Decoding of Neural Populations in Visual Cortex

Noam Roth

Washington University in St. Louis

August 7th, 2009

This project compared several decoders in how they extract information from neural populations in the Macaque visual cortex. Our goals were to better understand neural variability and information contained in neural population activity about visual stimuli. The project was done in the lab of Dr. Wei Ji Ma at Baylor College of Medicine, as part of the Rice/TMC/UH Computational Neuroscience REU, supported by grant DMS-0755294.

Introduction

In order to function in the world around us, our brain makes use of sensory information from its environment to produce corresponding behavioral responses. To do this, it creates and modifies representations of stimuli in the environment. The sensory information is encoded (or represented) by populations of neurons in the brain. For different stimuli, different neurons respond and create a population response which represents that stimulus. Additionally, the same stimulus produces different population responses on different trials, due to neural variability. In this project we compare different procedures for extracting information from neural representations of stimuli, or population responses. Specifically, using neural population activity in Macaque V1 neurons, we ask how much information it contains about visual stimuli.

Physiological Data

The physiological data that we used was collected by the Tolias lab at Baylor College of Medicine. This neural population response data was in the form of spike counts per second, recorded using multi-electrode (tetrode) recordings in the V1 region (visual cortex) of Macaque monkeys. The stimuli that the monkeys viewed were orientation gratings, in both low and high contrast. Orientations presented were 0, 22.5, 45, 67.5, 90, 112.5, 135, 157.5 degrees. The spike counts were recorded on four different days, with 82 neurons in total. For each orientation grating, neurons were recorded on many trials, varying from 10-40 depending on day.

Population Coding

Population coding is a way of representing information about a stimulus through the simultaneous activity of a large set of neurons sensitive to this feature (a population). Information about the stimulus is first *encoded* by a population of neurons, as the population response is different for different stimuli. Figure 1 displays the response of different neurons (in varying colors) to different orientations (stimuli). This response is shown as mean response, or tuning curves, denoted by f_i(s), where *i* is a neuron and s is the stimulus shown. The mode of each neuron's tuning curve is called its *preferred orientation*, as it is the orientation where the neuron will most likely fire the most. However, due to neural variability, the tuning curves shown are not a perfect representation of what occurs on individual trials. An example of variable single trial activity can be seen in figure 2.

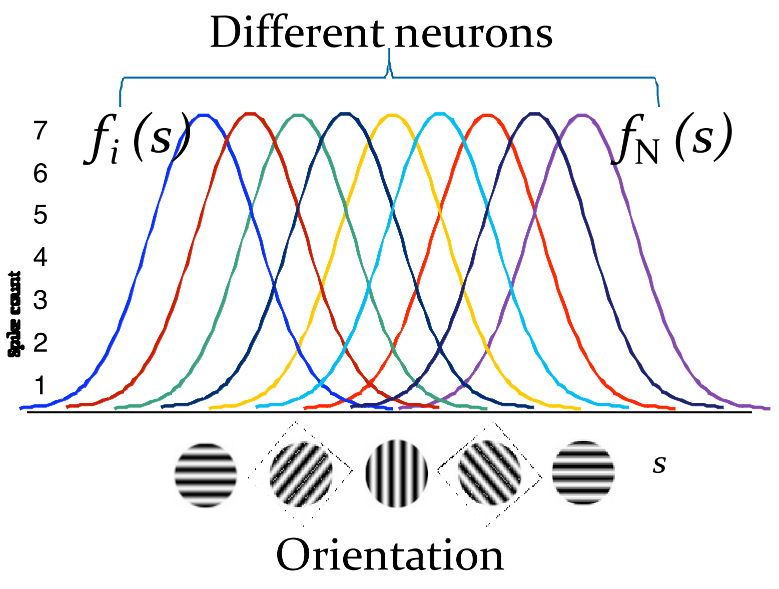
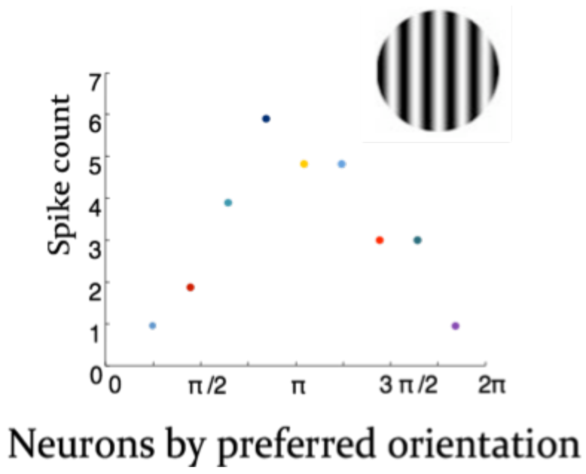


Figure 1: Idealized Gaussian tuning curves for different orientations. Each color represents a different neuron.

Neural activity is variable from trial to trial, so we can look at a conditional probability distribution: the probability of a population response given a specific stimulus. This probability is given by p(**r** | s), called the response distribution.



The second process involved in population coding is called *decoding*. A decoder is a procedure for estimating the stimulus from the population pattern of activity on a single trial. The estimated stimulus can then be compared to the true stimulus on that trial.

Figure 2: Colors correspond to neuron colors from figure 1. Grating shows the orientation for the trial shown.

This project's goal was to compare the performance of several decoders, both simple and machine learning, to see how much information they can extract from neural populations. This can broadly simulate how the brain decodes neural information, but understanding what decoders work optimally on population responses will mainly be useful from an experimenter's viewpoint.

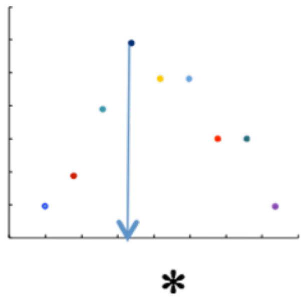
Studies in population coding also have potential applications in neural prosthetics. That is, as we begin to understand how to extract information from population responses, we can create artificial help for those whose brains cannot do so (both sensory and motor).

