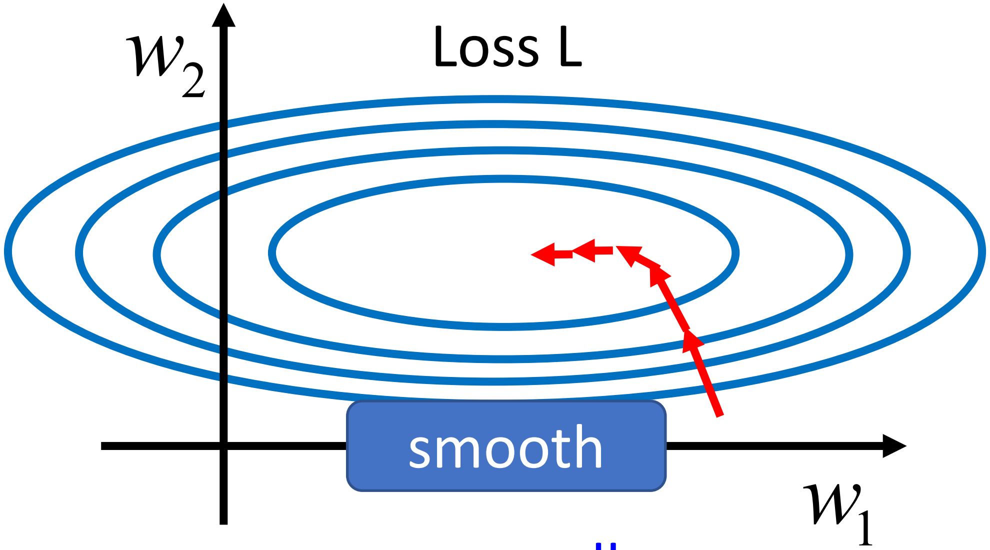
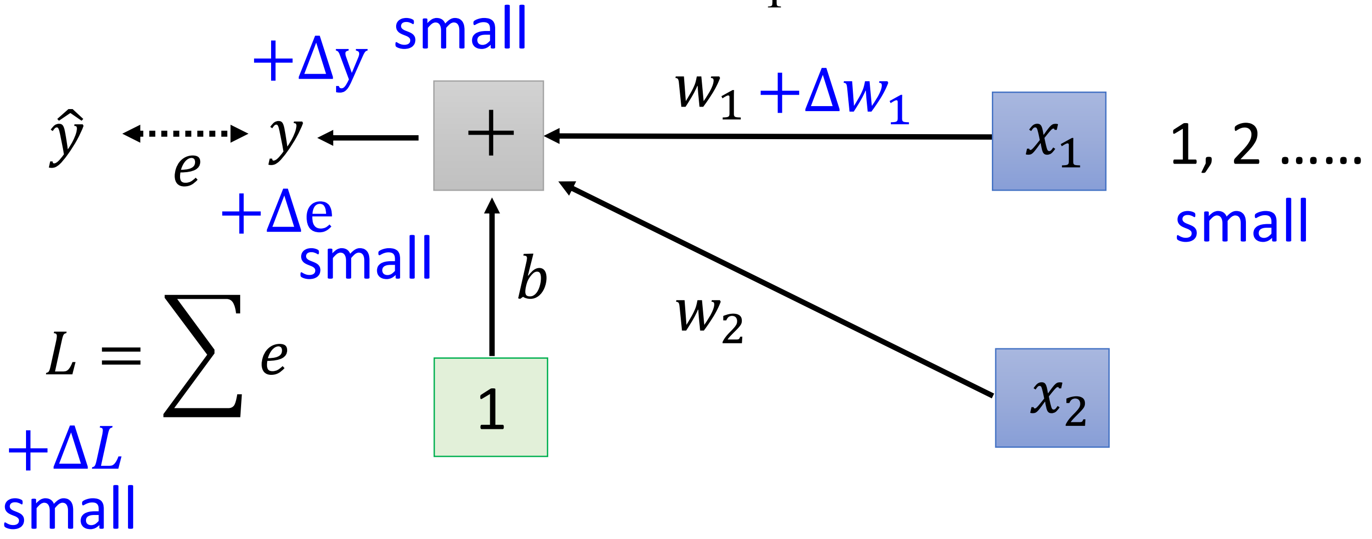
Quick Introduction of Batch Normalization

