Advanced Computational Neuroscience - Week 2
—
Sparce Coding

Sam Suidman

March 6, 2022

1 Introduction

When finding features in pictures a way to find them is by using principal component analysis (PCA). The way this works is that you represent an image I, which is normally represented by pixels with activity, in terms of its eigenvectors ϕ and corresponding feature activity a. You then look at which eigenvectors have the biggest eigenvalues and use this subset of eigenvectors to represent your image. The problem with PCA is that there can be many activities a that contribute to this. It seems that in nature coding (such as in the brain) is done with minimum entropy, which means that you want to code information with as little parameters as possible. For this there is an error term E used that will be minimized.

λ regulates how much of the error E_n is from the sparsity-term. This means that you need to look at the two terms E_n1 and E_n2. First the amount of E_n1 is bigger, because λ is 0. However if you increase λ you increase the part where this plays a bigger role. This has a maximum, because after that your model becomes so sparse that the other error terms takes over and the data is not described reliably anymore.

2 Theory

For PCA we start with images that have, for each of the d pixels located at $z = (x, y)$, a value between 0 and 1. There are in total N images and for each pixel we subtract the mean of all images such that we got a mean of 0 for each pixel. We represent this image as $I^n(z)$. We can compute the correlation matrix $C_0(z, z')$ with its eigenvalues λ_i and orthogonal eigenvectors $\phi_i(z)$, where we define

$$C_0(z, z') = \frac{1}{N}\sum_{n=1}^{N} I^n(z)I^n(z') \tag{1}$$

Each image can now be represented by its eigenvectors:

$$I^n(z) = \sum_{i=1}^{d} a_i^n \phi_i(z) \tag{2}$$

Here a_i^n is the activity of an image that corresponds to an eigenvector $\phi_i(z)$ (a feature). The $K < d$ biggest eigenvalues λ_i are chosen. So we now got representation of images via

$$I_{PCA}^n(z) = \sum_{i=1}^{K} a_i^n \phi_i(z) \tag{3}$$

where we can calculate

$$a^n_{i,PCA} = \sum_{z=1}^{d} \phi_i(z) I^n_{PCA}(z) \tag{4}$$

We now got the PCA activity $a^n_{i,PCA}$ for a set of images $I^n(z)$ corresponding to a set of eigenvectors ϕ_i. To create a sparse code we will minimize the following error function for each image $I^n(z)$ that has a term that tries to preserve information and a term that penalizes non-sparseness. This last term is dependent on the penalty parameter λ and here the penalty function is chosen as the absolute value of the activity, but can also be chosen differently.

$$E^n = E^n_1 + E^n_2 = \frac{1}{2} \sum_{z=1}^{d} \left[I^n(z) - \sum_{i=1}^{K} a^n_{i,PCA} \phi_i(z) \right]^2 + \lambda \sum_{i=1}^{K} |a^n_i| \tag{5}$$

We can calculate the total error E as the mean of the errors E^n over all images. In the approach I have done I looked at what happens to the activity a^n_i when minimizing the error function E^n while keeping the features $\phi_i(z)$ at their PCA values. This means that you want

$$\frac{\partial E^n}{\partial a^n_i} = 0 \tag{6}$$

We now define

$$b^n_i = \sum_{z=1}^{d} I^n(z) \phi_i(z) \tag{7}$$

and because the PCA features $\phi_i(z)$ are orthogonal

$$C_{ij} = \sum_{z=1}^{d} \phi_i(z) \phi_j(z) = \delta_{ij} \tag{8}$$

We can calculate b^n_i and C_{ij} from the images and PCA features. So when we combine equations 6, 7 and 8 we get a rule to calculate the sparse activities $a^n_{i,sparse}$ from the data.

For $|b^n_i| \leq \lambda$:

$$a^n_{i,sparse} = 0 \tag{9}$$

For $|b^n_i| \geq \lambda$:

$$a^n_{i,sparse} = b^n_i - \lambda sign(b^n_i) \tag{10}$$

3 Results

I made use of python with the packages numpy, matplotlib and skimage. I chose to have $K = 25$ features and two types of images. The image shown in Figure 1 is broken down in a subset of images with size 16×16 and random pixel data is generated. For this there are drawn random numbers x between 0 and 1 whereafter I calculated $I^n(z) = -log(x)$ with the mean of all images subtracted for each pixel as with the 'natural' flower image. Both sets of images are shown in Figure 2 and Figure 3.

When computing the first $K = 25$ principle components via PCA the features $\phi_i(z)$ are shown in Figure 4 if the patches are 8×8 pixels.

Now one thing that has not been discussed yet is the penalty parameter λ that determines how much the values a^n_i trade in preserving information for sparseness. Dependent on which value is chosen the error E can be calculated. It gets interesting when you look at the percentage that E^n_1 in equation 5 contributes to the total error E. For small λ this is of course low, but this is also the case if λ gets too big. Then too many a^n_i values will be 0 such that the coding is very sparse, but the effect is that in the end the images are not described reliably anymore and the other error term E^n_2 is taking over again. The sweet spot is where this taking over happens. Figures 5 and 6 show where this happens for the natural and pixel data.

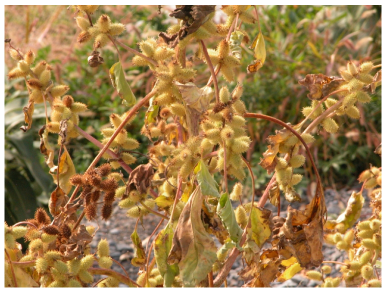

Figure 1: Flower image that is used.