Decoders

In order to understand what information we could extract from neural population coding, we compared the goodness of several decoders in estimating the stimulus presented on each trial. We implemented each decoder in MATLAB, and compared the estimated stimuli to the true presented stimuli. In this way we were able to quantify the performance of each decoder.

Winner-take-all

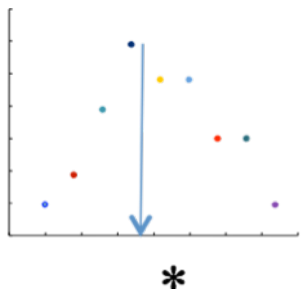
The estimated stimulus chosen is the preferred stimulus of the neuron with the highest response:



$$\hat{s} = s_j : j = \underset{i}{\operatorname{argmax}}\, r_i$$

The winner-take-all decoder estimates the presented stimulus by simply picking the neuron with the highest response. This decoder requires knowledge of preferred orientations of neurons (see figure 3). However, because of the immense neural variability and noise that occurs naturally in the brain, and because this decoder only utilizes information from one – the highest – neuron (as opposed to from the whole population) we expected this decoder to do very poorly.

Population Vector



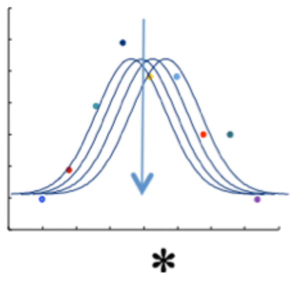
The estimated stimulus is chosen by weighing each neuron's preferred stimulus proportionally to its response:

$$\hat{s} = \frac{1}{2}\arctan\frac{\sum_{i=1}^{N} r_i \sin(2s_i)}{\sum_{i=1}^{N} r_i \cos(2s_i)}$$

By calculating a vector sum of the responses across all neurons – where the angle is the preferred orientation of the neuron and the vector length is its spike count response – each neuron votes for itself by how strongly it responds. This is a better decoder than winner-take-all, but it does not take into account any form of neural variability.

Template-matching

This decoder match the observed population response with a set of tuning curve templates (mean population responses for different stimuli), using the sum-squared difference as error measure:



$$\hat{s} = \underset{s}{\operatorname{argmin}} \sum_{i=1}^{N} (r_i - f_i(s))^2$$

While this decoder also does not take into account a form of neural variability, it does use information from the tuning curves of the neurons (more information than in ‘winner-take-all’, because it is the entire mean response and not just preferred orientation).

Maximum-likelihood

This decoder computes the probability that a stimulus value produced the given response, and select the stimulus value for which the probability is highest:

$$\hat{s} = \underset{s}{\operatorname{argmax}}\, p(\vec{r} \mid s)$$

This decoder takes neural variability into account. One assumption commonly made in past studies is that neural variability is independent between neurons and Poisson-distributed for each neuron, so the response distribution is the product of Poisson-distributed neurons:

$$p(\vec{r} \mid s) = \prod_{i=1}^{N} \frac{e^{-f_i(s)} f_i(s)^{r_i}}{r_i!}$$

However, the assumption of independence may be a faulty one because neurons, especially within populations of neurons, are highly interconnected, and thus cause each other to fire. This assumption will be addressed in the variational technique discussed later in this report.

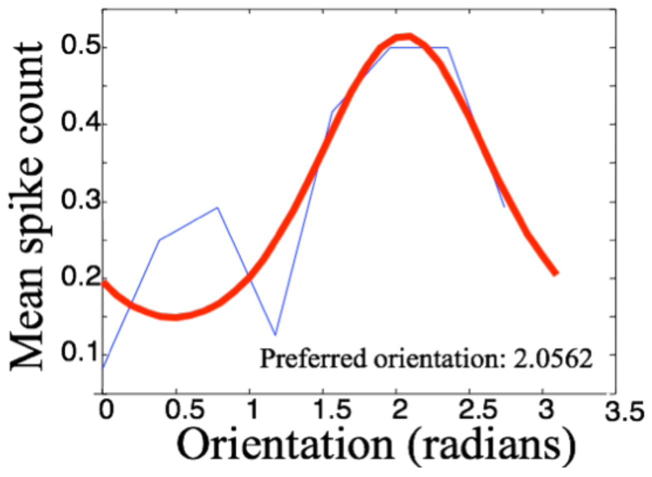
Many of the decoders discussed above require knowledge of mean response and preferred orientation. The data only contained discrete firing information at eight different orientations, so Von Mises (circular Gaussians) were fit using (see figure 3)

The Rayleigh's test for circular uniformity was also used on the data, producing a p value for each neuron which indicated how significantly peaked its mean response was.

For the decoders that required not just preferred orientation, but the entire tuning curve information (Template Matching and Maximum Likelihood), two techniques were used:

(1) using fitted Von Mises (circular Gaussian) tuning curves

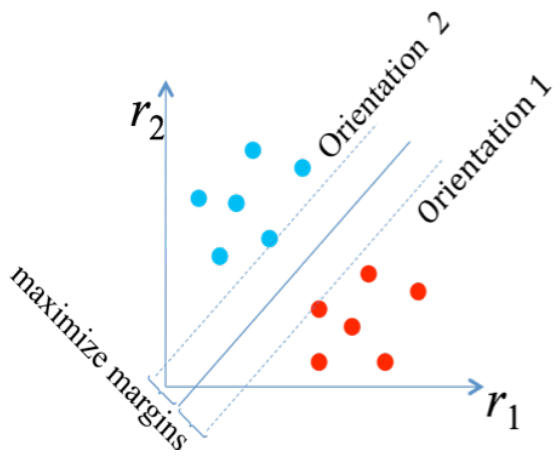
(2) using linear interpolation of mean activity



$$f_i(s) = ae^{\kappa \cos(2(s-s_i))} + b$$

Figure 3: An example neuron's fitted Von Mises tuning curve and preferred orientation.

