Cracking Neural Code for Facial Identity

Analyzing Changes in Neuron Response after Morphing Faces

A Report Submitted in Partial Fulfillment of the Requirements for
SYDE 556/750

Ajay Patel
20730719

Faculty of Math
Department of Computer Science

April 15, 2020

Course Instructor:
Andreas Stöckel

Contents

1 Introduction

How exactly does the brain recognize faces? What features of the face are seen first, and which play a bigger role in identifying someone that you know? This idea of how the brain represents the identity of a complex object is a central challenge of visual neuroscience. In machine learning, facial recognition has been solved already, but it does not address how an image is encoded in a real brain.

Here, we explore facial representation in a neuron population and try to determine which facial features play a larger role in representation. If we can learn more about this problem, then it would be interesting to see if the same methodology can be used for general visual object representation.

Although we know that visual information is processed in the visual cortex of the cerebral cortex, the exact code for face or object encoding used by single neurons is still unknown. Single neurons are understood to respond more to the preferred stimulus, and do not excite as much to other stimulus. We will build off of this idea, and try to manually morph our input to find mappings of a change in input to a change in neural response.

More specifically, we will do this by analyzing changes in neural response of a population with systematic changes in specific components of the face which have the most influence in facial identification.

1.1 Design Specification and Methods

For this project the workflow will look something like:

1. **Choosing a Suitable Dataset**
   We will use the same dataset that was used for Lecture 8, as it is a fairly small dataset and has appealing aspects for representation such as fixed eye location, and perfectly cropped to only include the face. The data set Utrecht ECVP was obtained from Chang and Tsao, 2017.

   The larger dataset was not used due to lack of computational power. However we had a powerful machine, a larger dataset would definately have been used.

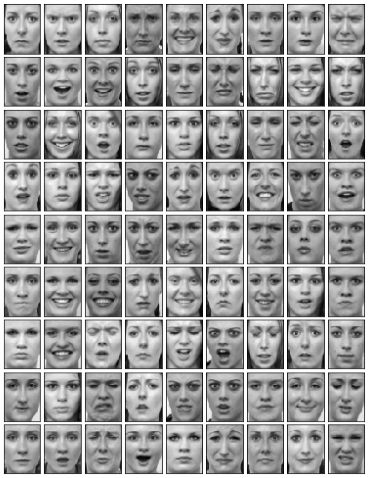


Figure 1: Images from Dataset

2. **Reducing Resolution of Images**
   These images need to reduced in resolution in order to compute faster. Hence we halved the resolution using the method demonstrated in class.

   For the rest of the report the following convention will be used: f_k is the kth face in our dataset, h is the height of our image, w is the width of our image, and $N = h * w$ which is the total number of pixels in our images.

3. **Finding Eigenfaces**
   Next we find the eigenfaces for this dataset using Principal Component Analysis (PCA). Here, PCA computes a linear basis transformation T that transforms the input faces f_k in a way such that the most important information (that results in the most variance when projected onto an axis), is explained by the first dimensions of the resulting $_k$. We call these $_k$ our eigenfaces.

   Intuitively these "eigenfaces" are basis images which can be used to reconstruct any of our original faces f_k from our orignal datbase via a simple linear combination. To reconstruct the original image f_k exactly, we would need all N principal components. However, if we would just like to build an approximation, then the first $n << N$ components suffice as all components after the n^{th} component do not account for much change in the resulting image.

4. **Determining a Suitable Encoder**
   By properties of PCA, we know that for each eigenface, the corresponding eigenvalue represents how much of an influence the eigenface has on the face image. Additionally, after some n^{th} principal component, the rest of the eigenfaces that follow have exponentially less influence on the face image.

Thus we plot the eigenvalues for the corresponding eigenfaces,and choose a suitable cutoff $n = 15$. That is, the encoders for the neural population will be the first 15 eigenfaces, because they can be used to represent an approximation of the input faces very accurately.

5. **Building a Neuron Ensemble and Obtaining Neural Activity**
   By the NEF principles, and since we are only representing an input, our model will be a simple ensemble.

