Computational Neuroscience 2

USN: 303039534

1 Unsupervised learning

For a general many-input-one-output network we can define an input vector $\mathbf{u}$ as the activity of the neurons in the input layer and a weight vector $\mathbf{w}$ as the strength of connections between the input neurons and the output neuron. The total input to the output neuron is thus.

$$v = \mathbf{w} \cdot \mathbf{u} \tag{1}$$

Based on Hebb's conjecture from 1949, we can write a simple learning rule where the strength of a synapse increases with pre- and post-synaptic coactivity:

$$\tau_w \frac{d\mathbf{w}}{dt} = v\mathbf{u} \tag{2}$$

If we assume that synaptic weight changes occur over a much slower timescale than the inputs, we can use 'batch learning' and average over all input data in every learning step:

$$\tau_w \frac{d\mathbf{w}}{dt} = \langle v\mathbf{u} \rangle \tag{3}$$

Since $v = \mathbf{w} \cdot \mathbf{u}$ and defining the correlation matrix $\mathbf{Q} = \langle \mathbf{uu} \rangle$, this allows us to learn weights using simple matrix multiplication

$$\tau_w \frac{d\mathbf{w}}{dt} = \mathbf{Q} \cdot \mathbf{w} \tag{4}$$

Note that this definition of 'correlation' is slightly odd, as for example the self correlation is given by $Q_{ii} = \langle u_i^2 \rangle$ rather than self correlations always being 1. We then continue to iterate over equation 4 until the weights have converged. If the activity of our neurons corresponds to firing rates, all u_i must be positive. The correlation matrix is therefore a non-negative matrix and we cannot learn negative correlations. Instead, it is therefore common to use a covariance-based learning rule where we let $\mathbf{u}' = \mathbf{u} - \langle \mathbf{u} \rangle$. Now defining the covariance matrix $\mathbf{C} = \langle (\mathbf{u} - \langle \mathbf{u} \rangle)(\mathbf{u} - \langle \mathbf{u} \rangle) \rangle$ we can rewrite our learning rule as

$$\tau_w \frac{d\mathbf{w}}{dt} = \mathbf{C} \cdot \mathbf{w} \tag{5}$$

We note that since these batch learning rules are linear in $\mathbf{w}$, we can treat them analytically. At any given time, we can decompose the weight vector in terms of the eigenvectors e_μ of $\mathbf{C}$:

$$\mathbf{w}(t) = \sum_\mu c_\mu(t)\mathbf{e}_\mu \tag{6}$$

Solving the linear differential equation 5 we get

$$c_\mu = \exp\left(\frac{\lambda_\mu t}{\tau_w}\right)(\mathbf{w}(0) \cdot \mathbf{e}_\mu) \tag{7}$$

Putting these results together gives

$$\mathbf{w}(t) = \sum_\mu \exp\left(\frac{\lambda_\mu t}{\tau_w}\right)(\mathbf{w}(0) \cdot \mathbf{e}_\mu)\mathbf{e}_\mu \tag{8}$$

If the covariance matrix is non-degenerate, we expect the eigenvector corresponding to the highest eigenvalue to dominate this sum at long times due to the coefficient being exponential in the eigenvalue. This in turn suggests that the weight vector will converge towards this first principal component unless it is orthogonal to the initial weight matrix. However, this basic Hebbian learning will lead to unphysiological unbounded weight increases and is therefore usually augmented with some sort of normalization which can alter the simple behavior expected from equation 8 as will be evident from the next two sections.

We note that since the timescale is arbitrary for the above equations, we can set $\tau_w = 1$ for the remainder of this section without loss of generality. We require dt to be smaller than τ_w for accurate Euler integration and therefore let dt=0.01. This is equivalent to using a discrete learning rule with $\epsilon = \frac{dt}{\tau_w} = 0.1$.

1.1 Multiplicative normalization

One commonly used form of normalization is multiplicative normalization where we subtract a term from the simple Hebbian learning rule proportional to the current weights at each iteration. We implement this using Oja's learning rule:

$$\tau_w \frac{d\mathbf{w}}{dt} = v\mathbf{u} - \alpha v^2 \mathbf{w} \tag{9}$$

To see how this leads to normalization, we take the dot product of equation 9 with $\mathbf{w}$

$$\tau_w \frac{d|\mathbf{w}|^2}{dt} = v\mathbf{w}\cdot\mathbf{u} - \alpha v^2 \mathbf{w}\cdot\mathbf{w} = v^2(1-\alpha|\mathbf{w}|^2) \tag{10}$$

Hence $\frac{d|\mathbf{w}|^2}{dt} \to 0$ as $\alpha|\mathbf{w}|^2 \to 1$ and Oja normalization constrains the modulus of the weight vector. At long times, we thus converge towards a weight vector of magnitude $|\mathbf{w}| \to \frac{1}{\sqrt{\alpha}}$. In the following, we let $\alpha = 1$ such that the weight vector converged to is normalized.

As above, we average over the training inputs and get a batch learning rule

$$\tau_w \frac{d\mathbf{w}}{dt} = \mathbf{Q}\cdot\mathbf{w} - \alpha(\mathbf{w}\cdot\mathbf{Q}\cdot\mathbf{w})\mathbf{w} \tag{11}$$

We then update weights according to equation 11 until the norm of the change in weights between iterations is $\epsilon < 10^{-6}$. Setting $\alpha = 1$ we note that when $\mathbf{w}$ becomes an eigenvector of $\mathbf{Q}$ with eigenvalue λ, we can write $\mathbf{Q}\cdot\mathbf{w} = \lambda\mathbf{w}$ and $(\mathbf{w}\cdot\mathbf{Q}\cdot\mathbf{w}) = \lambda$, giving $\tau_w \frac{d\mathbf{w}}{dt} = 0$. For Oja's rule, we thus expect the weight vector to converge to an eigenvector of $\mathbf{Q}$, and we expect this to be the largest eigenvector on the basis of equation 8.

We see an example of this in figure 1a where we train the network on a two-dimensional Gaussian dataset with each point corresponding to a two-dimensional input datapoint $\mathbf{u}$. The red line indicates the direction of the final weight vector, shifted vertically to run through the mean of the dataset, and we see that it does indeed align with the first principal component (the axis of maximum variability).

However, if we shift the mean of the gaussian input dataset to (3,3), the correlation matrix $\mathbf{Q}$ becomes a positive matrix and the largest eigenvalue now corresponds to the eigenvector (1,1) although there is still a negative covariance between the x and y component of the data (figure 1b). In this case, the converged weight vector thus ends up being perpendicular to the first principal component of the data.

However, we can salvage this behavior by using the covariance-based learning rule from equation 5 on the same dataset (figure 1c). This allows us to once again capture the negative covariance between the components $u_1 \& u_2$ as the converged weight vector aligns with the first principal component of the data.

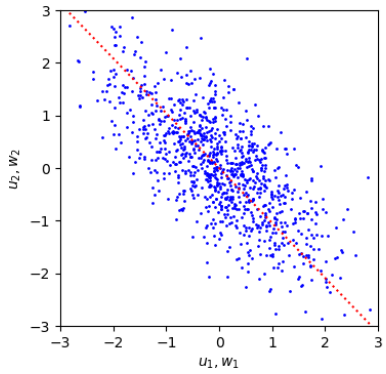
If instead we were to have a positive covariance between the u1 and u2 components of our input vectors, the final weight vector aligns with the principal component irrespective of whether the mean of our distribution is zero, positive or negative provided the mean of both the u1 and u2 components have the same sign. In this case we instead fail to recover the first principal component in the case where the mean of the u1 is positive while the mean of u2 is negative or vice versa. In that case we get a final vec of [0.692383, -0.72153] despite a principal component of [0.688815, 0.724937]. This is again salvaged by using a covariance-based learning rule.