Support Vector Machines



This decoder is a machine learning algorithm which classifies data points by separating them with a hyperplane which has a maximized margin from the data.

Figure 4: An illustration of how SVM works for a two dimensional (only two neuron) space and two orientations.

As SVM is a classifying algorithm, on its most simple level it classifies data into two classes. There are two ways to implement multi-class classification in SVM:

(1) One vs. Rest: separates one orientation from all the rest, and the classifier with the highest output function assigns the class.

(2) One vs. One: separates one orientation from one other orientation (pairwise), then classifies points to whichever class has the most "votes" for that point.

Support Vector Machines use kernel functions to fit the maximum-margin hyperplane in a high dimensional feature space. In our case, the transformation is non-linear: using a radial basis function. In order to learn the data well, we fit two hyperparameters that minimized error for the first day's data. The first was σ, which fit the kernel function (radial basis function) well to the data. Second was C, a soft margin parameter which allowed for mislabeling of some data points.

Variational Bayesian Logistic Regression

As discussed above, one goal of this project was to address the assumption of independence between neurons. To do so, we generalized our assumption of neural variability to the Poisson-like family of distributions, which does not assume independence. The response distribution is given below:

$$p(\vec{r} \mid s) = \varphi(\vec{r}) e^{\vec{h}(s)\cdot\vec{r}}$$

In order to estimate the presented stimulus based on the population response data, an algorithm, *Variational Bayesian Logistic Regression*. (Drugowitsch 2008) was used. The algorithm uses logistic regression and variational methods to find **h**(s). Between two classes (orientations), we need to find **h**(s$_1$)-**h**(s$_2$), or Δ**h**.

To choose a stimulus, we must look at the probability of a specific stimulus given the population response. We can manipulate this probability in the following way:

$$p(s_1 \mid \mathrm{r}) = \frac{p(\mathrm{r} \mid s_1)}{p(\mathrm{r} \mid s_1) + p(\mathrm{r} \mid s_2)} = \frac{\varphi(\mathrm{r}) e^{\mathrm{h}(s_1)\cdot \mathrm{r}}}{\varphi(\mathrm{r}) e^{\mathrm{h}(s_1)\cdot \mathrm{r}} + \varphi(\mathrm{r}) e^{\mathrm{h}(s_2)\cdot \mathrm{r}}} = \frac{1}{1 + e^{(\mathrm{h}(s_1)-\mathrm{h}(s_2))\cdot \mathrm{r}}} = \frac{1}{1 + e^{\Delta \mathrm{h}\cdot \mathrm{r}}} = \sigma(\Delta \mathrm{h} \cdot \mathrm{r})$$

This sigmoid model is then approximated by maximizing a lower bound, and its posterior is approximated by a variational posterior. By using these variational methods, we can find probabilities for classifying the data in each class, which is how the estimates of the stimuli are found.

How do we determine goodness?

To compare the different decoders, we used three methods of determining goodness:

(1) Low bias: The average estimate should equal the true orientation

(2) Low variance: The estimates should all be close to each other

(3) Combined error: Incorporates bias and variance (figure 5)

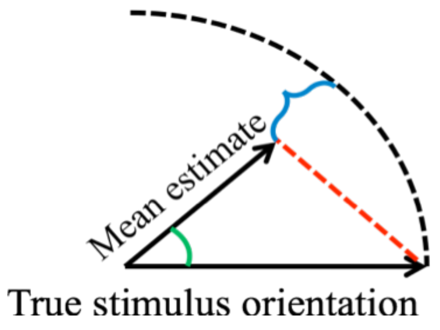


Figure 5: Bias in green, variance in blue, combined error in red.

Results

Results of the decoder comparison were found by calculating the errors of each decoder. Figure 6 displays a histogram for each decoder, per contrast, of the distance (in degrees) between the predicted orientation and true presented orientation. The number given in the top right of each histogram is the error measure, described above. The higher the error measure, the worse the performance of the decoder in extracting information from the population activity. Error itself is displayed in figure 7.

