Kohonen Maps on Hand-Written Digits

Allan S. Nielsen, Doctoral student in Mathematics
Chen Liu, 1st year student M.Sc. Computer Science
November 29, 2015

1 The Kohonen Network

We begin with a Kohonen network of 6x6 neurons arranged in a square grid with unit distance, the neighborhood function being a Gaussian with a constant standard deviation of $\sigma = 3$. Since the learning process is an iterative algorithm, we need a stopping criteria. Constructing this stopping criteria is not completely trivial, unlike the Batch-version, the Online implementation of the Kohonen map does not convergence in the sense that $\lim_{t\to\infty} \left\| \Delta\vec{w}_i^t - \Delta w_i^* \right\| = 0$ for some fixed point Δw_i^*. Instead, the statistics $\langle \Delta\vec{w}_i^t \rangle$ converge in the limit $t \to \infty$ for each prototype i, i.e., we may write

$$\lim_{t\to\infty} \left\| \langle \Delta\vec{w}_i^t \rangle \right\| = 0, \quad \forall i \tag{1.1}$$

as proven in exercise 1b during exercise-session 6. In order to use $\left\| \langle \Delta\vec{w}_i^T \rangle \right\|$ to somehow estimate whether or not the algorithm converged sufficiently, we must be able to compute an estimate for it on the fly. A simple idea is to compute the average of $\Delta\vec{w}_i^t$ on $t \in [T/n, T]$ for some constant $\{n \in \mathbb{N} : n > 1\}$. This has the correct limit as $t \to \infty$, see figure 0.1. Unfortunately, as is trivial to see, in the limit t large, all compute cycles will be spend trying to evaluate $\left\| \langle \Delta\vec{w}_i^T \rangle \right\|$ so to determine whether to stop the computation or not, instead of actually doing any learning. To avoid such

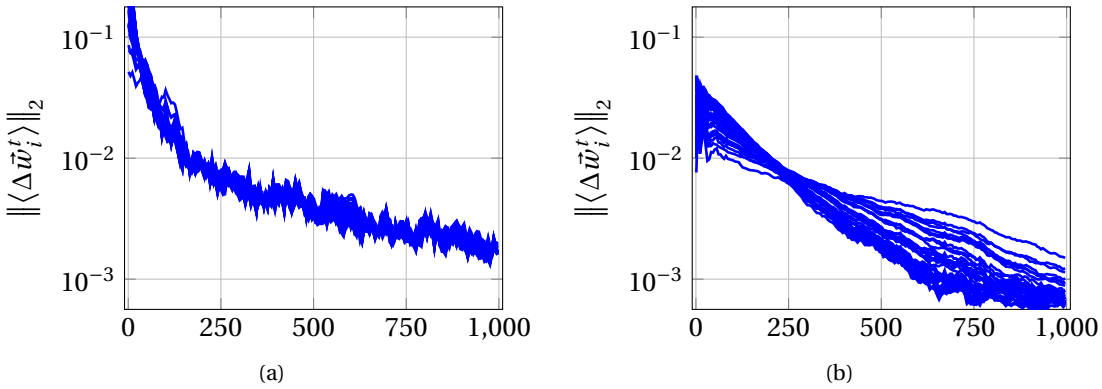


Figure 0.1: Each blue line represents the 2-norm of the average value $\left\| \langle \Delta\vec{w}_i^t \rangle \right\|_2$ for a prototype i at the t'th update. To estimate the average value at t for a given prototype i, a trail of the past values of $\Delta\vec{w}_i^t$ is used. The above two tests where both made using 6x6 Neurons, $\sigma = 3$ and constant learning rate (a) $\sigma = 3$, $\eta = 0.1$ (b) $\sigma = 3$, $\eta = 0.01$