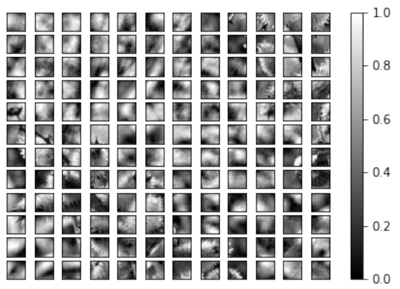

Figure 2: Flower is in greyscale and cut up in smaller images of size 16×16 pixels.

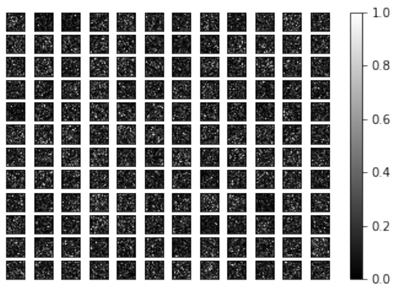

Figure 3: Random pixels are generated. The images have size 16×16 pixels.

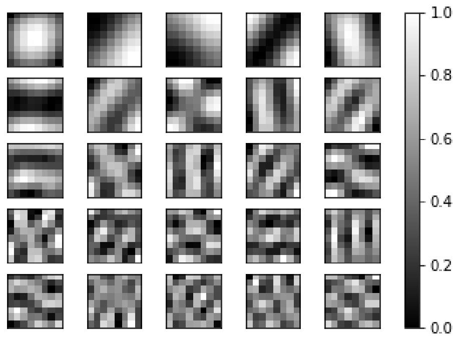

Figure 4: The first $K = 25$ features $\phi_i(z)$ for PCA when Figure 1 is broken down in images $I^n(z)$ of 8×8 pixels.

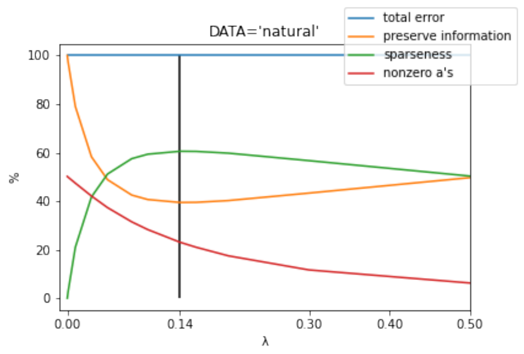

Figure 5: For different values of λ the error dependencies are shown with the percentage of non-zero a-values for the natural data.

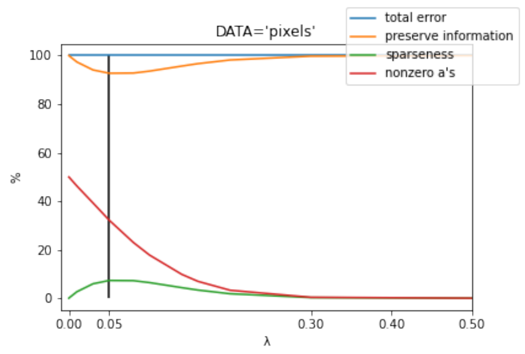

Figure 6: For different values of λ the error dependencies are shown with the percentage of non-zero a-values for the pixel data.

From the Figures we can see that this happens for the natural data around $\lambda = 0.14$, while for the pixel data this is $\lambda = 0.05$. In the Figures you can see that for the flower data the percentage of non-zero a-values has been taken down from 50.18% to 23.05% and for the pixel data this is from 49% to 31.3%. To be non-zero a_i^n needs to be bigger than a chosen threshold of 0.0001, because very small values of a_i^n that do not contribute significantly are otherwise not counted as 0. Taking the found λ values we can plot histograms for the different data types. Histograms for $a_{i,PCA}^n$ and $a_{i,sparse}^n$ can be seen in respectively Figure 7 and 8.

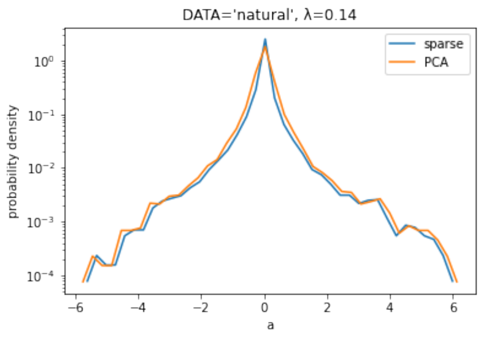

Figure 7: Histogram of a_i^n for the 'natural' flower image.

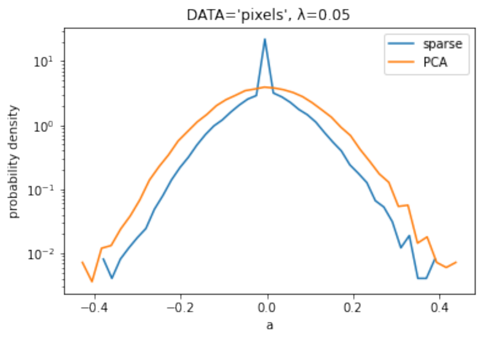

Figure 8: Histogram of a_i^n for the random generated pixels.

4 Conclusion

We see that minimizing the error function of equation 5 gives rise to the more sparse behaviour of Figure 7 and Figure 8 compared to PCA. This result is more clear in the case of random pixelated data than for natural images. By maximizing λ you find optimal sparseness and can see that the amount of non-zero a_i^n-values has dropped significantly while still preserving information.

5 Appendix

The code to run this can be found in the attached Jupyter Notebook and python file.