Tappoint Placement in Braindrop

Jeffrey B. Wang

October 31, 2016

1 Introduction

1.1 NEF

In NEF, our objective is to represent an input signal $\mathbf{x}$ as the weighted average of the firing rates of a number of neurons, which can be computed from the input as

$$a_i = G_i(\alpha_i \mathbf{e}_i^T \mathbf{x} + J_i^{\text{bias}}), \tag{1}$$

where a_i is the firing rate of the i^{th} neuron, G_i is the tuning curve of the neuron (i.e. the nonlinearity), α_i is the gain, $\mathbf{e}_i^T$ is the encoding vector, and J_i^{bias} is the bias term of the neuron. Intuitively, the encoding vector represents the preferred direction of the neuron, where inputs that are along this preferred direction will tend to fire more. For NEF, one of the expectations is that the encoding vectors are randomly chosen from the surface of the unit hypersphere, and that the x-intercepts of tuning curves after the input is projected onto the encoding vector are uniformly distributed on $[-1, 1]$. Then, the number of neurons needed to represent a certain signal with a given fidelity scales roughly linearly with the dimensionality of the input. The uniform distribution of encoding vectors however is critical for this assumption to hold, which makes the choice of encoding vectors when designing hardware implementations of NEF quite critical.

1.2 Braindrop Implementation of Encoding Vectors

As an effort to reduce the number of actual connections that have be made to transmit a single spike to all of the soma on the chip, Braindrop utilizes a shared-synapse architecture for carrying the input signal, which we describe in Figure 1. Each soma is connected via a hexagonal resistive network with conductance to ground of g and conductance to a neighbor of h. Then for every four somas there is one tap point, which can be assigned to one direction (i.e. sign) and one dimension of input. These tap points are highlighted as part of the green boxes.

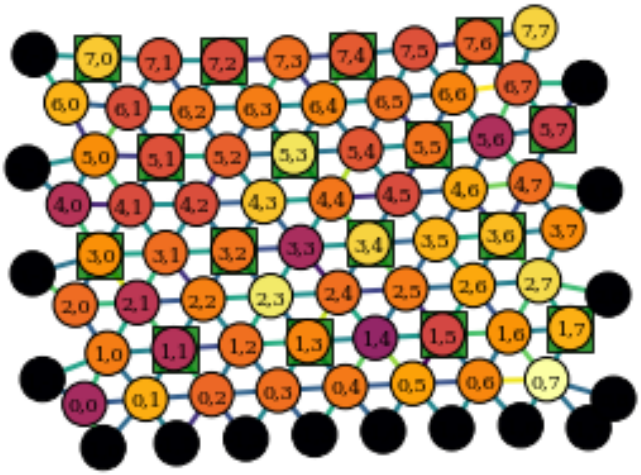

Figure 1: Shared synapse setup used in Braindrop. Each soma, represented as a circle, is assigned a unique ID and connected to up to six neighbors with a conductivity h, and are connected to ground with a conductance g. The input is carried only to the synapses, which are represented by the green squares, and then diffused across the resistive network.

The tap points then transmit a current I_{in} based on the input $\mathbf{x}$. We then transmit the current to ground I_g as the input into the soma for the neuron, which we can derive from

Kirchoff's Current Law as

$$I_g = gV_i = I_{\text{in}} - h\sum_{j\in\mathcal{N}}(V_i - V_j), \tag{2}$$

where V_i is the voltage level of the i^{th} neuron, and $\mathcal{N}$ is the neighborhood of the neuron. For computing the effective encoding vector for each neuron, we inject a test current of ± 1 (depending on the assigned direction to the tap point) at the tap points for a single dimension d, compute the response on the diffusor, and then load the result into the d^{th} element of the encoding vector.

1.3 Goal

To choose tap points on the diffusor to get an "optimal" set of encoding vectors for signal representation. Here, we assume that an optimal set of encoding vectors to be a set of vectors that span the entire hypersphere of dimensions.

2 Methods

2.1 Diffusor Properties

We can either have the number of neurons (and thus tap points) scale linearly with respect to the number of dimensions N_d (i.e. $64N_d$), or we can have it scale quadratically ($64N_d^2$). We will then fix the percentage of tap points to use (using more tap points introduces more traffic on the chip).

2.2 Evaluating Performance

We utilize a number of different metrics to gauge the performance of our strategy:

- Pick 2 dimensions, project all vectors to its 2D plane, then compute the angle of the encoding vector. Sort the angles into 100 bins from 0 to 2π (with some optional smoothing). Repeat for all combinations of 2 dimensions. For a perfectly distributed set of encoding vectors across the hypersphere, we expect unit circles for all combinations.
- Pick a random unit vector on the hypersphere and find the nearest normalized encoding vector. Compute the angle between them. Repeat 5000 times and collect statistics to determine the total coverage of the hypersphere. We wish to minimize this metric

2.3 General Strategy

Because of mismatch in the conductivity between individual somas (along with mismatch in the actual current response of the soma itself), it is probably too complex to try to take into consideration all of the features of the diffusor in a single optimization step. Therefore, we believe a piecewise strategy would be more effective:

1. Come up with a geometric placement strategy that disregards mismatch. In this case, we ignore the magnitude and try to have the direction of each encoding vector to span the unit hypersphere
2. Utilize individual gain bits to ensure that neurons will have roughly identical average firing rates (i.e. the encoding vectors are all unit length)
3. Utilize bias bits to distribute the x-intercepts of the tuning curves equally across [-1,1]

2.4 Useful Heuristics

When placing the tap points, with some experimentation, we believe that these heuristics help us achieve the goal of having uniformly distributed encoding vectors:

1. Positive and negative tap points of the same dimension should be as far as possible. This prevents positive and negative tap points from interfering with each other and reducing our SNR since the cancellation is not perfect due to mismatch
2. Tap Points for a single dimension and direction should be equally distributed across 50% of the diffusor if possible. This is to maximize the area that the tap points are covering for a lot of mixing
3. Ideally, we should be able to have areas dedicated to every possible combination of sign and dimension (e.g. for 2D: ++, +-, -+, –). Again, this lets us have as much mixing as possible. Note that this seems like a difficult task; we have quadratically increasing number of neurons at most, but there's an exponential number of combinations with respect to the number of dimensions!