Hung-yi Lee 李宏毅

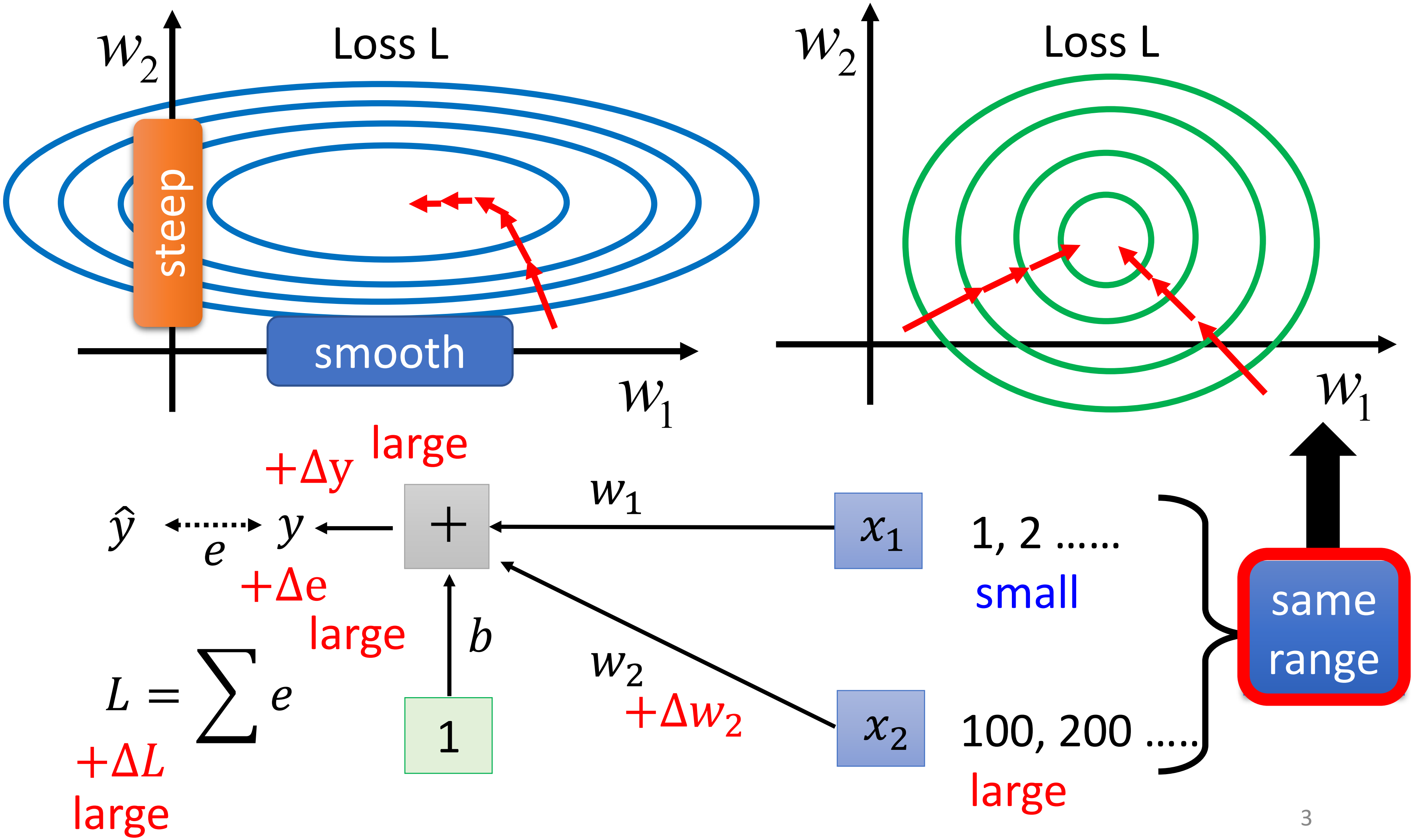
Changing Landscape


w_2
Loss L
smooth
w_1

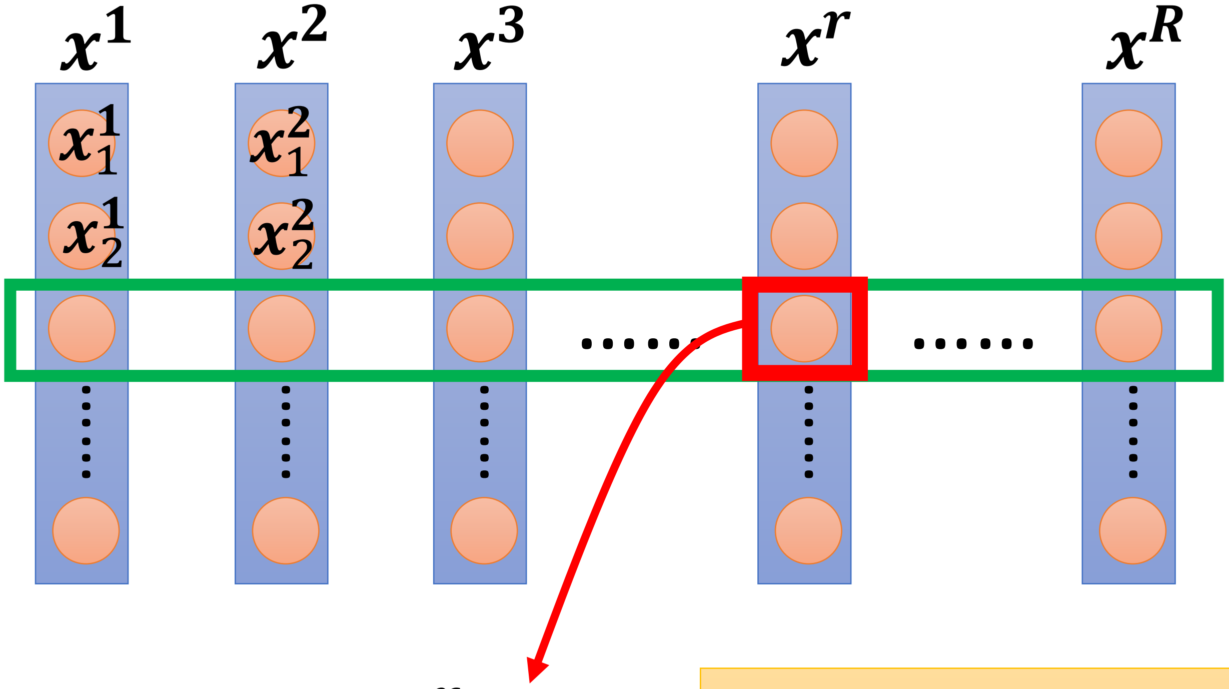

small
$+\Delta y$
$\hat{y}$
e
y
$+$
w_1 $+\Delta w_1$
x_1
1, 2 ……
small
$+\Delta e$
small
b
w_2
1
x_2
$+\Delta L$
small

$$L = \sum e$$

Changing Landscape


w_2
Loss L
steep
smooth
w_1
w_2
Loss L
w_1
large
$+\Delta y$
w_1
$\hat{y}$
e
y
$+$
x_1
1, 2 ……
small
same range
$+\Delta e$
large
b
w_2
$+\Delta w_2$
$L = \sum e$
1
x_2
100, 200 ……
$+\Delta L$
large
large

Feature Normalization


$\boldsymbol{x}^1$
$\boldsymbol{x}^2$
$\boldsymbol{x}^3$
$\boldsymbol{x}^r$
$\boldsymbol{x}^R$
$\boldsymbol{x}_1^1$
$\boldsymbol{x}_2^1$
$\boldsymbol{x}_1^2$
$\boldsymbol{x}_2^2$
……
……

For each dimension i:

mean: m_i

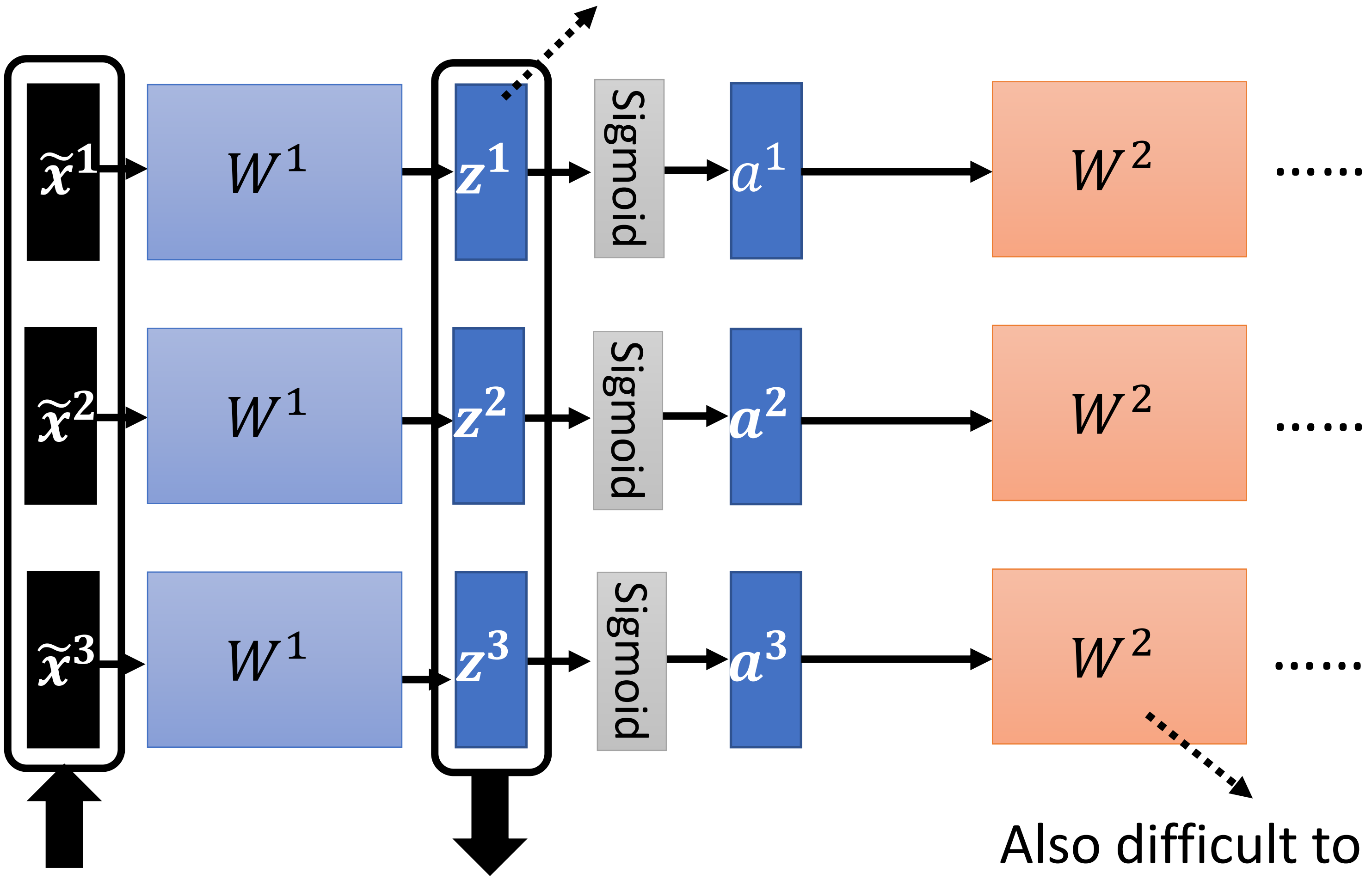
standard deviation: σ_i

$$\tilde{\boldsymbol{x}}_i^r \leftarrow \frac{\boldsymbol{x}_i^r - m_i}{\sigma_i}$$

The means of all dims are 0, and the variances are all 1

In general, feature normalization makes gradient descent converge faster.