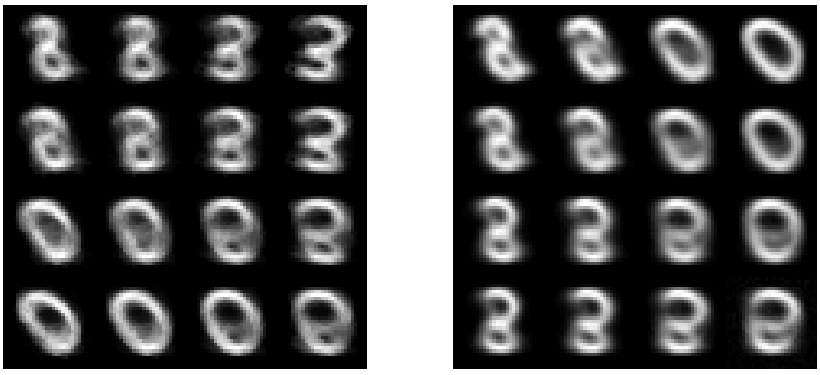
silliness, we only estimate the average using a fixed trail $t \in [T-n, T]$ for some $n \in \mathbb{N}^+$, say 100 or so. As the set from which the average is computed now does not grow, this implies that the sequence $\left(\langle \Delta \vec{w}_i^T \rangle\right)_T^\infty$ has no limit, e.g. it is divergent, however it is reasonable to assume that the amplitude of the oscillations in the limit $t \to \infty$ are small, and at least the computational cost of estimating $\langle \Delta \vec{w}_i^T \rangle$ is now bounded. To further limit the combined computational cost of checking for convergence, we only perform the check every 100 learning steps t. As for the limit that constitutes sufficient convergence, when is small sufficiently small? For this we choose, somewhat arbitrarily, that once an ϵ fraction of the largest $\left\|\Delta \vec{w}_i^t\right\|$ over all preceding learning steps t is reached, for every prototypes i, then we have achieved convergence, i.e., we write our convergence criteria as

$$\left\|\langle \Delta \vec{w}_i^T \rangle\right\| < \epsilon \max_{t \in [1,T]} \left\|\Delta \vec{w}_i^t\right\|, \quad \forall i \tag{1.2}$$

Since our estimator for $\langle \Delta \vec{w}_i^T \rangle$ is not consistent in the sense that it does not have the correct limit, it is possible to choose an ϵ too tight so that the algorithm never converges, therefore we introduce an upper bound on the number of learning steps, say $T = 20.000$, so that the loop is guaranteed to break at some point. In the sections to follow we observe choosing ϵ in 1%-10% appears to work well, depending on the choice of σ. Another simple idea on how to construct a convergence criteria is to use the labeling and accuracy test of the prototypes that we develop in section 3. Once the measured accuracy changes less than ϵ percentage between two consecutive tests, it may be a good time to stop the algorithm. The downside to this approach is that it is a computationally expensive test, and the check may thus be performed less frequent.

2 Visualizing the Prototypes

As requested in the project description we visualize the prototypes created, see figure 1.1. We use the name "Allan Svejstrup Nielsen" to generate our target digits, these being 0, 3, 5 and 8 respectively. Figure 1.1a shows the prototype of a kohonen map containing 4x4 Neurons after the learning process with constant $\sigma = 1$ and $\eta = 0.1$, in figure 1.1b the same but with the constant learning rate $\eta = 0.01$. The results are somewhat disappointing, some prototypes look like 8, some like 0, some like 3, but a lot of the prototypes are blurry to the extend that it is not clear in the eyeball number to which number they represent. Indeed in the following section we see that the two example in figure 1.1 only achieve around 60-65% accuracy when categorizing the data-set. Fortunately, as we will see later in section 4 and 5, by decreasing the width of the neighborhood function and the rate of learning η during the learning process, we can construct maps with a substantially higher accuracy.



(a) (b)

Figure 1.1: Check every 50 step for converging. Using criteria section 1, with epsilon = 3%. Visualizing the prototypes form a 4x4 map Kohonen, constant learning rate $Sigma = 1$ and (a) $eta = 0.1$, T=550 (b) $eta = 0.01$ T = 4650

3 Labeling & Evaluation

All label information is accessible via the file "label.txt". We use a majority-decision policy to assign labels to prototypes, the remaining patterns, not in majority, assigned to a prototype are considered wrongly clustered. We define the accuracy as the ratio of correctly classified patterns to the total number of patterns and use this as our prime indicator of the performance of the clustering. In figure 3.1 the accuracy as a function of neurons is presented when using a constant width σ of the neighborhood function and constant learning rate σ on the target digits set $0,3,5,8$. We note that using a smaller width σ with a large number of neurons perform better.