2.5 Strategy 1: Randomized Tap Point Placement

For a baseline, we simply randomly assign tap points to dimensions and signs, equally dividing the available tap points equally amongst the dimensions and signs. This is visualized in Figure 2

2.6 Strategy 2: Pairwise Assignment

The intuition behind is that we wish to get every possible pair of dimension and sign close to each other, so that we can effectively have good mixing between two given dimensions. For this method, we are fixed to the number of neurons $N = 32N_D(N_D - 1)$, where N_D is the

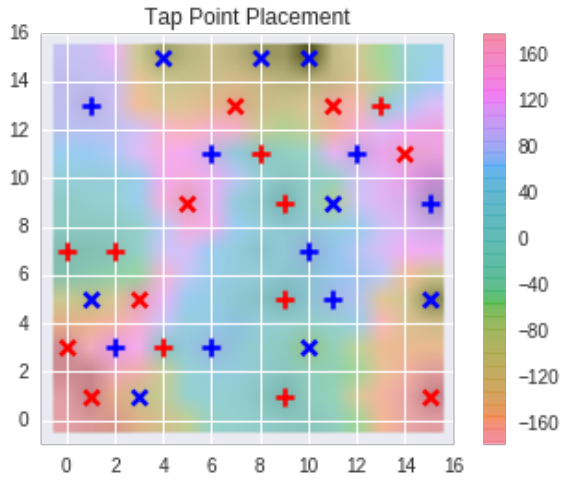

Figure 2: Example placement of randomized tap point placement for a 256 neuron pool for 4 dimensions. The red and blue represent placement of the first and second dimension, and the positive means that we place a positive (excitatory) tap point and the x indicates placement of a negative (inhibitory) tap point. The color indicates the angle of the encoding vector in the xy-plane.

number of dimensions and N is the number of neurons, which we can then divide amongst the $4N_D^2$ possible pairings of dimensions and sign. To realize this, we assign each dimension and sign to both a row and column of two neurons in our diffusor and then populate 50% of the tap points within that row and column. That way, each pairing would have a 2×2 small square of tap points (and thus 4×4 square of neurons) that are dedicated towards mixing that pairing. This is visualized in Figure 3.

2.7 Strategy 3: H-Tree Assignment

The strategy for this method is inspired by the H-Tree, which lets us draw out a binary tree in as compact of a space as possible, as shown in Figure 4

Each branch of the H-Tree can be considered a single sign for a dimension (e.g. going left is negative for the first dimension and going right is positive. Consider N_T to be the number of tap points we are willing to commit per dimension, per sign. H-Tree works by assigning general columns of where the positives and negatives for a single dimension should go. Then,

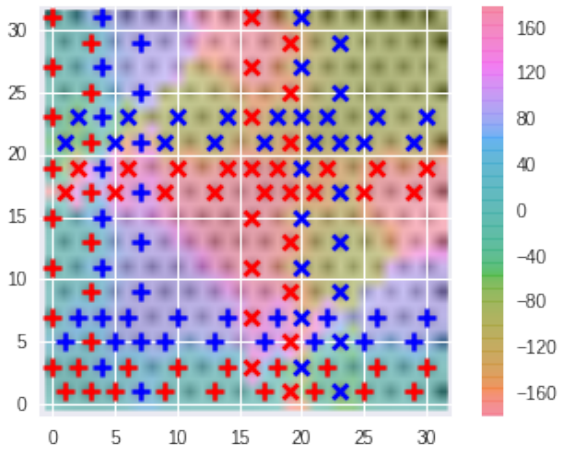

Figure 3: Example placement of pairwise tap point placement for a 256 neuron pool for 4 dimensions. The red and blue represent placement of the first and second dimension, and the positive means that we place a positive (excitatory) tap point and the x indicates placement of a negative (inhibitory) tap point. The color indicates the angle of the encoding vector in the xy-plane. Here, we see that we assign a single row and column to each dimension and sign, and we can see that the intersection of these rows and columns is where mixing can occur between the two dimensions.

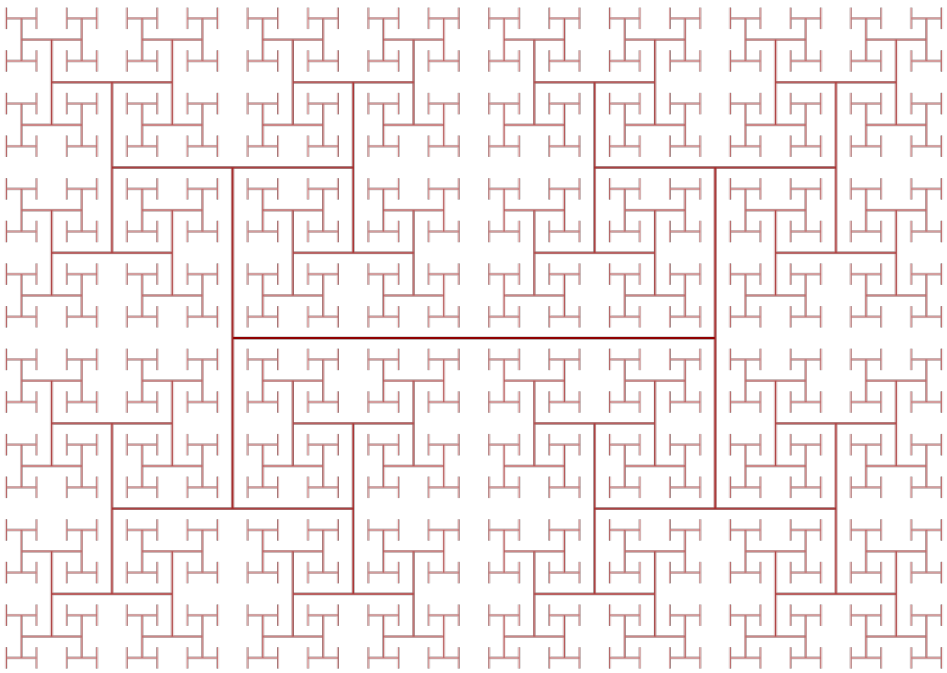

Figure 4: Example of an H-Tree.

we distribute the N_T points uniformly amongst the columns.

2.7.1 Concrete Example

For the first dimension, this is trivial; we just put all of the positive tap points on one side and then all of the negative tap points on the other side.

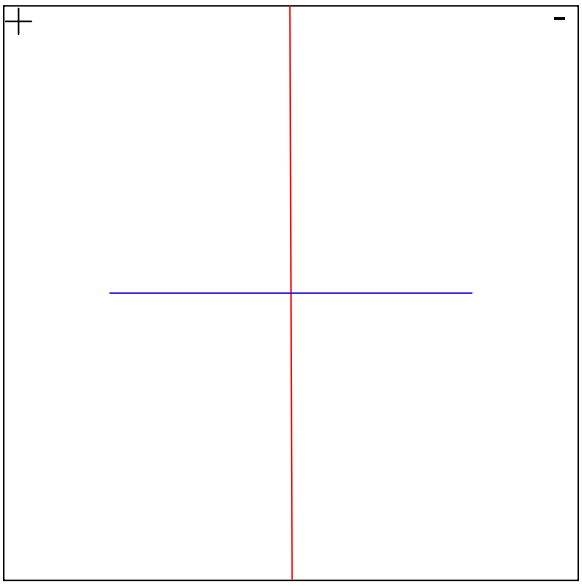

Figure 5: First dimension of H-Tree placement. There are only two columns; the left one is assigned to excitatory tap points and the right one is assigned to the inhibitory ones. The red lines are the divisons between the columns and the blue one is H-Tree.

