CSE 546 Homework #1
Fall 2016

Janet Matsen

October 31, 2016

For this assignment, I went to office hours and developed strategies with Serena Liu and David Flemming.

1 Programming: Multi-Class Classification using Least Squares

1.1 One vs all classification

1.1.1 Separate binary classifiers vs one multiclass system.

For one binary ridge regression solution, we had $\hat{w}_{ridge} = (H^T H + \lambda I_d)^{-1} H^T y$.
For this problem, X is not transformed into a different hypothesis space H.
Recall that computing the dot product of a $(n \times m)$ matrix with an $(m \times p)$ matrix is $\mathcal{O}(nmp)$.
Also, inverting a $(d \times d)$ matrix is $\mathcal{O}(d^3)$.

We can count up the computations.
Dotting H^T with H is $(d \times N)(N \times d) \to \mathcal{O}(Nd^2)$
Adding λI_{0+k} is negligible.
Inverting $(H^T H + \lambda I_{0+k})$ is inverting a $(d \times d)$, which is $\mathcal{O}(d^3)$
Multiplying that $(d \times d)$ by the $(d \times N)$ H^T is $\mathcal{O}(Nd^2)$
So the complexity of $(H^T H + \lambda I_{0+k})^{-1} H^T$ is $\mathcal{O}(Nd^2 + d^3 + Nd^2 + Nd)$

Multiplying by y again at the end is multiplying $(d \times N)$ by an $(N \times 1)$, which is $\mathcal{O}(Nd)$

Add this up, and we see that computing $\hat{w}$ has complexity
$\mathcal{O}(Nd^2 + d^3 + Nd^2 + Nd) \to \mathcal{O}(Nd^2 + d^3 + Nd^2)$.
If we want to do the predictions, then we are multiplying X by w, which is a $(N \times d)(d \times N)$ multiplication, and thus adds $\mathcal{O}(N^2 d)$.

So the overall complexity is $\mathcal{O}(N^2 d + Nd^2 + d^3)$

1.1.2 When you solve k problems jointly, the compute time is:

We can re-use the $H(H^T H + \lambda I_{0+k})^{-1} H^T$ computation; it only needs to be computed once and can be shared across the k classifiers.
We said earlier that computing $(H^T H + \lambda I_{0+k})^{-1} H^T \to \mathcal{O}(N^2 d + Nd^2 + d^3)$.
So $H(H^T H + \lambda I_{0+k})^{-1} H^T = \mathcal{O}(2N^2 d + Nd^2 + d^3) \to \mathcal{O}(N^2 d + Nd^2 + d^3)$.

Multiplying $H(H^T H + \lambda I_{0+k})^{-1} H^T$, which is $(N \times N)$, by y, which is $(N \times 1)$ is an $\mathcal{O}(N^2)$ operation that happens k times.
Our un-slick strategy sums to $\mathcal{O}(N^2 d + Nd^2 + d^3 + kN^2)$
This definitely saves compute time over solving the k models separately.

1.1.3 Run this classification algorithm on all 10 classes with appropriate regularization as needed.

Did a parameter sweep, holding out 10% of the training data for validation.

Figure 1: Regularization path for multi-class Ridge Regression. Lead to choosing $\lambda = 1e7 = 10,000,000$.

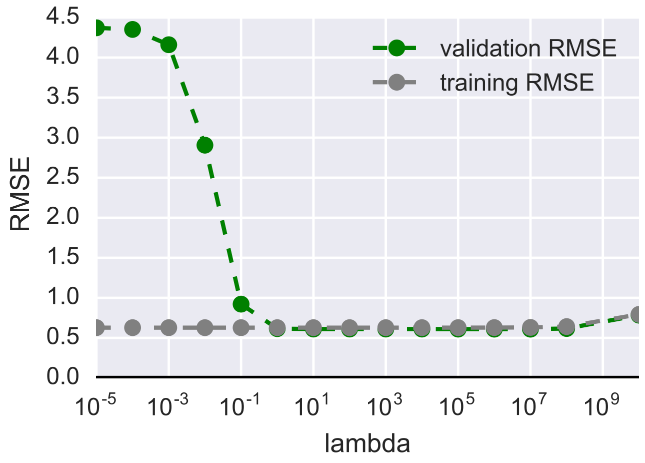
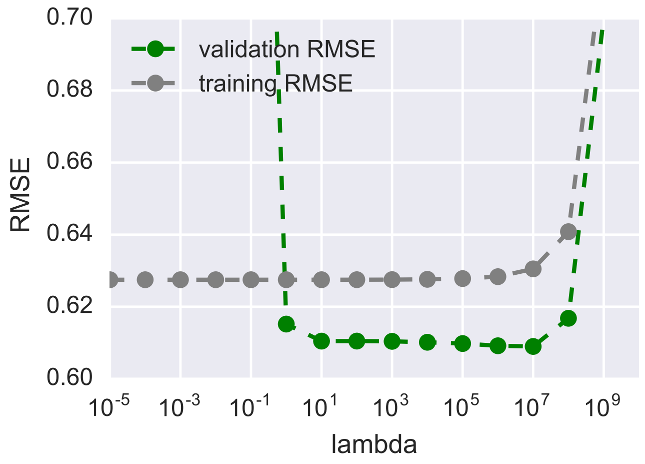


Figure 2: Zoomed in version of Figure ??