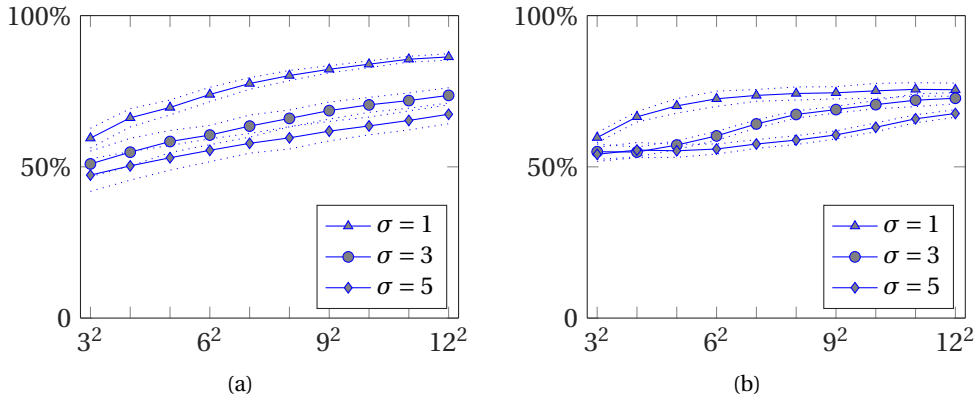


Figure 3.1: Accuracy as a function of number of Neurons when (a) $\eta = 0.1$ and (b) $\eta = 0.01$

4 Modifying the Neighborhood Function

Thus fare we have kept the width of the Gaussian neighborhood function constant, with limited success. We now test a version of the learning process where after N_σ number of learning steps, σ is made half as big, i.e., the width decreases over the course of the learning process. We choose $N_\sigma = 200$, the result of which is shown in figure 4.1, again using a monte-carlo style simulation with 20 instances to construct a mean and standard deviation around it. We note here that in particular for $\eta = 0.1$, reducing the width gradually substantially improves the quality of the clustering compared to that presented in figure 3.1 for a constant neighborhood function width. For $\eta = 0.01$ the results are less encouraging, though this may be because $N_\sigma = 200$ was chosen too small for such a slow learning rate.

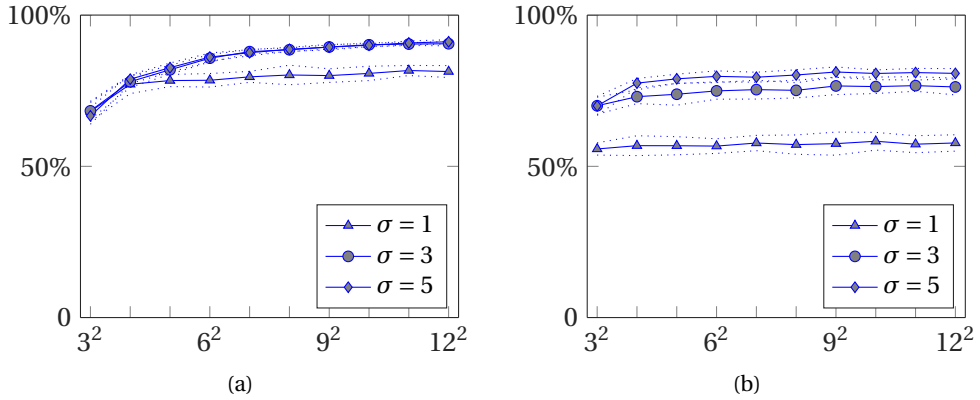


Figure 4.1: Modifying σ during the learning rate. During the first $N_\sigma = 200$ learning steps, σ as indicated in the legend is used. For each subsequent interval of N_σ learning steps t, σ is divided by two. Using constant learning rate (a) $\eta = 0.1$ and (b) $\eta = 0.01$