Now, if we want to enter a second dimension, then we realize that if we make rows perpendicular to the first dimension, like so:

Overlaying the two will make us realize that we get all 4 possible combinations of signs for each dimension. For the third dimension, things get a bit more complicated. Although the H-Tree continuation would make it appear that we have to create four columns, we can actually do the same in three, but merging the two negative signs together so that we are left with only three columns: two positive columns that each take $\frac{1}{4}$ of the area and one negative column that takes $\frac{1}{2}$ of the area.

If you overlay these grids, then you will realize that we still manage to cover all 8 possible

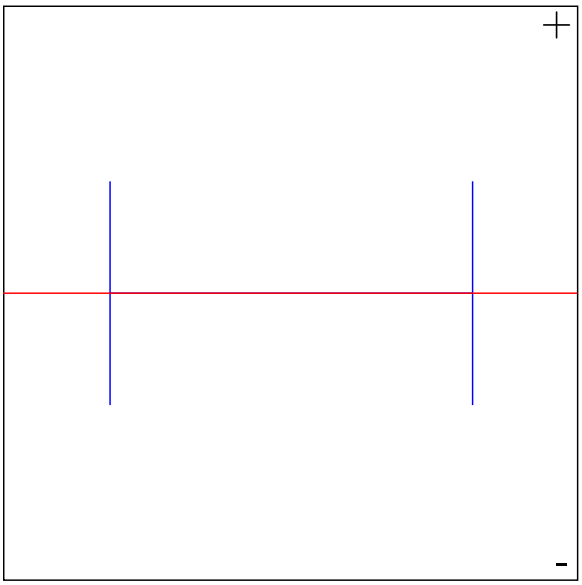

Figure 6: Second dimension of H-Tree placement. We do the same thing, except flipping the directionality

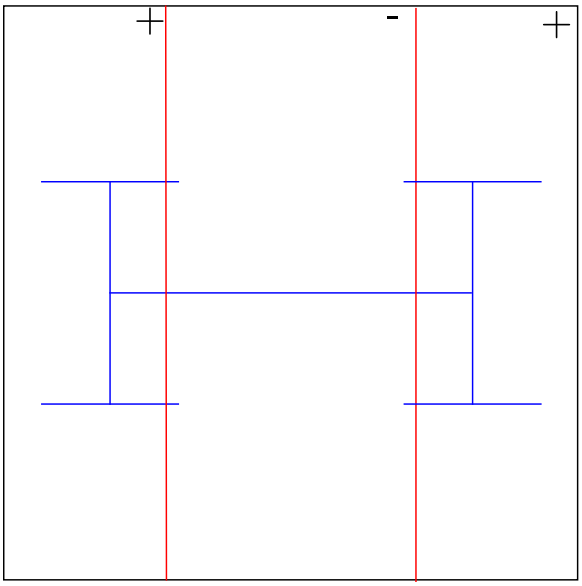

Figure 7: Third dimension of H-Tree placement.

combinations of dimensions and signs. Then for 4D We go ahead and just do the same thing, this time with rows:

Now for 5 dimensions, we go ahead and split this into a total of 5 columns: 3 of them take up $\frac{1}{4}$ of the area and outside two columns take up only $\frac{1}{8}$ of the area. Again, overlaying the

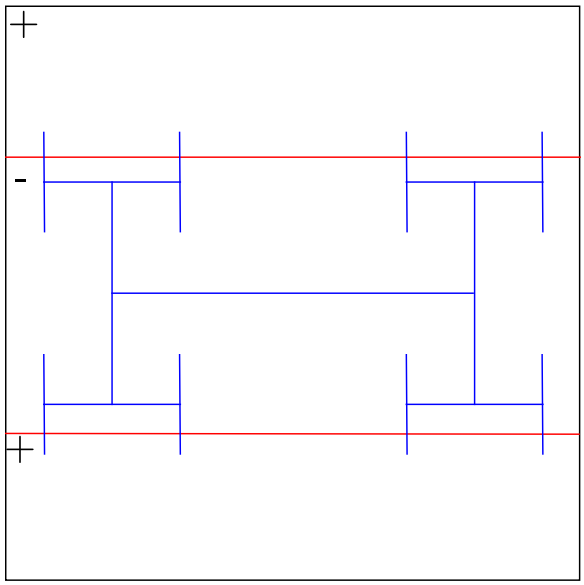

Figure 8: Fourth dimension of H-Tree placement.

columns will show we get all 32 combinations of signs!

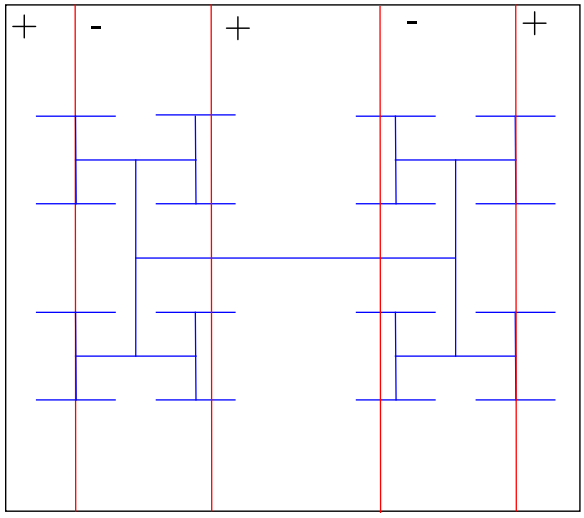

Figure 9: Fifth dimension of H-Tree placement.

Then, for the sixth dimension, we switch over to the rows, and continue to subdivide.

2.7.2 General Algorithm

The algorithm is described in Algorithm 1.