   In our case, there are N pixels we would like to represent in our image, and so it makes sense to use N neurons in our ensemble. Ideally if we wanted a very accurate representation of the input face, we would choose our dimensions to also be N. However, since from PCA, that the first 15 eigenfaces in our encoders have an exponentially larger influence on the face, and that the dimensions after 15 do not mean much, it makes sense to have our neurons to be 15-dimensional. We have already chosen our encoders for the neurons; the first 15 eigenfaces as they are enough to encode a good representation.

   The input to our ensemble will be one flattened face image. We create a nengo Node of this image, and create a connection from the input to the ensemble. Next, we make a probe which will record Neural Activity. This probe apply a synaptic filter with synapse = 0.1 to the neural spike data. This will be a 15 dimensional signal of N neurons, which will be our primary metric for analysis.

6. **Experiment with Morphed Faces to Find Resulting Changes in Neural Activity**
   First, we will begin by finding the neural activity for our original faces using the ensemble we have just created. Now that we have the expected neural activity, we can now play around and morph specific principal components of our faces, and see how it changes the neural activity of our ensemble. This may indicate that certain dimensions of the neurons are responsible for representing specific components. More regarding this in the analysis.

   Next, we will subtract the average face from all of our faces, and record neural activity. Now, we will run all of our experiments on both datasets and see which gives a higher variance or better changes in the neural activity with changes in the input (morphing).

   The simulations for each trial will only be run for $T = 0.1$ because of lack of computational power. However, if equipped with a better machine, these simulations would be run for much longer.

   For the following experiments, we morph both the orignial dataset, and the dataset with the man face removed.

1. **Darken Skin Colour**
   By inspection, it is clear that the first principal component corresponds to skin colour. That is, the colour of skin has the greatest influence on the image (which makes sense intuitively if we were to distinguish between people, its the easiest description).

   Thus we can subtract some more of the first principal component from each face to darken the skin colour. That is $f_k = f_k - cPC_0$ where c is some integer which represents how much we will darken the skin colour by.

2. **Brighten Skin Colour**
   We will brighten the skin colour of the faces by simply adding more of the first principal component to each face. That is $f_k = f_k + cPC_0$ where c is some integer which represents how much we will brighten the skin colour by.

1.2 Implementation

Now we will go through all the major components and see how they are actually implemented.

1. Loading Images and Reducing Resolution

In [1]:
```
import numpy as np
import gzip
import matplotlib.cm as cm
import matplotlib.pyplot as plt
import nengo
from nengo.utils.ensemble import response_curves
import os
import imageio
%matplotlib inline
```

In [2]:
```
def read_image_database(path='faces.npz'):
    images = []
    for file in os.listdir(path):
        images.append(imageio.imread(os.path.join(path,file), as_gray=True))
    mat = np.array(images, dtype=np.uint8)
    return mat
mat = np.load('faces.npz')['arr_0'].astype(np.float64)
def half_resolution(X,rep=1):
    for _ in range(rep):
        r,c = 2*(X.shape[0]//2), 2*(X.shape[1]//2)
        X = 0.25*(X[:r:2, :c:2] + X[1:r:2, 1:c:2] + X[1:r:2, :c:2] + X[:r:2, 1:
c:2])
    return X
images_small = []
for i in range(mat.shape[0]):
    images_small.append(half_resolution(mat[i], 2))
mat = np.array(images_small)
X = (2.0 * mat/255.0-1.0)

N = mat.shape[0]
h = mat.shape[1]
w = mat.shape[2]
```

Figure 2: Loading images and reducing resolution using code from class

The code for this part was used from lecture material.

2. Finding Eigenfaces and Finding Suitable Encoders
   Similar as before, the lecture material was used to find the principal components for this data set.