5 Modifying the Learning Rate

So far we have kept the learning rate a constant. In this section we present a few experiments on a Kohonen map where the learning rate decreases over the course of the learning process as this supposedly may improve the clustering. We modify the online learning rule by letting $\eta(i) = \frac{1}{N_i+1}$ where N_i is the number of patterns already associated with a given weight i. In figure 5.1a, the accuracy for a map constructed in this manner is presented. Another way of performing the learning process is so-called "batch learning", applicable if we already have a fixed set of patterns on which we want to perform clustering. Batch learning in a Kohonen network is done trough the following weigh update

$$\vec{w}_i^{t+1} = \frac{\sum_j \left(\Lambda(i,j) \sum_{\vec{x}_k \in C_j} \vec{x}_k\right)}{\sum_j \Lambda(i,j) N_j} \tag{5.1}$$

where $\Lambda(i,j)$ is the neighborhood function, $\vec{x}_k$ is the k'th data instance, $\vec{w}_i^{t+1}$ is the i'th prototype at learning step $t+1$ and N_j denotes the number of instances in cluster C_j. The result of using the batch learning process is presented in figure 5.1b. We note that the two as expected are pretty much equivalent.

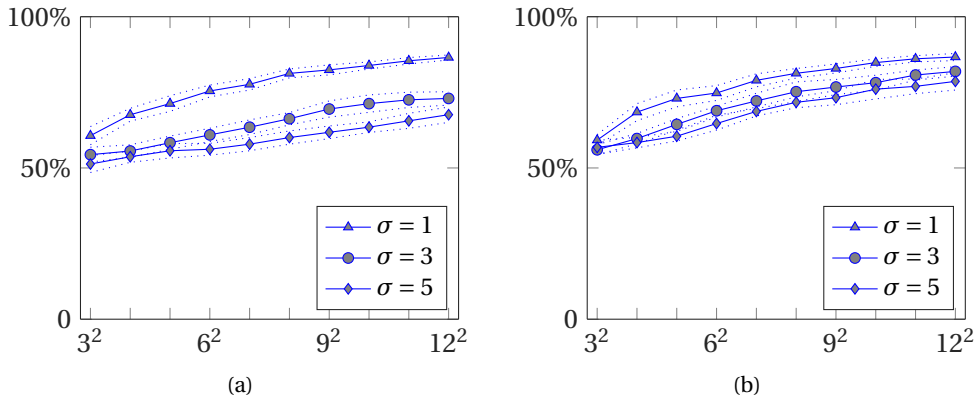


Figure 5.1: The accuracy of the clustering measured as a function of neurons in the Kohonen network. The learning process performed using (a) Batch learning, update as eq. 5.1 or (b) Online-learning with $\eta_{(i)} = \frac{1}{N_i+1}$

6 Summary

We looked at various aspects of the learning process in Kohonen networks and found that decreasing the width σ of the neighborhood function or decreasing the learning rate η during the learning process resulted in better clustering. For the case of constant η and σ on a square 6x6 neural network, the highest level of accuracy reached was 74%, see figure 3.1, using $\sigma = 1$ and $\theta = 0.1$. When modifying the learning rate η as presented in figure 5.1, for the 6x6 neural network the performance increased slightly to 76%. What really made a big difference though was to decrease the width of the neighborhood function over the course of the learning, doing so on the 6x6 kohonen map we reached 86% accuracy, see 4.1a. Throughout the study, we were using the target digest $\{0,3,5,8\}$ generated by the name "Allan Svejstrup Nielsen". It turns out that this is a fairly difficult combination, as when we tested the digits $\{0,2,4,7\}$ generated by "Chen Liu", the accuracy easily went up to around 95% when σ was allowed to decrease over the course of the learning process. Unsurprisingly, how well the input patterns are separated is thus an important factor in the quality of the clustering.

