

## Assignment 5: Embeddings All The Way Down

---

Aditya Shinde

Out October 19, 2017

### Questions

This homework assignment focuses on embedding strategies with a touch of neural networks.

**This homework requires more coding than previous assignments, so plan accordingly!** It also does NOT use autograding; whether that's a blessing or curse may vary, but it also means that no example data or code templates will be provided. Furthermore, you'll be asked to generate figures and embed them in your homework write-up.

### NEURAL NETWORKS [25PTS]

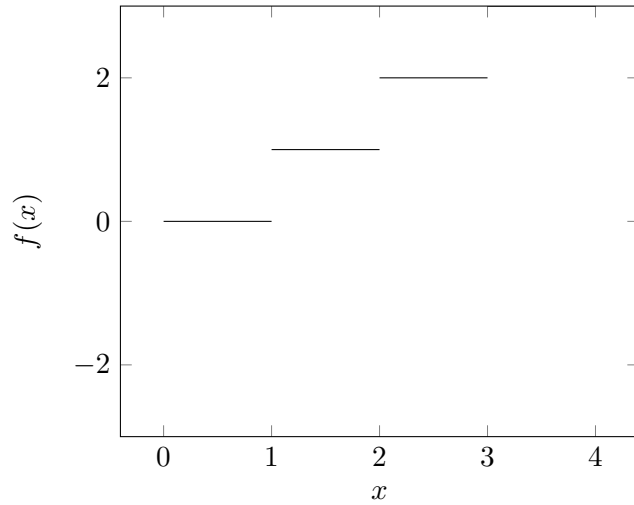
A neural network with a single hidden layer and a linear activation will form a linear hypothesis. So it will be able to model a polynomial with degree one very accurately.

For the hinge loss function,

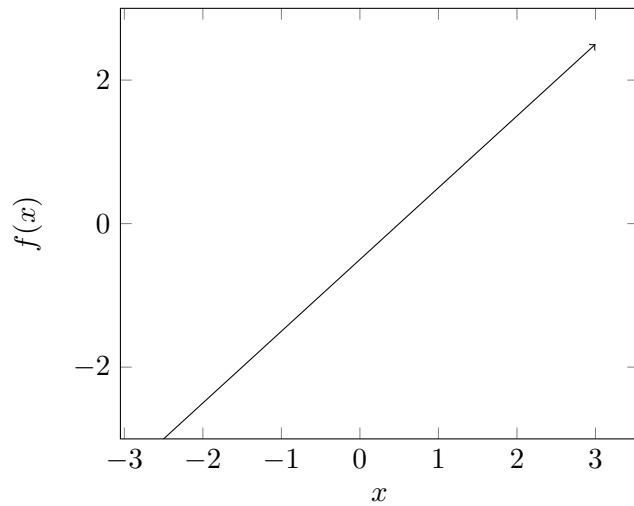
Polynomials of degree two cannot be represented by neural networks. These are non linear functions. And with linear activations, we can have only linear hypotheses.

Piecewise constant functions cannot be represented by linear networks. However, they can be approximated upto a certain degree. for instance a step function which is defined by:

$$f(x) = \begin{cases} 1 & x \geq 0 \\ 0 & \text{otherwise} \end{cases}$$



This function can be approximated in the following way.



Now this is certainly a very bad approximation. And this particular function is just one from the family of many piecewise constant functions. So the answer would be no. Since not all step functions can be approximated by a linear network.

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:

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

**[13pts]** 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.

### KERNEL SMOOTHING **[35PTS]**

In this problem, we'll look at nonparametric kernel smoothing for approximating a function from noisy data. We'll also throw in leave-one-out cross-validation to observe its effects on the learned function. For the sake of simplicity, we'll stick with one-dimensional data.

We have a "dataset"  $(x_1, y_1), (x_2, y_2), \dots, (x_n, y_n)$ , as follows:

$$y_i = f(x_i) + \epsilon_i$$

where  $\epsilon_i \sim \mathcal{N}(0, \sigma^2)$ .

The goal of any regression problem is to estimate the true  $f(x)$  with an empirical estimate  $\hat{f}(x)$ . The Nadaraya-Watson estimator is given by:

$$\hat{f}(x_k) = \frac{\sum_{i=1}^n y_i K\left(\frac{|x_i - x_k|}{h}\right)}{\sum_{i=1}^n K\left(\frac{|x_i - x_k|}{h}\right)}$$

where  $K(\cdot)$  is the kernel, and  $h$  is the bandwidth. In this example, we'll use the Gaussian kernel:

$$K(a) = \frac{1}{\sqrt{2\pi}} \exp\left\{-\frac{a^2}{2}\right\}$$

In this equation,  $h$  takes the place of the standard deviation, and the data point  $x$  will take the place of the mean.