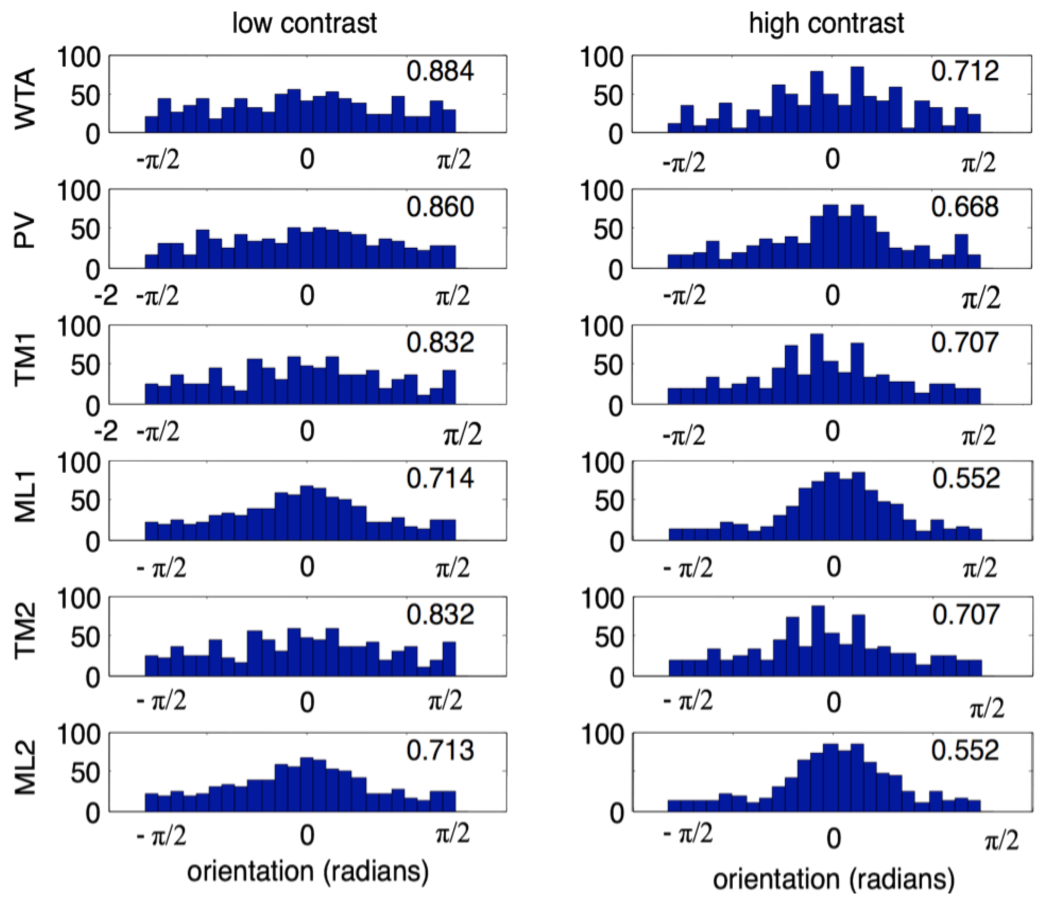


Figure 6: Histograms of distance from true stimulus for each decoder. ML1 and TM1 use Von Mises curves, ML2 and TM2 use linear interpolation.

Variance and Bias are shown in figure … Variance decreases by decoder from winner-take-all to maximum likelihood. Bias is generally very low, and is found to be either negative or positive.

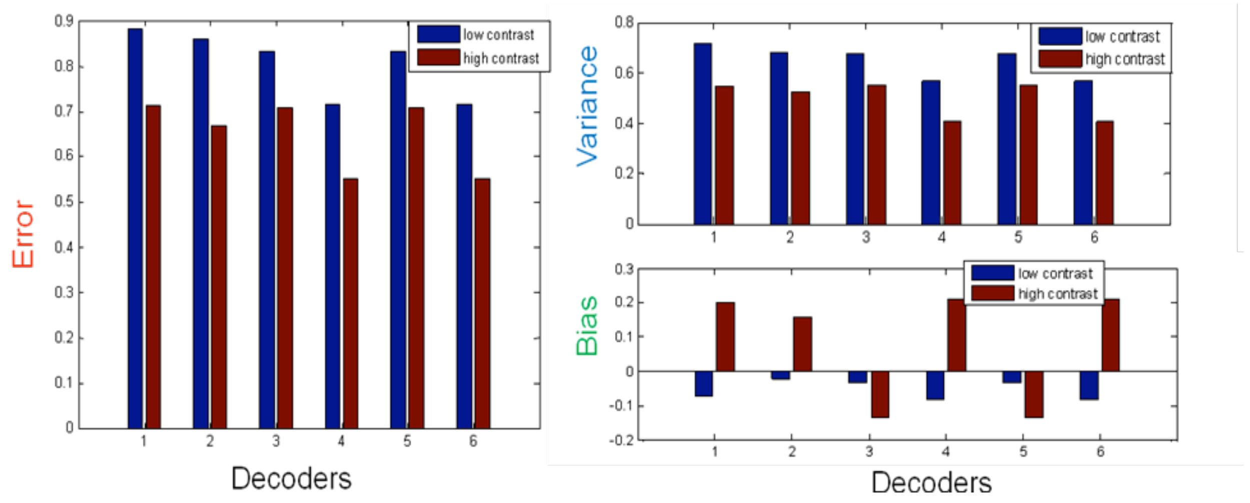


Figure 7: Combined error, variance, and bias. Decoders are numbered in the same order as figure 6.

In the implementation of the *Variational Bayesian Logistic Regression* in order to incorporate the Poisson-like variability, we had an overfitting problem due to the lack of hyperparameters. Because there was nothing in the algorithm that we could train on one part of the data and use in the rest of the data, the method overfitted the data, and thus the error measure was very low.

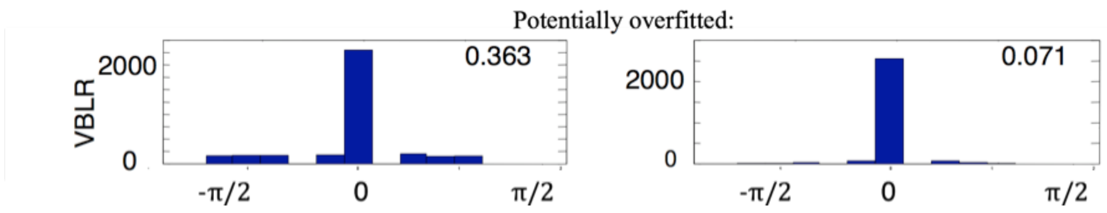


Figure 8: Histogram of distances from true orientation, variational method. Error measure in corner. Low contrast is left, and high contrast is right.

The SVM algorithm had similar overfitting problems, more due to basic implementation rather than theoretical hyperparameter problems. Histograms for one day's worth of data are displayed below. As can be seen, specifically to high contrast orientations, the variance was very narrow and thus SVM seems to be very overfitted.

