Name: John Min; jcm2199

COMS 4772 Homework Set 3

(1) Two points on logistic regression

(a) Bishop, ex. 4.14. Show that for a linearly separable data set, the maximum likelihood solution for the logistic regression model is obtained by finding a vector w whose decision boundary $w^T\phi(x) = 0$ separates the classes, and then taking the magnitude of w to infinity. *In optimization, a direction w along which a function remains bounded is known as a direction of recession.*

$p(t|w) = \prod_{n=1}^{N} y_n^{t_n}(1-y_n)^{1-t_n}$
$p(C_1|\phi) = y(\phi) = \sigma(w^\top\phi)$
$p(C_2|\phi) = 1 - \sigma(w^\top\phi)$.
For $p(C_1|\phi) = p(C_2|\phi) = 0.5$, $\log\left(\frac{\sigma(w^\top\phi)}{1-\sigma(w^\top\phi}\right) = 0 \Rightarrow \log(e^{w^\top\phi}) = 0 \Rightarrow w^\top\phi = 0$.

(b) Bishop, ex. 4.15, modified. Show that the Hessian matrix H for the logistic model, given by

$$H = \sum_{n=1}^{N} y_n(1-y_n)\phi_n\phi_n^T$$

is always positive semidefinite. What is a simple condition on the data that guarantees it is positive definite?

Since $y_n \in \{0,1\}$, $y_n(1-y_n) \geq 0$. The outerproduct, by definition, $\phi_n\phi_n^\top \geq 0$.
Thus, H is positive semi-definite.
Since $0 < y_n < 1$, for H to be positive definite, we see that the nullspace of the outerproduct, $\phi_n\phi_n^\top$, must be null. This occurs when the data is linearly independent and X, the data matrix is of full rank.

(2) Fun with Neural Nets: please download Le Roux's code nnetLib from the class website. The main file, `demo.m`, runs classification on the MNIST dataset (hand-written digits 0-9). The data file it uses, `mnist_small.mat`, has 12000 examples, with feature length 784. The code splits these into training and testing, runs the training, and then evaluates the test error.

There are several important switches:

- `params.Nh` in line 13 controls the number of hidden nodes.
  - `params.Nh = []` (meaning 'empty' in matlab) sets up the 10-class classification problem.
  - `params.Nh = 200` sets up a neural net with one hidden layer of size 200.
  - `params.Nh = [100 100 100]` sets up a neural net with three hidden layer of size 100.
- `params.nIter` controls the number of total iterations. 10 was set for demo; I changed it to 20. If you want to push the code with more layers, and larger layer sizes, you may want more iterations.
- `params.cost` in line 16 lets you pick the cost function to use, from {mse, ce, nll, class}.

(a) Show that a neural network with no hidden layer reduces to the multinomial logistic classification model (this should be straightforward using our notes).

A neural network with no hidden layer using the Negative Log-Likelihood loss is equivalent to the softmax loss. Thus, the net is equivalent to the multinomial logit.

$E(w) = -\sum_{n=1}^{N}\sum_{k=1}^{K} y_n^{(k)} \ln \hat{y}_n^{(k)}$ where $\hat{y}_n^{(k)} = \sigma(w^\top x)$. Set $\nabla E(w) = 0$.
We achieve the softmax result.

(b) The three cost functions {mse, ce, nll} are options in the function `computeCost.m`. The code processes the data in batches of size 20, and labels are given as elementary vectors of length 10. Therefore, both `output` and `labels` are always matrices of size 20×10. Only one entry in each row of `labels` is nonzero.

Given this information, study the code to write down formulas for the three objectives in `computeCost`. *My advice is to not think look at their names, but just to focus on the code.*

Let W be the weight decays cost, $\hat{y}$ be the predicted output, y label.