(yes: you're going to be writing some code!)

**[5pts]** Write a basic Python program that generates our dataset.

- Sample  $x_i \sim U(-5, 5)$  (that's a uniform distribution from -5 to 5)
- Sample  $\epsilon_i \sim \mathcal{N}(0, 0.1)$

- Set  $y_i = \sin(x_i) + \epsilon_i$

**[3pts]** Implement a squared loss function  $\ell(\cdot)$  (you can use vectorized NumPy arrays for this):

$$\ell(y, \hat{f}(x)) = (y - \hat{f}(x))^2$$

**[2pts]** Sample a dataset of size  $n = 100$  and plot it; you can use `matplotlib.pyplot.scatter`. Overlay the scatter plot with the true regression function (meaning compute the  $\sin(\cdot)$  of each  $x_i$  and plot that in addition to the  $y_i$  you computed); you can use `matplotlib.pyplot.plot`.

**[20pts]** Now, write a program which performs the following:

- Sample a “training set” of size  $n = 100$ , and a “testing set” of size  $m = 100$ .
- Compute the kernel smoother for a particular choice of  $h$ , along with the empirical error (average loss between all  $y$  and  $\hat{f}(x)$ ), leave-one-out cross-validation error (average loss), and testing error (average loss).
- Compute those measurements for the following values of  $h \in \{1.0, 0.75, 0.5, 0.25, 0.1, 0.05, 0.01, 0.005, 0.001\}$ .
- Construct scatter plots of test error versus empirical error, and test error versus leave-one-out cross-validation error. Test error should always be on the  $y$ -axis.
- Choose the function  $\hat{f}$  which minimizes leave-one-out cross-validation error, and plot the training data sample along with the value of this function evaluated on the training data  $x$  values.

**[5pts]** Explain why it is a bad idea to merely minimize the empirical risk in problems like this (*HINT*: refer to the last two plots).

**Include your code in a file named `assignment5_q2.py` when you submit to AutoLab, as this will be manually inspected. Include the plots in your write-up.**

## STOCHASTIC SVD [40PTS]

1

Implemented as the function `compute_Q(A,k)` in the code.

2

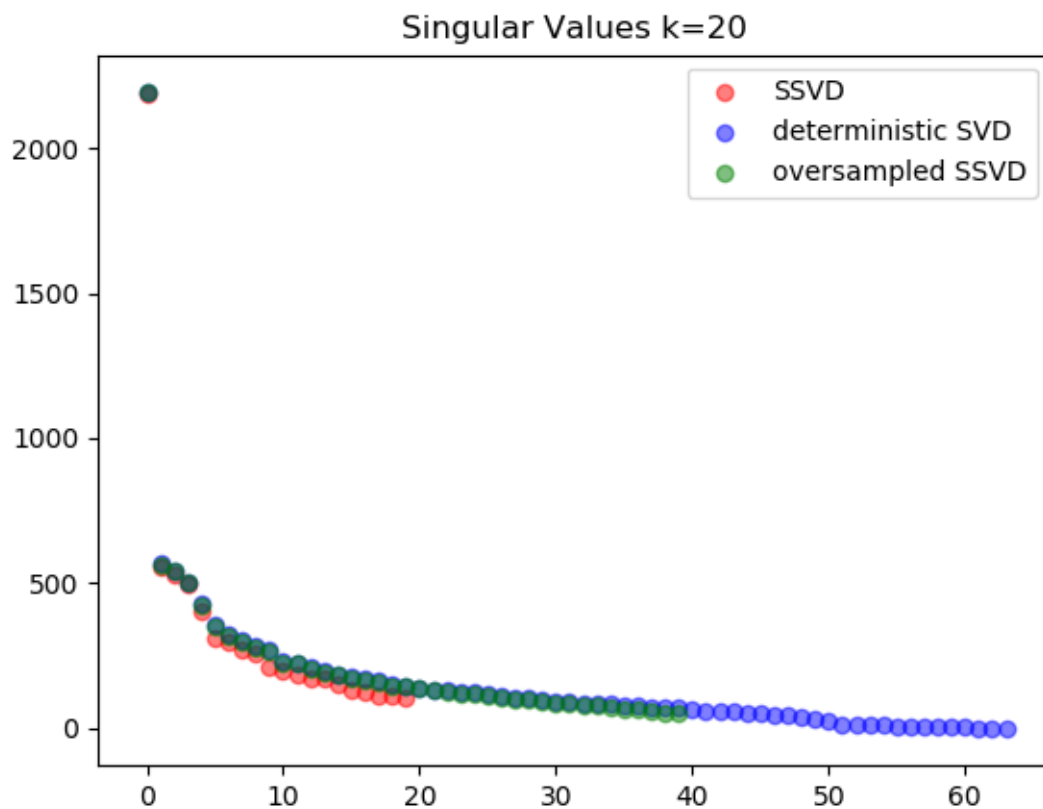
Implemented as the function `SSVD(A,Q)` in the code.

