Batch Normalization: Accelerating Deep Network Training by Reducing Internal Covariate Shift

Abstract

신경망을 학습시키면 특정 Input Layer의 Distribution이 계속 변합니다. 학습 과정에서 특정 Layer의 이전 layer들의 가진 parameter(Weight, Bias)가 바뀌기 때문입니다. 이 현상 때문에 Lower learning rate와 careful parameter initialization이 필요하다고 합니다. 이로 인해 전체 training process의 속도가 떨어지고 아울러 saturating non-linearity를 유발해 학습이 어려워집니다.

우리는 이러한 현상을 ***internal covariance shift***라고 부르기로 했습니다. 그리고 이러한 layer단의 input을 normalize하는 방법을 통해 이 문제를 해결하려고 했습니다. 특히 이 논문이 제시하는 방법은 mini batch 단위의 input normalization을 Network 구조의 일부로 포함할 수 있다는 장점이 있습니다.

Batch-normalization을 쓰면 아래와 같은 장점이 있습니다.

1. 높은 Learning rate를 사용할 수 있습니다.
2. Weight 초기화를 less careful하게 해도 됩니다.
3. 그리고 Batch-noramlizer는 몇가지 case에 대해서 Dropout을 적용하지 않아도 됩니다.

▼ Abstract 요약

특정 parameter 들 때문에 생기는 saturating non-linearity 문제를 해결하기 위해 BN이 구상됨

BN의 장점은

1. 높은 Learning rate를 사용할 수 있습니다.
2. Weight 초기화를 less careful하게 해도 됩니다.
3. 그리고 Batch-noramlizer는 몇가지 case에 대해서 Dropout을 적용하지 않아도 됩니다.

가 있음. 그래서 Dropout을 안해도 되기에 Accelerating하다고 말하는 것 같습니다.

Introduction

Deep learing은 Vision, Speech 와 다른 많은 영역에서 dramatically한 성능 향샹을 얻었습니다. SGD와 SGD + Momentum, **AdaGrad**은 deep learning network를 학습시키는데 효율적인 방법이라는 것은 이미 증명되었습니다. SGD optimizer는 전체 loss를 줄이는 방향으로 parameters를 찾아가는 최적화 알고리즘입니다.

$$\Theta = \arg\min_{\Theta} \frac{1}{N} \sum_{i=1}^{N} \ell(\mathrm{x}_i, \Theta)$$

▼ 수식 설명

SGD Optimizer는 전체 loss를 줄이는 방향으로 “Parameters”를 찾아가는 최적화 알고리즘입니다.

여기서 Optimizer는 Gradient Descent를 하는 방법론이라고 생각하면 편하다. 일반적인 Gradient Descent는 전체 Train set에서 loss를 계산해 전체 Model의 gradient를 갱신하는 방법인데, 가장 직관적이다. SGD는 mini-batch단위로 위 과정을 진행해 조금 점은 많이 찍히지만 빠르게 optimal한 공간을 찾아간다.

SGD는 Mini-batch 단위로 Loss를 측정하고 backprob를 진행하기 때문에 training set의 Minibatch로 전체 Model parameter들의 loss function에 대한 gradient를 근사할 수 있습니다.

Mini-Batch를 사용하면 아래와 같은 좋은 점들이 있습니다.

First. mini-batch에 대한 loss의 gradientsms 전체 training set에 대한 gradient를 근사할 수 있습니다.

Second. 1개의 Batch 단위로 연산은 현대 컴퓨팅 platform이 제공하는 parrellism 때문에 각각의 예에 대한 m번의 계산보다 더 효율적일 수 있습니다.

SGD는 network를 학습시키는데 쉽고 효율적이다. SGD는 learning rate, Model weight에 대한 초깃값들을 잘 설정해야지 학습이 잘된다는 단점도 있습니다.

각 layer의 input은 이전 layer의 model parameter에 영향을 받으므로 학습을 더 복잡해지는게 사실입니다. 특히나 network가 deep한 경우 network parameter가 조금만 바뀌어도 다음 layer의 영향에 증폭되어 나타나게 됩니다.

layer의 input distribution의 변화는 계속해서 새로운 distribution에 적응해야하는 문제를 야기합니다. learning system의 input distribution가 변하면 그것은 covariate shift를 경험했다고 말한다. 이는 이전에 domain adaptation이라는 방법으로 해결하는 시도가 있었습니다. 이 시도는 전체 training system 관점에서 문제를 해결하려고 했습니다. 하지만 우리는 이 개념을 network layer 관점에서 적용하려고 합니다.