Chose $\lambda = 1e7$. Trained again, using 100% of the training data. Used these new weights on the unseen test data.

Test Set	SSE	RMSE	0/1 loss	(0/1 loss)/N
Training	21466.4	0.6305	8171	0.1513
Test	3938	0.6272	1425	0.1425

Table 1: Results for multi-class classification, using $\lambda = 1e7$

The accuracy on all this multi-class classifier is $(1 - 0.151315)$, which is about 85% .

In class on 10/26, Sham coached us to include errors for all classifications in our RMSE (and thus SSE) calculations, not just the error for the correct class.
For a toy example to illustrate my calculations,
let y = [0, 1], Y=[[0, 1], [1, 0]], Yhat = [[0.01, 0.95], [0.99, 0.03]],
Then SSE = sum(0.01**2 + 0.05**2 + 0.01**2 + 0.03**2) = RSS
Note: this would not be equivalent to the binary classifier, which would only sum (0.05**2 + 0.03**2)

1.2 Neural nets with a random first layer: Using more features.

Each v_i is a vector in a high-dimensional space. When we do $v_j \cdot x$, we are getting the component of that x vector that is parallel to v_j. I was unable to run my algorithm with the $(10,000 \times 10,000)$ matrix inversion, even on a powerful machine.

Recalling kernel tricks from CSE 446, I found a kernel trick for ridge regression: `http://www.ics.uci.edu/~welling/classnotes/papers_class/Kernel-Ridge.pdf`
In our notation, $H^T(HH^T + \lambda I_N) = (\lambda I_d + H^T H)H^T$.
Applying this to our ridge equation, $w = H^T(HH^T + \lambda I_d)^{-1}y$. Now we are inverting an $(N \times N)$ matrix, instead of a $(d \times d)$ one. This kernel trick is only useful when the amount of data is less than the number of features. Therefore my plan was to split my training data into 30 subsets with N=2000 each. After training these 30 models, I planned to aggregate the individual classifiers by majority vote.

I did not have time to implement this plan, but submit code demonstrating efficacy for one of these classifiers. The matrix inversion step took only about 0.1 seconds. For the single classifier I trained using 2000 points with $\lambda = 100$, the resulting (0/1 loss/N) was 0.24. I would expect much better results than that when aggregating all 60 classifier's results. See code: `Q-1-2_multiclass_kernel_trick.ipynb` or code: `Q-1-2_multiclass_kernel_trick.py`

2 Programming: Multi-Class Classification using Logistic Regression and Softmax

2.1 Binary Logistic Regression

My binary logistic regression model was fit without bias, and using C=10 regressors rather than C-1 as the slides in class showed. My lambda was normalized by $||X||_2/N$.

I used my HyperparameterExplorer class to find optimal λ and η before training on all of the training model. As usual, I split off 10% of my training data for validation. Models were fit at each λ using an initial η that was about as large as could be used without divergence. The η values decayed with $1/\sqrt{s}$ where s is the number of steps minus then number corresponding to fast convergence (log loss increasing $>$ 10% each time). The input λ values were normalized by $||X||_2/N$. Aware of the fact that a model may not converge on the optimum for each run, I repeatedly trained subsequent models at a given λ using the previous solution as seed weights. I moved on to the next λ once the repeated training of models at the current λ stopped leading to improvements in the log loss.

2.1.1 Binary Logistic: What learning rate did you use?

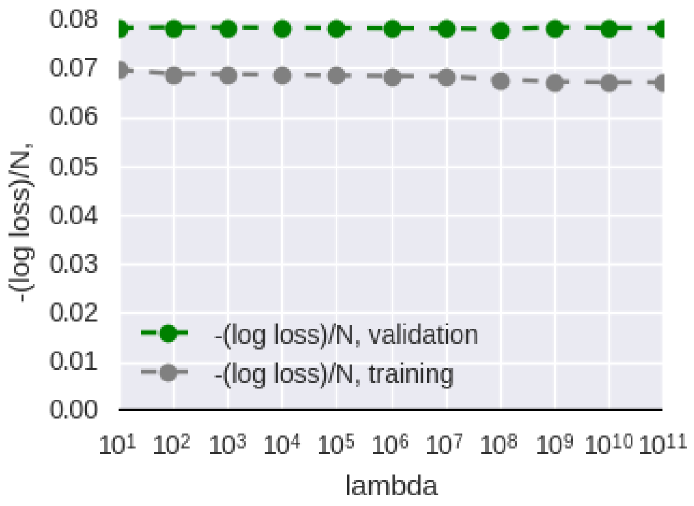


Figure 3: Regularization path for binary logistic regression, showing only weak dependence of validation fits on regularization strength. Lead to choosing $\lambda = 1e8$ (which when divided by $||X||_2$ is much smaller.)

I used $\lambda = 1e8$, which when normalized by $||X||_2/N$ is 180.1. My ν was initialized at 0.0002, is then scaled by multiplying by $1/N$. Thus the effective which decayed to $2.2e-10$ after 510 iterations. Note that I initialized my model's weights using the hyperparameter sweep's best weights, so the number of steps was small.

2.1.2 Log loss as a function of iteration for train and test set?

These results show the final fit, which included training my model on all 60,000 training points with an initial W matrix set to the best one seen in my hyperparameter sweep. Note that the test data was not used during model fitting.

