Assignment 1
CSE 547/Stat 548: ML for Big Data

University of Washington

0 Certify you read the HW Policies [0 points]

Please explicitly answer the three questions below and include your answers marked in a "problem 0" in your solution set. *If you have not included these answers, your assignment will not be graded.*

Readings: Read the required material.

Submission format: Submit your HW as a *single* typed pdf document; this document must contain all plots. Handwritten solutions (even if they are scanned in to the pdf) will *not* be accepted. Submit all your code in a gzipped tarball named `code.tgz`. It is encouraged that students use latex, though another comparable typesetting method is also acceptable. Some free tools that might help: ShareLaTeX (`www.sharelatex.com`), TexStudio (Windows), MacTex (Mac), TexMaker (cross-platform), and Detexify[2] (online). If you want to type, but don't know (and don't want to learn) LaTeX, consider using a markdown editor with real-time preview and equation editing (e.g., `stackedit.io`, `marxi.co`).

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.

Python source code: for the programming assignment. Please note that we will not accept Jupyter notebooks. Submit your code together with a neatly written README file to instruct how to run your code with different settings (if applicable). 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. If you use TensorFlow or PyTorch, you may *not* use any functions which define a neural network for you, e.g. no Keras is allowed. In PyTorch, you may *not* use the torch.nn.module (or any of the inherited functions). Basically, this means that you are not allowed to define any sort of neural network through the library: you must just write out the "forward pass" yourself; you may not use any built in functions/libraries which specify the number of nodes/layers in your network; the ML library you are using should not ever know you are coding up a neural net (it should be building computation graphs for the computations you specify). You are, however, allowed to use the torch.nn.functional interface provided by PyTorch.

On some questions, we will explicitly not allow ML packages, like Scikit-Learn, TensorFlow, or PyTorch. If in doubt, post to the message boards or email the instructors.

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.

0.1 List of Collaborators

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

0.2 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".

0.3 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. If you do not yet understand what it means not to use the PyTorch "torch.nn.module", please state the you will figure out how to avoid using the nn.module class (e.g. by making sure you post questions on the discussion board/going to office hours/websearch/etc if in doubt). It is fair game to use the "torch.nn.functional" mode.

0.4 Terms and Conditions to use the dataset

Please read the term and conditions to use the COCO dataset: see http://cocodataset.org/#termsofuse. We shall use this dataset for our homeworks and the default project. You must accept these terms and say: "I accept the terms and conditions to use the COCO dataset".

1 Certify you read the course policies on the website (0 points)

Read the course website, up until "Lecture Notes and Readings", so that you understand the course policies on grading, late policies, projects, requirements to pass etc. Write "I have read and understood these policies" to certify this. If you have questions, please contact the instructors.

2 AD in our example (16 points)

Consider the function from class (and the notes):

$$f(w_1, w_2) = (\sin(2\pi w_1/w_2) + 3w_1/w_2 - \exp(2w_2)) * (3w_1/w_2 - \exp(2w_2))$$

Suppose our program for this function uses the following evaluation trace:

input: $z_0 = (w_1, w_2)$

1. $z_1 = w_1/w_2$
2. $z_2 = \sin(2\pi z_1)$
3. $z_3 = \exp(2w_2)$
4. $z_4 = 3z_1 - z_3$
5. $z_5 = z_2 + z_4$
6. $z_6 = z_4 z_5$

return: z_6

2.1 The forward mode of AD

The forward mode for auto-differentiation is a conceptually simpler way to compute the derivative. Let us examine the forward mode to compute the derivative of one variable, $\frac{df}{dw_1}$. In the forward mode, we sequentially compute the *both* z_t and its derivative $\frac{dz_t}{dw_1}$ using the previous variables $z_1, \ldots z_{t-1}$ and the previous derivatives $\frac{dz_1}{dw_1}, \ldots \frac{dz_{t-1}}{dw_1}$.

1. (8 points) Explicitly write out the forward mode in our example. Write out the pseudocode computing all the intermediate derivatives as you would actually compute them in an implementation. You will be writing out a series of steps (the pseudocode) where you will be computing *both* z_t and its derivative $\frac{dz_t}{dw_1}$ using the previous variables $z_1, \ldots z_{t-1}$ and the previous derivatives $\frac{dz_1}{dw_1}, \ldots \frac{dz_{t-1}}{dw_1}$. You should have about 12 steps.

