Homework 3
CSE 446: Machine Learning

University of Washington

1 Policies *[0 points]*

Please read these policies. **Please answer the three questions below and include your answers marked in a "problem 0" in your solution set.** Homeworks which do not include these answers will not be graded.

Gradescope submission: When submitting your HW, please tag your pages correctly as is requested in gradescope. Untagged homeworks will not be graded, until the tagging is fixed.

Readings: Read the required material.

Submission format: Submit your report as a *single* pdf file. Also, please include all your code in the PDF file in a section at the end of your document, marked "Code"; also specify which problem(s) the code corresponds to. The report (in a single pdf file) must include all the plots and explanations for programming questions (if required). Homework solutions must be organized in order, with all plots arranged in the correct location in your submitted solutions. We highly recommend typesetting your scientific writing using LaTeX(see the website for references for free tools). Writing solutions by hand will be accepted provided they are neat; written solutions need to be scanned and included into a single pdf.

Written work: Please provide succinct answers *along with succinct reasoning for all your answers*. Points may be deducted if long answers demonstrate a lack of clarity. Similarly, when discussing the experimental results, concisely create tables and figures to organize the experimental results. In other words, all your explanations, tables, and figures for any particular part of a question must be grouped together.

Including your Python source code: For the programming assignments, submit your code in the pdf file along with a neatly written README file that instructs us how you ran your code with different settings (if applicable). Please note that we will not accept screenshots of Jupyter notebooks. If you do use Jupyter, you must export your code to a text file and put the text of your code in the submitted pdf file (in the last section) in a manner that can be executed in that order (without any extraneous or missing code).

We assume that you always follow good practice of coding (commenting, structuring); these factors are not central to your grade.

Coding policies: You must write your own code. You are welcome to use any Python libraries for data munging, visualization, and numerical linear algebra. Examples includes Numpy, Pandas, and Matplotlib. You may **not**, however, use any machine learning libraries such as Scikit-Learn,

TensorFlow, or PyTorch, unless explicitly specified for that question. If in doubt, post to the message boards.

Collaboration: It is acceptable for you to discuss problems with other students; it is not acceptable for students to look at another students written answers. It is acceptable for you to discuss coding questions with others; it is not acceptable for students to look at another students code. Each student must understand, write, and hand in their own answers. In addition, each student must write and submit their own code in the programming part of the assignment.

Acknowledgments: We expect the students not to refer to or seek out solutions in published material from previous years, on the web, or from other textbooks. Students are certainly encouraged to read extra material for a deeper understanding.

Extra Credit Policy: In order to get extra credit, you must do all the regular problems. Extra credit points will only be awarded if there are (honest attempts at) answers to *all* the regular questions. This is because they are not designed to be alternative questions to the regular questions.

1.1 List of Collaborators

List the names of all people you have collaborated with and for which question(s).

1.2 List of Acknowledgements

If you do inadvertently find an assignment's answer, acknowledge for which question and provide an appropriate citation (there is no penalty, provided you include the acknowledgement). If not, then write "none".

1.3 Certify that you have read the instructions

Please make sure to read and follow these instructions. Write "I have read and understood these policies" to certify this.

2 Understanding Maximum Likelihood Estimation *[11 points]*

Let us get more intuition on the maximum likelihood principle.

1. ***[1 point]*** Suppose our data are scalar numbers $z_1, \ldots z_N$, sampled from some underlying i.i.d. model $z_i \sim \mathcal{D}$. Write an expression that is a natural estimate for the mean of this distribution?
2. ***[4 points]*** Suppose now that we posit that our data come from a Gaussian model, where $z_i \sim N(\mu, \sigma^2)$ (even though it may be that this modeling assumption is not actually true). Suppose that we know the value of σ^2. Use the maximum likelihood principle to determine an estimate of μ. Show your derivation.
3. ***[1 point]*** In the previous problem, does our estimate of μ depend on σ^2?
4. ***[1 point]*** Suppose our data $z_1, \ldots z_N$ are binary outcomes (say Heads or Tails), sampled from a coin with probability of heads p. From our data, what is a natural estimate of the probability of heads?
5. ***[4 points]*** Now use the maximum likelihood principle to obtain an estimate of p, where you assume the underlying data generative model is a Bernoulli one (i.e. a Binomial distribution with unknown parameter p). Again, show your derivation.

