MAHARISHI UNIVERSITY MANAGEMENT

1000 N 4th ST, FAIRFIELD IOWA, IA 52557

PROJECT REPORT

Brain imaging data (fMRI)
Pattern recognition

Course:	CS582 - Machine Learning
Supervisor:	Dr . Emdad Khan
Student:	Ganijon Rahimov

24th March 2017

Abstract

Gaussian Naïve Bayes (GNB) classifiers and Support Vector Machines (SVM) have been used with fMRI brain imaging data to predict when the subject was reading a sentence versus perceiving a picture at a given time. Both of these classify 8-second windows of data into these two classes, achieving around 85% classification accuracy [Mitchell et al, 2004]. The main goal of this project is to explore the capacity of Artificial Neural Networks for classification of cognitive patterns, in particular, prediction when the subject is reading a sentence versus perceiving a picture during single window of trial. In addition, we identify which features to include for classifier input and a number of issues about efficient computation with the large data set.

Brain imaging data (fMRI) classifier

Background

Functional magnetic resonance imaging (fMRI) measures the metabolic changes that occur within the brain to obtaining 3D images related to neural activity in the brain through time. fMRI is becoming the diagnostic method of choice for learning how a normal, diseased or injured brain is working, as well as for assessing the potential risks of surgery or other invasive treatments of the brain.

Physicians and Psychologists perform fMRI to:

- examine the anatomy of the brain.
- determine precisely which part of the brain is handling critical functions such as thought, speech, movement and sensation, which is called brain mapping
- monitor the growth and function of brain tumors.
- guide the planning of surgery, radiation therapy, or other surgical treatments for the brain.

Blood oxygenation level dependent (BOLD) imaging is the standard technique used to generate images in functional MRI(fMRI) studies, and relies on regional differences in cerebral blood flow to delineate regional activity.

Blood flow in the brain is highly locally controlled in response to oxygen and carbon dioxide tension of cortical tissue. When a specific region of the cortex increases its activity in response to a task, the extraction fraction of oxygen from the local capillaries leads to an initial drop in oxygenated hemoglobin and an increase in local carbon dioxide (CO_2) and deoxygenated hemoglobin. Following a lag of 2-6 seconds, cerebral blood flow increases, delivering a surplus

of oxygenated hemoglobin, washing away deoxyhemoglobin. It is this large rebound in local tissue oxygenation which is imaged.
The reason fMRI can detect this change is due to a fundamental difference in the paramagnetic properties of oxygenated hemoglobin and deoxyhemoglobin.
Deoxygenated hemoglobin is paramagnetic whereas oxygenated hemoglobin is not, and therefore the former will cause local dephasing of protons, and thus reduce the returned signal from the tissues in the immediate vicinity.

fMRI Process
The typical research MRI scanner has a strength of three teslas, a force about 50,000 times stronger than the Earth's magnetic field.The data collected during an fMRI experiment consists of a sequence of individual magnetic resonance images, acquired in a manner that allows one to study oxygenation patterns in the brain
The sensitivity of fMRI in detecting neural activation is dependent on the relative level of signal and noise in the time-series data.

- the whole brain is represented/divided into a numbers of unit volume called Voxels.
- This will help in production of high dimensional data
- An fMRI scanner measure the value of fMRI signals at all the points in a three-dimensional grid every few seconds (4-6)

Problem Statement

Critics of the technique complain that fMRI overlooks the networked or distributed nature of the brain's workings, emphasizing localised activity when it is the communication among regions that is most critical to mental function.

Objective and Methods

The main goal of this project is to explore the capacity of Artificial Neural Networks for classification of cognitive patterns, in particular, prediction when the subject is reading a sentence versus perceiving a picture during single window of trial. In addition, we identify which features to include for classifier input and a number of issues about efficient computation with the large data set.

- Multilayer feed-forward perceptron network classifier is used for classification of cognitive activities.
- MATLAB is used for implementation.