2.2 The reverse mode of AD

Now let us consider the reverse mode to compute the derivative $\frac{df}{dw}$, which is a two dimensional vector.

1. (8 points) Explicitly write out the reverse mode in our example. Again write the pseudocode computing all the intermediate derivatives as you would actually compute them in an implementation. You may assume you have evaluated the trace and already stored all the $z_t's$ in memory already.

3 Computation and Memory in AD (18 points)

Suppose we seek to compute the derivative with respect to a real valued function $f(w) : \mathbb{R}^d \to \mathbb{R}$. Let us examine some of the computational and memory issues involved in AD.

Be sure to read the required readings.

3.1 Computation (8 points)

Let T be the computation time to compute $f(w)$ using our program.

1. (2 points) Suppose we want to find the derivative of one variable $\frac{df(w)}{dw_1}$ with respect to the variable w_1. In order notation, how does the computational complexity (the runtime) of the forward mode compare to the reverse mode $\frac{df(w)}{dw_1}$?

2. (2 points) Suppose we want to find the derivative $\frac{df(w)}{dw}$, which is a d-dimensional vector. How would we do this with the forward mode and, in order notation, what is the computational complexity? Again, in order notation, how does the computational complexity (the runtime) of the forward mode compare to the reverse mode to compute $\frac{df(w)}{dw}$?

3. (2 points) If we could easily parallelize our computation (not worrying about communication), do you see a way to speed up the reverse mode? If so, what would be the new serial runtime? If not, why?

4. (2 points) If we could easily parallelize our computation (not worrying about communication), do you see a way to speed up the forward mode? If so, how so and what would be the new serial runtime? If not, why?

Remark: It is easy to see that if one can parallelize the program of f itself, it is not difficult to see that the we can also parallelize the reverse mode in a similar manner. This is very handy when it comes to using AD with GPUs.

3.2 Memory (10 points)

Memory is often a bottleneck in practice (e.g. we load as much as we can on a GPU when doing our computation in batches of data). Let us understand some of these issues.

Suppose our input w is d-dimensional and our evaluation trace is T steps. Assume one unit of memory is required to store a real number. Also, assume that we are free to delete variables at any time to free up memory (in practice, this often occurs by overwriting variables).

In this question, when memory is used to refer to how much "scratch space" we need to utilize in order to run our program. In this question, order notation is sufficient. When stating your answers do *not* include the memory required to store the parameter w or the program itself. Also, when in your answers, do not include the memory required to write out the programs output (assume we have already allocated this memory). We are interested in how much *excess* memory the program needs to have free in order to store its intermediate variables and do all its computations.

1. (1 points) Suppose that we were just interested in computing $f(w)$. Is it often the case that we can get away with less than T units of memory? How so?

2. (3 points) Suppose we only need to use m units of memory to compute $f(w)$. If we only wanted to compute $\frac{df}{dw_1}$, how much memory would we require to compute this using the forward mode? Explain.

3. (3 points) Suppose we only need to use m units of memory to compute $f(w)$. If we only wanted to compute $\frac{df}{dw_1}$, how much memory would we require to compute this using the reverse mode? (We are assuming here that we are not blowing up the amount of computation). Explain.

4. (3 points) Suppose we only need to use m units of memory to compute $f(w)$ and suppose we want to compute $\frac{df}{dw}$. If we don't care about runtime, what is the most memory efficient algorithm you can come up with? How much memory is needed

4 PyTorch can give us some crazy answers. (14 points)

Now you will construct an example in which PyTorch provides derivatives that make no sense. The issue is in understanding when it *is* ok and when it *is not* ok to use dynamic computation graphs.

You are going to find a way code up the *same* function in two different ways so that PyTorch will return different derivatives at the *same* point. The purpose of this exercise is to better understand how dynamic computation graphs work and to understand what you are doing when you using various AD softwares.

