Course Notes: Deep Learning for Visual Computing

Peter Wonka

August 30, 2021

Contents

1 Input Data Normalization

1.1 Why Input Data Normalization?

- Input can be given in different **formats**
  - e.g. images can have values in the range $[0, 255]$ or values in the range $[0, 1]$
- **Different features** can naturally have **different scales**
  - e.g. age in $[0 - 150]$ and height in $[0 - 3]$
- Data being **0 centered** may improve training
  - depends on the specifics of the architecture and training algorithm

1.2 Data matrix review

- We want to store n **samples** (e.g. images) in a data matrix $\boldsymbol{X}$
  - We have n samples $\{\mathbf{x}_i\}_{i=1}^n$
  - **Dimension**: represents a different **variable** (or **feature**)
  - Each sample $\mathbf{x}_i$ has d dimensions, i.e., is a d-dimensional vector $\mathbf{x}_i \in \mathbb{R}^d$
  - $\boldsymbol{X} \in \mathbb{R}^{n \times d}$: each **row** of the data matrix represents a single sample (e.g. image)

$$\boldsymbol{X} = \begin{bmatrix} — & \mathbf{x}_1^T & — \\ — & \mathbf{x}_2^T & — \\ & \vdots & \\ — & \mathbf{x}_n^T & — \end{bmatrix} \in \mathbb{R}^{n \times d} \tag{1.1}$$

1.3 Example matrices

- **Note**: We discuss input data normalization for the case of vectorized data first, then comment how to adapt to higher order tensors
- **Single RGB Image.** Each row of the data matrix = 1 pixel, represented by its 3 RGB values; n =number of pixels, $d = 3$
- **Image Collection.** A collection of images of the same size. Each row (column) of the data matrix = one (vectorized) image. For imagenet, each row of the data matrix would have $224 \times 224 \times 3$ entries and the matrix would have as many rows as there are images
- **Video Collection**

1.4 Limitations

- We ignore all technicalities such as biased and unbiased estimators, known and unknown distributions, ...
  - e.g. difference between μ, $\hat{\mu}$, $\bar{x}$

1.5 Compute Values for Normalization

- Basic normalization: estimate **sample mean** and **variance** for each of the d columns
- Advanced normalization: estimate the sample **covariance** or **correlation** for any two of the d columns
- **Sample means** (column average/mean):

$$\bar{\mathbf{x}} = \begin{pmatrix} \frac{1}{n}\sum_{i=1}^{n} \boldsymbol{X}_{i1} \\ \cdots \\ \frac{1}{n}\sum_{i=1}^{n} \boldsymbol{X}_{id} \end{pmatrix} := \begin{pmatrix} \bar{x}_1 \\ \cdots \\ \bar{x}_d \end{pmatrix} \in \mathbb{R}^d \tag{1.2}$$

- The joint variability of any two variables/features/columns is given by the sample **covariance matrix** $\in \mathbb{R}^{d \times d}$

- The **sample covariance** between any two columns (p, q) is computed as:

$$c_{pq} = \frac{1}{n} \sum_{i=1}^{n} \left(\boldsymbol{X}_{ip} - \bar{x}_p \right) \left(\boldsymbol{X}_{iq} - \bar{x}_q \right), \forall p, q = 1, \cdots, d \tag{1.3}$$

- Note, when $p = q$, the covariance is the same as the variance.
- The vector of standard deviations is computed as $\mathbf{s} = (\sqrt{c_{11}}, \sqrt{c_{22}}, \ldots, \sqrt{c_{dd}})$
- We can collect the covariances for any two of the columns into a matrix, which is called the **covariance matrix** (see wiki):

$$\boldsymbol{C} = \begin{pmatrix} c_{11} & \ldots & c_{1d} \\ \vdots & \ddots & \vdots \\ c_{d1} & \ldots & c_{dd} \end{pmatrix} \in \mathbb{R}^{d \times d} \tag{1.4}$$

1.6 Normalization

- Literature: wikipedia
- We consider samples $\mathbf{x}_i$ with sample mean $\bar{\mathbf{x}}$ and sample standard deviation $\mathbf{s}$
- **Note:** $\mathbf{x}_i$ is a vector representing the i-th sample (not the i-th dimension)
  - Each $\mathbf{x}_i$ is a vector $\in \mathbb{R}^n$

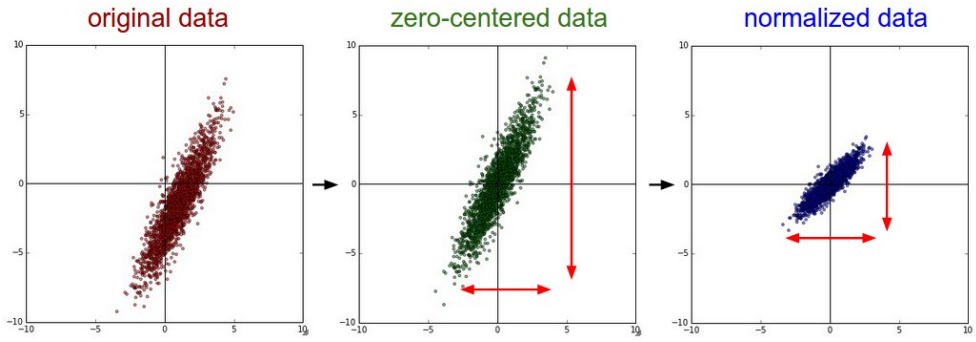
- **z-score normalization / standardization**:

$$\mathbf{x}_i^{\text{normalized}} = \frac{\mathbf{x}_i - \bar{\mathbf{x}}}{\mathbf{s}}$$

