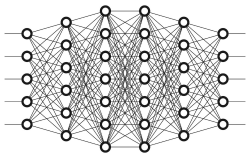


Massachusetts Institute of Technology
Dept. of Electrical Engineering and Computer Science
Fall Semester, 2018
MIT 6.S198: Deep Learning Practicum

Assignment 4: Embeddings and Generative Models

1.1: Visualizing datasets using the embedding projector (In-class Monday)

1. Spend some time using the embedding projector to make and share observations about the MNIST data. Do the different digits separate into distinct clouds?

The digits cluster in the space. For example, 1s tend to make a thin strip by themselves in the corner and are fairly distinct from another group. The 8s are in the center and overlap a bit with some other groups.

2. Are there images that are in the wrong cloud, and can you make sense of why they are wrong?

There are a few images that are in the wrong cloud. Some of these errors include 9s that look like 2s but rotated 180, 4s that look like 6s and 2s that look like incomplete 0s.

3. Are there images that are outliers from the rest of the data?

A few digits occupy the space in between the clusters as they look like intermediate shapes between two digits.

4. Are there digits that seem more separate from the others, and are there pairs of digits that are more easily confused?

Yes the 1st and 6s are separate and distinct from any other digits. On the other hand 4s and 9s overlap heavily.

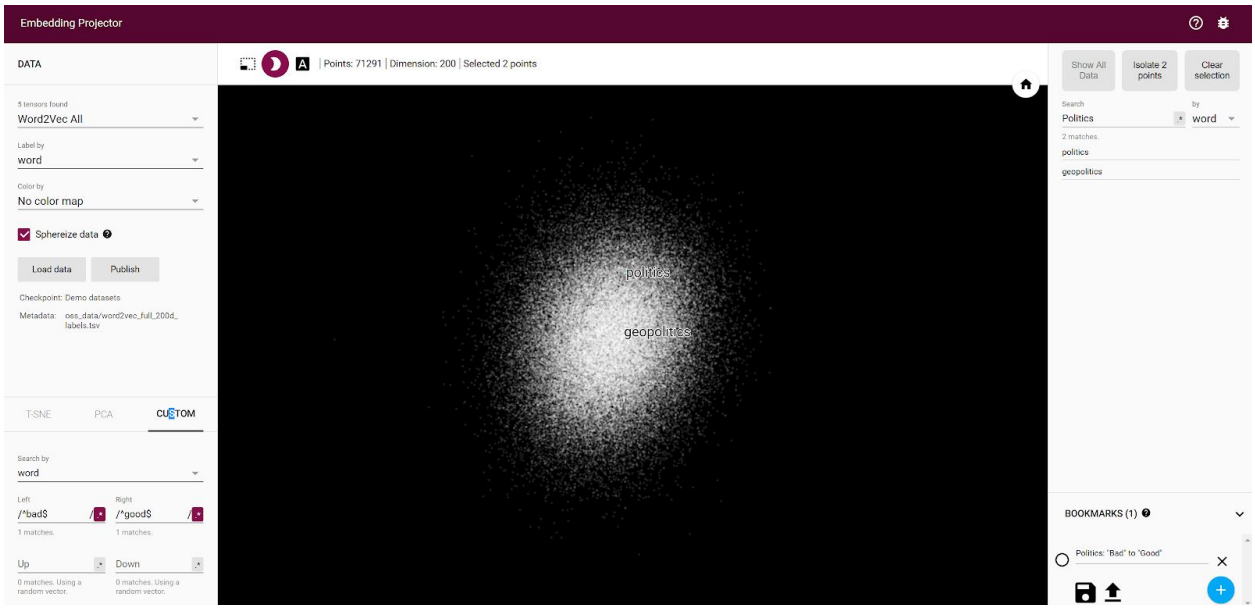
1.3: Word geometry

<WRITEUP REQUIRED>

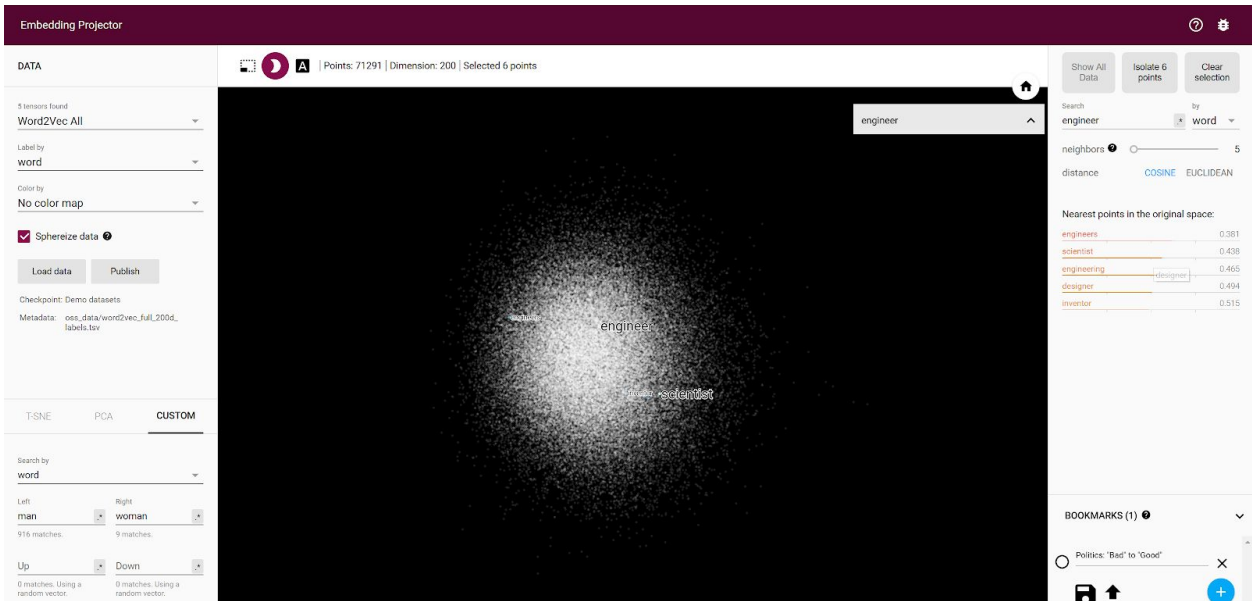
Remember that this geometry is not based on any word definitions, but rather only the frequencies with which words co-occur in phrases. Also keep in mind that the vertical positions of the words are random, although you can specify these, too, by setting “up” and “down”.

1. Experiment with various words to see if you can identify any insights about the data set. For example, try “politics” along the dimension from “bad” to “good”, or “engineer” along the spectrum from “man” to “woman”. Write up some notes on your observations, perhaps supplemented with a few pictures.

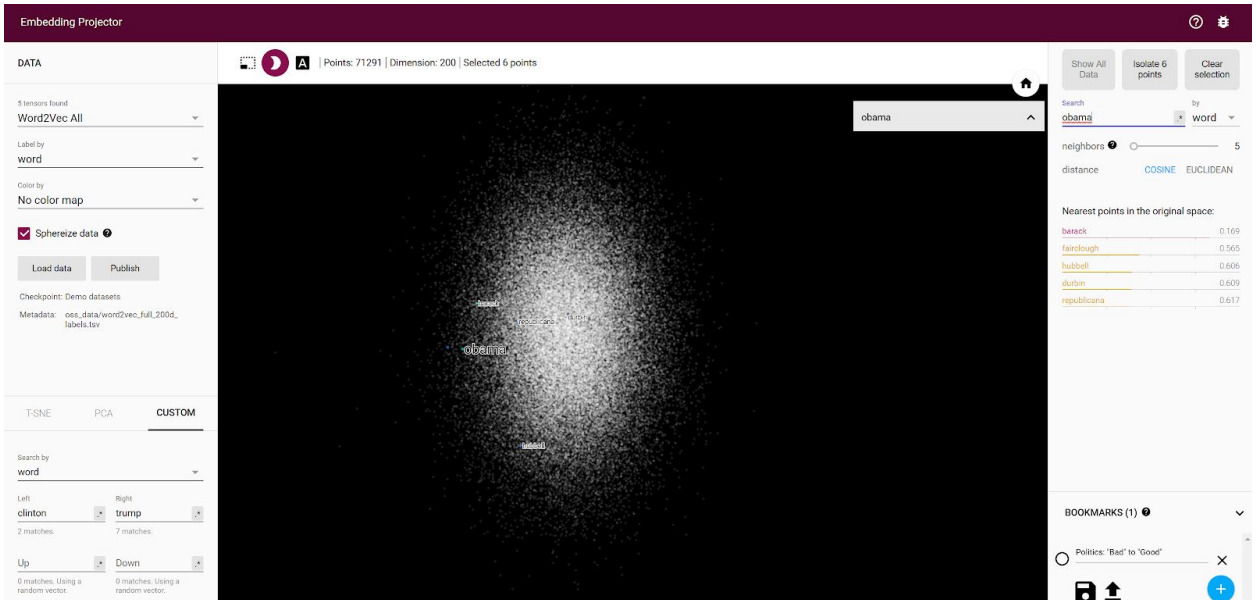
Politics is center right (in the good direction), but not heavily so.



Engineer is gender neutral but engineers is very male and scientist more female.

