CSC2515 – Assignment #2

Due: Nov.6, 3pm at the **START** of class
Worth: 18%
Late assignments not accepted.

1 Pseudo-Bayesian Linear Regression (3%)

In this question you will dabble in Bayesian statistics and extend the probabilistic model formulation to include not only the data but also the weights.

Consider the following conditional model of a scalar output y given some inputs $\mathbf{x}$:

$$
\begin{aligned}
p(y|\mathbf{x}) &= \int_{\mathbf{w}} p(y|\mathbf{x}, \mathbf{w}) p(\mathbf{w}) d\mathbf{w} \\
p(y|\mathbf{x}, \mathbf{w}) &= \mathcal{N}(y; \mathbf{w}^\top \mathbf{x}, b) \\
p(\mathbf{w}) &= \mathcal{N}(\mathbf{w}; \mathbf{0}, a\mathbf{I})
\end{aligned}
$$

Here the weight prior $p(\mathbf{w})$ is a Gaussian with mean zero and covariance $a\mathbf{I}$ ($\mathbf{I}$ is the identity matrix) and the data model is a conditional linear-Gaussian with mean $\mathbf{w}^\top \mathbf{x}$ and variance b.

- Find the posterior distribution over weights $p(\mathbf{w}|\{\mathbf{x}_n, y_n\}, a, b)$ given a finite training set $\{\mathbf{x}_n, y_n\}$ and the variances a, b. To do this, you will need to form the joint Gaussian $p(y, \mathbf{w}|\mathbf{x})$, marginalize out the weights to form $p(y|\mathbf{x})$ and then use Bayes rule to find $p(\mathbf{w}|\mathbf{x}, y)$.
- Show that the ridge regression weights, which minimize the cost $\lambda \mathbf{w}^\top \mathbf{w} + \sum_n (y_n - \mathbf{w}^\top \mathbf{x}_n)^2$ are equal to the mean (and mode) of the posterior above.
- Give the ridge regression penalty λ in terms of the variances a, b above.

2 Regularizing Linear Mixtures of Experts (2%)

In class I did not tell you anything about how to regularlize mixtures of experts models. This question asks you to work out the details for one method and to suggest one more. Recall that the linear mixture of experts model uses a gate to compute $p(j|\mathbf{x}_n)$ and then stochastically selects an expert according to these probabilites. The experts outputs are given by $p(\mathbf{y}|j, \mathbf{x}_n) = \mathcal{N}(\mathbf{y}; \mathbf{U}_j\mathbf{x}_n, \Sigma)$. Assume the gate is fixed, so that the quantities $p(j|\mathbf{x}_n)$ are known and do not change. Assume also that the output covariance Σ is known and fixed.

One obvious way to regularlize a *linear* mixtures of experts model using squared error is to do *ridge regression* at the experts instead of maximum likelihood regression. This means the likelihood function would become:

$$\ell^{new} = \sum_n \log \sum_j p(j|\mathbf{x}_n)\mathcal{N}(\mathbf{y}_n; \mathbf{U}_j\mathbf{x}_n, \Sigma) + \lambda \sum_{ijk} U_{ijk}^2$$

where U_{ijk} is the weight from input i to output k in expert j.

Recall that the gradient with respect to an expert's weights $\mathbf{U}_j$ in the original linear MOE was:

$$\partial \ell^{old} / \partial \mathbf{U}_j = \sum_n p(j|\mathbf{x}_n, \mathbf{y}_n) \left(\mathbf{y}_n - \mathbf{U}_j\mathbf{x}_n\right) \mathbf{x}_n^\top$$

which uses the *posterior* probability of each expert:

$$p(j|\mathbf{x}_n, \mathbf{y}_n) = \frac{p(j|\mathbf{x}_n)p(\mathbf{y}_n|j, \mathbf{x}_n)}{\sum_k p(k|\mathbf{x}_n)p(\mathbf{y}_n|k, \mathbf{x}_n)}$$

- When we use ridge regression, the new likelihood function will, of course, have different gradients with respect to the expert weights. Derive the new gradient $\partial \ell^{new} / \partial \mathbf{U}_j$ with respect to an expert's weights $\mathbf{U}_J$ under the ridge regression likelihood above.
- Suggest one other, different way of regularizing a linear mixture of experts architecture to prevent overfitting.

3 Adapting Centres in Radial Basis Networks (3%)

In this question you will find the derivatives which would allow you to adapt the centres in a radial basis network.