- MSE: $\frac{1}{2}\sum_{n=1}^{N}\sum_{k=1}^{K}\big(y_n^{(k)} - \hat{y}_n^{(k)}\big)^2 + W$
- CE: $-\sum_{n=1}^{N}\sum_{k=1}^{K}\Big[\hat{y}_n^{(k)}\mathbb{1}[y_n = k] + \sigma\big(-\hat{y}_n^{(k)}\big)\Big]^2 + W$
- NLL: $\sum_{n=1}^{N}\sum_{k=1}^{K}\Big[-\log\Big(\text{softmax}(\hat{y}_n^{(k)}\mathbb{1}[y_n = k])\Big)\Big] + W$

(c) Two of the three objectives should be familiar. What about the third? What is it trying to do? Do your best to give some intuition; you will get credit as long as you show you seriously considered it.

In class, we have discussed the mean squared error and negative log-likelihood loss functions. Here, we are introduced to the cross-entropy loss function, which we can see to be quite similar to the negative log-likelihood. What we can see is that the distinction between the CE and NLL loss functions is analogous to the slight difference between the sigmoid and softmax loss functions – we can summarze the two different multinomial classification approaches to being one-vs-one and one-vs-rest, respectively, for the cross-entropy and negative log-likelihood objective functions.

(d) Numerical evaluations. The code uses stochastic gradient; I have therefore fixed the random seed (line 2 in `demo.m`), to make sure you get the same output every time you run the code. Please do the following comparisons. Report each in a table. I made the tables for you, you just have to fill them in.
Whatever you decide for the iteration number (at least 20) please keep it the same for all experiments. This way we can at least compare all the methods

by computational effort, since we will not know that we solved the optimization problem fully.

- Test errors for multinomial regression vs. 1-layer network with different hidden layer sizes, across three objective functions:

	MSE	CE	NLL
`params.Nh = []`	21%	8.7%	8.1%
`params.Nh = 100`	5.4	5.3	5.7
`params.Nh = 200`	5.05	4.75	5.5
`params.Nh = 300`	5.1	4.8	5.5

- Test errors for multiple layers of the same size, across three objective functions:

	MSE	CE	NLL
`params.Nh = 100`	5.4%	5.3%	5.7%
`params.Nh = [100 100]`	4.85	4.85	5.15
`params.Nh = [100 100 100 ]`	5.4	4.55	5.25
`params.Nh = [100 100 100 100]`	5.4	4.45	4.95

- Summarize what you learned from the two experiments. Set up an experiment with your choice of layer size, layer number, and objective function, that beats the best test error you have seen so far, and report the error and the setup. Keep in mind the hidden layers don't need to be all the same size.

I have chosen the number of iterations to be 50. There are several lessons to be learned. The first is that optimizing for the different objective functions produce similar results. Optimizing on cross-entropy (CE) seemed to produce the best test outcomes. The second is that overfitting the model on the training data set is quite easy and will bump up the test error. Overfitting can come from adding more layers, increasing layer size, or performing a large number of iterations. The final consideration should take into account model training/compuation time as a legitimate trade-off versus model performance and it may be possible to train a well-performing model without constructing an extravagantly large neural net.

	CE
params.Nh = [50 50 50 50 50 50]	5.45%
params.Nh = [50 50 50 50 50 50 50]	5.45
params.Nh = [50 100 100 100 100]	5.5
params.Nh = [100 100 100 100 100]	**3.9**
params.Nh = [100 100 100 100 100 100]	**3.95**
params.Nh = [100 200]	4.9
params.Nh = [100 300]	4.9
params.Nh = [200 200]	4.5
params.Nh = [200 200 100 100]	**4.1**
params.Nh = [200 300]	**3.95**
params.Nh = [200 300 100 100]	4.3
params.Nh = [200 400]	4.4
params.Nh = [300 300]	4.45
params.Nh = [300 200]	4.55

(3) Bonus. Modify the `demo` file to contaminate a portion of your training data, by flipping a percentage of the labels.