Algorithm 1: H-Tree Algorithm

Data: Number of dimensions N_D

Result: Placement of Tap Points for each sign and dimension

for $d \in \{0, 1, \cdots, N_D - 1\}$ **do**
  if $d \bmod 2 = 0$ **then**
    Divide diffusor into $2^{\lfloor d/2 \rfloor + 1}$ columns;
  else
    Divide diffusor into $2^{\lfloor d/2 \rfloor + 1}$ rows;
  end
  Assign the first strip to the positive sign of the d^{th} dimension;
  while *There are* > 1 *strip remaining* **do**
    Assign the next two strips to a sign of the d^{th} dimension, alternating between negative and positive.
  end
  Assign the last single strip to be positive;
  Distribute the N_D/d tap points to the strips proportionally to the area of each strip;
  Arrange the tap points within each strip to be spaced out maximally (i.e. according to a packing solution with floating point precision);
end
for *All tap points* **do**
  Pick a random desired location according to the packing solution, and assign it to the closest available tap point
end

2.7.3 Analysis

As can be seen, this method will cover all of the hyperquadrants of the hypersphere, and tries to minimize the actual number of subdivisions we have to make by following 2 general strategies:

1. By flipping between columns and rows, we only have to increase the number of strips every other dimension
2. By merging together different arms of the H-Tree, we cut the number of strips needed by about half

However, the number of strips will scale exponentially with respect to the number of dimensions:

$$2^{\lfloor (N_D - 1)/2 \rfloor} + 1 \tag{3}$$

Thus, eventually this method will fail when the number of neurons along one dimension is less than the number of strips needed for the dimension.

3 Results

3.1 Random Tap Point Placement

Figure 10 is a histogram of the angle of the encoding vector when projected into each of the two dimensions. As can be seen, we do not have particularly good coverage of all of the possible angles according to these plots, but this serves as a useful benchmark.

3.2 Pairwise Matching

As can be seen in Figure 11, with this pairwise matching strategy, we do not have effective mixing because the tap points for each dimension is packed along a tight column, which is not very conducive for interacting with all of the other dimensions even though we cover every pairwise interaction between the dimensions. As a result, when we project onto the 2D surface, all of the neurons that are not directly within the row or column will not receive a lot of stimulation, causing the very sharp crosses, meaning we get even less coverage than random.

3.3 H-Tree

As seen in Figure 12, we can get more uniform coverage in each of the directions. However, because of the jagged nature of these plots, there is a need for a more quantitative method for comparison, which we pursue in the next section.

3.4 Profiling

Solving Equation 2 reveals that the behavior is only dependent on the ratio $\frac{g}{h}$. This can be demonstrated in Figure 13, where if we sweep over g or $h = \frac{1}{g}$ we get the same performance of the H-Tree and pairwise matching. Now we know that we only have to sweep over g while holding h constant. To compare the different methods, we utilize a measure where we take a random vector on the unit hypersphere, and measure the angle between that and the nearest encoding vector, averaging over many runs. We demonstrate the performance for four dimensions, utilizing 100% of the tap points in Figure 14. As can be seen, H-Tree performs a bit better than the random method, while pairwise assignment has both higher variance and lower performance. The magnitudes are quite similar, but the median for the H-Tree method is much closer to the 10th percentile, due to our bimodal distribution, which

means that assigning gains with the H-Tree method will be easier since there is a tighter distribution of magnitudes. Part of the reason that random does quite well is because we're using 100% of the tap points. As you reduce the number of tap points used to 50%, then H-Tree performs about 20% better than the random method.

We repeat the analysis for 8 Dimensions, and the performance difference is more accentuated, as shown in Figure 15.

In Figure 16, we plot the performance of the H-Tree method vs the random method, for both scaling the number of neurons linearly with the number of dimensions or quadratically. Note that random only does marginally worse than H-Tree in this case because we are using 100% of the tap points. When we reduce the number of available of tap points, H-Tree does substantially better. As can be seen, for 2D-4D, we can get away with utilizing about $64N_D$ neurons, but for anything more we need $64N_D^2$ neurons

3.5 Tuning Parameters

H-Tree Uses 3 Parameters:

1. Number of Neurons
2. Fraction of Tap Points used
3. Ratio of g/h

We already discussed (1). We are going to now discuss (2). In Figure 17, we plot the minimum percentage of tap points we need to use in order to achive within 5% performance of using all of the tap points.

Because we want all of the neurons to receive the same average firing rate for consistency, we should just fix a single percentage of tap points to use, which appears to be 70

Now we want to go ahead and optimize the spread of the diffusor, which is primarily con-

trolled by the conductance of each soma to ground: g. Here, we take various dimensions and plot the optimal length constant of decay (which we derive from Feinstein's Technical Report[1]

$$L = -\frac{1}{\log r} \tag{4}$$

$$r = \frac{1}{4}\left(4 + g + \sqrt{g(g+8)}\right) \tag{5}$$

The results of this sweep is found in Figure 18. Interestingly, as we increase the number of dimensions, we seem to want a longer length parameter, which at first glance seems to be counterintuitive, because we have more strips and thus each strip should on average become smaller. To try to make some sense of this, we now plot the same thing in Figure 19, except on the x-axis we now put the decay that will be measured at one end of a strip if a test current is applied at the other end. This allows to get a consistent tracking behavior that suggests that there might be some universal relationship that exists.

4 Conclusions

4.1 Recap of Strategy

1. For 2D-4D, utilize $8D$ Neurons, or else use $8D^2$ Neurons
2. Use H-Tree Algorithm to place tap points, utilizing 70% of tap points
3. Compute the average decay from the end to end of a strip, and use that to derive the optimal length parameter to set the conductivity of the diffusor

[1] `https://core.ac.uk/download/pdf/4891510.pdf`

4.2 Future Directions (in relative importance)

1. Get some better theory for understanding the interaction between two individual tap points when they are moved closer to each other. Perhaps that can give us a more solid theory as to what the optimal spread should be
2. The problem with H-Tree is that some dimensions have a very low spatial frequency and are thus at a higher advantage because there isn't as much interferance. Therefore, one potential solution would be to split the diffusor into 2: on one side the H-Tree is arranged dimensions 1-N_d and on the other side it's arranged N_d to 1. That way on average each dimension is treated fairly
3. Adjust H-Tree algorithm to utilize hexagonal grid rather than rectangular divisions. We can probably get away without doing this, because the hexagonal network is designed in such a way that it matches a rectangle pretty well
4. Inform the optimal number of gain/bias bits
5. Customize set of encoding vectors with respect to each function. Effectively, given a set of encoding vectors, can we create a set of tap points that are optimized for this