Reinforcement learning in Neural Networks

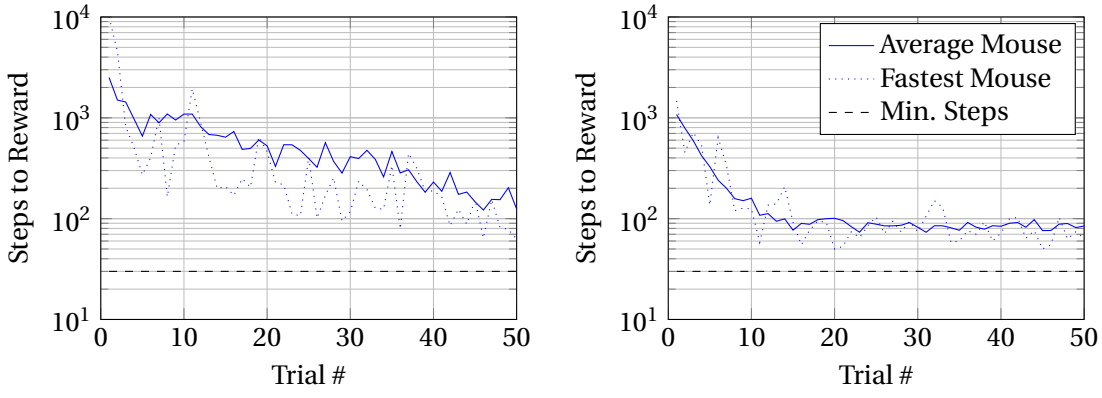
Allan S. Nielsen, Doctoral student in Mathematics
Chen Liu, 1st year student M.Sc. Computer Science
January 8, 2016

1 Implementation

In this project we are asked to make a "computational rat" that searches a map for a reward. To do so, we implement a reinforcement learning algorithm called SARSA. The state space is continuous and uniformly spanned by 20×20 discrete prototypes. When the rat is in the position $\mathbf{s}$, the weight of prototype $\mathbf{s}_j$ is given by $r_j(s) = exp(-\frac{\|\mathbf{s}-\mathbf{s}_j\|}{2\sigma^2})$. At every state, the rat has 8 possible actions, it may move in any of the directions $\frac{2\pi k}{8}$, $0 \leq k < 8$. Our learning model therefore has 400 inputs and 8 outputs, that is, 400×8 trainable parameters in total. Besides σ and ϵ the hyper-parameters in our program includes learning rate η, reward discount rate γ, eligibility trace decay rate λ and a boolean variable controlling whether or not to let ϵ decay over time.

2 Learning Curves

We measure and present the learning curves in figure 2.1 as asked in the project description. We note that the rats become better at finding the reward over time, but the progress appear to stagnate at some point. This is expected given the fixed exploration constant of 0.5, i.e., even if the rat at some point achieved perfect knowledge of the optimal direction, it would still spend a long time finding the reward since every two steps it will choose to diverge from the optimal course on average. Later in section 4 we will investigate this topic of exploration vs exploitation.

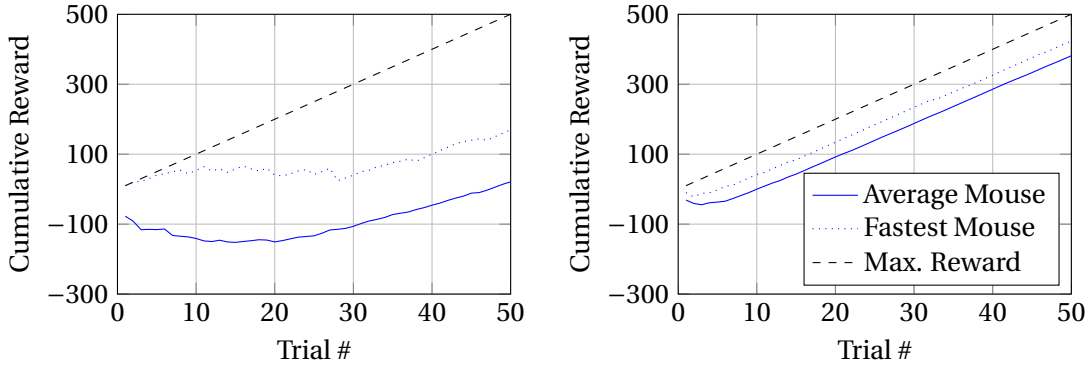


(a) (b)

Figure 2.1: Learning curves as measured when using parameters $\gamma = 0.95$, $\sigma = 0.05$ and constant $\epsilon = 0.5$ with (a) $\eta = 0.005$ and (b) $\eta = 0.05$.

