A Neural Network to Predict Redshift

Collin Farquhar

Introduction

The program, nn.cpp contains a neural network, written completely from scratch in C++. The neural network takes as input data color-intensity and galaxy-type of galaxies generated by the Sloan Digit Sky Survey [1]. Using these features the neural network then predicts the redshift of the galaxies.

There were there major tasks that I completed in writing this code for this neural network. First, writing a linear algebra library. Second, preprocessing of the data. And third, writing the neural network.

Motivation

The Sloan Digital Sky Survey is an ongoing project that measures the redshift of millions of objects in space. “Redshift and blueshift describe how light shifts toward shorter or longer wavelengths as objects in space (such as stars or galaxies) move closer or farther away from us” [2]. Redshift, z, is calculated by [2].

The data the neural network will primarily use as features is u, g, r, i, z (ultraviolet, green, red, near-infrared, infrared) color data. A bit unintuitively, the color data is given on a log scale running backwards, so lower values are more intense. The group of Ryan Rubenzhal, et al., advised by Matthew Graham at the La Serena School for Data Science claim accurate redshifts to accurately measure distances for use in containing the dark energy equation of state.

Writing a Linear Algebra Library

Setting out to write a neural network from scratch (something I have never attempted before), I knew that I would need access to some common linear algebra operations. The program matrix.hpp contains several useful function for use with arrayt.hpp[3] vectors and matrices. Matrix.cpp contains functions for the following uses:

Functions:

  dot:

      matrix multiplication

      matrix vector multiplication

      vector vector dot product

  transpose:

      transposes matrix or vector

  multiply:

      element-wise multiplication of matrices or vectors

  applyFunction:

      applies a function to each element of matrix or vector

  Overwrites '-' for matrices and vectors:

      element-wise subtraction

  Overwrites '+' for matrices and vectors:

      element-wise addition

  Overwrites '\*' for double and matrix/vector:

      scalar multiplication

  print:

      outputs matrix or vector

To test these functions and verify that they work as intended, one can run matrix\_test.cpp. This program showcases the functionality of matrix.cpp, and outputs results so that the user can easily see the results of the various functions.

Preprocessing the Data

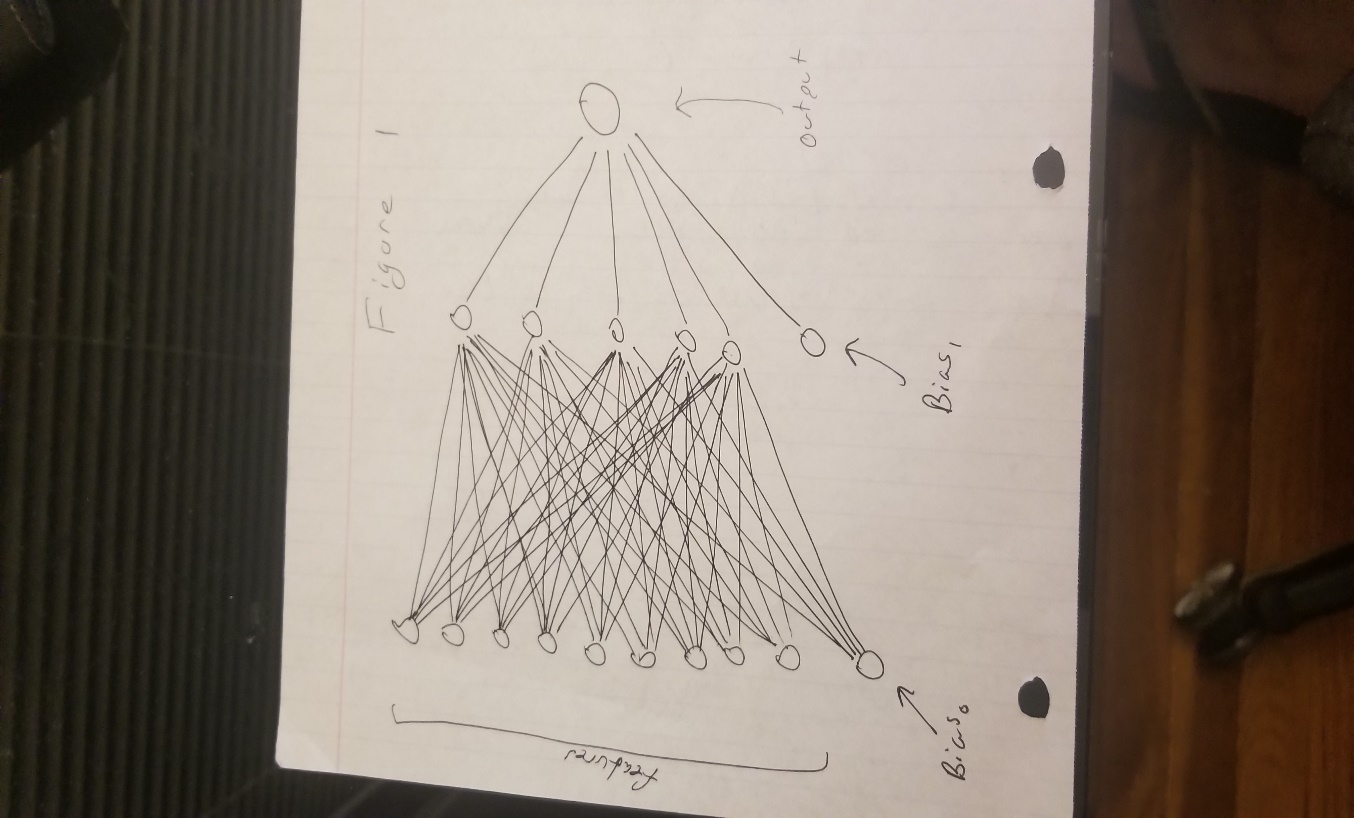
It’s likely no understatement that the data is the most important, and fundamental, component to the success of neural network (or other machine learning algorithm). The original data from the SDSS website contains a variety of data types; some data is irrelevant, some needs to be scaled, as well as the redshift data the neural network is designed to predict.

