CSCI 4360/6360 Data Science II
Department of Computer Science
University of Georgia

Assignment 5: Embeddings All The Way Down

DUE: Tuesday, November 12 by 11:59:59pm

Out October 29, 2019

Questions

This homework assignment wraps up embedding strategies and introduces the fundamentals of neural networks, as well as some questions related to the final project and how linear regression is related to the problem. In the latter case, you'll need to work with students from the other class! They'll be entering Slack during this assignment, so feel free to introduce yourselves.

1 Neural Networks **[22pts]**

In this question we'll look at some of the basic properties of feed-forward neural networks.

First, consider the myriad activation functions available to neural networks. In this problem, we'll only look at very simple ones, starting with a combination of linear activation functions and the hard threshold; specifically, where the output of a node y is either 1 or 0:

$$y = \begin{cases} 1 & \text{if } w_0 + \sum_i w_i x_i \geq 0 \\ 0 & \text{otherwise} \end{cases}$$

Which of the following functions can be exactly represented by a neural network with one hidden layer, and which uses linear and/or hard threshold activation functions? For each case, *briefly* justify your answer (sketching an example is fine).

[3pts] Polynomials of degree one.

[3pts] Hinge loss $h(x) = \max(1 - x, 0)$.

[3pts] Polynomials of degree two.

[3pts] Piecewise constant functions.

Consider the following XOR-like function in two-dimensional space:

$$f(x_1, x_2) = \begin{cases} 1 & x_1, x_2 \geq 0 \text{ or } x_1, x_2 < 0 \\ -1 & \text{otherwise} \end{cases}$$

We want to represent this function with a neural network. For some reason, we decide we only want to use the threshold activation function for the hidden units and output unit:

$$h_\theta(v) = \begin{cases} 1 & v \geq \theta \\ -1 & \text{otherwise} \end{cases}$$

[10pts] Show that the smallest number of hidden layers needed to represent this XOR function is two. Give a neural network with two hidden layers of threshold functions that represent f, the XOR function. Again, you are welcome to provide a drawing, but that drawing must include values being propagated from each neuron. Alternatively, you could draw a table showing the values at each layer.

2 Linear Regression and Regularization **[20pts]**

Assume we are given n training examples $(\vec{x}_1, y_1), (\vec{x}_2, y_2), ..., (\vec{x}_n, y_n)$, where each data point $\vec{x}_i$ has m real-valued features (i.e., $\vec{x}_i$ is m-dimensional). The goal of regression is to learn to predict y from $\vec{x}$, where each y_i is also real-valued (i.e. continuous).

The linear regression model assumes that the output Y is a linear combination of input features X plus noise terms ϵ from a given distribution, with weights on the input features given by β.

We can write this in matrix form by stacking the data points $\vec{x}_i$ as rows of a matrix X, such that x_{ij} is the j-th feature of the i-th data point. We can also write Y, β, and ϵ as column vectors, so that the matrix form of the linear regression model is:

$$Y = X\beta + \epsilon$$

where

$$Y = \begin{bmatrix} y_1 \\ \vdots \\ y_n \end{bmatrix}, \epsilon = \begin{bmatrix} \epsilon_1 \\ \vdots \\ \epsilon_n \end{bmatrix}, \beta = \begin{bmatrix} \beta_1 \\ \vdots \\ \beta_m \end{bmatrix}, \text{and } X = \begin{bmatrix} \vec{x}_1 \\ \vdots \\ \vec{x}_n \end{bmatrix},$$

and where $\vec{x}_i = \big[x_1, x_2, ..., x_n\big]$.

Linear regression seeks to find the parameter vector β that provides the best fit of the above regression model. There are lots of ways to measure the goodness of fit; one criteria is to find the β that minimizes the squared-error loss function:

$$J(\beta) = \sum_{i=1}^{n} (y_i - \vec{x}_i^T \beta)^2,$$

or more simply in matrix form:

$$J(\beta) = (X\beta - Y)^T (X\beta - Y), \tag{1}$$

which can be solved directly under certain circumstances:

$$\hat{\beta} = (X^T X)^{-1} X^T Y \tag{2}$$

(recall that the "hat" notation $\hat{\beta}$ is used to denote an *estimate* of a true but unknown–and possibly, unknow*able*–value)

When we throw in the ϵ error term, assuming it is drawn from independent and identically distributed ("i.i.d.") Gaussians (i.e., $\epsilon_i \sim \mathcal{N}(0, \sigma^2)$), then the above solution is also the MLE estimate for $P(Y|X; \beta)$.

