6.867: Homework 2

For this homework, you will submit a single PDF containing your solutions to the following problems. There are 2 parts to the homework: a theoretical part and a hands-on part. We impose a **strict 6-page limit** on your writeup, to ensure that you focus on the important parts of your implementation and analysis. **NOTE: Please typeset your solutions and do not submit handwritten notes.** We make a few notes about the collaboration policy, submission procedure below, grading policy and other procedural information below.

Collaboration Policy

- We *strongly* encourage you to work on the problems in groups. Group size can be arbitary but we recommend keeping it to no larger than 5. However, all write-ups must be done individually, comprised of your own work, figures, and solutions.
- If partners submit identical PDFs, all will receive no credit.
- You should never use results from other students, the staff (from this year or from previous years), or other *online resources* in preparing your solutions to homework problems. In addition, students should never share their solutions (or staff solutions) with other students, including through public code repositories such as Github.

Submission: CMT & Dates

CMT is an online conference management software that you will use to submit and peer-grade your submissions. Instructions on how to use CMT should have been posted to Piazza. A few notes on deadlines.

- Submit your paper via the CMT site by 11:59 PM on October 18, 2018.
- After submission on CMT, you will be assigned 3 papers to review.
- Reviews must be entered on the CMT site by 11:59PM on October 30, 2018.
- All reviews will be made visible shortly thereafter, and students will have a two day period in which to enter rebuttals of their reviews into CMT if they wish.

Procedure, grading, etc.

You will find a zip file with some useful code and data in the Stellar Materials page. You can do these assignments in any computational system that you are used to. We recommend Matlab or the Pylab/Numpy/Scipy/

Matplotlib cluster of packages and we'll try to provide help for those two systems. If you use anything else, you're on your own...

You will be turning in a single, readable "paper" (a single PDF file) with your solutions. In the Stellar Materials page we have provided some sample solutions from a previous year. The content in these examples is not directly relevant, but note that there are coherent explanations and nice illustrations of results in figures and tables. You should write your paper as if it is going to be read by someone who has taken a class in machine learning but has not read this assignment handout.

We will be emulating the process of submitting papers for publication to a conference. We will be using an actual conference review system (CMT) to have these papers peer reviewed (by the other students). We will send out details about CMT soon. This means that your answers have to be readable and understandable to your peers and, where possible, interesting. Note that when explanations are called for, you will need to convince the reviewers that you understand what you're talking about. The course staff will serve as the Program Committee for the conference and make all final decisions.

Grading rubric

Your paper must be anonymous (no identifying information should appear in the PDF file). If it is not, it will automatically receive a 20% deduction, and will be graded by a grumpy staff member.

The paper must be no more than 6 pages long in a font no smaller than 10 point. It should include whatever tables, graphs, plots, etc., are necessary to demonstrate your work and conclusions. *It should not include code.*

Each of the **four** parts of the assignment will be graded on a scale from 0 to 5 (where 0 is failing and 5 is an A) on two aspects:

- **Content:** Did the solution answer the questions posed? Were the answers correct? Were the experiments well-designed or examples well chosen?
- **Clarity:** Were the results written up clearly? Were the plots labeled appropriately and described well? Did the plots support the points in the paper? Did the discussion in the paper illuminate the plots?

As a reviewer, you will be asked to provide a score for each section, and at at least two paragraphs of feedback, per review, explaining things that were done well and things that could have been improved upon.

Your overall score for this assignment will be:

- **80%**: The average of all 8 scores on your assignment given by all three reviewers.
- **20%**: A score for the quality of your reviews. This will be full credit, by default. But we will skim reviews and examine some carefully and may reduce this grade for review commentary that is sloppy or wrong.

The course staff will spot-check submissions and reviews, paying careful attention to cases where there were rebuttals. The staff will act as the program committee and determine a final score. Our overall goals in this process are:

- To motivate you to work seriously on the problems and learn something about the machine learning material in the process
- To engage you in thinking critically and learning from other students' solutions to the problems

We will arrange to give full credit to anyone who submits a serious and careful solution to the problems and who gives evidence of having read carefully the solutions they were assigned and who writes thoughtful reviews of them.

The questions are the points that your paper should cover in order to receive full credit. Your presentation should roughly follow the order of these questions so that your reviewers can see what you're doing.