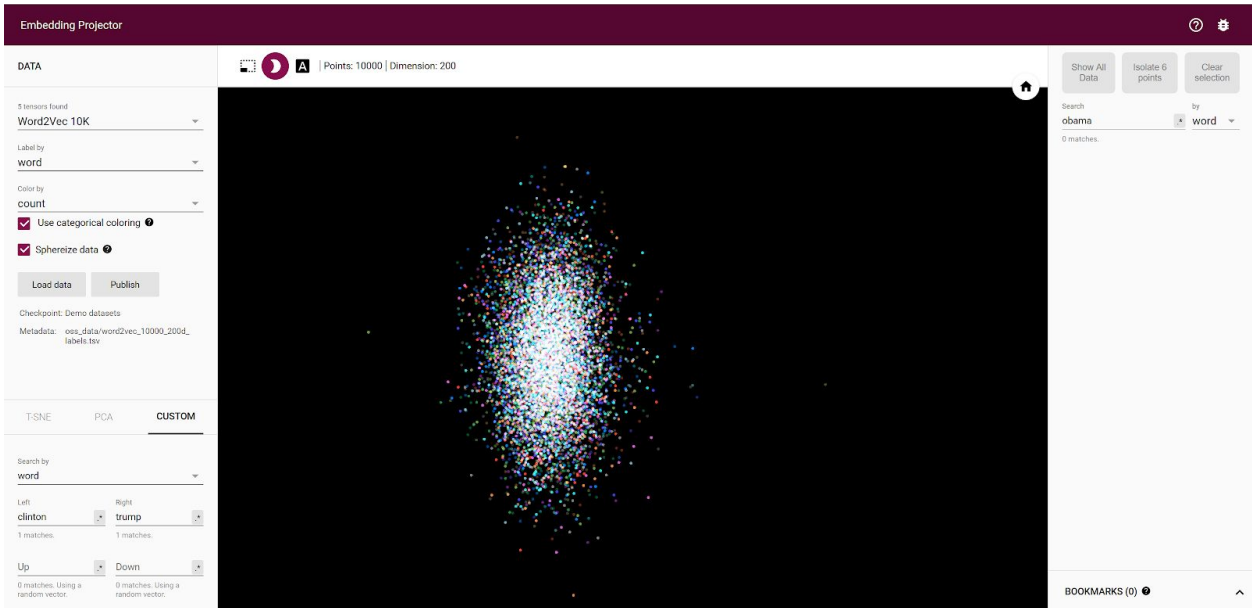


On the Clinton to Trump spectrum, Obama is pretty Clinton.

