How The Brain Responds to Images

Group 3
Members: Yew Hung Leong, Zachary Hargreaves, Ali Kayes

Background

Patients were given natural images to look at. While these patients were examining the images, an fMRI machine scanned the visual cortex of the brains of the test subjects. FMRI machines measure blood oxygen levels of organic tissue and measuring higher oxygen levels in specific regions of a brain would imply that there is more activity in that region.

Professor Jack Gallant of the University of California, Berkeley Department of Psychology supplied us with the data. The set consists of 1.25GB of volumetric fMRI brain data and the black and white flat images that they correspond to. Specifically, our training set consists of 1750 brain image responses to each of 25000 voxels while our validation set consists of 120 brain image responses corresponding to each of the voxels. Moreover, we were also supplied with information containing which region of interest correspond to which of the voxels. Our regions of interest were categorically mostly the major regions of the visual cortex: v1, v2, v3, v3a, v3b, v4, lateral occipital area, and a lump of other areas. Furthermore, we also had stimuli data where we were given 1750 training images and 120 validation images. Respectively, these images were stored in training and matrices where each column specified which image was presented of the time of the run. Lastly the order of stimuli data match the order of stimuli variables in the responses dataset.

Related works using similar data include “Reconstructing Visual Experiences from Brain Activity Evoked by Natural Movies,” by Nishimoto, Naselaris, Benjamini, Yu, and Gallant. It was developed in the same laboratory (Gallant Laboratory) that we were supplied our data from. Their task included creating visual data given brain fMRI data. To see some of the results of their work, refer to http://youtu.be/nsjDnYxJ0bo.

Problem Statement

The problem we were to solve was to answer the question of how the brain responds to two-dimensional natural images. To answer this question, we attempted numerous approaches. First, we attempted to use the example methods described the in the data documentation to accurately synthesize and properly interact with the dataset. However, because we did not suitably understand the mechanisms behind the methods that the documentation from the data described, we found ourselves at a dead end. While we were able to load in the estimated responses into python, the same method failed at allowing us to interact with the stimuli set, training stimuli, and the full resolution stimuli validation sets.

Eventually, we tried other tools that would help us to thoroughly interact with our dataset. By being able to properly interact with our data, we would find different quality metrics for answering different types of questions. Some of these quality metrics we found, were inherent to some of the tools that we would use to process our data.

For instance, one quality metric we tried was to compute the multinomial logistic loss for a type of multi-class classification system where predictions would pass through a normalized exponential so that we could get a probability distribution:

$$P(y = j|\mathbf{x}) = \frac{e^{\mathbf{x}^\top \mathbf{w}_j}}{\sum_{k=1}^{K} e^{\mathbf{x}^\top \mathbf{w}_k}}$$

Another quality metric we considered was to compute the cross-entropy logistic loss:

$$E = \frac{-1}{n} \sum_{n=1}^{N} [p_n \log \hat{p}_n + (1 - p_n) \log(1 - \hat{p}_n)]$$

However, we later realized that our targets were not probabilities but means. So we decided not to use this function.

Lastly we considered using an Euclidean Loss function:

$$E = \frac{1}{2N} \sum_{n=1}^{N} ||\hat{y}_n - y_n||_2^2$$

This allows us to compute the distance between data and measure the loss associated with the predictions.

In the end, because we understood the Euclidean Loss function the best and how it related with our dataset, we used Euclidean Loss as our primary success metric.

The major implications behind the success of processing our data are being able to answer questions pertaining to how the visual cortex associates with different categorical

objects such as faces, colors, and sizes. Our main focus, however, was to solve the problem of being able to accurately predict how a brain would respond to visual stimuli. More specific implications about how the visual cortex responds to images can be explored using our predictor.

