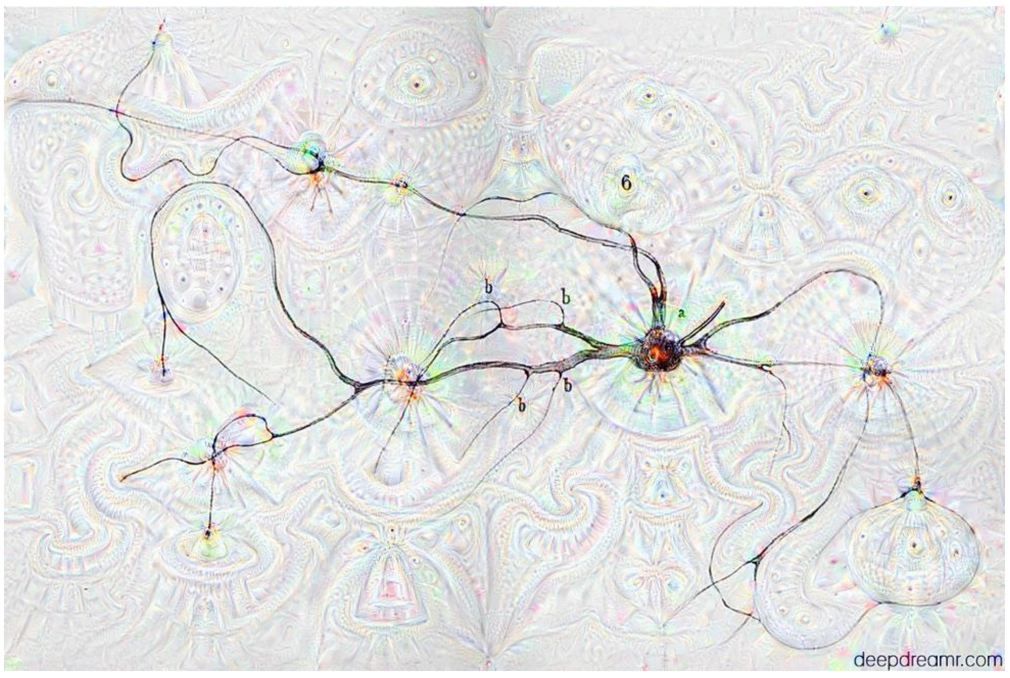
Neural Networks



...when robots hallucinate...
--The Atlantic

Introduction

In this project you will implement a Neural Network. First create a GitHub Classroom team and clone the project4 repository.

The code for this project (`project4`) consists of several files, some of which you will need to read and understand in order to complete the assignment, and some of which you can ignore.

Files you'll edit:

`preprocess.py`	Make the training data set zero mean and each feature should have the standard deviation of 1.
`grdescent.py`	Performs gradient descent. You can use your code from a previous project.
`forward_pass.py`	Computes the output for one pass through the neural network
`compute_loss.py`	Given the output of a network, computes the loss
`backprop.py`	Given the output of a network, computes the gradient of the weights

Files you might want to look at:

`initweights.py`	This function initializes the weights of the network given the structure of the network.
`deepnet.py`	Computes the loss and gradient for a particular feed forward neural net. Calls `forward_pass.py`, `compute_loss.py`, and `backprop.py`.
`bostondemo.py`	This script visualizes the RMSE error on the boston data set.
`bostontest.py`	This is a copy of the function the autograder will use to assess your code.
`best_parameters.pickle`	This stores the data used to evaluate your performance by the autograder

Allowed Libraries: Do not import any additional libraries in any file. This will cause the autograder to fail since using only numpy will be sufficient for a successful implementation.

How to submit: You can commit your code through the command line with git and submit on Gradescope either in a zip file or through Github. If the project is submitted before the initial deadline passes, you will receive information and a score for the performance evaluation (only once the deadline is reached). However, the autograder will not reveal any information on how your code performed for any projects submitted during the three day extension period. You can submit your project as many times as you want but the final submission score will count for your grade. If you submitted by the initial deadline and would like to improve your performance score, you can submit again during the extension period.