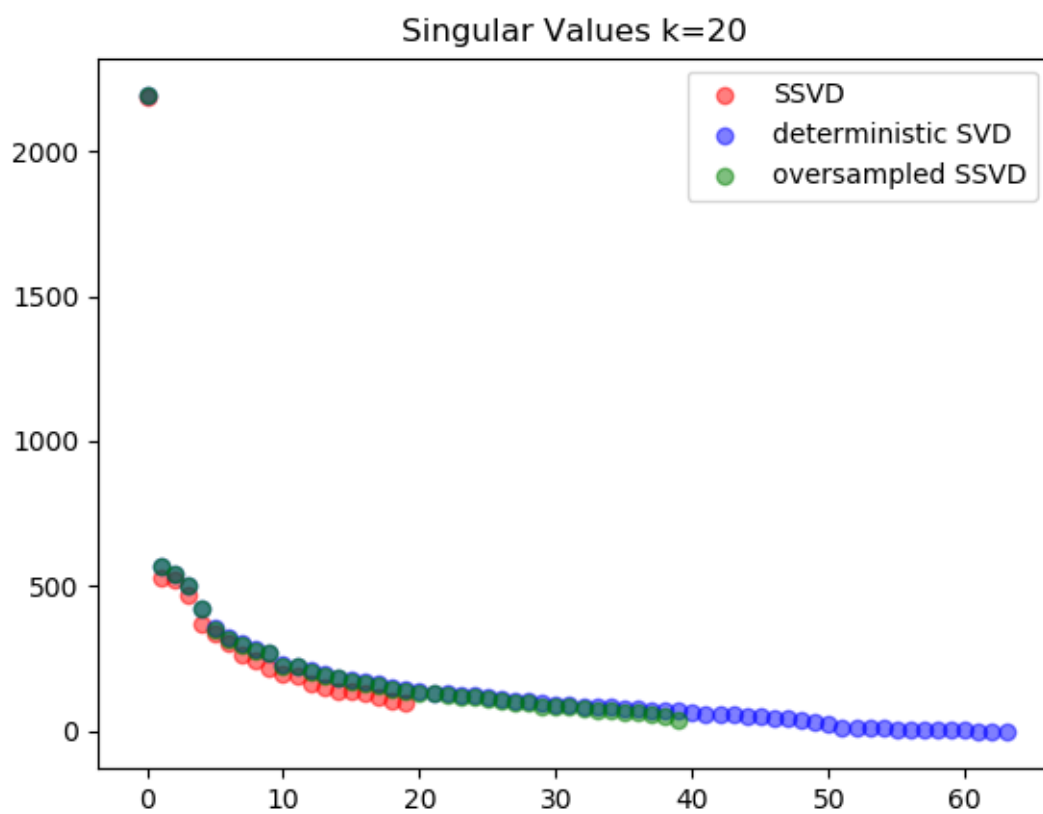
3

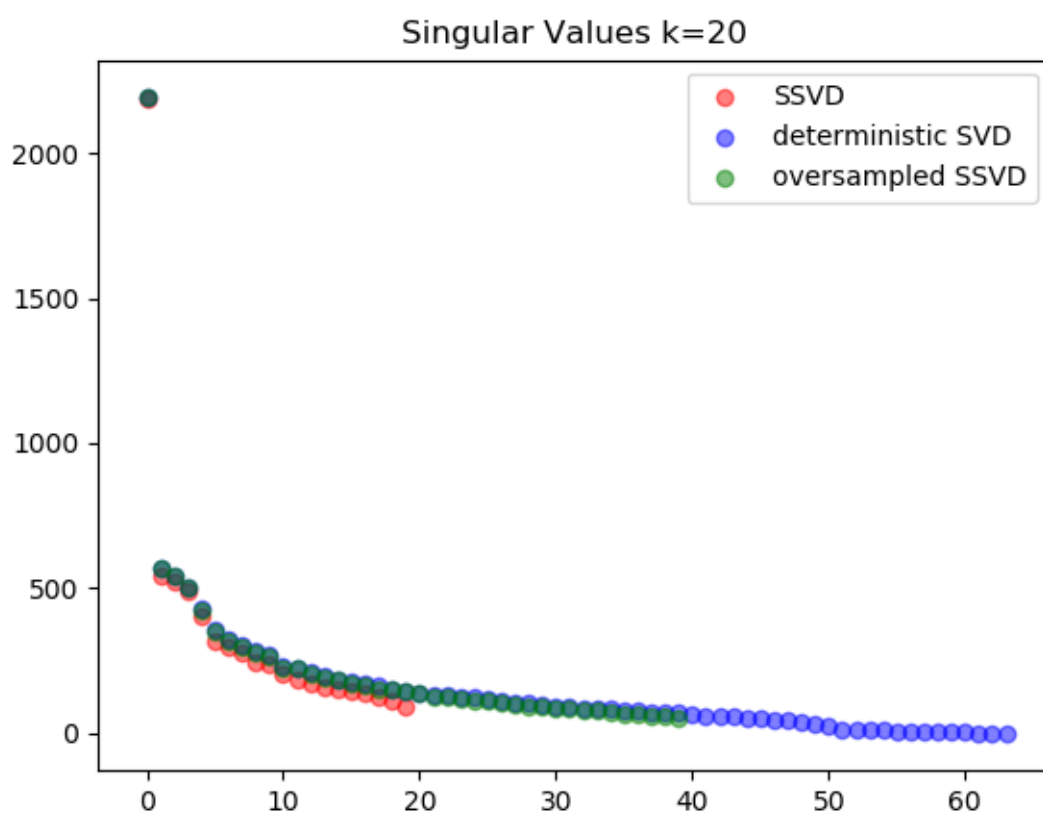
The plots for the singular values are plotted below. The points in red are the singular values for the SSVD operation for  $k = 20$  and the ones in blue are the singular values