To handle the preprocesses of the data, I wrote a program, prep.py, that reads the data, separates relevant training data and redshift data, processes the training data, and outputs the training and redshift data to a csv/txt file. In the processing of the training data, it is important that the data all be of the same scale. This is achieved in the preprocess() function which for every color feature z

Where and are the mean and standard deviation a given feature. This centers the data at 0, and within a relative spread. Some data had information about the type of galaxy in text format, classifying as Starburst, Starforming, Broadline, AGN, and AGN Broadline. To make use of this data, for each classification of galaxy I added a feature dimension to each data example such that the value of the feature would be 1 if the data was of that type, and 0 otherwise.

Prep.py also calculated the average redshift. This value will be used to evaluate the performance of the neural network.

Figure 1



Writing the Neural Network

The data was preprocessed such that there are 10 feature values for each data example, thus 10 nodes in the input layer. Each data example has a corresponding y-value for redshift that the network is trying to predict. This is a regression problem, so there is one output node, that is the predication. I set one hidden layer with 5 nodes between the input and output layers. There are two bias nodes, one connecting to every node in the hidden layer, and one connecting to the output. A drawing of the network can be seen in figure 1.

Inputs are propagated through the network to nodes in the next layer. They are scaled by a weight, and the sum of weighted inputs to a node in the hidden layer is then passed to an activation function, which allows the neural network to capture nonlinearities[4].

The process of forward propagation is given by[5]

Where is a vector of the values of the nodes in the hidden layer, is the input data, is a matrix of weights, connecting each input node to each hidden layer node, , is the bias (set equal to 1) and , is a leaky ReLU activation function applied to the weighted sum of inputs. This is a lesser known activation function, but I chose it so that nodes with negative values (or zero) would still have a nonzero gradient. I set the “leak” as a hyperparameter to be 0.5.

Then, it is a similar process to propagate from the hidden layer to the output node, . There is no activation function applied to the weighted sum of inputs to the output node, as this is a regression task in which we want to predict the exact value of redshift.

After forward propagation, we can compute the error and propagate the error backwards using gradient descent to adjust the weights. This neural network implements stochastic gradient descent. It computes back-propagation and updates the weights for every example giving to the network during training. I use a square loss function:

Where the factor of ½ is there for convenience when taking the derivate of the loss function.

The general formula for weight updates is [6]

Where is the learning rate, a set hyperparameter and is computed through the chain rule.

Conclusion

The loss function for the training data that we are trying to minimize is no strictly convex, thus the random initialization of the weights will lead to difference solutions. I think through better tuning of the neural network, I could improve the performance and make it more likely to find the global minimum.

To evaluate the performance of the network, I calculated the average redshift, and then calculated the mean squared error if you were to predict the average for every example, MSE = 0.0209728. I then ran the neural network on new data from SDSS training data that was not used for training. Depending on the initialization, the neural network frequently performs better than the average. The industry standard for calculating redshift is *Eureqa*, which claims an MSE of 0.00392, and has a $5000 license. Some initializations of my network beat this standard. For one initialization, I had MSE = 0.00259016. The final weights for this initialization as well as others are included at the end.

Before I sell my code for $6000, an important next step is to reduce network variance, and validate for overfitting. Regardless, the data shows that the network performs quite well, and still could be improved. I think it may be worth looking into improving this neural network further.

**References**

[1] <https://www.sdss.org/dr14/>

Funding for the Sloan Digital Sky Survey IV has been provided by the Alfred P. Sloan Foundation, the U.S. Department of Energy Office of Science, and the Participating Institutions. SDSS-IV acknowledges

support and resources from the Center for High-Performance Computing at

the University of Utah. The SDSS web site is www.sdss.org.