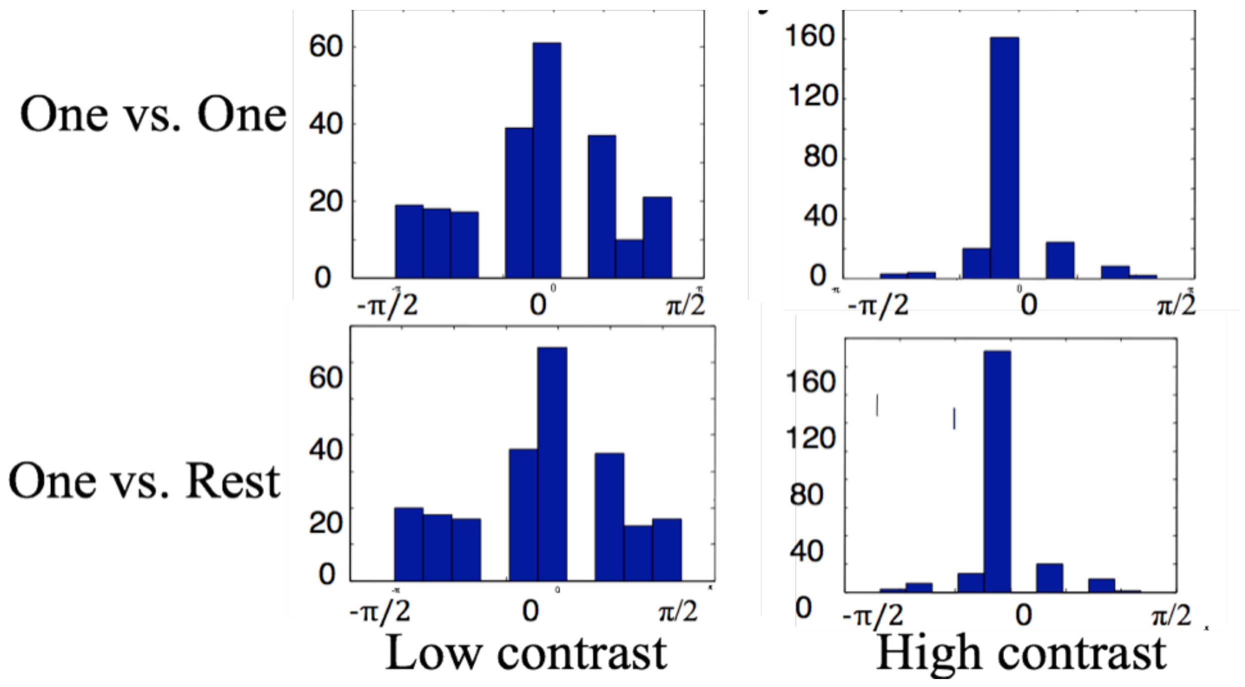


Figure 9: Histogram of distances from true orientation, SVM decoder.

Conclusions

Among the simple decoders, winner-take-all was worst, and maximum-likelihood was best, as expected. Additionally the high contrast data encode more information, as expected. Preliminary results for SVM and VBLR suggest that these decoders might do better than the simple decoders, however we have an overfitting problem.

While a lot was confirmed about the information encoded in population activity, future goals of this project are still numerous. Firstly, we wish to work out problems in the implementation of SVM, so that it can also be compared. Secondly, we want to find hyperparameters or a way to fix the overfitting problem in the variational method, in order to have a concrete comparison with the remaining decoders, and obtain a conclusion about the assumption of independence in neural variability.

References

Bishop, C. (2006). *Pattern Recognition and Machine Learning*. New York: Springer Science.

Drugowitsch, J. (2008). “Bayesian Logistic Regression.”

MacKay, D (2003). *Information Theory, Inference, and Learning Algorithms*. Cambridge: Cambridge University Press.

Appendix A: selected MATLAB code

Low level decoders:

```
clear all;
load gratingDat;
load variables;

degrees = [dat(1).conditions.orientation];  % real orientations
m = 6; % number of decoders tested
error = zeros(819,6,2);
orientationvec = [degrees(1:2:16) * pi/180]; %list of orientations (because
contrast alternates)
testorientationvec = 0:0.005:pi; % hypothesized orientations
for neuron = 1:length(s_i)
    f1(neuron,:) = a(neuron) .*exp(k(neuron).*cos(2*(testorientationvec-
s_i(neuron)))+ b(neuron)); % Estimated Von Mises tuning curves of all neurons
    f2(neuron,:) = f1(neuron,:);%interp1([orientationvec pi],
[mean_spike_count(neuron,:) mean_spike_count(neuron, 1)],
testorientationvec); % Interpolated tuning curves of all neurons
end



contrastvec = [1 2];
for contrastind = 1:length(contrastvec)
    conditionvec = contrastvec(contrastind):2:16;


    total_num_trials = 0;
    total_num_neurons = 0;
    for day = 1:4
        for conditionind = 1:length(conditionvec)
            condition = conditionvec(conditionind);

            [num_trials num_neurons] = size(dat(day).spikeCounts{condition});
% number of trials and number of neurons
            neuronindices = total_num_neurons + (1:num_neurons);
            trialindices = total_num_trials + (1:num_trials);

            decoders(trialindices,1, contrastind) =
orientationvec(conditionind) * ones(1,num_trials); % real orientations
            spike_count_matrix = dat(day).spikeCounts{condition}; % spike
counts for all neurons for all trials (that day, that condition)

            % Winner-take-all decoder
            [value index] = max(spike_count_matrix,[],2); % index is between
1 and numneurons(day)
            wta_s_hat = s_i(total_num_neurons + index);

            % Population vector decoder
            preferred_angles = repmat(2*s_i(neuronindices)', num_trials,1); %
preferred orientations, rescaled and copied across all trials
```

```
            pv_x_component = sum(spike_count_matrix .*
cos(preferred_angles),2)./sum(spike_count_matrix,2);
            pv_y_component = sum(spike_count_matrix .*
sin(preferred_angles),2)./sum(spike_count_matrix,2);
            pv_s_hat = atan(pv_y_component./pv_x_component);
            pv_s_hat = pv_s_hat + pi * (pv_x_component < 0);
            pv_s_hat = mod(pv_s_hat/2, pi);

            % Template matching decoder
            TMsum = spike_count_matrix * f1(neuronindices,:);
            value = max(TMsum,[],2);
            for i = 1:num_trials
                indices = find(TMsum(i,:)==value(i));
                index(i) = indices(ceil(rand * length(indices)));
            end
            tm1_s_hat = testorientationvec(index)';

            TMsum = spike_count_matrix * f2(neuronindices,:);
            value = max(TMsum,[],2);
            for i = 1:num_trials
                indices = find(TMsum(i,:)==value(i));
                index(i) = indices(ceil(rand * length(indices)));
            end
            tm2_s_hat = testorientationvec(index)';



            % Maximum-likelihood decoder

            MLsum = spike_count_matrix * log(f1(neuronindices,:));
            value = max(MLsum,[],2);
            for i = 1:num_trials
                indices = find(MLsum(i,:)==value(i));
                index(i) = indices(ceil(rand * length(indices)));
            end
            ml1_s_hat = testorientationvec(index)';

            MLsum = spike_count_matrix * log(f2(neuronindices,:));
            value = max(MLsum,[],2);
            for i = 1:num_trials
                indices = find(MLsum(i,:)==value(i));
                index(i) = indices(ceil(rand * length(indices)));
            end
            ml2_s_hat = testorientationvec(index)';


            decoders(trialindices, 2:m+1, contrastind) = [wta_s_hat pv_s_hat
tm1_s_hat ml1_s_hat tm2_s_hat ml2_s_hat];
            total_num_trials = total_num_trials + num_trials;
        end
        total_num_neurons = total_num_neurons + num_neurons;
    end

    error(:,:,contrastind) = decoders(:,2:m+1, contrastind)-
repmat(decoders(:,1, contrastind),1,m);
```

```
    error = error + pi*(error<-pi/2-pi/48) - pi*(error>=pi/2-pi/48);

    x = mean(cos(2*error(:,:,contrastind)),1);
    y = mean(sin(2*error(:,:,contrastind)),1);
    bias(:,contrastind) = atan(y./x);

    error_vector_length_decoders(contrastind,:) = sqrt((1-x).^2 + y.^2)
end

%% Plots
figure;
decodernames = {'WTA', 'PV', 'TM', 'ML', 'TM2', 'ML2'};
contrastnames = {'low contrast', 'high contrast'};
for contrastind = 1:length(contrastvec)
    for i = 1:m
        subplot(m,2,2*(i-1) + contrastind);
          hist(error(:,i, contrastind),-pi/2:pi/24:pi/2)
        if contrastind == 1
            ylabel(decodernames(i));
        end
        if i==1
            title(contrastnames(contrastvec(contrastind)));
        end
        text(1,80,num2str(error_vector_length_decoders(contrastind,i)));
        ylim([0 100])
    end
end
figure; bar(1:4, error_vector_length');
```