아래와 같은

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

Loss l을 Minimize 하는데 2개의 임의의 transformation(F_1, F_2)이 적용되고 Parameter θ_1, θ_2가 존재하는 network를 생각해보세요. 2번째 θ_2는 $x = F_1(u, \theta_1)$의 output과 함께 2번째 함수의 input으로 주어집니다.

$$\ell = F_2(\mathrm{x}, \Theta_2). \qquad \Theta_2 \leftarrow \Theta_2 - \frac{\alpha}{m}\sum_{i=1}^{m}\frac{\partial F_2(\mathrm{x}_i, \Theta_2)}{\partial \Theta_2}$$

예를들어, gradient descent step(Batch size = m, learing rate = a)인 경우에 F2를 학습하는 것은 첫번째 Transformation의 output을 X로 놓고 param을 updatae하는 과정으로 볼 수 있다. (첫 input은 u F1)을 통과한 이후에는 x, 즉 $F_1(u, \theta_1) = x$)

고로 우리는 이 과정을 통해 network를 이루는 sub network input distribution이 일정하게 유지된다면 covariate shift 와 같은 문제에 대해 고려하지 않아도 되므로 sub network의

parameter들이 학습되는 과정이 개선될 것이라고 생각했습니다.(따라서 domain adaptation을 sub network인 layer 관점에서도 적용할 수 있다는 뜻이라고 합니다)

sigmoid activation function과 $z = g(Wu + b)$ layer를 고려해봐라. 이때 u는 layer의 input, W는 Weight, b 는 bias Vector이다. 또한 $g = \frac{1}{1+exp(-x)}$인 activation function이다. 이때 x가 증가하면 $g^{'}(x)$는 0에 수렴한다. 이것은 W,b값이 변경되면서 sigmoid 미분값이 0으로 수렴해 없어지거나(vanish) 학습이 느려지는 문제가 발생할 수도 있다. 하지만 만약에 input을 장제로 이런 saturation non-linarity가 발생하지 않는 영역으로 Regulartization할 수 있다면 위의 gradient가 0으로 수렴하는 현상을 발생하지 않아 network의 깊이를 깊게할 수 있고, 학습 또한 accerlerate 할 수 있다.(반대로 gradient의 성질이 non-linearty 하지 않아서 생기는 문제도 있음. 그래서 비선형을 잃게 되어 이러한 non-linearty 성질을 보존하기 위해 Scale 및 Shift 연산을 수행하게 됨)

이러한 input의 distribution의 변화를 이 논문에서는 **Internal covariance shift**라고 부른다. 그리고 이 internal covariate shift를 제거하는 batch normalization algorithm을 제안한다. 이 batch normalization은 normalization step을 통해 layer의 input의 mean과 variance를 수정하는 것을 이룬다(accomplish).

Batch Normalization을 하면 Network가 가지고 있는 매개변수의 scale 혹은 매개변수의 초기값에 대한 gradient의 dependence를 줄임으로써 network를 통과하는 gradient flow에 beneficial(유익한) 영향을 미친다. 이것은 우리를 하여금 더 높은 learning rate를 divergence(발산)에 대한 위험없이 사용할 수 있도록 해준다. 또한 batch normalization을 이용한 모델 정규화는 Dropout을 사용할 이유를 없앤다.

마지막으로 Batch Noramlization은 saturate 상태에 빠저 stuck 되지 않도록 prevent하여 saturating nonlinearity(tanh, Sigmoid activation function)를 사용할 수 있게 만들어준다.

4.2절에서 Batch Normalization을 가장 성능이 좋은 ImageNet Classification 모델에 적용해 오직 7%의 학습 step만을 사용했음에도 불구하고 정확도를 기존보다 더 높일 수 있었다. 더 나아가 Batch Noramlization을 사용한 Model들간의 ensemble은 top-5 error rate에 대해 ImageNet Classification Task에서 최고를 달성한다.

