批归一化

王树森

特征标准化

为什么要做特征标准化

人的特征向量 $\mathbf{x}_1, \cdots, \mathbf{x}_n \in \mathbb{R}^2$.

- 第一个维度是一个人的收入 (单位 dollars).
  - 假设它是从高斯分布中随机抽取的，均值为 3000，标准差为400^2 .
- 第二个维度是一个人的身高 (单位 inch).
  - 假设它是从高斯分布中随机抽取的，均值为 69，标准差为 3^2 .

为什么要做特征标准化

人的特征向量 $\mathbf{x}_1, \cdots, \mathbf{x}_n \in \mathbb{R}^2$.

- 第一个维度是一个人的收入 (单位 dollars).
  - 假设它是从高斯分布中随机抽取的，均值为 3000，标准差为400^2 .
- 第二个维度是一个人的身高 (单位 inch).
  - 假设它是从高斯分布中随机抽取的，均值为 69，标准差为 3^2 .

- 最小二乘回归模型的Hessian矩阵

$$\mathbf{H} = \frac{1}{n}\sum_{i=1}^{n} \mathbf{x}_i \mathbf{x}_i^T = \begin{bmatrix} 9137.3 & 206.6 \\ 206.6 & 4.8 \end{bmatrix} \times 10^3.$$

- Condition number: $\frac{\lambda_{\max}(\mathbf{H})}{\lambda_{\min}(\mathbf{H})} = 9.2\times 10^4$.

条件数过大意味着梯度下降收敛速度慢！

译者注：Hessian矩阵为损失函数的二阶求导，可以用来衡量梯度。特征向量和特征值表示梯度下降的优化方向和收敛速度

condition number = 最大的收敛速度/最小的收敛速度

如果非常大，则表示收敛慢，甚至可能不收敛

为什么要做特征标准化

People's feature vectors: $\mathbf{x}_1, \cdots, \mathbf{x}_n \in \mathbb{R}^2$.

- 第一个维度是一个人的收入 (单位 thousand dollars).
  - 假设它是从高斯分布中随机抽取的，均值为 3，标准差为 0.4^2 .
- 第二个维度是一个人的身高 (单位 foot).
  - 假设它是从高斯分布中随机抽取的，均值为 5.75 标准差为 0.25^2 .
- 矩阵改变.
- 最小二乘回归模型的Hessian矩阵：

$$\mathbf{H} = \frac{1}{n}\sum_{i=1}^{n} \mathbf{x}_i \mathbf{x}_i^T = \begin{bmatrix} 9.1 & 17.2 \\ 17.2 & 33.1 \end{bmatrix}.$$

- Condition number: $\frac{\lambda_{\max}(\mathbf{H})}{\lambda_{\min}(\mathbf{H})} = 281.7.$ (相较于 92K.)

一维 数据的特征标准化

假设样本 $x_1, \cdots, x_n$ 是 一维的.

- **Min-max normalization**: $x_i' = \frac{x_i - \min(x_i)}{\max(x_i) - \min(x_i)}$.
- After the scaling, the samples $x_1', \cdots, x_n'$ are in $[0, 1]$.

一维 数据的特征标准化

Assume the samples $x_1, \cdots, x_n$ are one-dimensional.

- 标准化： $x_i' = \frac{x_i - \hat{\mu}}{\hat{\sigma}}$.
  - $\hat{\mu} = \frac{1}{n} \sum_{i=1}^{n} x_i$ 是样本平均值
  - $\hat{\sigma}^2 = \frac{1}{n-1} \sum_{i=1}^{n} (x_i - \hat{\mu})^2$ 是样本方差
  - After the scaling, the samples $x_1', \cdots, x_n'$ 均值为零，方差为 1 .

