Unsupervised Modeling of Partially Observable Environments

Vincent Graziano, Jan Koutník, and Jürgen Schmidhuber

IDSIA, SUPSI, University of Lugano,
Manno, CH-6928, Switzerland
{vincent,hkou,juergen}@idsia.ch

Abstract. We present an architecture based on self-organizing maps for learning a sensory layer in a learning system. The architecture, *temporal network for transitions* (TNT), enjoys the freedoms of unsupervised learning, works on-line, in non-episodic environments, is computationally light, and scales well. TNT generates a predictive model of its internal representation of the world, making planning methods available for both the exploitation and exploration of the environment. Experiments demonstrate that TNT learns nice representations of classical reinforcement learning mazes of varying size (up to 20×20) under conditions of high-noise and stochastic actions.

Keywords: Self-Organizing Maps, POMDPs, Reinforcement Learning.

1 Introduction

Traditional reinforcement learning (RL) is generally intractable on raw high-dimensional sensory input streams. Often a sensory input processor, or more simply, a sensory layer is used to build a representation of the world, a simplifier of the observations of the agent, on which decisions can be based. A nice sensory layer produces a code that simplifies the raw observations, usually by lowering the dimensionality or the noise, and maintains the aspects of the environment needed to learn an optimal policy. Such simplifications make the code provided by the sensory layer amenable to traditional learning methods.

This paper introduces a novel unsupervised method for learning a model of the environment, *Temporal Network Transitions* (TNT), that is particularly well-suited for forming the sensory layer of an RL system. The method is a generalization of the Temporal Hebbian Self-Organizing Map (THSOM), introduced by Koutník [7]. The THSOM places a recurrent connection on the nodes of the Self-Organizing Maps (SOM) of Kohonen [5]. The TNT generalizes the recurrent connections between the nodes in the THSOM map to the space of the agent's actions. In addition, TNT brings with it a novel aging method that allows for variable plasticity of the nodes. Theses contributions make SOM-based systems better suited to on-line modeling of RL data.

The quality of the sensory layer learned is dependent on at least (I) the representational power of the sensory layer, and (II) the ability of the RL layer get the agent the sensory inputs it needs to best improve its perception of the world.

D. Gunopulos et al. (Eds.): ECML PKDD 2011, Part I, LNAI 6911, pp. 503–515, 2011.

There are still few works where unsupervised learning (UL) methods have been combined with RL [1,8,4]. Most of these approaches deal with the UL *separately* from the RL, by alternating the following two steps: (1) improving the UL layer on a set of observations, modifying its encoding of observations, and (2) developing the RL on top of the encoded information. (1) involves running UL, without being concerned about the RL, and (2) involves running RL, without being concerned about the UL. It is assumed that the *implicit* feedback inherent in the process leads to both useable code and an optimal policy. For example, see the SODA architecture, introduced by Provost et al. [12,11], which first uses a SOM-like method for learning the sensory layer and then goes on to create high-level actions to transition the agent between internal states. Such approaches mainly ignore aspect (II). Still fewer approaches actually deal with the mutual dependence of the UL and RL layers, and how to solve the bootstrapping problem that arises when integrating the two [9].

The TNT system immediately advances the state-of-the-art for SOM-like systems as far as aspect (I) is concerned. In this paper we show how the TNT significantly outperforms the SOM and THSOM methods for learning state representations in noisy environments. It can be introduced into any pre-existing system using SOM-like methods without any fuss and will generate, as we shall see in Section 4, nice representations of challenging environments. Further, the TNT is a natural candidate for addressing both aspects (I) and (II) simultaneously, since a predictive model of the environment grows inside of it. We only touch on this in the final section of the paper, leaving it for a further study.

In Section 2 we describe the THSOM, some previous refinements, the Topological THSOM (T^2HSOM) [2], as well as some new refinements. Section 3 introduces the TNT, a generalization of the T^2HSOM architecture that is suited for on-line modeling noisy high-dimension RL environments. In Section 4 we study the performance of the TNT on partially observable RL maze environments, including an environment with 400 underlying states and large amounts of noise and stochastic actions. We make direct comparisons to both SOM and T^2HSOM methods. Section 5 discusses future directions of research.

