code

(https://github.com/jermwatt/control-notes)

share

(https://www.linkedin.com/cws/share?url=https%3A%2F%2Fjermwatt.github.io%2Fcontrol-notes%2Fposts%2FZCA_Sphereing%2FZCA_Sphereing.html)

(https://twitter.com/intent/tweet?ref_src=twsrc%5Etfw&tw_p=tweetbutton&url=https%3A%2F%2Fjermwatt.github.io%2Fcontrol-notes%2Fposts%2FZCA_Sphereing%2FZCA_Sphereing.html)

(https://www.facebook.com/sharer/sharer.php?u=https%3A%2F%2Fjermwatt.github.io%2Fcontrol-notes%2Fposts%2FZCA_Sphereing%2FZCA_Sphereing.html)

(https://www.reddit.com/submit?url=https%3A%2F%2Fjermwatt.github.io%2Fcontrol-notes%2Fposts%2FZCA_Sphereing%2FZCA_Sphereing

★ Our Project

On GitHub

(https://github.com/jermwatt/control-notes)

An introduction to ZCA-sphering

In this short post we describe *Zero Phase Component Analysis Sphering* - or *ZCA sphering* for short. ZCA sphering is a popular *input-normalization* technique akin standard normalization (https://jermwatt.github.io/machine_learning_refined/notes/9_Feature_engineer_select/9_3_Scaling.html) and PCA-sphering (https://jermwatt.github.io/machine_learning_refined/notes/9_Feature_engineer_select/9_5_PCA_sphereing.html), but made especially for image, video, and other naturally ordered data types (here we will focus on its application to image-based data. As with these other input-normalization schemes, when properly applied ZCA-sphering conditions data in such a way as to substantially accelerate the training of supervised and unsupervised learners (and convolutional networks in general). Note: this post assumes

- basic familiarity with edge-based feature extractors (like e.g., convolutional networks or even basic edge-based histogram features (https://jermwatt.github.io/machine_learning_refined/notes/9_Feature_engineer_select/9_2_Histogram.html)) - i.e., that you understand the importance of extracting edge information from images for un/supervised learning tasks involving image data
- familiarity with standard normalization (https://jermwatt.github.io/machine_learning_refined/notes/9_Feature_engineer_select/9_3_Scaling.html) and PCA-sphering (https://jermwatt.github.io/machine_learning_refined/notes/9_Feature_engineer_select/9_5_PCA_sphereing.html) techniques

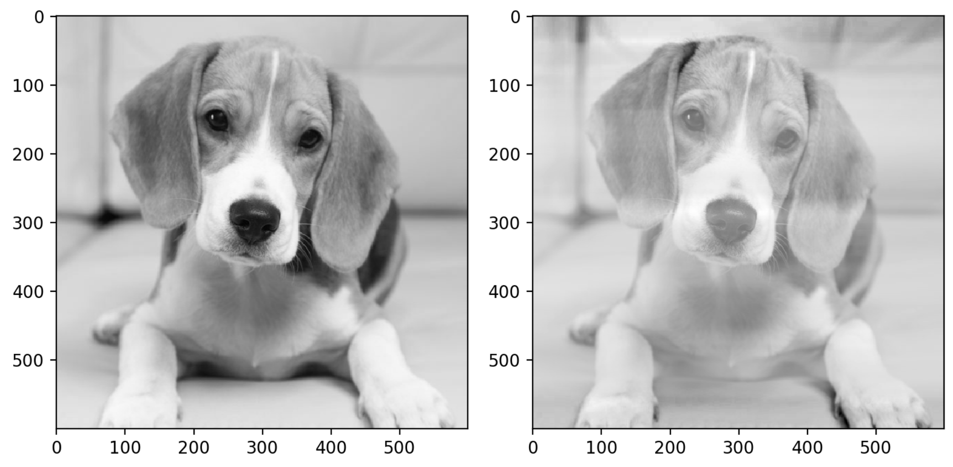
You can skip around this document to particular subsections via the hyperlinks below.

