Deep Learning Technology and Application

Ge Li

Peking University

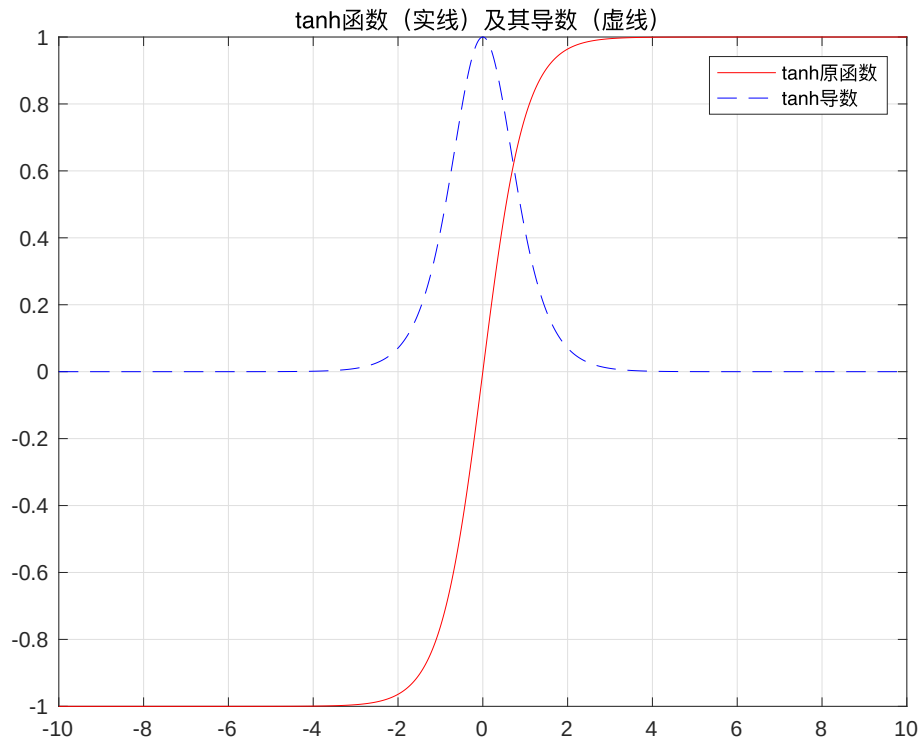
数据预处理

数据预处理

为什么需要进行初始化：

- 消除特征取值范围的影响
  - 训练数据中，每一维特征的来源以及度量单位不同，会造成这些特征值的分布范围差异较大。
  - 在训练过程中，某些取值范围大的特征会起到主导作用，特别是对于基于相似度比较的学习方法。
  - 对样本进行预处理，将各个维度的特征归一化到同一个取值区间，可以摆脱特征变化范围对学习方法的影响。
- 对训练效率的影响
  - 不同特征的取值范围，可能会导致训练效率的降低。

