CMPUT 466/551 — Assignment 3

Instructors: R Greiner, B Poczos
Due Date: 12:30pm, Tues, 24/Nov/09
The following exercises are intended to further your understanding of (Linear Algebra and Probability Theory) related to Gaussian Processes, Artificial Neural Nets, (including Line Search and Conjugate Gradient), Decision Trees, and Ensemble Methods.
Relevant reading: HTF: Ch 10 (8.7, 16), 11 + `http://www.gaussianprocess.org/`[Chapter 2]
Total points: UGrad: 99 Grad: 113
Grad students should answer every question; undergrads do not have to answer Q6b and Q9b.

The `ReadMe` file will specify the details about the datasamples, and other coding issues, including a specification of what to hand and how; for Q3, Q5, and Q7.

Question 1 *[4 points] Partitioned covariance and inverse covariance matrices*
Let $\Sigma \doteq \begin{pmatrix} \Sigma_{aa} & \Sigma_{ab} \\ \Sigma_{ba} & \Sigma_{bb} \end{pmatrix} \in \Re^{m \times m}$ be an arbitrary invertible matrix, partitioned into block matrices, and let $\Lambda \doteq \Sigma^{-1} = \begin{pmatrix} \Lambda_{aa} & \Lambda_{ab} \\ \Lambda_{ba} & \Lambda_{bb} \end{pmatrix}$
Prove that the following equations hold (assuming that the inverse marices exist):
a [2]: $\Lambda_{aa}^{-1}\Lambda_{ab} = -\Sigma_{ab}\Sigma_{bb}^{-1}$
b [2]: $\Lambda_{aa}^{-1} = \Sigma_{aa} - \Sigma_{ab}\Sigma_{bb}^{-1}\Sigma_{ba}$

[Hint: Use the blockwise matrix inversion formulas from:
`http://en.wikipedia.org/wiki/Invertible_matrix#Blockwise_inversion`]

Question 2 *[5 points] Posterior is normal*
Let $X_* = \begin{bmatrix} \mathbf{x}_{*1}^T \\ \vdots \\ \mathbf{x}_{*m}^T \end{bmatrix} \in \Re^{m \times D}$ be m test input points, and $X = \begin{bmatrix} \mathbf{x}_1^T \\ \vdots \\ \mathbf{x}_n^T \end{bmatrix} \in \Re^{n \times D}$ be n training input points. Let $k(\cdot,\cdot)$ be a kernel function, and let the joint distribution of the noisy observations $\mathbf{y} \in \Re^n$ and the noise free test outputs $\mathbf{f}_* \in \Re^m$ be given by

$$\begin{bmatrix} \mathbf{y} \\ \mathbf{f}_* \end{bmatrix} \sim \mathcal{N}_{\begin{bmatrix} \mathbf{y} \\ \mathbf{f}_* \end{bmatrix}} \left\{ \begin{bmatrix} \mathbf{0}_n \\ \mathbf{0}_m \end{bmatrix}, \begin{bmatrix} k(X,X) + \sigma^2 \mathbf{I}_n & k(X,X_*) \\ k(X_*,X) & k(X_*,X_*) \end{bmatrix} \right\}$$

Prove that the posterior distributions of the function values at the test points can be expressed as

$$P(\mathbf{f}_* | X, \mathbf{y}, X_*) = \mathcal{N}_{f_*}(\bar{\mathbf{f}_*}, cov(\mathbf{f}_*)) ,$$

where

$$\bar{\mathbf{f}_*} = \mathbb{E}[f_* | X, \mathbf{y}, X_*] = k(X_*, X)[k(X,X) + \sigma^2 I_n]^{-1}\mathbf{y} \quad \in \Re^m$$

and

$$\mathrm{cov}(\mathbf{f}_*) \;=\; k(X_*, X_*) - k(X_*, X)[k(X, X) + \sigma^2 I_n]^{-1} K(X, X_*) \quad \in \Re^{m \times m}$$

[Hint: What is the distribution of the marginals of a multivariate Guassian conditioned on the other marginals?]

Question 3 *[20 points] GP regression implementation*
Implement a Gaussian process algorithm for the regression problem. Assume that there is an ϵ noise on the observations:

$$y(\mathbf{x}) = f(\mathbf{x}) + \epsilon, \qquad \epsilon \sim \mathcal{N}(0, \sigma^2)$$

The `ReadMe` file specifies what to test, and what to hand in.