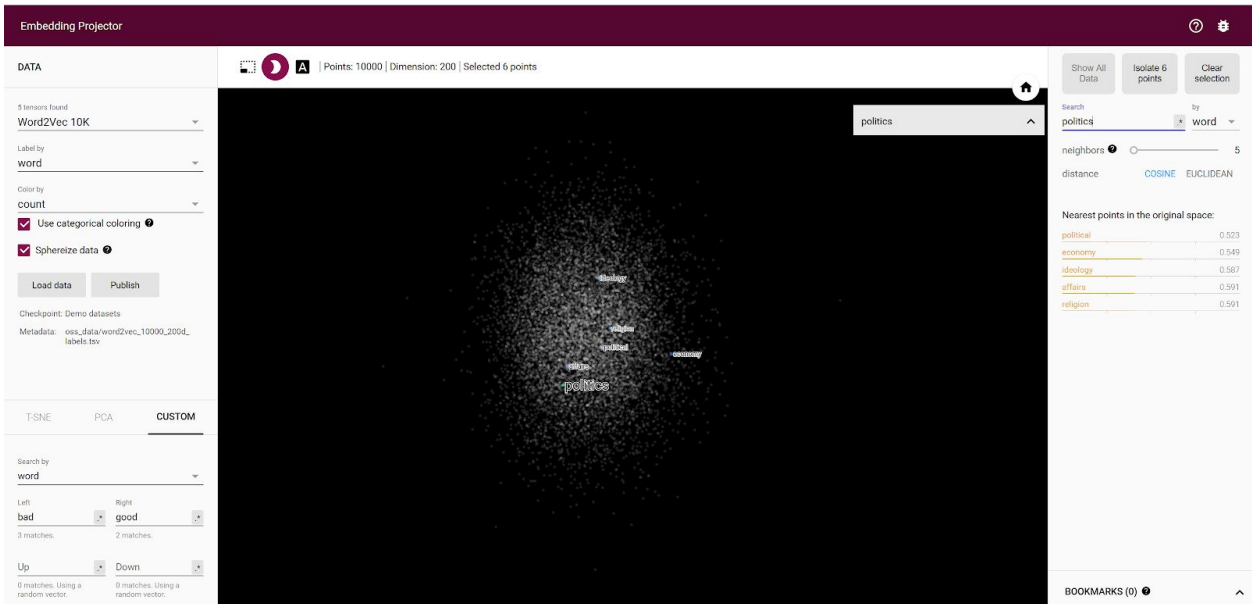


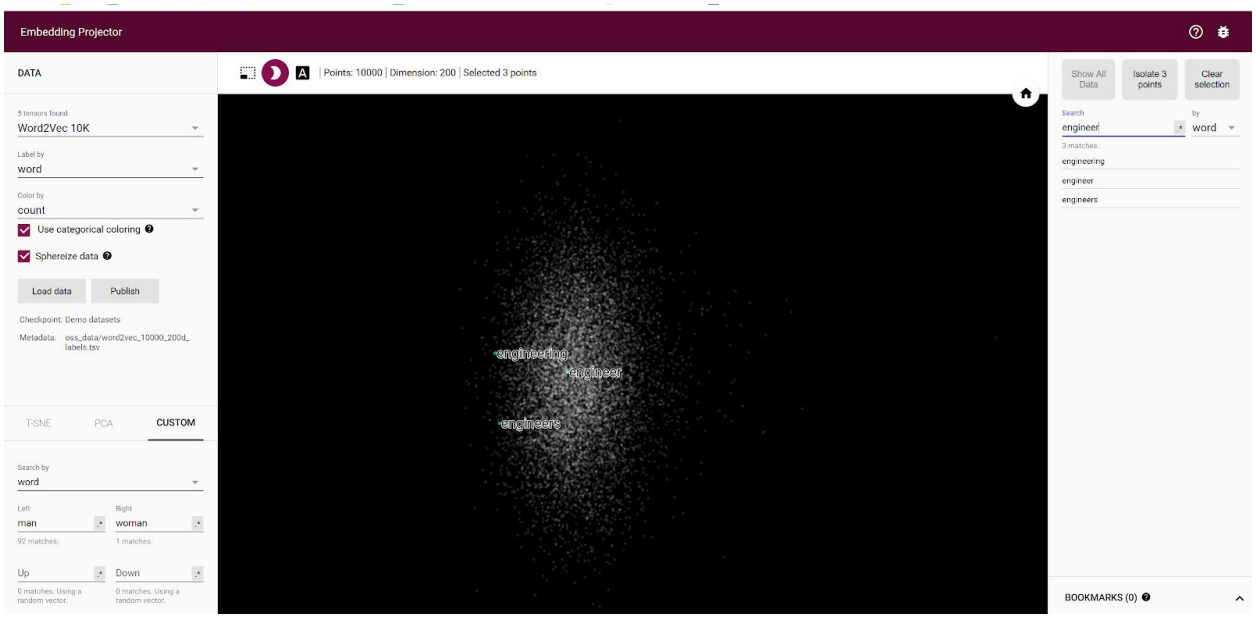
2. Do you get different or better results if you use Word2Vec All instead of 10K?

Some words are just not present int Word2Vec 10k, like Obama so the example from before breaks.



Otherwise it seems pretty similar on the examples we tested before.