Introduction to Theoretical Section

The first part of this assignment is 2 theoretical problems that will allow you to apply your more proof-oriented problem solving skills. Note that this is a bit different from homework problems of past years, which were focused entirely on more hands-on implementation. The writeup for the theoretical problems must to be included along with the writeup for the hands-on section, and will count towards your page limit.

1 Interpretations of Ridge Regression

The goal of this question is to understand the various interpretations of ridge regression. Suppose we have access to N data points: $(x^{(i)}, y^{(i)})$, $i = 1 \dots N$. Recall that the standard linear regression algorithm finds an appropriate model by solving the Ordinary Least Squares (OLS) question as follows.

$$\min_{w} \sum_{i=1}^{n} \{y^{(i)} - (w^T x^{(i)} + w_0)\}^2$$

where the closed form solution $\hat{w}^{OLS} = (X^T X)^{-1} X^T Y$, where $X \in \mathbb{R}^{N \times d}$ and $Y \in \mathbb{R}^{N \times 1}$.

The ridge regression objective, as we have seen in class, has a similar solution for regularization parameter λ: $\hat{w}^{Ridge}(\lambda) = (X^T X + \lambda I)^{-1} X^T Y$. In the following problems, we will examine two different interpretations of the ridge regression solution.

(a) Show that the closed form solution of ridge regression can be obtained by solving the ordinary least squares problem using the following augmented data set. Assume $w_0 = 0$.

First we *center* the data, computing a new feature matrix C, where $c_j^{(i)} = (x_j^{(i)} - \bar{x}_j)$, where $\bar{x}_j = \frac{1}{N} \sum_{i=1}^{N} x_j^{(i)}$ is the average value of the feature j. We then do the same for a new target matrix Z, where $z^{(i)} = (y^{(i)} - \bar{y})$ and $\bar{y} = \frac{1}{N} \sum_{i=1}^{N} y^{(i)}$. Finally, we augment the centered matrix C with d additional rows $\sqrt{\lambda} I$, and augment $\mathbf{Z}$ with d additional zeros.

Under this interpretation, by introducing artificial data with response value zero, the fitting procedure is forced to shrink the coefficients toward zero. This is related to the idea of using a regularization parameter to penalize the magnitude of the weight vector to prevent overfitting.

(b) Recall the Bayesian regression setup shown in class, where the posterior distribution on the weight parameter vector w is given by $\mathbb{P}(w|D) \propto \mathbb{P}(w)\mathbb{P}(D|w)$ where $D = \{x^{(i)}, y^{(i)}\}_{i=1\dots N}$. We then defined a Gaussian prior on $\mathbf{w} \sim \mathcal{N}(\mathbf{0}, \tau I)$ and a corresponding Gaussian likelihood $\mathbf{y} \sim \mathcal{N}(X\mathbf{w}, \sigma^2 I)$.

Show that the closed form solution of ridge regression is the mean (and mode) of the above posterior distribution. Find the relationship between the regularization parameter λ in the ridge formula, and the variances τ and σ^2 in the Gaussian formulation. Assume that the data is "centered", and thus we don't need a bias term.

Under this interpretation, the regularized least squares objective can be viewed as a Maximum A Posteriori (MAP) estimation under an assumption of normally distributed residuals. In this framework, the regularization terms of OLS can be understood as encoding priors on w.

2 Neural networks with one hidden layer can approximate continuous functions

In this question, we will show that neural networks with a d dimensional input, one hidden layer, a one-dimensional output, and sigmoidal activation can approximate any continuous function $f : \mathbb{R}^d \to \mathbb{R}$ on the interval $[0, 1]^d$.

This proof relies on four steps. Youll prove two of these and then well show how they combine with the other two to prove the overall theorem. **You do not need to prove Step 3, Step 4 or how the steps combine.**

Note that a one-hidden-layer neural network with N nodes in the hidden layer and activation function a has output:

$$F(x) = \sum_{i=1}^{N} \nu_i \, a(w_i^T x + b_i)$$

Let's first consider the set of all possible single (hidden) nodes in a neural network that use the activation function $a : \mathbb{R} \to \mathbb{R}$. We call this set $\mathcal{H}_a$ and can express it as follows:

$$\mathcal{H}_a = \left\{ x \mapsto a(w^T x + b) : w \in \mathbb{R}^d, b \in \mathbb{R} \right\}$$

We now consider the span of this set, i.e. all possible functions that are linear combinations of some functions in $\mathcal{H}_a$. A linear combination of these hidden nodes matches the output of the overall network, so $\mathrm{span}(\mathcal{H}_a)$ is, in fact, the set of all possible one-hidden-layer neural networks using a as the activation function.