- Contrast normalization
- Introducing unwanted edges with typical input normalizers
- What went wrong with PCA-sphering?
- ZCA-sphering
- Python implementation

Contrast normalization

Before discussing ZCA-sphering lets talk about *contrast normalization*. Contrast normalization is a standard pre-processing technique applied to almost all image-based datasets which simply involves *standard normalizing (https://jermwatt.github.io/machine_learning_refined/notes/9_Feature_engineer_select/9_3_Scaling.html) each image in a dataset*, that is subtracting its mean (pixel) value and dividing off its standard deviation. This helps adjust for *contrast differences* in the various images in a dataset and virtually always done prior to any *feature-wise normalization technique* like ZCA sphering.

Why normalize the image contrast first? Because we almost never want an image-based learner to be sensitive to contrast, since this would make natural image based learning tasks like e.g., object detection much more difficult to solve (see e.g., the simple example described in Figure 4.26 (https://github.com/jermwatt/machine_learning_refined/blob/gh-pages/sample_chapters/1st_ed/chapter_4.pdf) of the first edition of our textbook). In other words, contrast normalization helps our learners treat all images equally by normalizing their average intensity. Below we show an example image on the left and its contrast normalized version on the right.

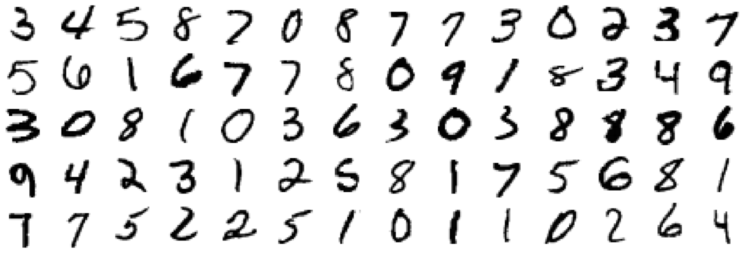


```
<matplotlib.image.AxesImage at 0x1a1f7948d0>
```

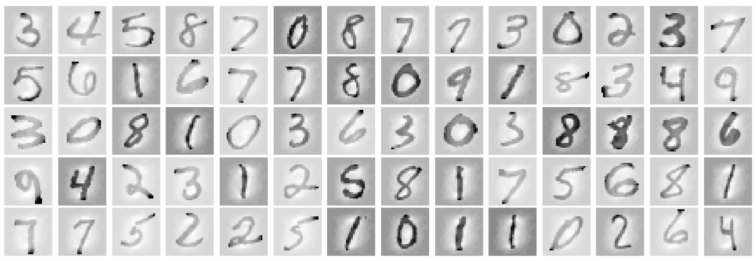
Introducing unwanted edges with typical input normalizers

In this set of notes (https://jermwatt.github.io/machine_learning_refined/notes/9_Feature_engineer_select/9_3_Scaling.html) you can see how the simple idea of *standard normalizing* the input of a machine learning dataset could drastically improve the learning ability of a zero or first order local optimization technique by improving the contours of any associated cost function. The natural evolution of this idea - PCA sphering (https://jermwatt.github.io/machine_learning_refined/notes/9_Feature_engineer_select/9_5_PCA_sphereing.html) - can help even further improve the nature of a cost function, allowing for even easier learning.

While there is nothing preventing us from applying standard normalization or PCA-based sphering to image-based data, doing so *destroys the natural spatial-correlation of image-based data* and thus much of the *edge information* in an image. To get a sense of this we show the result of each input-normalization procedure applied to a large set of handwritten digits from the MNIST dataset - a random subset of which are shown below.

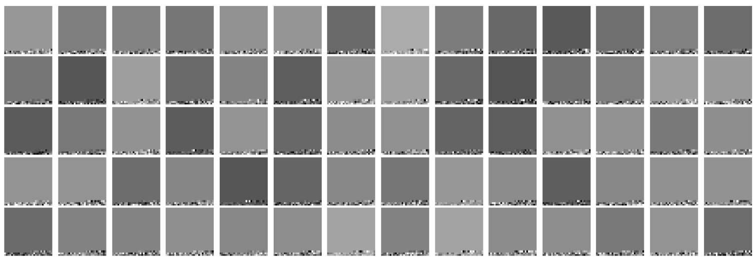


Below we show the result of *standard normalizing* (https://jermwatt.github.io/machine_learning_refined/notes/9_Feature_engineer_select/9_3_Scaling.html) a batch of these hand written digits - that is we normalize each feature (here each individual pixel) by mean centering and scaling by the standard deviation of all such pixels in the dataset. Notice that while each digit is still identifiable, doing this *introduces* edge-based artifacts into these images which can make edge-based feature extractors (like a convolutional network) much less effective.