2 Topological Temporal Hebbian Self-Organizing Map

We review the original THSOM and T^2HSOM architectures here, along with some refinements which appear for the first time here. This helps to introduce the TNT, and aids comprehension when the three methods are compared in Section 4. Briefly, a THSOM is a Self-organizing Map (SOM) that uses a recurrent connection (trained using a Hebbian update rule) on its nodes.

The THSOM consists of N nodes placed at the vertices of a finite lattice in Euclidean space. Each node i has a prototype vector $\boldsymbol{w}_i \in \mathbb{R}^D$, where D is the dimension of the observations. In what follows, the variable for the time step, t, is suppressed only when its omission cannot create confusion.

2.1 Network Activation

The activation $y_i(t)$ of node i at step t consists of a spatial component, $y_{i,S}(t)$, and a temporal component, $y_{i,T}(t)$. For observation $\boldsymbol{x}(t)$ at time t the spatial activation is

$$y(t)_{S,i} = \sqrt{D} - \|\boldsymbol{x}(t) - \boldsymbol{w}_i(t)\|, \tag{1}$$

and the temporal activation is

$$y(t)_{T,i} = \boldsymbol{y}(t-1) \cdot \boldsymbol{m}_i(t), \tag{2}$$

where $\boldsymbol{y}(t-1)$ is the normalized network activation from the previous time step, and $\boldsymbol{m}_i(t)$ is row i of the N x N temporal weight matrix M, whose entry $m_{i,j}$ represents the strength of the connection from node j to node i.

In the original formulation of the THSOM, the network activation, before normalization, was given by $y_i = y_{S,i} + y_{T,i}$. The factor $\sqrt{D}$ in the spatial activation was used to help equalize the influence the spatial and temporal components. Experimental evidence has shown that the network often benefits from a finer weighting of the two components. A parameter η is introduced to alter the balance between the components,

$$y_i = \eta y_{S,i} + (1-\eta) y_{T,i}. \tag{3}$$

The offset $\sqrt{D}$ in the spatial component (obviated by η) is kept for historical reasons, so that when $\eta = 0.5$ the original balance is recovered. Putting $\eta = 1$ makes the activation of the map, and therefore also the learning (as we shall later see), entirely based on the spatial component, i.e., a SOM. In the case of a deterministic HMM (a cycle of underlying states) a perfectly trained THSOM can 'predict' the next underlying state blindly, that is, without making use of the spatial component, $\eta = 0$, when given the initial underlying state.

Normalizing the Activation. It is necessary to normalize the activations with each step to maintain the stability of the network, otherwise the values become arbitrarily large. In the original formulation of the THSOM the activation of each node y_i was normalized by the maximum activation over all nodes, $y_i = y_i / \max_k y_k$. Normalization is now done using the *softmax* function and a parameter τ for the temperature. The temperature decreases as the network is trained; training provides an increasingly accurate model of the transition matrix, and a cooler temperature allows the network to make good use of the model.

2.2 Learning

The spatial and temporal components are learned separately. The spatial component is trained in the same way as a conventional SOM and the training of the temporal component is based on a Hebbian update rule. Before each update the node b with the greatest activation at step t is determined. This node

$b(t) = \arg\max_k \mathbf{y}_k(t)$ is called the *best matching unit* (BMU). The *neighbors* of a node are those nodes which are nearby on the *lattice*, and are not necessarily the nodes with the most similar prototype vector.

Learning the Spatial Component. The distance d_i of each node i on the lattice to b is computed using the Euclidean distance, $d_i = \|c(i) - c(b)\|$, where $c(k)$ denotes the location of node k on the lattice. The BMU and its neighbors are trained in proportional to the *cut*-Gaussian spatial neighborhood function c_S, defined as follows:

$$c_{S,i} = \begin{cases} \exp\left(-\frac{d_i^2}{2\sigma_S^2}\right) & \text{if } d_i \leq \nu_S, \\ 0 & \text{if } d_i > \nu_S, \end{cases} \tag{4}$$

where σ_S and ν_S are functions of the *age* of the network (discussed in Section 3.3). Typically, these functions are monotone decreasing with respect to t. The value of σ_S determines the shape of the Gaussian and ν_S is the topological neighborhood cut-off. The prototype vector corresponding to node i is updated using the following rule:

$$\boldsymbol{w}_i(t+1) = \boldsymbol{w}_i(t) + \alpha_S c_{S,i} \left(\boldsymbol{x}(t) - \boldsymbol{w}_i(t)\right), \tag{5}$$

where α_S is the learning rate, and is also a function of the age of the network.

