CPSC 540 Assignment 5 (due April 12)

Bayesian Statistics, Graphical Models, Deep Learning

1 Bayesian Statistics

Consider a $y \in \{1, 2, 3\}$ following a multinoulli distribution with parameters $\theta = \{\theta_1, \theta_2, \theta_3\}$,

$$y|\theta \sim \text{Mult}(\theta_1, \theta_2, \theta_3).$$

We'll assume that θ follows a Dirichlet distribution (the conjugate prior to the multinoulli) with parameters $\alpha = \{\alpha_1, \alpha_2, \alpha_3\}$,

$$\theta \sim \mathcal{D}(\alpha_1, \alpha_2, \alpha_3).$$

Thus we have

$$p(y|\theta, \alpha) = p(y|\theta) = \theta_1^{I(y=1)}\theta_2^{I(y=2)}\theta_3^{I(y=3)}, \quad p(\theta|\alpha) = \frac{\Gamma(\alpha_1 + \alpha_2 + \alpha_3)}{\Gamma(\alpha_1)\Gamma(\alpha_2)\Gamma(\alpha_3)}\theta_1^{\alpha_1 - 1}\theta_2^{\alpha_2 - 1}\theta_3^{\alpha_3 - 1}.$$

1.1 Posterior Distribution

Derive the posterior distribution,

$$p(\theta|y, \alpha).$$

1.2 Marginal Likelihood

Derive the marginal likelihood of y given the hyper-parameters α,

$$p(y|\alpha) = \int p(y, \theta|\alpha)d\theta,$$

Hint: You can use $D(\alpha) = \frac{\Gamma(\alpha_1)\Gamma(\alpha_2)\Gamma(\alpha_3)}{\Gamma(\alpha_1+\alpha_2+\alpha_3)}$ to represent the normalizing constant of the prior and $D(\alpha^+)$ to give the normalizing constant of the posterior.

1.3 Posterior Mean

Compute the posterior mean estimate for θ,

$$\mathbb{E}_{\theta|y,\alpha}[\theta_i] = \int \theta_i p(\theta|y, \alpha)d\theta,$$

which (after some manipulation) should not involve any Γ functions.

Hint: You will also need to use that $\Gamma(\alpha + 1) = \alpha\Gamma(\alpha)$. You may find it a bit cleaner to parameterize the posterior in terms of $\beta_j = I(y = j) + \alpha_j$, and convert back once you have the final result.

1.4 Posterior Predictive

Derive the posterior predictive distribution for a new independent observation $\hat{y}$ given y,

$$p(\hat{y}|y,\alpha) = \int p(\hat{y},\theta|y,\alpha)d\theta.$$

2 Type II Maximum Likelihood

Consider the model

$$y_i \sim \mathcal{N}(w^T\phi(x_i),\sigma^2), \quad w_j \sim \mathcal{N}(0,\lambda).$$

By using properties of Gaussians the marginal likelihood has the form

$$p(y_i|x_i,\sigma,\lambda) = (2\pi)^{-d/2}|C|^{-1/2}\exp\left(-\frac{y^TC^{-1}y}{2}\right),$$

which gives a negative log-marginal likelihood of

$$-\log p(y_i|x_i,\sigma,\lambda) \propto \log|C| + y^TC^{-1}y + \text{const.}$$

where

$$C = \frac{1}{\sigma^2}I + \frac{1}{\lambda}\Phi(X)\Phi(X)^T,$$

As discussed in class, the marginal likelihood can be used to optimize hyper-parameters like σ, λ, and even the basis ϕ.

The demo *example_basis* gives a solution to Question 2.2 of Assignment 1. There, we used a test set to choose the degree of the basis but here we'll use the marginal likelihood. Write a function, *leastSquaresEmpiricalBaysis*, that uses the marginal likelihood to choose the degree of the polynomial as well as the parameters λ and σ (you can assume that all λ_j are equal, and you can restrict your search for λ and σ to powers of 10). Hand in your code and report the marginally most likely values of the degree, σ, and λ. You can use the *logdet* function to compute the log-determinant.

Hint: computing $C^{-1}y$ by explicitly forming C^{-1} may give you numerical issues that lead to non-sensical solution. You can avoid these by using $y^TC^{-1}y = y^Tv$ where v is a solution to $Cv = y$ (Matlab will still give a warning due to ill-conditioning, but it won't return non-sensical results).

3 Neural Network Tuning

In this problem we will investigate handwritten digit classification. The inputs are 16 by 16 grayscale images of handwritten digits (0 through 9), and the goal is to predict the number given the image. If you run `example_neuralNetwork` it will load this dataset and train a neural network with stochastic gradient descent, printing the validation set error as it goes. To handle the 10 classes, there are 10 output units (using a $\{-1,1\}$ encoding of each of the ten labels) and the squared error is used during training. Your task in this question is to modify this training procedure/architecture to optimize performance. Report the best test error you are able to achieve on this dataset, and report the modifications you made to achieve this.

Hint: Below are additional features you could try to incorporate into your neural network to improve performance (the options are approximately in order of increasing difficulty). You do not have to implement

all of these, the modifications you make to try to improve performance are up to you and you can even try things that are not on this list. But, let's stick with neural networks models and only using one neural network (no ensembles).

- Change the network structure: the vector *nHidden* specifies the number of hidden units in each layer.
- Change the training procedure by modifying the sequence of step-sizes or using different step-sizes for different variables. Recall that *momentum* uses the update

$$w^{t+1} = w^t - \alpha_t \nabla f(w^t) + \beta_t (w^t - w^{t-1}),$$

  where α_t is the learning rate (step size) and β_t is the momentum strength. A common value of β_t is a constant 0.9.
- You could vectorize evaluating the loss function (e.g., try to express as much as possible in terms of matrix operations), to allow you to do more training iterations in a reasonable amount of time.
- Add ℓ_2 regularization (or ℓ_1-regularization) of the weights to your loss function. For neural networks this is called *weight decay*. An alternate form of regularization that is sometimes used is *early stopping*, which is stopping training when the error on a validation set stops decreasing.
- Instead of using the squared error, use a softmax (multinomial logistic) layer at the end of the network so that the 10 outputs can be interpreted as probabilities of each class. Recall that the softmax function is

$$p(y_i) = \frac{\exp(z_i)}{\sum_{j=1}^{J} \exp(z_j)},$$

  and you can squared error with the negative log-likelihood of the true label under this loss, $-\log p(y_i)$.
- Instead of just having a bias variable at the beginning, make one of the hidden units in each layer a constant, so that each layer has a bias.
- Implement "dropout", in which hidden units are dropped out with probability p during training, and weights are multiplied by $1 - p$ during testing. A common choice is $p = 0.5$.
- You can do 'fine-tuning' of the last layer. Fix the parameters of all the layers except the last one, and solve for the parameters of the last layer exactly as a convex optimization problem. E.g., treat the input to the last layer as the features and use techniques from earlier in the course (this is particularly fast if you use the squared error, since it has a closed-form solution).
- You can artificially create more training examples, by applying small transformations (translations, rotations, resizing, etc.) to the original images.
- Replace the first layer of the network with a 2D convolutional layer. You will need to reshape the USPS images back to their original 16 by 16 format. The Matlab `conv2` function implements 2D convolutions. Filters of size 5 by 5 are a common choice.