数据预处理

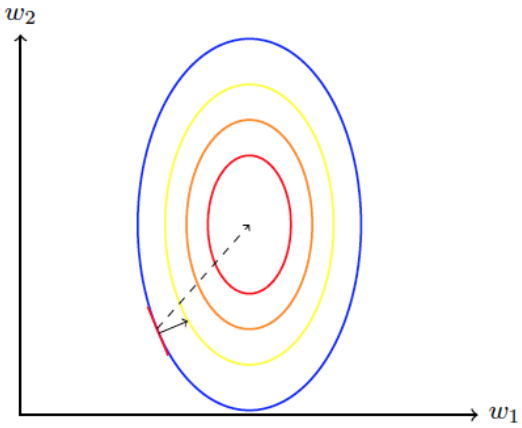


- 为什么预处理会影响训练效率？
- 激活函数的非饱和区具有一定范围。

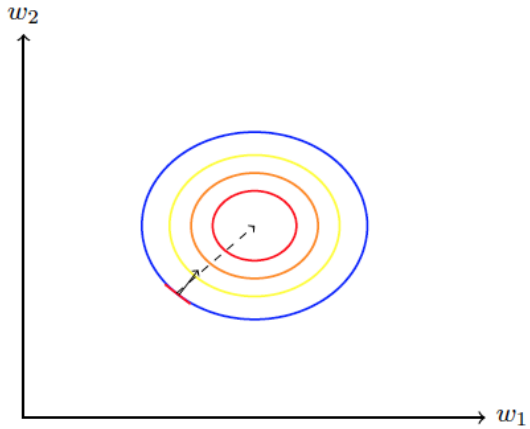
$$y = tanh(w_1x_1 + w_2x_2 + b)$$

- 若 x_1 或 x_2 的取值范围特别大，有可能造成梯度更新很慢

数据预处理



(a) 未归一化数据的梯度



(b) 归一化数据的梯度

- 不同特征取值范围差异较大时还会影响梯度下降法的搜索效率。
- 取值范围不同会造成在大多数位置上的梯度方向并不是最优的搜索方向。
- 当使用梯度下降法寻求最优解时，会导致需要很多次迭代才能收敛。
- 若把数据归一化为取值范围相同，大部分位置的梯度方向近似于最优搜索方向。
- 从而每一步梯度的方向都基本指向最小值，训练效率会大大提高。

数据预处理

Feature Scaling

- 通过对样本某维度值的缩放，将样本数值取值范围归一化到 $[0, 1]$ 或 $[-1, 1]$ 之间：

$$\widehat{x^{(i)}} = \frac{x^{(i)} - min(x)}{max(x) - min(x)}$$

其中 $min(x)$ 与 $max(x)$ 分别为这一维特征在所有样本上的最小值和最大值。

数据预处理

z-score Normalization

- 将每一个维特征都处理为符合标准正态分布的值（均值为 0，标准差为 1 的正态分布）。
- 假设有 N 个样本，设一个样本 x 的第 i 维为 $x^{(i)}$；
- 计算所有样本第 i 维的均值和标准差：

$$\mu = \frac{1}{N}\sum_{i=1}^{N} x^{(i)}$$

$$\sigma^2 = \frac{1}{N}\sum_{i=1}^{N}(x^{(i)} - \mu)^2$$

- 对所有样本的第 i 维进行归一化处理：

$$\widehat{x^{(i)}} = \frac{x^{(i)} - \mu}{\sigma}$$

数据预处理

Whitening (白化)

- 白化的目的是去除输入数据的冗余信息。
- 输入数据集 X，经过白化处理后的数据 $\hat{X}$ 满足两个性质：
  1. 特征之间相关性较低；
  2. 所有特征具有相同的方差；
- 两种常用的白化方法：
  1. PCA Whitening
  2. ZCA Whitening

数据预处理

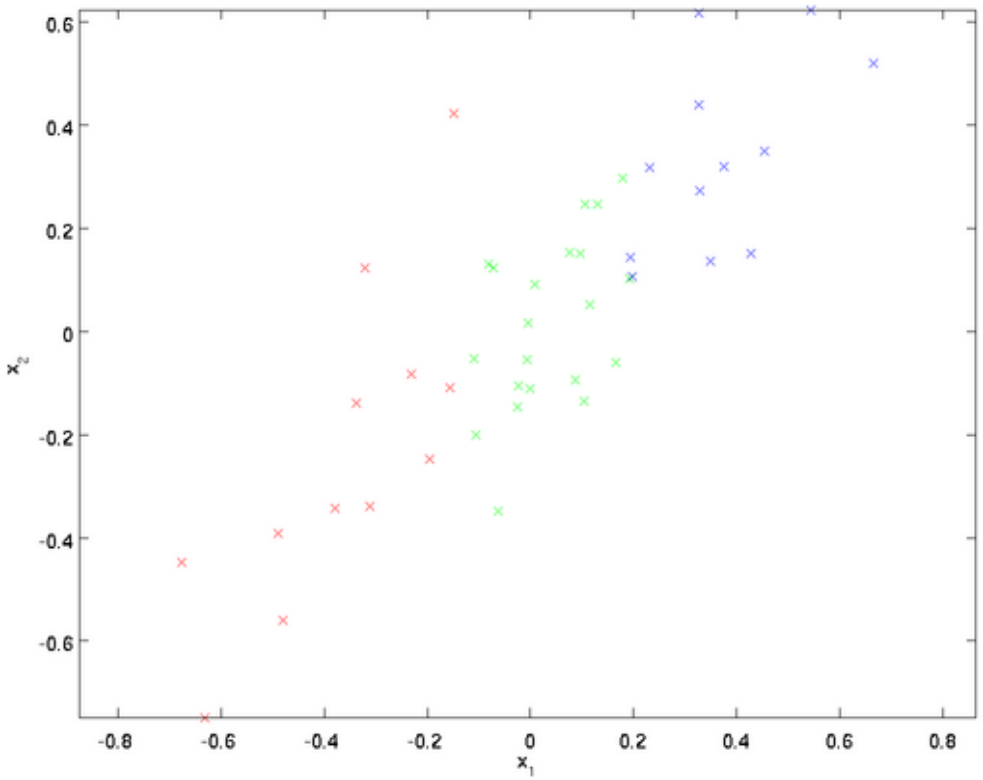
Principal Components Analysis (PCA) 主成分分析