高维 数据的特征标准化

- 对每个特征独立进行特征缩放
  - 例如，在缩放‘身高’特征时，忽略‘收入’特征

```
# Min-Max Normalization
import numpy

d = x.shape[1]
xmin = numpy.min(x, axis=0).reshape(1, d)
xmax = numpy.max(x, axis=0).reshape(1, d)
xnew = (x - xmin) / (xmax - xmin)
```

高维 数据的特征标准化

- 对每个特征独立进行特征缩放
  - 例如，在缩放‘身高’特征时，忽略‘收入’特征

```python
# Standardization
import numpy

d = x.shape[1]
mu = numpy.mean(x, axis=0).reshape(1, d)
sig = numpy.std(x, axis=0).reshape(1, d)
xnew = (x - mu) / sig
```

批归一化

批量归一化：隐藏层的标准化

- let $\mathbf{x}^{(k)} \in \mathbb{R}^d$ be the output of the k-th hidden layer.
- $\widehat{\boldsymbol{\mu}} \in \mathbb{R}^d$: sample mean of $\mathbf{x}^{(k)}$ evaluated on a batch of samples.
- $\widehat{\boldsymbol{\sigma}} \in \mathbb{R}^d$: sample std of $\mathbf{x}^{(k)}$ evaluated on a batch of samples.

- Standardization: $z_j^{(k)} = \frac{x_j^{(k)} - \widehat{\mu}_j}{\widehat{\sigma}_j + 0.001}$, for $j = 1, \cdots, d$.

Batch Normalization: Standardization of Hidden Layers

- Let $\mathbf{x}^{(k)} \in \mathbb{R}^d$ be the output of the k-th hidden layer.
- $\hat{\boldsymbol{\mu}} \in \mathbb{R}^d$: sample mean of $\mathbf{x}^{(k)}$ evaluated on a batch of samples.
- $\hat{\boldsymbol{\sigma}} \in \mathbb{R}^d$: sample std of $\mathbf{x}^{(k)}$ evaluated on a batch of samples.
- $\boldsymbol{\gamma} \in \mathbb{R}^d$: scaling parameter (trainable).
- $\boldsymbol{\beta} \in \mathbb{R}^d$: shifting parameter (trainable).
- Standardization: $z_j^{(k)} = \frac{x_j^{(k)} - \hat{\mu}_j}{\hat{\sigma}_j + 0.001}$, for $j = 1, \cdots, d$.
- Scale and shift: $x_j^{(k+1)} = z_j^{(k)} \cdot \gamma_j + \beta_j$, for $j = 1, \cdots, d$.

Batch Normalization: Standardization of Hidden Layers

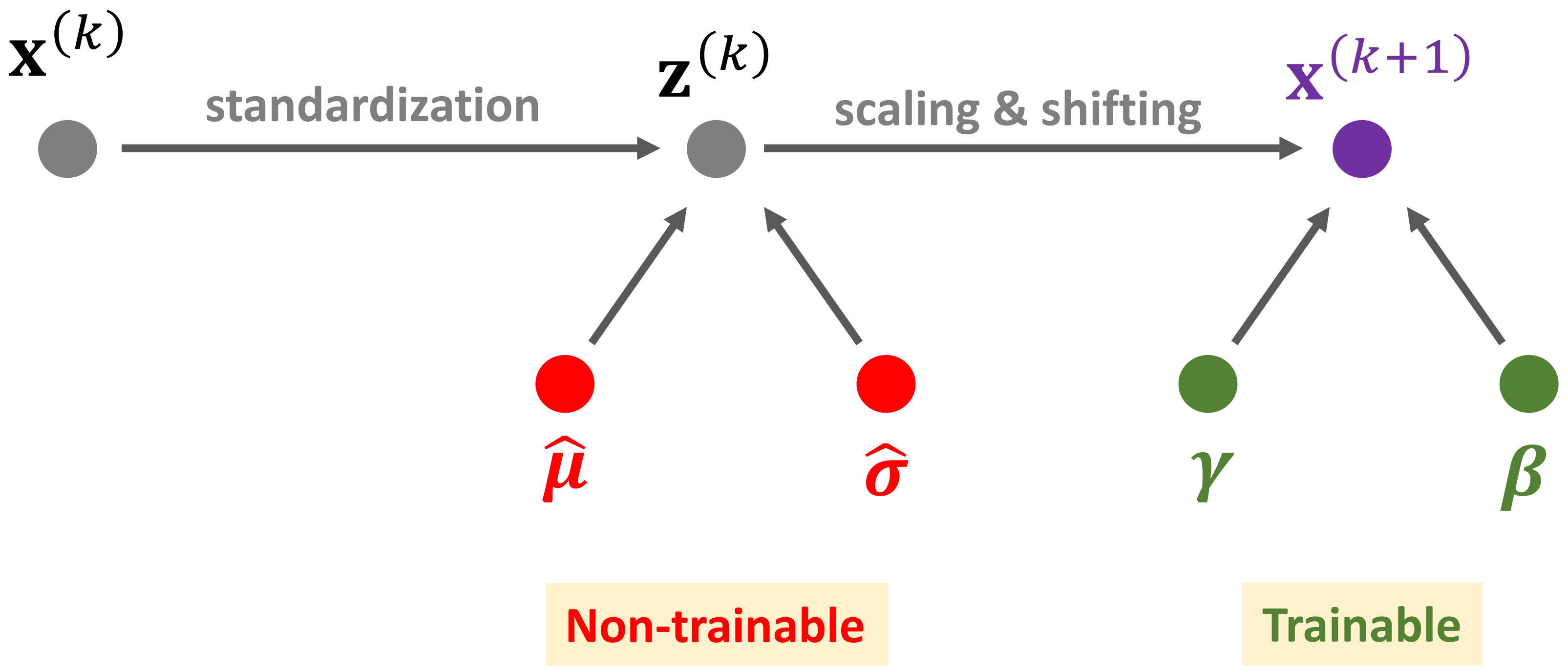
- Let $\mathbf{x}^{(k)} \in \mathbb{R}^d$ be the output of the k-th hidden layer.
- $\hat{\boldsymbol{\mu}} \in \mathbb{R}^d$:
- $\hat{\boldsymbol{\sigma}} \in \mathbb{R}^d$:

Non-trainable. Just record them in the forward pass; use them in the backpropagation.

- $\boldsymbol{\gamma} \in \mathbb{R}^d$: scaling parameter (trainable).
- $\boldsymbol{\beta} \in \mathbb{R}^d$: shifting parameter (trainable).
- Standardization: $z_j^{(k)} = \frac{x_j^{(k)} - \hat{\mu}_j}{\hat{\sigma}_j + 0.001}$, for $j = 1, \cdots, d$.
- Scale and shift: $x_j^{(k+1)} = z_j^{(k)} \cdot \gamma_j + \beta_j$, for $j = 1, \cdots, d$.

Backpropagation for Batch Normalization Layer

- Standardization: $z_j^{(k)} = \frac{x_j^{(k)} - \hat{\mu}_j}{\hat{\sigma}_j + 0.001}$, for $j = 1, \cdots, d$.
- Scale and shift: $x_j^{(k+1)} = z_j^{(k)} \circ \gamma_j + \beta_j$, for $j = 1, \cdots, d$.



Backpropagation for Batch Normalization Layer

- Standardization: $z_j^{(k)} = \frac{x_j^{(k)} - \hat{\mu}_j}{\hat{\sigma}_j + 0.001}$, for $j = 1, \cdots, d$.
- Scale and shift: $x_j^{(k+1)} = z_j^{(k)} \circ \gamma_j + \beta_j$, for $j = 1, \cdots, d$.

We know $\frac{\partial L}{\partial x_j^{(k+1)}}$ from the backpropagation (from the top to $x^{(k+1)}$.)

- Use $\frac{\partial L}{\partial \gamma_j} = \frac{\partial L}{\partial x_j^{(k+1)}} \frac{\partial x_j^{(k+1)}}{\partial \gamma_j} = \frac{\partial L}{\partial x_j^{(k+1)}} z_j^{(k)}$ to update γ_j;
- Use $\frac{\partial L}{\partial \beta_j} = \frac{\partial L}{\partial x_j^{(k+1)}} \frac{\partial x_j^{(k+1)}}{\partial \beta_j} = \frac{\partial L}{\partial x_j^{(k+1)}}$ to update β_j.

Backpropagation for Batch Normalization Layer

- Standardization: $z_j^{(k)} = \frac{\boxed{x_j^{(k)}} - \hat{\mu}_j}{\hat{\sigma}_j + 0.001}$, for $j = 1, \cdots, d$.
- Scale and shift: $x_j^{(k+1)} = z_j^{(k)} \circ \gamma_j + \beta_j$, for $j = 1, \cdots, d$.