▼ Introduction 요약

일반적으로 network를 학습을 시키면 backpropagation에 의해 weight들이 갱신된다. 이때 input layer을 제외한 hidden layer들은 각각의 layer들의 previous layer들의 output을 input으로 받게 되는데, previous layer 또한 backpropagation의 영향으로 weight를 update하게 됩니다. 그래서 previous layer들의 output의 distribution에 영향을 받게되고 현재의 layer들은 다시 변화된 previous output에 적응을 해야하는 문제가 발생하게 됩니다. 이때 현재 layer로 들어오는 Input의 distribution의 변화를 **Internal covariance shift**라고 합니다.

본 논문은 BN을 통해 이전 layer의 output을 normalize하는 과정을 통해 output distribution을 일정한 간격 사이로 보내려고 한다. 이를 통해 sigmoid나 tanh와 같은 saturate non-linearity를 유발하는 activation function을 사용하는 것도 충분히 고려될 수 있다. 또한 Dropuout에 대한 사용이유도 줄어들게 된다.

이때 BN은 layer의 input을 mean과 variance 값을 이용해 수정하는 것을 의미한다.

Toward reducing Internal Covariate Shift

우리는 ***Internal Covariate Shift***를 **"학습 중에 네트워크의 parameter의 변화로 인한 네트워크 activation의 분포 변화"**라고 정의합니다. training을 향상시키기 위해 우리는 internal covariate shift를 줄이려고 한다. 학습 progreese에서 x라고 하는 input layer의 distribution을 고침으로써 우리는 학습 속도를 증가시키는 것을 기대합니다. 입력이 whitened될 수록 네트워크를 학습시키는게 빨리 수렴한다는 사실은 (LeCun et al., 1998b; Wiesler & Ney, 2011)에 의해 오래전부터 알려졌습니다. 즉 평균은 0 그리고 unit variance(분산의 값이 1)를 가지도록 linear transform하는 것입니다. 각 layer들은 이전 layer에 의해 생성된 입력을 observe하기 때문에, 같은 whitening을 각각의 layer의 input으로 하게끔하면 더 좋을 것이다. 각 층에 대한 input을 whitening함으로써 우리는 internal covariate shift의 단점을 제거할 방향으로 input distribution을 고치는 방향으로 step을 밟을 것이다.
Network를 직접 수정하거나 optimization algorithm의 매개 변수를 netowrk의 activation value값에 depend on하도록 modifying하여 모든 training step 또는 some interval마다 whitening activation을 하는 것을 고려할 수 있다. 그러나 이러한 수정 사항이 최적화 단계 사이사이에 존재하는 경우 gradient descent step은 **이상한 방식**으로 진행될 수 있다.

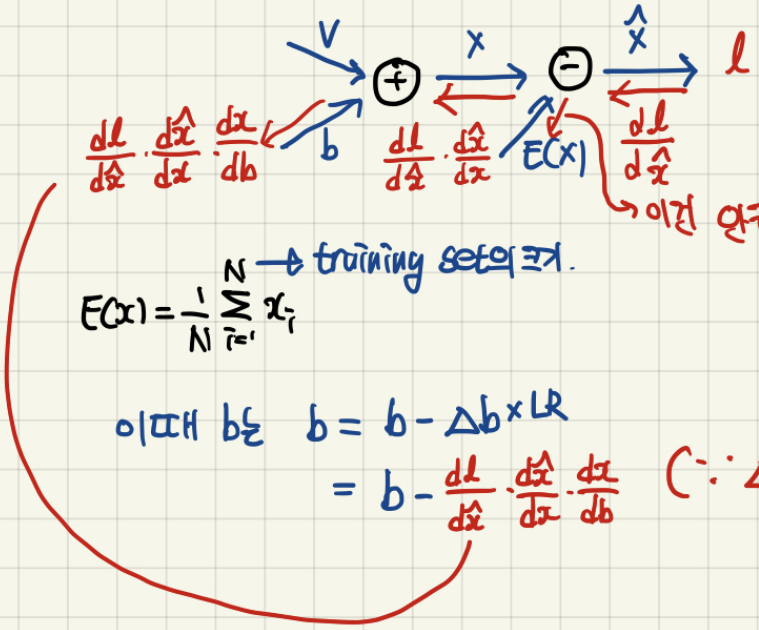
▼ 이상한 방식

저는 이런식으로 이해 했습니다.