3 Cumulative Rewards

Instead of looking at the time it takes to find the reward zone, we now investigate the total reward an average rat accumulates over the course of up to 50 trials. Using the same code that generated the data presented in figure 2.1, we present the accumulated reward over trials in figure 3.1. The results appear to be consistent with what observed when plotting the learning curves in the previous section. In the previous section we found that using a learning rate $\eta = 0.05$ resulted in the rats finding the reward quicker, in figure 3.1 we note that the cumulative reward is also higher for $\eta = 0.05$. However, we also observe in figure 2b that the slope converges to the optimal slope of a reward of 10 in each trial, that is, the rat is continuously able to find the reward, event though as seen from figure 2.1b it does so slowly.



(a) (b)

Figure 3.1: Cumulative rewards as a function of trials, averaged over 10 rats. All parameters as in section 2. (a) $\eta = 0.005$ and (b) $\eta = 0.05$.

4 Exploration & Exploitation

In figure 4.1, we present learning curves as previously done in figure 2.1, but now for three different exploration parameters, 0.1, 0.5 and 0.9 respectively. For trivial reasons, a fixed exploration parameter may be a bad idea. In figure 4.2 we present learning curves when ϵ is initially set to 0.5, and then halved every 5 trials. In doing so, the rats are much better at finding the reward, we note that in figure 4.2b, the fastest rat finds the reward in only 30 steps, which is the smallest number of steps possible.

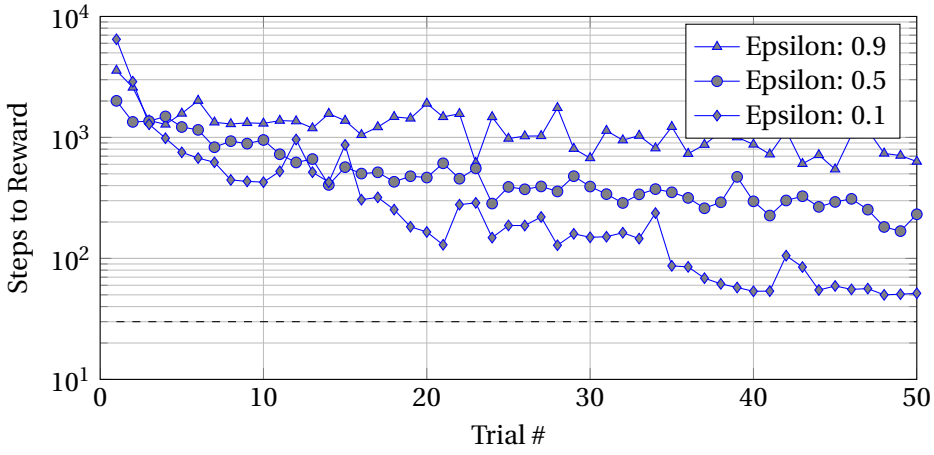


Figure 4.1: Learning curves for three exploration parameters $\epsilon = \{0.1, 0.5, 0.9\}$, learning rate $\eta = 0.005$ and all other parameters as in figure 2.1.

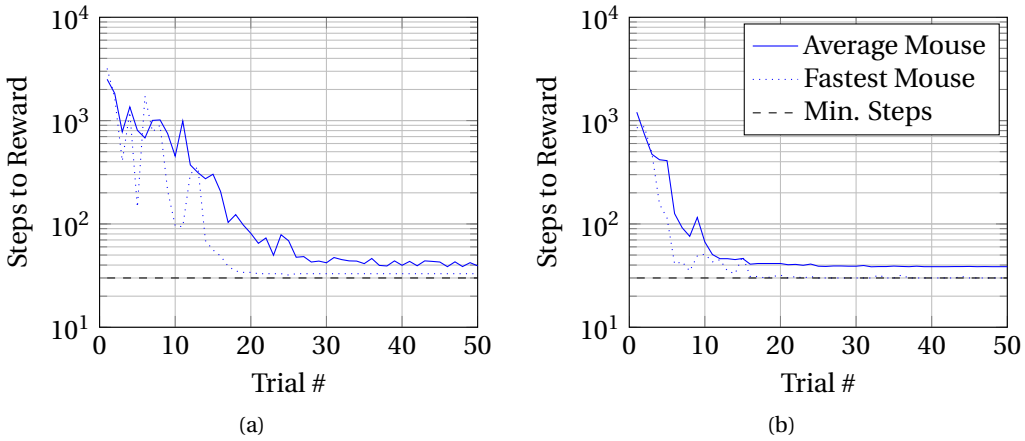


Figure 4.2: Learning curves for decaying ϵ. Note the massive improvement compared to figure 2.1. All parameters as in figure 2.1 and (a) $\eta = 0.005$ and (b) $\eta = 0.05$.