3 Binary Classification with Logistic Regression *[30 points]*

Recall the probabilistic model:

$$\begin{aligned} p_{\mathbf{w}}(y = 1 \mid \mathbf{x}) &= \frac{1}{1 + \exp\left(-\mathbf{w} \cdot \mathbf{x}\right)} \\ p_{\mathbf{w}}(y = 0 \mid \mathbf{x}) &= 1 - p_{\mathbf{w}}(y = 1 \mid \mathbf{x}) = \frac{1}{1 + \exp\left(\mathbf{w} \cdot \mathbf{x}\right)} \end{aligned}$$

Our objective function here is:

$$\mathcal{L}_\lambda(\mathbf{w}) = \frac{-1}{N} \sum_{n=1}^{N} \log p_{\mathbf{w}}(y = y_n \mid \mathbf{x}_n) + \frac{\lambda}{2} \|\mathbf{w}\|^2 .$$

Now let us minimize our cost.

It is helpful to define:

$$\widehat{y}_n = p_{\mathbf{w}}(y = 1 \mid \mathbf{x_n})$$

You can view this as a probabilistic prediction. This definition will help in allowing you to easily modify your previous code.

3.1 Gradient Derivation *[6 points]*

1. ***[4 points]*** Show that:

$$\frac{d\mathcal{L}_\lambda(\mathbf{w})}{d\mathbf{w}} = \frac{-1}{N} \sum_{n=1}^{N} (y_n - \widehat{y}_n)\mathbf{x}_n + \lambda \mathbf{w} .$$

Remark: You might find this expression to be rather curious! It looks identical to the expression for our gradient in the average squared error case. You are free to think about why this was a fortunate coincidence. The choice of $y \in \{0, 1\}$ was indeed intentional.

2. ***[2 points]*** Again, in order to make our code fast, simplify this gradient expresion with matrix algebra by expressing it in terms of $X \in \mathbb{R}^{N \times d}$, $Y \in \mathbb{R}^{N \times 1}$, $\widehat{Y} \in \mathbb{R}^{N \times 1}$ (and other relevant quantities).

3.2 Let's try it out! *[12 points]*

Implement logistic regression on our "2" vs "9" dataset. Note: here you need to modify the decision rule: we should label a digit as a "2", i.e. $y = 1$, if $\mathbf{w} \cdot \mathbf{x} \geq 0$ (You should be able to see why this threshold is appropriate.).

Make sure to explicitly include a bias term in the model and do not regularize this term.

To be precise, let the un-regularized loss be:

$$\mathcal{L}(\mathbf{w}, b) = \frac{-1}{N} \sum_{n=1}^{N} \log p_{\mathbf{w},b}(y = y_n \mid \mathbf{x}_n) \,,$$

and so

$$\mathcal{L}_\lambda(\mathbf{w}, b) = \mathcal{L}(\mathbf{w}, b) + \frac{\lambda}{2} \|\mathbf{w}\|^2 \,.$$

Here $p_{\mathbf{w},b}$ is the model which includes a bias term. Note that gradient descent here would be:

$$\begin{aligned} \mathbf{w} &\leftarrow \mathbf{w} - \eta \nabla_{\mathbf{w}} \mathcal{L}_\lambda(\mathbf{w}, b) \\ &= \mathbf{w} - \eta \left(\nabla_{\mathbf{w}} \mathcal{L}(\mathbf{w}, b) + \lambda \mathbf{w} \right) \\ b &\leftarrow b - \eta \frac{\partial \mathcal{L}_\lambda(\mathbf{w}, b)}{\partial b} \\ &= b - \eta \frac{\partial \mathcal{L}(\mathbf{w}, b)}{\partial b} \end{aligned}$$

where the last step follows since our cost function is not regularizing the bias term.