본래 입력 u에 b를 더해 x를 출력하는 layer 가 존재한다고 가정.

이때 이 출력 x에 E(x)를 빼서 normalized 된 $\hat{x} = x - E(x)$

<원래 layer>



$E(x) = \frac{1}{N}\sum_{i=1}^{N} x_i$ → training set의 크기.

이때 b는 $b = b - \Delta b \times LR$

$= b - \frac{dl}{d\hat{x}} \cdot \frac{d\hat{x}}{dx} \cdot \frac{dx}{db}$ ($\because \Delta b$는 $\frac{dl}{d\hat{x}}$에 비례함)

그러면 이제 새로 update된 network에서 $\hat{x}$은 이렇게 표현됨.

$x = u + b + \Delta b$, $\hat{x} = u + b + \Delta b - E(u + b + \Delta b)$

문제는 여기 Δb가 상수로 $E(\Delta b) = \Delta b$이므로 $\hat{x} = u + b - E(u + b)$도 그대로이다.

따라서 loss도 그대로이다. 그러므로 learning step이 거듭될수록

b에는 비슷한 크기의 Δb가 더해져 값이 발산하게 된다.

($\Delta b = \frac{dl}{d\hat{x}} \cdot \frac{d\hat{x}}{dx} \cdot \frac{dx}{db} \times LR$)에서 $\frac{d\hat{x}}{dx} = \frac{\partial x}{\partial b} = 1$이다.

덧셈이라 1.

$\therefore \Delta b = \frac{dl}{d\hat{x}}$

근데 $\hat{x}$가 같으므로 Δb도 매 training step 마다 같다.

그러므로 b에 대한 Update와 그에따른 normalization 변화의 조합은 output layer에서의 어떠한 변화도 초래하지 않았다. 훈련이 계속될 수록 b는 손실이 고정된 상태로 계속 진행되고, 이 문제는 normalization이 중심(mean값)만이 문제가 아니라 activation scale까지 확장하면 더 악화될 수 있다. normalization parameters가 gradient descent step 밖에서 계산될 때 model이 폭파(blow up)되는 것을 경험적으로 관찰했습니다.

이러한 issue의 문제는 gradient descent optimization는 normalization(whitening)에 대한 것을 고려하지 않기 때문이다. 이 문제를 해결하기 위해 **우리는 모든 parameters에 대해 network가 항상 원하는 distribution으로 activation을 produce(제공)할 수 있도록 하고자 한다**. 그렇게 함으로써 loss의 network parameter에 대한 gradient가 normalization을 고려하도록 할 수 있다.

어떤 layer의 input을 x라고 해보자 (x는 vector)

X는 전체 training dataset에 해당하는 X의 집합이면서 parameter로써 작용할 것이다.

즉 normalization에 해당하는 $Norm$함수는 X를 이용하면서 input output에 관려하는 parameter function이 되므로 $\hat{x} = Norm(x, X)$라고 할 수 있다. 이후 이것을 dataset(X)와 input(x)에 대한 미분으로 접근할 수 있고(왜냐하면 각각에 종속적임 함수$Norm$이기 때문) 이는 Jacobian으로 계산될 수 있습니다. (Jacobian으로 계산하는 이유는 x와 X 모두 multi dimension variable이기 때문) 이를 이용해 backprob을 진행할 수 있습니다. 이때 각각 x와 X로 흐르는 gradient flow에서 $\frac{\partial \hat{x}}{\partial x}$를 무시하게 되면 gradient가 explosion하게 됩니다. framework(이러한 구조)에서 매번 normalization layer의 인풋은 whitening 하는 것은 많은 연산을 동반하게 됩니다. 왜냐하면 parameter가 변경될 때마다 X가 변경되어 다시 whitening하는데 필요한 cov[X] (dataset에 대한 공분산)을 매번 계산해야하기 때문입니다. 때문에 우리는 input을 whitening을 할 수 있는 대안을 찾아야 합니다. (즉, 비슷한 기능을 하지만 다르게 연산되는 layer 연산을 찾아야 함)

이 논문에서는 전체 training set을 모두 whitening 하는 것은 연산량이 많기도 하고 모든 곳에서 differentiable하지 않기 때문에 우리는 mini-batch를 통해 사용하고자 합니다.

