STAT 538 Homework 4
Out February 8, 2012
Due February 16, 2012

©Marina Meilă
mmp@stat.washington.edu

Reading B&V chapters 2.1–2.3, 2.5., 3.1–3.3 For the problems in this homework that deal with convexity, try to find the most elegant solution. Elegant can mean that you give a short proof, based on an example in the textbook, or a property proven in the text, instead of a long proof starting from the definitions.

**Problem 1 – Some sets of probability distributions. B&V problem 2.15**

Only **a, c, e, f, h** . Give short proofs or explanations.

**Problem 2 – Some functions on the probability simplex. B&V problem 3.24**

Only **a, b, e, f**.

**Problem 3 – Log-concavity**

Do one of **BV 3.52, 3.53**

**Problem 4 – Multilayer Neural Network with Backpropagation**

*This problem is self-contained. You do not need knowledge of Neural Networks to solve it.*

Let $g(x) = \frac{1}{1+e^{-z}}$ be the *sigmoid function*, also called the *activation function* of the neural network. Any function that is monotonically increasing and bounded can be used as activation function, but the sigmoid has additional nice computational and statistical properties.

*Bring all your results to the simplest and most interpretable expression.*

**a.** Show that $g'(z) = g(z)(1 - g(z))$

**b.** We build a two-layer neural network with

| | | |
|---|---|---|
| Inputs | $x_k$ | $k = 1 : n$ |
| Bottom layer | $z_j = g(w_j^T x)$ | $j = 1 : m,\ w_j \in \mathbb{R}^n$ |
| Top layer | $f = g(\beta^T z)$ | $\beta \in \mathbb{R}^m$ |
| Output | $f$ | $\in [0, 1]$ |

In other words, the neural network implements the function

$$f(x) \ = \ g\left(\sum_{j=1}^{m} \beta_j z_j\right) \ = \ g\left(\sum_{j=1}^{m} \beta_j g(\sum_j w_{kj} x_k)\right) \tag{1}$$

The loss function we will use is the *logit loss* or *log-likelihood loss* which represents the log-likelihood of the class $y$ under the logistic regression model (1). In other words, assume $f(x)$ represents

$$f(x) \ = \ \hat{P}[Y = +1|x] \tag{2}$$

Assume we have a single observation $(x, y)$. Find the expression of the log-likelihood of this observation as a function of the parameters $\beta, w$. Denote this expression by $L_{logit}(f)$.

You can use the notation $y^* = \frac{1+y}{2}$ which maps $y \in \{\pm 1\}$ to $y^* \in \{1, 0\}$.

**c.** Find the partial derivatives $\frac{\partial L_{logit}}{\partial f}$ and $\frac{\partial L_{logit}}{\partial z_j}$.

**d.** Find the partial derivative $\frac{\partial L_{logit}}{\partial \beta_j}$. Your result should be a function of $y^*, f, z$.

**e.** Now find the partial derivative $\frac{\partial L_{logit}}{\partial w_{kj}}$ as a function of $y^*, x, z, \frac{\partial L_{logit}}{\partial z}, f$.

*Note that in these successive steps we have derived formulas for the gradient of* $L_{logit}$ *w.r.t the parameters* $\beta, w_{1:m}$. *It is a good exercise to actually collect these formulas and write the gradient as a large vector. Another illuminating exercise is to draw a schematic of the calculation of the gradient; the schematic will have a structure similar to the original neural net.*

**f.** The result in **e.** shows that the gradient w.r.t to the $w$ parameters in the second layer can be computed as a function of gradients w.r.t variables in the first layer. Generalize this finding to a multilayer network.

Assume that the network has layers $1, 2, \ldots M$ like this

$$x \equiv x^{(M)} \longrightarrow \boxed{g(x^{(M)}, w^{(M)})} \longrightarrow \ldots x^{(k+1)} \longrightarrow \boxed{g(x^{(k+1)}, w^{(k+1)})} \longrightarrow x^{(k)} \longrightarrow \ldots \longrightarrow x^{(1)} \longrightarrow \boxed{g(x^{(1)}, w^{(1)})} \longrightarrow x^{(0)} \equiv f(x).$$

In the above $x^{(k)} \in \mathbb{R}^{n_k}$, that is layer $k$ has $n_k$ "units" and $w^{(k+1)} \in \mathbb{R}^{n_{k+1} \times n_k}$, in other words, column $j$ of $w^{(k+1)}$ multiplies $x^{(k+1)}$ to produce $x_j^{(k)}$. Note the slight abuse of notation for $g$. Let $n_0 = 1$, meaning that the output of the multilayer neural network is a scalar.

We interpret the output $f(x)$ as in equation (2) and we use the logit loss function as in question **b**.

Derive a recursive formula for the gradient

$$\frac{\partial L_{logit}}{\partial w^{(k+1)}} \tag{3}$$

as a function of variables available at layer $k+1$ or $k$.

*Hint: It is good to think this computation in the following way. When data point $x$ is presented at the input, the values $x^{(k)}$ are computed recursively from $x^{(k+1)}$ in a "forward propagation" from input to output. The intermediate values are saved. Once $f(x)$ is obtained, we can compare with the true $y$ and obtain the cost $L_{logit}(y, f(x))$. Next we need to update the parameters $w$, and for this we will compute the gradient. Now the gradient calculation will proceed from the output layer "backwards" towards the input layer $M$. At each layer the gradient is computed from the values stored during the forward pass and the values calculated at the previous layer. This is the "Backpropagation" algorithm.*

**Extra credit:** Train a two layer neural network to solve the "circle" problem of homework 3. Initialize the $w, \beta$ parameters with random values. Explain why this is a good idea. Explain how you chose $m$, the number of units in the bottom layer.