We can also investigate the magnitude of our weight vectors over time since we expect these to converge smoothly to 1 as discussed above, and we see that this is indeed the case (figure 2). Interestingly, it appears that convergence to the [1,1] vector in our second simulation is much faster than converging to a [-1, -1] vector in the first and third simulations.

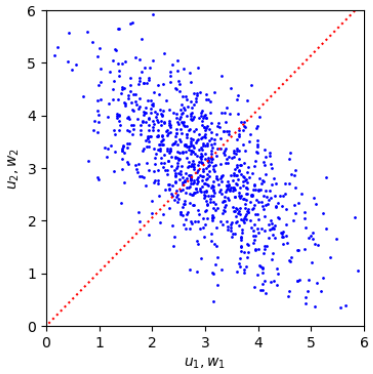
1.2 Subtractive normalization

A different form of normalization that is commonly used to prevent excessive growth of weights is subtractive normalization as specified in equation 12.

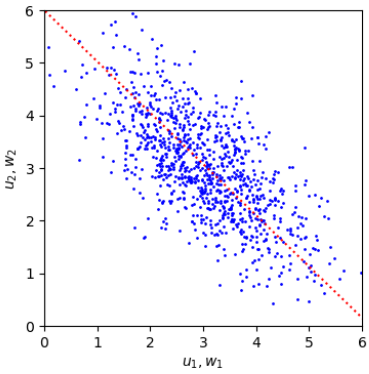
$$\tau_w \frac{d\mathbf{w}}{dt} = v\mathbf{u} - \frac{v(\mathbf{n}\cdot\mathbf{u})\mathbf{n}}{N_u} \tag{12}$$



(a) Mean (0,0) with correlation-based training. Final weight vector: [0.692, -0.722], principal components: [-0.692, 0.722] & [-0.722, -0.692]

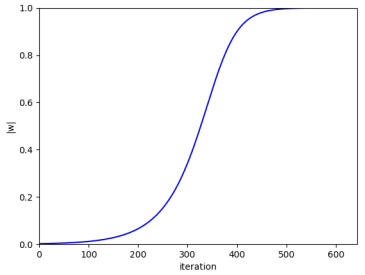


(b) Mean (3,3) with correlation-based training. Final weight vector [-0.697, -0.717], principal components[-0.692, 0.722] [-0.722, -0.692]

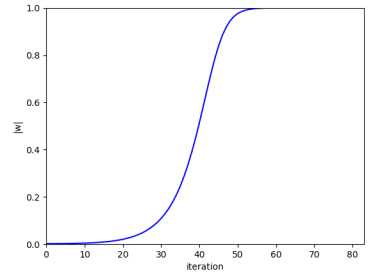


(c) Mean (0,0) with covariance-based training. Final weight vector [0.716, -0.698], principal components [-0.716, 0.698] [-0.698, -0.716]

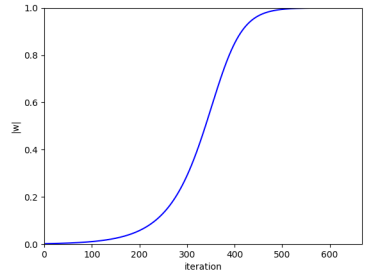
Figure 1: Learning weight vectors from 500 input data points with Oja's rule. In each case, we start from [w1, w2] = [0.001, 0.001] and iterate through equation 11 until convergence using either a correlation or covariance-based learning rule. Red dotted lines indicate the direction of the final weight vector. All datasets were generated with a standard deviation of 1 and $1 - \rho^2$ in the $u1$ and $u2$ directions and a correlation of $\rho = -0.7$.



(a)



(b)



(c)

Figure 2: Weight vector magnitue as a function of iteration number for the three simulations in figure **??**

This constrains the sum of the weights to be constant as can be proven by taking the dot product with $\mathbf{n}$ where $\mathbf{n}$ is a vector of ones:

$$\tau_w \frac{d\mathbf{n} \cdot \mathbf{w}}{dt} = \tau_w \frac{d\sum_i w_i}{dt} = v\mathbf{n} \cdot \mathbf{u}(1 - \frac{\mathbf{n} \cdot \mathbf{n}}{N_u}) = 0 \tag{13}$$

The total sum of the weights in the system is thus constant, but these weights must still be thresholded to avoid unbounded growth of complementary rates towards $\pm\infty$. This is commonly achieved by constraining all weights to be ≥ 0 in weight case there is an upper saturation limit of $w_{max} = \sum_i w_i(0)$.

We once again average our learning rule from equation 12 over the training data and get

$$\tau_w \frac{d\mathbf{w}}{dt} = \mathbf{Q} \cdot \mathbf{w} - \frac{(\mathbf{w} \cdot \mathbf{Q} \cdot \mathbf{n})\mathbf{n}}{N_u} \tag{14}$$

Even using identical initial weights, we will still converge to [0,1] rather than obeying the symmetry of the initial weights, since the data is noisy which leads to our covariance and correlation matrices having non-identical diagonal elements, leading to symmetry breaking. This also makes sense in the context of these learning rules being approximations to learning by presentation of individual datappoints, as we would require every individual datapoint to be symmetric and u1 and u2 to conserve the initial symmetry of the weights.

We can of course just set rho to 0 or 1 and get data

aligned with the x or y axis! Also fix diagonal elements and let the initial weight vector be either parallel or perpendicular to the principal components. (p296). try positively and negatively correlated data.

2 Ocular dominance columns

In the following, we consider a many-input-many-ouput recurrent circuit with input vector $\mathbf{u}$, feedforward weight matrix $\mathbf{W}$ and thus total input vector $\tilde{\mathbf{u}} = \mathbf{W} \cdot \mathbf{u}$. We additionally let the system have a fixed recurrent weight matrix $\mathbf{M}$. The dynamics of this system follow differential equation 15

$$\tau_r \frac{d\mathbf{v}}{dt} = -\mathbf{v} + \mathbf{W} \cdot \mathbf{u} + \mathbf{M} \cdot \mathbf{v} \tag{15}$$

Provided that the eigenvalues of $\mathbf{M}$ have real parts < 1, a stability analysis shows that this system will have a stable fixed point with steady state activity given by

$$\mathbf{v} = \mathbf{W} \cdot \mathbf{u} + \mathbf{M} \cdot \mathbf{v} \tag{16}$$

Which has the solution

$$\mathbf{v} = \mathbf{K} \cdot \mathbf{W} \cdot \mathbf{u} \tag{17}$$

Where $\mathbf{K} = (\mathbf{I} - \mathbf{M})^{-1}$.