Normalization via Mini-Batch Statistics

각각의 layer에 대한 input들을 whitening하는 것은 costly하고 어디에서나 미분가능하지 않다. 우리는 2가지의 simplification하는 것을 필요로한다. 첫번째는 layer의 input과 output을 jointly(연결되게, 관련있게)하게 whitening하지 않는다. D차원의 입력($x = (x^1, ..., x^d)$) 우리는 training dataset에 대해 기대값(평균)과 분산을 각 차원에 대해 normalize할 것이다.

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

그런데 이러한 Normalize는 본래 네트워크가 representation(표현) 하는 방식을 변경한다. 예를 들어 sigmoid의 input을 normalize했다면, 그 값이 sigmoid의 linearity가 만족하는 구간으로 무조건적으로 제한되게 된다. 하지만 전체 네트워크의 성능을 최적화 하는게 이런 강제적인 transform은 필요하지 않을 수도 있다. 따라서 추가로 몇개의 parameter를 사용해 필요하다면 normalization 과정이 그냥 들어온 것을 내보내는(linear) 기능을 할 수도 있게 만들어야 한다. 이 기능을 위해 우리는 아래와 같은 방법을 사용합니다.

우리는 2개의 새로운 파라미터$\gamma^{(k)}$, $\beta^{(k)}$를 가지고 와서 normalized된 값 $x^{(k)}$를 아래와 같이 rescale하려고 합니다.

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

이 파라미터값들은 우너래 네트워크의 파라미터와 함께 train됩니다. 그리고 필요에 따라 기존(원본) network의 출력을 복원(restore)하는 용도로 사용될 수 잇습니다. 즉 원래 출력을 지키는 것이 loss를 줄이고 network를 최대 성능을 내도록 하는 방향이라면 각 파라미터를 아래와 같이 설정함으로써 input을 원래 input 형태로 되돌려 놓을 수 있다. (수식 전개하면 쉽게 나옵니다)

$$\gamma^{(k)} = \sqrt{Var[x^{(k)}]}, \beta = E[x^{(k)}]$$

위와 같은 batch setting은 각 training set을 하나의 traing step으로 가정할 때 normalize activation setting입니다. 그러나 이것은 stocahstic optimization을 사용할 때는 그다지 실용적이지 않습니다.

그래서 우리는 2번째 simplification (가설)을 세울 것입니다.

우리가 stochastic gradient training을 할 때 mini-batch를 사용하기 때문에, 각 mini-batch activation의 mean과 variance을 estimate해 사용하는 것입니다. 이렇게 함으로써 backpropagation 과정에서도 normalization을 완전히 고려할 수 있습니다.

크기가 m인 mini-batch B를 가정해봅시다. normalization이 each activation indepently 하므로 특정 activation $x^{(k)}$에 집중해 봅시다.(이후 k를 빼고 작성할 예정)

▼ 혹시나 스스로 헷갈릴까봐

여기서 x^k에서 k는 2차원 평면 상의 mini-batch data를 상상했을 때 column으로 k번째 column을 의미하는 듯 하다. 그래서 k-dimension이라는 워딩이 나오는 듯 하다.

이후 normalize를 시켜서 $\{\hat{x_1}, ..., \hat{x_m}\}$을 뽑고 이후 linear transformation을 거쳐 $y_1, ..., y_m$을 만들어냈습니다. 우리는 이러한 transform을 Batch Normalization Transform이라는 refer합니다...(?ㅋㅋ) 우리는 Algorithm 1에서 BN transform을 설명합니다.

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

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

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

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

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

BN transform은 activation을 조작하기 위해 network에 추가될 수 있습니다. $y = BN_{\gamma,\beta}(x)$라는 표기는 감마와 베타가 학습 가능해 변경이 가능한 paremter라는 것을 보여주기 위한 표기이다. (왜 BN에 linear transform인 gamma와 beta가 있는건지 아직 이해 x)

Batch-noramlization transformation은 개별 training smaple에 대해서 독립적으로 시행되는 건 아니다. 샘플뿐만 아니라 mini-batch 내의 모든 training smaple을 이용하게 된다. 감

마와 베타를 이용해 scaled, shifting 된 값 y는 다른 layer의 input으로 제공될 수 있다.