Learning the Temporal Component. To learn the temporal weight matrix M, the BMUs $b(t-1)$ and $b(t)$ are considered and three Hebbian learning rules are applied:

1. Temporal connections from node $j = b(t-1)$ to $i = b(t)$ and its neighbors are strengthened (see Figure 1(a)):

$$m_{i,j}(t+1) = m_{i,j}(t) + \alpha_T c_{T,i} \left(1 - m_{i,j}(t)\right). \tag{6}$$

2. Temporal connections from all nodes j, except $b(t-1)$, to $i = b(t)$ and its neighbors are weakened (see Figure 1(b)):

$$m_{i,j}(t+1) = m_{i,j}(t) - \alpha_T c_{T,i} m_{i,j}(t). \tag{7}$$

3. Temporal connections from $j = b(t-1)$ to all nodes outside some neighborhood of $b(t)$ are weakened (see Figure 1(c)):

$$m_{i,j}(t+1) = m_{i,j}(t) - \alpha_T (1 - c_{T,i}) m_{i,j}(t). \tag{8}$$

The temporal neighborhood function $c_{T,i}$ is computed in the same way the spatial neighborhood $c_{S,i}$ (Equation 4) is except that temporal parameters are used. That is, the temporal learning has its own parameters, σ_T, ν_T, and α_T, all of which, again, are functions of the age of the network.

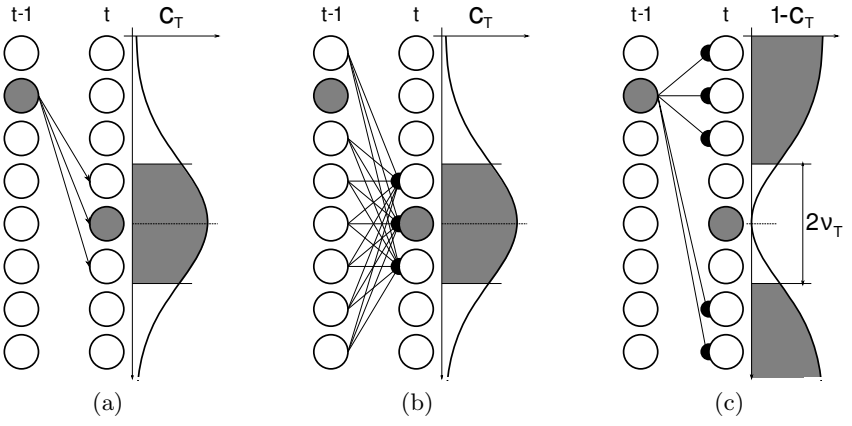


Fig. 1. T^{2}HSOM learning rules: The BMU nodes are depicted with filled circles. (a) excitation of temporal connections from the previous BMU to the current BMU and its neighbors, (b) inhibition of temporal connections from all nodes, excluding the previous BMU, to the current BMU and its neighbors, (c) inhibition of temporal connections from the previous BMU to all nodes outside some neighborhood of the current BMU. The connections are modified using the values of a neighborhood function (Gaussian given by σ_T), a cut-off (ν_T), and a learning rate (α_T). Figure adapted from [2].

The original THSOM [6,7] contained only the first 2 temporal learning rules, without the use of neighborhoods. In [2], Ferro et al. introduced the use of neighborhoods for the training of temporal weights as well as rule (3). They named the extension the Topological Temporal Hebbian Self-organizing Map (T^{2}HSOM).