Considering Deep Learning


$\tilde{x}^1$
$\tilde{x}^2$
$\tilde{x}^3$
W^1
W^1
W^1
z^1
z^2
z^3
Sigmoid
Sigmoid
Sigmoid
a^1
a^2
a^3
W^2
W^2
W^2
……
……
……

Different dims have different ranges.

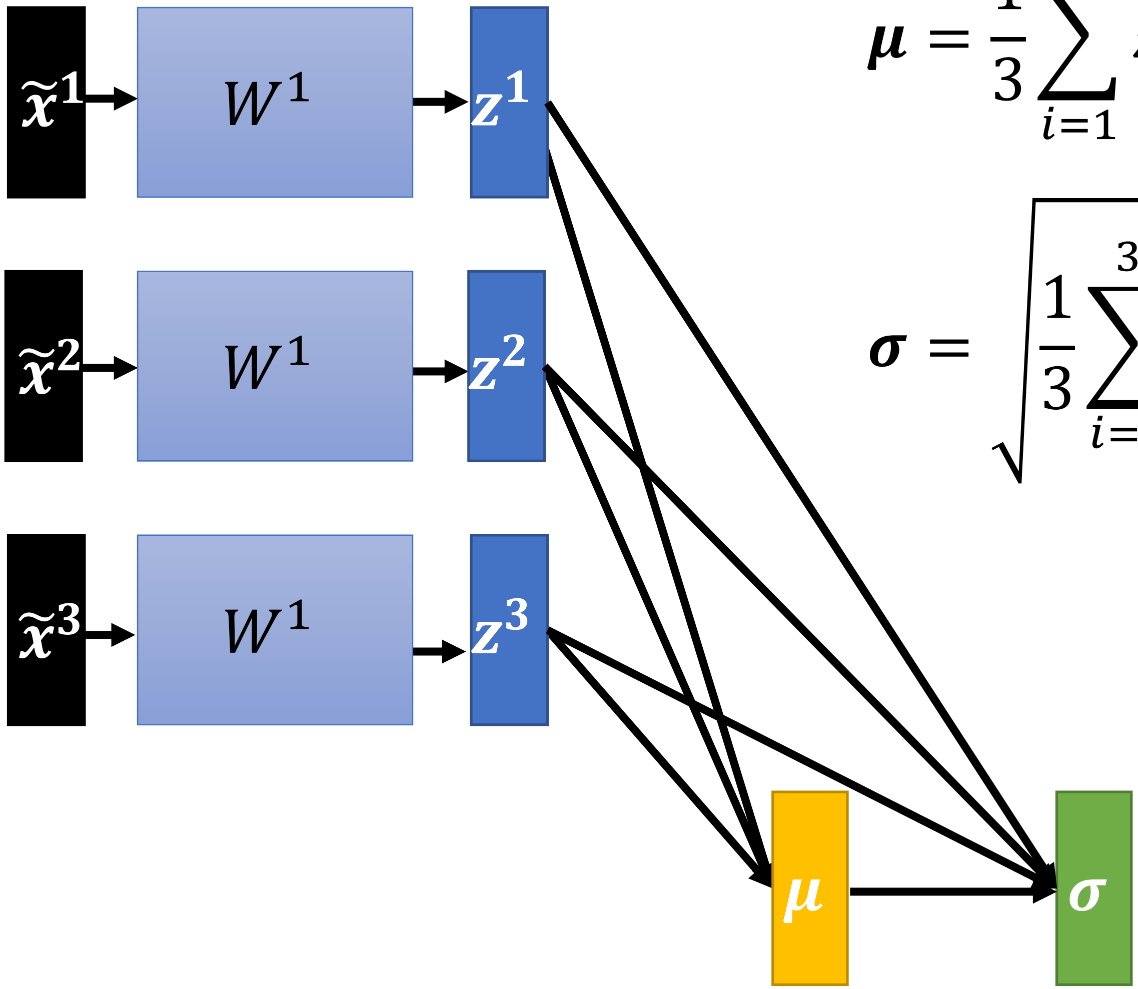
Feature Normalization

Also need normalization

Also difficult to optimize

放在 activation function 没太大区别

Considering Deep Learning


$\tilde{x}^1$
W^1
z^1
$\tilde{x}^2$
W^1
z^2
$\tilde{x}^3$
W^1
z^3
μ
σ

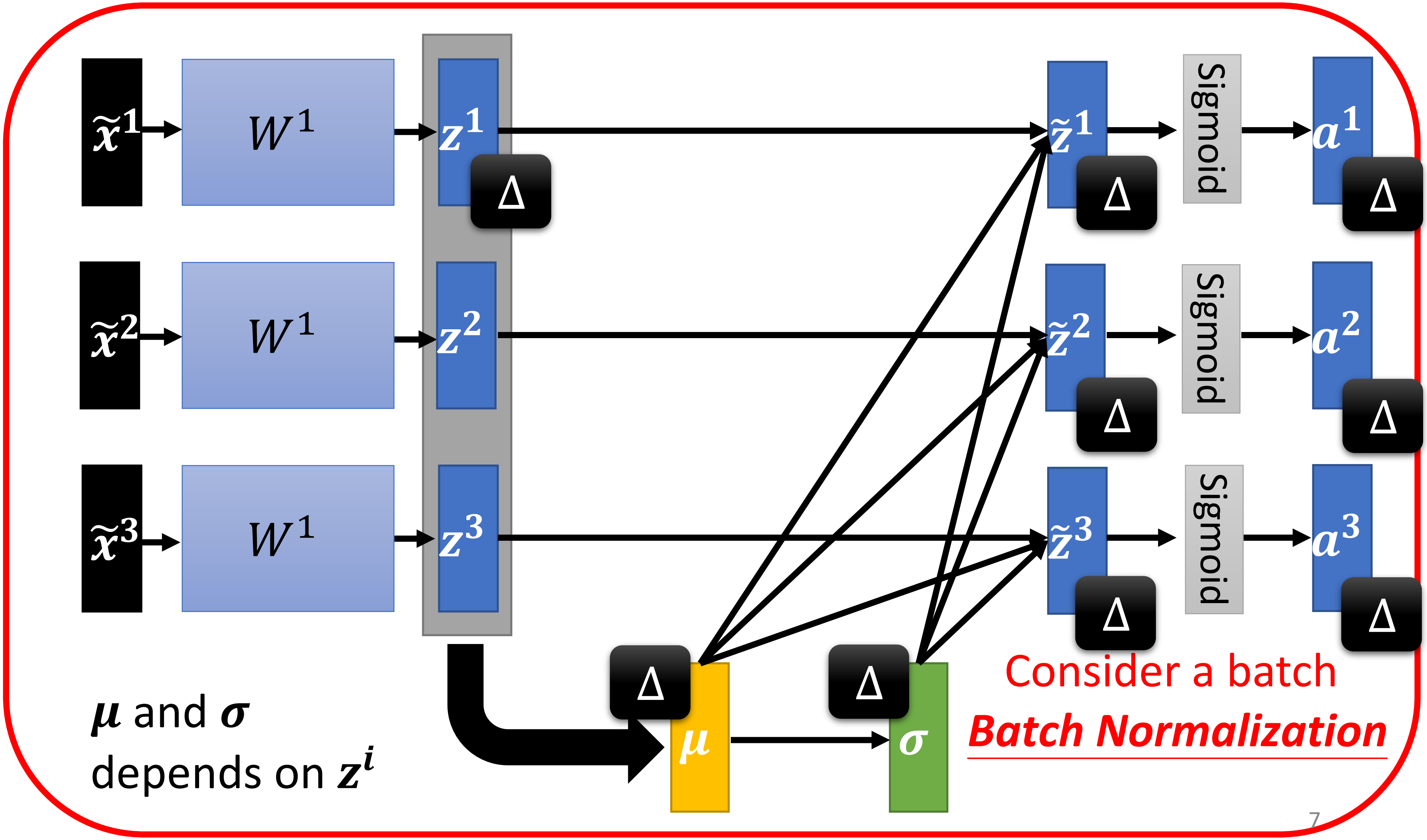
$$\mu = \frac{1}{3}\sum_{i=1}^{3} z^i$$

$$\sigma = \sqrt{\frac{1}{3}\sum_{i=1}^{3}\left(z^i - \mu\right)^2}$$

Considering Deep Learning

$$\tilde{z}^i = \frac{z^i - \mu}{\sigma}$$

This is a large network!



$\tilde{x}^1$
$\tilde{x}^2$
$\tilde{x}^3$
W^1
W^1
W^1
z^1
z^2
z^3
$\tilde{z}^1$
$\tilde{z}^2$
$\tilde{z}^3$
Sigmoid
Sigmoid
Sigmoid
a^1
a^2
a^3
μ
σ
Δ