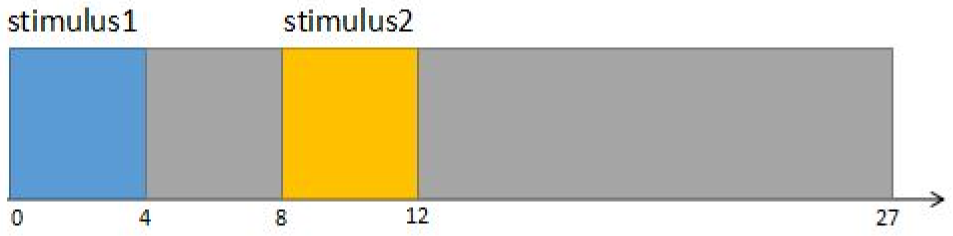
fMRI Data set

The data set includes different subjects. For each subject there are different sets of FMRI images that were taken when subject was reading a sentence, and other sets of FMRI images were taken when the subject was watching the picture. The final goal is to design a classifier to correctly find out which sets of images were taken when the subject was reading a sentence, and which were taken when the subject was watching the a picture.
Each set of FMRI pictures includes 16 3D images, which are taken every 500sec in time. Each image contains up to 56 x 56 x 8 voxels. Each images has regions of interests defined on it, which correspond to specific biologic areas of the brain.
The classifier was trained using cross-validation, that is, it is trained on 54 sets of images and tested on tested.

The Experiment

The experiment consists of a set of parts, and the data is separated into parts. For some of these parts, the subject was simply rested, or looking at a fixation point on the screen. For other trials, the subject was given a picture and a sentence, and to press a button to decide whether the sentence correctly described the picture. For these parts, the sentence and picture were shown in sequence, with the picture presented first on half of the parts, and the sentence presented first on the other half of the parts. Forty such experiment are available for each subject. The timing within each such experiment is as follows:

- The first stimulus (sentence or picture) was presented at the beginning of the trail (image=1).
- Four seconds later (image=9) the stimulus was removed, replaced by a blank screen.
- Four seconds later (image=17) the second stimulus was presented. This remained on the screen for four seconds, or until the subject pressed the mouse button, whichever came first.
- A rest period of 15 seconds (30 images) was added after the second stimulus was removed from the screen. Thus, each trial lasted a total of approximately 27 seconds (approximately 54 images).



Images were collected every 500msec, each trial lasted a total of approximately 27 seconds (approximately 54 images).

Data Structure

meta

This variable gives the information about the fMRI data set. Relevant fields are shown in the follow:

```
meta =
        study: 'data-starplus'
        subject: '05710'
        ntrials: 54
        nsnapshots: 2800
        nvoxels: 4634
        dimx: 64
        dimy: 64
        dimz: 8
```

info

This variable defines the experiment in terms of a sequence of 'part of experiment'. 'info' is a 1x54 struct array, it is describing the 54 time parts of the all experiment. Most of these time intervals correspond to trials during which the subject views a single picture and a single sentence, and presses a button to indicate if the sentence correctly describes the picture. The relevant fields of info are illustrated in the following example:

```
info(18)
 mint: 894
 maxt: 948
 cond: 2
 firstStimulus: 'P'
```

data

This variable includes the raw observed data. The fMRI data is a sequence of pictures collected over time, one picture each 500 msec. The data structure 'data' is a [54x1] cell array, with one cell per 'trial' in the experiment.
Each element in this cell array is an NxV array of observed fMRI activations. The element data{x}(t,v) gives the fMRI observation at voxel v, at time t within trial x. Here t is the within-trial time, ranging from 1 to info(x).len. The full image at time t within trial x is given by data{x}(t,:).

Feature Selection

This dataset contains all the data. It may contain noisy data that is not helpful to be trail data so we need to have a function that can filter out the non-noisy and the function below selects the non-noisy trails from the dataset uploaded. the data in [i,d,m] is a trail data and can be used for training data.

`transformIDM_selectTrials(info,data,meta,trials)` returns a copy of info, data, meta containing only the specified trials. The input parameter 'trials' is an array listing the indices of trials to be included.