Below we show the result of going one step further and *PCA-sphering* (https://jermwatt.github.io/machine_learning_refined/notes/9_Feature_engineer_select/9_5_PCA_sphereing.html) this handwritten digit dataset, and plot a random subset of the resulting PCA-sphered data. Note: these are the same images (transformed by the sphering process) we showed above.

Here we can see that PCA-sphering completely destroys the spatial structure of the original images - none of the original numbers are visible in these transformed versions. There is little point to using edge-based feature extractors (like a convolutional network) if there are no longer any (useful) edges in our images.



What went wrong with PCA-sphering?

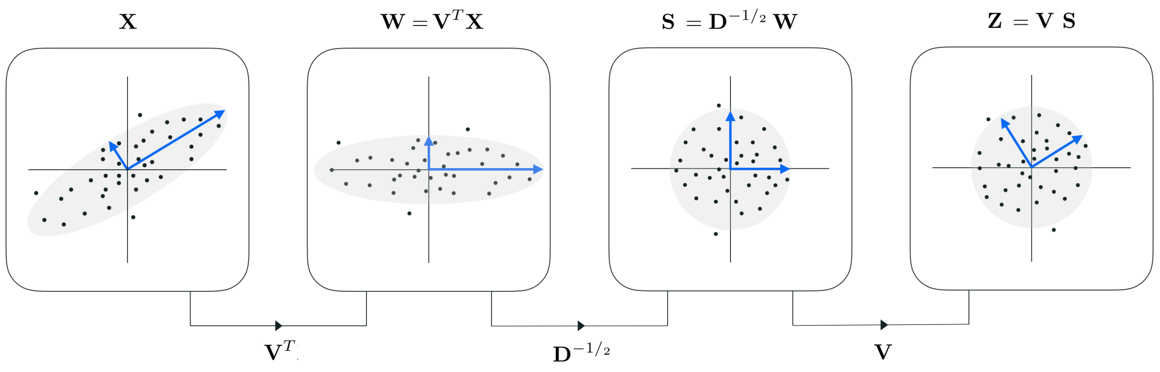
As detailed in the previous Example, in terms of the actions we perform on the data itself with PCA-sphering (or whitening) we:

- **rotate and reflect** by multiplying the data by $\mathbf{V}^T$ (where $\mathbf{V}$ is the set of eigenvectors of the data's covariance matrix) the data so that its largest orthogonal directions of variance align with the coordinate axes (this is done by the standard PCA transformation)
- **normalize** these coordinate axes by dividing by their individual standard deviations by multiplying the rotated/reflected data by $\mathbf{D}^{-1/2}$, the diagonal matrix of inverted square roots of the eigenvalues of the data's covariance matrix (this extra bit of normalization added to the PCA transformation to makes it the PCA-sphering operation)

Both of these actions contribute to the destruction of spatial correlation - but it is the **rotation** / **reflection** component that is by far the greater culprit. We can get a visceral sense of this fact with our current dataset by simply applying the standard PCA transform - which will rotate / reflect the space so that the largest directions of variance coincide with the coordinate axes - without normalizing the result (which led to the PCA-sphering result above).

ZCA-sphering

The rotation / reflection from the PCA transformation utterly destroys the spatial correlation structure of our images, a phenomenon that is true more generally speaking for other image datasets as well. What can we do if we would still like the added optimization-boosting benefit of PCA-sphering normalization but do not want to destroy the spatial correlation of our input data by rotating / reflecting it? Well if indeed the rotation / reflection is producing the greatest challenge in terms of maintaining the spatial correlation of our data, why don't we simply rotate / reflect our dataset back to its original orientation after we finish sphering it (i.e.,normalizing it along its largest orthogonal directions of variance)? Since we know that it multiplication by $\mathbf{V}^T$ that produces the original rotation / reflection, multiplying by PCA-sphered data by $\left(\mathbf{V}^T\right)^{-1} = \mathbf{V}$ (where the equality follows from the fact that $\mathbf{V}$ is an orthogonal matrix) will return the sphered data to its original orientation in the space. This is illustrated in the Figure below.