In [4]:
```
X = X.reshape(N, w*h)
X_zero_mean = X-np.mean(X,axis=0)
X_cov = (X_zero_mean.T @ X_zero_mean)/(X.shape[0]-1)
D,V = np.linalg.eigh(X_cov)
D = D[::-1]
V = V.T[::-1,:]
V = V / np.linalg.norm(V,axis=0)
```

In [6]:
```
NN = int(np.floor(np.sqrt(N)))
cmap = cm.get_cmap('gray')
fig, axs = plt.subplots(NN, NN, figsize=(5.5, 7))
for i in range(NN):
    for j in range(NN):
        axs[i, j].imshow(V[NN * i + j].reshape(h, w), vmin=-0.125, vmax=0.125, cmap=cmap)
        axs[i, j].set_xticks([])
        axs[i, j].set_yticks([])
fig.tight_layout(w_pad=0.0, h_pad=0.0)
```

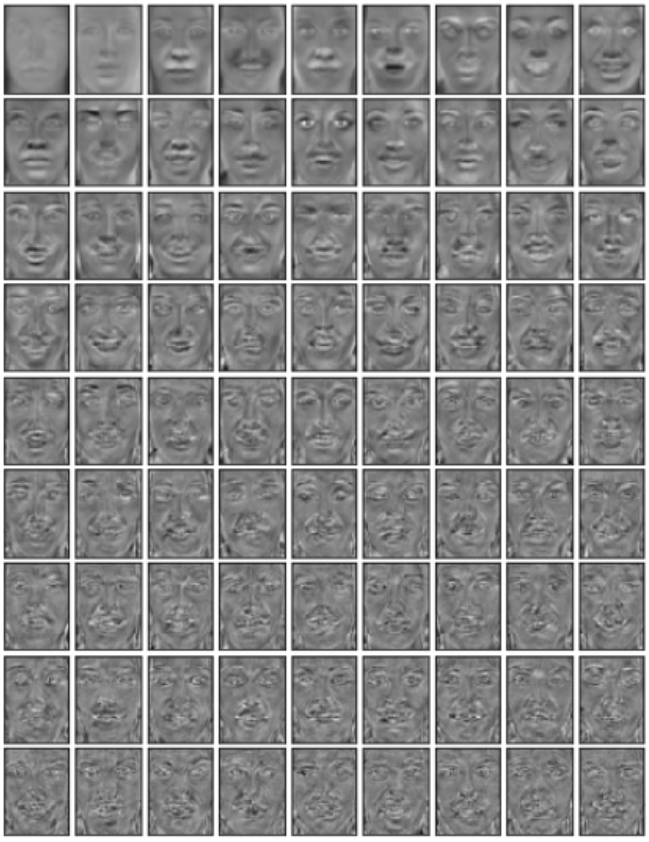


Figure 3: The Principal Components or eigenfaces of our dataset.

Now that we have found the eigenfaces, we need to determine how many of them to keep and use for our encoders. That is, we need to find a cut off n and keep all eigenfaces $e_f < n$ and discard all eigenfaces $e_f > n$. How do we do this? Well as mentioend in section 1.1.4, we can plot the eigenvalues of the corresponding eigenfaces, and determine the cutoff n after which the eigenvalues plateau.

```
In [5]: plt.subplot(1,2,1)
        plt.plot(D)
        plt.xlabel("nth eigenface")
        plt.ylabel("eigenvalue")
        plt.subplots_adjust(right=2)

        plt.subplot(1,2,2)
        plt.plot(D)
        plt.xlim(-1, 60)
        plt.xlabel("nth eigenface")
        plt.ylabel("eigenvalue")
        plt.title("zoomed in")

Out[5]: Text(0.5, 1.0, 'zoomed in')
```