To select only non-noisy trials:

```
        [i,d,m]=transformIDM_selectTrials(info,data,meta,find([info.cond]~=0));
```

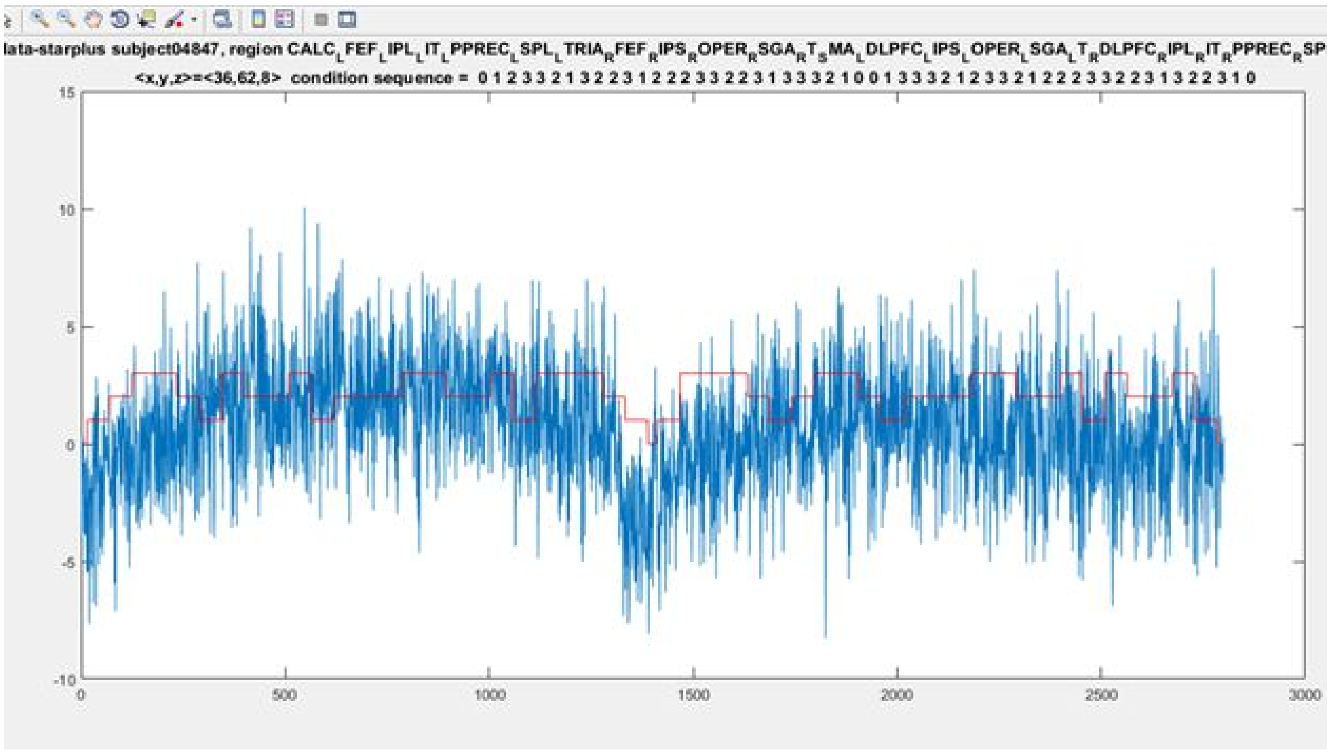
To select specified trails as as trails 3 and 5 we write the following function:

```
        [info2,data2,meta2] = transformIDM_selectTrials(info,data,meta,[3,5])
```

We can also plot the time course for just specific trials of specific voxels example: for trials 3 and 4 of voxel <36,62,8>

```
        [i,d,m]=transformIDM_selectTrials(info,data,meta,[3,4]);
        plotVoxel(i,d,m,36,62,8);
```

This can plot a for the dataset depending on the voxels specified above.



Preprocessing

To read picture and sentence separately: slice time correction is applied. Each voxel is acquired at different time point in one Time of repetition (picture in first 16 seconds and sentence in next 16 seconds or vice versa). Convert IDM structure to examples. Combine examples, labels structure, which can be used in training and applying a classifier.