Tools

The data provided was formatted into .mat files and in turn Matlab was used for simple image viewing and small debugging on the raw images. The data was already separated into training and validation sets represented as matlab matrices. Due to our unfamiliarity with Matlab's internal libraries, we opted to export the matlab files into raw HDF5 representation so that python could be used for the data wrangling. We needed to merge some files together because some of the data was provided as two separate matrices instead of one (due of Matlab limitations). We experimented with how many labels to apply to the data set and ended up comparing results for the maximum amount of labels versus applying labels only to regions of interest (ROI). This label manipulation was done using python. After the data was formatted appropriately it was imported into caffe, which ran our neural nets over the data set on the Amazon EC2 server.

We initially tried running neural nets on the data set utilizing only Matlab and its respective toolboxes. The way neural nets were ran in Matlab was non-trivial and a lot of time a energy was spent trying to understand the nuances of required parameters to run a test. The structure of the neural nets also was represented a little different than what was taught in class (caffe which used easily mutable DAGs). Matlab provided tools for for curve fitting, pattern recognition, clustering, and dynamic time series. These tools were very robust but understanding the interaction between the hidden layers and the output layers proved to abstract away too much for us, and no true understanding of the process was being learned. In addition to these issues, only one member of our group understood the Matlab work environment and had access to the neural network toolboxes necessary to run the experiments. With all this in mind, we chose to migrate over to using caffe.

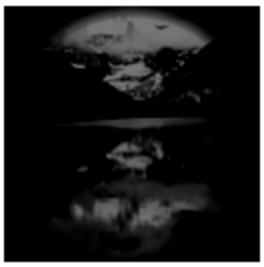
Caffe had its own respective issues but we were able to overcome them due to support from the course staff. There was severe lack of documentation with regards to how the different types of layers worked, and users were assumed to have extensive domain knowledge about how the layers were computed. Because Caffe is still in developmental stages, their "accuracy layer" (common in neural networks) does not

support multiple labels and in turn was the source of major delays and confusion. We decided to use a loss layer instead of an accuracy layer, but we were not confident about how to correctly assess our outcomes. However, the labs in Caffe helped as a primer for constructing DAGs and proved valuable for debugging and parameter tuning.

Methods with Tools

Preprocessing

The raw data provided were a set of training and validation data. The inputs were a set of grayscale images as shown below. Matlab was used to extract some of these images for data visualization.



These were some of the images in the data set. Along with the stimuli images are a set of estimated brain responses that correspond to the specific image shown. In addition to these two data sets are two other matrices that map the estimated brain response to a region of interest or a voxel in a 64 x 64 x 18 brain space. A short summary of the data format is given below:

Stimuli (train) <S1/S2> – 2 sets (different subjects) of 3D Matrix containing 1750 training 128x128 images. [1750 x 128 x 128]
Stimuli (test) <S1/S2> – 2 sets (different subjects) 3D Matrix containing 120 testing 128x128 images. [120 x 128 x 128]

Estimated Response (train) <S1/S2> – 2 sets (different subjects) of 2D Matrix containing 1750 readings of ~25,000 voxels readings corresponding to the training image shown. Each column is a reading. [1750 x ~25,000]

Estimated Response (test) <S1/S2> – 2 sets (different subjects) of 2D Matrix containing 120 readings of ~25,000 voxels readings corresponding to the validation image shown. Each column is a reading. [120 x ~25,000]

ROI <S1/S2> – A column vector that maps each estimated response (S1/S2) dataset's voxels into a region of interest labeled between 0 to 7. [~25,000 x 1]
VoxID <S1/S2> – A column vector that maps each estimated response (S1/S2) dataset's voxels into a 3D brain space. Positions are labeled between 0 to 73728 (64 x 64 x 18). [~25,000 x 1]

Matlab was first used to convert the raw files into a Python h5py readable format . Python and Numpy were used to do most of the data preprocessing. The code is available under the /src/ directory submitted.