computed using the deterministic implementation of SVD in scipy. The plots are shown on the next page along with the plots for oversampling.

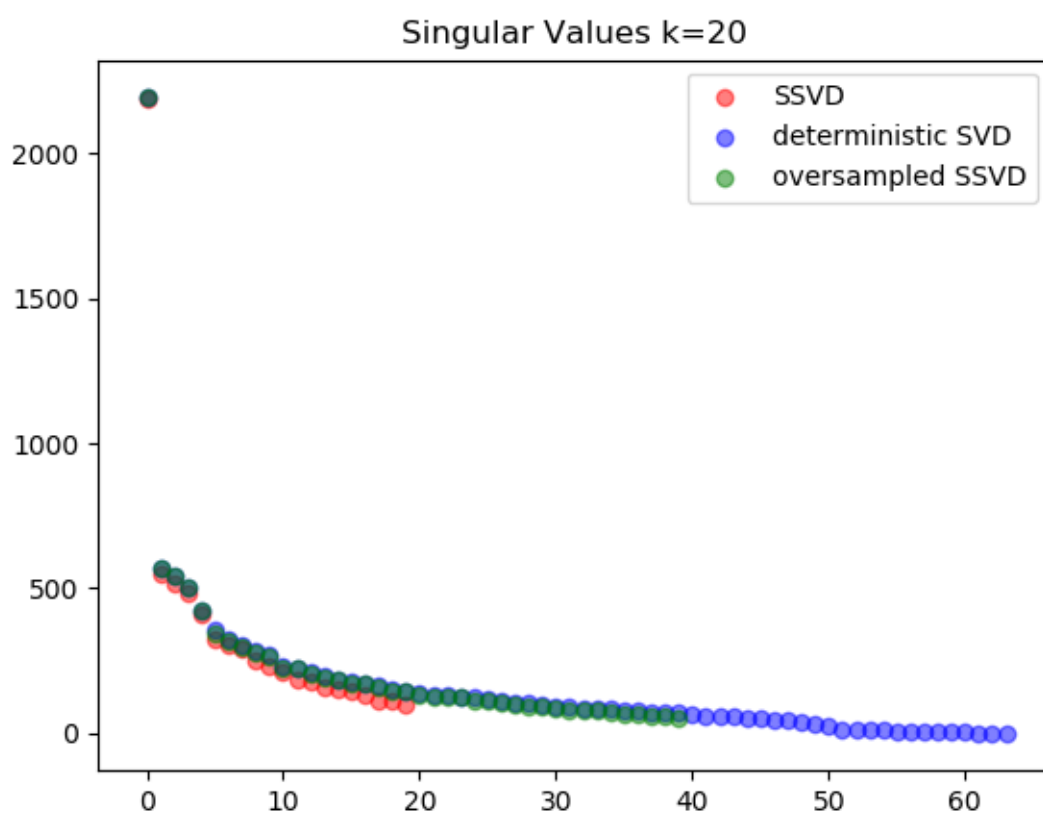
4

Plots are on the next page along with the plots for the previous question.