Now run gradient descent:

1. ***[1 point]*** Specify *all* your parameter choices (this should be your step size and λ). What stepsize do you find works well, and what value of λ did you use? You might have to search around a little (it helps to search by going up or down in multiples of 10).
2. ***[5 points]*** Show your log loss on the training set and the development set (where the y-axis is the log loss and x-axis is the iteration). You should be able to convince yourself that the log loss (sometimes referred to as the cross entropy) can be computed as:

$$\frac{-1}{N} \sum_{n} \left(y_n \log \widehat{y}_n + (1 - y_n) \log(1 - \widehat{y}_n) \right) \,,$$

which can be directly computed in python with operations on the vectors Y and $\widehat{Y}$. Both curves should be on the same plot. What value of λ did you use?

3. ***[4 points]*** Make this plot again (with both curves), except use the misclassification error, as a percentage, instead of the average log loss. Here, make sure to start your x-axis at a slightly later iteration, so that your error starts below 5%, which makes the behavior more easy to view (it is difficult to view the long run behavior if the y-axis is over too large a range).
4. ***[2 points]*** Again, it is expected that you obtain a good test error (meaning you train long enough and you regularize appropriately, if needed). Report your lowest test error.

3.3 Let's use stochastic gradient descent *[12 points]*

Now use stochastic gradient descent, using one point at a time:

1. ***[1 point]*** Roughly, what is the largest stepsize at which SGD makes progress? (above this you will find that things start behaving very poorly).
2. ***[1 point]*** Specify your step size schedule, if you choose to decay it or what you did.
3. ***[5 points]*** After every 500 updates (starting before your first update at 0), make a plot showing your training average log loss and your development average the log loss on the y-axis and the iteration on the x-axis. Both curves should be on one same plot. What value of λ did you use?
4. ***[5 points]*** Make this plot again (with both curves, except use the misclassification error, as a percentage, instead of average log loss. Here, make sure to start your x-axis at a slightly later iteration, so that your error starts below 5%, which makes the behavior more easy to view (it is difficult to view the long run behavior if the y-axis is over too large a range). Again, it is expected that you obtain a good test error (meaning you train long enough and you regularize appropriately, if needed). Report the lowest test error.

3.4 EXTRA CREDIT: Mini-batch, stochastic gradient descent *[10 points]*

Again, due to the manner in which matrix multiplication methods (as opposed to using "For Loops") allow for faster runtimes (often through GPU processers), it is often much faster to use "mini-batch" methods by sampling m points at a time. By increasing the batch size, we reduce the variance in stochastic gradient descent. In practice (and in theory), this tends to be very helpful as increase m, and then there tends to be (relatively sharp) diminishing returns.

1. Now run stochastic gradient descent, using a mini-batch size of $m = 100$ points at a time. Here, each parameter updates means you use $m = 100$ randomly sampled training points.
   (a) ***[1 point]*** Roughly, what is the stepsize at which SGD starts to make significant progress? (above this it is poorly behaved) You might find it interesting that this stepsize is different than the $m = 1$ case.
   (b) ***[4 points]*** After every 500 updates (starting before your first update 0), make a plot showing your training average log loss and your development log loss on the y-axis and the iteration on the x-axis. Both curves should be on one same plot. What value of λ did you use (if you used it)? Specify your learning rate scheme if you chose to decay your learning rate.

Remark Note that every update now touches 100 points. However, an update should not be 100 times slower (even though, technically, your computer is doing 100 as much computation). This is,

again, due to how matrix multiplication is implemented.

(c) ***[4 points]*** Make this plot again (with both curves), except use the misclassification error, as a percentage, instead of average log loss. Here, make sure to start your x-axis at a slightly later iteration, so that your error starts below 5%, which makes the behavior more easy to view (it is difficult to view the long run behavior if the y-axis is over too large a range). Again, it is expected that you obtain a good test error (meaning you train long enough and you regularize appropriately, if needed). Report the lowest test error.

2. ***[1 point]*** Comment on how your plots differ from using SGD with $m = 1$.

4 Convexity: Linear and Logistic Regression *[14 points]*

Let us understand a few properties of linear regression (under the square loss) and logistic regression (under the log loss). Suppose we use $\text{sign}(w \cdot x)$ for predicting a label. Here, assume $y_n \in \{-1, 1\}$.

1. For logistic regression:

   (a) ***[3 points]*** Suppose our training data are linearly separable and $\lambda = 0$. What does our weight vector converge to and what is our misclassification error converging to? Why?

   (b) ***[2 points]*** Suppose now that $d \geq n$, $\lambda = 0$, and our n data points are all linearly independent. What does our weight vector converge to? Why?