3 Temporal Network for Transitions

SOMs lack the recurrent connection found in the T^{2}HSOM. As a result, when using a SOM to learn an internal state-space representation for RL the following factor plays an important role: noise may conflate the observations arising from the underlying states in the environment, preventing an obvious correlation between observation and underlying state from being found. Noise can make the disambiguation of the underlying state impossible, regardless of the number of nodes used. The recurrent connection of the T^{2}HSOM keeps track of the previous observation, allowing the system to learn a representation of the environment that can disambiguate states that a SOM cannot. The theoretical representational power of the T^{2}HSOM architecture is that it can model HMMs (which can be realized as POMDPs with a single action) but not POMDPs. While the T^{2}HSOM does consider the previous observation for determining the current internal state it ignores the action that was taken between the observations. The TNT architecture extends the T^{2}HSOM by making explicit use of the action taken between observations. As a result, TNT can *in theory* model a POMDP.

3.1 Network Activation

As in the THSOM, two components are used to decide the activation of the network.

Spatial Activation. As with SOMs, the spatial matching for both the TNT and THSOM can be carried-out with metrics other than the Euclidean distance. For example a dot-product can be used to measure the similarity between an observation and the prototype vectors. In any case, the activation and learning rules need to be mutually compatible. See [5] for more details.

Temporal Activation. The key difference between the THSOM and TNT architectures is the way in which the temporal activation is realized. Rather than use a single temporal weight matrix M, as is done in the THSOM, to determine the temporal component, the TNT uses a separate matrix M_a, called a *transition-map*, for each action $a \in \mathcal{A}$. The 'temporal' activation is now given by

$$y(t)_{T,i} = \boldsymbol{y}(t-1) \cdot \boldsymbol{m}_{\boldsymbol{i}}^{\boldsymbol{a}}(t), \tag{9}$$

where $\boldsymbol{m}_{\boldsymbol{i}}^{\boldsymbol{a}}$ is row i of transition-map M_a and $\boldsymbol{y}(t-1)$ is the network activation at the previous time step (see Equation 2).

Combining the Activations. We see two general ways to combine the components: additively and multiplicatively. The formulations of the THSOM and T^{2}HSOM considered only the additive method, and can be used successfully with the TNT as well. The two activations are summed using a balancing term η, precisely as they were in Equation 3.

Since the transition-maps are effectively learning a model of the transition probabilities between the nodes on the lattice it is reasonable to interpret both the spatial and the temporal activations as likelihoods and use an element-wise product to combine them. The spatial activation roughly gives a likelihood that an observation belongs to a particular node, with better matching units being considered more likely. In the case of the Euclidean distance, the closer the observation is to a prototype vector the more likely the correspondence, whereas with the dot-product the better matched units take on values close to 1.

For example, when using a Euclidean metric with a multiplicative combination the spatial component can be realized as

$$y(t)_{S,i} = \exp\left(-\eta \left\|\boldsymbol{x}(t) - \boldsymbol{w}_i(t)\right\|\right), \tag{10}$$

where η controls how quickly the "likelihood" values drop-off with respect to the distance. This is important for balancing the spatial and temporal components when using a multiplicative method.

After the two activations are combined the value needs to be normalized. Normalization by the maximum value, the length of the activation vector, and the *softmax* all produce excellent results. In our experiments, for simplicity, we have chosen to normalize by the length of the activation vector.

The experiments in Section 4 were carried out using a multiplicative method with the spatial activation determined by Equation 10. The multiplicative approach produces excellent results, and though the additive method also produces significant results when compared to the SOM and THSOM we have found it to be less powerful than the multiplicative approach. As such, we do not report the results of the additive approach in this paper.

3.2 Learning

The learning step for the TNT is, mutatis mutandis, the same as it was for the T^2HSOM. Rather than training the temporal weight matrix M at step t, we train the appropriate transition-map M_a (which is given by the action a_t).

3.3 Aging the TNT

Data points from RL environments do not arrive in an independent and identically distributed (i.i.d.) fashion. Therefore, it is important to impart any system learning an internal representation of the world with a means to balance plasticity and stability. Prototype nodes and transition-maps that have been updated over many samples should be fairly stable, and less likely to forget while maintaining the ability to slowly adapt to a changing environment. Likewise, untrained nodes and transition-maps should adapt quickly to new parts of the environment.