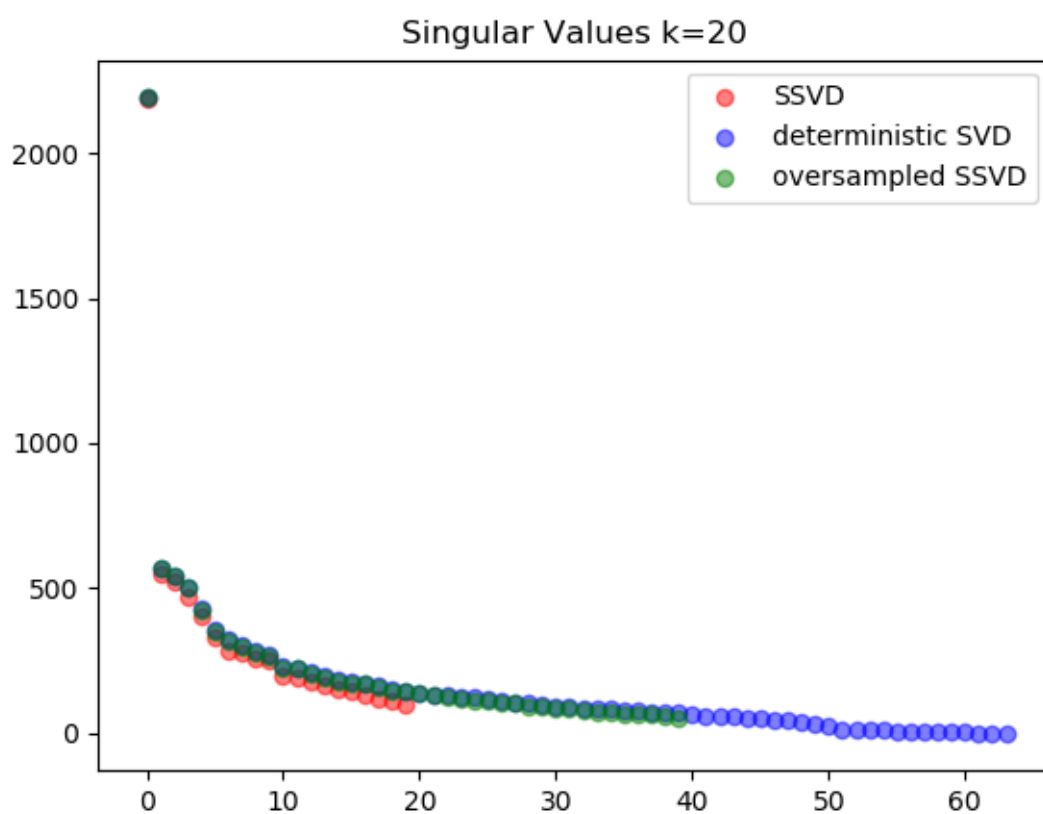


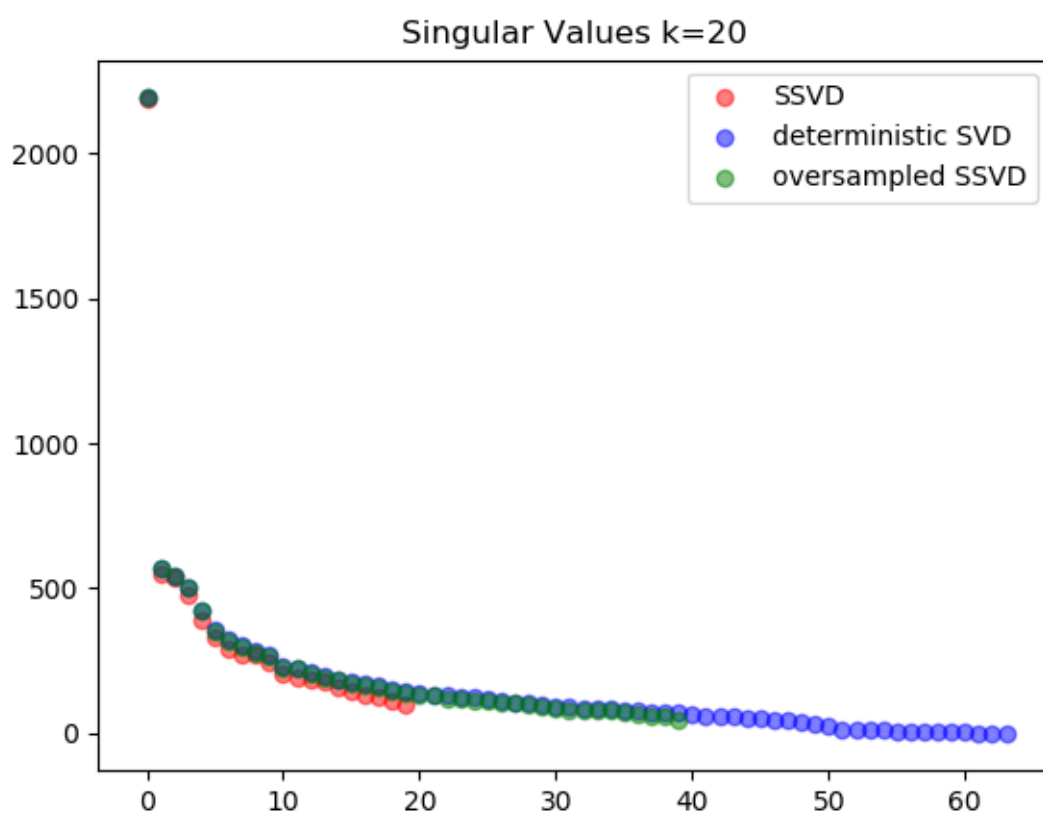


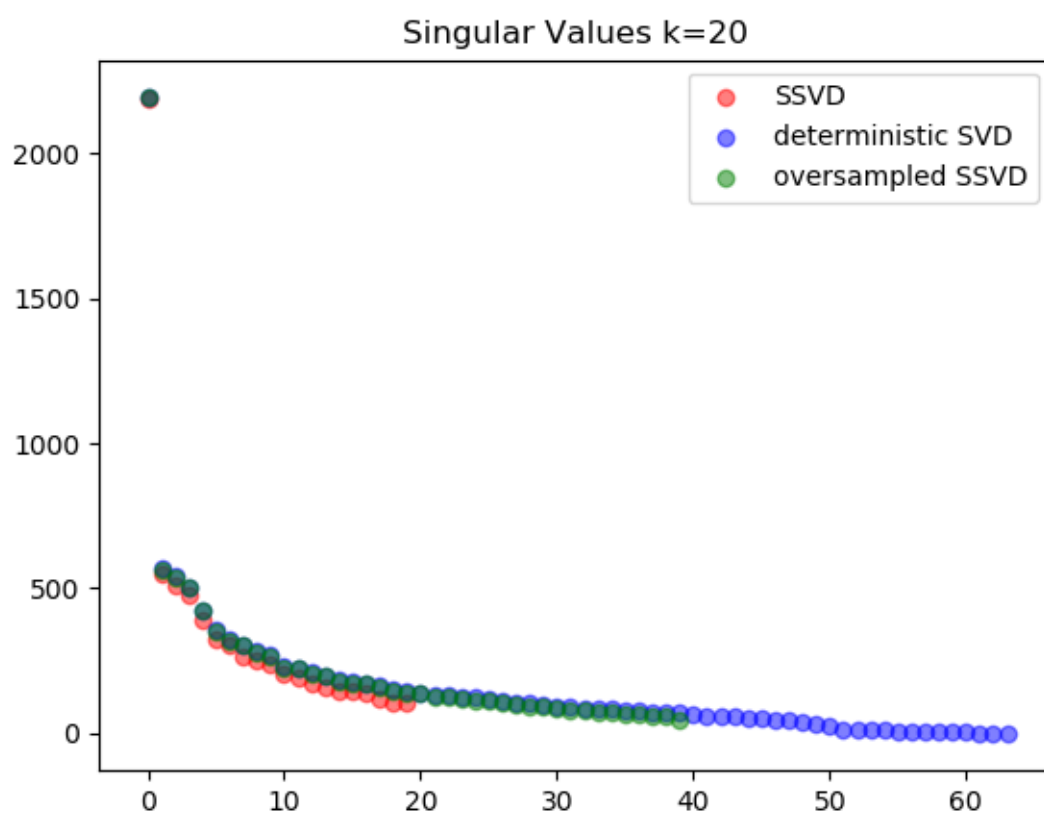


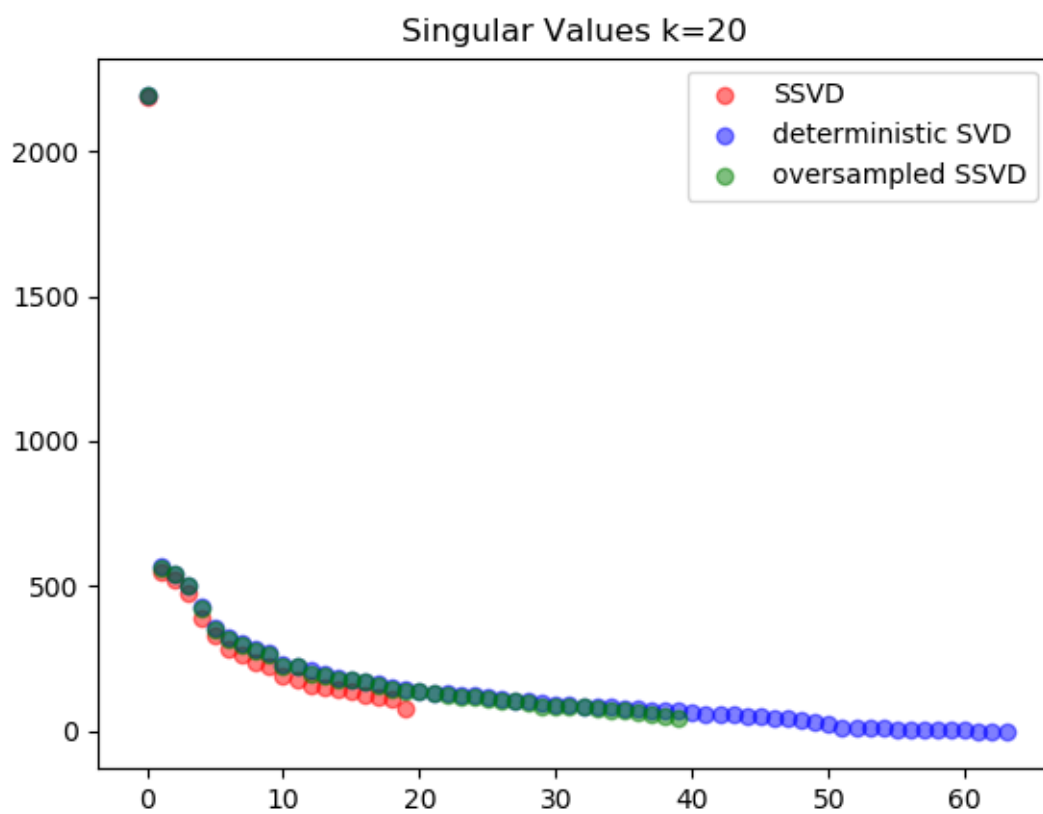


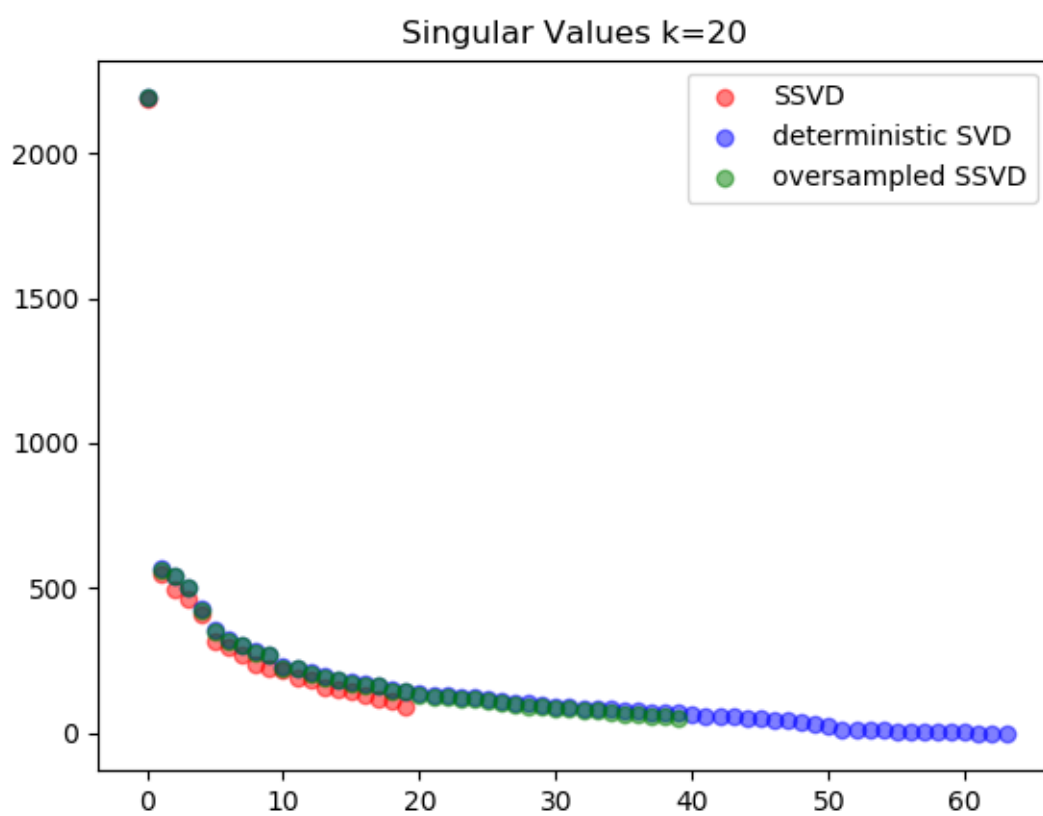


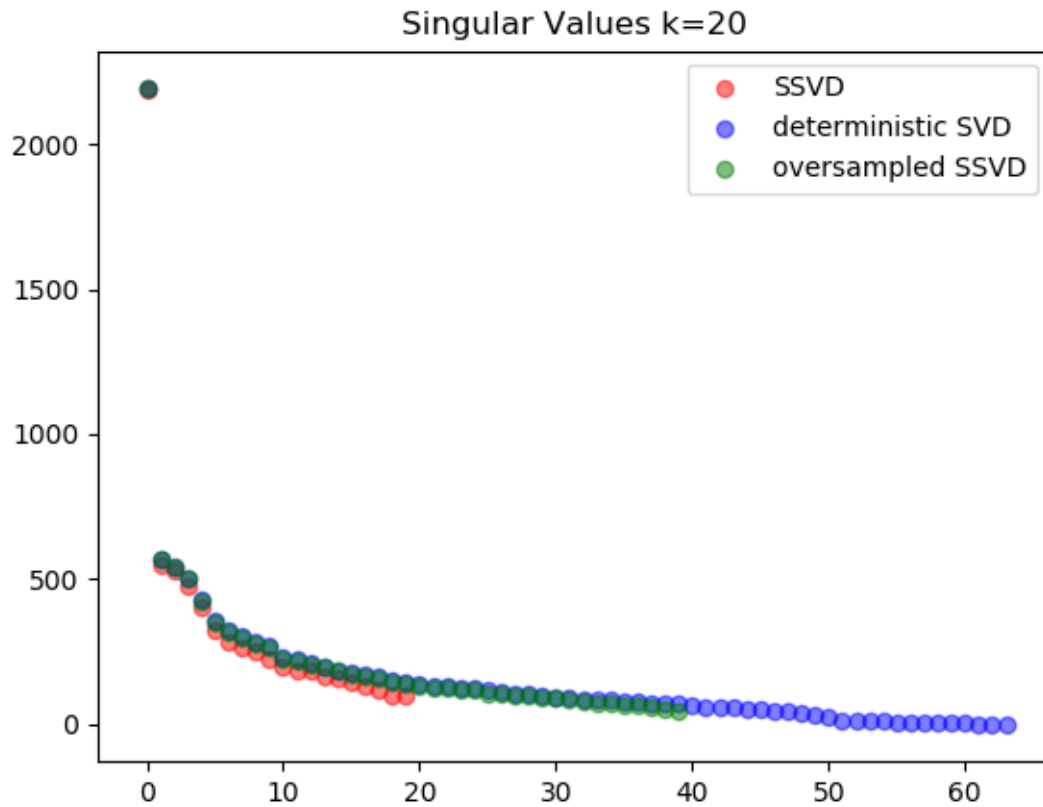












2

**BONUS [20pts]** 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  $Q$ .

Write a Python function that takes an initial  $Q_0$  preconditioner, a number of power iterations  $q > 0$ , and the data matrix  $A$ . It will return  $Q_q$ , the preconditioner with  $q$  power iterations applied to it.

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

