Assignment 1
IFT 6268, Fall 2013

Date posted: September 25, 2013
Due: October 2, 2013, at the beginning of class

---

1. (2/10) The discrete Fourier Transform (DFT) of the one-dimensional signal $s(t)$ is defined as

$$S(k) = \sum_{t=0}^{T-1} s(t) e^{-i\frac{2\pi}{T}kt} \quad k = 0, \ldots, T-1$$

What happens to amplitude and phase spectrum of the signal, if we perform the translation:

$$s(t) \rightarrow s(t+\tau)$$

You may assume that the translation is with "wrap-around" (modulo the length of the signal).

2. (2/10) Using the definition of the one-dimensional DFT in the previous question, show that for *real valued signals* $s(t)$ the following is true:

$$S(l) = \bar{S}(T-l)$$

where $\bar{c}$ is the complex conjugate of c.

Hint:

- You may use the following property of complex numbers c_t: $\sum_t \bar{c}_t = \overline{(\sum_t c_t)}$

3. (2/10) Download the file

`www.iro.umontreal.ca/~memisevr/teaching/ift6268_2013/cifarmini_gray_images_train.txt`

and save it locally.

Each file contains 2000 rows, each of which contains 1024 integers separated by white space. The 1024 integers in each row represent a 32×32 gray value image from the CIFAR-10 object recognition dataset. (It is a subset of the CIFAR training set).

Using an FFT package of your choice, make a plot of the *average amplitude spectrum* of the images.

Then show the reconstruction of 10 of the images of your choice by combining their amplitude spectra with the phase spectrum of just *one* the images (same phase spectrum for all 10). Make another plot for the reconstruction of the same 10 images by combining their phase spectra with the amplitude spectrum of just one of the images (same amplitude spectrum for all 10). What does the result tell you about the relative importance of amplitude vs. phase for recognition?

Hints:

- The amplitude spectrum should be clearly visible. Use a log transform if necessary.
- All spectra should be displayed with the **0**-frequency vector in the center.
- Display real-valued images only. If the inverse FFT returns complex numbers you will need to deal with this somehow.
- Use gray values not colors to display the spectra.
- For those using Python 2.x/pylab:
  - The complex number i is represented by the expression "1j" in python.
  - You can access the real and imaginary parts of a complex number c using "c.real" and "c.imag", respectively.
  - To display the spectra you can use the function "imshow".
  - You may want to make use of the functions "numpy.fft.fft2", "numpy.fft.ifft2", "numpy.fft.fftshift", and "numpy.fft.ifftshift".
  - You can read data from a txt-file using numpy.loadtxt(FILENAME)
  - Be sure to represent your data as floats, otherwise strange things will happen.
  - You may make use of the function dispims at
    `www.iro.umontreal.ca/~memisevr/teaching/ift6268_2013/dispims.py`
    to display a set of images.

What to hand in: (a) One image of the average amplitude spectrum and your short description. (b) 30 images: 10 originals of your choice, the 10 images reconstructed using a single phase spectrum, the 10 images reconstructed using a single amplitude spectrum, as well as your short interpretation.

Do *not* hand in any program code.

4. (4/10) In this question you will implement the regularized multiclass logistic regression ("softmax" regression) model and apply it to some image 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:

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

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

Train the model on the training data provided in the following files (the first of which is the same file as in the previous question):

`www.iro.umontreal.ca/~memisevr/teaching/ift6268_2013/cifarmini_gray_images_train.txt`

`www.iro.umontreal.ca/~memisevr/teaching/ift6268_2013/cifarmini_labels_train.txt`

Then test the model on the data from

`www.iro.umontreal.ca/~memisevr/teaching/ift6268_2013/cifarmini_gray_images_test.txt`

`www.iro.umontreal.ca/~memisevr/teaching/ift6268_2013/cifarmini_labels_test.txt`

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 $\frac{\partial E}{\partial \mathbf{W}_{\cdot k}} = \sum_n \left(p(\mathcal{C}_k|\mathbf{x}_n) - t_{nk}\right)\mathbf{x}_n + \lambda \mathbf{W}_{\cdot k}$ 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_2013/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 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!

What to hand in:

- Train the model with $\lambda = 1.0$. What is the percentage of correctly classified training cases and of correctly classified test cases after training? What is the log-probability of the training data? Which learning rate did you use, and for how many iterations did you train?