Assignment 2
IFT 6268 Winter 2015

Date posted: February 11, 2015
Due: February 25, 2015, at the *beginning* of class

---

1. (1/10) Show that independent random variables are uncorrelated.

2. (1/10) Show what happens to the covariance matrix, C, of a random variable $\boldsymbol{x}$, if we multiply $\boldsymbol{x}$ by a matrix M. Use this to show that the covariance matrix of the ICA generative model with generative weights A is given by AA^{T}.

3. (1/10) The inverse of the ZCA transform is also called the *square-root of the data covariance matrix*. Explain why.

4. (1/10) Download the CIFAR-10 dataset from

   `http://www.cs.toronto.edu/~kriz/cifar.html`

   (Same dataset as in Assigment 1). As before, if the 350MB size of the dataset causes problems, talk to instructor for a solution. Like in Assigment 1, turn the RGB-images into gray-value images using the formula

   ```
   trainimages = (trainimages*numpy.array([[[0.299], [0.587], [0.144]]])).sum(1)
   ```

   Use only the training set for this question. Crop patches (at least a few thousand), each of size 10×10 pixels, from random positions in the images. Make sure to crop patches from multiple different images (for example, one randomly positioned patch per image). DC center and contrast normalize each patch. Then compute the PCA whitening matrix and the ZCA whitening matrix of the (vectorized) patches. Display the 100 learned PCA und ZCA basis images as gray value images, sorted according to their corresponding eigenvalues.

   Hints:

   - To display a stack of filters you may use the function defined here:

     `http://www.iro.umontreal.ca/~memisevr/teaching/ift6268_2015/dispims.py`

   - You may get some very small eigenvalues. Make sure to avoid any numerical problems, for example, by adding a small constant when performing divisions.

   What to hand in: One image showing the 100 PCA filters, one image showing the 100 ZCA filters. Do *not* hand in any program code.

5. (1/10) Like the previous question, but this time using the original (*color*) images. *Hints:*

   - To display multiple color images that are contained in a 4D-"tensor", you may use the function defined in the module

     `http://www.iro.umontreal.ca/~memisevr/teaching/ift6268_2015/dispims_color.py`

     For example,

     `dispims_color.dispims_color(patches[:100])`

     would display color patches of size 32×32 pixels, contained an array with dimensions $(100, 32, 32, 3)$.
   - Be sure to represent your data as floats, otherwise strange things will happen.

   What to hand in: One image showing the first 100 color PCA filters, one image showing the first 100 color ZCA filters. Do *not* hand in any program code.

6. (5/10) In this question you will implement the regularized multiclass logistic regression ("softmax" regression) model and apply it to the color CIFAR data.

   The model is given by a matrix of parameters $\mathbf{W}$ and a vector of bias parameters $\mathbf{b}$. It defines the conditional distribution:

   $$p(\mathcal{C}_k|\mathbf{x}) = \frac{\exp(\mathbf{W}_{.k}^{\mathrm{T}}\mathbf{x} + b_k)}{\sum_i \exp(\mathbf{W}_{.i}^{\mathrm{T}}\mathbf{x} + b_i)} = \exp\left(\mathbf{W}_{.k}^{\mathrm{T}}\mathbf{x} + b_k - \log \sum_i \exp(\mathbf{W}_{.i}^{\mathrm{T}}\mathbf{x} + b_i)\right)$$

   where $\mathbf{W}_{.k}$ represents the k^{th} column of $\mathbf{W}$. Note that the form on the right is numerically stable when implemented using the "logsumexp"-function.

   What you need to do:

   - Implement a function cost() that, given the parameters $\mathbf{W}$ and $\mathbf{b}$, a training data set and a value for λ, computes the *penalized* log probability of the training data:

     $$-\sum_{nk} t_{nk} \log p(t_{nk}|\mathbf{x}_n) + \lambda \|\mathbf{W}\|^2$$

   - Implement a function grad() that computes the gradients containing partial derivatives $\frac{\partial E}{\partial \mathbf{W}_{.k}} = \sum_n \left(p(\mathcal{C}_k|\mathbf{x}_n) - t_{nk}\right)\mathbf{x}_n + \lambda \mathbf{W}_{.k}$ and $\frac{\partial E}{\partial b_k} = \sum_n \left(p(\mathcal{C}_k|\mathbf{x}_n) - t_{nk}\right)$ of the cost with respect to the model parameters.
   - You may use the logsumexp function provided at

     `www.iro.umontreal.ca/~memisevr/teaching/ift6268_2015/logsumexp.py`

     for numerical stability when computing cost and gradients.
   - Use gradient descent to train the model as follows: Initialize $\mathbf{W}$ and $\mathbf{b}$ to small random values. Then repeatedly update parameters by adding small multiples of the negative gradients to the parameters and re-evaluate the log-probability of the training data after each step. You may update your parameters either by iterating over your training set, visiting one point (or a few) at a time, or you may use *batch-learning* where a gradient-step involves the sum over all training cases.

- It may be necessary to experiment with the learning rate and parameter initializations to find settings that yield a fast and stable optimization.
- Before training the model, normalize each image by subtracting the DC-component and subsequently dividing by the standard deviation of the image. Don't forget to do the same normalization also for the test data!
- Train the model on the raw images, using a subset of the *training images* for validation in order to find a reasonable value for lambda.
- Repeat the experiment with data that is PCA-dimensionality reduced to 800 dimensions.
- Repeat the experiment once more, using a local feature extraction scheme proposed in "The Importance of Encoding Versus Training with Sparse Coding and Vector Quantization" by Coates and Ng, 2011. To this end follow these steps:
  (a) For each image (train and test), crop patches from all positions within the boundaries of the image. You should get 484 patches per image. Use a small subset of the training- and test-data if this leads to memory or efficiency problems. Mention in your report which dataset size you used. You will not loose any points for using a small subset. If problems persist, resize the images and describe in your report how you resized them (you will then get fewer patches per image).
  (b) DC center and contrast normalize each patch.
  (c) Compute the PCA whitening matrix from a subset of the *training* patches (you may want to use your code from the previous question). Do *not* use any test image patches for this!
  (d) Whiten all patches using your PCA matrix.
  (e) Designate 100 randomly chosen, whitened *training* patches as "prototypes", which we call $\boldsymbol{f}_1, \ldots, \boldsymbol{f}_{100}$ in the following. You may want to save them off in a separate matrix.
  (f) Turn all of your images into a feature vector as follows: For each image I, and for each patch $\boldsymbol{x}$ within the image, compute a 100-dimensional feature vector, whose i^{th} component is given by:

$$\max(0, \boldsymbol{f}_i^{\mathrm{T}} \boldsymbol{x})$$

  Then represent each image, I, as the *mean* over its 484 feature vectors.
  (g) Each image is now represented by a 100-dimensional (mean) feature vector.
  (h) Train your logistic regression model using $\lambda = 0.0$ on the *training* feature vectors, then test it on the test feature vectors. You may search for a better λ if you want to (but you don't have to). If you do, mention in your report the value you used.
  (i) Repeat the whole experiment using the ZCA whitening matrix instead of the PCA matrix. Then repeat the whole experiment using *no* whitening (simply replace your whitening matrix by the identity).

What to hand in:

- For the global (whole image) model, what value for λ did you find? What is the percentage of correctly classified training cases and of correctly classified test cases after training with this value for lambda? What is the log-probability of the training data and of the test-data? Which learning rate did you use, and for how many iterations did you train your model?
- The same information for the global, PCA-reduced data.
- The train and test-cost (and λ, if applicable) for the local feature extraction experiment.