μ and **σ** depends on z^i

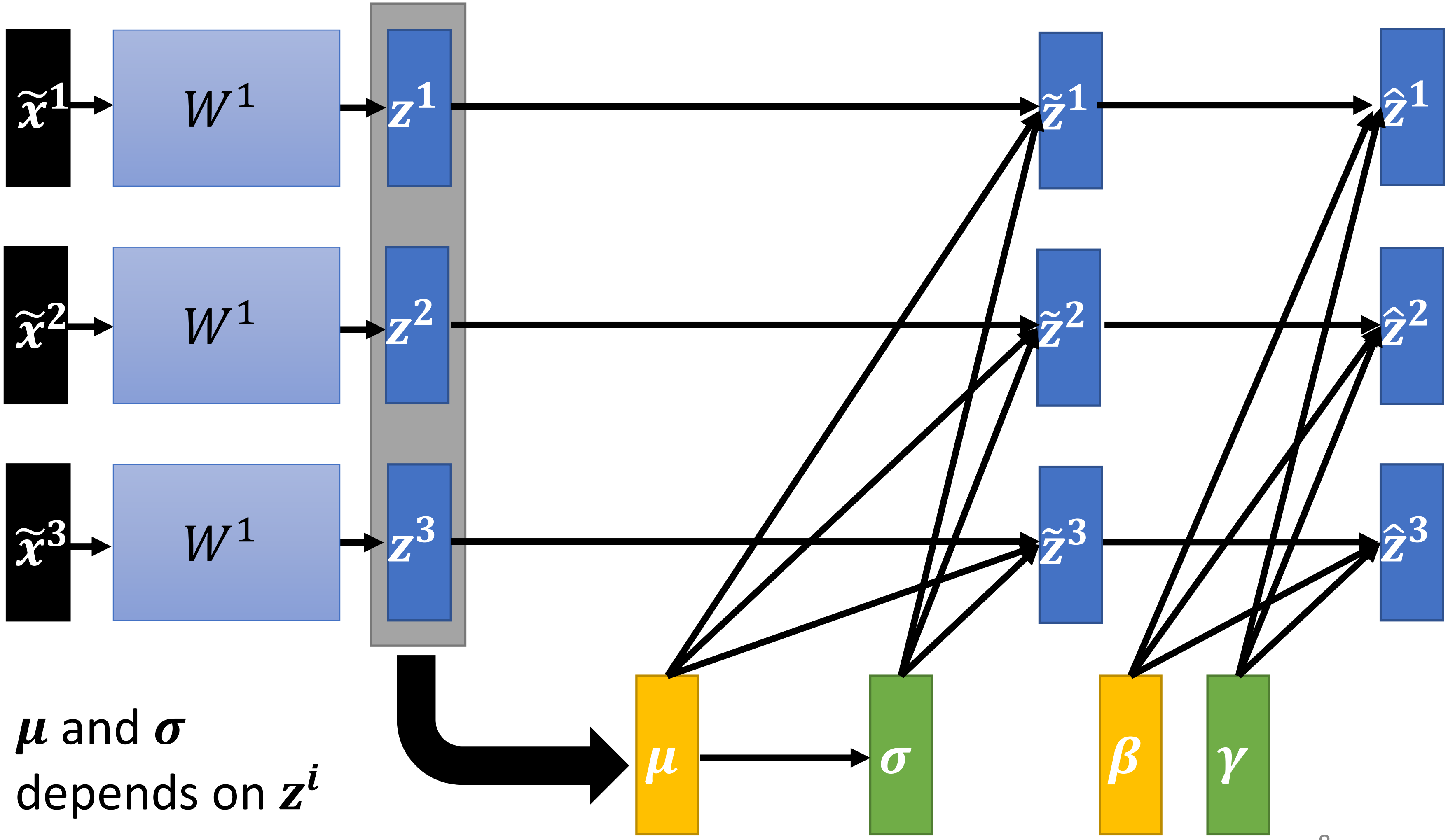
Consider a batch

Batch Normalization

Batch normalization

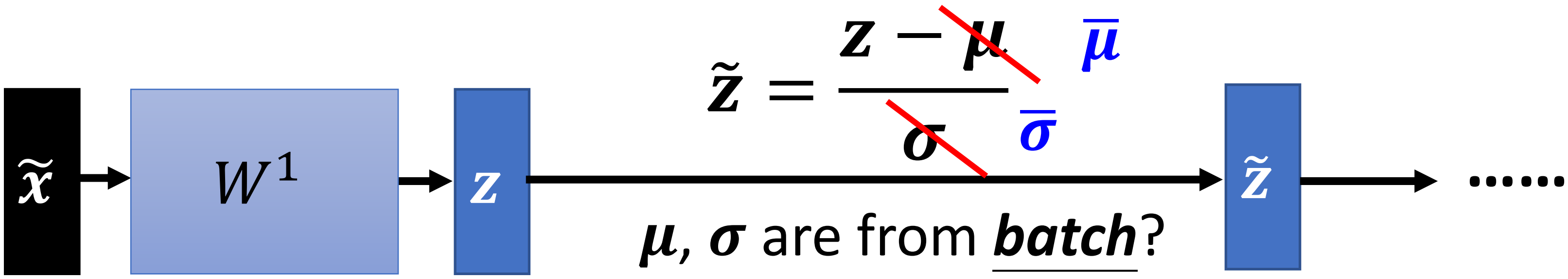
$$\tilde{\boldsymbol{z}}^i = \frac{\boldsymbol{z}^i - \boldsymbol{\mu}}{\boldsymbol{\sigma}}$$

$$\hat{\boldsymbol{z}}^i = \boldsymbol{\gamma} \odot \tilde{\boldsymbol{z}}^i + \boldsymbol{\beta}$$


$\tilde{x}^1$
W^1
z^1
$\tilde{z}^1$
$\hat{z}^1$
$\tilde{x}^2$
W^1
z^2
$\tilde{z}^2$
$\hat{z}^2$
$\tilde{x}^3$
W^1
z^3
$\tilde{z}^3$
$\hat{z}^3$
μ
σ
β
γ

$\boldsymbol{\mu}$ and $\boldsymbol{\sigma}$
depends on $\boldsymbol{z}^i$

Batch normalization – Testing



$$\tilde{\boldsymbol{z}} = \frac{\boldsymbol{z} - \cancel{\boldsymbol{\mu}}}{\cancel{\boldsymbol{\sigma}}} \quad \bar{\boldsymbol{\mu}} \quad \bar{\boldsymbol{\sigma}}$$

$\boldsymbol{\mu}$, $\boldsymbol{\sigma}$ are from ***batch***?

