Autoencoders

December 21, 2017

1 Simple Autoencoder for Principle Component Analysis

1.1 Create some data and scale it

1.2 The Linear Autoencoder

1.2.1 Placeholder

Notice there is no real label here, just X.

1.2.2 Layers

Using the fully_connected layers API, we do not provide an activation function!

1.2.3 Loss function

1.2.4 Optimizer

1.2.5 Init

1.2.6 Running the Session

2 Linear Autoencoder Exercise

2.1 The Data

1. Import numpy, matplotlib, and pandas.

2. Use pandas to read in the csv file called anonymized_data.csv . It contains 500 rows and 30 columns of anonymized data along with 1 last column with a classification label, where the columns have been renamed to 4 letter codes.

3. Take a look at the head.

4. Take a look at the info().

2.2 Scale the data

5. Use scikit learn to scale the data with a MinMaxScaler. Remember not to scale the Label column, just the data. Save this scaled data as a new variable called scaled_data.

2.3 The Linear Autoencoder

6. Import tensorflow and import fully_connected layers from tensorflow.contrib.layers.

7. Fill out the number of inputs to fit the dimensions of the data set and set the hidden number of units to be 2. Also set the number of outputs to match the number of inputs. Also choose a learning_rate value.

8. Create a placeholder fot the data called X.

9. Create the hidden layer and the output layers using the fully_connected function. Remember that to perform PCA there is no activation function.

10. Create a Mean Squared Error loss function.

11. Create an AdamOptimizer designed to minimize the previous loss function.

12. Create an instance of a global variable intializer.

2.4 Running the Session

13. Now create a Tensorflow session that runs the optimizer for at least 1000 steps. (You can also use epochs if you prefer, where 1 epoch is defined by one single run through the entire dataset.

14. Now create a session that runs the scaled data through the hidden layer. (You could have also done this in the last step after all the training steps.

15. Confirm that your output is now 2 dimensional along the previous axis of 30 features.

16. Now plot out the reduced dimensional representation of the data. Do you still have clear separation of classes even with the reduction in dimensions? Hint: You definitely should, the classes should still be clearly seperable, even when reduced to 2 dimensions.

3 Stacked Autoencoder

3.1 Parameters

3.2 Activation function

3.3 Placeholder

3.4 Weights

Initializer capable of adapting its scale to the shape of weights tensors.

With `distribution="normal"`, samples are drawn from a truncated normal distribution centered on zero, with `stddev = sqrt(scale / n)` where n is: - number of input units in the weight tensor, if mode = "fan_in" - number of output units, if mode = "fan_out" - average of the numbers of input and output units, if mode = "fan_avg"

With `distribution="uniform"`, samples are drawn from a uniform distribution within [-limit, limit], with `limit = sqrt(3 * scale / n)`.