Grading: Your code will be autograded for technical correctness, efficiency, and performance. Please *do not* change the names of any provided functions or classes within the code, or you will wreak havoc on the autograder. However, the correctness of your implementation -- not the autograder's output -- will be the final judge of your score. If necessary, we will review and grade assignments individually to ensure that you receive due credit for your work.

PYTHON Version in Autograder: The autograder uses PYTHON 3.6. To rule out any incompatabilites of differnt versions we recommend to use any version of PYTHON 3.6 or newer for the implementation projects.

Regrade Requets: Use Gradescope for regrade requests.

Academic Dishonesty: We will be checking your code against other submissions in the class for logical redundancy. If you copy someone else's code and submit it with minor changes, we will know. These cheat detectors are quite hard to fool, so please don't try. We trust you all to submit your own work only; *please* don't let us down. If you do, we will pursue the strongest consequences available to us.

Getting Help: You are not alone! If you find yourself stuck on something, contact the course TAs for help. Office hours and Piazza are there for your support; please use them. If you can't make our office hours, let us know and we will schedule more. We want these projects to be rewarding and instructional, not frustrating and demoralizing. But, we don't know when or how to help unless you ask.

Dataset

In this project, you will implement a neural network and test is on the Boston data set.

This data set contains housing prices as targets and community statistics as features. You can load it with the following line of code:

```
bostonData = sio.loadmat('./boston.mat')
```

Implementing a Neural Network

You will now implement the function deepnet.py. As a very first step, you should open the file and take a look at it. We broke it apart into three functions and a pre-processing step.

1. First implement the preprocess function

   ```
   preprocess(xTr, xTe)
   ```

   It takes as input the training and the test data. This should make the training data set zero-mean and each feature should have standard deviation 1. Make sure to only use the training dataset to learn the transformation and then apply exactly the same transformations to the test data set.

   1. HINT: Each input vector should be transformed by $\vec{x}_i \rightarrow \Sigma(\vec{x}_i - m)$, where Σ is a diagonal $d \times d$ matrix with entries $\frac{1}{\sigma_j}$.
   2. HINT 2: check the lecture materials on dimensionality reduction to get this step done. Be careful with numerical issues with the involved operations.

2. Now implement the forward pass function

   ```
   forward_pass(W,xTr,trans_func)
   ```

   It takes the weights for the network, the training data, and the transition function to be used between layers. It should output the result at each node for the forward pass. W[0] stores the weights for the last layer of the network.

3. Now compute the loss for the network

   ```
   compute_loss(zs, yTr)
   ```

   It takes the output of the forward pass and the training labels. It should compute the loss for the entire training set averaging over all the points:

   $$L(x,y) = \frac{\frac{1}{2}(H(x) - y)^2}{n}$$

4. Now compute the gradient for the weights

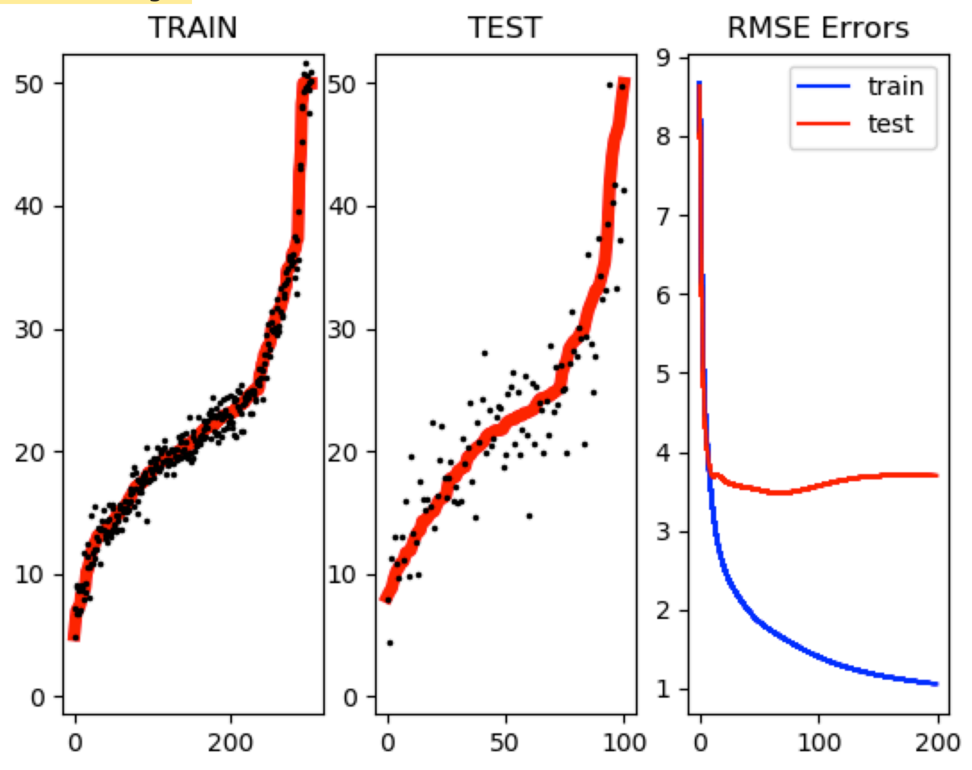
   ```
   backprop(W, as, zs, yTr,  der_trans_func)
   ```

   It takes the weights for the network, the outputs of the forward pass, the training labels, and the derivative of the transition function. Use the chain rule to calculate the gradient of the weights.

5. If you did everything correctly, you should now be able to visualize your RMSE error on the boston data:

   ```
   >> python bostondemo.py
   ```

   The result should look similar to this image:



Each dot shows the training / testing error of a house price prediction example. The houses are sorted by increasing price. The very right plot shows the training and testing error.

Evaluation

Tests.

80% of the grade for your project4 submission will be assigned based on the correctness of your implementation.

Performance. 10% of your grade will come from the speed of your implementation, and 10% from the accuracy.

Timing Score: This score is based on how fast your neural network trains on the data. A faster implementation earns more points.

Accuracy Score: This score is based on how accurate your neural network predicts a secret test set after being trained on the same training data you have access to. A lower sum squared error earns more points.

Look at the file `bostontest.py`. This file shows how the autograder will evaluate your implementation for performance (this is just a copy though, feel free to edit it). In its current state the file is not runnable since you don't have access to `boston_secret.mat`, but with easy modifications you can use this file to choose the parameters you would like to submit to the autograder. Remember that the times and accuracies you get locally will not guarantee the same score on the autograder. Both times and accuracies will be different, but you should be able to get an idea of what works well.

`main.py` contains code to save your parameter choices to `best_parameters.pickle` . Main will not be used in the autograder, it is to show you how to save parameters, so you can use it test your code anyway you want. You should edit these parameters to improve your performance score. If you don't, the default values described below will be used. In this case you should recieve a 5/10 on efficiency and 5/10 on accuracy giving you 90% overall assuming you implement the rest of your neural network correctly.

Parameters You Can Edit:

`TRANSNAME`	The transition function. Either 'sigmoid', 'ReLU2', 'tanh', or 'ReLU'. Default: 'sigmoid'.
`ROUNDS`	The number of times the entire network will be optimized. Also known as epochs. Default: 200.
`ITER`	The maximum number of iterations in each gradient descent step. Default: 50.
`STEPSIZE`	The stepsize parameter that gets passed to your gradient descent function. This will either be the constant stepsize or the initial stepsize. Default: .1.
`wst`	The network architecture. Each element in this array is the number of nodes in that layer from back to front. You can also change the number of layers. The network must start with 1 and end with 13. Default: [1 20 20 20 13].

Note that it will be very difficult to get 100% on this project. The point is to figure out ways to improve the accuracy of your neural network without making it take longer and vice versa. Try running your test data on `bostontest.py` as a validation set to choose your parameters and make sure to take note of how your changes affect both runtime and accuracy.

Credits: Project adapted from Kilian Weinberger (Thanks for sharing!).