We do not always have ***batch*** at testing stage.

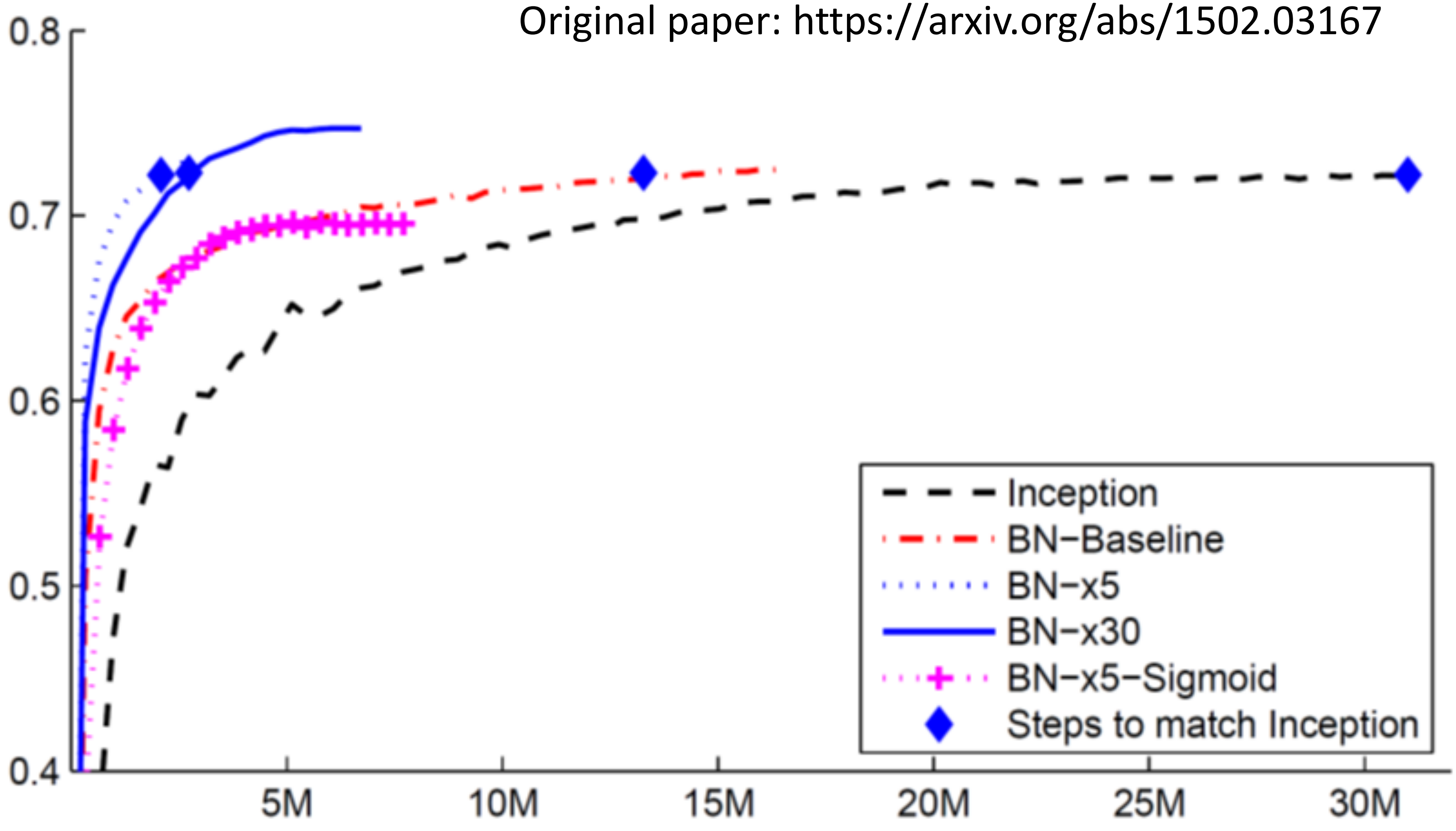
Computing the moving average of $\boldsymbol{\mu}$ and $\boldsymbol{\sigma}$ of the batches during training.

$$\boldsymbol{\mu}^1 \quad \boldsymbol{\mu}^2 \quad \boldsymbol{\mu}^3 \quad \ldots\ldots \quad \boldsymbol{\mu}^t$$

$$\bar{\boldsymbol{\mu}} \leftarrow p\bar{\boldsymbol{\mu}} + (1-p)\boldsymbol{\mu}^t$$

Batch normalization

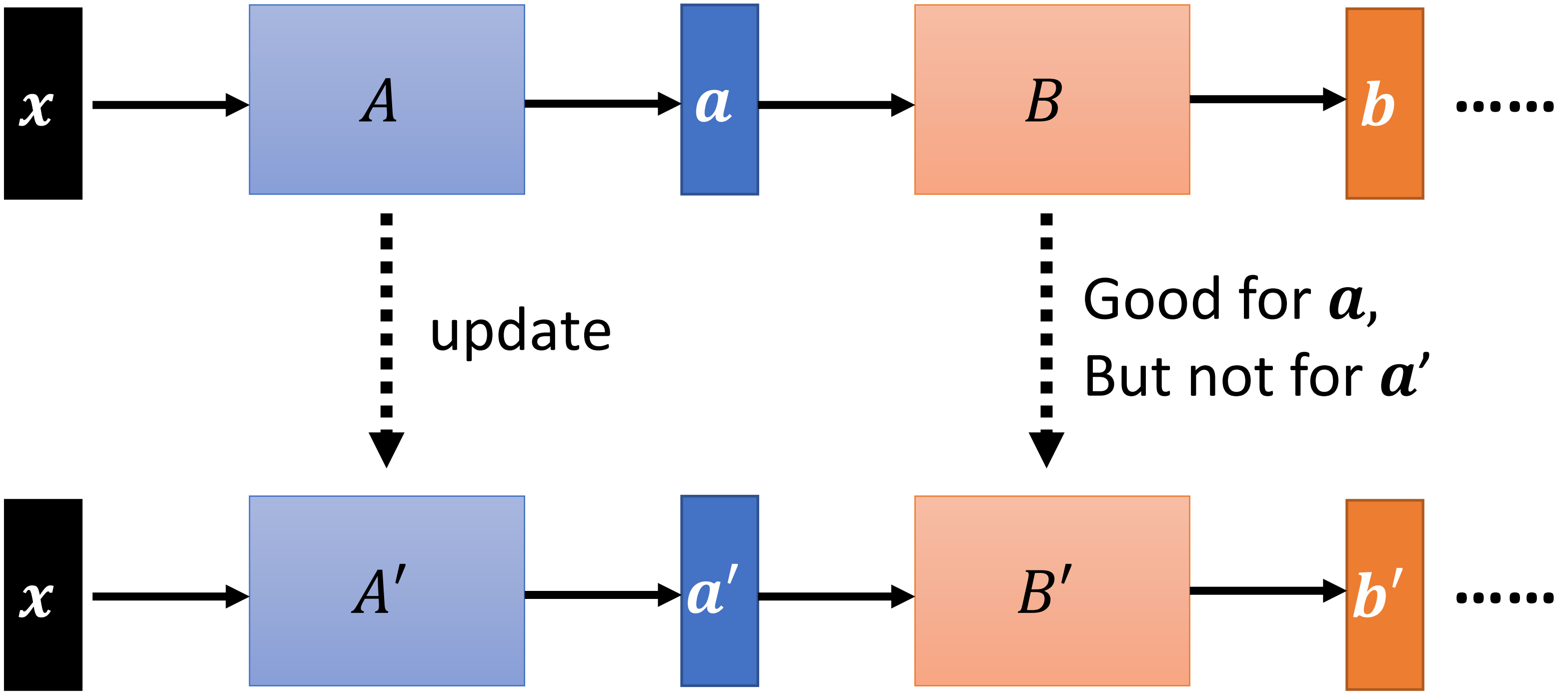
Original paper: https://arxiv.org/abs/1502.03167