All told, then, we can make predictions $\hat{Y}$ using $\hat{\beta}$ (X could be the training set, or new data altogether):

$$\hat{Y} = X\hat{\beta} + \epsilon$$

Now, when we perform least squares regression, we make certain idealized assumptions about the vector of error terms ϵ, namely that each ϵ_i is i.i.d. according to $\mathcal{N}(0, \sigma^2)$ for some value of σ. In practice, these idealized assumptions often don't hold, and when they fail, they can outright implode. An easy example and inherent drawback of Gaussians is that they are sensitive to outliers; as a result, noise with a "heavy tail" (more weight at the ends of the distribution than your usual Gaussian) will pull your regression weights toward it and away from its optimal solution.

In cases where the noise term ϵ_i can be arbitrarily large, you have a situation where your linear regression needs to be *robust* to outliers. Robust methods start by weighting each observation *unequally*: specifically, observations that produce large residuals are down-weighted.

[10pts] In this problem, you will assume $\epsilon_1, ..., \epsilon_n$ are i.i.d. drawn from a Laplace distribution (rather than $\mathcal{N}(0, \sigma^2)$); that is, each $\epsilon_i \sim \text{Lap}(0, b)$, where $\text{Lap}(0, b) = \frac{1}{2b}\exp(-\frac{|\epsilon_i|}{b})$.

Derive the loss function $J_{\text{Lap}}(\beta)$ whose minimization is equivalent to finding the MLE of β under the above noise model.

Hint #1: Recall the critical point above about the form of the MLE; start by writing out $P(Y_i|X_i; \beta)$.

Hint #2: Logarithms nuke pesky terms with exponents without changing linear relationships.

Hint #3: Multiplying an equation by -1 will switch from "argmin" to "argmax" and vice versa.

[5pts] Why do you think the above model provides a more robust fit to data compared to the standard model assuming the noise terms are distributed as Gaussians? Be specific!

When the number of features m is much larger than the number of training examples n, or very few of the features are non-zero (as we saw in Assignment 1), the matrix $X^T X$ is not full rank, and therefore cannot be inverted. This wasn't a problem for logistic regression which didn't have a closed-form solution anyway; for "vanilla" linear regression, however, this is a show-stopper.

Instead of minimizing our original loss function $J(\beta)$, we minimize a new loss function $J_R(\beta)$ (where the R is for "regularized" linear regression):

$$J_R(\beta) = \sum_{i=1}^{n}(Y_i - X_i^T\beta)^2 + \lambda\sum_{j=1}^{m}\beta_j^2,$$

which can be rewritten as:

$$J_R(\beta) = (X\beta - Y)^T(X\beta - Y) + \lambda||\beta||^2 \tag{3}$$

[5pts] Explain what happens as $\lambda \to 0$ and $\lambda \to \infty$ in terms of J, J_R, and β.

3 Final Project Introduction **[16pts]**

In this project–encompassing the remainder of the semester, and extending beyond the deadline of this assignment–you'll be working with students from the Digital Humanities to answer questions from literature that require a collaboration between language experts and data scientists. You dipped your toes into this with Assignment 4; here, we'll take the next step.

A critical part of this collaboration will be the ability to communicate effectively across disciplines. To facilitate this work, we'll be breaking down the major research questions into a series of prompts. You'll work on one here.

First, look for your name among the following topics–that's what you'll be working on!

1. Using an assortment of novels from the 19th century, compare and contrast works written by male and female authors to determine if there is any systemic difference to distinguish them (`#gender-classification`).
    - Tori Pirtle
    - Alexander Kimbrell
2. Analyze texts in the victorian genre by english authors and compare their themes to those traditionally cited in critical analyses of the Victorians (`#victorian-literature-themes`).
    - Zirak Khan
    - Jialin Yang
    - Sasha Popov
3. Assess the difference in the mystery and fairytale/fable genres using data analytics to identify the literary conventions and styles of each (`#mystery-fairytale`).
    - Cheng Chen
    - Yulong Wang
    - Will Moore
4. Based on a data set consisting of autobiographies, letters, and memoirs between 1745 and 1919, what patterns can be seen within the themes of agency (individual uniqueness, mastery of self) and communion (relationships with others, cooperation and affiliation) based on the type of writing and over time (`#agency-vs-communion`)?
    - Aditya Patel
    - Michael Hearn
    - Anh Tran
5. Analyze the presence and portrayal of women in novels (authors and characters) written in the years surrounding the 1920s and any changes which coincided with the women's suffrage movement in the United States (`#before-and-after-women-suffrage`).
    - Elika Bozorgi
    - Hao Yang
    - Matthew Pooser