Separate P1st and S1st trials as follows:

```
[infoP1,dataP1,metaP1]=transformIDM_selectTrials(info1,data1,meta1,find(
[info1.firstStimulus] == 'P'));
[infoS1,dataS1,metaS1]=transformIDM_selectTrials(info1,data1,meta1,find(
[info1.firstStimulus]=='S'));
```

After separating the pictures and sentences. We need to read them separately as follows. The first picture is read in the first 16 seconds and the first sentence is read and then the vice versa. Separate reading P vs S as follows:

```
[infoP2,dataP2,metaP2]=transformIDM_selectTimewindow(infoP1,dataP1,metaP1,[1:16]);
[infoP3,dataP3,metaP3]=transformIDM_selectTimewindow(infoS1,dataS1,metaS1,[17:32]);
[infoS2,dataS2,metaS2]=transformIDM_selectTimewindow(infoP1,dataP1,metaP1,[17:32]);
[infoS3,dataS3,metaS3]=transformIDM_selectTimewindow(infoS1,dataS1,metaS1,[1:16]);
```

After separately reading them convert to examples as follows:

```
[examplesP2,labelsP2,exInfoP2]=idmToExamples_condLabel(infoP2,dataP2,metaP2);
[examplesP3,labelsP3,exInfoP3]=idmToExamples_condLabel(infoP3,dataP3,metaP3);
[examplesS2,labelsS2,exInfoS2]=idmToExamples_condLabel(infoS2,dataS2,metaS2);
[examplesS3,labelsS3,exInfoS3]=idmToExamples_condLabel(infoS3,dataS3,metaS3);
```

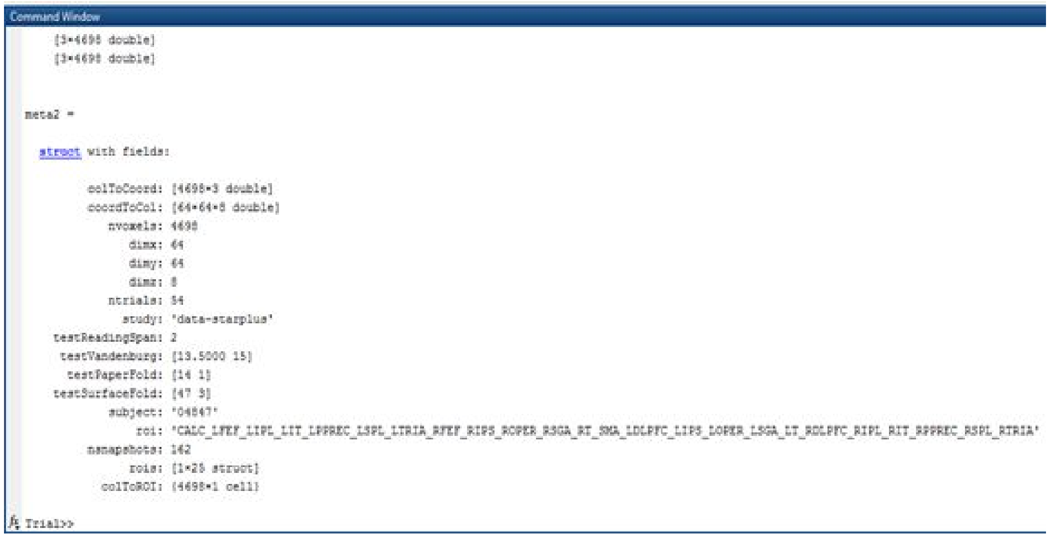
After that combine examples and create labels. Label 'picture' 1, label 'sentence' 2.

```
examplesP=[examplesP2;examplesP3];
examplesS=[examplesS2;examplesS3];
labelsP=ones(size(examplesP,1),1);
labelsS=ones(size(examplesS,1),1)+1;
examples=[examplesP;examplesS];
labels=[labelsP;labelsS];
```

`transformIDM_selectTimewindow(info,data,meta,snapshots)` returns a copy of info,data,meta containing only the specified snapshots in time within each trial. The input parameter 'snapshots' is an array listing the indices of snapshots to be included, assuming the index of the first snapshot of each trial is 1.