Internal Covariate Shift?

How Does Batch Normalization Help Optimization?

https://arxiv.org/abs/1805.11604


x
A
$\boldsymbol{a}$
B
$\boldsymbol{b}$
……
x
A'
$\boldsymbol{a}'$
B'
$\boldsymbol{b}'$
……

update

Good for $\boldsymbol{a}$,
But not for $\boldsymbol{a}'$

Batch normalization make $\boldsymbol{a}$ and $\boldsymbol{a}'$ have similar statistics.

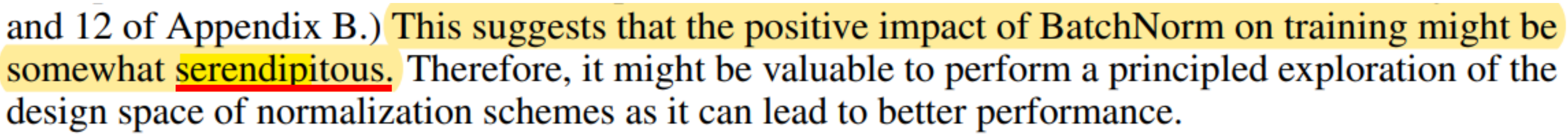
Experimental results do not support the above idea.

Internal Covariate Shift?

How Does Batch Normalization Help Optimization?

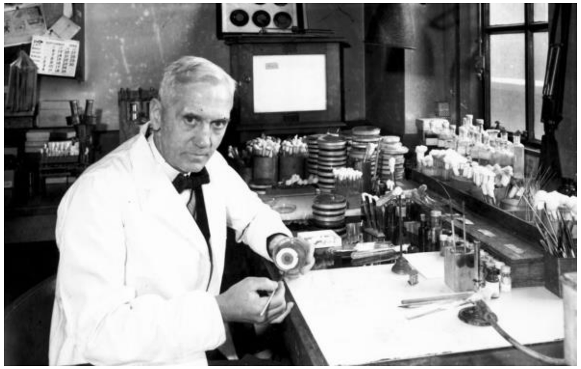
https://arxiv.org/abs/1805.11604

Experimental results (and theoretically analysis) support batch normalization change the landscape of error surface.



and 12 of Appendix B.) This suggests that the positive impact of BatchNorm on training might be somewhat serendipitous. Therefore, it might be valuable to perform a principled exploration of the design space of normalization schemes as it can lead to better performance.

serendipitous (偶然的)



penicillin

To learn more ……

- Batch Renormalization
  - https://arxiv.org/abs/1702.03275
- Layer Normalization
  - https://arxiv.org/abs/1607.06450
- Instance Normalization
  - https://arxiv.org/abs/1607.08022
- Group Normalization
  - https://arxiv.org/abs/1803.08494
- Weight Normalization
  - https://arxiv.org/abs/1602.07868
- Spectrum Normalization
  - https://arxiv.org/abs/1705.10941

Q&A

有一篇博客写的太好了，所以直接拿来用了，

链接是：https://blog.csdn.net/weixin_44023658/article/details/105844861

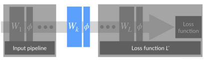
Batch Normalization（BN）超详细解析

原创 越前浩波 2020-04-29 18:33:51 11481 收藏 165 原力计划
分类专栏： 图像 机器/深度学习 python 文章标签： 神经网络 计算机视觉 机器学习

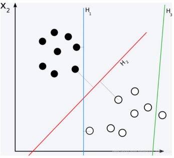
版权声明：本文为博主原创文章，遵循 CC 4.0 BY-SA 版权协议，转载请附上原文出处链接和本声明。
本文链接：https://blog.csdn.net/weixin_44023658/article/details/105844861

图像 同时被 3 个专栏收录 2 订阅 27 篇文章 订阅专栏

单层视角



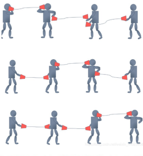
神经网络可以看成是上图形式，对于中间的某一层，其前面的层可以看成是对输入的处理，后面的层可以看成是损失函数。一次反向传播过程会同时更新所有层的权重W1,W2,…,WL，前面层权重的更新会改变当前层输入的分布，而跟据反向传播的计算方式，我们知道，对Wk的更新是在假定其输入不变的情况下进行的。如果假定第k层的输入节点只有2个，对第k层的某个输出节点而言，相当于一个线性模型y=w1x1+w2x2+b，如下图所示，



假定当前输入x1和x2的分布如图中圆点所示，本次更新的方向是将直线H1更新成H2，本以为切分得不错，但是当前面层的权重更新完毕，当前层输入的分布换成了另外一番样子，直线相对输入分布的位置可能变成了H3，下一次更新又要根据新的分布重新调整。直线调整了位置，输入分布又在发生变化，直线再调整位置，就像是直线和分布之间的"追逐游戏"。对于浅层模型，比如SVM，输入特征的分布是固定的，即使拆分成不同的batch，每个batch的统计特性也是相近的，因此只需调整直线位置来适应输入分布，显然要容易得多。而深层模型，每层输入的分布和权重在同时变化，训练相对困难。

多层视角

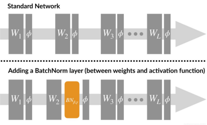
上面是从网络中单拿出一层分析，下面看一下多层的情况。在反向传播过程中，每层权重的更新是在假定其他权重不变的情况下，向损失函数降低的方向调整自己。问题在于，在一次反向传播过程中，所有的权重会同时更新，导致层间配合"缺乏默契"，每层都在进行上节所说的"追逐游戏"，而且层数越多，相互配合越困难，文中把这个现象称之为 Internal Covariate Shift，示意图如下。为了避免过于震荡，学习率不得不设置得足够小，足够小就意味着学习缓慢。



为此，希望对每层输入的分布有所控制，于是就有了Batch Normalization，其出发点是对每层的输入做Normalization，只有一个数据是谈不上Normalization的，所以是对一个batch的数据进行Normalization。

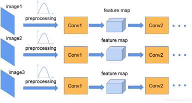
Batch Normalization原理

Batch Normalization，简称BatchNorm或BN，翻译为"批归一化"，是神经网络中一种特殊的层，如今已是各种流行网络的标配。在原paper中，BN被建议插入在（每个）ReLU激活层前面，如下所示，