- 主成分分析（PCA）是一种能够极大提升无监督特征学习速度的数据降维算法。
- 假设使用图像来训练算法，因为图像中相邻的像素高度相关，输入数据是有一定冗余的。

  例如我们正在训练的 16×16 灰度值图像，记为一个 256 维向量 $x \in R^{256}$，其中特征值 x_j 对应每个像素的亮度值。

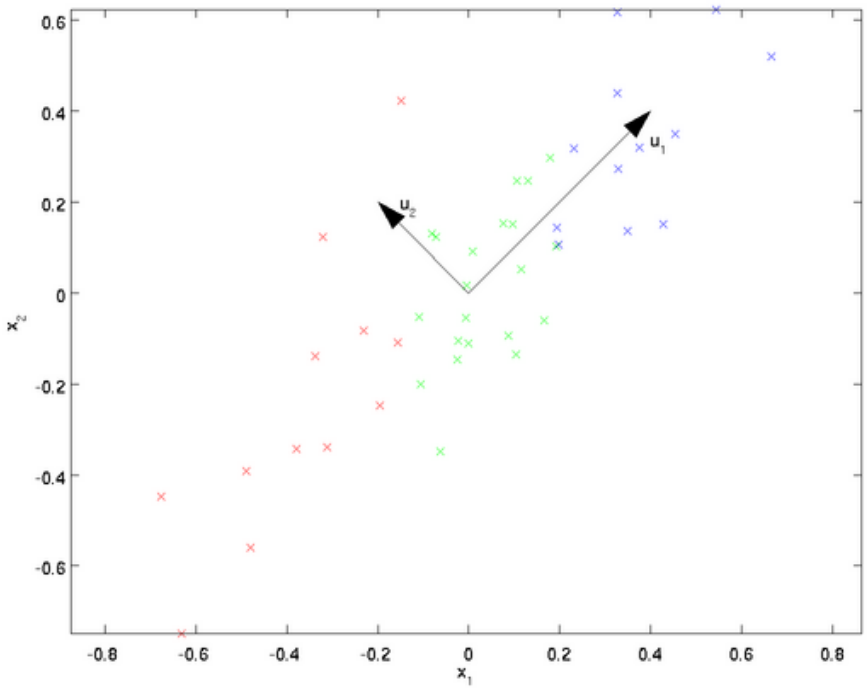
  由于相邻像素间的相关性，PCA 算法可以将输入向量转换为一个维数低很多的近似向量，而且误差非常小。

PCA



- 设输入数据集为 $x_1, x_2, \ldots, x_m$，且 $x_i \in R^2$，即数据维度为 2；
- 设上述数据已经经过预处理，即每个特征 $x^{(1)}, x^{(2)}$ 具有相同的均值和方差；
- PCA 算法将寻找一个低维空间来投影我们的数据。

