CMPUT 670—Numerical Optimization

Assignment 1—update (13/10/2005)

Fall 2005
Department of Computing Science
University of Alberta

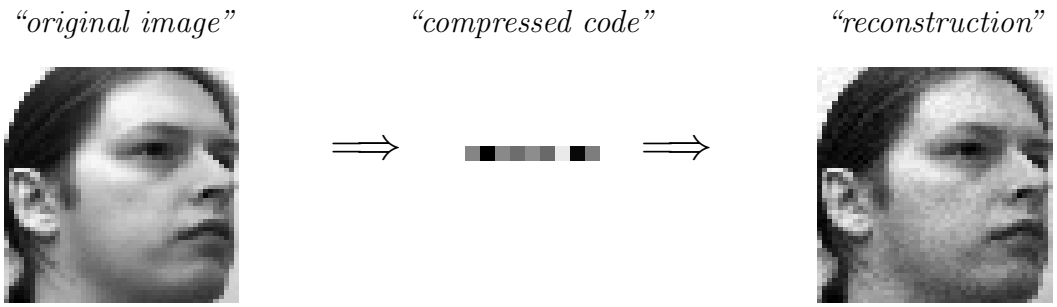
Due: *Midnight, Sunday, October 16*
Worth: 20% of final grade
Instructor: Dale Schuurmans, Ath409, x2-4806, dale@cs.ualberta.ca

---

Note Please submit your assignment to me by email. Most of the questions require you to write Matlab programs, however for the written questions please format your answers in a single *postscript* file. When finished, please send a *single* zip file containing all of your files to dale@cs.ualberta.ca with a subject heading "A1 solutions".

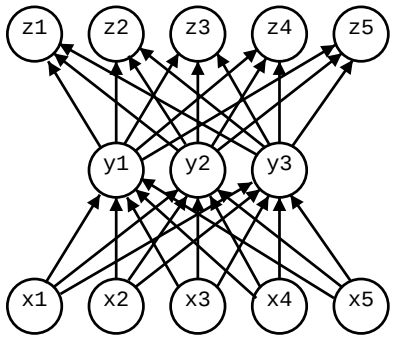
Neural network image compression

In this assignment you are going to train neural networks to perform image compression and reconstruction.



The idea is to take a bunch of images and "train" a neural network to compress and then decompress the images with as little loss as possible. This training process is an optimization problem that you will have to solve.

First, some background on the neural network model you will use. For this assignment, we will consider the simple "bottleneck" network architecture shown below.

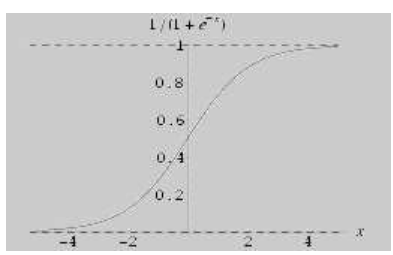


The first (bottom) layer receives an image represented in a vector form. (That is, we will take an original image X—a matrix of numbers—and first convert it to a vector $\mathbf{x} = (x_1, ..., x_N)^\top$ by stacking the columns.) Then, the middle layer, $\mathbf{y} = (y_1, ..., y_K)^\top$, computes a compressed encoding of the original image. Typically, $K << N$ so this results in a significant compression. Finally, given a compressed code $\mathbf{y}$, the second (top) layer computes a reconstruction, $\mathbf{z}$, of the original image $\mathbf{x}$. The functions (compression and reconstruction) are computed as follows:

$$\begin{aligned} \mathbf{y} &= \boldsymbol{\sigma}(W\mathbf{x}) && \text{(computes compressed code from input)} \\ \mathbf{z} &= \boldsymbol{\sigma}(V\mathbf{y}) && \text{(computes image reconstruction from code)} \end{aligned}$$

where W is a $K \times N$ matrix of weights used in the compression, $W_{\ell j}$, $\ell = 1, ..., K$, $j = 1, ..., N$; V is an $N \times K$ matrix of weights used in the reconstruction, $V_{i\ell}$, $i = 1, ..., N$, $\ell = 1, ..., K$; and $\boldsymbol{\sigma}$ is a sigmoid function, σ, applied componentwise to a vector input, such that, for example $\boldsymbol{\sigma}(\mathbf{u}) = (\sigma(u_1), ..., \sigma(u_M))^\top$. The sigmoid function, σ, is given by

$$\sigma(u) = \frac{1}{1 + e^{-u}}$$



$$0 \leq \sigma(u) \leq 1$$

$$\sigma(0) = 1/2$$

Note that weight $W_{\ell j}$ connects input x_j to code component y_ℓ, and weight $V_{i\ell}$ connects code component y_ℓ to output z_i. Also note that the compression and reconstruction functions are nonlinear. However, they are both continuous and arbitrarily differentiable.