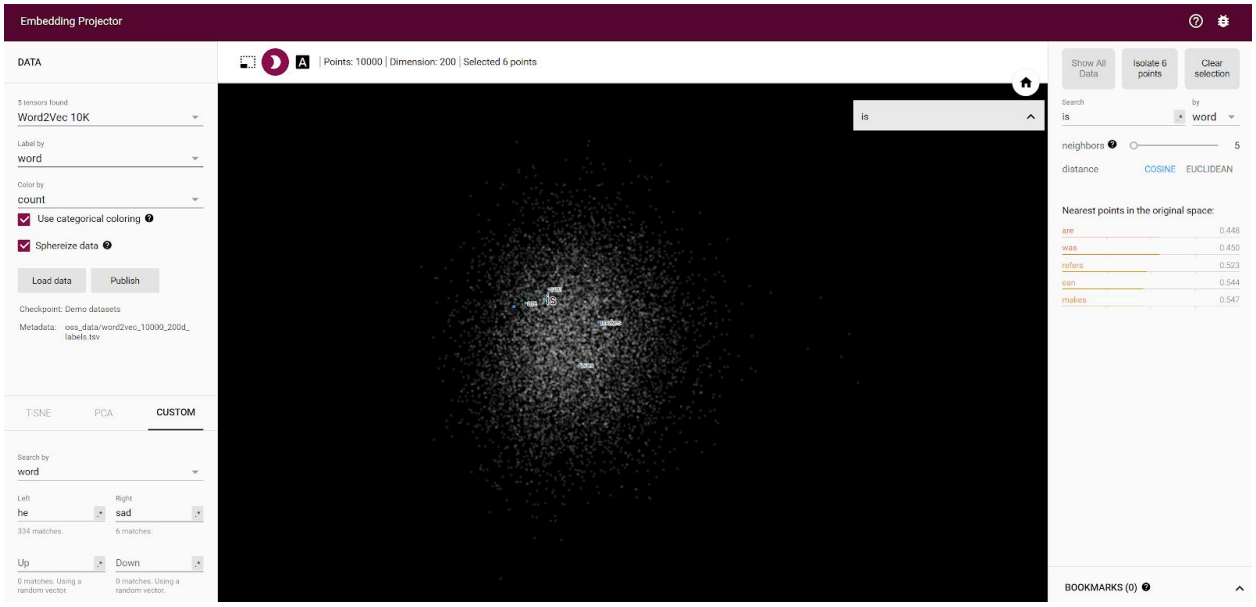




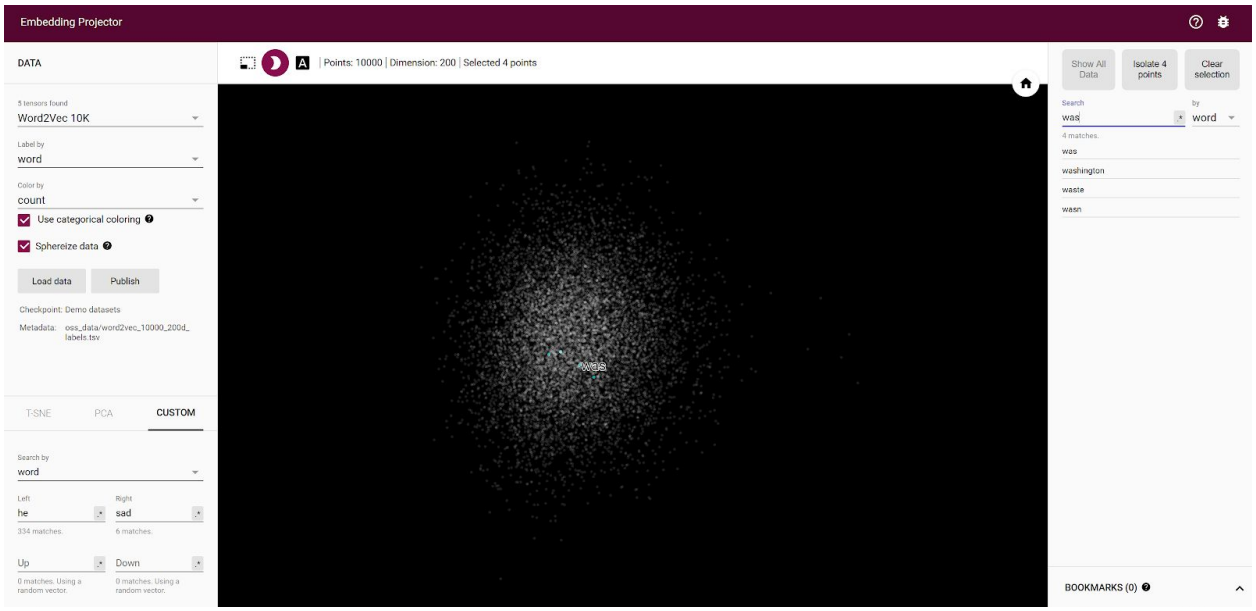
3. Did you find any interesting examples that speak to how words are used news articles?

If we go from a spectrum of he to sad, the words in the center tend to be words that you could use to make the sentence complete.

I.e. He is sad



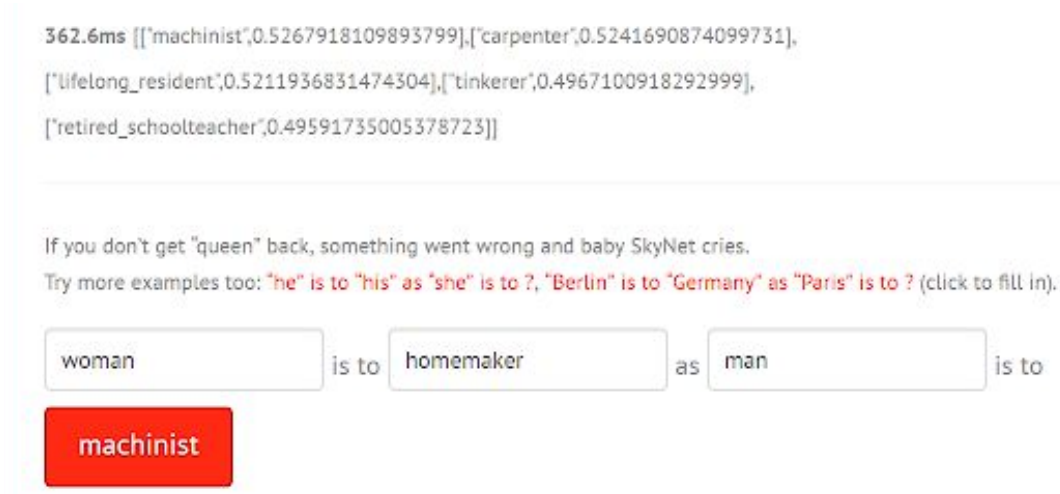
Or He was sad



1.4: Finding word analogies with vector algebra

1. Spend a few minutes experimenting with the demo on https://rare-technologies.com/word2vec-tutorial/ (under “Bonus App”) which uses Word2Vec vector algebra to solve analogies. Make a note of any interesting examples you find.