If we now fix the recurrent weights and let the feedforward weights change according to a Hebbian learning rule, the time evolution of the weights is determined by equation 18 with $\mathbf{Q}$ defined as in section 1.

$$\tau_w \frac{d\mathbf{W}}{dt} = \langle \mathbf{vu} \rangle = \mathbf{K} \cdot \mathbf{W} \cdot \mathbf{Q} \tag{18}$$

We will use a discretized form of this equation with $\epsilon = 0.01$:

$$\mathbf{W}_{n+1} = \mathbf{W}_n + \epsilon \mathbf{KWQ} \tag{19}$$

We now consider a highly simplified model of ocular dominance including only a single direction along the cortex and a single point in the visual field. This gives rise to only two input activities u_R and u_L corresponding to the input from the right and left eyes. We include 512 output units in the model indexed with the label a which also specifies their location along the 1-dimensional cortex of length $L = 10$mm. We also assume the cortical interactions specified by $\mathbf{K}$ to be translationally invariant and impose periodic boundary conditions requiring $v_0 = v_{512}$ to avoid edge effects.

Assuming that the right and left eye are statistically equivalent, we can now write the input correlation matrix as

$$\mathbf{Q} = \begin{bmatrix} \langle u_R u_R \rangle & \langle u_R u_L \rangle \\ \langle u_L u_R \rangle & \langle u_L u_L \rangle \end{bmatrix} = \begin{bmatrix} q_S & q_D \\ q_D & q_S \end{bmatrix}$$

Expanding the weight matrix $\mathbf{W}$ into its component vectors $\mathbf{w}_R$ and $\mathbf{w}_L$, we can consider the in-phase and out-of-phase combination of these vectors $\mathbf{w}_+ = \mathbf{w}_R + \mathbf{w}_L$ and $\mathbf{w}_- = \mathbf{w}_R - \mathbf{w}_L$ seperately. When expanding out equation 19, these evolve according to

$$\mathbf{w}_+^{n+1} = \epsilon(q_S + q_D)\mathbf{K} \cdot \mathbf{w}_+^n \tag{20}$$

$$\mathbf{w}_-^{n+1} = \epsilon(q_S - q_D)\mathbf{K} \cdot \mathbf{w}_-^n \tag{21}$$

Using subtractive normalization for each pair of weights $w_L(a)$ and $w_R(a)$, we can leave $\mathbf{w}_+$ fixed while $\mathbf{w}_-$ changes. We therefore consider only $\mathbf{w}_-$ in the following and investigate how this weight vector changes over time.

Since the growth of $\mathbf{w}_-$ is proportional to $(q_S - q_D)$, these correlation parameters do not effect the long term behavior of the system but merely the timescale over which it changes provided that $q_D > q_S > 0$. We therefore arbitrarily let $q_S = 1$ and $q_D = 0.7$ for the remainder of this section.

In the present case, we implement subtractive normalization naively by performing the following update steps at each iteration:

$$\mathbf{W}_{n+1} = \mathbf{W}_n + \epsilon \mathbf{KWQ} \tag{22}$$

$$W_{n+1} = W_{n+1} + 0.5(1 - \text{rowsum}(W_{n+1})) \tag{23}$$

$$W_{n+1}[W_{n+1} < 0] = 0 \tag{24}$$

$$W_{n+1}[W_{n+1} > 1] = 1 \tag{25}$$

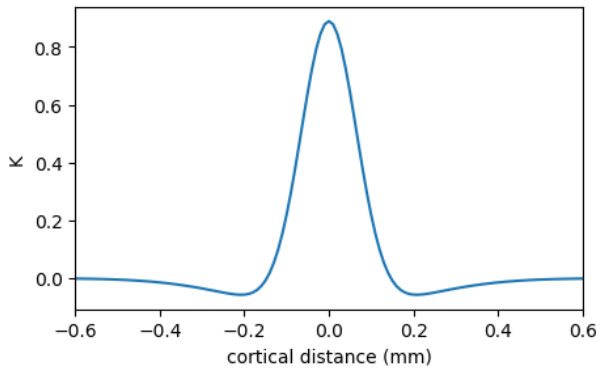
We intialize the weights such that $\mathbf{w}_L = 0.5 + \epsilon$ and $\mathbf{w}_R = 1 - \mathbf{w}_L$ where $\epsilon = \mathcal{N}(0, 0.01)$. This approach ensures that $\mathbf{w}_+ = 1$ at all times and implements subtractive normalization with thresholds of 0 and 1. This could have been implemented in a similar way to section 1 by altering equation 19 directly, but the present implementation is conceptually simpler and less error-prone in its implementation while achieving the same result.

For this model, we generate cortical interactions as a function of intercortical distance according to equation 26 with $\sigma = 0.066$mm.

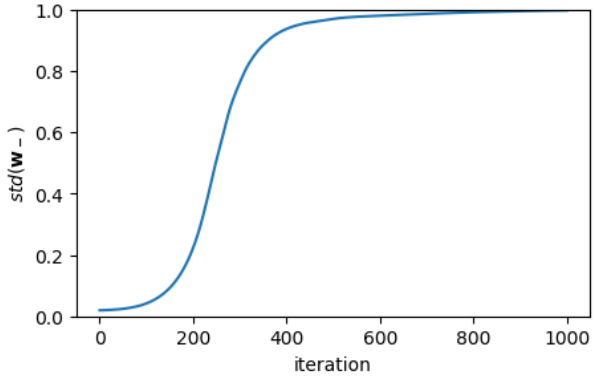
$$K_{aa'} = \exp\left(-\frac{(a-a')^2}{2\sigma^2}\right) - \frac{1}{9}\exp\left(-\frac{(a-a')^2}{18\sigma^2}\right) \tag{26}$$

The strength of interaction has been plotted as a functional of relative cortical positions in figure 3a. We see that close-range interactions are strongly excitatory while long-range interactions are weakly inhibitory. This is what will result in an oscillatory pattern of ocular dominance.

Given this definition of cortical interactions, we can simulate the system of 512 cortical neurons by calculating the 512x512 interaction matrix $\mathbf{K}$ and implementing equations 22 - 25. In figure 3b, we plot the standard deviation of $\mathbf{w}_-$ which allows us to follow the progress of the simulation as $\mathbf{w}_-$ goes from having a standard deviation of 0 when all elements are ≈ 0.5 to having a standard deviation of 1 when the elements are ± 1. We see that the simulation has converged by 1000 timesteps and run all remaining simulations for 1000 timesteps unless otherwise noted.



(a) $K_{aa'}$ where a has been fixed a position 0 and a' varies from -0.6mm to 0.6mm.



(b) Standard deviation of $\mathbf{w}_-$ over time illustrating how $\mathbf{w}_-$ progresses from $\approx \mathbf{0}$ to a vector of ± 1 after 800-1000 iterations.

Figure 3

We can also extract the vector $\mathbf{w}_-$ at different points in the simulation and plot the vector as a 1-dimensional heatmap along the x-direction where individual neurons are coloured from white to black according their ocular dominance (figure 4).