Given the set of "training" image vectors, $\mathbf{x}^{(1)}, ..., \mathbf{x}^{(T)}$, the problem is to compute the compression and reconstruction weights, W and V, to minimize the total reconstruction loss:

$$\begin{aligned}
err &= \sum_{t=1}^{T} err^{(t)} \\
err^{(t)} &= \|\mathbf{x}^{(t)} - \mathbf{z}^{(t)}\|^2 = \sum_{i=1}^{N} (x_i^{(t)} - z_i^{(t)})^2 \\
\mathbf{z}^{(t)} &= \boldsymbol{\sigma}(V\mathbf{y}^{(t)}) \\
\mathbf{y}^{(t)} &= \boldsymbol{\sigma}(W\mathbf{x}^{(t)})
\end{aligned}$$

That is, we would like to solve for

$$W^*, V^* = \arg\min_{W,V} err$$

You will implement procedures for solving this optimization problem.[1] Interestingly, once the weights W and V have been optimized, they can be used to compress and reconstruct new images and codes. Unfortunately, the optimization objective *err* is not a convex function of W and V, and can in fact have multiple local minima. (This is one reason why neural networks are not as popular as they once were.) Nevertheless, we will tackle this optimization problem in this assignment.

[1]Note that from the perspective of the optimization problem the weights W, V are the variables we wish to optimize, and we think of *err* as a function of W and V alone. That is, the training images just determine the particular objective function, and therefore they are just parameters that are not themselves being optimized.

Question 1 (Gradient and Hessian derivation—3%)

(a) Derive

$$\frac{\partial err^{(t)}}{\partial V_{i\ell}} \quad \text{and} \quad \frac{\partial err^{(t)}}{\partial W_{\ell j}}$$

(b) Derive

$$\frac{\partial^2 err^{(t)}}{\partial V_{i\ell}^2} \quad \text{and} \quad \frac{\partial^2 err^{(t)}}{\partial V_{i\ell}\,\partial V_{jk}} \quad \text{for } j \neq i \text{ or } \ell \neq k$$

$$\frac{\partial^2 err^{(t)}}{\partial W_{\ell j}^2} \quad \text{and} \quad \frac{\partial^2 err^{(t)}}{\partial W_{\ell j}\,\partial W_{ki}} \quad \text{for } j \neq i \text{ or } l \neq k$$

$$\frac{\partial^2 err^{(t)}}{\partial V_{i\ell}\,\partial W_{\ell j}} \quad \text{and} \quad \frac{\partial^2 err^{(t)}}{\partial V_{i\ell}\,\partial W_{kj}} \quad \text{for } l \neq k$$

By derive, I mean find expressions for these partial derivatives that can be computed from things you will know: components of the observed input vector, $\mathbf{x}^{(t)}$, and components of the current weight values, W and V. Please format your answer in a postscript file "A1answers.ps".

Hint You need to use the chain rule a lot.
Recall also that if $h(x_1, ..., x_n) = f(g_1(x_1, ..., x_n), ..., g_k(x_1, ..., x_n))$ then

$$\frac{\partial h}{\partial x_j} = \sum_{\ell=1}^{k} \frac{\partial h}{\partial g_\ell} \frac{\partial g_\ell}{\partial x_j}$$

Also, the nonlinear sigmoid function, σ, has some very nice properties you will find useful.

$$\begin{aligned} \sigma(-u) &= 1 - \sigma(u) \\ \sigma'(u) &= \sigma(u)(1 - \sigma(u)) \\ \sigma''(u) &= \sigma(u)(1 - \sigma(u))(1 - 2\sigma(u)) \end{aligned}$$

(You can easily verify these on your own from the definition of σ given above.) Finally, if one is uncomfortable with vector notation you can always write $err^{(t)}$ out as

$$\begin{aligned} err^{(t)} &= \sum_{i=1}^{N} \left(x_i^{(t)} - z_i^{(t)}\right)^2 \\ &= \sum_{i=1}^{N} \left(x_i^{(t)} - \sigma\left(\sum_{\ell=1}^{K} V_{i\ell} y_\ell^{(t)}\right)\right)^2 \\ &= \sum_{i=1}^{N} \left(x_i^{(t)} - \sigma\left(\sum_{\ell=1}^{K} V_{i\ell}\ \sigma\left(\sum_{j=1}^{N} W_{\ell j} x_j^{(t)}\right)\right)\right)^2 \end{aligned}$$

Question 2 (Gradient implementation—3%)

To implement a weight optimization procedure it is convenient to represent the entire set of weights, W, V, by a single weight vector, `weights = [wvec; vvec]`, such that `wvec = W(:)` and `vvec = V(:)`. From now on, assume this convention. Also assume that images are represented by column vectors `x`.