   (c) ***[2 points]*** In both of these cases, why does regularization or "early stopping" make sense? Make sure to consider the implications for the true error in your answer.

2. For linear regression:

   (a) ***[1 point]*** 2 Suppose our training data are linearly separable. Suppose we run gradient descent for the case of linear regression: is our squared error converging to 0?

   (b) ***[2 points]*** Suppose now that $d \geq n$, $\lambda = 0$, and our n data points are all linearly independent. Suppose we run gradient descent for the case of the linear regression: is our squared error converging to 0?

3. ***[3 points]*** Suppose we are running gradient descent (for the logistic regression problem under the log loss). Suppose at some iteration our misclassification error hits exactly 0 on our training set. If we continue to run gradient descent, do we expect that we will continue to update our parameters? Why or why not?

5 Multi-Class Classification using Least Squares *[16 points]*

The MNist dataset, http://yann.lecun.com/exdb/mnist/, has been historically interesting. Also, much of the earlier focus on certain methods and algorithms in machine learning was partly due to obtaining good results on this dataset. If you work on the extra credit problem later on, you will gain a little more perspective on how many of these issues are not so relevant, once we start having a much more "flexible" representation.

Let us now build a classifier for digit recognition on all 10 digits.

You will use the full dataset (same as you used for PCA in HW2), were x is 784 dimensions.

5.1 "One vs all classification" with Linear Regression

In the previous two class problem, we used linear regression with $y \in \{0, 1\}$. Now we have 10 classes. Here we will use a "one-hot" encoding of the label. The label y_n will be a 10 dimensional vector, where the k-th entry is 1 if the label is for the k-th class and all other entries will be 0.

1. ***[0 points]*** Create a label matrix of size $Y \in \mathbb{R}^{N \times 10}$ for both your training, dev. and test set.

Here, we can consider a vector valued prediction:

$$\widehat{y}_n = W^\top \cdot \mathbf{x}_n \,.$$

where sized $W \in \mathbb{R}^{d \times 10}$ matrix.

As discussed in class, we can define the objective function here as:

$$\mathcal{L}_\lambda(W) := \frac{1}{N} \sum_{n=1}^{N} \frac{1}{2} \|y_n - W^\top \mathbf{x}_n\|^2 + \frac{\lambda}{2} \|\mathbf{W}\|^2$$

where you can view the penalty as the sum of the squares of the entries of W.

Note that this formulation is literally the same as doing k-binary classification problems on each of the classes separately, you will do a linear regression where you label a digit as $Y = 1$ if and only if the label for this digit is k (for $k = 0, 1, 2, \ldots 9$).

It is straightforward to verify that the solution is:

$$W_\lambda^* = \left(\frac{1}{N} X^\top X + \lambda \mathbb{I}_d \right)^{-1} \left(\frac{1}{N} X^\top Y \right) \,.$$

Note that here are stacking our the vectors y_n and $\widehat{y}_n$ into the matrices $Y \in \mathbb{R}^{N \times 10}$ and $\widehat{Y} \in \mathbb{R}^{N \times 10}$.

For classification, you will then take the largest predicted score among your 10 predictors.

Def. of the misclassification error: We say a mistake is made on an example (x, y) if our prediction in $\{1, \ldots k\}$ does not equal the label y. The $\%$ misclassification error (on our training, dev, test, etc) is the $\%$ of such mistakes made by our prediction method on our, respective, dataset.

Remark: This is sometimes referred to as "one against all" prediction. Note that we are just doing 10 separate linear regressions and are stacking our answers together.

Also, the gradient of this loss function can be expressed as:

$$\frac{d\mathcal{L}_\lambda(\mathbf{w})}{dW} = \frac{-1}{N} \sum_{n=1}^{N} \mathbf{x}_n (y_n - \widehat{y}_n)^\top + \lambda W \,. \tag{1}$$

Note that this expression is of size $d \times k$.

Dataset You will use the MNIST dataset you used from the last assignment. It contains all 10 digits with the labels. The instructor will post a function on Piazza for computing the misclassification % error that you are free to use.