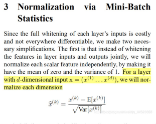

Standard Network
W_1 ϕ W_2 ϕ W_3 ϕ W_L ϕ
Adding a BatchNorm layer (between weights and activation function)
W_1 ϕ W_2 $BN_{\beta,\gamma}$ ϕ W_3 ϕ W_L ϕ

如果batch size为m，则在前向传播过程中，网络中每个节点都有m个输出，所谓的Batch Normalization，就是对该层每个节点的这m个输出进行归一化再输出。

我们在图像预处理过程中通常会对图像进行标准化处理，这样能够加速网络的收敛，如下图所示，对于Conv1来说输入的就是满足某一分布的特征矩阵，但对于Conv2而言输入的feature map就不一定满足某一分布规律了**（注意这里所说满足某一分布规律并不是指某一个feature map的数据要满足分布规律，理论上是指整个训练样本集所对应feature map的数据要满足分布规律）**。而我们Batch Normalization的目的就是使我们的feature map满足均值为0，方差为1的分布规律。



下面是从原论文中截取的原话，注意标黄的部分：

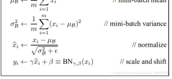


3 Normalization via Mini-Batch Statistics

Since the full whitening of each layer's inputs is costly and not everywhere differentiable, we make two necessary simplifications. The first is that instead of whitening the features in layer inputs and outputs jointly, we will normalize each scalar feature independently, by making it have the mean of zero and the variance of 1. For a layer with d-dimensional input $\mathrm{x} = (x^{(1)} \ldots x^{(d)})$, we will normalize each dimension

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

"对于一个拥有d维的输入x，我们将对它的每一个维度进行标准化处理。" 假设我们输入的x是RGB三通道的彩色图像，那么这里的d就是输入图像的channels即d=3，，其中就代表我们的R通道所对应的特征矩阵，依此类推。标准化处理也就是分别对我们的R通道，G通道，B通道进行处理。上面的公式不用看，原文提供了更加详细的计算公式：



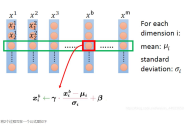
```
Input: Values of $x$ over a mini-batch: $\mathcal{B} = \{x_{1\ldots m}\}$;
       Parameters to be learned: $\gamma, \beta$
Output: $\{y_i = \mathrm{BN}_{\gamma,\beta}(x_i)\}$

    $\mu_\mathcal{B} \leftarrow \frac{1}{m}\sum_{i=1}^{m} x_i$                          // mini-batch mean
    $\sigma_\mathcal{B}^2 \leftarrow \frac{1}{m}\sum_{i=1}^{m}(x_i - \mu_\mathcal{B})^2$   // mini-batch variance
    $\widehat{x}_i \leftarrow \frac{x_i - \mu_\mathcal{B}}{\sqrt{\sigma_\mathcal{B}^2 + \epsilon}}$            // normalize
    $y_i \leftarrow \gamma\widehat{x}_i + \beta \equiv \mathrm{BN}_{\gamma,\beta}(x_i)$       // scale and shift
```

其操作可以分成2步，

1. Standardization：首先对m个x进行 Standardization，得到 zero mean unit variance的分布x^。
2. scale and shift：然后再对x^进行scale and shift，缩放并平移到新的分布y，具有新的均值β方差γ。

假设BN层有d个输入节点，则x可构成d×m大小的矩阵X，BN层相当于通过行操作将其映射为另一个d×m大小的矩阵Y，如下所示，


x^1
x^2
x^3
x^b
x^m
x_1^1
x_2^1
x_1^2
x_2^2
……
……
For each dimension i:
mean: μ_i
standard deviation: σ_i
$x_i^b \leftarrow \gamma \cdot \frac{x_i^b - \mu_i}{\sigma_i} + \beta$

将2个过程写在一个公式里如下

$$y_i^{(b)} = BN(x_i)^{(b)} = \gamma \cdot \left(\frac{x_i^{(b)} - \mu(x_i)}{\sqrt{\sigma(x_i)^2 + \epsilon}}\right) + \beta$$

其中，x(b)i表示输入当前batch的b-th样本时该层i-th输入节点的值，xi为[x(1)i,x(2)i,...,x(m)i]构成的行向量，长度为batch size m，μ和σ为该行的均值和标准差，ϵ为防止除零引入的极小量（可忽略），γ和β为该行的scale和shift参数，可知

- μ和σ为当前行的统计量，不可学习。
- γ和β为待学习的scale和shift参数，用于控制yi的方差和均值
- BN层中，xi和xj之间不存在信息交流(i≠j)

可见，无论xi原本的均值和方差是多少，通过BatchNorm后其均值和方差分别变为待学习的β和γ。

Batch Normalization的反向传播

对于目前的神经网络计算框架，一个层要想加入到网络中，要保证其是可微的，即可以求梯度。BatchNorm的梯度该如何求取？

反向传播求梯度只需抓住一个关键点，如果一个变量对另一个变量有影响，那么他们之间就存在偏导数，找到直接相关的变量，再配合链式法则，公式就很容易写出了。

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

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

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

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

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

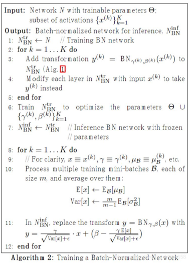
$$\frac{\partial \ell}{\partial x_i} = \frac{\partial \ell}{\partial \widehat{x}_i} \cdot \frac{1}{\sqrt{\sigma_\mathcal{B}^2 + \epsilon}} + \frac{\partial \ell}{\partial \sigma_\mathcal{B}^2} \cdot \frac{2(x_i - \mu_\mathcal{B})}{m} + \frac{\partial \ell}{\partial \mu_\mathcal{B}} \cdot \frac{1}{m}$$

Batch Normalization的预测阶段

在预测阶段，所有参数的取值是固定的，对BN层而言，意味着μ、σ、γ、β都是固定值。

γ和β比较好理解，随着训练结束，两者最终收敛，预测阶段使用训练结束时的值即可。

对于μ和σ，在训练阶段，它们为当前mini batch的统计量，随着输入batch的不同，μ和σ一直在变化。在预测阶段，输入数据可能只有1条，该使用哪个μ和σ，或者说，每个BN层的μ和σ该如何取值？可以采用训练收敛最后几批mini batch的 μ和σ的期望，作为预测阶段的μ和σ，如下所示，



```
Input: Network $N$ with trainable parameters $\Theta$;
       subset of activations $\{x^{(k)}\}_{k=1}^K$
Output: Batch-normalized network for inference, $N_{\mathrm{BN}}^{\mathrm{inf}}$
 1: $N_{\mathrm{BN}}^{\mathrm{tr}} \leftarrow N$    // Training BN network
 2: for $k = 1 \ldots K$ do
 3:     Add transformation $y^{(k)} = \mathrm{BN}_{\gamma^{(k)},\beta^{(k)}}(x^{(k)})$ to
        $N_{\mathrm{BN}}^{\mathrm{tr}}$ (Alg. 1)
 4:     Modify each layer in $N_{\mathrm{BN}}^{\mathrm{tr}}$ with input $x^{(k)}$ to take
        $y^{(k)}$ instead
 5: end for
 6: Train $N_{\mathrm{BN}}^{\mathrm{tr}}$ to optimize the parameters $\Theta \cup \{\gamma^{(k)}, \beta^{(k)}\}_{k=1}^K$
 7: $N_{\mathrm{BN}}^{\mathrm{inf}} \leftarrow N_{\mathrm{BN}}^{\mathrm{tr}}$    // 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:
            $\mathrm{E}[x] \leftarrow \mathrm{E}_{\mathcal{B}}[\mu_{\mathcal{B}}]$
            $\mathrm{Var}[x] \leftarrow \frac{m}{m-1}\mathrm{E}_{\mathcal{B}}[\sigma_{\mathcal{B}}^2]$
11:     In $N_{\mathrm{BN}}^{\mathrm{inf}}$, replace the transform $y = \mathrm{BN}_{\gamma,\beta}(x)$ with
        $y = \frac{\gamma}{\sqrt{\mathrm{Var}[x]+\epsilon}} \cdot x + \left(\beta - \frac{\gamma\,\mathrm{E}[x]}{\sqrt{\mathrm{Var}[x]+\epsilon}}\right)$
12: end for
```