1. (2 points) Define *and* plot a one dimensional, real valued function which is not continuous (so even the definition of sub-differential does not make sense. You may use a function which is not differentiable at some point, both from the right or from the left). Your plot should clearly show the discontinuity.

2. (5 points) Now write out your function in PyTorch. You should be able to define this function so that PyTorch returns a derivative of 0 at some point x_0. Cut and paste the definition of your PyTorch function in the pdf file (*only* the def of the function), so that we can see easily what you did (you must still submit your working code in a tar file).

3. (5 points) Now find another way to write out your function in PyTorch; do this so that it is *exactly* the same function (remember that two functions are equivalent if they have the same input/output behavior at *all* inputs). Do this in a way so that PyTorch now returns a derivative of 2 at exactly the same point x_0 that you obtained a derivative of 0 in the previous question. Cut and paste the definition of your PyTorch function here (*only* the def of the function), so that we can verity it is in fact the same function.

4. (2 points) There is a no sane definition of the derivative for your function. Yet, you should not only found a way to get PyTorch to provide a derivative, you should have also found a

way to give you two *different* derivatives at exactly the same point (on the same function). What went wrong?

Remark: PyTorch goes beyond what we should be able to provably auto-differentiate (and it doesn't flag violations). TensorFlow does not provide us with true auto-differentiation; in TensorFlow, we are forced into building the computation graph ourselves as opposed to just defining a function $f(w)$ which can be legitimately auto-differentiated. With TensorFlow, in practice, we can often just ignore that we are building a computational graph (and it often works fine), until we get a confusing error message and realize something is strange is under the hood. In principle, Baur-Strassen shows very clearly that we should be able to provide a tool which rigorously provides auto-differentiation; furthermore, it seems like we should have a more mature library which would flag us for violations (unlike PyTorch or TensorFlow).

4.1 EXTRA CREDIT: a little more subtle... (50 points)

1. (15 points) Provide a *differentiable* function, where you can code it up in PyTorch in two different ways and where you can get two different derivatives at the same point x_0. Provide a plot of your function along with snippets of your PyTorch code in the pdf file. (Hint: If you understood the previous question, you should be able to use the same underlying idea).

2. (35 points) Use some of the built in (continuous) functions in Auto-Grad (the underlying numerical linear algebra toolkit which both TensorFlow and PyTorch call) to output a derivative at some point, where the derivative is not formally defined at that point (you are not permitted to use the ReLu function for your questions as this is not a function defined in AutoGrad). Again, show what your code returns and argue why it is not a correct derivative.

4.2 EXTRA CREDIT: find a more serious bug... (200 points)

1. Find an error in PyTorch or TensorFlow that lies within the auto-diff model, as opposed to an error due to permitting you to write code in a way that does not respect auto-differentiation. In particular, suppose you are interested in finding the derivative of $\frac{df(w,x)}{dw}$ (there derivative with respect to w not x); it is fine if your code branches using x though you must not brach using w. Specifically, write code which *should* provide a correct derivative (if the computation graph was built correctly and if the primitive derivatives are taken correctly) and where you show that PyTorch or TensorFlow returns an incorrect derivative.

5 (Source: KM Exercise 8.6) Review: elementary properties of l_2 regularized logistic regression (18 points)

This should be a review of your knowledge of binary and multi-class logistic regression.

5.1 The binary case

Consider minimizing

$$J(\mathbf{w}) = -l(\mathbf{w}, \mathcal{D}_{\text{train}}) + \lambda \|\mathbf{w}\|_2^2$$

where

$$l(\mathbf{w}, \mathcal{D}) = \sum_j \ln \mathbf{P}(y^j | \mathbf{x}^j, \mathbf{w})$$

is the log-likelihood on data set $\mathcal{D}$, for $y^j \in \{-1, +1\}$.

State if the following are True or False. Briefly explain your reasoning.

(a) (2 points) With $\lambda > 0$ and the features x_k^j linearly separable, $J(\mathbf{w})$ has multiple locally optimal solutions.

(b) (2 points) Let $\hat{\mathbf{w}}$= arg $\min_{\mathbf{w}} J(\mathbf{w})$ be a global optimum. $\hat{\mathbf{w}}$ is typically sparse (has many zero entries).