From our previous considerations, we expect the long-term behavior of $\mathbf{w}_-$ to be dominated by the prinpical eigenvector of $\mathbf{K}$. In the case of periodic boundary conditions as imposed here, the eigenvectors of $\mathbf{K}$ are given by

$$e_a^\mu = \cos\left(\frac{2\pi\mu a}{512} - \phi\right) \tag{27}$$

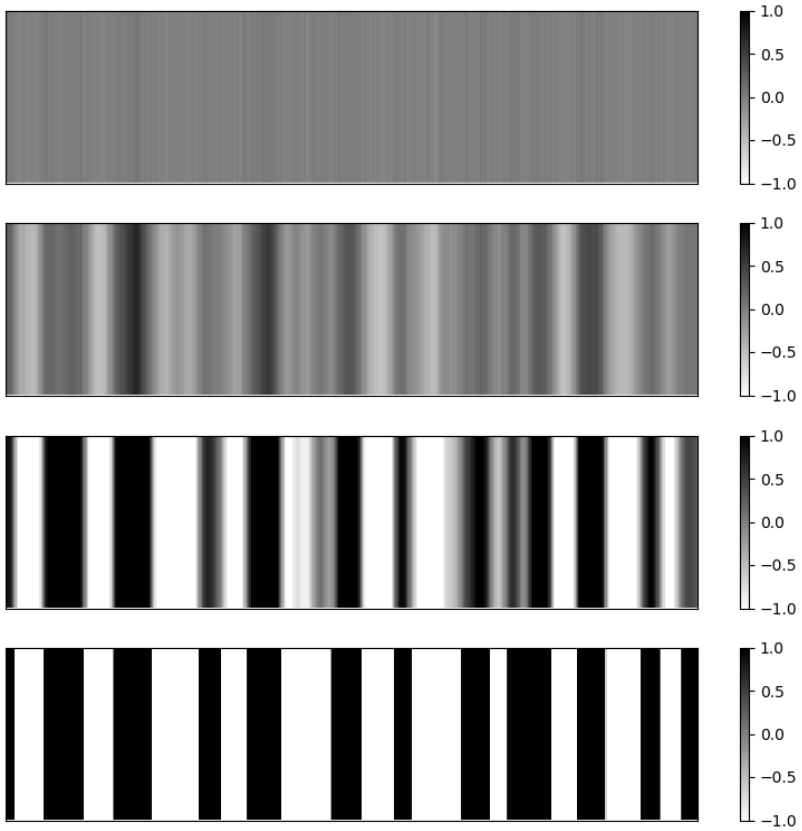


Figure 4: $\mathbf{w}_-$ as a heatmap after 50, 200, 300 and 1000 iterations. We see that ocular dominance becomes increasingly strong over time and that the short-range excitation with long-range inhibition drives the formation of an oscillatory pattern of dominance.

Where $\mu = \dfrac{512 \cdot d \cdot k}{2\pi}$ takes integer values and the eigenvalues are given by the discrete fourier transform $\tilde{K}(\mu)$.

Here, k is the spatial frequency of the ocular dominance columns and d is the separation between sites a and $a+1$; i.e. $d = \dfrac{10mm}{512}$. The principal eigenvector is thus the eigenfunction e_μ with μ corresponding to the maximum of $\tilde{K}(\mu)$, the discrete fourier transform of $\mathbf{K}$.

To find the principal eigenvector of $\mathbf{K}$ and thus the expected long-term behavior of the system, we first find $\tilde{K}(k)$.

$$\tilde{K}(k) = \int_{-\infty}^{\infty} K(x) e^{-2\pi i k x} dx \tag{28}$$

We know that the fourier transform is a linear operator and that the fourier transform of a gaussian is given by

$$G(k) = \int_{-\infty}^{\infty} \frac{1}{\sqrt{2\pi\sigma^2}}\, e^{\dfrac{-x^2}{2\sigma^2}}\, e^{-2\pi i k x} dx = e^{\dfrac{-k^2\sigma^2}{2}} \tag{29}$$

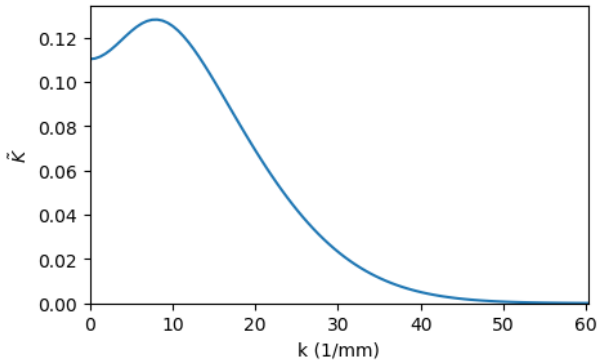
This allows us to easily calculate the required fourier transform

$$\tilde{K}(k) = \sqrt{2\pi\sigma^2} e^{\dfrac{-k^2\sigma^2}{2}} - \frac{1}{9}\sqrt{18\pi\sigma^2} e^{\dfrac{-9k^2\sigma^2}{2}} \tag{30}$$

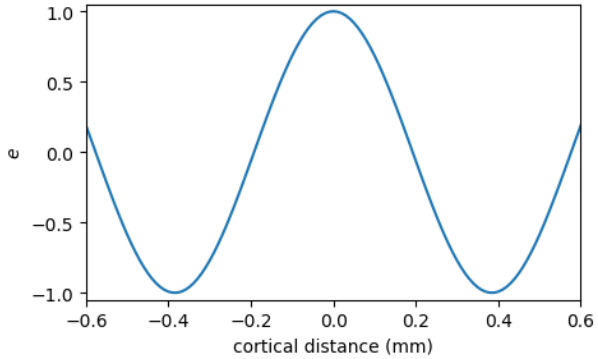
We plot this in figure 5a and find that the maximum of $\tilde{K} = 0.128$ occurs at $k = 8.17$. We can now find the value of μ corresponding to this maximum $\tilde{K}$ as

$$\mu_{max} = \frac{512}{2\pi}\ \frac{8.17}{}\ \frac{10}{512} = 13 \tag{31}$$

This allows us to calculate the principal eigenvector $e_{\mu_{max}}$ as a function of cortical distance and we plot this in figure 5b. We note that the pattern of inhibition and excitation follows that of K in figure 3a. Since $\mu_{max} = 13$, we expect to generate 13 periods of left and right dominance in our system in a given simulation which is consistent with the results in figure 4.



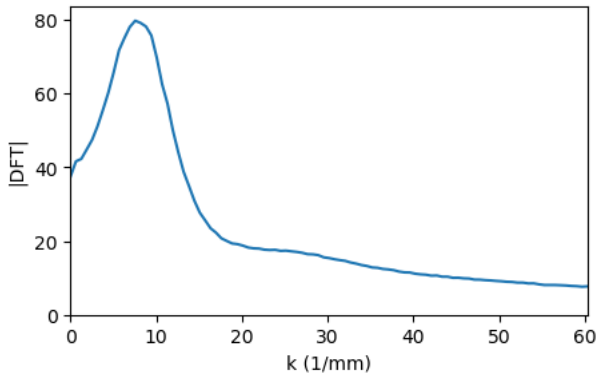
(a) Fourier transform of K as a function of spatial frequency k. This has a maximum of $k = 8.17$ corresponding to $\mu = 13$ suggesting that we will generate 13 periods of ocular dominance in a simulation with these parameters.



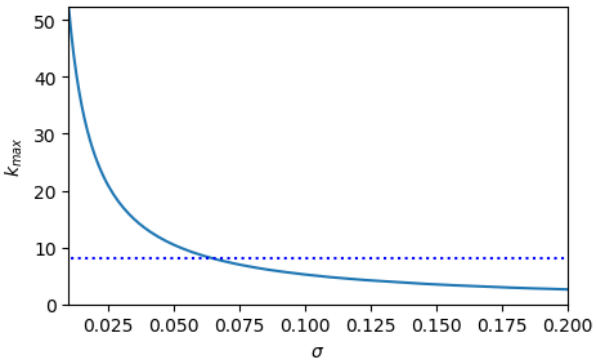
(b) Principal eigenvector of K capturing the pattern of short-range excitation and long-range inhibition.