Man is to engineer as homan is to Homemaker has been corrected, but not in the reverse direction.



1.5: Exploring fonts with the Embedding Projector (All homework due before class next Wednesday)

<WRITEUP REQUIRED>
Do the following activities:

1. View the fonts with PCA embedding. Do you see any clumps/areas with obvious characteristics? Record a few Font IDs for distinct characteristics/groupings that you find interesting (hover over a character to get its font ID), such as bold, italics, cursive...etc. You will use this in the homework.

Fonts that surround the letter in circles: 3061,3155,138,3053,3156

2. Change to the embedding to T-SNE. Record how many iterations you let T-SNE run for for and whether or not you were able to get interesting groupings. Again, record Font IDs for interesting fonts/groupings.

104 Iterations and got an interesting sorting into a line.
Strange Shapes: 5793, 5748, 6678, 7886, 328

3. Find a font you like, get its ID, and type that into the search bar at the right-hand side of the screen. Use the "neighbors" slider to isolate a few dozen points and record the Font IDs of the 10 nearest neighbors that make sense. Repeat this for 3 or 4 fonts. If you find a font that doesn't have nearest neighbors that look similar, note that down as well.

Font with no neighbor: 4075
Fonts similar to 1385: 669, 1084, 7916, 3887, 3808, 6994, 180, 3803, 4144, 5648
Fonts similar to 7672: 6454, 3753, 6454, 2773, 6432, 1745, 6433, 3049, 7199, 849

Create a folder on your machine to hold the code. Then connect to that folder and run the following:

```
git clone https://github.com/mintingle/font-explorer.git
cd font-explorer
yarn prep
./scripts/watch-demo demos/latent-space-explorer/
```

http://localhost:4000/ should open automatically after a minute or two. Check that the page opens.

***If you have a Windows machine, please make note of the following changes to line 34 in the watch-demo file that you may need to make. We are attempting to execute the command "./scripts/watch-demo demos/latent-space-explorer/". Windows machines are unable to recognize the command stored in the constant cmd. If you are using a git bash shell, change line 34 to:

const cmd = `sh node_modules/.bin/poi${demoFolder }-o`;

After this, run:
node scripts/watch-demo demos/latent-space-explorer

If you are using the classic Windows command prompt, you might have to change line 34 to:
const cmd = `node node_modules/.bin/poi ${demoFolder} -o`;

and then run
node scripts/watch-demo demos/latent-space-explorer

2.1: Generative models

So far this semester, we've used deep networks to classify existing data objects. We can also use networks to generate new objects. Now, we'll generate new fonts.

On Monday, we saw how to think of embeddings in terms of a high-dimensional space with certain geometries. Each data element's coordinates correspond to attributes of the data element. Attributes that are computed rather than directly observed are called "latent attributes", and the embedding space is called "latent space". For the font example, the latent space is 40-dimensional. We can regard a collection of fonts a collection of points in latent space. You've already used the Embedding projector in the homework to look at a collection of 8,192 fonts. Now, you'll pick new points in latent space that are not in the original font set and generate new fonts corresponding to those points.

The font embeddings in the Latent Space Explorer were created by a training a multilayer neural network using the images for about 50,000 fonts. Each font is identified by a font Id, and each font has 62 characters—upper and lower case letters and a few more. Each character image is 64×64 =

4096 gray-scale pixels. The neural network is designed to convert font Ids and charter numbers to images of the characters.

In more detail, the input to the network consists of a character number from 0 to 61 and a font Id. The character number is one-hot encoded as a 62-dimensional vector. The font ID is represented as a single value that is used as an index into a table. The table has one row per font (50,000 rows in all) and one 40-dimensional vector per row. (The number 40 was chosen arbitrarily.) Given a font id and a character number, we append these to get 102-dimensional vector: the 40 numbers associated with the given row in the table, plus the one-hot encoded character number. This 102-dimensional vector will be the input layer to the network.

The output of the network is the 4096-dimensional layer that will determine a gray-scale value for each the pixel.

In between the input and output layers there are 3 fully-connected layers separated by ReLU layers. The width of the FC layers is 1024. (The number 1024 was chosen arbitrarily.)

The network can create an image given a character number and font id: Use the associated 102-dimensional vector as the input to the first layer of the network and propagate this through the network. The output layer then specifies the $64 \times 64 = 4096$ pixel values for the image.

We train the network as follows: We start by initializing the table row to 40 random values. Then, for each character and font, generate the candidate image, compare that with the true image, which we know since we have the actual font and use back propagation to adjust both the weights in the layers and the values in the table. We stop when we've done a lot of training with all the fonts. (The network took weeks to train for 50,000 fonts.)