1. ***[4 points]*** Based on the above gradient expression, write out the matrix algebra expression for this gradient in terms of $X \in \mathbb{R}^{N \times d}$, $Y \in \mathbb{R}^{N \times 10}$, $\widehat{Y} \in \mathbb{R}^{N \times 10}$ (and other relevant quantities), where there is no "sum over n" in your expression.
2. ***[12 points]*** Decide which method you would like to use: the closed form expression, GD, or SGD. Specify your method along with all your parameters. On the training set, dev set, and test set, what are your average square losses and what is your misclassification % error?

5.2 EXTRA CREDIT: Take a matrix derivative on your own *[5 points]*

Prove equation 1. To do this, lookup some facts about matrix derivatives on the internet (there are all sorts of "matrix cookbooks", "cheat sheets", etc. out there). Provide the one or two rules for how one takes a matrix derivative to obtain the proof. The proof should be just a few steps.

Also, you should be able to convince yourself as to how this follows from the vector proof you did earlier.

6 Multi-Class Classification using the the softmax *[20 points]*

We now turn to the softmax classifier. Here, y takes values in the set $\{1, \ldots k\}$. The model is as follows: we have k weight vectors, $w^{(1)}, w^{(2)}, \ldots w^{(k)}$. Here, we can view these parameters as columns in a matrix of size $W \in \mathbb{R}^{d \times 10}$ matrix. For $\ell \in \{1, \ldots k\}$,

$$p_W(y = \ell | x) = \frac{\exp(w^{(\ell)} \cdot x)}{\sum_{i=1}^{k} \exp(w^{(i)} \cdot x)}$$

Again, note that this is a valid probability distribution (the probabilities are positive and they sum to 1). Also, note that we have "over-parameterized" the model, since:

$$p_W(y = k | x) = 1 - \sum_{i=1}^{k-1} p_W(y = i | x)$$

We could define the model without using $w^{(k)}$. However, the instructor likes this choice as the derivative expressions become a little simpler (and it makes it easier to re-use code).

As before, it is helpful to define the "prediction vector":

$$\widehat{y}_n = p_{\mathbf{w}}(y \mid \mathbf{x_n})$$

where we view $p_{\mathbf{w}}(y \mid \mathbf{x_n})$ as k-dimensional (column) vector where the i-th component is $p_W(y = i | x_n)$.

As before, the (negative) likelihood function on an N size training set is:

$$\mathcal{L}_\lambda(W) = \frac{-1}{N}\sum_{n=1}^{N} \log p_W(y = y_n \mid \mathbf{x}_n) + \frac{\lambda}{2}\|W\|^2 .$$

where the sum is over the N points in our training set (where $y_n \in \{1, \ldots k\}$, so are we not using the one hot encoding in this expression).

For classification, one can choose the class with the largest predicted probability.

1. ***[6 points]*** Write out the derivative of the log-likelihood of the soft-max function as a sum over the N datapoints in our training set and in terms of the vectors x_n, the one hot encoding y_n, and the (vector) prediction $\widehat{y}_n$. (Hint: at this point, you likely will be able to guess the answer. You should still be able to write out a concise derivation). Make sure the dimensions give you a gradient of size $d \times k$.
2. ***[2 points]*** Now write out the matrix algebra expression for this derivative in terms of $X \in \mathbb{R}^{N\times d}$, $Y \in \mathbb{R}^{N\times 10}$, $\widehat{Y} \in \mathbb{R}^{N\times 10}$ (and other relevant quantities).
3. Now use either gradient descent or stochastic gradient descent (using one point at a time):
   (a) ***[1 point]*** Specify your parameter choices (your stepsize, λ, and and your step size schedule, if you choose to decay it or what you did.)
   (b) ***[5 points]*** Show your log loss on the training set and the development set (where the y-axis is the log loss and x-axis is the iteration).
   (c) ***[4 points]*** Make this plot again (with both curves), except use the misclassification error, as a percentage, instead of the average log loss. Here, make sure to start your x-axis at a slightly later iteration, so that your error starts below 15%, which makes the behavior more easy to view (it is difficult to view the long run behavior if the y-axis is over too large a range).
   (d) ***[2 points]*** Again, it is expected that you obtain a good test error (meaning you train long enough and you regularize appropriately, if needed). Report your lowest test error.