5 The Navigation Map

One way of visualizing the preferred direction of the rat would be to the draw the 20x20 grid, and at the state of each cell-center draw a vector pointing in the direction corresponding to the action leading to a new state with the largest Q. Since this is a visualization of the decision map at a given stage of learning, one could argue that it is a fair representation of the "preferred direction". However, with this approach, we ignore information regarding the value of moving in other directions that has also been accumulated over the course of the learning process.

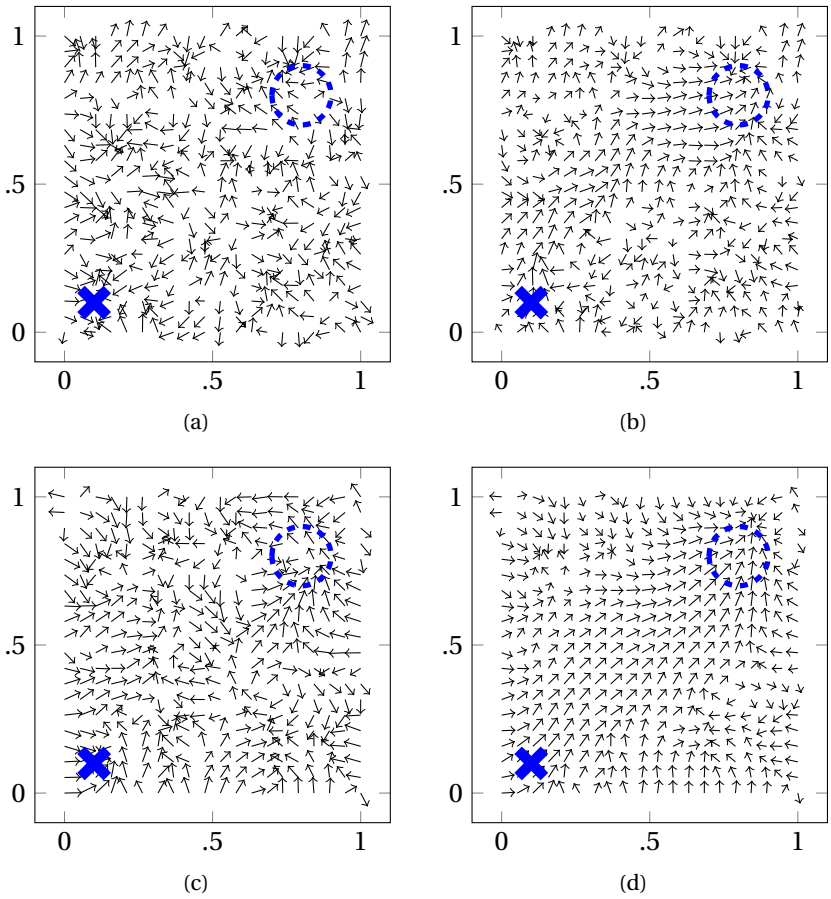


Figure 5.1: Preferred direction $\vec{D}_{ij}$. (a) A single rat at the 5th trial. (b) A single rat at the 50th trial. (c) Averaged over 10 rats at the 5th trial. (d) Averaged over 10 rats at the 50th trial.