To achieve this variable responsiveness a local measure of *age* or *plasticity* is required. In classical SOM-like systems the learning parameters decay with respect to the global *age* of the network. In [5], the use of individual learning rates[1], α_S, is discussed but dismissed, as they are unnecessary for the successful training of SOMs on non-RL data. Decay with respect to a global age is sufficient in batch-based settings where the ordering of the data is unimportant. Assigning an *age* to the nodes allows for a finer control of all the parameters. As a result, powerful learning dynamics emerge. See Figure 2 for examples.

Formally, training in SOM-like systems amounts to moving points P, prototype vectors $\boldsymbol{w}$, towards a point Q, the input vector $\boldsymbol{x}$, along the line connecting them:

$$P \leftarrow (1 - h)P + hQ,$$

where h is the so-called *neighborhood function*. E.g., see Equation 5. This neighborhood function, for both the spatial and temporal learning, is simply the product of the learning rate of the prototype vector, i, and the cut-Gaussian of the BMU, b,

$$h_i(t) = \alpha_i(t)c_b(t).$$

[1] Marsland [10] uses counters to measure the activity of the nodes in a growing SOM. The counters are primarily used to determine when to instantiate new nodes. They are also used to determine the learning rate of the nodes, though in a less sophisticated way than introduced here.

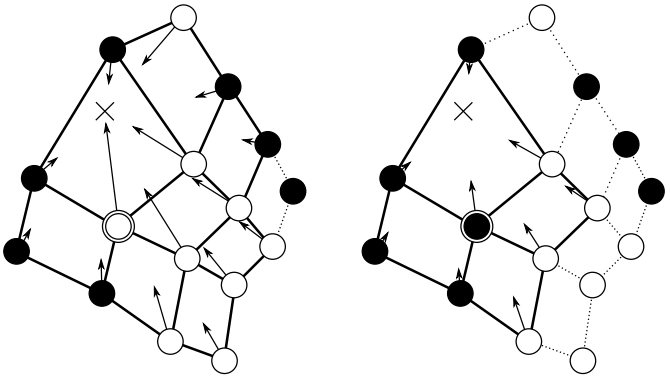


Fig. 2. Plasticity: (a) when the BMU is *young*, it strongly affects training of a large perimeter in the network grid. The nodes within are dragged towards the target (×), the *old* nodes are stable due to their low learning rate. (b) when the BMU is *old*, a small neighborhood of nodes is weakly trained. In well-known parts of the environment new nodes tend not be recruited for representation, while in new parts the young nodes, and their neighbors are quickly recruited for representation, while the older nodes are left in place.

Each node i is given a spatial age, $\xi_{S,i}$, and a temporal age $\xi^a_{T,i}$ for each action $a \in \mathcal{A}$. After determining the activation of the TNT and the BMU, b, the age of the nodes are simply incremented by the value of their spatial and temporal neighborhood functions, $h_{S,i}$ and $h_{T,i}$ respectively:

$$\xi_i(t+1) = \xi_i(t) + \alpha_i(t)c_b(t). \tag{11}$$

3.4 Varying the Parameters

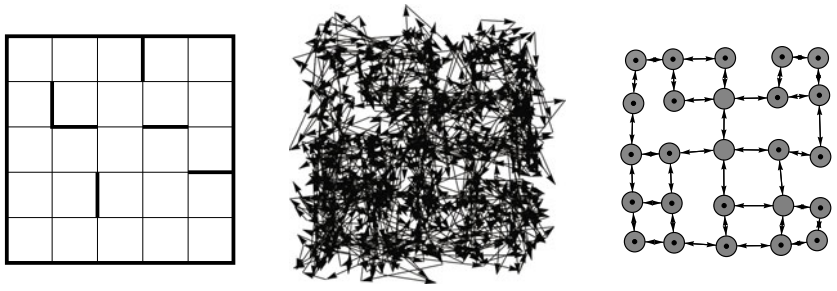
Now that the nodes have individual ages we can determine the learning rate and cut-Gaussian locally. One approach is to decay the spatial, σ_S, ν_S, α_S, and temporal, σ_T, ν_T, α_T, learning parameters using the exponential function. For each parameter Λ choose an initial value $\Lambda_\circ$, a decay rate Λ_k, and an asymptotic value Λ_∞. The value of any parameter is then determined by

$$\Lambda(\xi) = (\Lambda_\circ - \Lambda_\infty)\exp(-\xi/\Lambda_k) + \Lambda_\infty, \tag{12}$$

where ξ is the appropriate age for the parameter being used. Note that setting Λ_k to "∞" makes the parameter constant.