Second, join the Slack channel corresponding to your project (in parentheses at the end of the description of each above). Finally, with your teammates' help, answer the following questions:

[2pts] What question could you ask about the topic?

[3pts] What is your theory? How would you answer the question (what about the data informs your intuition)?

[3pts] Why is the question from the previous part important?

[2pts] What needs to be defined to answer the question?

[3pts] How would you measure and test these assumptions?

[3pts] What is the next step in the hypothesis?

4 Stochastic SVD **[42pts]**

In this question, you'll implement Stochastic SVD (SSVD) and compare its performance in certain applications. You are free to use the `scikit-learn` and `scipy.linalg` libraries.

The strength of SSVD lies in its reliance on randomization to generate an initial basis. In doing so, the rest of the algorithm becomes highly parallelizable; a 2011 PhD thesis proposed this method for computing the SVD of extremely large datasets in a single pass.

Fundamentally, SSVD has two main phases. In the first phase, you are computing a *preconditioner matrix* Q that, when applied to the data matrix A, "conditions" the system such that it is quantitatively better-behaved. Formally, it reduces the *condition number* κ of the linear system, where $\kappa = \frac{|\lambda_{\max}|}{|\lambda_{\min}|}$ ($\lambda_{\max}$ is the largest eigenvalue of A, while $\lambda_{\min}$ is the smallest). In general, this quantity is a strong proxy for "stability" of the corresponding system: if the condition number of a system is small, then it tends to be a smooth, continuous system: small changes in input result in small changes in output. Accordingly, reducing this quantity has all kinds of benefits, chief in particular that makes the system easier to solve (in terms of finding the eigenvalues and eigenvectors, which as we know, SVD is related to that task).

In the second phase, we use our preconditioner Q to compute a small matrix whose singular vectors *approximate* the basis of A, our data matrix. Then, by projecting the vectors back into the space of A using our preconditioner Q, we arrive at an estimate of the true singular vectors of A, and therefore, an estimate of the eigen-decomposition of A.

4.1 1A **[14pts]**

Start by implementing a Python function that computes the preconditioner matrix Q from the data matrix $A \in \mathbb{R}^{n \times m}$ for some number of basis vectors k, where $1 \leq k \leq m$:

1. Create a matrix $\Omega \in \mathbb{R}^{m \times k}$, where $\Omega \sim \mathcal{N}(0, 1)$ (*HINT*: look into the `numpy.linalg.standard_normal` function).
2. Form the matrix $Y \in \mathbb{R}^{n \times k}$ from the product $A\Omega$.
3. Perform a QR decomposition of Y. This creates two matrices: Q, which is orthogonal and unitary (and our $n \times k$ preconditioner!), and R, and upper-triangular matrix that we actually don't need (*HINT*: look into the `scipy.linalg.qr` function).

Next, implement the SSVD itself using our preconditioner Q and our data matrix A:

1. Precondition the system by forming the matrix $B \in \mathbb{R}^{k \times m}$ from the product $Q^T A$.
2. Perform the SVD (can be "truncated") of $BB^T = \hat{U}\Sigma^2\hat{U}^T$.
3. We can then extract the left singular vectors of A as $U = Q\hat{U}$.

Use the provided functions in the handout script to load the data, as well as to write out the U singular vectors and Σ singular values (the latter as a 1D array of numbers).

4.2 1B **[14pts]**

A potentially fatal flaw in this SSVD formulation is that, by relying on randomization in $\Omega \in \mathbb{R}^{m \times k}$, we are creating a matrix with rank *at most* k, our desired number of dimensions. However, thanks to the inherent stochasticity, it's very likely that on any given draw our Ω may actually have a rank that is smaller, ultimately culminating in estimated singular vectors of A that are pure noise.