▼ 나름 대로의 설명(?)

위 $y = BN_{\gamma,\beta}(x)$의 표기 때문에 BN tranform이 개별 sample만 가지고 수행하는 transform이라고 생각할 수 있는데, 수식처럼 모든 sample(Data)를 고려해 Mean과 Variance를 구해서 y값으로 변환하게 됩니다. γ, β는 학습 가능한 parameter입니다.

위 식에 $\hat{x}$가 존재하지 않지만, $\hat{x}$는 전체 batch를 고려한 normalization이라는 중요한 의미를 가집니다.

mean = 0, variance = 1인 특성을 가지는 $x^{(k)}$는 $y^{(k)} = \gamma^{(k)}\hat{x^{(k)}} + \beta b^{(k)}$를 수행하는 sub-network의 input으로 간주될 수 있습니다.

그리고 이 subnetwork의 output은 원래 network의 flow대로 다음 layer로 전달되는 것입니다.

$\hat{x^{(k)}}$끼리 joint distribution이 training 과정에서 변경되더라도 개별 $\hat{x^{(k)}}$의 mean과 variance가 유지 되므로 $y^{(k)} = \gamma^{(k)}\hat{x^{(k)}} + \beta b^{(k)}$를 수행하는 sub-network의 효율을 개선하고 전체 network를 학습을 개선할 것임을 기대할 수 있다.

학습을 위해 gradient of loss인 l은 아래와 같은 transformation을 통과해야합니다.

이를 위해 아래와 같은 chain rule을 사용해 BN transformation을 통과하는 gradient를 계산합니다.

$$\frac{\partial \ell}{\partial \widehat{x}_i} = \frac{\partial \ell}{\partial y_i} \cdot \gamma$$

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

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

$$\frac{\partial \ell}{\partial x_i} = \frac{\partial \ell}{\partial \widehat{x}_i} \cdot \frac{1}{\sqrt{\sigma_{\mathcal{B}}^2 + \epsilon}} + \frac{\partial \ell}{\partial \sigma_{\mathcal{B}}^2} \cdot \frac{2(x_i - \mu_{\mathcal{B}})}{m} + \frac{\partial \ell}{\partial \mu_{\mathcal{B}}} \cdot \frac{1}{m}$$

$$\frac{\partial \ell}{\partial \gamma} = \sum_{i=1}^{m} \frac{\partial \ell}{\partial y_i} \cdot \widehat{x}_i$$

$$\frac{\partial \ell}{\partial \beta} = \sum_{i=1}^{m} \frac{\partial \ell}{\partial y_i}$$

▼ **수식 해석(추가 예정)**

Batch-normalization transform은 네트워크에 정규화된 activation(이전 레이어의 activation) 제공하는 미분 가능한 레이어입니다. (따라서 트레이닝 과정에서 gradient flow 를 넘겨 backpropagation으로 학습을 할 수 있습니다) 그리고 이 batch noramliztion layer 때문에 우리는 model이 항상 일정한 distribution을 가지는 input 값을 가지고 학습을 할 수 있습니다. 이는 전체 학습 과정을 accelerating하고 성능을 좋게 합니다.

마지막으로 Batch-Normalization의 두가지γ와 β parameter는 필요에 따라 배치의 분포를 원래의 것과 동일하게 유지할 수 있는 능력을 BN에 제공합니다.

그리고 이 $\hat{x^{(t)}}$에 적용된 affine transform인 $y^{(k)} = \gamma^{(k)}\hat{x^{(k)}} + \beta^{(k)}$이 전체 network를 optimize하는 방향으로 단순 whitening 된 $\hat{x^{(k)}}$를 transform(backprob)합니다. 이때 사용되는 γ와 β는 학습 과정에서 자동으로 학습힙니다.

따라서 위에서 봤던 것처럼 normalization 과정이 동적으로 학습됨에 따라 이전에 설명했던 것과 같이 gradient descent 과정(같은 gradient가 매번 역방향으로 flow 되어 parameter 들이 발산하는)이 오작동 하는 경우를 방지할 수 있습니다.

Training and Inference with Batch Normalized Networks

Summary

Reference

- Paper : https://arxiv.org/pdf/1502.03167.pdf