Figure 5

We now repeat our simulation of the system 1000 times and calculate the magnitude of the discrete fourier transform (DFT) at each trial. We plot the mean of the DFTs in figure 6a and see that the major components of our equilibrium $\mathbf{w}_-$ vector do indeed correspond to the values of k for which $\tilde{K}$ is highest in figure 5a.



(a) Mean magnitude of the discrete fourier transform of the steady state $\mathbf{w}_-$ vector across 1000 simulations.



(b) Principal eigenvector of $\mathbf{K}$

Figure 6

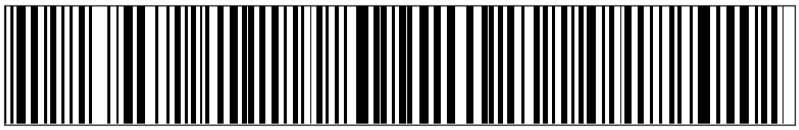
We find that the mean of the DFTs has a maximum of 79.7 at $\mu = 13$ which is consistent with our analytical considerations suggesting that the largest component of the long-time $\mathbf{w}_-$ vector should correspond to the principal eigenvector which does indeed have $\mu = 13$

The stripe width in this model is determined by the spatial range of excitatory and inhibitory interactions which in turn is specified by the scale parameter σ in equation 26. In order to decrease the stripe width in the model, we need to decrease the scale parameter σ which leads to an increase in the value of k that maximizes our discrete fourier transform and thus an increased frequency of the principal eigenvector of $\mathbf{K}$. In order to increase the stripe width, we instead increase σ.

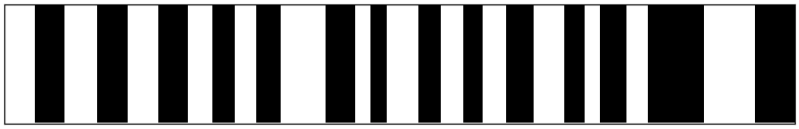
The relationship between σ and k has been investigated numerically, and the value of k that maximizes $\tilde{K}$ for a given σ has been plotted in figure 6b which shows a rapid increase in frequency as $\sigma \to 0$.

To validate these results, we run two additional simulations; one with $\sigma = 0.012$ for which we expect $k = 40.31$ and

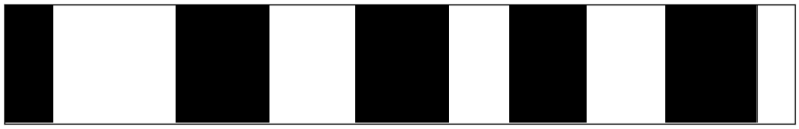
thus $\mu = 69$, and one with $\sigma = 0.174$ for which we expect $k = 3.01$ and $\mu = 5$. We plot the resulting steady state $\mathbf{w}_-$ vectors in figure 7 and see that we do indeed get increasingly wider stripes with increasing σ and that the number of full periods matches the expected value of μ.



(a) $\sigma = 0.012$ giving $k = 40.31$ and $\mu = 69$



(b) $\sigma = 0.066$ giving $k = 8.17$ and $\mu = 13$



(c) $\sigma = 0.174$ giving $k = 3.01$ and $\mu = 5$

Figure 7: Simulations of the ocular dominance system for different values of σ corresponding to different interaction ranges. Longer-range interactions lead to wider stripes.

We could of course also alter the cortical interactions in other ways, e.g. by altering the ratio of the two gaussians from $1 : \frac{1}{9}$ or by changing the relative standard deviations from $\sigma_2 = 3\sigma_1$, but these options are not explored further as they do not preserve the form of the interactions.

3 The elastic net

The Travelling salesman problem is an example of an NP complete problem, and there is thus no (current) algorithm that can solve it in polynomial time. As the number of nodes to be visited N increases, we are therefore forced to use various heuristic algorithms to try to approximate optimal solutions. One such algorithm dubbed the 'elastic net' was implemented by Durbin and Willshaw in a 1987 letter to *Nature*.

In this algorithm, there are N cities to be visited, and this is achieved by distorting an initial path of M points until it runs through all N cities in a way that locally minimizes the path length L. We denote the cities to be visited $\{x_i\}$ and the M points on the path travelled on $\{y_j\}$. At each timepoint the elastic net then updates each point on the path according to

$$\Delta y_j = \alpha \sum_i w_{ij}(x_i - y_j) + \beta K(y_{j+1} - 2y_j + y_{j-1}) \tag{32}$$

The first term serves to decrease the distance between a point on the path and a city, ensuring that all points on the path will eventually run through a city. The second term serves to bring the points as close to each other as possible, ensuring that the path length is minimized.

In equation 38, the weights w_{ij} are given by

$$w_{ij} = \frac{\phi(|x_i - y_j|, K)}{\sum_k \phi(|x_i - y_k|, K)} \tag{33}$$

Where

$$\phi(d, K) = \exp\left(\frac{-d^2}{2K^2}\right) \tag{34}$$

This implementation of the elastic net corresponds to minimizing an energy function

$$E = -\alpha K(\sum_i \ln \sum_j \phi(|x_i - y_j|, K)) + \beta \sum_j |y_{j+1} - y_j|^2 \tag{35}$$

Where again the first term penalizes when the path is far from a city and the second term penalizes a long path. Our update step from equation 38 then has the property

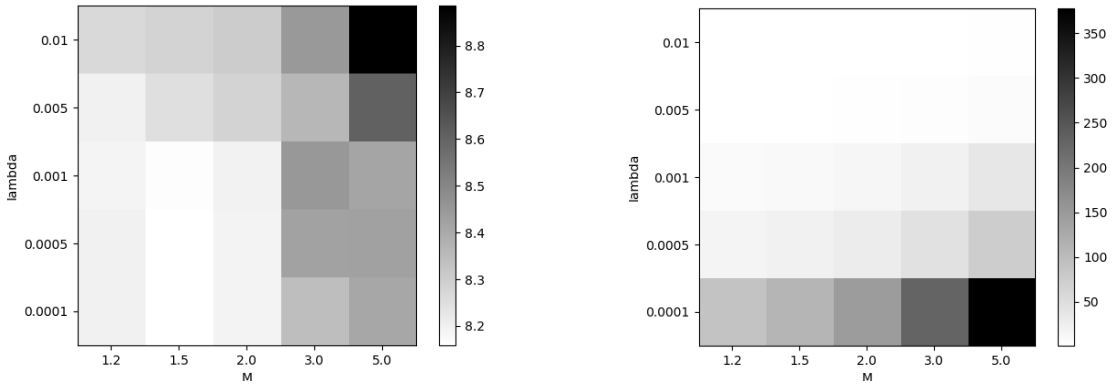
$$\Delta y_j = -K \frac{\partial E}{\partial y_j} \tag{36}$$

We are thus carrying out gradient descent on equation 35 with adaptive learning rate K, ensuring that we will eventually arrive at a local minimum of the energy.