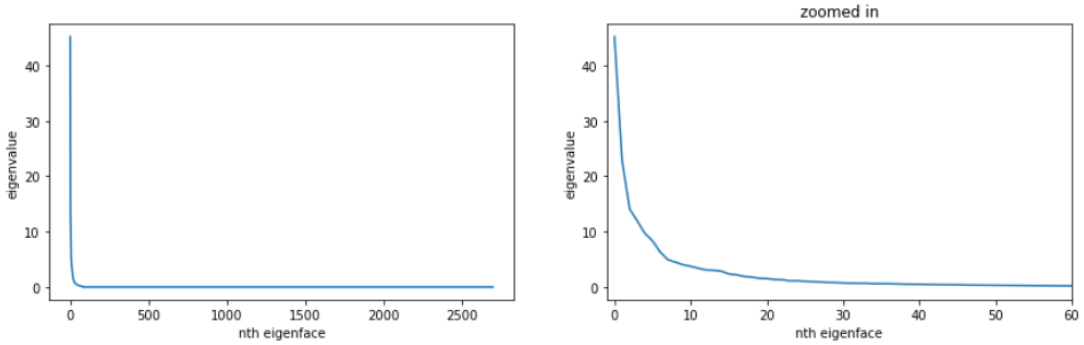


Figure 4: Eigenvalues of the corresponding eigenfaces or principal components.

Looking at the zoomed in graph on the right, we see that selecting only the first 20 eigenfaces or even the first 15 will suffice for our Encoders, as the eigenvalues for any eigenface after the 15th one is very small compared to the prior.

This claim is further reinforced by figure 1.2.2. We can see that all eigenfaces after the 15th one just look like ghostly figures with no indication of a significant feature of a face image they are encoding.

Note that we initially chose the first 15 eigenfaces + random combinations of the first 15 eigenfaces as our encoders. However, due to lack of computational power, we had to discard thhe random combinations.

3. Building a Neuron Population and Obtaining Neural Activity

Here we defined a simple single ensemble neuron population which takes in an image, and outputs neural activity on 15 dimensions.

In [8]:
```
numEigenfaces = 15
encoders = V[:numEigenfaces]
```

In [9]:
```
# want to keep the same ensemble and see how it represents faces
# if we reinitialze the ensemble for each input, then the curves
# are meaningless
def Model(X, numNeu, d, E,synap, T, isPlot):
    lenX = len(X)
    populationCurves = []
    representations = []
    sts = []
    model = nengo.Network(seed=581)
    with model:
        ensA = nengo.Ensemble(n_neurons=numNeu,dimensions=d,encoders=E.T,normali
ze_encoders=False)
    for i in range(len(X)):
        with model:
            inp = nengo.Node(X[i])
            il = nengo.Connection(inp,ensA.neurons)
            inpProbe = nengo.Probe(inp)
            spikes = nengo.Probe(ensA.neurons)
            voltage = nengo.Probe(ensA.neurons, 'voltage')
            filtered = nengo.Probe(ensA,synapse=synap)
        with nengo.Simulator(model, progress_bar=False, optimize=True) as sim:
            if i%10==0: print("running simulation for step:",i)
            sim.run(T)
            # x_axis = 50 x vals
            # y_axis = n_neurons * 50 y values
            responseCurves = response_curves(ensA,sim)
            populationCurves.append(responseCurves)
            representations.append(responseCurves[1])
            sts.append(sim.data[filtered].T)
            if isPlot:
                plt.subplot(12,7,i+1)
                plt.plot(sim.trange(), sim.data[filtered])
    plt.show()
    return populationCurves, representations, sts
```

Figure 5: Neuron Population model used for representation

We have also inserted many probes which would store output at certain points in the process which will later help in our analysis. And the model returns the data recorded at these probes; we will only care about the filtered spike trains in our analysis. That is the probe defined as: filtered = nengo.Probe(ensA,synapse=synap).

4. Experiments

   We first obtained the neural response for the original faces. And for each of the following experiments, everything was kept constant (encoders, ensemble population seed, etc) except for the input. That way we will be analyzing the representation made by

the same population for different input.

```
In [12]: curves1,reps1,st1 = Model(X=X,numNeu=2700,d=15,E=encoders,synap=0.01,T=0.1,isPlot=0)
```

Figure 6: Representation of un-morphed dataset

Next, we show the methods used to remove the mean face and the representation.

```
In [11]: meanFace = np.mean(X,axis=0)
         X1 = [x-meanFace for x in X]
         plotImages(np.array(X1).reshape(84,60,45))
```