(c) (2 points) If the training data is linearly separable, then some weights w_j might become infinite if λ =0.

(d) (2 points) $l(\hat{\mathbf{w}}, \mathcal{D}_{\text{train}})$ always increases as we increase λ.

(e) (2 points) $l(\hat{\mathbf{w}}, \mathcal{D}_{\text{test}})$ always increases as we increase λ.

5.2 multi-class logistic regression

In multi-class logistic regression, suppose $Y \in \{y_1, ..., y_R\}$. A simplified version (with no bias term) is as follows: When $k < R$, the posterior probability is given by:

$$P(Y = y_k | X) = \frac{exp(\langle w_k, X \rangle)}{1 + \sum_{j=1}^{R-1} exp(\langle w_j, X \rangle)}$$

For $k = R$, the posterior is:

$$P(Y = y_k | X) = \frac{1}{1 + \sum_{j=1}^{R-1} exp(\langle w_j, X \rangle)}$$

Where $\langle w_j, X \rangle = \sum_{i=1}^{n} w_{ji} X_i$ (i.e. the dot product). We can replace the two equations above by a single equation, to simplify notation. For such, we introduce a fixed, pseudo parameter vector $w_R = [0, 0, 0, ..., 0]$. Now, for any label y_k, we write:

$$P(Y = y_k | X) = \frac{exp(\langle w_k, X \rangle)}{1 + \sum_{j=1}^{R-1} exp(\langle w_j, X \rangle)}$$

(a) (2 points) How many parameters do we need to estimate? What are these parameters?

(b) (2 points) Given N training samples $\{(x^1, y^1), (x^2, y^2), ..., (x^N, y^N)\}$, write down explicitly the log-likelihood function and simplify it as much as you can:

$$L(w_1, ..., w_{R-1}) = \sum_{j=1}^{N} ln(P(y^j|x^j, w))$$

(c) (2 points) Compute the gradient of L with respect to each w_k and simplify it.

(d) (2 points) Now add the regularization term $\frac{\lambda}{2}$ and define a new objective function:

$$L(w_1, ..., w_{R-1}) = \sum_{j=1}^{N} ln(P(y^j|x^j, w)) - \frac{\lambda}{2} \sum_{l=1}^{R-1} \|w_l\|_2^2$$

Compute the gradient of this new L with respect to each w_k

6 Getting familiar with our dataset (50 points)

Let us consider solving a binary classification problem (labels being $\{-1, 1\}$) with the dataset provided. We will use the square loss and consider training two models, namely (i) a linear model and (ii) a multi-layer perceptron. Note that this the same dataset that we will start branching out on, for the purposes of later assignments and the (default) course project.

The dataset contains two supercategories: vehicle and animal, and a number of categories for each supercategory. In the small dataset provided, each image contains objects of a single supercategory, say vehicle, and potentially multiple objects from the supercategory, such as car, boat, etc. In this exercise we shall build a classifier that learns to classify between these supercategories, by optimizing a square loss objective with the above models. For the purposes of learning these classifiers, we shall use features from a convolutional neural network (as opposed to the raw pixels from these images). We have provided starter code to read these features.

6.1 SGD and Linear Regression [20 points]

Here, the objective function we choose to optimize is:

$$L(w) = \frac{\lambda}{2}\|w\|^2 + \frac{1}{n} \sum_{i=1}^{n} \tfrac{1}{2} \cdot (y_i - \langle w, x_i \rangle)^2,$$

where, $y_i \in \{-1, 1\}$ is the label, $x_i \in \mathcal{R}^d$ are the features, $w \in \mathcal{R}^d$ is the linear model that we wish to optimize for and $\lambda > 0$ is the strength of ℓ_2 regularization.

Now consider running stochastic gradient descent on $L(w)$, where the stochastic gradient is computed using a single sample (i.e. a batch size of 1).

1. (4 points) Report the stepsize at which SGD starts to diverge (specified up to, say a factor of 2 from the actual value). Why would you expect the algorithm to diverge at too large a learning rate?