Over the course of the optimization, Durbin and Willshaw reduce K by 1% every 25 iterations from $K = 0.2$ to $K = 0.01$ and fix $M = 2.5N$. However, given the advances in computing power since 1987, we can explore the effect of K and M on the performance of the algorithm more thoroughly.

For the remainder of this section we set $\alpha = 0.2$ and $\beta = 2.0$ as in Durbin and Willshaw, and we choose points on our initial path to be evenly spread out at a distance $0.1 + \text{unif}(-0.001, 0.001)$ from the centroid of the cities. Cities are generated as a grid of 100 randomly distributed points in a 1x1 square.

We let K decay exponentially according to $K = 0.2 * \exp -\lambda n$ where n is the number of timesteps, and a simulation is considered finished when $K < 0.001$. We can now vary lambda and the number of points M on our path to optimize the TSP optimization with the result of a coarse-grained parameter grid given in figure 8.



(a) Length of final path

(b) Time to convergence (seconds)

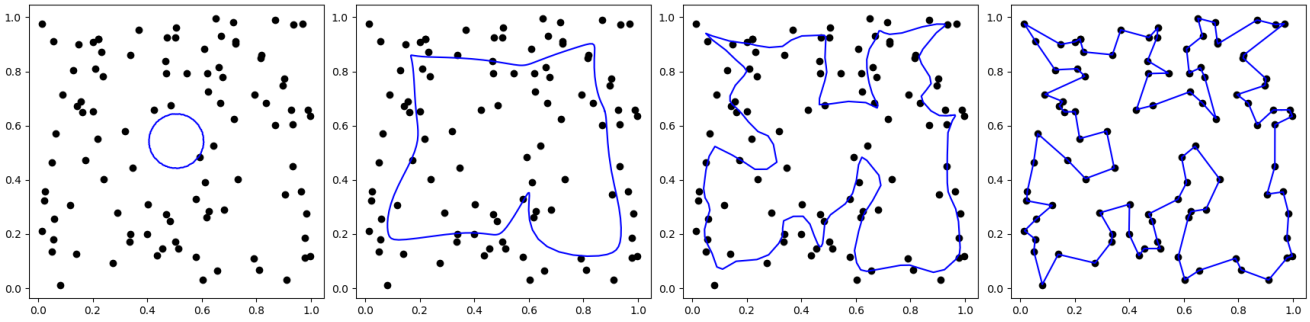
Figure 8: Performance and computational time for the elastic net approach to the travelling salesman problem as a function of decay rate λ and number of points on the path M. All simulations were run on the same set of 100 cities with remaining parameters as specified in the main text.

We see from figure 8a that in contrast to the parameters used by Durbin and Willshaw, we actually ahieve the best performance with a relatively small value of M=1.5N since small values of M tend to reduce the popensity of the system to get caught in higher-energy local minima compared to $M > 2N$. We also note that there is an increase in performance with decreasing lambda as expected, but this effect is neglible for $\lambda < 0.0005$.

From figure 8b we see that the time taken for a simulation is approximately linear in both λ and M. Considering this data together with figure 8a, we therefore pick M=1.5 and $\lambda = 0.0005$ for the remainder of our simulations. More finegrained optimization could be carried out, but this was not deemed particularly informative since the details of the optimization will depend upon the specific grid being investigated. However, the above analysis was repeated with two additional random grids and similar patterns were observed.

We can now investigate how the path develops as K decreases for our optimum parameters of M=1.5 and $\lambda = 0.0005$ (figure 9). We see that the path initially expands relatively uniformly to decrease the distance from the far-away cities to the path. The path then locally distorts as it moves closer to every individual point. This is finally achieved at K=0.001 where the path has converged and the salesman visits every city. When M gets too close to N, the algorithm occasionally converges to a local minimum where the path does not visit every city. For $M \geq 1.5N$ this has not been observed.

At a first glance, this looks like a reasonable route as there are no obvious detours or crossovers. However, to get a better idea of how efficient the elastic net method is, we can compare it with the well established travelling salesman method of simulated annealing. For this comparison, we implement simulated annealing as described in *Stochastic Processes* (Chang 2007).



(a) K=0.2 (b) K=0.11 (c) K=0.04 (d) K=0.001

Figure 9: Timecourse of the travelling salesman simulation with M=1.5 and $\lambda = 0.0005$

We can consider a given TSP route (order of cities) to be a point in a set $\mathcal{S}$ of $\frac{(n-1)!}{2}$ possible routes. Our aim is to find a route that minimizes the total length L by moving between neighboring points on $\mathcal{S}$. Here, we define 'neighboring points' as routes that can be interconverted by reversing the part of the path between two cities.

In the simulated annealing algorithm, one iteration involves moving uniformly at random to a neighboring point in $\mathcal{S}$ and calculating the length $L1$ of the new route obtained. If $L1 < L0$, we accept this new route as our current route. If $L1 > L0$, we accept the new route with probability $p = \exp\left(-\frac{L1 - L0}{T}\right)$. Otherwise we retain the old route.

This finite probability of accepting a worse route allows us to move out of local minima and thus to approach the global minimum with a higher probability than a simple gradient descent algorithm. This is similar to e.g. the annealing process in glass (from which the algorithm has its name), or protein folding in a cell where the finite temperature allows for stochastic movement out of local minima in conformational space.

Over the course of the simulation, we decrease the tempeature according to

$$T_n = \frac{T_0}{ln(n)} \tag{37}$$

Here n is the number of iterations. The decrease in temperature makes it increasingly unlikely that we will move to a worse route, and in the limit of $T \to 0$, we will move to the nearest local minimum as there is no thermal energy in the system to drive it to a longer path.

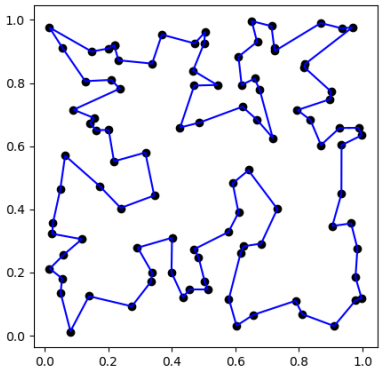
Setting $T_0 = 10$ and running 10^7 iterations has been found empirically to give good results on a timescale similar to the one observed for the elastic net ($\tilde{2}0$ seconds). To compare the two methods, we use the map of 100 cities considered above as well as two new randomly generated maps. We then quantify the optimum route generated by the elastic net in terms of both route length and time to convergence. We compare this to 30 trials of simulated annealing since the simulated annealing process is stochastic and results thus vary per trial. A summary of the results is given in table 1.

	Map 1	Map 2	Map 3
Elastic net	8.159	7.771	7.815
Time (s)	23.32	23.819	24.91
Simulated annealing	7.956 (0.047)	7.768 (0.052)	7.633 (0.075)
Time (s)	19.06 (1.33)	19.47 (1.97)	18.47 (1.12)

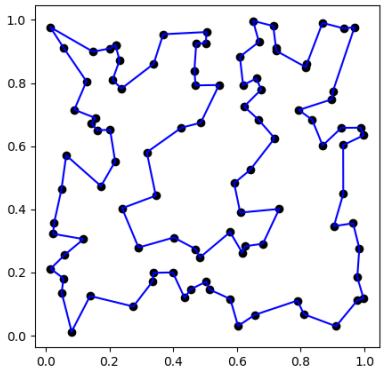
Table 1: Performance and computational time for the elastic net and simulated annealing across three different maps of 100 randomly generated cities. For simulated annealing, the numbers in brackets represent standard deviations from 30 trials.