Therefore, we take a different approach at visualizing the preferred direction, henceforth denoted by $\vec{D}_{ij}$. Instead of letting the vectors point in the direction of the action a_k for which $\max_{1\leq k\leq 8} Q(s,a_k)$, we sum all 8 direction vectors, with the length of each vector equal to $Q(S_{ij},a_k)$. I.e., for a state S_{ij}, we write the preferred direction vector $\vec{D}_{ij}$ as

$$\vec{D}_{ij} = \sum_{k=1}^{8} \vec{v}_k Q(S_{ij},a_k), \quad 1 \leq i,j \leq 20 \tag{5.1}$$

Where $\vec{v}_k$ is the normalized direction vector associated with an action a_k. In figure 5.1, we present the preferred direction maps as they appear when generated by the algorithm with decreasing ϵ as introduced in the previous section. In figure 5.1a, we note that a single rat is still pretty confused after 5 trials, though as it is clear from figure 5.1b, the rat is able to almost directly find the reward after 50 trials. From the two figures 5.1c and 5.1d, we also note that the collective knowledge of 10 rats is much better than that of a single rat.

In figure 5.2, we present a slightly more challenging case for our rats to try. We impose a trap for our poor rat. This in the form of a circle of radius 0.1 centered at $(0.5,0.5)$. Every time a rat perform an action that leads it to enter the circle, or stay there, it receives a negative reward of -5. From figure 5.2a we note that as before, after 5 trials, the rat is still somewhat confused as to what direction to move in so to get the reward, though it appears that it has already developed some knowledge on the whereabouts of the trap. In figure 5.2b, we see that after 50 trials the rat is fully aware of the presence of a trap in the center of the map, and it knows that moving to the right around the trap will lead it to the reward. As also noted in the trap-less test case, the collective knowledge of 10 rats appear to be much better than that of a single rat, see figures 5.2c and 5.2d.

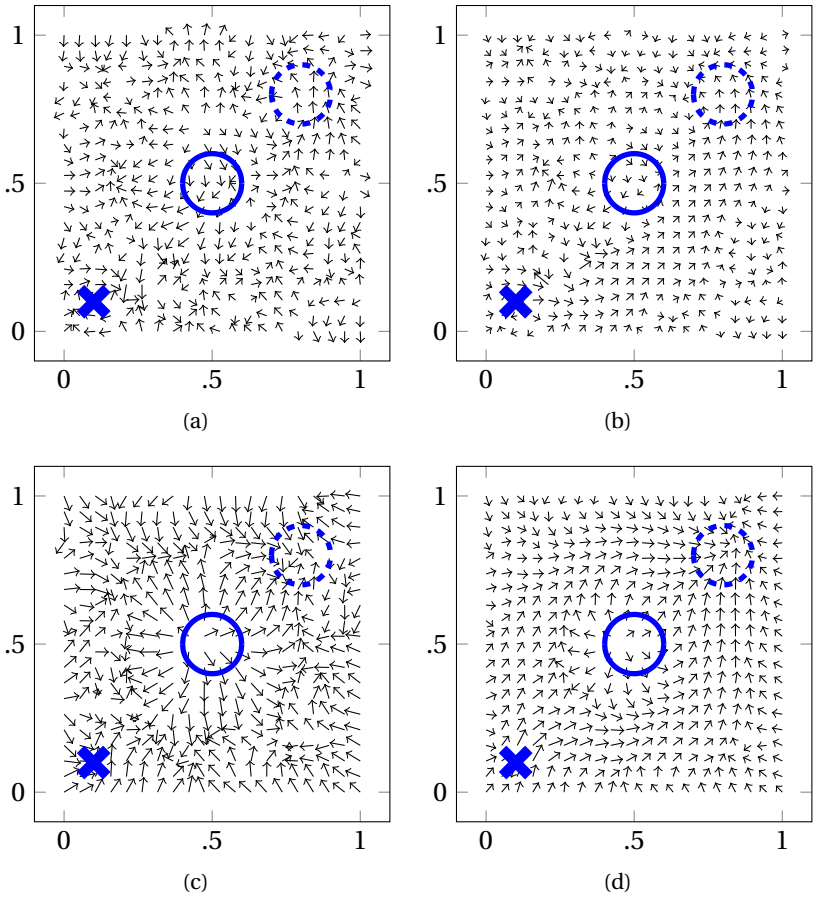


Figure 5.2: Preferred direction $\vec{D}_{ij}$. (a) A single rat at the 5th trial. (b) A single rat at the 50th trial. (c) Averaged over 10 rats at the 5th trial. (d) Averaged over 10 rats at the 50th trial.