Figure 7: Faces with mean face removed

The following function was used to morph the first ith principle component. And the change parameter represented whether we increase or decrease the intensity of this component in our face image. The factor parameter represents how much we modify our original face by.

```
In [12]: # will darken the skin colour by increasing the intensity of the
         # first principal component, which by inspection represnts skin colour.
         def changeComponent(change,i, PCs, originalFaces, factor):
             newFaces = []
             for f in originalFaces.reshape(N,h*w):
                 newFace = f - factor*PCs[i] if change=="darken" else f +factor*PCs[i]
                 newFaces.append(newFace)
             return newFaces
```

Figure 8: changeComponent

2 Results and Discussion

2.1 Methods for Comparison of two Experiments

We first discuss how we will be simplifying our highly dimensional neural responses from each experiment, and what experiments will be compared.

So our neural response from our population are (84,15,5000) dimensional. And it would be very difficult to compare this highly-dimensional data from experiment to experiment and decipher any relationships in the encoding of our neural population. Thus we define the following functions to help us consolidate our filtered spike trains which will help us analyze our results from our experiments better. Once again, our goal is to analyze

the changes in the average neural activity of all images on each of the 15 representation dimensions with changes to the input.

```
# each filtered spike train has shape (84,15,5000)
# so for each of the 84 faces, average the spike activity on each of the 15
# dimensions that the neurons span.
def spikeAve(data):
    dimensionAve = []
    for face in data:
        spikeAves = []
        for i,st in enumerate(face):
            #average neural activity on ith dimension for the face
            spikeAves.append(np.mean(st))
        dimensionAve.append(spikeAves)
    return dimensionAve

# find difference of average neuron spiking activity on each dimension
# that is we find the difference between 2 representations of the face,
# and see where the most change occured and the least change occured.
def getDiff(aves1,aves2):
    diffs = []
    for a1,a2 in zip(aves1,aves2):
        diff = []
        for x,y in zip(a1,a2):
            diff.append(x-y)
        diffs.append(diff)
    return np.transpose(diffs)

# find on which dimension did the representations vary the most by averaging
# activity on each dimension for each experiment, and taking the difference
# at each dimension of representation.
def getChanges(exp1,exp2):
    changes = getDiff(spikeAve(exp1),spikeAve(exp2))
    dims,N = np.shape(changes)
    sumChanges = []
    for d in range(dims):
        sumChange = 0
        for n in range(N):
            sumChange += changes[d][n]
        sumChanges.append(sumChange)
    del sumChanges[0]
    return sumChanges

def plotChanges(exp1,exp2,s1,s2):
    plt.plot(getChanges(exp1,exp2),'.-')
    plt.plot(np.zeros(14))
    plt.xlabel('Dimension of neuron representation')
    plt.ylabel('Spike activity change')
    plt.title('Average difference in representation in '+s1+' and '+s2)
```

Figure 9: Analysis functions

The spikeAve function will take in the output of our experiments (a (84,15,5000) filtered spike train obtrained from a probe), and it will average the spike activity in each of the 15

dimensions. And so spikeAve will take in a matrix in the shape (84,15,5000) and return an (84,15) matrix representing the average neural activity on the 15 dimensions for each of the 84 faces.

The getDiff function will simply find the difference of the what spikeAve would return. That is, getDiff will take in the averaged spike data for two experiments, will return the average change on each of the 15 dimensions for all 84 faces before and after morphing. Input shape (84,15), return shape (15).

The getChanges method will now simply wrap the two functions above, and return a 15 dimensional array representing the change in neural activity of representation of the experiment 1 and experiment 2.

Plot changes will plot these changes.