Re-run your SSVD function on the MNIST data 10 times for  $q = 0, 1, 2$ , and 3 respectively (you can also retain the oversampling from before). Plot the singular values and compare them to the built-in SVD solver. How do they stack up?

**BONUS [10pts]** As of `scikit-learn` version 0.18, Nathan Halko’s SSVD solver has actually been integrated! Let’s see how your SSVD implementation compares to the “official” one. You’ll find this implementation if you use `sklearn.decomposition.PCA` with the `svd_solver` parameter set to “randomized” in the constructor.

Run this solver on the MNIST data, as well as your custom-built SSVD solver, and plot the resulting singular values. How do they compare? What about to a deterministic solver? Is your SSVD better than `scikit-learn`’s?

**BONUS [5pts]** If you implemented power iterations, you can also compare those directly against the one in `scikit-learn` via the “`iterated_power`” argument in the constructor (this is essentially  $q$ ). Again, try a few different values of  $q$  in your solver and the one in `scikit-learn` and plot the resulting singular values. How do they look?

**Include your code in a file named `assignment5_q3.py` when you submit to AutoLab, as this will be manually inspected. Include the plots in your write-up.**

## Administration

### 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_q2.py`: the Python script that implements kernel smoothing
2. `assignment5_q3.py`: the Python script that implements SSVD
3. `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!

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

```
> tar cvf assignment5.tar *.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 2, 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).

## 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.