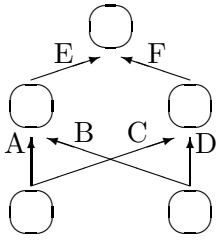
Question 4 *[10 points] Gradient Descent*
Consider a new *ptron* unit with inputs x_1 and x_2, which produces a real-valued output defined as

$$s_{\mathbf{w}}(x_1, x_2) \quad = \quad w_0 \;+\; (w_0\, w_1)\, x_1 \;+\; (w_1\, w_2)\, x_2$$

Describe an algorithm for learning the weights w_0, w_1 and w_2 of a single ptron, from a datasample $S = \{[x_1^i, x_2^i; y^i]\}_{i=1..m}$, using gradient descent, based on the error function

$$\mathrm{err}(\mathbf{w}; S) \quad = \quad \sum_{[x_1, x_2; y] \in S} (y - s_{\mathbf{w}}(x_1, x_2))^2$$

Question 5 *[18 points] Neural Net Learning*



Consider the neural net
where all units are sigmoid — *i.e.*, each uses the logistic $\sigma(x) = 1/(1 + e^{-x})$ squashing function.

a [10]: Implement the back-propagation learning algorithm for this network, in Matlab. (Just the simplest version; set the learning rate $\eta = 0.05$, initialize the weights randomly, ...)
(The `ReadMe` file specifies what to test, and what to hand in.)

For the remaining parts of this question, assume we begin with all weights having exactly the same value and run back-propagation on some set of data. Which statements below will be always true? For each, explain your reasoning in at most a few sentences.

b [2]: Weights A and B never differ.

c [2]: Weights A and C never differ.

d [2]: Weights E and F never differ.

e [2]: Give a short explanation of one good reason NOT to initialize all weights to the same value when running backprop.

Question 6 *[3 + 7 points] Conjugate Gradient Factoids*
Show the following, using the notation defined in the class lecture notes.

a [3]: Show that $\mathbf{d}_j^T \mathbf{g}_k = 0$ for all $j < k$.
[Hint: First show this holds for $k = j + 1$.]

b [7 Grad students only]: Show that $\mathbf{g}_i^T \mathbf{g}_j = 0$ for $i \neq j$.

Question 7 *[30 points] Conjugate Gradient*
The purpose of this question is to understand Conjugate Gradient (GC). For now, let's forget about the *estimation* part of parameter optimization by assuming we already know everything there is to know about the error function, and just want to find the values of the parameters that minimize this function. To make life easy, we will consider an error function that is a simple quadratic over the 2 parameters, x and y:

$$f(x, y) \quad = \quad a \times x^2 \; + \; b \times y^2 \tag{1}$$

for some specified values of a and b. Our task is to (write a program that can) efficiently "descend" from a given initial vector (x_1, y_1) to produce the values of (x, y) that minimize this function. (Yes, we all know this minimal value is at $(x^*, y^*) = (0, 0)$. But here the path is important...)

You will implement three routines — simple gradient descent (GD), GD with line search, and CG — and compare their behavior over the following initial set-ups.

a	b	initial_weight vector, (x_1, y_1)
1	1	$[10, 0]$
1	1	$[10, 10]$
1	10	$[0.1, 0.1]$
1	10	$[20, 10]$
1	1000	$[40, 40]$

In particular, you should indicate how many iterations each algorithm required to converge — here to an error of 1E-6, or alternatively when your program stopped "prematurely". You should also submit a ZIP file of the relevant programs, such that the top-level Matlab call

`>> Weights = GradientDescent( 1,10, [20,10], χ)`

deals with the $a = 1$, $b = 10$ and initial_weight= $[20, 10]$ setting; here `Weights(i,:)` should be the weights associated with the i^{th} iteration of the algorithm. This routine should cover all three different processes shown below, depending on the value of χ provided; see below. You may also wish to plot these results, to help visualize what is going on.

For notation, at iteration j, you have the current weights w_j and can directly compute the gradient g_j, as well as the (constant) Hessian H, which you will use to compute the new weights w_{j+1}.

a [10]: Setting the third argument "$\chi = 0$" should call your implementation of simple gradient descent, which uses the simple heuristic of traveling $\eta = \kappa/j$ along the gradient g_j;

you get to decide on the value for κ, but you need to specify this value and it should be the same for all set-ups.

b [3]: Given the weight vector w_j and gradient g_j, you want to descend along g_j to a new vector $w_{j+1} \leftarrow w_j + \eta\, g_j$ that produces the smallest $f(.)$ value — *i.e.*,

$$\eta \quad = \quad \underset{t}{\text{argmin}}\, f(w_j + t g_j) \tag{2}$$

In class, we provided an iterative way to do this, which is useful if $f(\cdot)$ is unknown. Here, we know $f(\cdot)$. Provide a simple direct formula for computing Equation 2 based on our particular $f(\cdot)$ from Equation 1.