We know $\frac{\partial L}{\partial x_j^{(k+1)}}$ from the backpropagation (from the top to $x^{(k+1)}$.)

Compute $\boxed{\frac{\partial L}{\partial z_j^{(k)}}} = \frac{\partial L}{\partial x_j^{(k+1)}} \frac{\partial x_j^{(k+1)}}{\partial z_j^{(k)}} = \frac{\partial L}{\partial x^{(k+1)}} \gamma.$

Compute $\frac{\partial L}{\boxed{\partial x_j^{(k)}}} = \boxed{\frac{\partial L}{\partial z_j^{(k)}}} \frac{\partial z_j^{(k)}}{\partial x_j^{(k)}} = \boxed{\frac{\partial L}{\partial z_j^{(k)}}} \frac{1}{\hat{\sigma}_j + 0.001}$ and pass it to the lower layers.

Batch Normalization Layer in Keras

Batch Normalization Layer

- Let $\mathbf{x}^{(k)} \in \mathbb{R}^{d}$ 是 k 维隐藏层的输出.
- $\hat{\boldsymbol{\mu}}$, $\hat{\boldsymbol{\sigma}} \in \mathbb{R}^{d}$: 不可训练的 参数.
- $\boldsymbol{\gamma}$, $\boldsymbol{\beta} \in \mathbb{R}^{d}$: 可训练的 参数.
- Standardization: $z_j^{(k)} = \frac{x_j^{(k)} - \hat{\mu}_j}{\hat{\sigma}_j + 0.001}$, for $j = 1, \cdots, d$.
- Scale and shift: $x_j^{(k+1)} = z_j^{(k)} \cdot \gamma_j + \beta_j$, for $j = 1, \cdots, d$.

Batch Normalization Layer

- Let $\mathbf{x}^{(k)} \in \mathbb{R}^{d}$ 是 k 维隐藏层的输出.
- $\hat{\boldsymbol{\mu}}$, $\hat{\boldsymbol{\sigma}} \in \mathbb{R}^{d}$: 不可训练的 参数.
- $\boldsymbol{\gamma}$, $\boldsymbol{\beta} \in \mathbb{R}^{d}$: 可训练的 参数.

困难：有 $4d$ 个参数存储在内存中. d 可能会非常大!

例如:

- The 1st Conv Layer in VGG16 Net outputs a $150 \times 150 \times 64$ tensor.
- The number of parameters in a single Batch Normalization Layer would be $4d = 1.44\mathrm{M}$.

Batch Normalization Layer

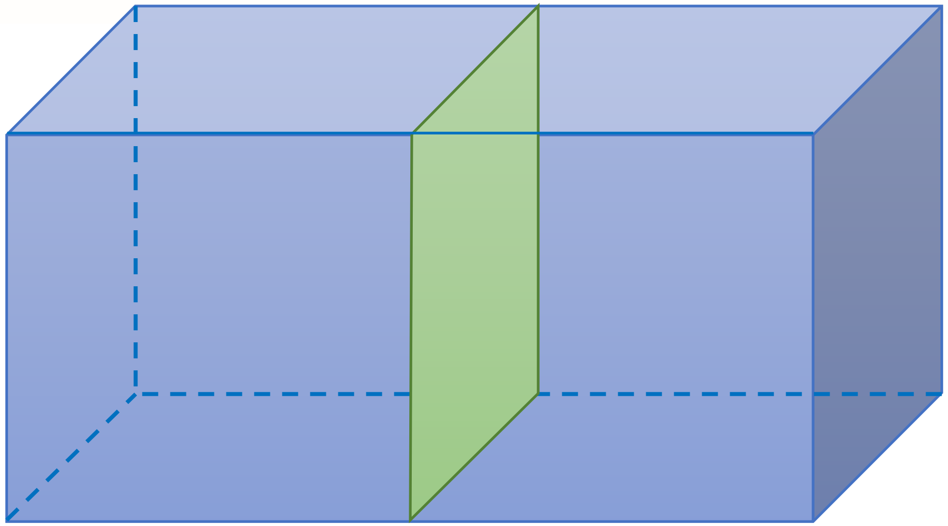
Solution:

- Make the 4 parameters $1 \times 1 \times 64$, instead of $150 \times 150 \times 64$.