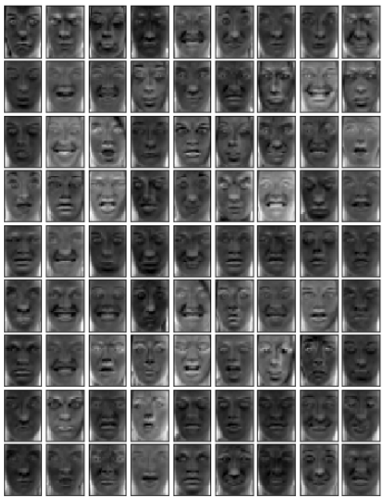
2.2 Visualization of Morphed Faces (Inputs)

Now we will show the input data for each experiment, and the analytics of the neural response for the datasets. There were a total of 6 datasets we represented in our experiments:

(a) Original faces

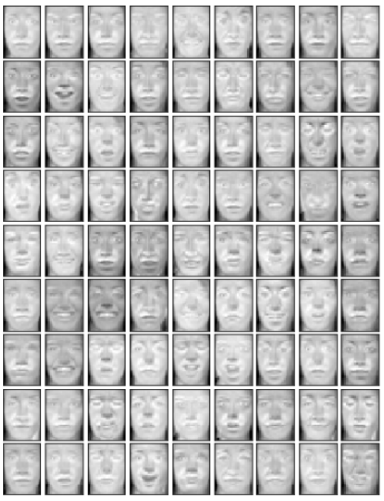
(b) Mean faces removed

(c) Darkened skin of original faces



(d) Brightened skin of original faces

(c) Darkened skin of mean faces removed dataset



(d) Brightened skin of mean faces removed datase

Figure 10: Inputs

2.3 Results and Discussion

2.3.1 First lets talk about how we will be analyzing our results:

Lets restate our goal here; we wish to compare neural activity on each of the 15 dimensions of representations, before and after morphing the faces. Thus we compare the results from the following to find a relationship in neural response to our morphing:

1. Original Dataset vs Mean Face Removed Dataset
2. Original Dataset vs Darkened Original Dataset
3. Original Dataset vs Brightened Original Dataset
4. Mean Face Removed Dataset vs Darkened Mean Face Removed Dataset
5. Mean Face Removed Dataset vs Brightened Mean Face Removed Dataset

The reason for removing the mean face from the dataset was to have our neural population represent more distinctive features from faces, and also to support any of the findings we make by comparing the results from orignal and morphed, with mean face removed and morphed. And so if both neural differences of the original vs morphed and mean face removed vs morphed have proportionally similar, then we know that there may be a relationship between the principal component that was changed, and the dimension which changed.

Now lets address what type of things we look for in our plots to make reasonable conclusions.