Figure 4: Normalized log loss of my final model fit, after seeding with the previous best weights from the hyperparameter sweep. Note this path to convergence is much more efficient than if I had seeded with a matrix of zeros.

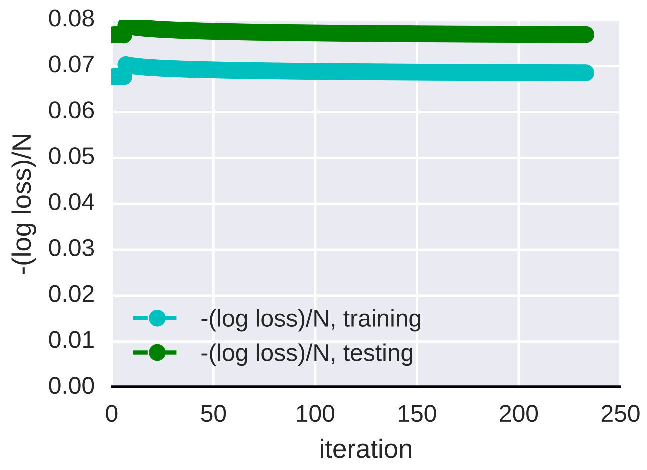
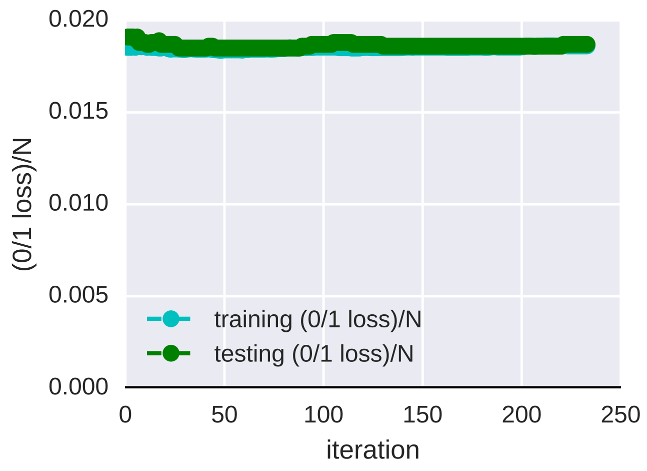


Figure 5: Normalized 0/1 loss of my final model fit, after seeding with the previous best weights from the hyperparameter sweep. Note this path to convergence is much more efficient than if I had seeded with a matrix of zeros.



2.1.3 For the 0/1 loss, use a threshold of 0.5 for classifying if a digit is a 2. On the training and test set, what are the final log losses and 0/1 loss?

Test Set	Log Loss	-(Log Loss)/N	0/1 loss	(0/1 loss)/N
Training	-4111.2	0.0685	1117	0.0186
Test	-768.4	0.0768	187	0.0187

Table 2: Question 2.1.3 results. We did much better with logistic regression than ridge regression.

2.2 Softmax Classification: gradient descent

2.2.1 Write out the derivative of the soft-max function.

Let $v_j = exp(w^{(j)} \cdot x)$, $v_\ell = exp(w^{(\ell)} \cdot x)$.
Chain rule:

$$\frac{\partial}{\partial w^{(i)}} \log Pr(v_j(w^{(i)}) = \left[\frac{d}{dv_j} \log Pr(v_j(w^{(i)}))\right] \left[\frac{d}{dw^{(i)}} v_j\right]$$

Do the first term:

$$\begin{aligned}
\frac{d}{dv_j} \log Pr(v_j) &= \frac{d}{dv_j} \log \left[\frac{\exp(v_\ell)}{1 + \sum_{i=1}^{k-1} \exp(v_i)}\right] \\
&= \frac{d}{dv_j} \log(\exp(v_\ell)) - \frac{d}{dv_j} \log(1 + \sum_{i=1}^{k-1} \exp(v_i)) \\
&= \frac{1}{(1 + \sum_{i=1}^{k-1} \exp(v_i))} \frac{d}{dv_j}(1 + \sum_{i=1}^{k-1} \exp(v_i)) \\
&= \frac{1}{(1 + \sum_{i=1}^{k-1} \exp(v_i))} \exp(v_j)) \\
&= \mathbb{1}(j = \ell) Pr(Y = \ell | x, w)
\end{aligned}$$

Do the second term:

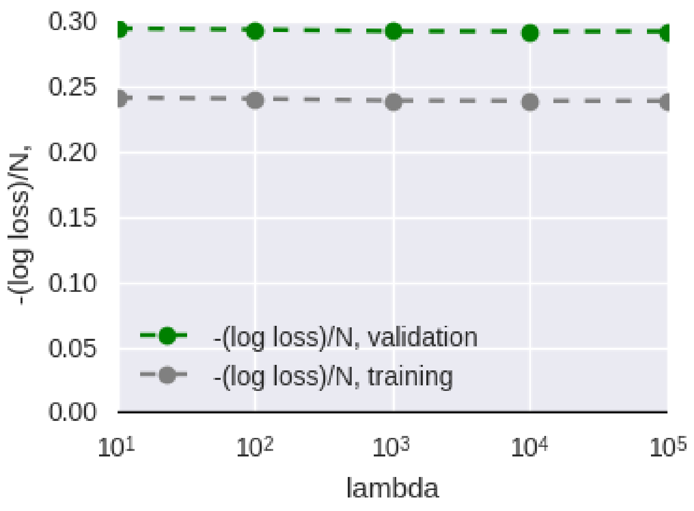
$$\begin{aligned}
\frac{d}{dw^{(\ell)}} v_j &= \frac{d}{dw^{(\ell)}} \sum_{z=1}^{d} w_z^{(j)} \cdot x_j \\
&= \frac{d}{dw^{(\ell)}} w^{(\ell)} \cdot x_j \\
&= x_\ell
\end{aligned}$$