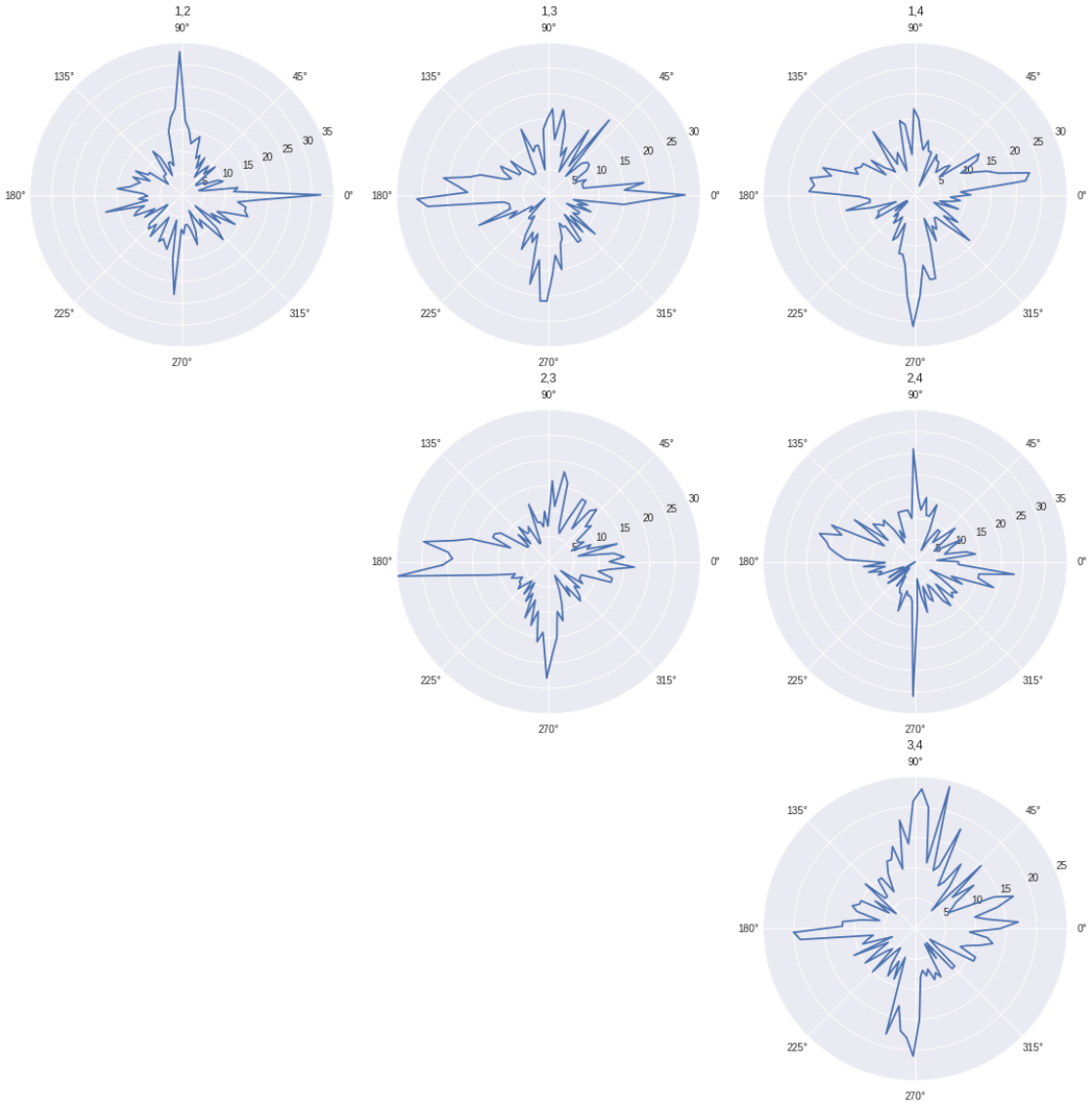

Figure 10: Histogram of angle of encoding vector when projected into each possible pair of two dimensions when using random tap point assignments.

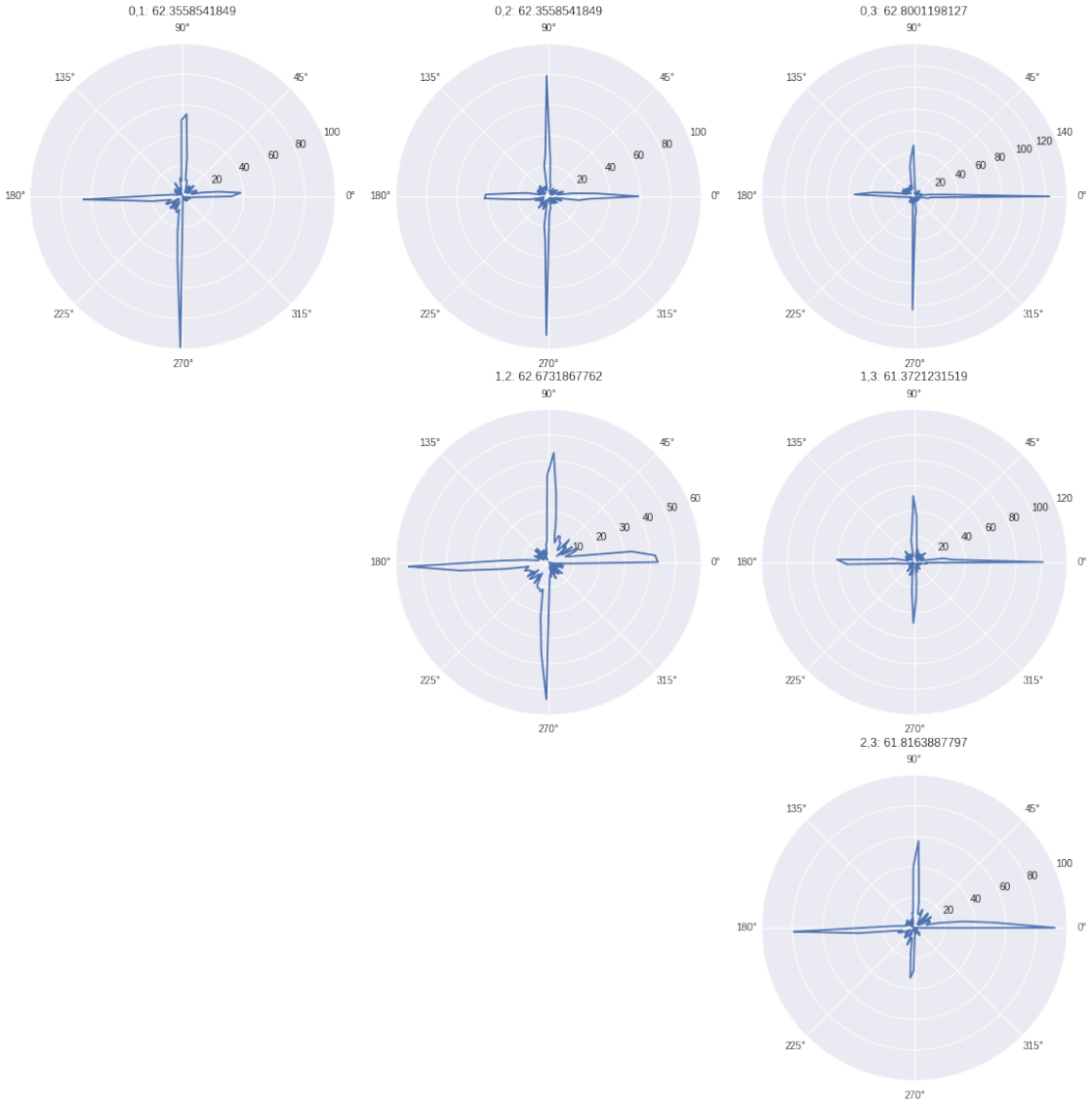

Figure 11: Histogram of angle of encoding vector when projected into each possible pair of two dimensions when using pairwise tap point assignments.

