A IMPLEMENTATION DETAILS AND HYPERPARAMETERS

A.1 IMPLEMENTATION

We train DTQN off-policy. Before the agent begins training, the replay buffer is seeded with 50,000 timesteps generated by a random policy. At each timestep of training, DTQN receives the agent's previous k observations, and outputs k sets of Q-values, one for each observation. During training, the beahvior policy acts in an ϵ-greedy way; that is, with probability ϵ, the agent selects an action randomly, and with probability $(1 - \epsilon)$, selects the action with the highest predicted Q value. We anneal ϵ linearly from 1.0 to 0.1 throughout the first 10% of timesteps, and then hold it fixed at 0.1 for the rest of training. The target values used for training are selected via the double DQN (Van Hasselt et al., 2016) strategy. Every 10,000 training timesteps, the target network updates its parameters by copying the policy network's current parameters. The evaluation policy behaves greedily and always selects the action with the highest predicted Q-value. We trained DTQN on a NVIDIA GeForce RTX 2070 GPU, where each run took about four hours per one million timesteps.

A.2 HYPERPARAMETERS

The full list of hyperparameters is listed in Table 2. We prioritized consistency across domains, although in the *Hallway*, *HeavenHell*, and *CarFlag* domains, we set d_{model} to 64 rather than 128.

Table 2: Hyperparameters used for our DTQN experiments.

Parameter	Setting
Heads	8
Layers	2
Context length, k	50
Embed features per observation feature	8
Model dimensionality, d_{model}	128
Target update frequency	10,000
Optimizer	Adam
Learning rate	3^{-4}
Batch size	32
Replay buffer size	500,000

A.3 ATTENTION BACKGROUND

DTQN uses a variant of attention called multi-headed scaled dot-product self-attention. Given a sequence of observations in an episode, $\{o_i\}$, we first project the observations into the model's dimensionality using an embedding layer. The attention mechanism then performs linear transformations against learnable weight matrices W^Q, W^K, and W^V to map the sequence of tokens to a sequence of queries, Q, a set of keys, K, and a set of values, V. Once we have the keys, queries, and values, we compute the attention as follows:

$$
\begin{aligned}
H &= \text{Embedding}(\{o_i\}) \\
Q = HW^Q, K &= HW^K, V = HW^V \\
\text{Attention}(Q, K, V) &= \text{softmax}(\frac{QK^T}{\sqrt{d_k}})V
\end{aligned}
$$

where d_k is the dimensionality of K. The result of the softmax above gives an attention score for each query. By splitting the weight matrices W^Q, W^K, and W^V into smaller components, we form several independent heads, hence the name multi-headed attention. The intuition behind multi-headed attention is to give each head the ability to attend to different parts of the input. The result of each

head is concatenated before the final linear projection result

$$\text{MultiHead}(Q, K, V) = \text{Concat}(head_1, ..., head_h)W^O$$
$$\text{where head}_i = \text{Attention}(QW_i^Q, KW_i^K, VW_i^V)$$

Where W_i^Q, W_i^K, and W_i^V are learnable weight matrices to embed the queries, keys, and values.

B Domain details

B.1 Classic POMDPs

We use Hallway (Littman et al., 1995) and HeavenHell (Bonet, 1998) from the classic POMDP literature. Hallway is a hallway gridworld with four rooms to the hallway's south. The agent's goal is to navigate to the fourth southern room despite very stochastic transition and observation dynamics. Due to the stochasticity, an agent must take several localizing actions before it can be certain of its surroundings. Hallway's observation space is an integer representing which walls the agent can see in its current cell (e.g. if the agent is at the end of the hallway, it would see the wall to its left, its right, and in front of it), and there are 5 actions: no-op, move forward, turn right, turn left, and turn around. HeavenHell consists of a T-shaped grid with an oracle priest at the southern end, heaven at one end of the northern fork, and hell at the other end. On each episode reset, the location of heaven and hell may be swapped, and the agent can only learn heaven's location by visiting the priest. The optimal agent first navigates south, away from its goal, to the priest, then uses the priest's observation to go to heaven. HeavenHell's observation space is an integer, representing the agent's position in the world unless the agent is visiting the priest, in which case it recieves an indicator of whether heaven is on the left or the right side of the fork. A HeavenHell agent can take one of 4 actions: move north, south, east, and west.

