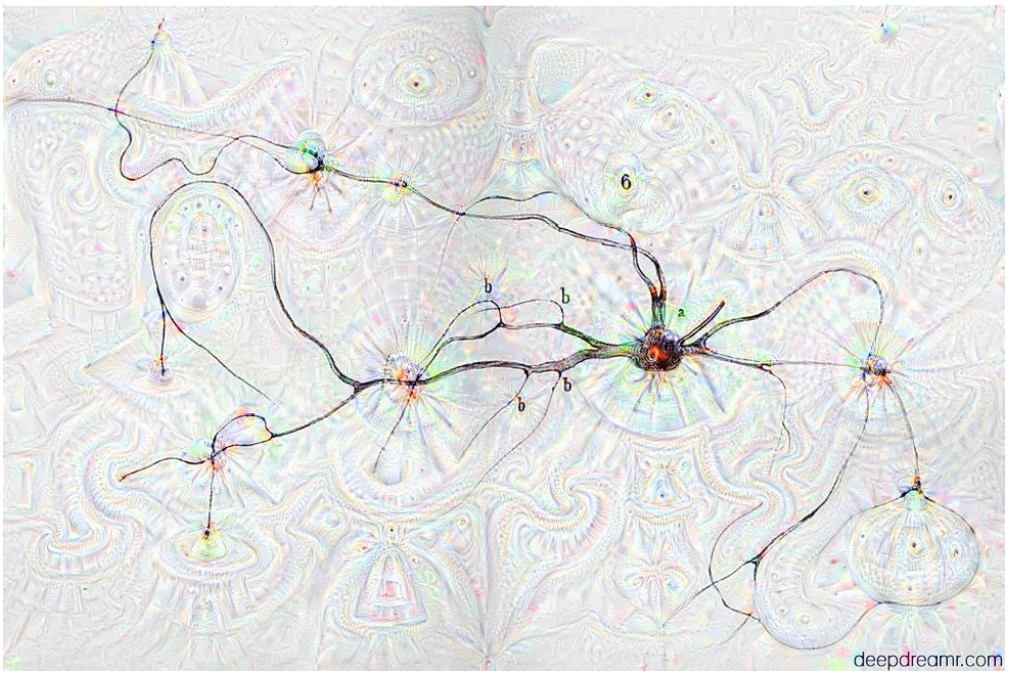
Neural Networks



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

Files you'll edit:

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

Files you might want to look at:

`initweights.m`	This function initializes the weights of the network given the structure of the network.
`deepnet.m`	Computes the loss and gradient for a particular feed forward neural net. Calls `forward_pass.m`, `compute_loss.m`, and `backprop.m`.
`bostondemo.m`	This script visualizes the RMSE error on the boston data set.
`xordemo.m`	This function shows the neural network applied to XOR data.

Introduction

In this project, you will implement a neural network.

Boston data set

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

```
>> load boston
```

Implementing a Neural Network

You will now implement the function `deepnet.m`. 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 $\vec{x}_i \rightarrow U\vec{x}_i - m$, where u can be a diagonal $d \times d$ matrix.
   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. $W\{1\}$ 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)
   ```

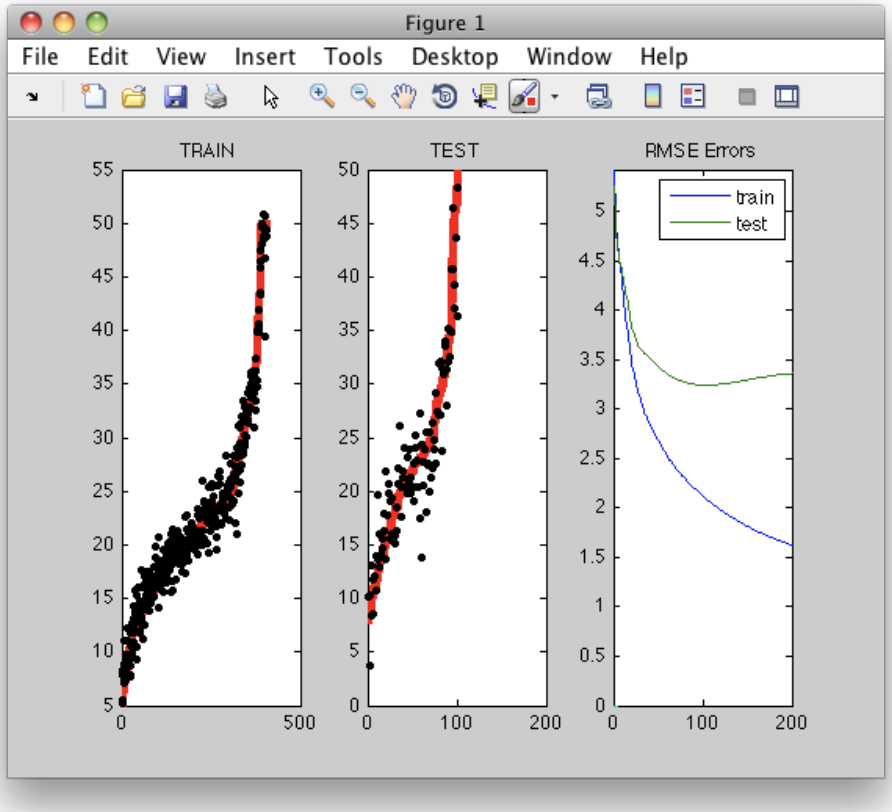
   It takes the weights for the network, the outputs of the forward pass, the training labels, and the derivative of the transition function. Use the chain rule to calculate the gradient of

the weights.

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

```
>> bostondemo
```

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.

Play around with different wyas to perform data preprocessing and experiment with different network architectures. You can edit the file `bostondemo.m` to change your network layout. Make sure you submit the *best* version in `bostondemo.m` and run it before doing your final commit as we will have a look at the training and test errors for your settings after the final deadline.

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