Put them together:

$$\frac{\partial}{\partial w^{(i)}} \log Pr(v_j(w^{(i)}) = \left[\frac{d}{dv_j} \log Pr(v_j(w^{(i)}))\right]\left[\frac{d}{dw^{(i)}} v_j\right]$$
$$\frac{\partial}{\partial w^{(i)}} \log Pr(v_j(w^{(i)}) = x_\ell \left[\mathbb{1}(j = \ell) Pr(Y = \ell | x, w)\right]$$

2.2.2 Implement batch gradient descent. What learning rate did you use?

Figure 6: I did not see sensitivity to λ for $\lambda < 10^5$. My computer crashed before I could run higher values. See `Q-2-2-REG-PATH-SoftMax-Logistic--batch_grad_desc.pdf` for development of this graph.



My hyperparameter exploration showed that a good learning rate was $1e - 4$.

2.2.3 Plot log loss as a function of the iteration number: include both the training and test losses in the same plot. Also do for 0/1 loss.

After my hyperparameter exploration, I trained on all of the data. Both log-loss and 0/1 loss dropped very quickly.

Figure 7: I did not see sensitivity to λ for $\lambda < 10^5$. My computer crashed before I could run higher values. See for development of this graph.

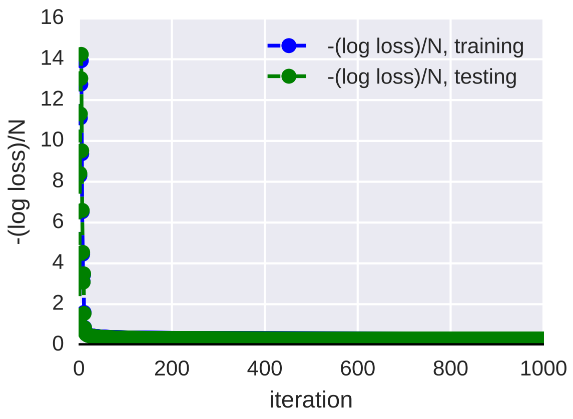
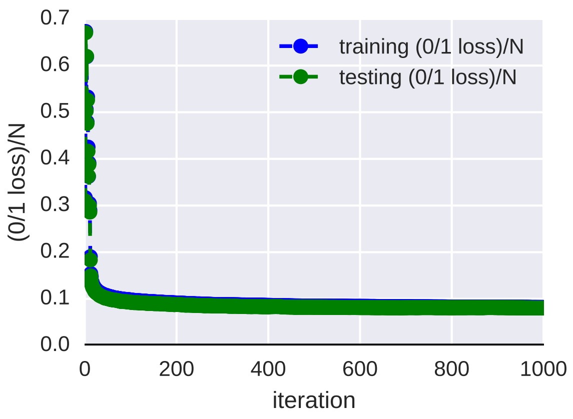


Figure 8: I did not see sensitivity to λ for $\lambda < 10^5$. My computer crashed before I could run higher values. See for development of this graph.



2.2.4 On the training set & test set, what are your final log losses and 0/1 losses?

TODO: make the tables go the right places!!

Test Set	Log Loss	-(Log Loss)/N	0/1 loss	(0/1 loss)/N
Training	-15807.1	0.263	4316	0.0719
Test	-2751.1	0.275	764	0.0764

Table 3: Question 2.2.4 results: multiclass batch gradient descent on MNIST

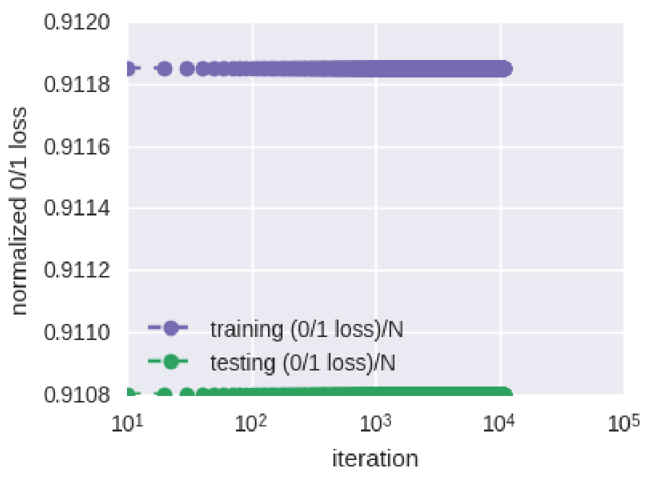
2.3 Softmax classification: stochastic gradient descent