In Hallway, the agent receives a living reward of 0, and a reward of 1 only after it reaches the goal state. In HeavenHell, the agent receives a reward of 1 for reaching heaven, and -1 for reaching hell.

B.2 Gym-gridverse

Gym-Gridverse (Baisero & Katt, 2021) contains a set of gridworlds featuring challenging and partially observable tasks. We evaluate DTQN and our baselines on Gridverse's memory and memory-four-rooms environments. The agent's observations consist of a limited forward view based on the agent's position and orientation. In our experiments, the observation space is a vector of length 6 integers (representing the 2×3 grid as seen in Figure 3), and there are 6 actions: move forward, move backward, move left, move right, turn left, and turn right. Memory consists of a central hallway with the southern end containing identical colored beacon tiles while the northern end holds two differently colored flags – one representing heaven and one representing hell. Memory four rooms is a procedurally generated gridworld with a similar goal to Memory in that the agent must first find the colored beacon to know which colored flag to reach. However, memory four rooms is much harder than memory because the layout of the grid as well as the beacon and flag locations generated randomly on each episode reset. The agent must gather information and explore its surroundings until it knows where the color of the beacon and the location of the flags.

B.3 Memory cards

Memory cards is a new domain based on the children's card game Memory. 5 pairs of cards are randomly shuffled and their locations are hidden. At each time step, the agent sees the position of one random card (i.e. it is flipped face-up) and must guess the position of that card's pair. If guessed correctly, the pair of cards are removed from play; otherwise, the card is turned back face-down, and a new random card is revealed. This process continues until all N pairs have been removed from play. Each correct guess rewards the agent with a reward of 0, each incorrect guess a reward of -1. The observation space is a vector of 10 integers, where each integer denotes the card at that position is hidden, face-up (in which case it shows that card's value), or removed from play. The agent's action is the index for which card it thinks matches the currently revealed card. Given enough time, a random policy can solve this task. However, policies that can effectively remember their history and reason about unknown positions will perform best.

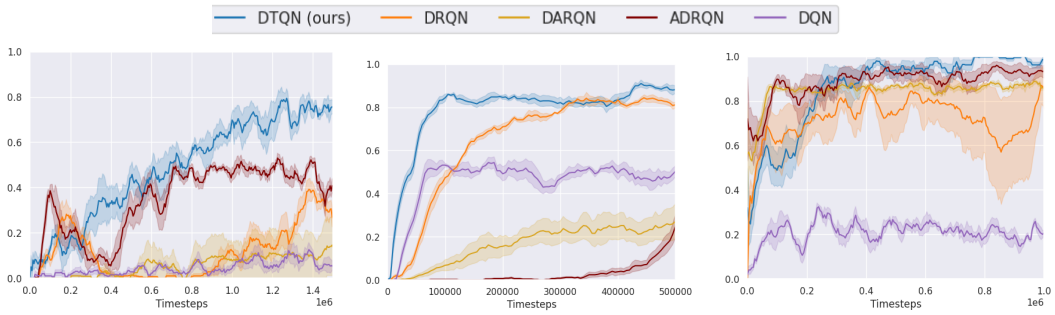
B.4 CAR FLAG

Car flag tasks a car with driving across on a 1D line to the correct flag. The car must first drive to the oracle flag and then to the correct endpoint. The agent observation is a vector of 3 floats, including its position on the line, its velocity at each timestep, and, when it is at the oracle flag, it is also informed of the goal flag's location. The agent's action alters its velocity; it may accelerate left, perform a no-op (i.e. maintain current velocity), or accelerate right. The agent receives a reward of 1 for reaching the goal flag, a reward of -1 for reaching the incorrect falg, and 0 otherwise.