7 EXTRA CREDIT: Let's get to state of the art on MNIST! *[40 points]*

If you seek extra credit for this problem, you must also do the extra credit problem in Q3. If you do this, you should have the all the code you need to get going! Also, you must answer all of the regular questions as well.

We will now shoot to get to "state of the art" on MNIST, without "distortions" or "pre-processing". The table in http://yann.lecun.com/exdb/mnist/, shows which strategies use this pre-processing. In fact, you may even be curious if you can shoot to get "state of the art" performance with vanilla least squares (or you may be curious if logistic regression provides a notable improvement over least squares when you have lots of features?).

At the recent "NIPS" conference (one of the premier machine learning conferences), this talk, youtube video, created quite some buzz; it is related to how we will tackle this problem. There were many differing opinions expressed in subsequent discussions on the state of machine

learning. The instructor's hope is that, with more hands on experience, you will be better informed about the issues in play. The talk is relevant, since we are basically implementing the method discussed in this paper. The paper itself does not provide the most lucid justification; the method is really just a "quick and dirty" procedure to make features. In practice, there are often better feature generation methods; this one is remarkably simple.

In this problem, we will engage in the bad practice where *we do not have a dev set*. To a large extent, looking "a little" at the test set is done in practice (and this shouldn't hurt us too much if we understand how confidence intervals work). However, this has been done for quite sometime on this dataset, which is why the instructor is suspect of the test errors below 1.2%, among those methods that do no use "distortions" or "pre-processing" or "convolutional" methods (we should expect the latter methods to give performance bumps).

The views of the instructor are that about 1.4% or less is "state of the art", without "distortions" or "pre-processing" or "convolutional" methods (as discussed on the MNIST website). If we wanted even higer accuracy, we should really move to convolutional methods, which we may briefly discuss later in the class.

Finally, the approach below might seem a little non-sensical. However, an important lesson is that *large* feature representations, appropriately blown up, often perform remarkably well once you have a lot of labeled data.

Making the features

Grab the "mnist_all_50pca_dims.gz" dataset. It contains all the datapoints reduced down to 50 dimensions. There is no dev set. And there are 60,000 training points. The inputs have been normalized so that the features vectors x are, on average, unit length, i.e. $\mathbb{E}[\|x\|^2] = 1$. Load the modified MNIST dataset in Python as follows:

```
import gzip, pickle
with gzip.open("mnist_all_50pca_dims.gz") as f:
    data = pickle.load(f, encoding="bytes")
    Xtrain, Xtest = data[b"Xtrain"], data[b"Xtest"]
    Ytrain, Ytest = data[b"Ytrain"], data[b"Ytest"]
```

Now let us try to make "better" features; we are not going to be particularly clever in the way we make these features, though they do provide remarkable improvements. Let x be an image (as a vector in $\mathbb{R}^d$). Now we will map each x to a k-dimensional feature vector as follows: we will first construct k random vectors, $\mathbf{v}_1$, $\mathbf{v}_2$,..., $\mathbf{v}_k$ (these will be sampled form a Gaussian distribution). In other words, you first sample a matrix $V \in \mathbb{R}^{d \times k}$, where the columns of this matrix are $\mathbf{v}_1$ to $\mathbf{v}_k$; this can be done in python with the command np.random.randn(d, k).

Then our feature vector will be the following vector:

$$\phi(\mathbf{x}) = (\sin(2\mathbf{v}_1^\top \mathbf{x}), \sin(2\mathbf{v}_2^\top \mathbf{x}), \ldots \sin(2\mathbf{v}_k^\top \mathbf{x}))$$

Note that $\phi(x)$ is a k dimensional vector; $\sin(\cdot)$ is the usual trigonometric function; and the factor of 2 is a hyperparameter chosen by the instructor [1]. You are welcome to try and alter the 2 to another value if you find it works better. Note that you only generate V once; you always use the same V whenever you compute ϕ.

We will use (drumroll please....) $k = 60{,}000$ features. This seems like an unwieldy number. However, it will not actually be so bad since we you never actually explicitly construct and store this dataset. You will construct it "on the fly".

Tips

With only your laptop in hand (or the compute resources provided, which are hopefully not particularly impressive), this problem is just hard enough that it will force you to undersand many of the issues at play in large scale machine learning. In fact, if you try to explicitly contstruct your feature matrix of size $N \times k$, which is of size 60,000 $\times$ 60,000, you will hopefully run out of memory.