Once the network has been trained, we'll take the 40 values in the table row for a font to be the 40 attributes of the font and use those attributes for the embedding into 40-dimensional latent space.

2.2: Generate new fonts with the Latent Space Explorer

You'll now use an application called the "Latent Space Explorer" to generate new fonts. This application uses the full collection of ~50k fonts, rather than only the ~8k fonts displayed in the

embedding projector. To generate characters in a new font, use the same trained network that produced the attributes. Change the value of some of attributes, use the changed attributes plus character Ids as input to the network, and take the resulting output images.

You'll generate new fonts in the way, using the Latent Space Explorer source code that you installed before class (see instructions above). To run it, connect to the installation directory you created for the code, and run:

./scripts/watch-demo demos/latent-space-explorer/

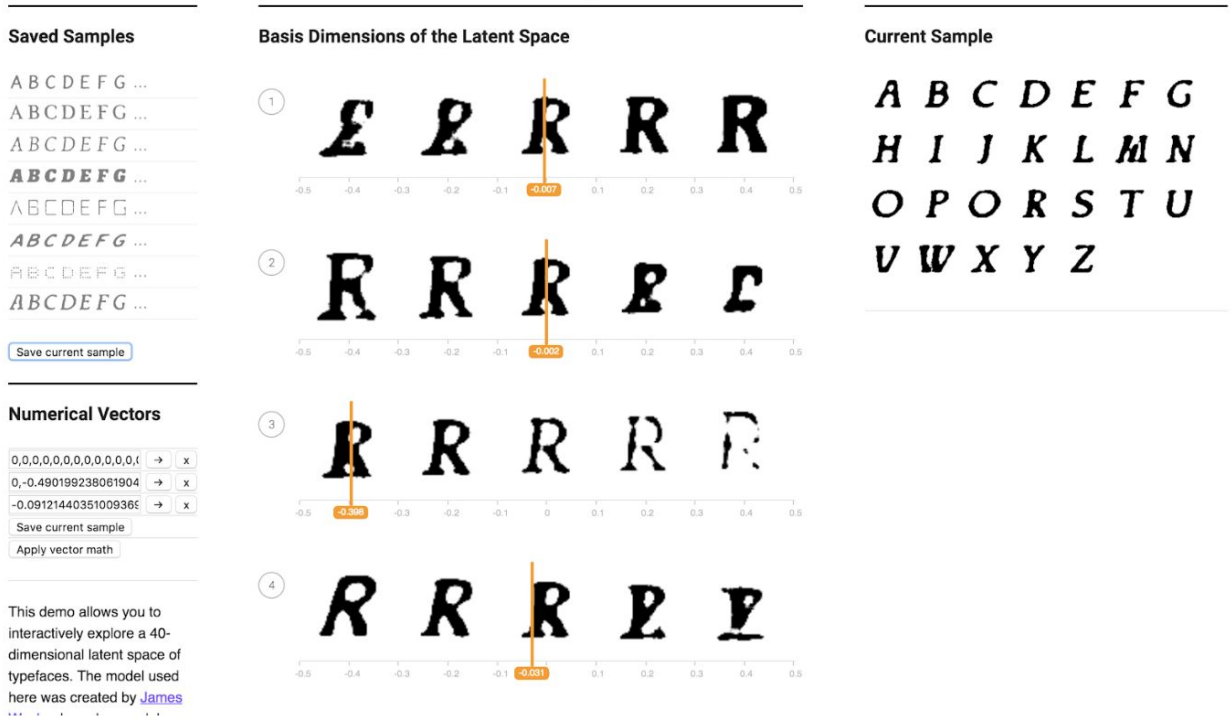
The page http://localhost:4000/ should open automatically.

In the right-hand column, you see the entire lowercase alphabet for the currently displayed font.

In the middle column, you see the values for all 40 dimensions of the current font. The example character being displayed along each of the 40 attributes is currently "A". For each attribute, you may click anywhere along the -0.5 to 0.5 axis to change that attribute for the entire font. For example, dragging the first attribute's value closer to 0.5 will make the font bolder and blockier, the change of which will be reflected in the right column. When one attribute is changed, the range of how all of the other attributes look will also be changed accordingly. Tweaking these attributes essentially allow you to generate new fonts that may not be part of the 50k fonts currently in the space.

In the left-hand column, you first see a short list of "Saved Samples". You may click on any of them to display that font -- initially, the first font is displayed. The "Save current sample" button allows you to add the currently displayed font to that list. Below that are the numerical vector tools. The first [0, 0, 0,...0] text field is a saved 40-dim font vector. You may press the → button to load that font or press the × button to delete that vector. The "Save current sample" button allows you to add the currently displayed font as a saved 40-dim vector. The screenshot below, shows current sample and with two new font vectors added. Clicking the "Apply vector math" button just adds the [1, 0, 0,...0] vector to the currently displayed font. You will be editing this button's functionality later on to do more useful things.