4 Experiments

The aim of the experiments is to show how the recurrent feedback (based on previous observations and actions) empowers the TNT to discern underlying-states from noisy observations.



(a) (b) (c)

Fig. 3. 5×5 maze experiment: (a) two dimensional maze with randomly placed walls, (b) noisy observations of a random walk ($\sigma = 1/3$ cell width), (c) trained TNT. Disks depict the location of the prototype vectors, arrows represent the learned transitions, and dots represent transitions to the same state.

4.1 Setup

The TNT is tested on two-dimensional mazes. The underlying-states of the maze lie at the centers of the tiles constituting the maze. There are four possible actions: up, down, left, and right. A wall in the direction of intended movement, prevents a change in underlying-state. After each action the TNT receives a noisy observation of the current state. In the experiments a random walk (Figure 3(b)) is presented to the TNT. The TNT tries to (1) learn the underlying-states (2-dimensional coordinates) from the noisy observations and (2) model the transition probabilities between states for each of the actions. The observations have Gaussian noise added to the underlying-state information. The networks were trained on a noise level of $\sigma = 1/3$ of the cell width, so that 20% of the observations were placed in a cell of the maze that did not not correspond to the underlying-state.

Mazes of size 5×5, 10×10, and 20×20 were used. The length of the training and random walks were 10^4, 5×10^4, and 10^5 respectively.

A parameter γ determines the reliability, or determinism, of the actions. A value of 0.9 means that the actions move in a direction other than the one indicated by their label with a probability of $1 - 0.9 = 0.1$.

The parameters used for training the TNT decay exponentially according to Equation 12. Initially, the spatial training has a high learning rate at the BMU, 0.9, and includes the 12 nearest nodes on the lattice. This training decays to a final learning rate of 0.001 at the BMU and also includes the neighboring 4 nodes. The temporal training only effects only the BMU, with an initial learning rate of 0.25 that decays to 0.001. The complete specification of the learning parameters is recorded in Table 1.

After a network is trained, disambiguation experiments are performed. The TNT tries to identify the underlying-state from the noisy observations on newly generated random walks. The amount of noise and the level of determinism are

Table 1. TNT learning parameters:

	α_S	σ_S	ν_S	α_T	σ_T	ν_T
$\Lambda_\circ$	0.90	1.75	2.10	0.25	2.33	0.5
Λ_∞	0.001	0.75	1.00	0.001	0.75	0.5
Λ_k	10	10	10	10	10	∞

varied throughout the disambiguation experiments. 10 trials of each disambiguation experiment were run, with the average results reported. Added noise was drawn from normal distributions with $\sigma = \{\frac{1}{6}, \frac{1}{3}, \frac{1}{2}, 1, \frac{3}{2}\}$, and the determinism of the actions was either 1.0 or 0.9. Putting $\sigma = 1/6$ results in only slightly noisy observations, whereas with $\sigma = 3/2$ more than 80% of the observations are placed in the wrong cell, effectively hiding the structure of the maze, see Table 2.

We compare the TNT to a SOM and a T^2HSOM on mazes, with varying levels of noise and determinism. The SOM and T^2HSOM were also trained on-line using the aging methods introduced for the TNT. We then examine the performance of the TNT on larger environments with determinism $\gamma = 0.9$, while varying the amount of noise.

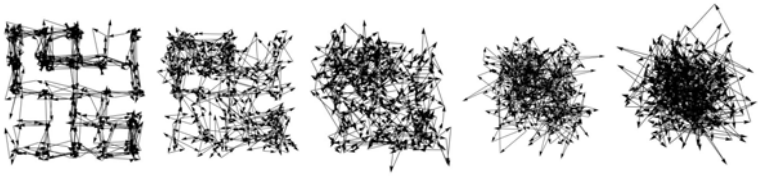
4.2 Results

In the 5×5 mazes the variable plasticity allows for the prototype vectors (the spatial component) of all three methods to learn the coordinates of the underlying states arbitrarily well. *The performance of the SOM reached its theoretical maximum while being trained on-line on non-i.i.d data.* That is, it misidentified only those observations which fell outside the correct cell. At $\sigma = 1/2$ the SOM can only identify around 55% of the observations correctly. The TNT on the other-hand is able to essentially identify 100% of all such observations in a completely deterministic environment ($\gamma = 1.0$), and 85% in the stochastic setting ($\gamma = 0.9$). Both the SOM and THSOM are unaffected by the level of stochasticity, since they do not model the effects of particular actions. See Table 2 for further comparisons.