PCA

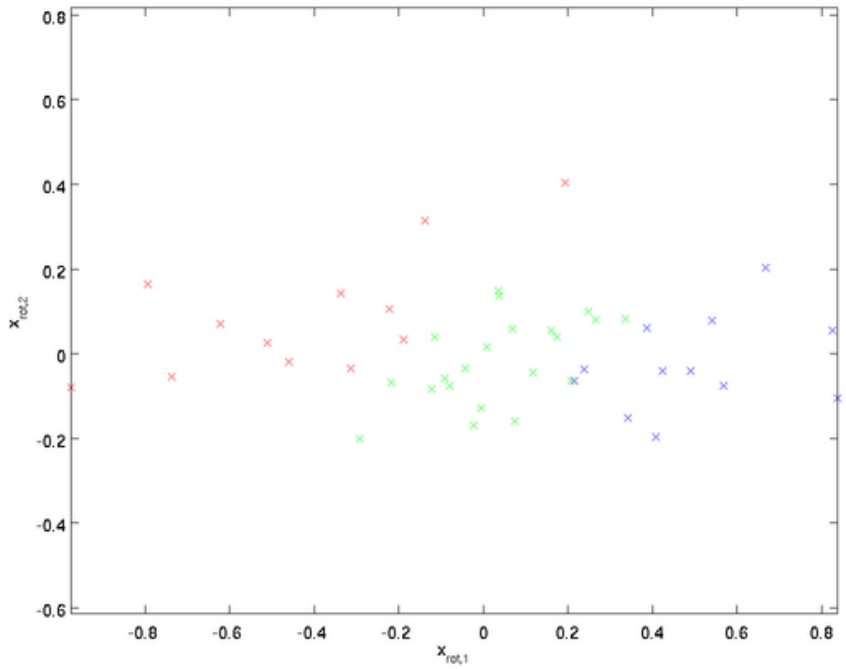


- u_1 是数据变化的主方向，u_2 是次方向；
- $u_1^T x$ 是样本点 x 在维度 u_1 上的投影的长度（幅值），$u_2^T x$ 是样本点 x 在维度 u_2 上的投影的长度；
- 可以用一组新的基重新表示每一个输入向量 x，即将训练数据旋转到基 $u_1, u_2, \ldots, u_n$ 上：

$$x_{rot} = U^T x = \begin{bmatrix} u_1^T x \\ u_2^T x \end{bmatrix}$$

- 矩阵 U 具有正交性，即 $U^T U = U U^T = 1$，若想将旋转后的数据旋转回去，可以取 $x = U x_{rot}$

PCA



如何求取 U？

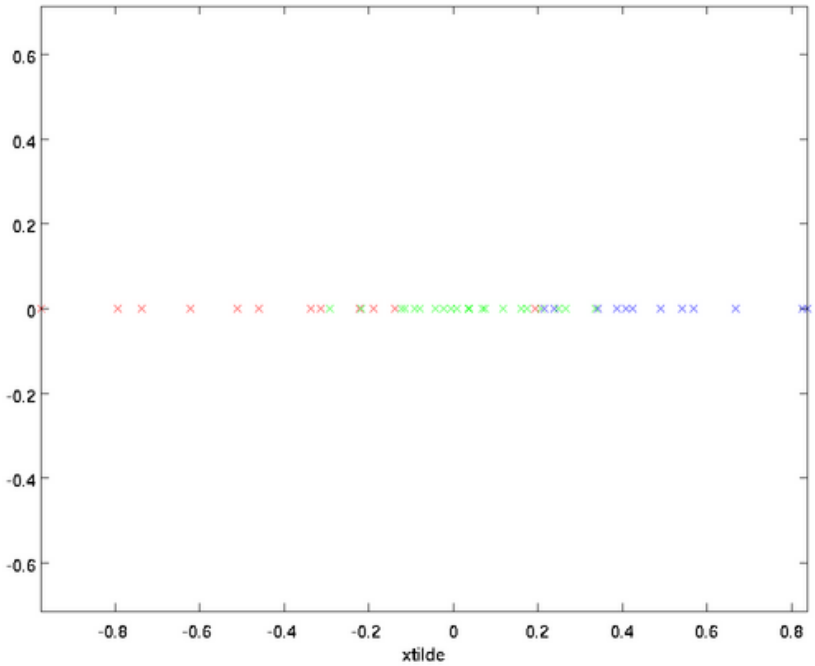
- 首先计算输入数据的协方差矩阵 Σ：

$$\Sigma = \frac{1}{m}\Sigma_{i=1}^{m}(x^{(i)})(x^{(j)})^T$$

- 然后，通过代数运算求得该矩阵的特征向量：

$$\begin{bmatrix} | & | & \cdots & | \\ u_1 & u_2 & \cdots & u_n \\ | & | & \cdots & | \end{bmatrix}$$

PCA



- 可以将训练数据旋转到基 $u_1, u_2, \ldots, u_n$ 上，这样就保留了所有的原始训练数据；
- 但也可以只讲训练数据旋转到部分基 $(u_1, u_2, \ldots, u_k)$ 上，这样就达到了“有选择的保留训练数据的目的”，也就实现了数据的“降维”。
- 例如，左图是将原二维数据降到一维的结果。