To mitigate this, we can *oversample* in creating Ω. Return to the code you wrote in the first part and, for whatever k is provided as the target dimensionality, include an oversampling parameter p, where the dimensionality of Ω is $\mathbb{R}^{m \times (k+p)}$. We are still targeting a rank-k decomposition, but we are simply oversampling in this one step to greatly enhance our chances that Ω is, in fact, k-rank.

You can implement this using the `-p` command-line option that is already implemented in the sample script.

4.3 1C **[14pts]**

Another way of stabilizing SSVD beyond oversampling is to perform *power iterations* during the orthogonalization step of the preconditioner computations. After computing the initial Q matrix from the QR decomposition, but before applying it to compute B,

some number of power iterations q are applied to the system to "refine" the preconditioner Q.

Each power iteration i consists of two discrete steps:

1. Form the product $Y = AA^T Q_{i-1}$
2. Re-run the QR decomposition to find Q_i using Y from the first step

BONUS [5pts] How does the spectrum of singular values deviate from those of, say, the built-in `scipy` SVD solver? Is there a pattern in the deviations–something systemic–or are they random?

BONUS [15pts] Prove your assertion in the previous bonus question.

Administration

1 Script Design

Your code should be able to process: an input file containing the $N \times M$ matrix, the number of SVD components k, oversampling parameter p and number of power iterations q, random seed r, and an output directory to write the stochastic singular vectors and values. Most of these parameters are optional, except for the input and output flags.

You'll also be provided the boilerplate to read in the necessary command-line parameters:

1. `-i`: a file path to a text file containing the data
2. `-k`: number of SVD components to take in the decomposition
3. `-p`: oversampling rate for generating the random basis Ω
4. `-q`: number of power iterations to perform on the preconditioner Q
5. `-r`: random seed to use (for debugging)
6. `-o`: a filesystem path to an output directory, where the singular values and vectors will be written

The format of the input file will be whitespace-delimited, where a single row of the input matrix will be on one line, and individual values are separated by whitespace. You can use the provided utility functions in the boilerplate script, `_save_data` and `_load_data`, to save outputs and load inputs respectively. These functions are already written in the `assignment5.py-TEMPLATE` file that will handle reading in data and parsing command-line arguments.

The format of the output file should be two separate files: one containing the k singular vectors (in identical format to the input, with one row of the matrix $\hat{U}$ per line), and one containing the k singular values (one number per line). You can very easily test how well your SSVD is doing–you can use the built-in SciPy SVD solver. The autograder on AutoLab has been scaled so that SSVD with the properties mentioned in each subproblem should receive full credit, if implemented correctly.

2 Submitting

All submissions will go to **AutoLab**. You can access AutoLab at:

- https://autolab.cs.uga.edu

You can submit deliverables to the **Assignment 5** assessment that is open. When you do, you'll submit two files:

1. `assignment5.py`: the Python script that implements SSVD
2. `assignment5.pdf`: the PDF write-up with any questions that were asked

These should be packaged together in a tarball; the archive can be named whatever you want when you upload it to AutoLab, but the files in the archive should be named **exactly** what is above. Deviating from this convention could result in my annoyance (and autograder failures, but let's be honest the former is the more important)!

To create the tarball archive to submit, run the following command (on a *nix machine):

```
> tar cvf assignment5.tar assignment5.py assignment5.pdf
```

This will create a new file, `assignment5.tar`, which is basically a zip file containing your Python scripts and PDF write-up. Upload the archive to AutoLab. There's no penalty for submitting as many times as you need to, but keep in mind that swamping the server at the last minute may result in your submission being missed; AutoLab is programmed to close submissions *promptly* at 11:59pm on November 12, so give yourself plenty of time! A late submission because the server got hammered at the deadline will *not* be acceptable (there is a *small* grace period to account for unusually high load at deadline, but I strongly recommend you avoid the problem altogether and start early).

3 Reminders

- If you run into problems, ping the `#questions` room of the Slack chat. If you still run into problems, ask me. But please please please, **do NOT** ask Google to give you the code you seek! I will be on the lookout for this (and already know some of the most popular venues that might have solutions or partial solutions to the questions here).

- Prefabricated solutions (e.g. `scikit-learn`, OpenCV) are NOT allowed! You have to do the coding yourself! But you **can** use the various modules mentioned throughout this write-up, so long as they don't replace the required code.
- If you collaborate with anyone, just mention their names in a code comment and/or at the top of your homework writeup.