1. *Step 1:* Let's first suppose that we are given two functions $\rho : \mathbb{R} \to \mathbb{R}$ and $\phi : \mathbb{R} \to \mathbb{R}$. We want to show that, if it possible to approximate ϕ using $\mathrm{span}(\mathcal{H}_\rho)$ in one dimension, then it is also possible to approximate $g \in \mathcal{H}_\phi$ using $\mathrm{span}(\mathcal{H}_\rho)$ in d dimensions.

   More precisely, suppose that for any interval $[r, s]$ and any $\tau > 0$, we can always find some $h \in \mathrm{span}(\mathcal{H}_\rho)$ where:

   $$\sup \{ |h(x) - \phi(x)| : x \in [r, s] \} \leq \tau$$

   Note that here $h : \mathbb{R} \to \mathbb{R}$.

   Show that, for any $\epsilon > 0$ and any $g \in \mathcal{H}_\phi$ with $g : \mathbb{R}^d \to \mathbb{R}$, we can choose some function $f \in \mathrm{span}(\mathcal{H}_\rho)$ where:

   $$\sup \left\{ |f(x) - g(x)| : x \in [0, 1]^d \right\} \leq \epsilon$$

2. *Step 2:* Prove that $\exp$ – i.e. the exponential function ($x \mapsto e^x$) – is Lipschitz continuous on any bounded interval. In other words, prove that there exists some real number $K \geq 0$ such that for any two real numbers x_1, x_2 in some given interval $[r, s]$, we have:

$$|e^{x_2} - e^{x_1}| \leq K|x_2 - x_1|$$

We now set σ to be the sigmoid function. (Technically, we only require it to be continuous, nondecreasing, and have the limits at $-\infty$ and $+\infty$ to be 0 and 1, respectively. Clearly, the sigmoid function satisfies these.)

Step 3: Given a $\tau > 0$ and interval $[r, s]$, if a function $\psi : \mathbb{R} \to \mathbb{R}$ is Lipschitz continuous along the interval $[r, s]$ then there exists an $h \in \operatorname{span}(\mathcal{H}_\sigma)$ that satisfies

$$\sup \{|h(x) - \psi(x)| : x \in [r, s]\} \leq \tau$$

Step 4: Given any continuous function $f : \mathbb{R}^d \to \mathbb{R}$ and $\epsilon > 0$, there exists a $g \in \mathcal{H}_{\exp}$ such that

$$\sup \left\{|f(x) - g(x)| : x \in [0, 1]^d\right\} \leq \epsilon$$

We omit the proofs of these steps.

As $\exp$ is Lipschitz continuous on a bounded interval (Step 2), we can approximate $\exp$ using $\operatorname{span}(\mathcal{H}_\sigma)$ in one dimension (Step 3). Therefore, we can approximate any member of $\mathcal{H}_{\exp}$ on inputs in $[0, 1]^d$ using $\operatorname{span}(\mathcal{H}_\sigma)$ in d dimensions (Step 1).

And so, we combine the knowledge that we can approximate any continuous function on inputs in $[0, 1]^d$ using an element in $\mathcal{H}_{\exp}$ (Step 4) with the knowledge that we can approximate any element in $\mathcal{H}_{\exp}$ on inputs in $[0, 1]^d$ using $\operatorname{span}(\mathcal{H}_\sigma)$. It thus follows that we can approximate any continuous function on inputs in $[0, 1]^d$ using $\operatorname{span}(\mathcal{H}_\sigma)$.

As $\operatorname{span}(\mathcal{H}_\sigma)$ represents the set of all possible one-hidden-layer neural networks with sigmoidal activation, we've shown that these neural networks can approximate an arbitrary continuous function on inputs in $[0, 1]^d$.