PCA

降维可多可少，如何确定该降到多少维呢？
即，如何确定 k 值呢？

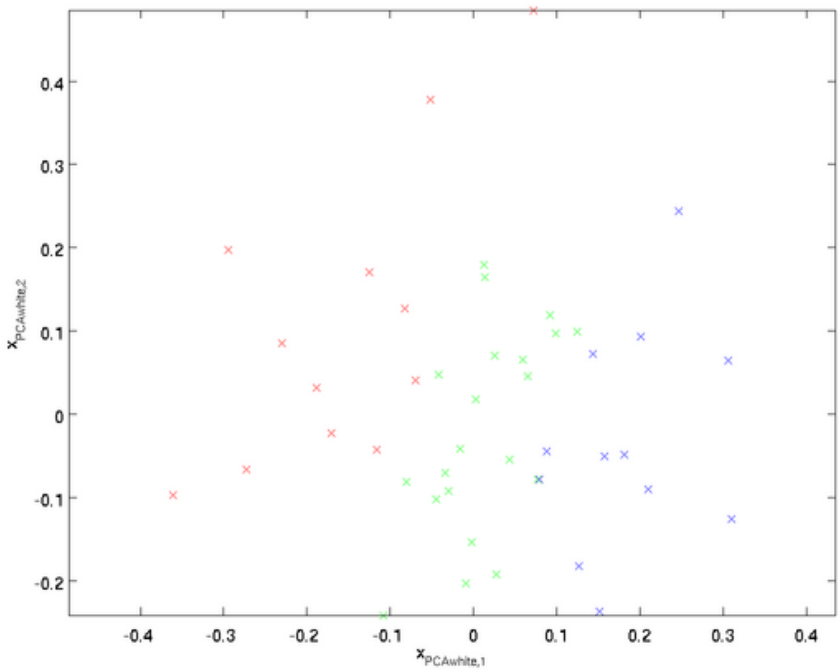
- k 太大，则压缩率太低；k 太小，则近似误差太大；
- 决定 k 值时，我们通常会考虑不同 k 值可保留的方差百分比：

  设：$\lambda_1, \lambda_2, \ldots, \lambda_n$ 为协方差矩阵 Σ 的特征值，按由大到小排列，即 λ_j 为对应于特征向量 j 的特征值，则保留前 k 个成分的方差百分比为：

$$\frac{\Sigma_{j=1}^{k} \lambda_j}{\Sigma_{j=1}^{n} \lambda_j}$$

- 以图像处理为例，通常取方差百分比大于 0.99 的最小的 k；

PCA 白化

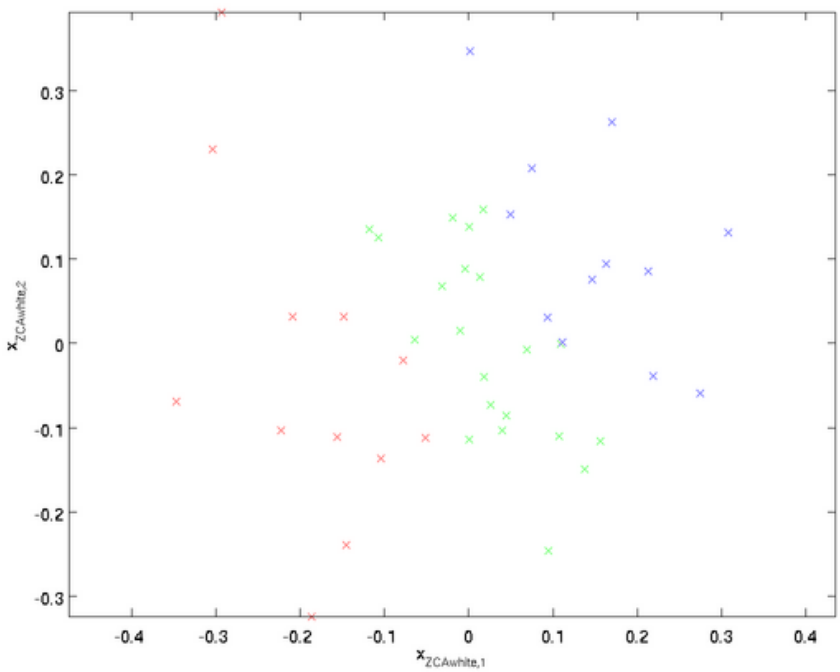


- 上面已经消除了输入特征 x_i 之间的相关性；
- 即在上例中 $x_{rot}^{(1)}$ 与 $x_{rot}^{(2)}$ 是不相关的；
- 现在，为了使每个输入特征具有单位方差，可以直接使用 $\frac{1}{\sqrt{\lambda_i}}$ 作为缩放因子来缩放每个特征 $x_{rot}^{(i)}$，即