[Hint: Take the derivative of values along this line.]

c [7]: Setting "$\chi = 1$" should call your implementation of gradient descent using "line-search": that is, your code should determine the optimal distance to travel along the gradient g_j, to minimize $f(x, y)$ — using your answer to (b) above. (This should be a simple direct computation — it does NOT require any iterations.)

d [10]: Setting "$\chi = 2$" should call your implementation of conjugate gradient: Here, at iteration j, you should compute the new weights w_{j+1} using the following formulae:

$$\begin{aligned} w_{j+1} &= w_j \;+\; \alpha_j\, d_j \\ d_1 &= -g_1 \\ d_{j+1} &= -g_{j+1} + \beta_j d_j \qquad \text{for } j \geq 1 \end{aligned}$$

$$\beta_j \quad = \quad \frac{g_{j+1}^T (g_{j+1} - g_j)}{g_j^T g_j} \qquad \alpha_j \quad = \quad -\frac{d_j^T g_j}{d_j^T H d_j}$$

Once again, you can just directly compute the Hessian matrix H, and every other quantity needed to go from w_j to w_{j+1}, without any iterations.

Question 8 *[6 points] Decision Tree Classification*

In the recursive construction of decision trees, we may sometimes produce a leaf node based on a mixed set of positive and negative examples — *e.g.*, due to pruning, or perhaps after all the attributes have been used. Suppose that we have P positive and N negative examples.

a [3]: One approach is to return the majority classification — *i.e.*, "true" if $P > N$; else "false". Show that this approach minimizes the *absolute error*, over the set of examples that reach this leaf node.

b [3]: Suppose the decision tree, on reaching this leaf, returns the probability α. This means the "squared error" for each positive instance is $(1 - \alpha)^2$ as the decision tree should have returned 1 here; and the squared error for each negative instance is $(0 - \alpha)^2$. Find the value of α (as a function of P, N, and any other relevant aspect, such as number of attributes, etc.) that minimizes the sum of these squared errors.

Question 9 *[3 + 7 points] Ensemble: Mixing using Variance*

Suppose you have K regressors $\{h_k(\cdot)\}$, each of which returns, for each $\mathbf{x}$, an estimate $h_k(\mathbf{x})$ of the response $\mu(\mathbf{x})$, along with the variance σ_k^2 of this estimate — *i.e.*, $h_k(\mathbf{x}) = \mu(\mathbf{x}) + \epsilon_k(\mathbf{x})$ where $\epsilon_k(\mathbf{x}) \sim \mathcal{N}(0, \sigma_k^2(\mathbf{x}))$. Note that the standard deviation of the error, $\sigma_k(\mathbf{x})$, can differ for different $\mathbf{x}$s. To simplify notation, however, we will suppress the $(\mathbf{x})$ — *i.e.*, we will consider only a single $\mathbf{x}$.

a [3]: First, assume that these estimates are uncorrelated with each other — $E[\epsilon_j \epsilon_k] = 0$ for $j \neq k$ — and are also independent of x.

What is the best linear combination of these predictors: that is, the values of $\{\alpha_k\}$ such that

$$h^* \;=\; \sum_k \alpha_k h_k$$

minimizes the expected squared error:

$$err(\alpha) = E[(\mu - h^*)^2]$$

where

$$\sum_k \alpha_k = 1 \ .$$

[Hint: Think of Lagrange Multipliers...]

b [7 Grad students only)]: Now allow the errors to be *correlated* — that is

$$\epsilon \;=\; \begin{bmatrix} \epsilon_1 \\ \vdots \\ \epsilon_n \end{bmatrix} \quad \sim \quad \mathcal{N}\left(\begin{bmatrix} 0 \\ \vdots \\ 0 \end{bmatrix}, \; \Sigma \right)$$

where this $\Sigma = (\sigma_{jk})$ covariance matrix is not (necessarily) diagonal — *i.e.*, $\sigma_{jk} = E[\epsilon_j \epsilon_k]$ may be non-0 when $j \neq k$. Now what is the "best" linear combination of these predictors (using the equations show above)?