C ADDITIONAL LEARNING CURVES

C.1 ADDITIONAL BASELINES

In addition to DRQN and DQN, we also compare our method to *Action-based* DRQN (ADRQN) (Zhu et al., 2017) and Deep Attention Recurrent Q-Network (DARQN) (Sorokin et al., 2015). ADRQN uses the same structure as DRQN, but conditions the Q-function on previous actions as well as observations. DARQN similarly uses the structure of DRQN, but applies attention at each step between the LSTM's last hidden state and the current observation. The results for these comparisons are shown in Figure 5. In all cases, DTQN outperforms the baselines. ADRQN performed better as a baseline than DRQN in gridverse memory 7x7 and hallway, but still performed worse than DTQN in terms of final success rate. In the Hallway domain, ADRQN and DARQN achieve high performance early, but DTQN achieves the best final success rate. Unfortunately, we were unable to run these baselines on all our domains due to compute limitations, as DARQN took 12 hours per one million timesteps (compared to DTQN's four hours per one million timesteps).



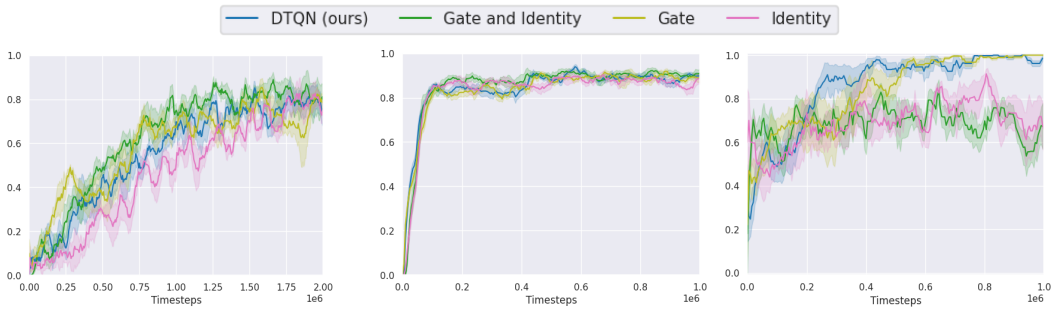
(a) Gridverse memory 7x7 (b) Memory cards (c) Hallway

Figure 5: Baseline comparison of DTQN (blue) with DRQN (orange), DARQN (yellow), ADRQN (maroon), and DQN (purple), measured by evaluation success rate during training. Lines show mean success rate and shaded regions represent standard error across five random seeds. In all three cases, DTQN achieves the highest final success rate among all five algorithms.

C.2 ABLATIONS

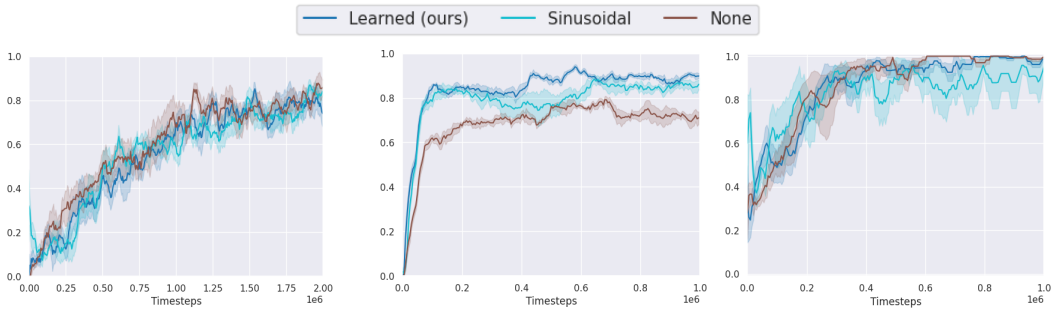
In this section, we plot the learning curves for the ablations in Table 1. Figure 6 refers to the "Transformer structure" section of Table 1. DTQN outperforms or performs competitively with all ablated versions. The learning curves show both the ablated version of DTQN with GRU-like combine step and identity map reordering as well as the version containing only identity map reordering perform much worse in the Hallway domain compared to the original DTQN and the version of DTQN with only GRU-like gating. In gridverse memory 7x7, the identity map reordering version of DTQN learns slower than the other versions. Figure 7 refers to the "Positional encodings" section of Table 1. Again, DTQN performs better than or is similar to the other forms of positional encodings. Particularly, the memory cards domain shows the benefit of learned positional encodings, as our version clearly performs the best compared to the sinusoidal encodings and the variant of DTQN with no positional encodings. For more discussion of positional encodings, refer to Appendix E. Figure 8 refers to the "Intermediate Q-value prediction" section of Table 1. In all cases, DTQN