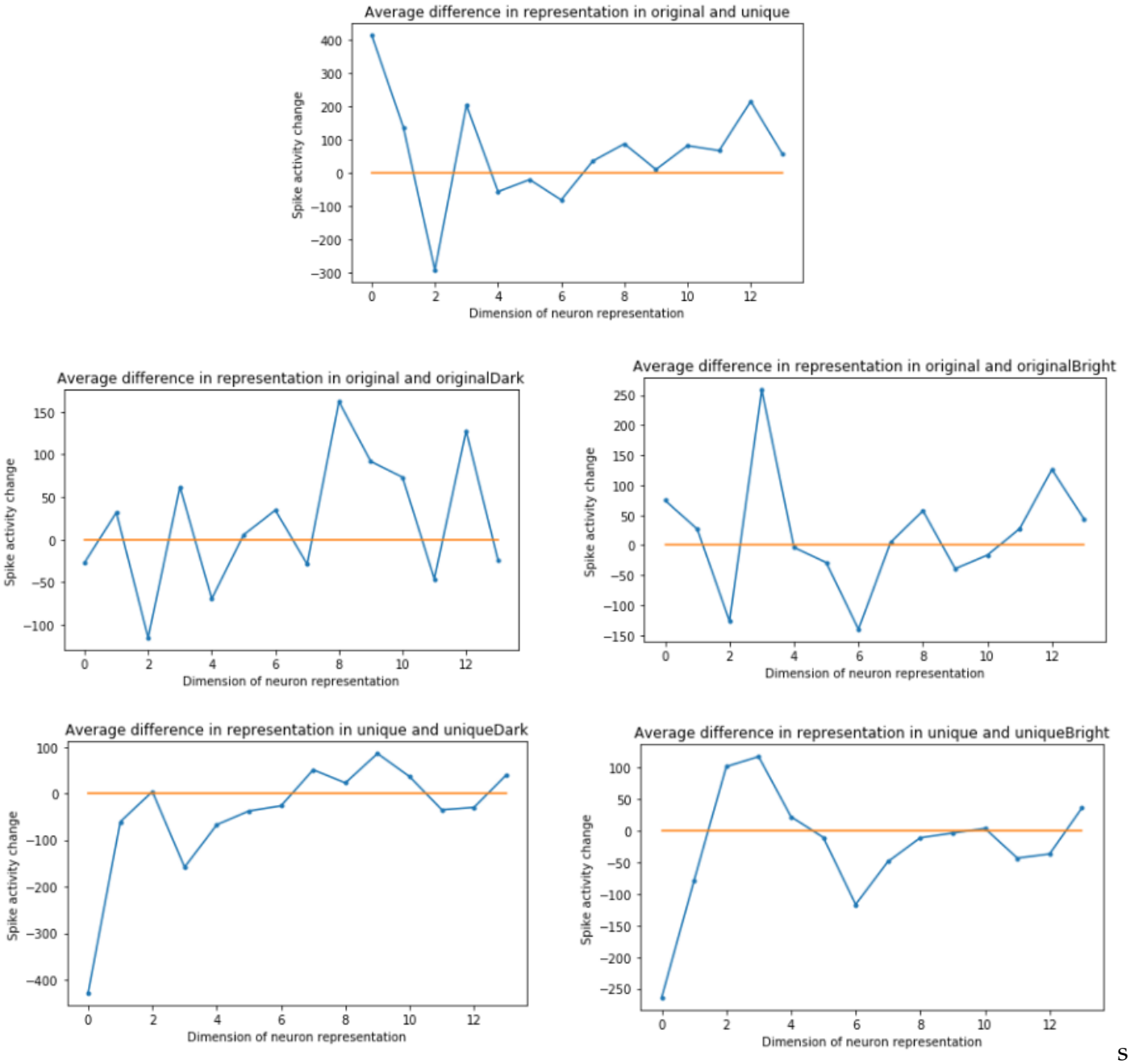
First, if for some dimension d (where d is in [1,15]), there was no change in the average neural activity before and after morphing the face inputs, then this might mean that the principal component that was morphed is not being represented in dimension d since there was no change in neural activity given a change in input.

Now if for some other dimension d' (where d' is also in [1,15]) there was significant change in the average neural activity before and after morphing the face inputs, then this might mean that the principal component that was morphed is being represented by this dimension d since there was large change in neural activity given a change in input.

And if we find one of the two behaviours mentioned above in one of our original vs morphed original experiments, then we compare and see if the same behaviours exist in mean face removed vs morphed mean face original experiments. If we do, then we

have strong evidence that in our neural population, the dimension with/without change isn't/is representing the principal component that was morphed.

2.3.2 Result Plots



s

Figure 11: Results

2.3.3 Discussion:

For the discussion, the experiment with the mean face removed will be referred to as unique. The experiment with the original dataset will be referred to as original. And note that we cannot conclude powerful mappings from change in input to a change of neural activity in dimension x until we check all of our experiments and see if there are is strong enough evidence.

For original vs unique: We see that overall there is significant amount of change in the neural activity in all dimensions. And there is the most difference at the first few dimensions. This tells us that the average or facial features common to all 84 faces are represented in the first few dimensions of the neural representation. And this makes sense because our neural encoders, are principal components sorted from most influence to least influence. And hence, the encoding difference with the mean face removed shows divergence from the original representation in the first few dimensions. There is almost no difference in dimension 7 and 9. This indicates that dimension 7 and 9 might encode for something that's unique to each face (perhaps expression, see figure 1.1.1).