To select just time snapshots 3, 4, and 7 from each trial

```
[info2,data2,meta2] = transformIDM_selectTimewindow(info,data,meta,[3,4,7])
```


Command Window

```
    [3*4698 double]
    [3*4698 double]


meta2 =

  struct with fields:

        colToCoord: [4698*3 double]
        coordToCol: [64*64*8 double]
           nvoxels: 4698
              dimx: 64
              dimy: 64
              dimz: 8
           ntrials: 54
             study: 'data-starplus'
    testReadingSpan: 2
     testVandenburg: [13.5000 15]
      testPaperFold: [14 1]
    testSurfaceFold: [47 3]
           subject: '04847'
               roi: 'CALC_LFEF_LIPL_LIT_LPPREC_LSPL_LTRIA_RFEF_RIPS_ROPER_RSGA_RT_SMA_LDLPFC_LIPS_LOPER_LSGA_LT_RDLPFC_RIPL_RIT_RPPREC_RSPL_RTRIA'
        nsnapshots: 162
              rois: [1*25 struct]
          colToROI: {4698*1 cell}

fx Trial>>
```

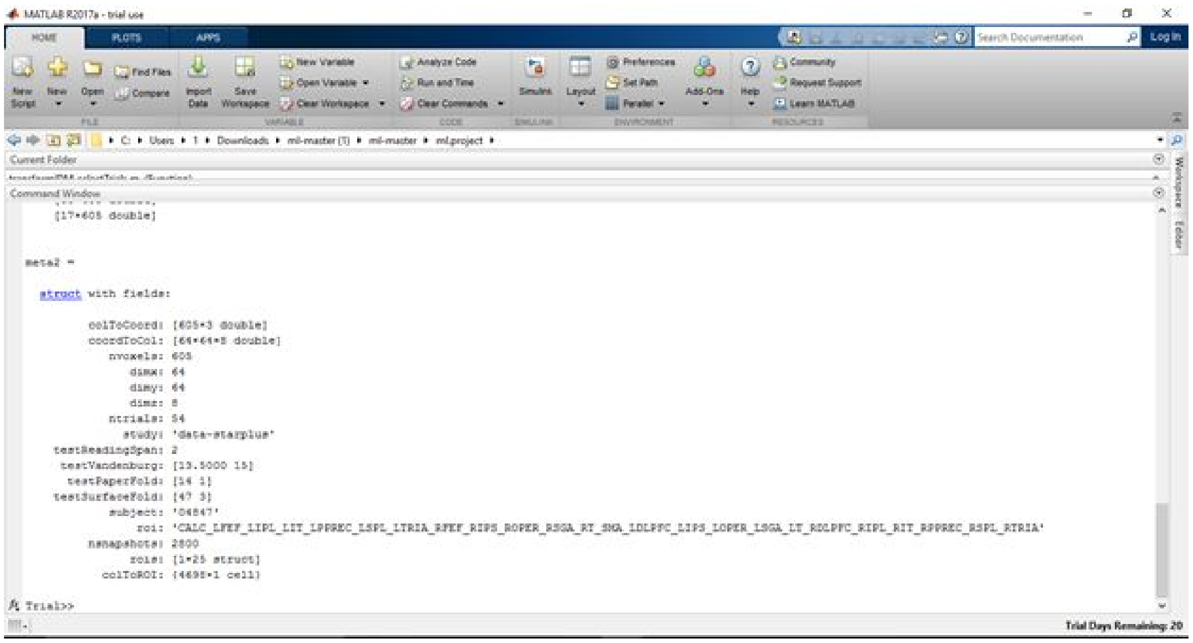
Regions of Interest

Several ROIs are selected since they are enough to classify picture and sentence. Information, data and meta where the data contains just the voxels belonging to the seven regions are returned. Info, Data, Meta-Data containing only the specified snapshots in time within each trial is collected. A copy of info, data, and meta, selecting voxels that belong to ROIs found on ROI list.

```
        transformIDM_selectROIVoxels(info,data,meta,ROIlist)
```

Returns a copy of information, data, and meta, selecting only voxels that belong to ROIs found on ROIlist.