$$x_{PCAwhite}^{(i)} = \frac{x_{rot}^{(i)}}{\sqrt{\lambda_i}}$$

经过上述处理以后，$x_{PCAwhite}$ 中不同的特征之间不相关，且具有单位方差。

ZCA 白化



- PCA 白化中，对数据进行了旋转，在实际处理中，可以再对 PCA 白化以后的结果进行还原旋转，以得到更加贴近原始数据的结果；即：

$$x_{ZCAwhite} = U x_{PCAwhite}$$

- 当使用 ZCA 白化时，通常保留数据的全部 n 个维度，不降低它的维数。

Table of contents

Batch Normalization

Batch Normalization

进行 Batch Normalization 的原因：

- 神经网络的训练目标是在输出层得到原始输入数据数据分布的一个映射，即在训练过程中，应该力图保证数据分布的映射关系；若数据分布在训练过程中发生了偏移，则会降低网络的泛化能力；
- 在网络训练过程中，后一层网络的输入是前一层的输出，因此，前一层网络参数的变化，将导致后一层输入数据分布的改变，且这种改变会在训练过程中向后传递并被逐步放大；这种在训练过程中，数据分布的改变称为“Internal Covariate Shift”;
- 在 Batch-based Training 中，若个 Batch 的分布各不相同，网络需要在每个 Batch 的训练中适应不同的分布，从而大大降低训练速度；

所以，为了避免上述问题，可以考虑针对每层网络的每个 Batch 进行 Batch Normalization. 这是一种提高训练速度和效果的非常有效的方法.

Batch Normalization

进行 Batch Normalization 的条件：

1. 起到 Normalize 的作用：控制数据的均值与方差在一定范围内；
2. 保持 Normalize 之前的数据与 Normalize 之后数据之间的映射关系；
3. 保证 Normalize 方法 / 函数的可导性；

论文 [1] 提出了一种针对每个 Batch 进行 Normalize 的方法：

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

[1]Ioffe, Sergey, and Christian Szegedy. "Batch normalization: Accelerating deep network training by reducing internal covariate shift." arXiv preprint arXiv:1502.03167 (2015).

Batch Normalization

Input: Values of x over a mini-batch: $\mathcal{B} = \{x_{1...m}\}$;
Parameters to be learned: γ, β

Output: $\{y_i = \mathrm{BN}_{\gamma,\beta}(x_i)\}$

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

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

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

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

Batch Normalization

反向传播阶段的计算：

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

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

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

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

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

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

Batch Normalization

数据测试阶段：

因为测试阶段输入数据可能只有一个，因此，我们使用所有 Batch 的 μ_B 的期望值代替上述公式中的 $E[x]$；使用所有 Batch 方差 δ_B^2 的无偏估计代替上述公式中的 $\text{Var}[x]$，即：

$$\begin{aligned}\mathrm{E}[x] &\leftarrow \mathrm{E}_{\mathcal{B}}[\mu_{\mathcal{B}}] \\ \mathrm{Var}[x] &\leftarrow \frac{m}{m-1}\mathrm{E}_{\mathcal{B}}[\sigma_{\mathcal{B}}^2]\end{aligned}$$

又因为，上述公式中：

$$\widehat{x}_i \leftarrow \frac{x_i - \mu_{\mathcal{B}}}{\sqrt{\sigma_{\mathcal{B}}^2 + \epsilon}}$$