outperforms or is competitive with our ablated versions, showing the important of the intermediate Q-value prediction training regime. For more discussion of these results, refer to Section 5.5.



(a) Gridverse memory 7x7 (b) Memory Cards (c) Hallway

Figure 6: Ablations for the "Transformer structure" section of Table 1. DTQN (blue) compared to an ablated version of DTQN consisting of both gated combine step as well as the identity map reordering (green), just gated combine step (olive), and just identity map reordering (pink). The y-axis shows the evaluation success rate, with the shaded region depicting the standard error across five random seeds. In all three cases, our version is competitive with the ablated versions of DTQN.



(a) Gridverse memory 7x7 (b) Memory Cards (c) Hallway

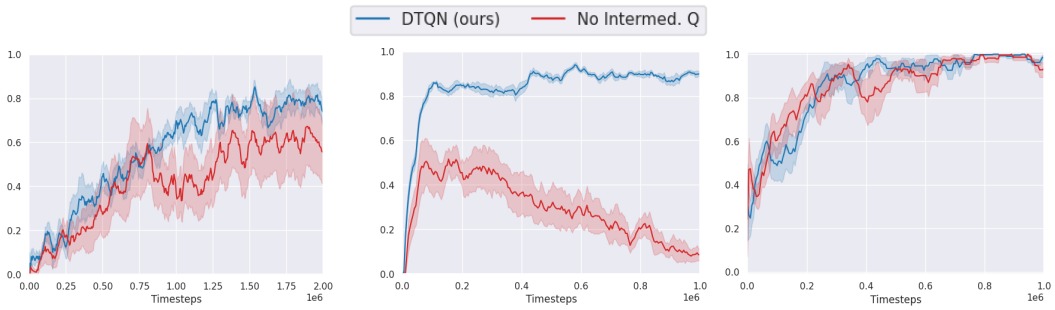
Figure 7: Ablations for the "Positional encodings" section of Table 1. Learned positional encodings, used by DTQN, is shown in dark blue, while sinusoidal positional encodings from the original transformer (Vaswani et al., 2017) is shown in light blue, and a variant of DTQN without positional encodings is colored brown. In all cases, our learned position encodings outperform or are competitive with the ablated versions.

C.3 Intermediate Q-value Analysis

One advantage presented by intermediate Q-value prediction is the denser training signal it provides to DTQN. In Figure 9, we compare our baseline DTQN agent with a variant that does not use intermediate Q-value prediction, and increase its batch size to as many as 16 times more than our baseline configuration. While increasing batch size improves performance, the only stable agent is the one utilizing intermediate Q-value prediction. By training on q-values generated with little to no context, we produce a more robust and stable agent than training only on full contexts.

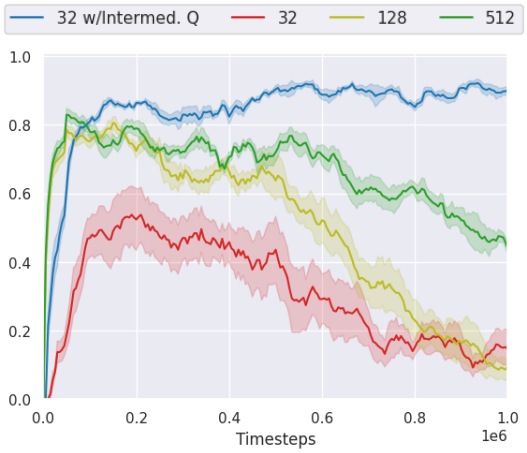
C.4 Effect of Intermediate Q-Value Prediction and Transformer Structure