(a) Write Matlab functions that compute the objective value and gradient vector for the local objective $err^{(t)}$ defined by a single image, as follows:
`[loss] = errt_loss(w,x)`, `[loss,grad] = errt_grad(w,x)`.
Submit these functions in files "errt_loss.m" and "errt_grad.m" respectively.

(b) Write Matlab functions that compute analogous quantities for the full objective err, as follows: `[loss] = err_loss(w,X)`, `[loss,grad] = err_grad(w,X)`.
Here `X` is an $N \times T$ matrix of image vectors where each image is a column.
Submit these functions in files "err_loss.m" and "err_grad.m" respectively.

Note The dimensionality of the compressed code, `y`, can be inferred from the dimensionality of the weight vector `w` being passed in as an argument. Recall that `w` has length equal to the total number of weights in the model, which in this case is $M = 2NK$ such that M is the length of `w`. Since N is just the length of `x`, we can then compute $K = M/(2N)$. If desired, you can then recover the matrix form of the weights, W and V, by splitting `w` into the `wvec` and `vvec` parts and using `reshape(wvec,K,N)` and `reshape(vvec,N,K)`.

Question 3 (Gradient descent—4%)

Write a Matlab function, `[w] = graddescent(fun,w0,tol,Pars)`, that implements a *gradient descent* optimization of objective function `fun` parameterized by `Pars`, starting from initial point `w0` and trying to achieve accuracy `tol` in the final answer `w`. The function `fun` should return an objective value and gradient. Submit your function in a file "graddescent.m".

Note that you can use necessary support functions like `feval`, but cannot use functions in `help optim` or `help nnet`. That is, you have to implement your own gradient descent optimizer and not just call someone else's. You can use any stepsize control method you wish, except that it should converge. Please document your stepsize control method in "A1answers.ps" and explain why it converges.

Finally, note that your `graddescent` function can be used to optimize the weight parameters for the neural network by using `weights = graddescent('err_grad',w0,tol,X)`.

Question 4 ("Stochastic" gradient descent—2%)

Write a Matlab function, `[w] = stochgrad(fun,w0,tol,Pars)`, that implements a *"stochastic"* gradient descent, where it is assumed that `Pars = [p1 ... pT]` consists of several parameter vectors. That is, the overall objective is to minimize

$$\sum_{t=1}^{T} \texttt{feval(fun,w,pt)}$$

but this is done by looping through each local parameter vector and taking gradient descent steps in each `feval(fun,w,pt)` one at a time repeatedly, until convergence. Submit your function in a file "stochgrad.m". You can use any stepsize control method you wish, but again, document it in "A1answers.ps".

Finally, note that the `stochgrad` function can also be used to optimize the weight parameters for the neural network by using `weights = stochgrad('errt_grad',w0,tol,X)`.

Question 5 (Experimental comparison—3%)

In this question you will compare the quality and efficiency of the two previous optimizers, `graddescent` and `stochgrad`, for optimizing the parameters of the neural network on real image data. On the course webpage there is a datafile, data1.mat. Download this file and load it into Matlab by typing `load data1.mat`. There will be two matrices `Xtrain` and `Xtest`. Using an initialization strategy of your choice, run both optimizers on `Xtrain` using the same initial `w0` and `tol`. Use compression `K=64`. For each optimizer, record:

(1) total optimization time,
(2) total number of gradient descent steps,
(3) average run time per step,
(4) final objective value achieved on `Xtrain`, and
(5) total reconstruction loss, *err*, on `Xtest` (that is, just call `err_loss(w,Xtest)`).

Describe the relative advantages and disadvantages of each method. Include your answer in "A1answers.ps".

Note If you wish to view the images that are stored as columns in a matrix `X`, first type `colormap gray` in Matlab, then convert each image vector back into a matrix using `reshape` and view using `imagesc` as follows: `imagesc(reshape(X(:,i),16,16))`.

Question 6 (Global optimization—5%)

Write a Matlab function, `[w] = globalopt(fun,w0,tol,Pars)`, that implements any global optimization strategy that you wish. Submit your function in a file "globalopt.m", and explain your method in "A1answers.ps". Please ensure your function can be used to solve the same problem as above using `weights = globalopt('err_grad',w0,tol,X)`.

Bonus! (2%)

I will give extra marks for the global optimizer that achieves the best objective value on `Xtrain` and the best reconstruction error on `Xtest` within a reasonable time limit.