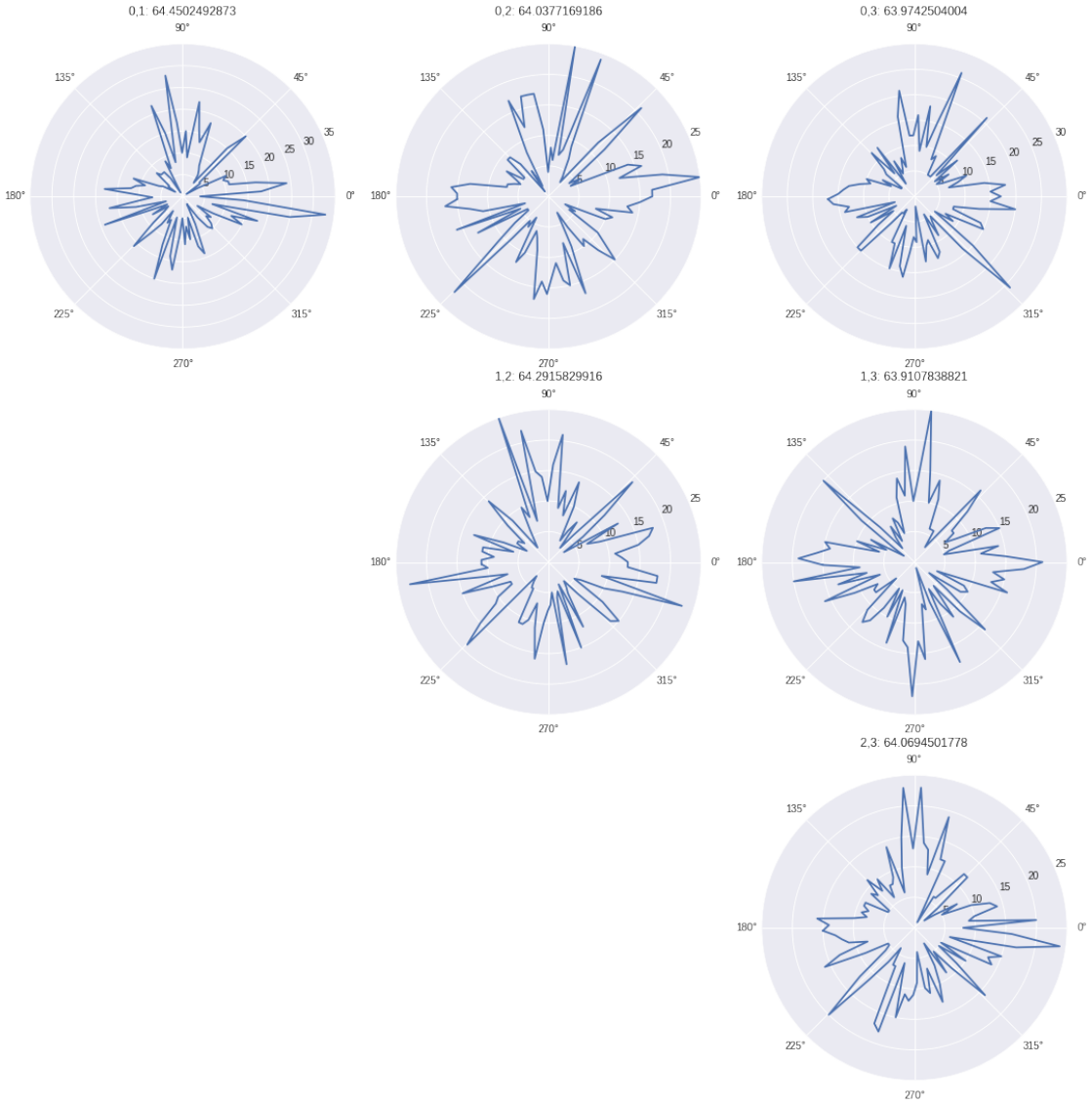

Figure 12: Histogram of angle of encoding vector when projected into each possible pair of two dimensions when using H-Tree tap point assignments.

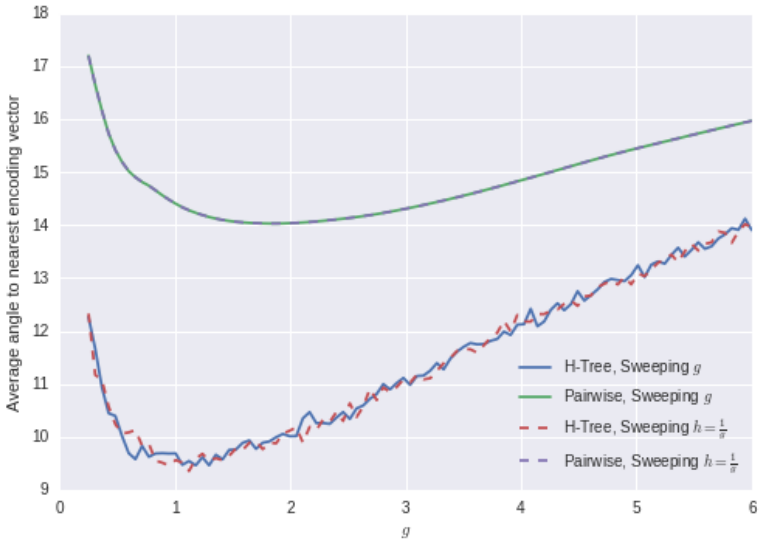

Figure 13: Sweeping over g and h for the H-Tree and Pairwise matching. Solid lines representing sweeping over g while holding $h = 1$, while dotted line is sweeping over the same value but instead setting $h = \frac{1}{g}$