We see that simulated annealing consistently outperforms the elastic net, both in terms of the converged route and the time taken to find this route. This can be seen qualitatively in figure 15.

However, for map 2 simulated annealing only slightly outperforms the elastic net and performs worse for half the trials, raising the question of whether there are any profound insights to be gained from this difference between maps. For map 1, we find that the optimum route from simulated annealing appears very asymmetric. It is thus less likely to

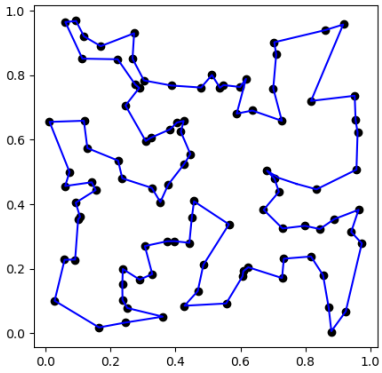


(a) Elastic net (L=8.159)

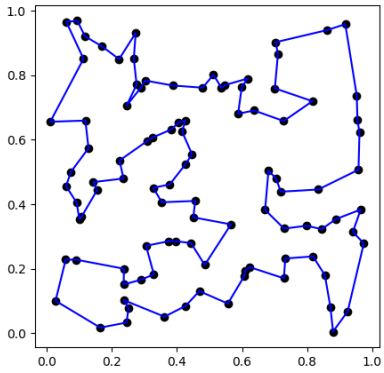


(b) Simulated annealing (L = 7.919)

Figure 10: Optimum routes for map 1 found by the elastic net method and simulated annealing. Simulated annealing finds a shorter route than the elastic net, and this route involves only visiting the inner region of the map once.



(a) Elastic net (L = 7.771)



(b) Simulated annealing (L = 7.685)

Figure 11: For map 2, the solutions found by simulated annealing are more symmetrical, and the elastic net comes close to the performance of simulated annealing.

be converged upon by the elastic net, the dynamics of which lead to an initial expansion of the route followed by the formation of local invaginations (figure **??**).

For map 2, on the contrary, the optimum route from simulated annealing appears more symmetrical which might explain why the elastic net comes close to the performance of simulated annealing in this case.

Simulated annealing is thus a superior method to the elastic net in terms of solving the travelling salesman problem, but we see that the elastic net still produces reasonable routes that are comparable in length to those generated by simulated annealing albeit generally more symmetric. The elastic net methodology is therefore still somewhat interesting due to its different dynamics and blah blah...

To use the elastic net to model ocular dominance in 2 dimensions, we imagine the tour as being a plane representing a region of the visual cortex. Each point has elastic connections to each other point. The retinal input are represented by two horizontal planes corresponding to the left and right eyes respectively with each point being the equivalent of a city in the TSP. We now index our tour with i, j and the retinal positions by a, b. This gives a new 20

$$\Delta y_{ij} = \alpha \sum_{a,b} w_{ab,ij}(x_{ab} - y_{ij}) + \beta K(y_{i\,j+1} + y_{i+1\,j} - 4y_{i\,j} + y_{i\,j-1} + y_{i-1\,j} \tag{38}$$

4 Cart pole balancing problem

Associate Search Element (ASE). Picks a decision based on a reward signal and eligibility trace. *Adaptive Critic Element* (ACE) provides the ASE with a predicted reward over continuous time rather than a discrete reward at the

end of a trial.

State of the cartpole balancing problem defined by a vector $(x, \theta, \dot{x}, \dot{\theta})$ where x is the position of the cart on a one-dimensional track with $x \in [-2.4, 2.4]$. θ is the angle of the pole from vertical and dots represent time derivatives. A trial is considered to have failed when $|x| \geq 2.4$ or $|\theta| \geq 12°$.

We divide the state space into 162 distinct regions defined by
$x \in [-2.4, -0.8], [-0.8, 0.8], [0.8, 2.4]$ m
$\theta \in [-12, -6], [-6, -1], [-1, 0], [0, 1], [1, 6], [6, 12]$ °
$\dot{x} \in [-\infty, -0.5], [-0.5, 0.5], [0.5, \infty]$ m/s
$\dot{\theta} \in [-\infty, -50], [-50, 50], [50, \infty]$ °/s

At any time t, the system will be in a given state $state(t)$ and it receives an impulse either to the right or to the left based on the relative predicted reward for right and leftwards impulses for as represented by $w_i(t)$ for state i. We therefore map a given state vector to a binary 162-dimensional vector that has all elements equal to zero except for the element corresponding to the current state of the system.

The output of an element is then determined by

$$y(t) = \text{sign}[\sum_{i}^{162} w_i(t)x_i(t) + \mathcal{N}(0, \sigma^2)] = \text{sign}[w_{state(t)}(t) + \mathcal{N}(0, \sigma^2)] \tag{39}$$

Where $x_i(t)$ is 1 if the system is in state i at time t and zero otherwise, and $w_{state(t)}(t)$ is the weight at time t corresponding to the state of the system at time t. The Gaussian noise both simulates noise in real neural systems and allows our cart pole balancing system to explore the available state space.

For the ASE we update the weights according to

$$w_i(t+1) = w_i(t) + \alpha r(t) e_i(t) \tag{40}$$

This is a three-factor learning rule where α is the learning rate, $r(t)$ is the global reward at time t and $e_i(t)$ is the eligibility trace of unit i at time t. For the pole balancing problem, $r = 0$ throughout a trial and becomes 1 when failure occurs. We update the eligiblity traces according to

$$e_i(t+1) = \delta e_i(t) + (1 - \delta) y(t) x_i(t) \tag{41}$$

where δ determines the trace decay rate and x, y are as defined above.

This system does not perform particularly well given the sparse error signal which only occurs after a long sequence of events. Performance is better if we include an ACE which estimates a predicted error at all times. The predicted error is given by

$$\hat{r}(t) = r(t) + \gamma p(t) - p(t-1) \tag{42}$$

Here, γ leads to the predicted reward tending towards 0 in the case of prolonged periods without external reinforcement. $p(t)$ is a prediction of eventual reinforcement, given by

$$p(t) = \sum_{i}^{162} v_i(t) x_i(t) = v_{state(t)}(t) \tag{43}$$

This requires us to update the weights v_i such that they remain an accurate predictor of reward. This is achieved using the learning rule

$$v_i(t+1) = v_i(t) + \beta[r(t) + \gamma p(t) - p(t-1)]\bar{x}_i(t) \tag{44}$$

Here, β determines the learning rate and $r(t)$ is the external reinforcement signal as above. $\bar{x}_i(t)$ is similar to the eligibility trace $e_i(t)$ for the ASE and is updated according to

$$\bar{x}_i(t+1) = \lambda \bar{x}_i(t) + (1 - \lambda) x_i(t) \tag{45}$$

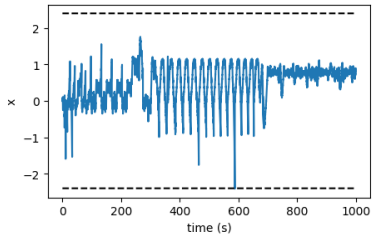
Where λ is the trace decay rate. This is equivalent to the update rule for $e_i(t)$ with $y_i(t) = 1$.

