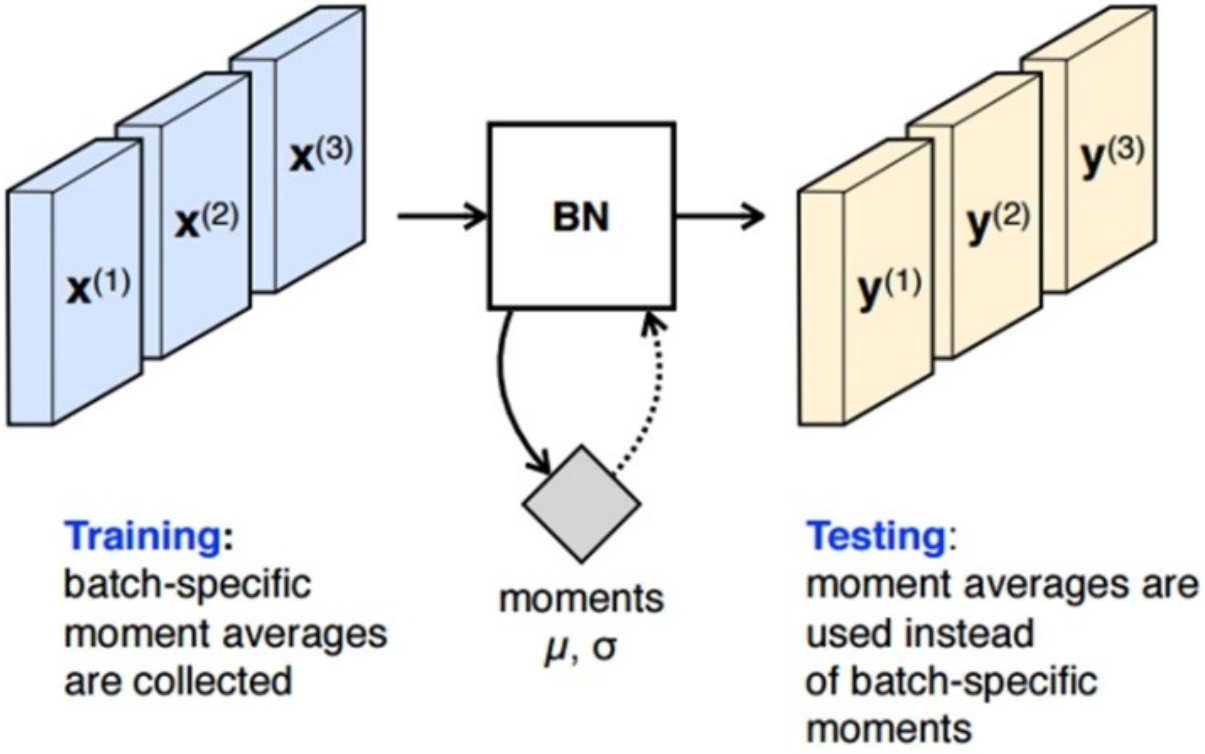
BN은 신경망에 포함이 되기 때문에 back-propagation을 통해 학습이 가능하며, back-propagation 시에는 아래와 같은 chain rule이 적용 된다.



$$\frac{\partial \ell}{\partial \widehat{x}_i} = \frac{\partial \ell}{\partial y_i} \cdot \gamma$$
$$\frac{\partial \ell}{\partial \sigma_{\mathcal{B}}^2} = \sum_{i=1}^{m} \frac{\partial \ell}{\partial \widehat{x}_i} \cdot (x_i - \mu_{\mathcal{B}}) \cdot \frac{-1}{2}(\sigma_{\mathcal{B}}^2 + \epsilon)^{-3/2}$$
$$\frac{\partial \ell}{\partial \mu_{\mathcal{B}}} = \sum_{i=1}^{m} \frac{\partial \ell}{\partial \widehat{x}_i} \cdot \frac{-1}{\sqrt{\sigma_{\mathcal{B}}^2 + \epsilon}}$$
$$\frac{\partial \ell}{\partial x_i} = \frac{\partial \ell}{\partial \widehat{x}_i} \cdot \frac{1}{\sqrt{\sigma_{\mathcal{B}}^2 + \epsilon}} + \frac{\partial \ell}{\partial \sigma_{\mathcal{B}}^2} \cdot \frac{2(x_i - \mu_{\mathcal{B}})}{m} + \frac{\partial \ell}{\partial \mu_{\mathcal{B}}} \cdot \frac{1}{m}$$
$$\frac{\partial \ell}{\partial \gamma} = \sum_{i=1}^{m} \frac{\partial \ell}{\partial y_i} \cdot \widehat{x}_i$$
$$\frac{\partial \ell}{\partial \beta} = \sum_{i=1}^{m} \frac{\partial \ell}{\partial y_i}$$

Training과 Test 시 차이

BN은 학습시와 테스트시에 적용하는 방법이 좀 다르다. 학습 시에 각 mini-batch 마다 γ와 β를 구하고 그 값을 저장해 놓는다. Test시에는 학습 시 mini-batch마다 구했던 γ와 β의 평균을 사용한다는 점이 다르며, 아래 그림처럼 표현이 가능하다.