Regardless, the problem is very much solvable, in a timely manner, with even meager compute resources. The suggestions below are more broadly applicable to how we address many of the issues in large scale machine learning.

- (mini-batching) Mini-batching helps. It is too costly to try to full gradient updates. Use $m = 50$.
- (memory) As the dimension is large in this problem, we seek to avoid explicitly computing and storing the full feature matrix, which is of size $N \times k = N \times N$. Instead, you can compute the feature vector $\phi(x)$ 'on the fly', i.e. you recompute the vector $\phi(x)$ whenever you access an image x. In particular, you must do this on your minibatch with matrix operations for your code to be fast enough. If $\widetilde{X}$ is your $m \times d$ min-batch data matrix, do you see how the matrix $\sin 2\widetilde{X}V$ relates to the features you desire? Here $\sin$ is applied component wise.
- (regularization) It is up to you to determine how (and if) you set it. We do expect you to get good performance.
- (learning rates) With the square loss case, I like to set my learning rates large. And then I decay them only if I need to.
- (interrupting your code) Sometimes I find it helpful to be able to interrupt my code (with "Ctrl-C" or whatever you use) and have the ability to restart it without loosing my the current state of my parameters. Make sure you understand how to do this, and feel free to discuss this on the discussion board. This can be helpful. For example, for some problems, I may want to adjust my learning rate "by hand", and this allows me to do this.

Loss functions

You are free to try out both square loss and the logistic loss. The "objective function error" refers to either the square loss or the logistic/softmax loss (whichever you used). It is encouraged you also try the square loss as well (if you tried both, tell us!). In practice, in the small feature regime

[1] The aforementioned normalization of the data by the instructor makes this factor of 2 naturally correspond to a certain scale of the data. You can understand this more by looking at the paper in the link. It is analogous to the choice of a "bandwidth" in certain radial basis function kernel methods.

the softmax usually dominates the square loss; once you have lots of features sometimes the square loss does as well as logistic regression. We will be giving credit based on overall performance, and we will also give partial credit.

7.1 Let's implement it, starting small. *[15 points]*

It is best to start small, which makes it easier for you to debug your code. Start with $k = 5{,}000$ features.

1. ***[1 points]*** Roughly, what is the stepsize at which SGD starts to diverge? What value of λ did you use (if you used it)? Specify your learning rate scheme if you chose to decay your learning rate. Let us know here if you used the square loss or the logistic loss as your objective function. Or if you tried both!
2. ***[3 points]*** After every 500 updates, make a plot showing your average objective function error and your test average objective function error, with your average objective function error on the y-axis and the iteration # on the x-axis. Both curves should be on one same plot. Also make sure to start these plots sufficiently many updates after 0 and to label the x-axis appropriately based on where you start plotting (if you start plotting at update 0, your plots will be difficult to read and interpret due to the average objective function error initially dropping so quickly).
3. ***[3 points]*** For the misclassification error, make the same plots (again, with two curves. Do not start your plots at update 0. Make sure your plots are readable). Again, there should be two curves.
4. ***[2 points]*** Plot the euclidean norm of the weight vector, where the x-axis is the iteration number and y-axis is the norm of the weight vector at that update (you need only compute/store the norm every 500 iterations, as before). It is often helpful to plot the norms of you weight vectors, and you might find it striking how this curve behaves.
5. ***[3 points]*** What is the lowest training and test average objective function error losses achieved during your runs? Make sure you have run for long enough.
6. ***[3 points]*** What is the lowest training and test misclassification errors achieved during your runs? What is the smallest number of *total* mistakes made (out of the 60K points) on your training set and on your test set (over all updates)? Note you can just derive this from your lowest misclassification % errors on your train and test sets, respectively. Comment on overfitting.

7.2 Go big! *[25 points]*

Now let us use $k = 60{,}000$ features. Here, when you estimate your average objective function error and misclassification errors on your training set (for plotting purposes), you could use some *fixed* $10,000$ training points (say the first 10K points in the training set) if you have issues with speed/memory (you do *not* want to ever create a 60K $\times$ 60K matrix). If you do this, you should still ensure you are training on *all* 60K training points. Credit for this problem will be based on the quality of your plots and your test error; we want you to figure out how to get good performance!