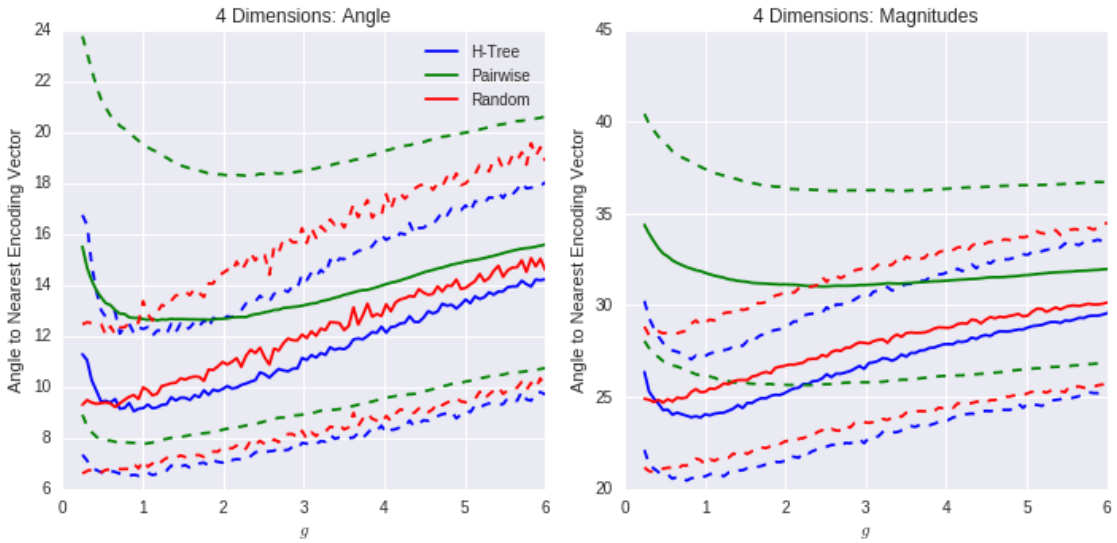

Figure 14: Performance of methods for 4 dimensions, utilizing all tap points. **LEFT**: We plot the coverage of the hypersphere utilizing the angle metric described above (lower is better). Dotted Lines are the 25th and 75th percentile for all of the randomly sampled points for a single trial and tap point assignment. Solid lines is the 50th percentile, while different colors represent different methods. **RIGHT**: Distribution of magnitudes. Dotted Lines are 90th and 10th percentile respectively.

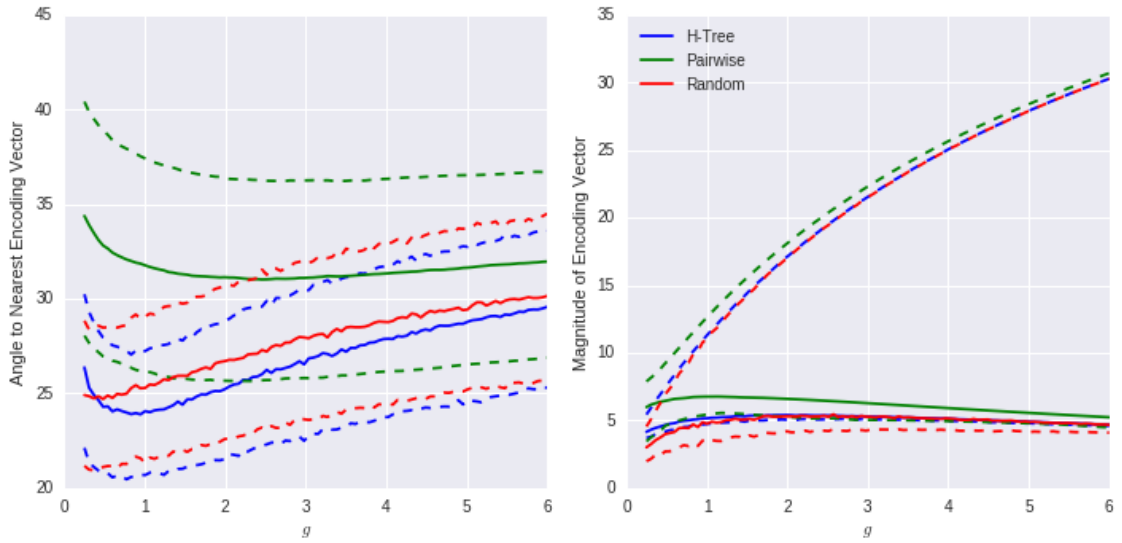

Figure 15: Performance of methods for 4 dimensions, utilizing all tap points. **LEFT**: We plot the coverage of the hypersphere utilizing the angle metric described above (lower is better). Dotted Lines are the 25th and 75th percentile for all of the randomly sampled points for a single trial and tap point assignment. Solid lines is the 50th percentile, while different colors represent different methods. **RIGHT**: Distribution of magnitudes. Dotted Lines are 90th and 10th percentile respectively.

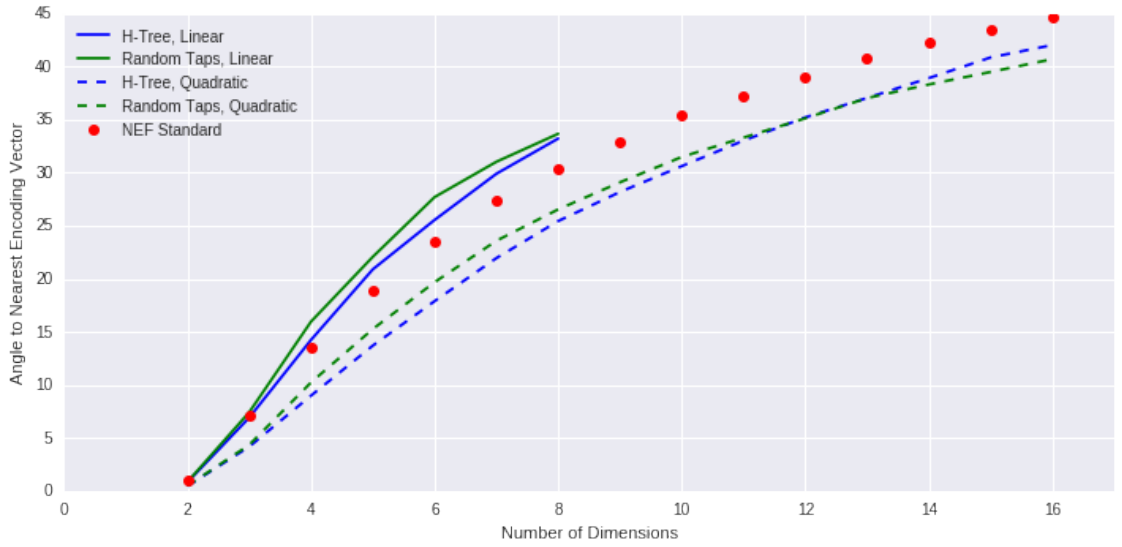

Figure 16: Comparison of random and H-Tree tap point assignment for linear and quadratic number of neurons against the expectations of NEF with randomized encoding vectors.