2.3.1 SGD with mini-batch size 1. After every 15,000 points, plot log loss vs iteration #.

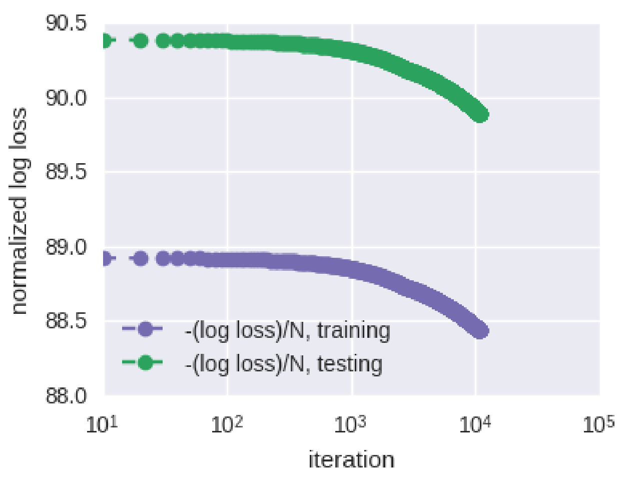
I didn't have time to run my algorithm on the full training set, but have these results using 3,000 training points. I used $\lambda = 1$, $\eta 0 = 0.1$, and 1000 passes through the entire data set due to time constraints. After deciding I didn't have time for more explorations of hyperparameters, I ran my final training using the previous model's weights and it passed through the data 5 times more.

Figure 9: Minibatch size 100. Each iteration corresponds to training on an additional 100 points.

```
model.plot_test_and_train_01_loss_during_fitting()
```



```
model.plot_test_and_train_log_loss_during_fitting()
```



2.3.2 Compare the complexity it took batch and semi- reach comparable 0/1 losses on the training sets.

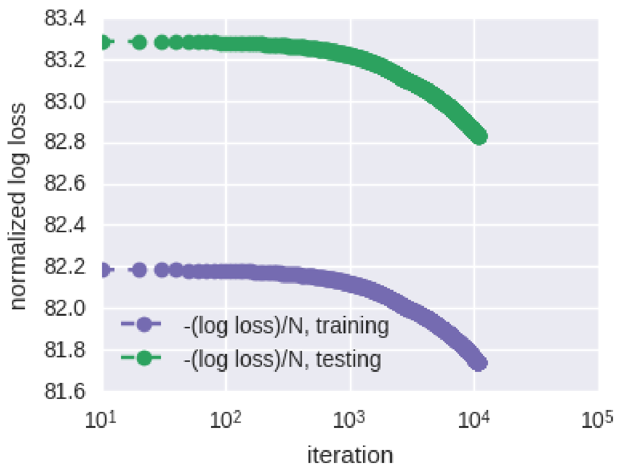
The efficacy will depend on the extent to which you consider each model to have been "converged". This model did appear to be converging faster than the other models I was training. I can say that this model converges faster than the minibatch size of 100. One x-axis unit in the mini-batch size 1 plot corresponds to 100 in the plot below.

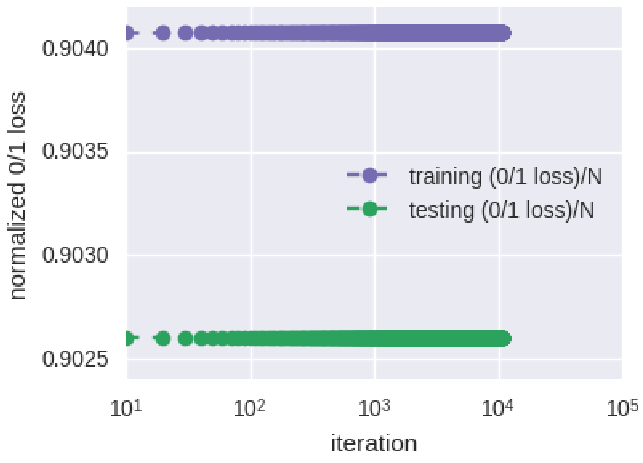
2.3.3 Implement SGD with mini-batch size of 100. Show plots.

Like in Q-2.3.1, I didn't have time to run my algorithm on the full training set, but have these results using 3,000 training points. I ran at the same conditions: $\lambda = 1$, $\eta 0 = 0.1$, and only 5 passes through the entire data set due to time constraints. I also trained a final training using the previous model's weights and it passed through the data 5 times more.

The learning rate for both training sessions was 0.01, with decay of $1/\sqrt{s}$ where s are the number of steps since the learning rate slowed down below 2%.

Figure 10: Minibatch size 1. Each iteration corresponds to training on an additional 1 point.





2.3.4 Compare mini-batch to just using one point at a time, in terms of total computational time.

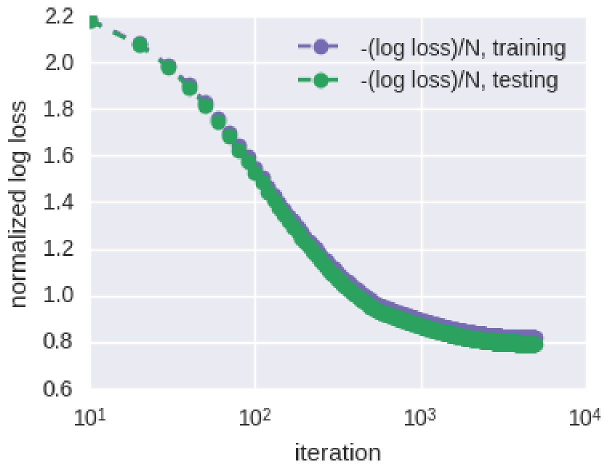
One axis point in this plot corresponds to 100 axis points for mini-batch size 1. Though we didn't converge on an optimal solution with the limited run time, we can certainly see that mini-batch size 1 is much more efficient per pass over N data points.