(a) Evaluate the effect of proportion of flipped labels on the testing error:

	MSE	CE	NLL
`params.Nh = [], 0% contaminated`	21%	8.7%	8.1%
`params.Nh = [], 1% contaminated`	20.6	8.95	8.4
`params.Nh = [], 5% contaminated`	20.95	9.4	10.05
`params.Nh = [], 10% contaminated`	21.05	10.6	9.95
`params.Nh = [], 25% contaminated`	20.9	12.2	11.65
`params.Nh = 100, 0% contaminated`	5.4	5.3	5.7
`params.Nh = 100, 1% contaminated`	5.9	5.55	5.3
`params.Nh = 100, 5% contaminated`	5.5	5.1	5.4
`params.Nh = 100, 10% contaminated`	5.85	6.3	5.85
`params.Nh = 100, 25% contaminated`	6.25	7.1	7.15

(b) Add two robust options to the `computeCost.m` file (analogous to MSE), one using the Huber function, and one using the Student's t log-likelihood:

$$f(r) = \sum \ln(\nu + r_i^2)$$

(c) Compare your robustified versions on the experiment you developed:

	MSE	huber MSE	student MSE
`params.Nh = [], 0% contaminated`	21 %	31.3%	27.5 %
`params.Nh = [], 1% contaminated`	20.6	32	31.05
`params.Nh = [], 5% contaminated`	20.95	28.1	33.35
`params.Nh = [], 10% contaminated`	21.05	37.4	28.75
`params.Nh = [], 25% contaminated`	21.09	30.45	27.55
`params.Nh = 100, 0% contaminated`	5.4	6.15	8.65
`params.Nh = 100, 1% contaminated`	5.9	5.7	8.55
`params.Nh = 100, 5% contaminated`	5.5	8.95	5.8
`params.Nh = 100, 10% contaminated`	5.85	9.75	5.65
`params.Nh = 100, 25% contaminated`	6.25	14	5.95

(d) Did the robust measures help? How did they fare compared to the ML estimators for contaminated data?

For the Huber loss, I chose $\delta = .25$. For the student t, I decided on $\nu = 1$.

As the contamination factor increased, the robust methods performed relatively better. Overall, the robust measures, did not seem to significantly help, but I did not thoroughly explore the parameter space with respect to these alternative mean squared error formulations - whether it be the "delta" parameter for the Huber loss or the degrees of freedom ν parameter for the

student t log-likelihood.

While more investigation needs to be done with regards to this experiment, it is evident that the MSE losses do not perform as well as the Cross-Entropy and Negative Log-Likelihood formulations of the losses.

```
function [errors, gradient] = computeCost(layers, labels, cost)

output = layers(end).output;
[nSamples nLabels] = size(output);
nLayers = length(layers);

% Compute the cost due to weight decays.
if isfield(layers(1), 'wdCost')
        wdCost = sum([layers.wdCost]);
else
        wdCost = 0;
end

switch cost
case 'mse'
        diff = output - labels;
        errors = .5*sum(diff.^2,2) + wdCost;
        gradient = diff/nSamples;
case 'ce'
        errors = -sum(output.*labels - log(1 + exp(output)), 2) + wdCost;

        % Avoid overflows.
        errors(output > 20) = -sum(output(output > 20).*labels(output > 20) - o
        gradient = -(labels - sigm(output))/nSamples;

case 'nll'
        softmaxOutput = softmax(output, 2);
        errors = sum(-log(softmaxOutput).*labels,2) + wdCost;
        gradient = (- labels + softmaxOutput)/nSamples;
case 'class'
        if size(output, 2) == 1
                errors = ( sign(output) ~= (2*labels-1) );
        else
                [~, valueOutput] = max(output, [], 2);
                errors = ( valueOutput ~= labels*(1:nLabels)' );
                if nargout > 1, error('This cost is not designed for training')
    end
```

```
case 'huber mse'
    diff = output - labels;
    delta = .25;
    errors = sum(arrayfun(@Huber, diff));
    gradient = delta*sign(diff)/nSamples;

case 'student mse'
    v = 1; %degrees of freedom: n-1
    diff = output - labels;
    errors = .5*sum(log(v+diff.^2),2) + wdCost;
    gradient = 2*(diff./(v+diff.^2))/nSamples;
end
end

function [a] = Huber(a)
    delta = .25;
    if abs(a) <= delta
        a = .5*a^2;
    else
        a = delta*(abs(a) - delta/2);
    end
end
```