1. ***[2 points]*** Roughly, what is the stepsize at which SGD starts to diverge? What value of λ did you use (if you used it)? Specify your learning rate scheme if you chose to decay your learning

rate.

2. ***[3 points]*** After every 500 updates, make a plot showing your training average objective function error and your test average objective function error, with your average objective function error on the y-axis and the iteration # on the x-axis. Both curves should be on one same plot. Also make sure to start these plots sufficiently many updates after 0 and to label the x-axis appropriately based on where you start plotting (if you start plotting at update 0, your plots will be difficult to read and interpret due to the average objective function error initially dropping so quickly).
3. ***[3 points]*** For the misclassification error, make the same plots (again, with two curves. Do not start your plots at update 0. Make sure your plots are readable). Again, there should be two curves.
4. ***[3 points]*** Plot the euclidean norm of the weight vector, where the x-axis is the iteration number and y-axis is the norm of the weight vector at that update (you need only compute/store the norm every 500 iterations, as before). It is often helpful to plot the norms of you weight vectors, and you might find it striking how this curve behaves.
5. ***[2 points]*** What is the lowest training and test average objective function error achieved during your runs? Make sure you have run for long enough.
6. ***[6 points]*** What is the lowest training misclassification % error achieved over all of your runs? What is the smallest number of *total* mistakes made (out of the 60K points) on your training set and on your test set (over all updates)? Note you can just derive this from your lowest misclassification % errors on your train and test sets, respectively (remember that you need to divide by a factor of 100 when dealing with %'s!). If you estimated your training error with a $10K$ subset, then make sure to multiply by a 6 when estimating the *total* number of errors on your training set.
7. ***[6 points]*** Provide a short discussion (about a paragraph) on overfitting. Do you see your training average objective function error rise? Did you make an extremely small number of total mistakes on your training set and was this very different from your test set? Comment on your findings.

7.3 Reflections *[0 points]*

You are welcome to jot down a few thoughts about what you found here. If you tried both square loss and logistic loss, please let us know. We may give credit adjustments if you tried both the square loss and the logistic loss.

8 EXTRA CREDIT: Proving a rate of convergence for GD for the least squares problem *[20 points]*

This is a fundamental convergence result in mathematical optimization. With a good understanding of the SVD, the proofs are short and within your reach.

Let us consider gradient descent on the least squares problem.

$$\mathcal{L}(\mathbf{w}) = \frac{1}{N}\frac{1}{2}\|Y - X\mathbf{w}\|^2$$

Gradient descent is the update rule:

$$\mathbf{w}^{(k+1)} = \mathbf{w}^{(k)} - \eta \nabla \mathcal{L}(\mathbf{w}^{(k)})$$

Let $\lambda_1, \lambda_2, \ldots \lambda_d$ be the eigenvalues of $\frac{1}{N} X^\top X$ in descending order (so λ_1 is the largest eigenvalue).

1. ***[8 points]*** In terms of the aforementioned eigenvalues, what is threshold stepsize such that, for any η above this threshold, gradient descent diverges, and, for any η below this theshold, gradient descent converges? You must provide a technically correct proof.
2. ***[8 points]*** Set η so that:

   $$\|\mathbf{w}^{(k+1)} - \mathbf{w}^*\| \leq \exp(-\kappa) \|\mathbf{w}^{(k)} - \mathbf{w}^*\|$$

   where κ is some (positive) scalar. In particular, set η so that κ is as large as possible. What is the value of η you used and what is κ? Again, you must provide a proof. You should be able to upper bound your expression so that you can state it in terms of the maximal eigenvalue λ_1 and the minimal eigenvalue λ_d. The above equation shows a property called *contraction*.
3. ***[4 points]*** Now suppose that you want your parameter to be ϵ close to the optimal one, i.e. you seek $\|\mathbf{w}^{(k)} - \mathbf{w}^*\| \leq \epsilon$. How large does k need to be to guarantee this?

9 Code

Please include all your code in the PDF file in this section. Specify which problem(s) the code corresponds to. Re Jupyter: refer to the policies section of the HW.