2. (8 points) After every 500 updates (index the first update at 0), compute $L(w)$ evaluated over the training/development/test dataset and make a plot with these values in the y-axis and the iteration on the x-axis. All three curves should be on one same plot. What value of λ did you use? Specify your learning rate scheme if you chose to decay your learning rate.

3. (8 points) Compute the misclassification error every 500 updates and plot these quantities (in a single plot) evaluated over the training, development and test dataset. Here, make sure to start your x-axis at a slightly later iteration to make the behavior of the $0/1$ error more easy to view (it is difficult to view the long run behavior if the y-axis is over too large a range). Report the lowest test error.

Note:

- It is expected that you obtain a good test error (meaning you train long enough and you regularize appropriately, if needed).

- For this part, you could either obtain gradients via autograd or code them up yourselves (it is easy for a "single layer model" such as this one). The former is easier but the latter would be faster in terms of wall clock time.

- It is crucial/fundamental to set any (hyper-) parameter of the algorithm (such as the learning rate/regularization parameter) based on performance on the training/development set for scientific integrity. Inference based on performance on the test set is equivalent to cheating the system.

6.2 Implement a Multi-Layer Perceptron (MLP) [20 points]

Here, the objective function we choose to optimize is:

$$L(w) = \frac{1}{n}\sum_{i=1}^{n} \tfrac{1}{2}(y_i - f_i(w))^2, \text{ with } f_i(w) = \langle w_2, \text{relu}(w_1^\top x_i)\rangle.$$

where, $y_i \in \{-1, 1\}$ is the label, $x_i \in \mathcal{R}^d$ are the features, $w_1 \in \mathcal{R}^{d\times h}$, $w_2 \in \mathcal{R}^h$ and $\text{relu}(x) = \max\{x, 0\}$, where the $\max$ is applied element wise, and h is the number of hidden nodes in our MLP.

Now consider running stochastic gradient descent on $L(w)$, where the stochastic gradient is computed using a single sample (i.e. a batch size of 1). In this exercise, answer the questions below with the number of hidden nodes being (a) 10, (b) 100, (c) 500.

1. (4 points) Report the stepsize at which SGD starts to diverge (specified up to, say a factor of 2 from the actual value).

2. (8 points) Compute $L(w)$ evaluated over the training, development and test dataset every 500 updates and plot these values in a single plot. Specify your learning rate scheme if you chose to decay your learning rate.

3. (8 points) Compute the misclassification error every 500 updates and plot these quantities (in a single plot) evaluated over the training, development and test dataset. As in the case of the linear model, make sure to start your x-axis at a slightly later iteration so as to make the behavior of the $0/1$ error more easy to view. Report the lowest test error.

6.3 Get familiar with AWS [10 points]

Use Amazon AWS to run the code developed in sections 6.1, 6.2. Attach a screenshot showing the progress of the algorithm (and the result) as viewed in the remote machine.

The purpose of this exercise is to get familiar with AWS and set yourselves up for future assignments and the course project.

Note:

1. Be sure to debug and test your code on your laptop (with perhaps the tiny version of the dataset) before moving to AWS. This is to ensure that you do not spend your AWS credits in debugging your code. For this exercise, you can conduct hyperparameter tuning, etc locally. We only require evidence that you were able to run your code on the cloud. In practice, however, hyperparameter search can be run in an embarrassingly parallel manner on the cloud.

2. Remember to terminate your EC2 instances without fail upon completion of jobs. You will continue to be charged for the instance until termination.

3. You are responsible for rationing your AWS credits over the entire quarter. Should you run out of your $\$100$ credit before the end of the quarter, it is your responsibility to figure out an alternate computation resources for the course (or pay for AWS yourself).

4. The free tier of EC2 (t2.micro) that we used for the demo is quite poor in performance. You might want to use either a t2.medium or t2.large. Experiment with these and other tiers of EC2 and discuss with your peers as to which is appropriate. You might want to stick to spot instances as they are generally significantly cheaper.

If you do not plan to use AWS: If you have other remote cluster resources, which you plan to use for your project, then please let us know what you are planning to use here. Also, take a screenshot. Your cluster resources must be such that you have to think about job scheduling and other parallelization issues.