Hands-On Section

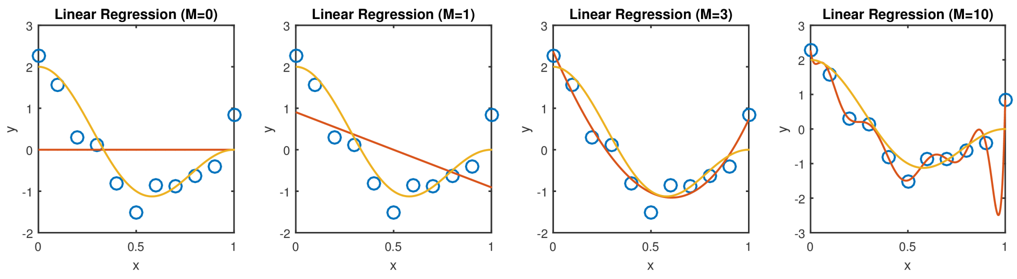
The 2nd part of this assignment deals with hands-on implementation questions.

3 Linear Basis Function Regression

In this problem, we will implement gradient descent in the context of linear regression models with varying classes of basis functions. We have already seen the closed-form solution for the optimal weight vector (see Murphy Section 7.2, Equation 7.16, and Bishop Equation 3.15). These closed-form solutions will help benchmark how well our gradient descent solutions perform.

For this problem, we have provided a text file, *p3curvefitting.txt*, that contains 11 points generated according to the function $y = \cos(\pi x) + \cos(2\pi x) + \eta$, where η is some random noise drawn according to $\mathcal{N}(0, \sigma^2)$ for some small value of σ. We have also provided you with Python and Matlab scripts to read this data and generate some example plots like the ones shown in Figure 1.

Figure 1: Plots of the optimal weight vector (red) for polynomial bases of various orders M used to fit the data (blue circles). The true function we are trying to learn is $\cos(\pi x) + \cos(2\pi x)$ (yellow).



1. First, assume we do not know the true underlying basis for these points and want to experiment with a polynomial fit. Write a procedure to compute the closed-form optimal weight vector for a polynomial basis of the form $(\phi_0(x), \phi_1(x), \ldots, \phi_M(x))$, where $\phi_0(x) = 1$, $\phi_1(x) = x$, $\ldots$, $\phi_M(x) = x^M$. Your algorithm should take in (a) an array of one-dimensional points X, (b) the vector Y of corresponding target values Y, and (c) M, the maximum order of the polynomial basis. Verify that your code works by replicating the plots from Figure 1.

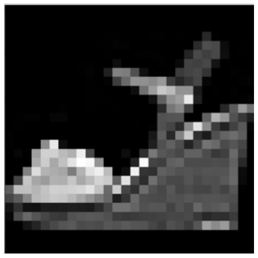
2. Now, implement a gradient descent procedure for each polynomial basis hypothesis using the sum-of-squares error (SSE) value as the target loss function. How did various hyperparameters (step size, initialization, convergence threshold) affect the overall performance of the algorithm compared to the closed-form solution? Compare the results of using both batch gradient descent and stochastic gradient descent on the same data with the same polynomial basis and explain your findings in relation to the function being optimized and the properties of the algorithms.

3. Finally, suppose that we now know that our basis function should involve cosine functions rather than polynomials - i.e. the basis is of the form $(\phi_1(x), \phi_2(x), \ldots, \phi_M(x))$, where $\phi_1(x) = \cos(\pi x)$, $\phi_2(x) = \cos(2\pi x)$ $\ldots$, $\phi_M(x) = \cos(M\pi x)$ - but that we still do not know how large of a basis to use. Write a procedure to calculate the optimal weight vector for the cosine basis function from $M = 1$ up to $M = 8$. How do these weight vectors compare to the true function used to generate this data? As an optional exercise - what happens if we modify our cosine basis to include $\phi_0(x) = 1$?

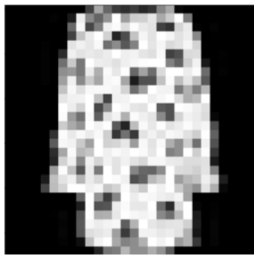
4 Neural Network - What went wrong?

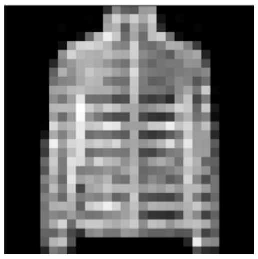
Daniel's wardrobe is a mess. He tries to keep it organized, but whenever he does laundry he has no idea where to put each piece of clothing. His friends Alice, Bob and Chelsea want to help. They decide that what Daniel needs is a way to quickly classify a piece of clothing so he can put it in the right place.