`[info2,data2,meta2] = transformIDM_selectROIVoxels(info,data,meta,{'CALC' 'LIT'})` returns an IDM where the data contains just the voxels belonging to   CALC and LIT.

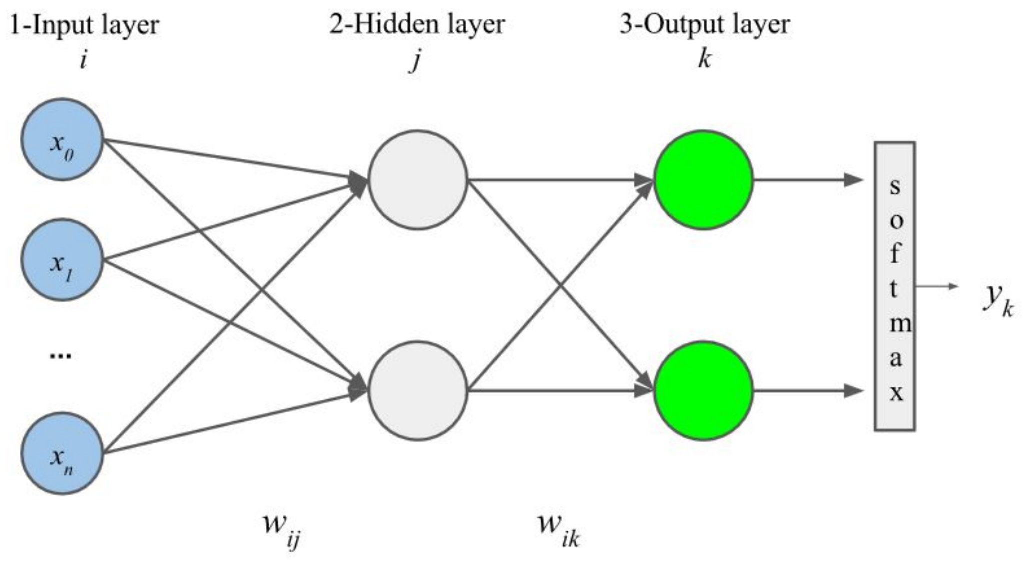


```
    [17×605 double]


meta2 =

  struct with fields:

        colToCoord: [605×3 double]
        coordToCol: [64×64×8 double]
           nvoxels: 605
              dimx: 64
              dimy: 64
              dimz: 8
           ntrials: 54
             study: 'data-starplus'
    testReadingSpan: 2
    testVandenburg: [13.5000 15]
     testPaperFold: [14 1]
   testSurfaceFold: [47 3]
           subject: '04847'
               roi: 'CALC_LFEF_LIPL_LIT_LPPREC_LSPL_LTRIA_RFEF_RIPS_ROPER_RSGA_RT_SMA_LDLPFC_LIPS_LOPER_LSGA_LT_RDLPFC_RIPL_RIT_RPPREC_RSPL_RTRIA'
        nsnapshots: 2800
              rois: [1×25 struct]
          colToROI: {4698×1 cell}

Trial>>
```

Feed-forward Neural Network

The objective of pattern classification is to use a set of examples and infer an underlying regularity which can subsequently be used to predict new instances. In the case of feed-forward networks, the set of examples (training set), comprises set of inputs with corresponding set of desired outputs. The training set is used to define an error function in terms of the discrepancy between the predictions of the network, for given inputs, and the desired outputs given by the training set. The learning process then involves adjusting the values of the parameters to minimize the value of the error function. Once the network has been trained, i.e. once suitable weights have been determined, new inputs can be applied and the corresponding predictions calculated.