SDSS-IV is managed by the Astrophysical Research Consortium for the

Participating Institutions of the SDSS Collaboration including the

Brazilian Participation Group, the Carnegie Institution for Science,

Carnegie Mellon University, the Chilean Participation Group, the French Participation Group, Harvard-Smithsonian Center for Astrophysics,

Instituto de Astrof\'isica de Canarias, The Johns Hopkins University, Kavli Institute for the Physics and Mathematics of the Universe (IPMU) /

University of Tokyo, the Korean Participation Group, Lawrence Berkeley National Laboratory,

Leibniz Institut f\"ur Astrophysik Potsdam (AIP),

Max-Planck-Institut f\"ur Astronomie (MPIA Heidelberg),

Max-Planck-Institut f\"ur Astrophysik (MPA Garching),

Max-Planck-Institut f\"ur Extraterrestrische Physik (MPE),

National Astronomical Observatories of China, New Mexico State University,

New York University, University of Notre Dame,

Observat\'ario Nacional / MCTI, The Ohio State University,

Pennsylvania State University, Shanghai Astronomical Observatory,

United Kingdom Participation Group,

Universidad Nacional Aut\'onoma de M\'exico, University of Arizona,

University of Colorado Boulder, University of Oxford, University of Portsmouth,

University of Utah, University of Virginia, University of Washington, University of Wisconsin,

Vanderbilt University, and Yale University.

[2] Estimating Photometric Redshifts Using Symbolic Regression. La Serena School for Data Science. August 28, 2018. Elisabeth Younce, Brianna Thomas, Manuel Ignacio P ́erez, & Ryan Rubenzahl.

[3] Dr. Kirkland, Cornell University

[4] SEARCHING FOR ACTIVATION FUNCTIONS. Prajit Ramachandran∗ , Barret Zoph, Quoc V. Le Google Brain <https://arxiv.org/abs/1710.05941>

[5] <http://www.cs.toronto.edu/~hinton/csc2515/notes/lec6tutorial.pdf>

[6] Lecture notes of Dr. Kilian Weinburger http://www.cs.cornell.edu/courses/cs4780/2018fa/lectures/lecturenote20.pdf

Some Sample Data

stopping at iteration 3638

w0:

1.08322 0.994607 0.955597 1.1233 0.267193

1.70128 1.57029 0.953394 1.07461 1.15906

-0.308198-0.0396831 0.438816 -0.276622 0.583375

-0.195415 0.217287 0.421119 0.439484 0.397997

-0.318999-3.06654e-05 -0.327183 0.032619 -0.304915

0.318663 0.294366 -0.352976 0.0994495 0.055624

-0.17027 -0.312268 0.0528477 0.0989968 0.439002

-0.376378 0.192396 0.034494 0.366539 -0.307043

-0.0206881 0.455186 -0.299869 0.384371 0.292677

-0.126386 0.274927 -0.32441 0.216746 -0.373289

0.134668 0.242579 0.635138 -0.409797 -0.27478

W1:

-0.0301922

-0.0116052

0.61485

0.18685

-0.35206

-0.029172

validation mse = 0.00259016

benchmark mse = 0.0209728

stopping at iteration 3526

w0:

0.16026 0.606545 0.622978 0.0240696 -0.142682

0.339061-0.0161747 0.437616 0.322708 0.579272

0.165919 -0.336599 0.430645 0.332572 -0.328458

-0.0112985 0.204976-0.0950274 0.340944 0.363707

0.180038 -0.348082 0.413193 0.151549 -0.259766

0.432997 0.0539425 0.243644 0.40345-0.0275756

0.140458 0.494934-0.0124101 0.0772262 0.188251

-0.255822 0.116988 -0.325956 0.466153 -0.298869

0.105926 -0.150701 0.41884 -0.160454 -0.487506

0.0272423 -0.223028 0.0444864 -0.154769 -0.150408

-0.415282 -0.374546 0.208778 -0.400066 0.19574

W1:

-0.246459

-0.266364

0.00539997

0.464048

0.374695

0.250709

validation mse = 0.00420434

benchmark mse = 0.0209728

------------------------------------------------------------------------------------------

stopping at iteration 8

w0

-0.339131 0.251554 0.185078 0.188069 -0.164508

-0.0806236 -0.474175-0.0846544 0.0415441-0.0675371

-0.446939-0.0268207 -0.120543 0.444066-0.0238509

-0.435486 0.333458 -0.346517 0.371497 0.401804

-0.19845 0.175586 0.119216 -0.141735 0.201192

0.284507 0.219966 0.0202306 0.0705531 -0.355874

0.0378675 -0.127406 -0.159263 0.382339 0.2677

0.102064 -0.303005 -0.379979 -0.327596 0.253689

-0.296203 -0.186806 0.387925 0.223869 0.114019

-0.481898 -0.15766 0.324748 0.0346514 0.438518

-0.413836 0.404356 -0.129131 -0.135252 -0.419498

W1:

-0.243786

0.220132

-0.137041

-0.259776

-0.194946

0.13859

validation mse = 0.00906467

benchmark mse = 0.0209728