则，Batch Normalization 层计算的公式为：

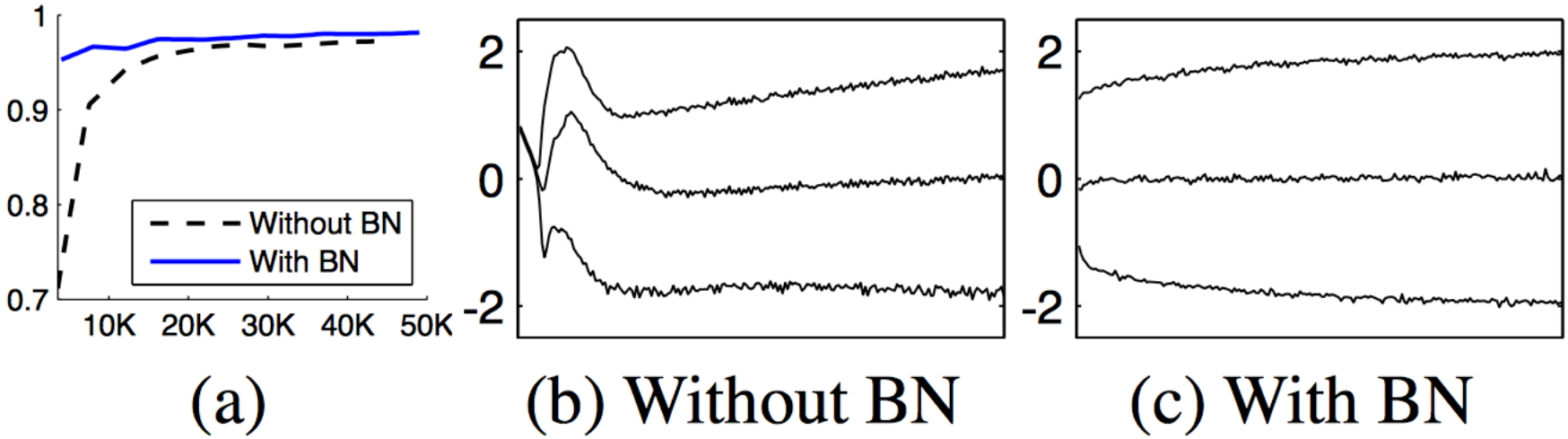
$$y = \frac{\gamma}{\sqrt{\mathrm{Var}[x]+\epsilon}} \cdot x + \left(\beta - \frac{\gamma\,\mathrm{E}[x]}{\sqrt{\mathrm{Var}[x]+\epsilon}}\right)$$

Batch Normalization

批量归一化操作可以看作是一个特殊的神经层，加在每一层非线性激活函数之前：

$$a^{(l)} = f(BN_{\gamma,\beta}(z^{(l)})) = f(BN_{\gamma,\beta}(Wa^{(l-1)}))$$

Batch Normalization 的效果：



(a) (b) Without BN (c) With BN

Layer Normalization

层归一化

- 批量归一化是对一个中间层的单个神经元进行归一化操作，因此要求小批量样本的数量不能太小，否则难以计算单个神经元的统计信息。
- 如果一个神经元的净输入的分布在神经网络中是动态变化的，比如循环神经网络，那么就无法应用批量归一化操作。
- 层归一化是和批量归一化非常类似的方法，不同的是，层归一化是对一个中间层的所有神经元进行归一化。

Lei Jimmy Ba, Ryan Kiros, and Geoffrey E. Hinton. Layer normalization. CoRR, abs/1607.06450, 2016. URL http://arxiv.org/abs/1607.06450.

Layer Normalization

层归一化

- 设神经网络第 l 曾神经元的输入为 $z^{(l)}$；
- 设 n^l 为第 l 层神经元的数量，计算其均值和方差：

$$\mu^{(l)} = \frac{1}{n^l}\Sigma_{i=1}^{n^l} z_i^{(l)}$$

$$\sigma^{(l)^2} = \frac{1}{n^l}\Sigma_{k=1}^{n^l}(z_i^{(l)} - \mu^{(l)})^2$$

- 设 γ 与 β 分别为缩放和平移的参数向量（与 $z^{(l)}$ 维度相同），则层归一化的计算为：

$$\hat{z^{(l)}} = \frac{z^{(l)} - \mu^{(l)}}{\sqrt{\sigma^{(l)^2} + \epsilon}} \odot \gamma + \beta$$

$$\equiv LN_{\gamma,\beta}(z^{(l)})$$

Layer Normalization

例：在循环神经网络中

- 设 t 时刻循环神经网络的隐藏层为 h_t ；
- 设 t 时刻的输入为 x_t，U，W 分别为网络参数，则层归一化的更新计算为：

$$z_t = Uh_{t-1} + Wx_t;$$
$$h_t = f(LN_{\gamma,\beta}(z_t))$$

循环神经网络中，循环神经层的净输入可能会随着时间慢慢变大或变小，从而导致梯度爆炸或消失，而层归一化可以有效缓解这种状况。

Thanks.