SVM:

```
%% create X and Y (data)
clear all;
tic
kappa = 1;
c = 0;
num_trials = 100;
contrast_matrix = [.1 .2];
presentedorivec = (10:10:180) * pi/180; % presented orientations, only used
in machine learning methods
preforivec = (10:10:180) * pi/180; % preferred orientations of neurons
N = length(preforivec);


[pres pref] = meshgrid(presentedorivec, preforivec);
f_matrixbase_presented = exp(kappa*cos(2*(pres - pref))) + c; %neuron by
condition

for contrastind = 1:2
    contrast = contrast_matrix(contrastind);
    f_matrix_presented = contrast * f_matrixbase_presented;
    r_train = poissrnd(repmat(f_matrix_presented, [1,1,num_trials])); %neuron
by condition by trial
    spike_counts(:,:,contrastind) = reshape(r_train, N, N*num_trials)';
%trial*condition by neuron
end


cd spiderMain
load gratingDat;
use_spider;
load variables;
day = 1;

X_low  = [dat(day).spikeCounts{1};
    dat(day).spikeCounts{3};
    dat(day).spikeCounts{5};
    dat(day).spikeCounts{7};
    dat(day).spikeCounts{9};
    dat(day).spikeCounts{11};
    dat(day).spikeCounts{13};
    dat(day).spikeCounts{15}];


X_high  = [dat(day).spikeCounts{2};
    dat(day).spikeCounts{4};
    dat(day).spikeCounts{6};
    dat(day).spikeCounts{8};
    dat(day).spikeCounts{10};
    dat(day).spikeCounts{12};
    dat(day).spikeCounts{14};
    dat(day).spikeCounts{16}];
```

```
X_fake_low = spike_counts(:,:,1);

X_fake_high = spike_counts(:,:,2);


Y_low = [ones(size(dat(day).spikeCounts{1},1),1), -
ones(size(dat(day).spikeCounts{1},1),7);
-ones(size(dat(day).spikeCounts{3},1),1),
ones(size(dat(day).spikeCounts{3},1),1), -
ones(size(dat(day).spikeCounts{3},1),6);
-ones(size(dat(day).spikeCounts{5},1),2),
ones(size(dat(day).spikeCounts{5},1),1), -
ones(size(dat(day).spikeCounts{5},1),5);
-ones(size(dat(day).spikeCounts{7},1),3),
ones(size(dat(day).spikeCounts{7},1),1), -
ones(size(dat(day).spikeCounts{7},1),4);
-ones(size(dat(day).spikeCounts{9},1),4),
ones(size(dat(day).spikeCounts{9},1),1), -
ones(size(dat(day).spikeCounts{9},1),3);
-ones(size(dat(day).spikeCounts{11},1),5),
ones(size(dat(day).spikeCounts{11},1),1), -
ones(size(dat(day).spikeCounts{11},1),2);
-ones(size(dat(day).spikeCounts{13},1),6),
ones(size(dat(day).spikeCounts{13},1),1), -
ones(size(dat(day).spikeCounts{13},1),1);
-ones(size(dat(day).spikeCounts{15},1),7),
ones(size(dat(day).spikeCounts{15},1),1)];

Y_high = [ones(size(dat(day).spikeCounts{2},1),1), -
ones(size(dat(day).spikeCounts{2},1),7);
-ones(size(dat(day).spikeCounts{4},1),1),
ones(size(dat(day).spikeCounts{4},1),1), -
ones(size(dat(day).spikeCounts{4},1),6);
-ones(size(dat(day).spikeCounts{6},1),2),
ones(size(dat(day).spikeCounts{6},1),1), -
ones(size(dat(day).spikeCounts{6},1),5);
-ones(size(dat(day).spikeCounts{8},1),3),
ones(size(dat(day).spikeCounts{8},1),1), -
ones(size(dat(day).spikeCounts{8},1),4);
-ones(size(dat(day).spikeCounts{10},1),4),
ones(size(dat(day).spikeCounts{10},1),1), -
ones(size(dat(day).spikeCounts{10},1),3);
-ones(size(dat(day).spikeCounts{12},1),5),
ones(size(dat(day).spikeCounts{12},1),1), -
ones(size(dat(day).spikeCounts{12},1),2);
-ones(size(dat(day).spikeCounts{14},1),6),
ones(size(dat(day).spikeCounts{14},1),1), -
ones(size(dat(day).spikeCounts{14},1),1);
-ones(size(dat(day).spikeCounts{16},1),7),
ones(size(dat(day).spikeCounts{16},1),1)];

Y_fake_low = -ones(1800,18);
Y_fake_high = -ones(1800,18);
```

```
for i = 1:18
    Y_fake_low((i-1)*100+1:i*100, i) = 1;
    Y_fake_high((i-1)*100+1:i*100, i) = 1;
end


d1 = data(X_low, Y_low)
d2 = data(X_high, Y_high);


cd ..

[C1 rbf1 error1 errorlength1 tr_1] = svmdecodertry(d1,1);
errormatrixlow(:,1,1) = error1;
[C2 rbf2 error2 errorlength2 tr_2] = svmdecodertry(d1,2);
errormatrixlow(:,1,2) = error2;
[C3 rbf3 error3 errorlength3 tr_3] = svmdecodertry(d2,1);
errormatrixhigh(:,1,1) = error3;
[C4 rbf4 error4 errorlength4 tr_4] = svmdecodertry(d2,2);
errormatrixhigh(:,1,2) = error4;
```