For this project we have designed 3-layer feed-forward perceptron network with 2 nodes in hidden layer and 2 nodes on the output layer. Logically, we selected only 2 nodes for output layer, because we are trying to classify the inputs into 2 classes, picture and sentences. And, we picked 2 nodes for the hidden layer for the simplicity reasons with linear activation function. As for the activation function for output nodes we identified *softmax activation function* as a most suitable option for the typical classification problems.



Conceptual Design of 3-layer feed-forward perceptron network

Preprocessing

Normalization of examples is worth considering before feeding examples to classifier if there is a chance that some voxels will have much wider variation in signal amplitude than others. It is different from previous preprocessing of neuroimaging data, e.g. motion correction, time-slice correction or noise reduction. The transformation is done on the examples, considered as a matrix where each row is an example and each column is a feature. We normalized each column to have mean 0 and standard deviation 1 within examples coming from the same trial.

Another preprocessing step that is required for the neural network classifier is conversion of labels into *1-of-N encoding* from their current encoding as classes 1 and 2. This transformation is done fairly straightforward by creating a new matrix of zeros with size N x M, where N is the number of data points in data set and M is the number of classes, then setting only one of the entries to 1. The basic reason for doing this preprocessing step is we are employing *softmax* activation function in output and hidden layers of our network architecture, which is common option for classification problems.

Training

We made number of futile attempts while selecting the architecture of the neural network classifier. Finally, we chose the simple the multi-layer feedforward perceptron network with simple configuration that can be trained for pattern recognition. The training process requires a matrix of examples (inputs) and matrix of labels (target outputs). The process of training a neural network is about tuning the values of the weights and biases of the network to optimize network performance.

Batch mode was our choice for mode of training. There are two different ways in which training can be implemented: sequential mode and batch mode. In incremental mode, the gradient is computed and the weights are adjusted after each input is applied to the network. In batch mode, all the inputs in the training set are applied to the network before the weights are updated. In most cases, batch training is fairly faster and makes errors than incremental training.

Scaled conjugate gradient optimization. For training multilayer feedforward networks, any standard numerical optimization algorithm can be used to optimize the performance function, but there are a few key ones that have shown excellent performance for neural network training. These optimization methods use the gradient of the network performance with respect to the network weights. The gradient descent is calculated using backpropagation algorithm that involves performing computations backward through the network.

Like standard backpropagation, conjugate gradient methods iteratively try to get closer to the minimum. But while standard backpropagation always proceeds down the gradient of the error function, a conjugate gradient method will proceed in a direction which is conjugate to the directions of the previous steps. Thus the minimization performed in one step is not partially undone by the next, as it is the case with standard backpropagation and other gradient descent methods.

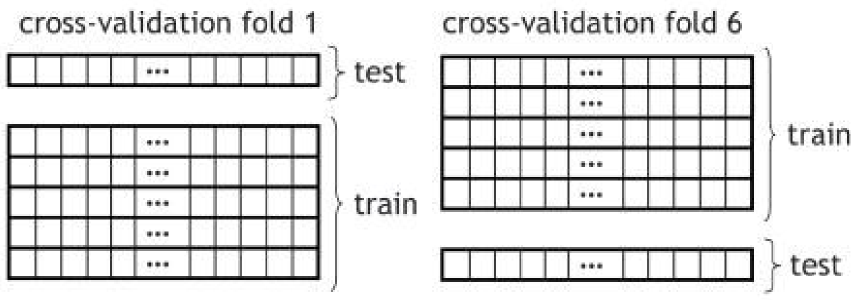
Cross-validation

To avoid overfitting of the neural network and, at the same, to use our data and compute resources more efficiently we used *leave-one-out cross validation* method. It is an improvement over holdout and k-fold techniques. In, classifier algorithm we use this method to compute the maximum number of iterations for training.

In *holdout* method the data set is separated into two sets, the training set and the testing set. The function approximator fits a function using the training set only. Then the function approximator is asked to predict the output values for the data in the testing set. The errors are accumulated to give the mean absolute test set error, which is used to evaluate the model.