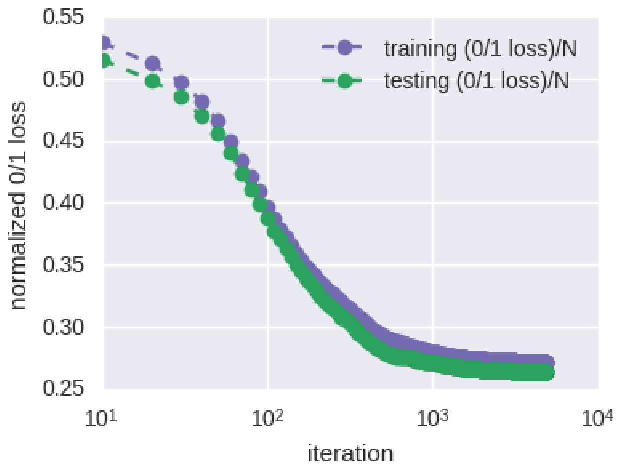
2.4 Neural Nets with a random first layer: Using more features.

2.4.1 Run your favorite algorithm from the previous softmax experiments on the k-dimensional feature vector with appropriate regularization as needed.

Regularization: I only had time to use $\lambda = 0$, but the curve didn't seem to overfit on the 10% of training data I held out.

Figure 11: Model fitting for $\lambda = 0$ using SGD on the "more feature" data.





2.4.2 On the training and test sets, what are the losses?

I didn't have time to tune hyperparameters, or train very long, but got these results for $\lambda = 0$, batch size $= 100$, $\eta 0 = 0.005$, and passing over 7260 points:

Test Set	Log Loss	-(Log Loss)/N	0/1 loss	(0/1 loss)/N
Training	-39988.1	0.666	12823	0.2137
Test	-6623.07	0.662	2130	0.213

Table 4: Table of results

3 Baby learning Theory

3.1 Confidence interval for a coin

$$\begin{aligned}
Pr(|\bar{Z}-\theta| \geq \epsilon) &\leq 2\exp(-2N\epsilon^2) \\
1 - Pr(|\bar{Z}-\theta| \leq \epsilon) &\leq 2\exp(-2N\epsilon^2) \\
-Pr(|\bar{Z}-\theta| \leq \epsilon) &\leq -1 + 2\exp(-2N\epsilon^2) \\
Pr(|\bar{Z}-\theta| \leq \epsilon) &\geq 1 - 2\exp(-2N\epsilon^2)
\end{aligned}$$

$Pr(|\bar{Z}-\theta| \leq \epsilon)$ defines the confidence interval, and we want it to hold with probability $1-\delta$

$$\begin{aligned}
1 - 2\exp(-2N\epsilon^2) &= 1-\delta \\
\exp(-2N\epsilon^2) &= \delta/2 \\
-2N\epsilon^2 &= \ln(\delta/2) \\
\epsilon^2 &= -\frac{\ln(\delta/2)}{2N} \\
\epsilon &= \sqrt{-\frac{\ln(\delta/2)}{2N}}
\end{aligned}$$

$\bar{Z}$ is has probability greater than $1-\delta$ of being in the interval $\left[\theta - \sqrt{-\frac{\ln(\delta/2)}{2N}}, \theta + \sqrt{-\frac{\ln(\delta/2)}{2N}}\right]$.

3.2 Estimating performance of a given classifier

The classifier's loss is $\frac{1}{N}\sum_{i=1}^{N} \mathbb{1}(f(x) \neq Y)$, the average number of points that were classified incorrectly.
Thus $L(f)$ is a random variable, just like $\bar{Z}$, except that it depends on X and y instead of the Z training data.
$\hat{L}(F)$ is analogous to θ, in that it is the best value that can be seen with an infinite amount of training data.
This means the confidence interval from Question 3.1 holds. $|L(f) - \hat{L}(f)| \leq \sqrt{-\frac{\ln(\delta/2)}{2N}}$

3.3 ERM revisited

3.3.1 f^* confidence interval

The confidence interval we stated earlier relies on i.i.d. samples. The estimator f^* is defined with respect to the underlying distribution, not a particular sample. Thus the confidence interval holds. $|L(f^*) - \hat{L}(f^*)| \leq \sqrt{-\frac{\ln(\delta/2)}{2N}}$

3.3.2 $\hat{f}$ confidence interval

No, the same loss formula does not hold. $\hat{f}$ differs from f^* in that $\hat{f}$ is chosen based on a finite sample. When we use the estimator formula $\hat{L}(f) = \frac{1}{N}\sum\limits_{i=1}^{N} 1(f(x_i \neq y_i)$, we are putting the same points that the estimator was trained on back into the loss calculations. This violates the i.i.d. assumption of the confidence interval we derived, and invalidates it.