Algorithm 2: Training a Batch-Normalized Network

因为Standardization和scale and shift均为线性变换，在预测阶段所有参数均固定的情况下，参数可以合并成y=kx+b的形式，如上图中行号11所示。

Batch Normalization的作用

使用Batch Normalization，可以获得如下好处，

- 可以使用更大的学习率，训练过程更加稳定，极大提高了训练速度。
- 可以将bias置为0，因为Batch Normalization的Standardization过程会移除直流分量，所以不再需要bias。
- 对权重初始化不再敏感，通常权重采样自0均值某方差的高斯分布，以往对高斯分布的方差设置十分重要，有了Batch Normalization后，对与同一个输出节点相连的权重进行放缩，其标准差σ也会放缩同样的倍数，相除抵消。
- 对权重的尺度不再敏感，理由同上，尺度统一由γ参数控制，在训练中决定。
- 深层网络可以使用sigmoid和tanh了，理由同上，BN抑制了梯度消失。
- Batch Normalization具有某种正则作用，不需要太依赖dropout，减少过拟合。

几个常见问题

卷积层如何使用BatchNorm?

```
For convolutional layers, we additionally want the normalization to obey the convolutional property – so that differ

...

so for a mini-batch of size m and feature maps of size p × q, we use the effective mini-batch of size m′

= |B| = m · pq. We learn a pair of parameters γ(k) and β(k) per feature map, rather than per activation.

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

1个卷积核产生1个feature map，1个feature map有1对γ和β参数，同一batch同channel的feature map共享同一对γ和β参数，若卷积层有n个卷积核，则有n对γ和β参数。

没有scale and shift过程可不可以?
BatchNorm有两个过程，Standardization和scale and shift，前者是机器学习常用的数据预处理技术，在浅层模型中，只需对数据进行Standardization即可，Batch Normalization可不可以只有Standardization呢？

答案是可以，但网络的表达能力会下降。

直觉上理解，浅层模型中，只需要模型适应数据分布即可。对深度神经网络，每层的输入分布和权重要相互协调，强制把分布限制在zero mean unit variance并不见得是最好的选择，加入参数γ和β，对输入进行scale and shift，有利于分布与权重的相互协调，特别地，令γ=1,β=0等价于只用Standardization，令γ=σ,β=μ等价于没有BN层，scale and shift涵盖了这2种特殊情况，在训练过程中决定什么样的分布是适合的，所以使用scale and shift增强了网络的表达能力。

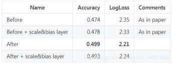
表达能力更强，在实践中性能就会更好吗？并不见得，就像曾经参数越多不见得性能越好一样。在caffenet-benchmark-batchnorm中，作者实验发现没有scale and shift性能可能还更好一些。

BN层放在ReLU前面还是后面?
原paper建议将BN层放置在ReLU前，因为ReLU激活函数的输出非负，不能近似为高斯分布。

```
The goal of Batch Normalization is to achieve a stable distribution of activation values throughout training, and ir

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

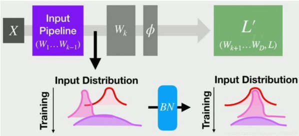
但是，在caffenet-benchmark-batchnorm中，作者基于caffenet在ImageNet2012上做了如下对比实验，



Name	Accuracy	LogLoss	Comments
Before	0.474	2.35	As in paper
Before + scale&bias layer	0.478	2.33	As in paper
After	**0.499**	**2.21**	
After + scale&bias layer	0.493	2.24	

实验表明，放在前后的差异似乎不大，甚至放在ReLU后还好一些。

放在ReLU后相当于直接对每层的输入进行归一化，如下图所示，这与浅层模型的Standardization是一致的。

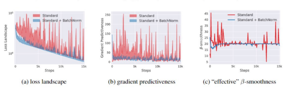


caffenet-benchmark-batchnorm中，还有BN层与不同激活函数、不同初始化方法、dropout等排列组合的对比实验，可以看看。

所以，BN究竟应该放在激活的前面还是后面？以及，BN与其他变量，如激活函数、初始化方法、dropout等，如何组合才是最优？可能只有直觉和经验性的指导意见，具体问题的具体答案可能还是得实验说了算（微笑）。

BN层为什么有效?
BN层的有效性已有目共睹，但为什么有效可能还需要进一步研究，这里有一些解释，

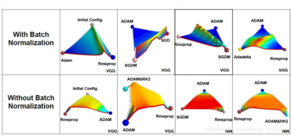
- BN层让损失函数更平滑。论文How Does Batch Normalization Help Optimization中，通过分析训练过程中每步梯度方向上步长变化引起的损失变化范围、梯度幅值的变化范围、光滑度的变化，认为添加BN层后，损失函数的landscape(losssurface)变得更平滑，相比高低不平上下起伏的loss surface，平滑losssurface的梯度预测性更好，可以选取较大的步长。如下图所示，



(a) loss landscape (b) gradient predictiveness (c) "effective" β-smoothness

Figure 4: Analysis of the optimization landscape of VGG networks. At a particular training step, we measure the variation (shaded region) in loss (a) and ℓ_2 changes in the gradient (b) as we move in the gradient direction. The "effective" β-smoothness (c) refers to the maximum difference (in ℓ_2-norm) in gradient over distance moved in that direction. There is a clear improvement in all of these measures in networks with BatchNorm, indicating a more well-behaved loss landscape. (Here, we cap the maximum distance to be $\eta = 0.4\times$ the gradient since for larger steps the standard network just performs worse (see Figure 1). BatchNorm however continues to provide smoothing for even larger distances.) Note that these results are supported by our theoretical findings (Section 4).

- BN更有利于梯度下降。论文An empirical analysis of the optimization of deep network loss surfaces中，绘制了VGG和NIN网络在有无BN层的情况下，loss surface的差异，包含初始点位置以及不同优化算法最终收敛到的local minima位置，如下图所示。没有BN层的，其loss surface存在较大的高原，有BN层的则没有高原，而是山峰，因此更容易下降。



- 这里再提供一个直觉上的理解，没有BN层的情况下，网络没办法直接控制每层输入的分布，其分布前面层的权重共同决定，或者说分布的均值和方差"隐藏"在前面层的每个权重中，网络若想调整其分布，需要通过复杂的反向传播过程调整前面的每个权重实现，BN层的存在相当于将分布的均值和方差从权重中剥离了出来，只需调整γ和β两个参数就可以直接调整分布，让分布和权重的配合变得更加容易。

这里多说一句，论文How Does Batch Normalization Help Optimization中对比了标准VGG以及加了BN层的VGG每层分布随训练过程的变化，发现两者并无明显差异，认为BatchNorm并没有改善 Internal Covariate Shift。但这里有个问题是，两者的训练都可以收敛，对于不能收敛或者训练过程十分震荡的模型呢，其分布变化是怎样的？大家可以自己实验看看。

参考自：
日拱一卒

越前浩波 关注 40 2 165 专栏目录

人工智能--卷积神经网络基础精讲视频 05-18
本课程主要介绍人工智能模型中，卷积神经网络的基础内容和应用。主要包括卷积神经网络的基础概念，梯度计算，反向传播，网络训练等方面的内容。...