테스트 시의 유사 코드는 아래와 같다. 유사 코드를 보면 알 수 있듯이 평균은 각 mini-batch에서 구한 평균들의 평균을 사용하고, 분산은 분산의 평균에 m/(m-1)을 곱해주는 점이 다르다. 여기서 m/(m-1)을 곱해주는 이유는 통계학적으로 unbiased variance에는 "Bessel's correction"을 통해 보정을 해주는 것이다. 이는 학습 전체 데이터에 대한 분산이 아니라 mini-batch들의 분산을 통해 전체 분산을 추정할 때 통계학적으로 보정을 위해 베셀의 보정값을 곱해주는 방식으로 추정하기 때문이다. (위키백과 참고)

Input: Network N with trainable parameters Θ;
subset of activations $\{x^{(k)}\}_{k=1}^{K}$

Output: Batch-normalized network for inference, $N_{\text{BN}}^{\text{inf}}$

1: $N_{\text{BN}}^{\text{tr}} \leftarrow N$ // Training BN network
2: **for** $k = 1 \ldots K$ **do**
3: Add transformation $y^{(k)} = \text{BN}_{\gamma^{(k)},\beta^{(k)}}(x^{(k)})$ to $N_{\text{BN}}^{\text{tr}}$ (Alg. 1)
4: Modify each layer in $N_{\text{BN}}^{\text{tr}}$ with input $x^{(k)}$ to take $y^{(k)}$ instead
5: **end for**
6: Train $N_{\text{BN}}^{\text{tr}}$ to optimize the parameters $\Theta \cup \{\gamma^{(k)}, \beta^{(k)}\}_{k=1}^{K}$
7: $N_{\text{BN}}^{\text{inf}} \leftarrow N_{\text{BN}}^{\text{tr}}$ // Inference BN network with frozen // parameters
8: **for** $k = 1 \ldots K$ **do**
9: // For clarity, $x \equiv x^{(k)}, \gamma \equiv \gamma^{(k)}, \mu_{\mathcal{B}} \equiv \mu_{\mathcal{B}}^{(k)}$, etc.
10: Process multiple training mini-batches $\mathcal{B}$, each of size m, and average over them:

$$\text{E}[x] \leftarrow \text{E}_{\mathcal{B}}[\mu_{\mathcal{B}}]$$
$$\text{Var}[x] \leftarrow \frac{m}{m-1}\text{E}_{\mathcal{B}}[\sigma_{\mathcal{B}}^2]$$

11: In $N_{\text{BN}}^{\text{inf}}$, replace the transform $y = \text{BN}_{\gamma,\beta}(x)$ with $y = \frac{\gamma}{\sqrt{\text{Var}[x]+\epsilon}} \cdot x + \left(\beta - \frac{\gamma\,\text{E}[x]}{\sqrt{\text{Var}[x]+\epsilon}}\right)$
12: **end for**

Algorithm 2: Training a Batch-Normalized Network

종합하면, Batch Normalization은 단순히 평균과 분산을 구하는 것이아니라, scale(γ)과 shift(β)를 통한 변환을 통해 훨씬 유용하게 되었으며, BN이 신경망의 layer중간에 위치하게 되어 학습을 통해 γ와 β를 구할 수 있게 되었다. Covariate shift 문제로 인해 망이 깊어질 경우 학습에 많은 어려움이 있었지만, BN을 통해 Covariate shift 문제를 줄여줌으로써 학습의 결과도 좋아지고 빨라지게 되었다.

Benefit

논문에서 주장하는 Batch Normalization의 장점은 다음과 같다.

1. 기존 Deep Network에서는 learning rate를 너무 높게 잡을 경우 gradient가 explode/vanish 하거나, 나쁜 local minima에 빠지는 문제가 있었다. 이는 parameter들의 scale 때문인데, Batch Normalization을 사용할 경우 propagation 할 때 parameter의 scale에 영향을 받지 않게 된다. 따라서, learning rate를 크게 잡을 수 있게 되고 이는 빠른 학습을 가능케 한다.
2. Batch Normalization의 경우 자체적인 regularization 효과가 있다. 이는 기존에 사용하던 weight regularization term 등을 제외할 수 있게 하며, 나아가 Dropout을 제외할 수 있게 한다 (Dropout의 효과와 Batch Normalization의 효과가 같기 때문.) . Dropout의 경우 효과는 좋지만 학습 속도가 다소 느려진다는 단점이 있는데, 이를 제거함으로서 학습 속도도 향상된다.

Reference

- Benefit

  https://shuuki4.wordpress.com/2016/01/13/batch-normalization-%EC%84%A4%EB%AA%85-%EB%B0%8F-%EA%B5%AC%ED%98%84/

  https://yjucho1.github.io/deep learning paper/batchnorm/

- Algorithm

  https://nittaku.tistory.com/293

- Ect

https://subinium.github.io/introduction-to-normalization/