We initialize all $e_i, w_i, v_i = 0$ at time $t = 0$. We intialize the cart at x=0 and the pole at an angle of 0 degrees. the first impulse is thus stochastic and determined entirely by the noise term.

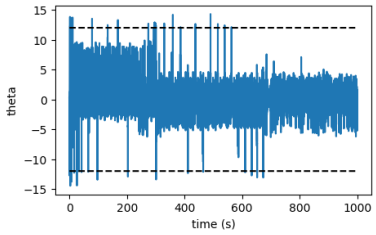
Brato et al. using Euler integration with a timestep of 0.02 seconds in the original paper. However, we find this to give highly inaccurate intergration and instead use a timestep of 0.001

$\alpha = 1000$ $\beta = 0.5$ $\delta = 0.9$ $\gamma = 0.95$ $\lambda = 0.80$ $\sigma = 0.01$

Maintaining the cart within the bounds and maintaining the pole upright are of course related; for the default specs, the pole never falls over. However, we can shorten the pole half length to 0.1m in which case the pole does fall over but the cart never reaches the bounds of the track. At 0.2, both can happen.



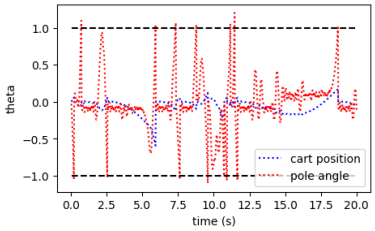
(a) Elastic net (L=8.159)



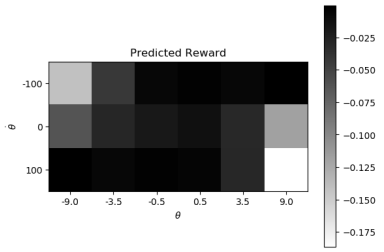
(b) Simulated annealing (L = 7.919)

Figure 12: Optimum routes for map 1 found by the elastic net method and simulated annealing. Simulated annealing finds a shorter route than the elastic net, and this route involves only visiting the inner region of the map once.

Set l=0.1

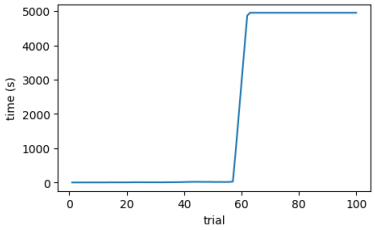


(a) Elastic net (L=8.159)

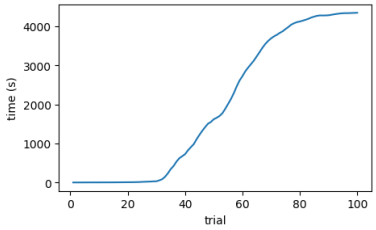


(b) Simulated annealing (L = 7.919)

Figure 13: Optimum routes for map 1 found by the elastic net method and simulated annealing. Simulated annealing finds a shorter route than the elastic net, and this route involves only visiting the inner region of the map once.



(a) Elastic net (L=8.159)



(b) Simulated annealing (L = 7.919)

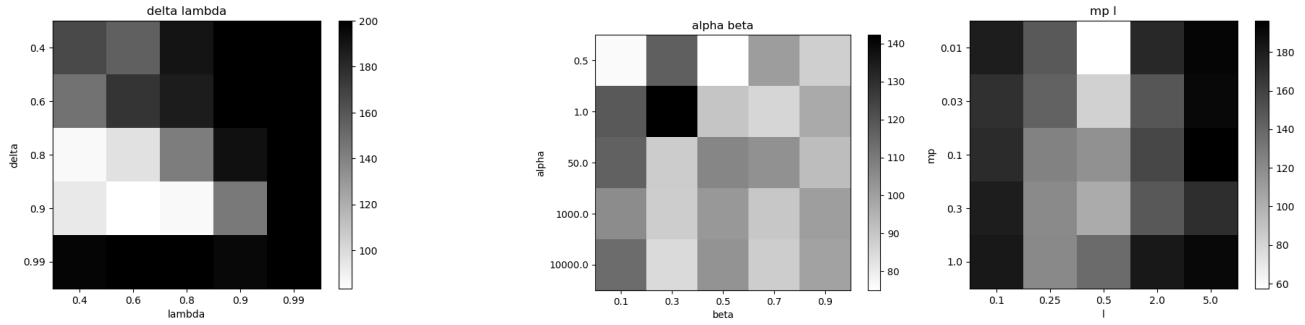
Figure 14: Optimum routes for map 1 found by the elastic net method and simulated annealing. Simulated annealing finds a shorter route than the elastic net, and this route involves only visiting the inner region of the map once.

lots of parameters in the model; won't vary all of them but will vary a few.

Eligibility trace ecay rates of the ASE and ACE (delta and lambda) Length and mass of the pole learnings rates (alpha and beta) predicted reward decay rate (gamma)

We can add a second layer to the network and denote this z with weights from the input signal given by the matrix D

$$z_i(t) = g(\sum_{j=1} D_{ij}(t)x_j(t)) \tag{46}$$



(a) Elastic net (L=8.159) (b) Simulated annealing (L = 7.919) (c) Simulated annealing (L = 7.919)

Figure 15: Optimum routes for map 1 found by the elastic net method and simulated annealing. Simulated annealing finds a shorter route than the elastic net, and this route involves only visiting the inner region of the map once.

Where $g(s) = \dfrac{1}{1+e^{-1}}$.

This allows us to calculate a probability

$$P_(t) = g(\sum_{i=1} w_i(t)x_i(t) + \sum_{i=1} f_i(t)z_i(t)) \tag{47}$$

where f_i are the weights from the hidden layer to the output layer (equivalent to w_i for the input layer). We then have $y(t) = 1$ with probability $P(t)$ and $y(t) = -1$ with probability $1 - P(t)$.

Our ACE also gets a hidden layer q with connectivity matrix A from the input layer and activities

$$q_i(t) = g(\sum_{j=1} A_{ij}(t)x_j(t)) \tag{48}$$

$$p_(t) = \sum_{i=1} v_i(t)x_i(t) + \sum_{i=1} c_i(t)q_i(t) \tag{49}$$

We then update weights according to

$$w_i(t+1) = w_i(t) + \alpha r(t)(\max(y(t), 0) - P(t))x_i(t) \tag{50}$$

$$f_i(t+1) = f_i(t) + \alpha r(t)(\max(y(t), 0) - P(t))z_i(t) \tag{51}$$

$$D_{ij}(t+1) = D_{ij}(t) + \alpha r(t)z_i(t)(1 - z_i(t))\text{sign}(f_i(t))(\max(y(t), 0) - P(t))x_j(t) \tag{52}$$

$$v_i(t+1) = v_i(t) + \beta r(t)x_i(t) \tag{53}$$

$$c_i(t+1) = c_i(t) + \beta r(t)q_i(t) \tag{54}$$

$$A_{ij}(t+1) = A_{ij}(t) + \beta r(t)q_i(t)(1 - q_i(t))\text{sign}(c_i(t))x_j(t) \tag{55}$$

in the present case, the additional layer does not significantly improve performance; however, Anderson had made the task more difficult by scaling, shifting and random initialization of the task and found that the second layer only improved performance after a large number of iterations. However, in the present simpler case, the network rapidly learns the task and the second layer is therefore not necessary.