In *k-fold* cross validation the data set is divided into k subsets, and the holdout method is repeated k times. Each time, one of the k subsets is used as the test set and the other k-1 subsets are put together to form a training set. Then the average error across all k trials is computed. The advantage of this method is that it matters less how the data gets divided. Every data point gets to be in a test set exactly once, and gets to be in a training set k-1 times.

The *leave-one-out* is a k-fold cross validation technique, where k equals to N, the number of data points in the data set. The optimization function is trained on all the data except for one point and a prediction is made for that point. The average error is computed and used to evaluate the model. The evaluation given by leave-one-out cross validation error is good, although it takes slightly more time to compute than holdout or k-fold techniques, which, in our case, is fairly tolerable.



Leave-one-out cross-validation: each of k folds of examples takes a turn

Testing

We had faced real challenge in preparation of testing set as we tried to train a classifier with as much data as possible. Obviously, we cannot train and test on the same data if the goal is achieve a reliable classifier with higher probability that it would label new test example correctly.

Splitting the dataset by 50:25:25 ratio for training, validation and testing did not work and never brought us any results. Fortunately, there was a *leave-one-out* procedure that allowed us to use almost all of the data for training, validation and testing.

- leave one example out, train on the others, make a prediction for this example
- repeat for each example in the data set
- compute the accuracy of the predictions made for all the examples

Results

Testing results of successful training of the classifier we represented in confusion matrix. Total number of inputs 80, of which 40 items classified to class-1 and other 40 classified as class-2. The accuracy 100%.

	1	2
1	40	0
2	0	40

Further Research

Initially, our project was designed to study capacity of Tree Augmented networks for classification of cognitive activities. However, we found this area of research was not explored enough by the neuroscience and machine learning communities. We narrowed down the scope of the project to focus on Artificial Neural network classifiers to be on schedule.

While exploring the subject we learned that Multi-Voxel Pattern Analysis (MVPA) is increasingly growing area of research. On the other hand, deep learning methods have recently made significant advances in the tasks of classification and representation learning. These tasks are very crucial for brain imaging and neuroscience discovery, making them attractive for porting into neuroimaging research.

It would make sense to study the application of deep learning methods, in particular, deep belief networks to hierarchical feature extraction of high dimensional data. We could use it to analyze the effect of parameter choices on data transformations. This could show that deep learning methods are able to learn physiologically important representations and detect latent relations in neuroimaging data.

References

1. Keller, T. A., Just, M. A., & Stenger, V. A. (2001). Reading span and the time-course of cortical activation in sentence-picture verification. Annual Convention of the Psychonomic Society, Orlando, FL.
2. Wang, X., Mitchell, T., Detecting cognitive states using machine learning. Interim working paper, October 2002.
3. Ramish, J., Learning common features from fMRI data of multiple subjects. Summer project report, August 2004.
4. "Learning to Identify Overlapping and Hidden Cognitive Processes from fMRI Data,"R. Hutchinson, T.M. Mitchell, I. Rustandi, *submitted to HBM 2005*.
5. "Learning to Decode Cognitive States from Brain Images,"T.M. Mitchell, R. Hutchinson, R.S. Niculescu, F.Pereira, X. Wang, M. Just, and S. Newman, *Machine Learning*, Vol. 57, Issue 1-2, pp. 145-175. October 2004.
6. "Training fMRI Classifiers to Detect Cognitive States across Multiple Human Subjects ," X. Wang, R. Hutchinson, and T. M. Mitchell, *Neural Information Processing Systems 2003*. December 2003.
7. "Classifying Instantaneous Cognitive States from fMRI Data," T. Mitchell, R. Hutchinson, M. Just, R.S. Niculescu, F. Pereira, X. Wang, *American Medical Informatics Association Symposium*, October 2003.
8. Using machine learning to detect cognitive states across multiple subjects, X. Wang, CALD KDD Project report, May, 2003.
9. Pattern Recognition and Feed-forward Networks, C.M. Bishop, Microsoft Research, The MIT Encyclopedia of the Cognitive Sciences, 1999