  - the division denotes a component wise division!
  - normalized data centered around 0
  - most values will be between $[-1, 1]$, but minimum and maximum can vary depending on the input.

```
X = np.random.randn(4,2)

# z-score normalization
mean = np.mean(X, axis = 0)
std = np.std(X, axis = 0)
Xnew = (X - mean) / std
```


original data
zero-centered data
normalized data

- **min-max normalization**:

$$\mathbf{x}_i^{\text{normalized}} = \frac{\mathbf{x}_i - \mathbf{x}_{\text{min}}}{\mathbf{x}_{\text{max}} - \mathbf{x}_{\text{min}}}$$

  - the division denotes a component wise division!
  - where $\mathbf{x}_{\text{min}}$ is the d dimensional vector of minima per column
  - $\mathbf{x}_{\text{max}}$ is the vector of maxima per column
  - all output values are in the range $[0, 1]$

- **general min-max normalization**:

$$\mathbf{x}_i^{\text{normalized}} = \mathbf{a} + \frac{(\mathbf{x}_i - \mathbf{x}_{\text{min}})(\mathbf{b} - \mathbf{a})}{\mathbf{x}_{\text{max}} - \mathbf{x}_{\text{min}}}$$

  - scaling such that the minimum values of the ranges are given by vector $\mathbf{a}$, and maximum values of the ranges by vector $\mathbf{b}$.
  - i-th feature will be in the range $[a_i, b_i]$

- **Maximum Absolute Value Normalization**:

$$\mathbf{x}_i^{\text{normalized}} = \frac{\mathbf{x}_i}{|\mathbf{x}_{\text{max}}|}$$

  - scale each feature by its maximum absolute value
  - the absolute values of the output are mapped in the range $[0, 1]$

- **Robust Normalization**

$$\mathbf{x}_i^{\text{normalized}} = \frac{\mathbf{x}_i - \mathbf{x}_{0.5}}{\mathbf{x}_{q_2} - \mathbf{x}_{q_1}}$$

  - where $\mathbf{x}_q$ is the d dimensional vector of q-quantile per column, e.g., $\mathbf{x}_{0.5}$ is the median (per column)
  - by default, take $q_1 = 0.25$ and $q_2 = 0.75$
  - Scale features using statistics (quantiles) that are robust to outliers.
  - removes the median and scales the data according to the quantile range.

1.7 Normalization on Images

- Assume images are $\in \mathbb{R}^{w \times h \times 3}$
- Idea 1: Compute a mean image $\in \mathbb{R}^{w \times h \times 3}$ and subtract it from each image
- Idea 2: Compute a mean pixel color $\in \mathbb{R}^3$ and subtract it from each pixel
- Idea 3: z-score normalization per pixel (use a pixel mean $\in \mathbb{R}^3$ and pixel std $\in \mathbb{R}^3$)

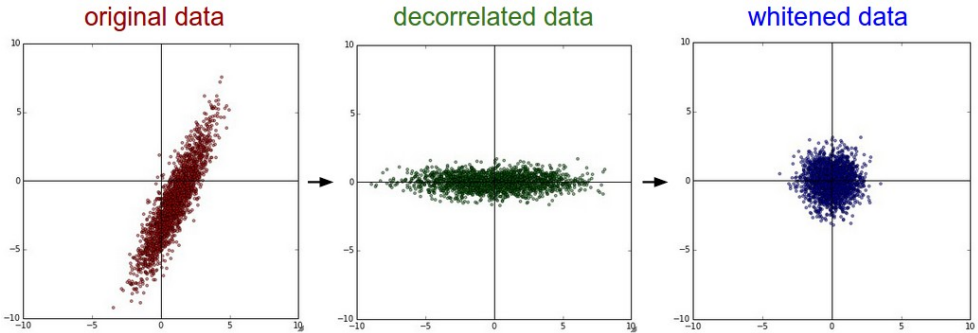
```
normalize = transforms.Normalize(mean=[0.485, 0.456, 0.406],
                                  std=[0.229, 0.224, 0.225])
```

- Other Ideas:
  - normalize to $[0, 1]$
  - compute a complete mean and std image
  - normalize to $[0, 1]$ and use a normalization layer as first layer

1.8 Discussion

- Initial data normalization interacts with other normalization in the network, e.g. batch normalization
- Any preprocessing statistics (e.g. the data mean) must only be computed on the **training data**
- The values estimated from the training data can then be applied to the **validation** and **test data**
- Mistake 1: Use train + test + validation set to compute statistics
- Mistake 2: Only normalize the training data
- Mistake 3: Use different normalization as was used during training

1.9 Advanced Normalization



- PCA
  - PCA computes an eigenvalue, eigenvector decomposition of the covariance matrix.
  - Use PCA to rotate / decorrelate the data
  - Use PCA to do dimension reduction as preprocess

- Whitening
  - In addition to decorrelating the data, scale by the eigenvalues
- Not common in Deep Learning
  - expensive for large data sets