Compressing Data Into ROI

One method that we tested was to compress the estimated responses into their respective ROIs. Each estimated response column was reduced into their respective ROI index specified by the ROI vector. After they were all binned into their respective ROI labels, they were averaged out. This is done for all columns of estimated responses and saved as a HDF5 files, as required by Caffe. Overall, the data is left with 8 labels per reading, a 8 x 1750 or a 8 x120 matrix respectively for training and validation sets.

Mapping to Voxel Space

Another method tested was to map the data into voxel space, 64 x 64 x 18, using the VoxID vector to map each reading respectively to their positions. This had to be done as the both estimated responses matrix, . They are again saved as a HDF5 files. The final data is left with 73728 labels per reading, a 73728 x 1750 or a 73728 x 120 matrix respectively for training and validation sets.

Caffe Data Blobs

The 3D image matrices had to be converted to a Caffe compatible format, which is a 4D matrix called a blob. Expansion of the ndarrays were done and the new format 1750 x 1 x 128 x 128 and 120 x 1 x 128 x 128 matrices were created and saved as HDF5 files.

Caffe Processing

Caffe Deep Learning networks are nets written in protobuf syntax. The nets that the data were trained and tested on were one provided by the examples. They were then

modified to fit the data that we are provided with. Below were the nets used and can be found in the /nets/<roi or voxid>/ directory submitted:

Basic Logistic Regression: This method uses three total layers and one computational layer. The computational layer computer and inner product.
Basic Two Layer Neural Net: An expansion from basic logistic regression, this neural net computes an inner product and then passes that information to a convolution layer.
LeNet Convolution Net: This neural net is known for its accuracy in classifying digits. The Sigmoid activation layer had been replaced with a Rectified Linear Unit layer so that neurons could be handled. Moreover, its computational layers consist of a convolution layer then a pooling layer then another convolution layer and pooling layer. Then the inner product is computed and passed to the Rectifies Linear Unit which then passes the information to another inner product.
CIFAR-10: This is another convolutional neural net. Like LeNet, it uses a convolution layer followed by a pooling layer. But unlike LeNet, it then uses a Rectified Linear Unit and a normalizing layer for the pooling layer. The normalized data is passed to another convolution layer which then is pooled and and passes through a rectified linear unit. This same process is repeated once more and in the end, an inner product is computed.
ImageNet: This is a heavily layered model. The method was proposed by Krizhevsky, Sutskever, and Hinton in their NIPS 2012 paper. It has 24 layers and is a very deep net.

These nets were trained and tested on the ROI and VoxID estimated responses. They training set and the test set were divided into training and testing phase, respectively, as specified by Caffe. For all nets, the testing phase layer were switched to Euclidean loss instead, which the reason is explained below. The data layer contains a directory parameter that points to a file which then points to the datasets. The files are found in the /data/<roi or voxid>/ directory submitted. The dataset are not included due to their file sizes, but they can be generated by pulling the raw data and running the Python preprocessing scripts.

There is also Caffe Solver file which holds parameters such as learning rate, iterations, and more. This can also be found in the same directory.

Measurement

Caffe generates predicted labels when an image from the test set is provided. The predicted labels can then be tested with the ground truth provided by the validation estimated response data set. Caffe does this automatically by defining the testing phase.

One of the issues we ran into was measuring accuracy. Because the labels that are predicted are multi-labels, the accuracy layer does not support such functionality for calculating on multi-labels. We decided to substitute this layer with Euclidean loss as it seemed to represent multi-label accuracy the most. As a result, methods such as cross-entropy logistic loss and multinomial logistic loss were replaced in favor of Euclidean loss.

Another issue we ran into was processing the voxel space mapped data. This is because the Euclidean loss values after processing this data resulted as a NaN. The reason for is most likely due to the fact that when we took the mean in compressing the data into eight regions of interest, we were only concerned with properly formed data. However the voxel space mapped data most likely contains information that is unusable such as extraneous noise from the environment. So, attempting to run our methods on the uncompressed data resulted in NaN Euclidean loss values.