Original vs Darkened Original: Here we see that dimensions of representation 1, 5, 6 and 7 have to little change in neural activity after darkening skin colour of the faces from the original dataset. This indicates that the neurons in our ensemble do not represent skin colour on dimension 1, 5, 6 or 7. Dimension 8 and 12 have the largest difference. This indicates that dimensions 8 and 12 encode for skin colour in our neural model, since the neural activity for our ensemble changed the most after a systematic change in our input.

Unique vs Darkened Unique: Here we see that dimensions of representation 5, 6, and 12 have close to zero change in neural activity after darkening skin colour from the unique dataset. This might indicate that the neurons in our ensemble do not represent skin colour on dimensions 5, 6, and 12. Dimensions 0, 3, and 9 have the largest difference. This indicates that dimensions 0, 3 and 9 encode for skin colour in our neural model, since the neural activity for our ensemble changed the most after a change in our input.

Original vs Brightened Original: Here we see that dimensions of representation 1,4,5,10 and 11 have close to zero change in neural activity after brightening skin colour from the original dataset. This might indicate that the neurons in our ensemble do not represent skin colour on these dimensions. Dimensions 2,3,6, and 12 have the largest difference. This indicates that these dimensions encode for skin colour in our neural model, since the neural activity for our ensemble changed the most after a change in our input.

Unique vs Brightened Unique: Here we see that dimensions of representation 4,5,8,9, and

10 have close to zero change in neural activity after brightening skin colour from the unique dataset. This might indicate that the neurons in our ensemble do not represent skin colour on these dimensions. Dimensions 0,2,3, and 6 have the largest difference. This indicates that these dimensions encode for skin colour in our neural model, since the neural activity for our ensemble changed the most after a change in our input.

3 Discussion and Conclusion

From our results, we would like to conclude which dimensions might and might not represent skin colour (our first principle component or eigenface).

In all four of our experiments, we saw that in dimension 5 had very little change in neural activity after morphing our first principal component. Therefore, we can conclude with strong evidence that the neurons in our population do not model skin colour on the 5th dimension of input representation. Dimensions 1, 4 and 10 had positive correlations in two experiments, and so there might be some encoding of principal component 0 in these dimensions in our neural population but we are not sure. There is not strong enough evidence.

In 3 experiments, dimension 3 had significant changes in neural activity after morphing our first principal component. Therefore we can conclude with good evidence that the neurons in our population model skin colour on the third dimension of representation. Dimension 2 appeared to have significant changes in neural activity in two experiments, and so there might be some encoding of skin colour in this dimension, but we are not sure.

In two experiments dimension 6 had large changes in neural activity and in two other experiments, it had very little change in neural activity. Thus, there is no explanation to whats being encoded onto this dimension by our neural population. If we had to make an estimate, it would be something that is 50% correlated with principal component 1. That may explain why half of the time, it has large changes and half of the time it has no changes.

Thus, from our experiments, we can conclude with strong evidence that information regarding principal component 1 is not encoded in the 5th dimension of representation of our neural population. And we can conclude with good evidence that this information is encoded in dimension 3. The disparity in the results amoung the 4 experiments is caused by the limiting design choices we had to make due to lack of computational power.

In conclusion, we have learned more about the neural code for facial representation, by comparing neural representation of faces before and after morphing the skin colour. Our results do not give strong evidence there is a such an encoding of facial representation in real brains, or the exact encoding of faces in neurons, but it did tell us that certain facial information may be encoded in certain dimensions of neural representation.

4 Future

Now how can we improve our results to make stronger claims? Well due to lack of computational power, the complexity of the model that we used was very limited. With a more powerful machine, we would have represented the faces on a higher dimensional output, we would have used more neurons in our model, ran the simulation for longer, used higher resolution images and a larger dataset.

In the future, we would like to find a better machine, and run this experiment with the changes mentioned above and see if our claims in this paper still hold.

References

Chang, Le and Doris Y. Tsao (2017). *The Code for Facial Identity in theh Primate Brain*. California: https://doi.org/10.1016/j.cell.2017.05.011. 31 pp.