After training, in both deterministic and stochastic settings, the transition-maps accurately reflect the transition tables of the underlying Markov model. The tables learned with deterministic actions are essentially perfect, while the ones learned in the stochastic case, suffer somewhat, see Table 3. This is reflected in the performance degradation with higher amounts of noise.

The degradation seen as the size of the maze increases is partially a result using the same learning parameters for environments of different size; the trouble is that the younger nodes need to recruit enough neighboring nodes to represent new parts of the environment while stabilizing before some other part of the maze is explored. The learning parameters need to be sensitive to the size of the environment. Again, this problem arises since we are training on-line and the data is not i.i.d. This problem can be addressed somewhat by making the ratio of nodes-to-states greater than 1. We return to this issue in Section 5.

Table 2. Observation disambiguation: For a variance of Gaussian noise relative to the maze cell width, the percentage of ambiguous observations is shown on the first line. A SOM can, at best, achieve the same performance. The second line shows the percentage of observations correctly classified by the T^2HSOM (single recurrent connection). The third line shows the percentage of observations correctly classified by the TNT when the actions are deterministic, $\gamma = 1$. The last lines gives the performance of the TNT when the actions are stochastic, $\gamma = 0.9$.



	$\sigma = 1/6$	1/3	1/2	1	3/2
SOM	99.6	79.4	**55.6**	25.5	16.6
T^2HSOM	96.6	80.4	**59.2**	27.8	17.9
TNT, $\gamma = 1$	100.0	100.0	**99.9**	99.0	98.2
TNT, $\gamma = 0.9$	96.4	94.4	**85.0**	73.1	56.0

Table 3. Observation disambiguation for larger mazes: Percentage of observations assigned to correct states in stochastic setting, $\gamma = 0.9$

	$\sigma = 1/6$	1/3	1/2	1	3/2
TNT, 5×5	96.4	94.4	**85.0**	73.1	56.0
TNT, 10×10	93.8	92.2	**77.0**	56.9	29.4
TNT, 20×20	86.8	83.6	**72.9**	45.3	23.0

It is important to note that the percentage of ambiguous observations that can actually be distinguish decreases sharply as the noise increases in stochastic settings. Simply, when the noise is sufficiently high there is no general way to disambiguate an observation following a misrepresented action from an observation following a well-represented action with high noise. See Figure 4.

5 Discussion

Initial experiments show that the TNT is able to handle much larger environments without a significant degradation of results. The parameters while they do not require precise tuning, do require that they are reasonably matched to the environment. The drop-off seen in Table 3 can be *mostly* attributed to not having used better suited learning parameters, as the same decay functions were used in all the experiments. Though the aging rules and variable plasticity have largely addressed the on-line training problem, they have not entirely solved it. As a result, we plan to explore a constructive TNT, inspired by the "grow

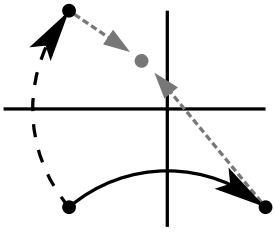


Fig. 4. The trouble with stochastic actions: The "go right" action can transition to 4 possible states. Noisy observations coupled with stochastic actions can make it impossible to discern the true state.

when required" [10] and the "growing neural gas" [3] architectures. Nodes will be added on the fly making the recruitment of nodes to new parts of the environment more organic in nature; as the agent goes somewhere new it invokes a new node. We expect such an architecture to be more flexible, able to handle a larger range of environments without an alteration of parameters, bringing us closer a general learning system.