Amazon EC2 Instance

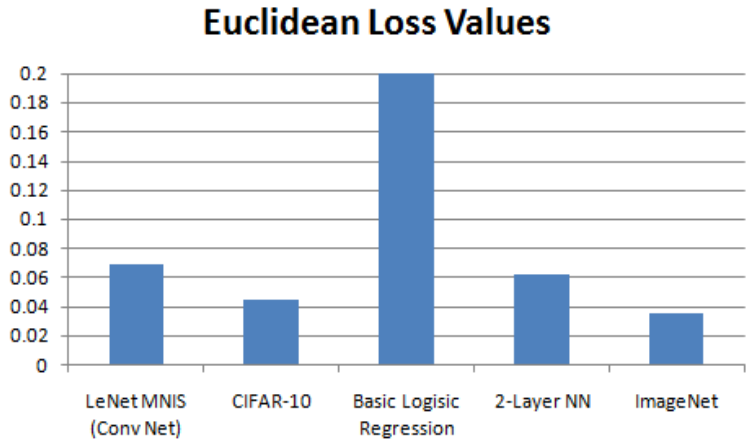
It is worth noting that the Amazon EC2 instance was a major tool in allowing us to do the data processing of large data sets, but also to run Caffe calculations using a GPU.

Results

ROI Mapped Data

Below are the results of the Euclidean loss for each of the tested nets:

Net Type:	Iterations	Learning Rate	Batch Size	Euclidean Loss Value
LeNet MNIST Convolution Net	10000	0.001	10	0.0693844
CIFAR-10	60000	0.001	10	0.0443331
Basic Logistic Regression	10000	0.001	10	4.9575
2-Layer NN	10000	0.001	10	0.0624781
ImageNet	10000	0.001	10	0.0351738



Basic Logistic Regression continued off of the chart

Logistic regression showed the largest Euclidean loss whereas the ImageNet showed the lowest loss. A general trend is such that deeper and complex nets such as the ImageNet tend to show better results. A simpler net such as the logistic regression 1-layered net showed higher loss.

Voxel Space Mapped Data

We ran into issues processing this dataset on Caffe. One major issue is that the Euclidean loss values were always NaN. It is presumed that this was because that there were too many labels, most of them being zeroes. No valuable data was retrieved here.

Lessons

Caffe is still in developmental stages and the “accuracy layer” (common in neural networks) does not support multiple labels and in turn was the source of major delays when starting out ETL and data processing.

There was severe lack of documentation with regards to how the different types of layers worked respectively. Caffe assumed prior experience with how neural layers work. Much focus was given to our loss layer/s because we were not confident in how to correctly assess our outcomes. In retrospect, many of the loss layers performed similar to one another and a lot of unnecessary energy was put into code that was already working.

Contributions

Yew Hung Leong

- Project Proposal
- Presentation
- Poster, Tools section and About Caffe section.
- Data preprocessing, Matlab data extraction, and Python/Numpy data reformatting and HDF5 conversion
- Responsible for setting up, running, and collecting data of Caffe Nets (logistic regression, 2-layered net, LeNet, CIFAR-10)
- Method with Tools and Results sections of the report

Ali Kayes

- Project proposal
- Presentation
- Poster: Background on how the brain responds to images section, Methods section, Tools section
- Panel Presentation at BIDS
- PyLab and Pytable attempts at tinkering with data
- Responsible for fine tuning ImageNet Krizhevsky et al method to work with our dataset
- Background and Problem Statement sections and general editing of final report

Zachary Hargreaves

- Importing files into Matlab
- Matlab image familiarity (displaying images of the data)
- Presentation
- Loss function research
- Matlab libraries for neural nets (not used)
- ETL from Matlab into Caffe
- Final Report: Tools, Results
- Poster: Data Preparation, Results, Lessons Learned

Final Contributions

- Hung - 40
- Ali - 30
- Zach - 30