3.3.3 Confidence interval for losses of all f in $\mathcal{F}$ with probability of error δ

Choose the confidence interval $|L(f_i) - \hat{L}(f_i)| \leq \sqrt{\frac{\ln\left(\frac{\delta}{2K}\right)}{-2N}}$.
Define the event $\epsilon_i = |L(f_i) - \hat{L}(f_i)| \leq B/K$.
In Question 3.1, we shoed that $P(\epsilon_i) \geq 1 - \delta/K$, which is equivalent to $P(\text{not } \epsilon_i) = \delta/K$.
If we have K classifiers, $\sum_{i=1}^{k} \leq (\delta/K)K = \delta$
By the union bound, $P(\text{not } \epsilon_1 + \text{ not } \epsilon_2 + \ldots \text{ not } \epsilon_K) \leq \delta$

$$\begin{aligned}
P(\text{not } \epsilon_1 + \text{not } \epsilon_2 + \ldots \text{not } \epsilon_K) &\leq \delta \\
1 - P(\epsilon_1 + \text{ and } \epsilon_2 + \ldots \text{ and } \epsilon_K) &\leq \delta \\
-1 + P(\epsilon_1 + \text{ and } \epsilon_2 + \ldots \text{ and } \epsilon_K) &\geq -\delta \\
P(\epsilon_1 + \text{ and } \epsilon_2 + \ldots \text{ and } \epsilon_K) &\geq 1 - \delta
\end{aligned}$$

$$P\left(\text{for all } f \in F, |L(f) - \hat{L}(f)| \leq \sqrt{\frac{\ln\left(\frac{\delta}{2K}\right)}{-2N}}\right) \geq 1 - \delta$$

This means $B = \sqrt{\frac{\ln\left(\frac{\delta}{2K}\right)}{-2N}}$

3.3.4 Confidence interval for the loss of $\hat{f}$ that holds with probability greater than $1 - \delta$

$\hat{f}$ is an estimator in F, so the bound for Q 3.3.3 holds. $|L(\hat{f}) - \hat{L}(\hat{f})| \leq \sqrt{\frac{\ln\left(\frac{\delta}{2K}\right)}{-2N}}$

3.3.5 R for $L(\hat{f}) - L(f^*) \leq R$

Use our previous findings:
$|L(f^*) - \hat{L}(f^*)| \leq \sqrt{-\frac{\ln(\delta/2)}{2N}}$ (Q 3.3.1)

$|L(\hat{f}) - \hat{L}(\hat{f})| \leq \sqrt{\frac{\ln\left(\frac{\delta}{2K}\right)}{-2N}}$ (Q 3.3.4)

Also used:

- the general rule that if $|a-b| \leq c$, $a \leq c+b$, and $b \leq c+a$.

- the fact that $\hat{L}(\hat{f}) \leq \hat{L}(f^*)$, because $\hat{f}$ was selected to minimize $\hat{L}(\hat{f})$

$$\begin{aligned} |L(\hat{f}) - \hat{L}(\hat{f})| &\leq \sqrt{\frac{\ln\left(\frac{\delta}{2K}\right)}{-2N}} \\ L(\hat{f}) &\leq \sqrt{\frac{\ln\left(\frac{\delta}{2K}\right)}{-2N}} + \hat{L}(\hat{f}) \\ L(\hat{f}) &\leq \sqrt{\frac{\ln\left(\frac{\delta}{2K}\right)}{-2N}} + \hat{L}(f^*) \end{aligned}$$

Make a new expression for $\hat{L}(f^*)$ to substitute in:

$$\begin{aligned} |L(f^*) - \hat{L}(f^*)| &\leq \sqrt{-\frac{\ln(\delta/2)}{2N}} \\ \hat{L}(f^*) &\leq \sqrt{-\frac{\ln(\delta/2)}{2N}} + L(f^*) \end{aligned}$$

Combine expressions:

$$\begin{aligned} L(\hat{f}) &\leq \sqrt{\frac{\ln\left(\frac{\delta}{2K}\right)}{-2N}} + \hat{L}(f^*) \\ L(\hat{f}) &\leq \sqrt{\frac{\ln\left(\frac{\delta}{2K}\right)}{-2N}} + \sqrt{-\frac{\ln(\delta/2)}{2N}} + L(f^*) \\ L(\hat{f}) - L(f^*) &\leq \sqrt{\frac{\ln\left(\frac{\delta}{2K}\right)}{-2N}} + \sqrt{-\frac{\ln(\delta/2)}{2N}} \end{aligned}$$

$R = \sqrt{-\frac{\ln(\delta/2)}{2N}} + \sqrt{\frac{\ln\left(\frac{\delta}{2K}\right)}{-2N}}$
$R = \sqrt{\frac{1}{2N}}\left(\sqrt{-\ln(\delta/2)} + \sqrt{-\ln(\delta/(2K))}\right)$

3.3.6 How large a hypothesis class can we utilize?

3.3.6.1 If K is a constant:

If K is a constant, we can see from the formula $R = \sqrt{\frac{1}{2N}}\left(\sqrt{-\ln(\delta/2)} + \sqrt{-\ln(\delta/(2K))}\right)$ that $\left(\sqrt{-\ln(\delta/2)} + \sqrt{-\ln(\delta/(2K))}\right)$ is a constanat and thus as $N \to \infty$, $R \to 0$.
So yes, we can learn pretty well.
The regret, R, is $\mathcal{O}(\sqrt{\frac{1}{N}})$