$\mathbf{X}$
$\mathbf{W} = \mathbf{V}^T\mathbf{X}$
$\mathbf{S} = \mathbf{D}^{-1/2}\mathbf{W}$
$\mathbf{Z} = \mathbf{V}\ \mathbf{S}$
$\mathbf{V}^T$
$\mathbf{D}^{-1/2}$
$\mathbf{V}$

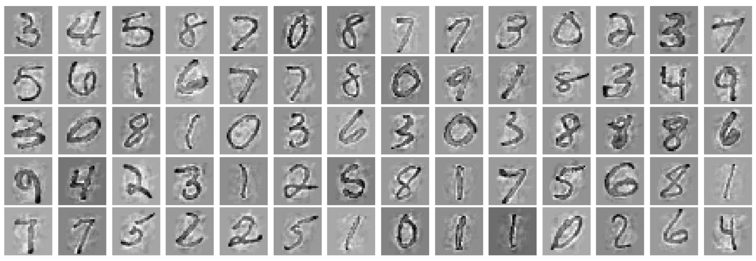
Figure 2: *ZCA-sphering illustrated.*

Since our PCA-sphered transformation of the input data took the form $\mathbf{S} = \mathbf{D}^{-1/2}\mathbf{V}^T\mathbf{X}$ this re-rotation/reflection gives us the related formula

$$\text{(ZCA-sphered data)} \qquad \mathbf{Z} = \mathbf{V}\mathbf{S} = \mathbf{V}\mathbf{D}^{-1/2}\mathbf{V}^T\mathbf{X}\,. \qquad (1)$$

For historical reasons this re-rotated version of our PCA-sphered data is often referred to as *Zero-phase Component Analysis (ZCA) sphering*.

Below we show the result of ZCA-sphering our dataset. What a difference! Indeed it was the rotation/reflection of PCA that destroyed most of the spatial correlation, since these ZCA-sphered images retain much of the spatial correlation present in their original versions. Now we have the best of both worlds: a global normalization scheme (that will help speed up training) that retains the spatial structure of data leveraged e.g., by edge detectors and convolution operations.



Just for kicks we show the same ZCA-sphered data below using a red-yellow-blue colormap.

Python implementation

Here we provide a simple implementation in `Python`. We begin with a short `PCA` function for computing the principle components of an input data matrix `x`. The interested reader can see this set of notes on PCA (https://jermwatt.github.io/machine_learning_refined/notes/8_Linear_unsupervised_learning/8_5_Classic.html) for further details about this implementation.

```
# sphereing pre-processing functionality
def PCA(x,**kwargs):
    # regularization parameter for numerical
stability
    lam = 10**(-7)
    if 'lam' in kwargs:
        lam = kwargs['lam']

    # create the correlation matrix
    P = float(x.shape[1])
    Cov = 1/P*np.dot(x,x.T) + lam*np.eye(x.shape
[0])

    # use numpy function to compute eigenvalu
es / vectors of correlation matrix
    d,V = np.linalg.eigh(Cov)
    return d,V
```

Our `ZCA_spherer` below employs our `PCA` function in producing the required principal components.

```
# ZCA spherer
def ZCA_spherer(x,**kwargs):
    # Step 1: mean-center the data
    x_means = np.mean(x,axis = 1)[:,np.newaxis]
    x_centered = x - x_means

    # Step 2: compute pca transform on mean-c
entered data
    d,V = PCA(x_centered,**kwargs)

    # Step 3: divide off standard deviation o
f each (transformed) input,
    # which are equal to the returned eigenva
lues in 'd'.
    stds = (d[:,np.newaxis])**(0.5)

    # check to make sure thta x_stds > small
threshold, for those not
    # divide by 1 instead of original standar
d deviation
    ind = np.argwhere(stds < 10**(-5))
    if len(ind) > 0:
        ind = [v[0] for v in ind]
        adjust = np.zeros((stds.shape))
        adjust[ind] = 1.0
        stds += adjust

    # create normalizer / inverse-normalizer
    return np.dot(V,np.dot(V.T,x_centered)/stds)
```