Batch Normalization Layer

Solution:

- Make the 4 parameters $1 \times 1 \times 64$, instead of $150 \times 150 \times 64$.
- How?
- A scalar parameter for a slice (e.g., a 150×150 matrix) of the tensor.
- Of course, you can make the parameters $150 \times 1 \times 1$ or $1 \times 150 \times 1$.



CNN for Digit Recognition

```
from keras import models
from keras import layers

model = models.Sequential()
model.add(layers.Conv2D(10, (5, 5), input_shape=(28, 28, 1)))
model.add(layers.BatchNormalization())
model.add(layers.Activation('relu'))
model.add(layers.MaxPooling2D((2, 2)))
model.add(layers.Conv2D(20, (5, 5)))
model.add(layers.BatchNormalization())
model.add(layers.Activation('relu'))
model.add(layers.MaxPooling2D((2, 2)))
model.add(layers.Flatten())
model.add(layers.Dense(100))
model.add(layers.BatchNormalization())
model.add(layers.Activation('relu'))
model.add(layers.Dense(10, activation='softmax'))
```

CNN for Digit Recognition

```
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
conv2d_1 (Conv2D)            (None, 24, 24, 10)        260
_________________________________________________________________
batch_normalization_1 (Batch (None, 24, 24, 10)        40
_________________________________________________________________
activation_1 (Activation)    (None, 24, 24, 10)        0
_________________________________________________________________
max_pooling2d_1 (MaxPooling2 (None, 12, 12, 10)        0
_________________________________________________________________
conv2d_2 (Conv2D)            (None, 8, 8, 20)          5020
_________________________________________________________________
batch_normalization_2 (Batch (None, 8, 8, 20)          80
_________________________________________________________________
activation_2 (Activation)    (None, 8, 8, 20)          0
_________________________________________________________________
max_pooling2d_2 (MaxPooling2 (None, 4, 4, 20)          0
_________________________________________________________________
flatten_1 (Flatten)          (None, 320)               0
_________________________________________________________________
dense_1 (Dense)              (None, 100)               32100
_________________________________________________________________
batch_normalization_3 (Batch (None, 100)               400
_________________________________________________________________
activation_3 (Activation)    (None, 100)               0
_________________________________________________________________
dense_2 (Dense)              (None, 10)                1010
=================================================================
Total params: 38,910
Trainable params: 38,650
Non-trainable params: 260
```