3.3.6.2 If $K = N^p$:

$R = \sqrt{\frac{1}{2N}}\left(\sqrt{-\ln(\delta/2)} + \sqrt{-\ln(\delta/(2N^p))}\right)$
We can ignore constants when determining Orders.

$$\begin{aligned}
R &\to \sqrt{\frac{1}{N}}\sqrt{-\ln(\delta/(N^p))} \\
&\to \sqrt{\frac{-\ln(\delta/(N^p))}{N}} \\
&\to \sqrt{\frac{-(\ln(\delta) - p\ln(N))}{N}} \\
&\to \sqrt{\frac{p\ln(N)}{N}} \\
&\to \sqrt{\frac{\ln(N)}{N}}
\end{aligned}$$

The regret, R, is $\mathcal{O}(\sqrt{\frac{\ln(N)}{N}})$.
We learn less efficiently than the previous case, but this still tends to 0 as $N \to \infty$.

3.3.6.3 If $K = \exp\sqrt{N}$:

$$\begin{aligned}
R &\to \sqrt{\frac{1}{2N}}\left(\sqrt{-\ln(\delta/2)} + \sqrt{-\ln(\delta/(2\exp\sqrt{N}))}\right)\\
&\to \sqrt{\frac{1}{N}}\sqrt{-\ln(\delta/(2\exp\sqrt{N}))}\\
&\to \sqrt{\frac{-\ln(\delta/(2\exp\sqrt{N}))}{N}}\\
&\to \sqrt{\frac{-\ln(\delta) + 2\ln\exp\sqrt{N}}{N}}\\
&\to \sqrt{\frac{\sqrt{N}}{N}}\\
&\to N^{1/4}
\end{aligned}$$

The regret, R, is $\mathcal{O}(N^{1/4})$.
With this larger model class, it takes much more training points to get a regret as small as for the prior classes.

3.3.6.4 If $K = \exp 10N$:

$$\begin{aligned}
R &\to \sqrt{\frac{-\ln(\delta/(2\exp 10N)}{N}}\\
&\to \sqrt{\frac{-\ln(\delta) + \ln(2\exp 10N)}{N}}\\
&\to \sqrt{\frac{\ln(2) + \ln(\exp 10N)}{N}}\\
&\to \sqrt{\frac{10N}{N}}\\
&\to \sqrt{\frac{N}{N}}\\
&\to 1
\end{aligned}$$

The regret, R, is $\mathcal{O}(1)$
We will have nonzero regret even with infinite samples. Our inability to lower regret with increased sampling means we are not learning any more. ☹

4 SVMs: Hinge loss and mistake bounds

4.1 Argue that the function $\ell((x,y),w) = \max\{0, 1-yw^Tx\}$ is convex (as a function of w.)

Wikipedia says the maximum of two convex functions is convex.
Straight lines are convex, so 0 is convex.
To show that $1 - yw^Tx$ is convex, we need to show that the definition holds.
The definition of convexity is
$\forall x_1, x_2 \in X, \forall t \in [0,1] : f(tx_1 + (1-t)x_2) \le tf(x_1) + (1-t)f(x_2)$.
Let $f((x,y),w_1) = 1 - yw^Tx$.
f is convex if $tf((x,y),w_1) + (1-t)f((x,y),w_2) \ge f((x,y), tw_i + (1-t)w_2)$.
Show this:

$$\begin{aligned}
tf((x,y),w_1) + (1-t)f((x,y),w_2) &\ge t(1 - yw_1^Tx) + (1-t)(1 - yw_2^Tx) \\
&\ge t - tyw_1^Tx + 1 - yw_2^Tx - t - tyw_2^Tx \\
&\ge 1 - tyw_1^Tx - yw_2^Tx - tyw_2^Tx \\
&\ge 1 - y(tw_1^Tx + w_2^Tx + tw_2^Tx) \\
&\ge 1 - y(tw_1^Tx + (1-t)w_2^Tx) \\
&\ge f((x,y), tw_1 + (1-t)w_2)
\end{aligned}$$

Both 0 and $1 - yw^Tx$, so ℓ is convex.

4.2 What range of hinge loss?

When a point is correctly classified, the sign of $y_iw^Tx_i$ is always positive.
This means that $1 - yw^Tx$ is largest when $yw^Tx = 0$.
Te max argument also binds us to positive values.
Thus the hinge loss for correctly classified points is thus in the range $[0,1]$.

4.3 Let M(w) be the number of mistakes made by w on our dataset (in termsof the classification loss). Show that: $\frac{1}{n}M(w) \le \frac{1}{n}\displaystyle\sum_{i=1}^{n} \max\{0, 1 - y_iw^Tx_i\}$

Define a new function l such that $l(z) = 0 for z > 0$, and $l(z) = 1 for z < 0$.
Let $h(z) = max\{0, 1-z\}$.
Note that $l(z) \le h(z)$.
When we make a mistake, this corresponds to $y_iw^Tx_i$ being negative, and thus $l(y_iw^Tx_i = 1$.
This means $M(w) = \sum_i l(y_iw^Tx_i) \le \sum_i h(y_iw^Tx_i) = \sum_{i=1}^n max\{0, 1 - y_iw^Tx_i\}$.
Thus $\frac{1}{n}M(w) = \frac{1}{n}\sum_{i=1}^n max\{0, 1 - y_iw^Tx_i\}$.