Recall that a radial basis network is a generalized linear model of the form

$$y = \sum_j w_j h_j(\mathbf{x})$$

$$h_j(\mathbf{x}) = \exp\left[-(\mathbf{x} - \mathbf{z}_j)^\top (\mathbf{x} - \mathbf{z}_j)/\alpha\right]$$

Assume we are using a Gaussian noise model so that the likelihood becomes squared error:

$$E = \sum_n (y_n - \sum_j w_j h_j(\mathbf{x}_n))^2$$

As you know, for fixed α and fixed $\mathbf{z}_j$, the maximum likelihood weights under this Gaussian noise can be found by linear regression. But if $\mathbf{z}_j$ are unknown we would like to adapt them.

- Compute the gradient $\partial E/\partial \mathbf{z}_f$ of E with respect to $\mathbf{z}_f$, given α, a finite training set $\{\mathbf{x}_n, y_n\}$ and the current values of $\mathbf{z}_j$ for all j.

4 Training a Neural Network (10%)

IMPORTANT: please do your computing on CDF or your own machine and not on CSLAB machines since they have limited matlab licenses.

In this question you will train a one-hidden-layer neural network to perform a simple medical prediction task. You can find out more about the task and data in the file `a2data.txt`.

I have provided matlab code for training a neural network using backpropagation. Currently it uses squared error at the output and the *sigmoid* nonlinearity at the hidden units and on the final output units. You do not have to use this matlab code if you don't want to, but you have to read it and understand it to answer the first two questions.

To use the code, select the number of hidden units in your network, initialize the input-to-hidden weights (and biases) and hidden-to-output weights, and run the function `trainmlp` which will do gradient descent with momentum on the error using backprop to compute derivatives. You need to choose a momentum parameter and a maximum number of iterations. To find the output of the trained network on some new data, use the `backprop` function with only one output argument. Hence, a typical training procedure for a 12 hidden unit network with low momentum might look like this:

```
H=12; alpha=0.1; maxiters=1e5;
ww0=1e-4*randn(size(xtrn,1)+1,H); vv0=1e-4*randn(H+1,size(ytrn,1));
[ww,vv,yy,trainerr,testerr] = trainmlp(xtrn,ytrn,xtst,ytst,ww0,vv0,maxiters,alpha);
```

- Show the changes you would have to make to `bprop.m` in order to convert this program into one for a network using squared error (as it is now) at the output but the *tanh* nonlinearity.
- [Harder]. Show the changes you would have to make in order to convert this program into one for a network using the sigmoid nonlinearity but a new error function called *cross-entropy* error at the output instead of squared error. Cross entropy error is defined by $E_{ce} = -\sum_n y_n \log f(\mathbf{x}_n) + (1-y_n)\log(1-f(\mathbf{x}_n))$, where $f(\mathbf{x}_n)$ is the network output for input $\mathbf{x}_n$.
- Using the data provided in the file `a2data.mat` (or `a2datax.ascii` if you aren't using matlab), train the squared-error sigmoid network with 20 hidden units on the training data. Monitor the test error on the test data (but don't cheat and use it for training!). [The program I provided does all this for you.] Initialize with weights drawn independently from a Gaussian with mean zero and variance 10^{-4}. Train for at least 30,000 iterations using very high momentum (I suggest 0.95). Hand in a single plot, clearly labelled, showing both training and testing errors, normalized by the sizes of training and test sets as training proceeds (i.e. average error per case). Save the initial weights you used, because you will need them in the next question.
- Mark on your plot the iteration at which the minimum of training error occurs.
  Mark on your plot what the test error is there.
  Mark on your plot the iteration at which the minimum of *test* error occurs.
  Mark on your plot what the test error is there.
  Does the 20 hidden unit network overfit?
- Starting from the same random initial weights as before, retrain the 20 unit network for the optimal number of iterations to get the best test error. Using the resulting weights, predict the outputs on the test cases. Hand in a single plot showing the true test outputs and the network outputs on the test cases after stopping at the optimal training iterations.
- Train again, this time using only 1 hidden unit. Hand in another plot showing normalized test and training errors. (You don't need to mark the minima this time.)
  Does the 1 hidden unit network overfit?

Note: Do not hand in a printout of a program. For the first two questions, only hand in the *changes* you would have to make, which are at most a few lines. Even if you don't use matlab for the rest of the question, your changes should still be in matlab notation. Do not edit the original program and just print out the new version. Hand in only the *changes*. Don't hand in other code. Really, we're not kidding.

Hint: you can check you answers to the first two questions by modifying the program (or modifying your own program if you don't want to use matlab) and making sure the error always goes down.