Spent a little time looking a different fonts and playing with the vector attribute numbers to see if you notice anything interesting.



2.3: Creating new fonts

<WRITEUP REQUIRED>
For homework, you will be modifying the latent space explorer code to create new fonts with various characteristics. Make sure to clearly writeup what font IDs you used for each question and take screenshots of your results.

The code files that you'll be modifying are in deeplearnjs/demos/latent-space-explorer. These files are components/VectorChooser.vue (Vue is just a JS framework; you will need to write only basic JS in this file) and utils/FontModel.js. Please briefly look through these files and familiarize yourself with their content.

Of course, you're welcome to look at the other code files and make changes as you like, but the above files include all that is required for this assignment.

0. Edit FontModel.js so another character besides "r" is being displayed as the sample character for the 40 attributes. Edit Alphabet.vue so that all of the uppercase, lowercase, and numerical sample font characters are displayed in the right side of the screen instead of just the lowercase characters. Take a screenshot for your writeup.

1. Edit VectorChooser.vue to add a new button under the "Apply vector math" button. The new button should show the font Id of the "nearest neighbor" font that is "most similar to" the current font, out of all the fonts in the 50K training set.

The file deeplearnjs/demos/latent-space-explorer/embeddings.json contains a large array where the 0th element is the 40-dim vector for font Id 0 and so on for all 50K fonts.

As mentioned in lecture for Assignment 1, the similarity of two vectors can be taken to be the "cosine similarity" which ranges between 0 and 1. This can be computed as the absolute value of the dot product of the two vectors, divided by the product of the norms of the two vectors. Compute the similarities of the current font to each of the 50K fonts. Use console.log to record the similarity values and the font IDs as you generate them.

When you obtain the nearest neighbor Font Id, navigate to the Font Finder website at https://courses.csail.mit.edu/6.s198/spring-2018/fontfinder/. This is an app that lets you input a font Id and shows you the font. Document your work in your writeup and show a few examples. Can you find examples of nearest neighbor fonts that don't make sense?

2. What is the “average” of a set of fonts? (Hint: What does that mean in terms of the 40 dimensions)? Write some code to Find the nearest font neighbor to the average font.

3. Similar to Monday's word analogy exercise, we will now be working with font analogies. Feel free to use the Font Finder website to explore different fonts and obtain their attribute vectors.

a. Here is a basic "bolding vector" that Natalie obtained by getting the vector of thickest font in the “Saved Sample” space:

“0.053,0.026,-0.060,-0.014,-0.093,-0.018,0.087,0.096,-0.039,0.071,0.111,-0.022,-0.007,0.022,-0.032,0.141,0.126,-0.015,-0.075,0.120,0.068,0.021,0.104,-0.065,-0.085,-0.018,-0.038,-0.059,-0.051,-0.056,0.154,-0.044,-0.116,0.038,-0.144,-0.103,-0.032,0.059,-0.076,-0.030”

Try to add this vector to any font. You may do the basic vector math by modifying the “Apply Vector Math” button in VectorChooser.vue or adding a new button/functionality.

What does adding this vector to a font do? Does it perform bolding well at thickening the original font? Are there other characteristics that seem to also carry over? Try this with another characteristic. It may be helpful to use the characteristics you gathered previously.

b. Find 10 fonts for a specific quality (i.e. bolding, dotting, fancy, serif...etc) and average them out to find a characteristic vector (for example, “bolding vector”). Try applying that vector to another font. Again, you can do this by manual computation and then modifying the added vector in the "Apply Vector Math" button, or you can make new buttons. Does it work better or worse than the previous one-sample method? Try this with at least 2 characteristics.

c. Find 10 fonts the exhibit for a specific characteristic and 10 fonts for the opposite of that characteristic (i.e. bold vs. corresponding non-bold). Subtract the vectors for each pair and take the average difference. How does this work as a vector for the characteristic (e.g., as a "bolding vector")? Does this work better or worse than the method above to obtain your desired characteristic? Try this with at least 2 characteristics.

4. Can you figure out how to create a vector that makes uppercase fonts lowercase? (Hint: You can try finding fonts that only have uppercase/lowercase characters and finding analogous vectors, or you can try playing with the 40 attribute sliders directly.)

5. Can you create a vector based on your personal font tastes (i.e. a personal-likeable vector)? Please also make note of any interesting vector directions you find.

6. Link your code in a text-readable format with your submission.

3: Submission

Create a page for assignment 4 on your class homework submission site. Include your name and email address and the required writeups as indicated above together with code and images, as appropriate.

Use This form to hand in Assignment 4
Due Wednesday, October 10 at 10AM.