SVM function:

```
function [C rbf error error_vector_length tr] = svmdecodertry(d, method);
%tr c_trained]
orientation  = (0:7) * 22.5 * pi/180;
% orientation2  = (0:17) * 10 * pi/180;

% if orientation == 1
%     orientation = orientation1;
% else
%     orientation = orientation2;
% end
if method == 1 % one vs rest
    rbfvec = 16:19;
    Cvec = [0.0001:0.001:0.008];
else % one vs one
    rbfvec = 12:17;
    Cvec = [2:1:22];
end

a = svm;
param_matrix = [];
% tr_matrix = [];
% c_trained_matrix = [];
error_vector_length = [];
```

```
i = 1;
for rbfind = 1:length(rbfvec)
    rbf = rbfvec(rbfind);
    for Cind = 1:length(Cvec)
        C = Cvec(Cind);
        param_matrix(i,:) = [C rbf];
        a.C = C;
        a.child = kernel('rbf',rbf);
        if method == 1
            b = one_vs_rest(a);
        else
            b = one_vs_one(a);
        end
        c = cv(b);
        [tr c_trained] = train(c,d);

        tr_matrix.i = tr;
        c_trained_matrix.i = c_trained;

        truth = [tr{1}.X; tr{2}.X; tr{3}.X; tr{4}.X; tr{5}.X];
        classification = [tr{1}.Y; tr{2}.Y; tr{3}.Y; tr{4}.Y; tr{5}.Y];
       % tst=test(a,d1) %%%%%%fix d2

      %  test_classification =  [tst{1}.X; tst{2}.X; tst{3}.X; tst{4}.X;
tst{5}.X];

        [value true_orientation_index(:,i)] = max(truth,[], 2);
        [value classified_orientation_index(:,i)] = max(classification, [],
2);
     %   [value test_classified_orientation_index(:,i)] =
max(test_classification, [], 2);
        i = i+1
    end
end
error = orientation(classified_orientation_index) -
orientation(true_orientation_index);
error = error + pi*(error<-pi/2-pi/48) - pi*(error>=pi/2-pi/48);
x = mean(cos(2*error));
y = mean(sin(2*error));
error_vector_length = sqrt((1-x).^2 + y.^2);



[value index] = min(error_vector_length);
%index
%size(tr_matrix)
C = param_matrix(index,1);
rbf = param_matrix(index,2);
tr = tr_matrix
%c_trained = c_trained_matrix.index;

error = error(:,index);
```