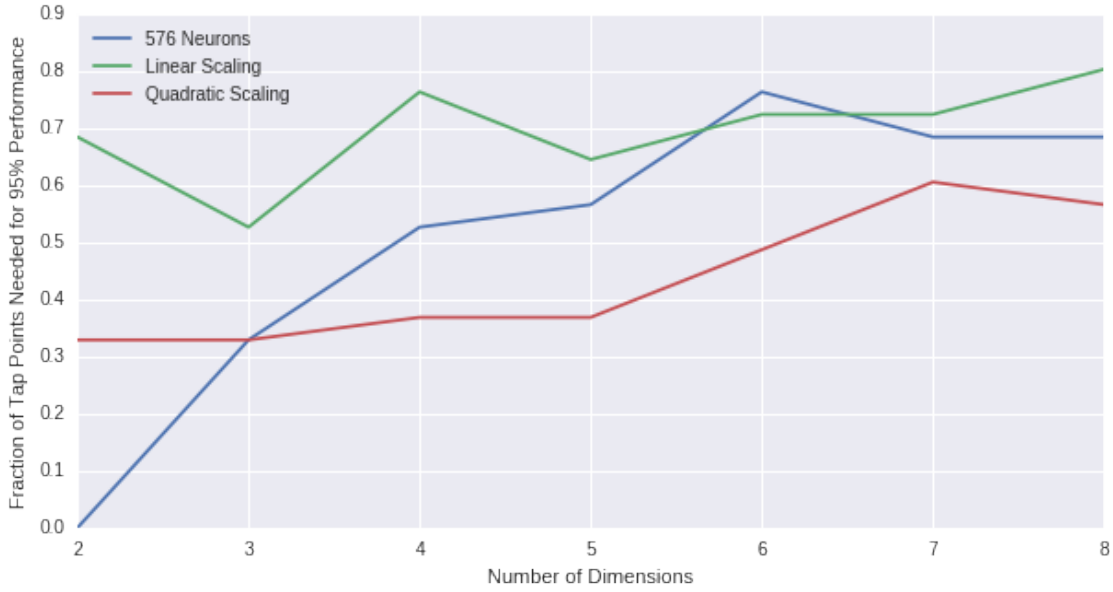

Figure 17: Percentage of tap points necessary to achieve 5% of the performance of the H-Tree method when all tap points are used.

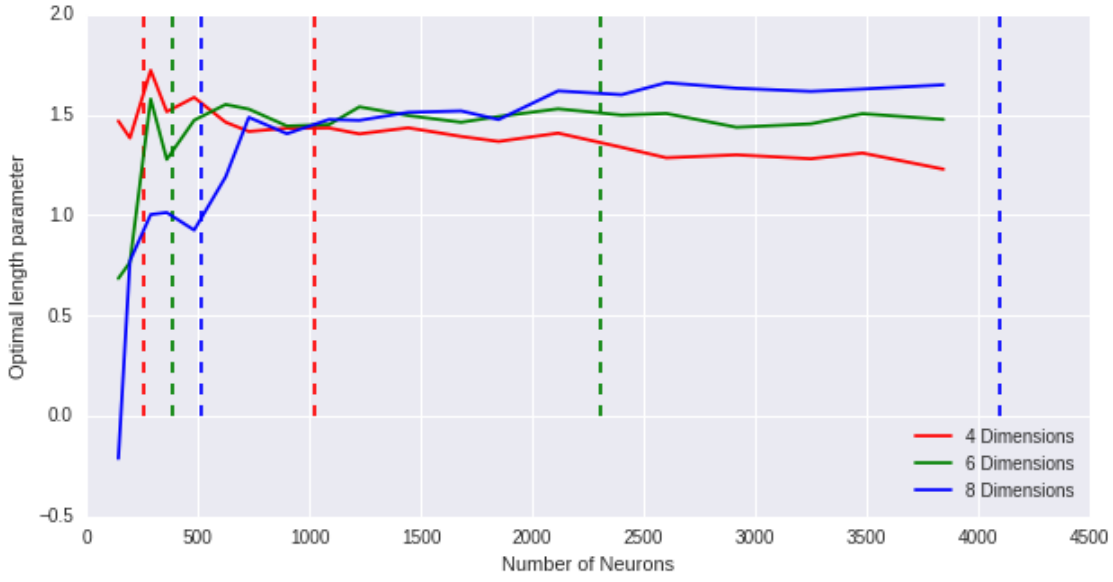

Figure 18: Optimal length parameter for different number of dimensions and neurons. The dotted lines represent the number of neurons demanded for linear and quadratic scaling with respect to the number of dimensions.

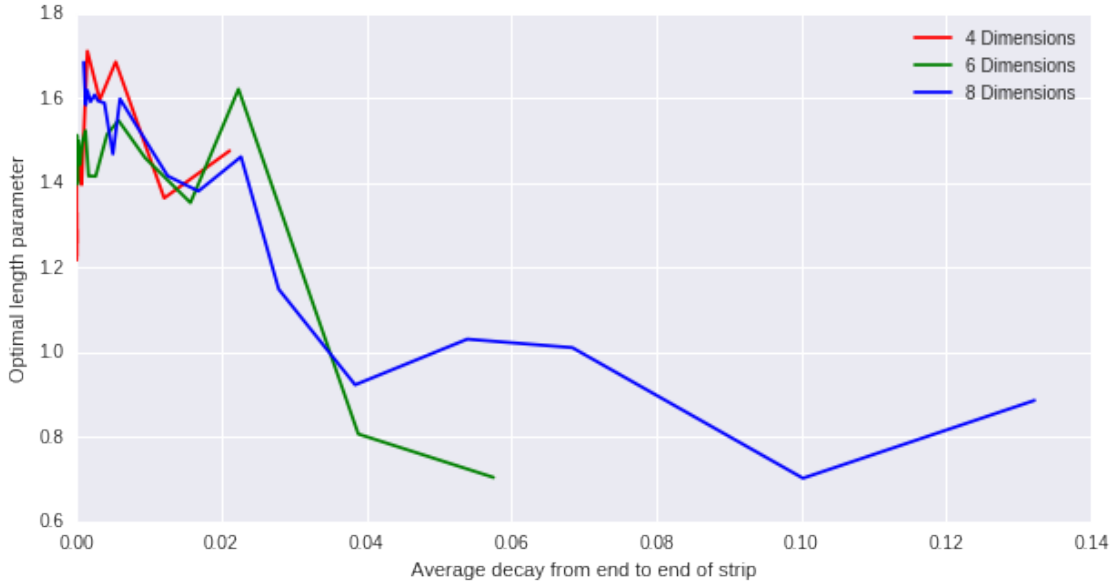

Figure 19: Optimal length parameter for different number of dimensions and neurons. The dotted lines represent the number of neurons demanded for linear and quadratic scaling with respect to the number of dimensions.