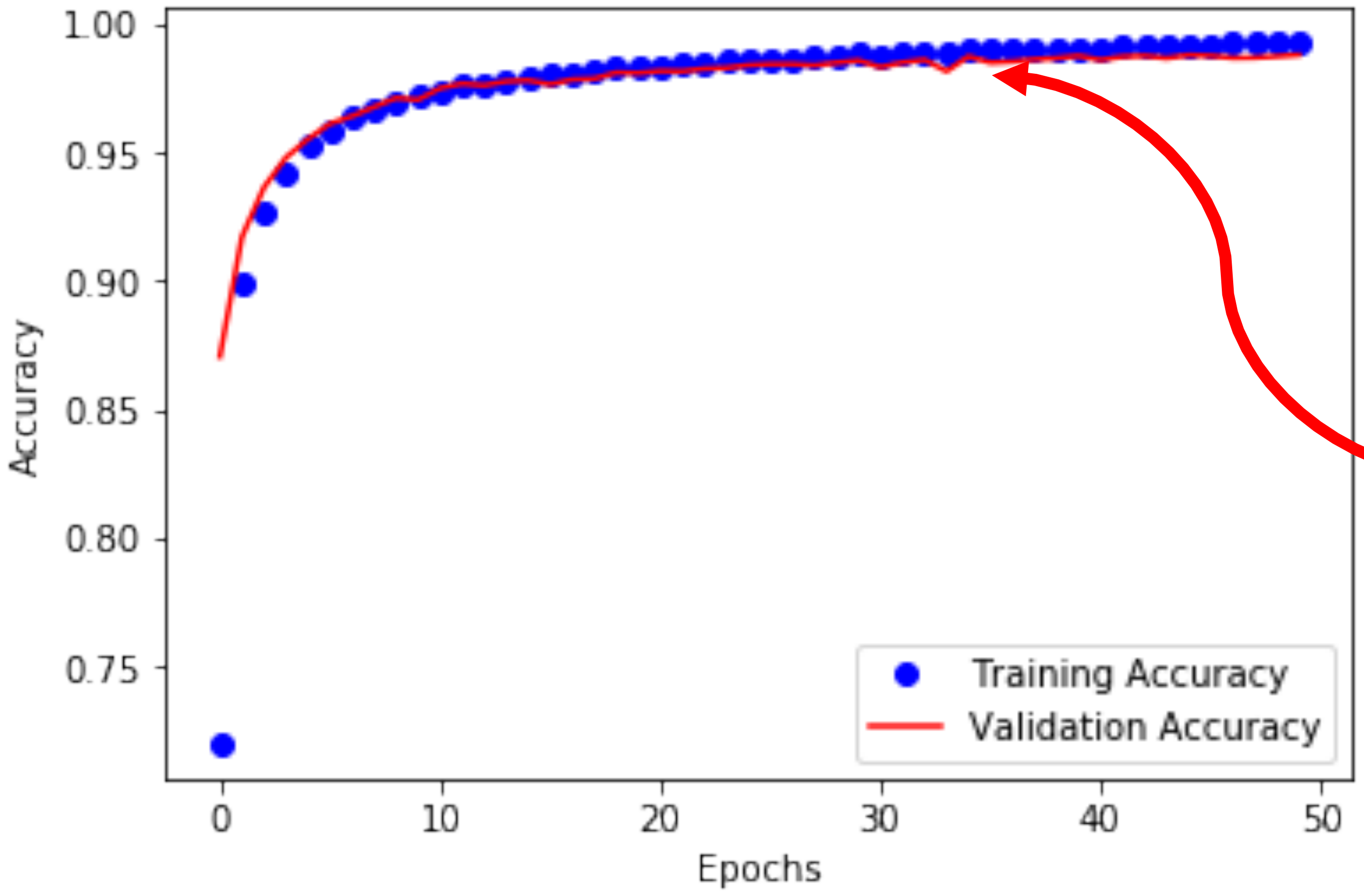
CNN for Digit Recognition

Train the model (**with** Batch Normalization) on MNIST ($n = 50{,}000$).

```
Train on 50000 samples, validate on 10000 samples
Epoch 1/3
50000/50000 [==============================] - 29s 580us/step -
loss: 0.1599 - acc: 0.9595 - val_loss: 0.1165 - val_acc: 0.9644
Epoch 2/3
50000/50000 [==============================] - 26s 516us/step -
loss: 0.0468 - acc: 0.9858 - val_loss: 0.0562 - val_acc: 0.9822
Epoch 3/3
50000/50000 [==============================] - 25s 508us/step -
loss: 0.0325 - acc: 0.9902 - val_loss: 0.0494 - val_acc: 0.9832
```

CNN for Digit Recognition

Train the model (**without** Batch Normalization) on MNIST ($n = 50{,}000$).



Without Batch Normalization, it takes 10x more epochs to converge.

总结

特征变化

- 使所有特征的尺度具有可比性
- 为什么？更优的Hessian矩阵条件数 → 更快收敛

特征变化

- 使所有特征的尺度具有可比性
- 为什么？更优的Hessian矩阵条件数 → 更快收敛
  - 方法：
  - \- 最小-最大归一化：将特征缩放到 [0, 1] 区间。
  - \- 标准化：使每个特征具有零均值和单位方差。

批归一化

- 隐藏层的特征标准化。
- 为什么？加速收敛。
- 2个可训练参数：平移和缩放。
- 2个不可训练参数：均值和方差。

批归一化

- -隐藏层的特征标准化。
- 为什么？加速收敛。
- 2个可训练参数：平移和缩放。
- 2个不可训练参数：均值和方差。
- Keras 提供了 layers.BatchNormalization()。
- 将 BN 层放在卷积层（Conv）之后，激活函数（Activation）之前。