Each friend decides to train a neural network on the Fashion MNIST dataset to help Daniel out. Fashion MNIST consists of greyscale images that are each 28 by 28 pixels classified into the following categories: T-shirt/top, trouser, pullover, dress, coat, sandal, shirt, sneaker, bag, ankle boot. A few example images from the dataset are given below.


sandal

T-shirt


dress


coat

The friends are all having trouble getting their networks to train well. They know that you're taking 6.867 so they've come to you to help work out what's going wrong.

Setup

For this problem, we'll be using PyTorch, a commonly used machine learning library for Python. So, you'll need to install Python 3 and PyTorch.

- To install Python 3, download it at `https://www.python.org/downloads/`.
- To install PyTorch, run `pip3 install torch torchvision` if you use Linux or Mac. If you use Windows, run `pip3 install http://download.pytorch.org/whl/cu90/torch-0.4.1-cp36-cp36m-win_amd64.whl` and then run `pip3 install torchvision`. If you'd like to install it some other way (e.g. from source, so that you can use GPU acceleration on Mac), follow the instructions at `https://pytorch.org/`.

It's not a bug; it's a feature

1. Alice follows the lectures closely and attends her weekly recitations, so she is very familiar with classification problems using neural networks. She decides to implement a simple convolutional neural network (CNN) to help Daniel classify his clothes, but her network is not performing as well as she expected. She suspects that maybe one of her hyperparameters is poorly set up. Can you help Alice debug what went wrong?
   (a) Run `alice_net.py` and report both the training error and the test error after 10 epochs. Inspect `alice_net.py` and report her architecture along with her chosen hyperparameters.
   (b) Help Alice fix her network and report your best test error for after 10 epochs. Report what was wrong and how did you fix it?
   (c) Compare the behavior of the training and test loss for different values of the erroneous hyperparameter. What happened when that value is too high or too low? Report your findings and plot the training and test loss for at least 5 different values of that hyperparameter.
2. Bob also tried to follow the lectures closely, but he often grew tired and fell asleep in class, so he made his best attempt at a CNN for classification. Though the network architecture makes sense to him, Bob is not quite sure why his network is not performing as well as Alice's. Can you help explain what went wrong in Bob's approach?

(a) Run `bob_net.py` and report both the training error and the test error after 10 epochs. Inspect `bob_net.py` and report his architecture along with his chosen hyperparameters.

(b) Explain what Bob was trying to do and justify your arguments with code samples from the `train()` and `test()` functions?

(c) Explain why Bob's approach is solving the wrong problem, and suggest a possible fix. Justify why your fix would work better in this situation.

3. Chelsea is a little concerned about the sheer size of the Fashion MNIST dataset – surely Daniel doesn't need *that* many samples to train a good classifier.

   (a) Chelsea has written a method `chelsea_net.py` that specifies a training procedure using only N of the 60000 training examples (sampled uniformly at random). After running her modified procedure several times, Chelsea proudly declares that only 100 training points are actually required to get a good classifier. Do you believe her? Why or why not?

   (b) This begets the question - how many training points do you think are actually needed to consistently train a high-performing model? Run `chelsea_net.py` with many different values of N (ex. to get started, try at least 100, 500, 1000, 5000, 10000, and 30000 points) and create a **learning curve** plotting the training and test losses and accuracies as functions of N. Use your results to justify your choice of the optimal N. **Note:** We recommend you run the training procedure multiple times for each value of N, since the sampling process is random and thus each individual training set could be biased.

4. Daniel's really excited to try out the final model that his friends trained with your help. He decides to quickly test the model before he trusts it to classify all of his clothing. He takes photos of some of his pieces of clothing and then gets his friends to carefully label them for him. The code to train the final neural network is given in `final_net.py`.

   (a) Try training this network on 10 epochs. Report the final training and test accuracies on the Fashion MNIST dataset, and also report the accuracy on Daniel's photos.

   (b) Daniel's a bit concerned. Despite the test accuracy being over 80%, the real-world performance is much worse! Use the `visualize` method to inspect some of the images in the training dataset, test dataset and in Daniel's photo set. What do you notice? Can you explain the poor real-world accuracy? Are there any simple strategies that could be used to improve the real-world accuracy?

5. **OPTIONAL:** The TA's solution gets 89% accuracy on the test set. Can you do better? Feel free to change the model architecture and any of the hyperparameters. Evaluate yourself on the test set – don't worry about Daniel's photos for this part. Report your best accuracy, and also report the model architecture and hyperparameters you used to achieve that accuracy.