In huge environments where continual-learning plays an increasing role, the TNT should have two additional, related features. (1) The nodes should be able to forget, so that resources might be recruited to newly visited parts of the environment, and (2) a minimal number of nodes should be used to represent the same underlying state. (1) can be solved by introducing a youthening aspect to the aging. Simply introduce another term in the aging function which slightly youthens the nodes, so that nodes near the BMU increase in age overall, while nodes further away decrease. (2) is addressed by moving from a cut-Gaussian to a similar "Mexican hat" function for the older nodes. This will push neighboring nodes away from the expert, making him more distinguished.

We have found that the Markov model can be learned when the nodes in the TNT are not in 1-to-1 correspondence with the underlying states. Simple clustering algorithms, based on the proximity of the prototype vectors and the transition-maps, are able to detect likely duplicates. An immediate and important follow-up to this work would consider continuous environments. We expect the topological mixing, inherent to SOM-based architectures, to give dramatic results.

A core challenge in extending reinforcement learning (RL) to real-world agents is uncovering how such an agent can select actions to autonomously build an effective sensory mapping through its interactions with the environment. The use of artificial curiosity [13] with planning to address this problem has been carried out in [9], where the sensory layer was built-up using vector quantization (a SOM without neighborhoods). Clearly, as we established in this paper, a TNT can learn a better sensory layer than any SOM. The transition-maps effectively model the internal-state transitions and therefore make planning methods *naturally* available to a learning system using a TNT. A promising line of inquiry, therefore,

is to derive a curiosity signal from the learning updates inside the TNT to supply the agent with a principled method to explore the environment so that a nicer representation of it can be learned.

Acknowledgments. This research was funded in part through the following grants: SNF– Theory and Practice of Reinforcement Learning (200020-122124/1), and SNF– Recurrent Networks (200020-125038/1).

References

1. Fernández, F., Borrajo, D.: Two steps reinforcement learning. International Journal of Intelligent Systems 23(2), 213–245 (2008)
2. Ferro, M., Ognibene, D., Pezzulo, G., Pirrelli, V.: Reading as active sensing: a computational model of gaze planning during word discrimination. Frontiers in Neurorobotics 4 (2010)
3. Fritzke, B.: A growing neural gas network learns topologies. In: Advances in Neural Information Processing Systems, vol. 7, pp. 625–632. MIT Press, Cambridge (1995)
4. Gisslén, L., Graziano, V., Luciw, M., Schmidhuber, J.: Sequential Constant Size Compressors and Reinforcement Learning. In: Proceedings of the Fourth Conference on Artificial General Intelligence (2011)
5. Kohonen, T.: Self-Organizing Maps, 3rd edn. Springer, Heidelberg (2001)
6. Koutník, J.: Inductive modelling of temporal sequences by means of self-organization. In: Proceeding of Internation Workshop on Inductive Modelling (IWIM 2007), pp. 269–277. CTU in Prague, Ljubljana (2007)
7. Koutník, J., Šnorek, M.: Temporal hebbian self-organizing map for sequences. In: ICANN 2006, vol. 1, pp. 632–641. Springer, Heidelberg (2008)
8. Lange, S., Riedmiller, M.: Deep auto-encoder neural networks in reinforcement learning. In: The 2010 International Joint Conference on Neural Networks (IJCNN), pp. 1–8 (July 2010)
9. Luciw, M., Graziano, V., Ring, M., Schmidhuber, J.: Artificial Curiosity with Planning for Autonomous Perceptual and Cognitive Development. In: Proceedings of the International Conference on Development and Learning (2011)
10. Marsland, S., Shapiro, J., Nehmzow, U.: A self-organising network that grows when required. Neural Netw. 15 (October 2002)
11. Provost, J.: Reinforcement Learning in High-Diameter, Continuous Environments. Ph.D. thesis, Computer Sciences Department, University of Texas at Austin, Austin, TX (2007)
12. Provost, J., Kuipers, B.J., Miikkulainen, R.: Developing navigation behavior through self-organizing distinctive state abstraction. Connection Science 18 (2006)
13. Schmidhuber, J.: Formal Theory of Creativity, Fun, and Intrinsic Motivation (1990–2010). IEEE Transactions on Autonomous Mental Development 2(3), 230–247 (2010)