Variational Bayesian Logistic Regression:

```
clear;
load gratingDat;
degrees = [dat(1).conditions.orientation];  % real orientations
orientationvec = degrees(1:2:16) * pi/180; %list of orientations (because
contrast alternates)
presentedorivec = orientationvec; % presented orientations, only used in
machine learning methods
combinationmatrix = nchoosek(1:length(presentedorivec), 2); % all possible
presented orientation pairs

k = 2;

contrastvec = [1 2];
for day = 1:4
    day;
    classificationmatrix = [];
    VBLRerror = [];
    classificationmatrix2 = [];
    VBLRerror2 = [];
    for contrastind = 1:length(contrastvec)
        classification = [];
        r_train_matrix = [];
        r_test_matrix = [];
        contrast = contrastvec(contrastind);
        if contrast == 1
            conditionvec = 1:2:15;
        elseif contrast == 2
            conditionvec = 2:2:16;
        end
        for conditionind = 1:length(conditionvec)
            condition = conditionvec(conditionind);
            temp = dat(day).spikeCounts{condition};
            temp2(conditionind)  = size(temp,1);
        end
        temp3 = min(temp2);
        temp4 = floor(temp3/k);
        for conditionind = 1:length(conditionvec)
            condition = conditionvec(conditionind);
            spike_count_matrix = dat(day).spikeCounts{condition};
            num_trials(conditionind)= size(spike_count_matrix,1);
            r_train = spike_count_matrix(1:(k-1)*temp4,:);
            r_train_matrix(:,:,conditionind) = r_train;
            r_test = spike_count_matrix((k-1)*temp4+1:k*temp4,:);
            r_test_matrix(:,:,conditionind) = r_test;
        end
        dh = [];
        for i = 1:size(combinationmatrix,1)
            class1 = combinationmatrix(i,1);
            class2 = combinationmatrix(i,2);
            X_VBLR =
[squeeze(r_train_matrix(:,:,class1));squeeze(r_train_matrix(:,:,class2))];
            X_test =
[squeeze(r_test_matrix(:,:,class1));squeeze(r_test_matrix(:,:,class2))];
            Y = [ones((k-1)*temp4,1); -ones((k-1)*temp4,1)];


            [w, V, invV, logdetV, E_a, L] = bayes_logit_fit(X_VBLR,Y);
            out = bayes_logit_post(X_test, w, V, invV);


            dh(:,i) = w;

            temp = (out >= .5);
            chosenclass = class1 .* temp + class2 .* (1-temp);
            trueclass = [class1 * ones(temp4,1); class2 * ones(temp4,1)];
            classification = [classification; [trueclass chosenclass]];



       %% switching contrast
            if contrast ==1
                conditionvec2 = 2:2:16;
            else
                conditionvec2 = 1:2:15;
            end

              classification2 = [];
              r_train_matrix2 = [];
              r_test_matrix2 = [];
              conditionvec = 1:2:15;
            for conditionind = 1:length(conditionvec)
                condition = conditionvec(conditionind);
                temp_2 = dat(day).spikeCounts{condition};
                temp2_2(conditionind) = size(temp,1);
            end
            temp3_2 = min(temp2_2);
            temp4_2 = floor(temp3_2/k);
            for conditionind = 1:length(conditionvec)
                condition = conditionvec(conditionind);
                spike_count_matrix2 = dat(day).spikeCounts{condition};
                num_trials2(conditionind)= size(spike_count_matrix,1);
                r_train2 = spike_count_matrix2(1:(k-1)*temp4_2,:);
                r_train_matrix2(:,:,conditionind) = r_train2;
                r_test2 = spike_count_matrix2((k-1)*temp4_2+1:k*temp4_2,:);
                r_test_matrix2(:,:,conditionind) = r_test2;
            end
            dh2 = [];
            for i = 1:size(combinationmatrix,1)
                class1_2 = combinationmatrix(i,1);
                class2_2 = combinationmatrix(i,2);
                X_VBLR2 =
[squeeze(r_train_matrix2(:,:,class1_2));squeeze(r_train_matrix2(:,:,class2_2)
)];
                X_test2 =
[squeeze(r_test_matrix2(:,:,class1_2));squeeze(r_test_matrix2(:,:,class2_2))]
;
                Y2 = [ones((k-1)*temp4,1); -ones((k-1)*temp4,1)];


                [w2, V2, invV2, logdetV2, E_a2, L2] =
bayes_logit_fit(X_VBLR2,Y2);
                out = bayes_logit_post(X_test, w2, V2, invV2);

                dh(:,i) = w2;

                temp = (out >= .5);
                chosenclass2 = class1_2 .* temp + class2_2 .* (1-temp);
                trueclass2 = [class1_2 * ones(temp4_2,1); class2_2 *
ones(temp4_2,1)];
                classification2 = [classification2; [trueclass2
chosenclass2]];
            end


 %%

        end
        classificationmatrix(:,:, contrastind) = classification;
        classification_orientation_matrix =
presentedorivec(classificationmatrix);
        VBLRerror(:, contrastind) =
classification_orientation_matrix(:,2,contrastind) -
classification_orientation_matrix(:,1,contrastind);

        classificationmatrix2(:,:, contrastind) = classification2;
        classification_orientation_matrix2 =
presentedorivec(classificationmatrix2);
        VBLRerror2(:, contrastind) =
classification_orientation_matrix2(:,2,contrastind) -
classification_orientation_matrix2(:,1,contrastind);

    end % of contrast loop

    VBLRerror = VBLRerror + pi*(VBLRerror<-pi/2) - pi*(VBLRerror>=pi/2);
    switch day
        case 1
            VBLRerror_matrix1 = VBLRerror;
        case 2
            VBLRerror_matrix2 = VBLRerror;
        case 3
```

```
            VBLRerror_matrix3 = VBLRerror;
        case 4
            VBLRerror_matrix4 = VBLRerror;
    end
    x = mean(cos(2*VBLRerror),1);
    y = mean(sin(2*VBLRerror),1);
    VBLRerror_vector_length(:,day) = sqrt((1-x).^2 + y.^2);


    VBLRerror2 = VBLRerror2 + pi*(VBLRerror2<-pi/2) - pi*(VBLRerror2>=pi/2);
    switch day
        case 1
            VBLRerror2_matrix1 = VBLRerror2;
        case 2
            VBLRerror2_matrix2 = VBLRerror2;
        case 3
            VBLRerror2_matrix3 = VBLRerror2;
        case 4
            VBLRerror2_matrix4 = VBLRerror2;
    end
    x2 = mean(cos(2*VBLRerror2),1);
    y2 = mean(sin(2*VBLRerror2),1);
    VBLRerror_vector_length2(:,day) = sqrt((1-x2).^2 + y2.^2);

end % of day loop

    VBLRerror_matrix = [VBLRerror_matrix1; VBLRerror_matrix2;
VBLRerror_matrix3; VBLRerror_matrix4];

    VBLRerror_matrix2 = [VBLRerror2_matrix1; VBLRerror2_matrix2;
VBLRerror2_matrix3; VBLRerror2_matrix4];
%Plots
figure;
subplot(1,2,1);
hist(VBLRerror_matrix(:,1))
text(1,1600,num2str(mean(VBLRerror_vector_length(1,:))));

subplot(1,2,2);
hist(VBLRerror_matrix(:,2))
text(1,2400,num2str(mean(VBLRerror_vector_length(2,:))));


figure;
subplot(1,2,1);
hist(VBLRerror_matrix2(:,1))
text(1,1000,num2str(mean(VBLRerror_vector_length2(1,:))));

subplot(1,2,2);
hist(VBLRerror_matrix2(:,2))
text(1,1000,num2str(mean(VBLRerror_vector_length2(2,:))));
```