In our ablations from Table 1, we compared the baseline DTQN agent with various transformer architectural changes, such as the identity map reordering and gated combine step from GTrXL Parisotto et al. (2020). In Figure 10, we add an agent using the gate and identity map reordering, but remove DTQN's Intermediate Q-value Prediction (IQP). In all three tested domains, the baseline DTQN agent achieves higher final performance than the variant with gate and identity but without IQP. This



(a) Gridverse memory 7x7 (b) Memory Cards (c) Hallway

Figure 8: Ablations for the "Intermediate Q-value prediction" section of Table 1. DTQN is shown in blue and the ablated version without using intermediate q-value prediction is shown in red. Using the intermediate Q-value prediction enables the network to learn more efficiently and robustly, which is highlighted in the memory cards domain.



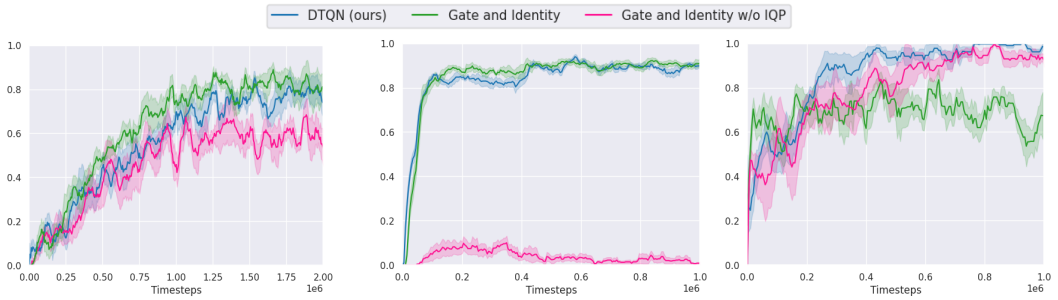
(a) Gridverse memory 7x7

Figure 9: Success rate of DTQN on the Gridverse memory 7x7 domain. Legend shows the batch size per training step used. Our baseline agent, shown in blue, outperforms the variants of DTQN without intermediate Q-value prediction, even when the variant's batch size is increased by 16 times. DTQN with intermediate Q-value prediction is more stable than without.

difference is most notable in the Memory Cards domain, where the gate and identity without IQP agent completely fails to solve the task.

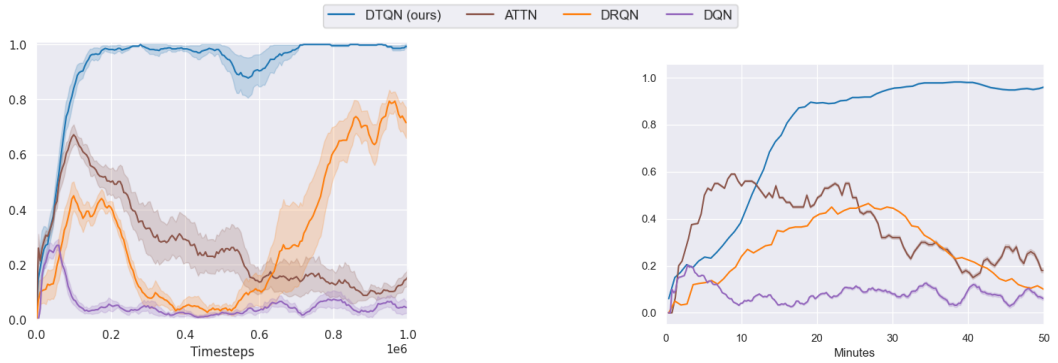
C.5 Wall-Clock Time Comparison

We also compare DTQN, DRQN, and DQN in terms of model size and speed in the Gridverse Memory 5x5 domain. Table 3 shows the model size in terms of number of parameters as well as the amount of time required to reach 1M timesteps in the environment. DTQN contains significantly more parameters than DRQN and yet reaches 1M timesteps about 5 minutes slower than DRQN. We believe this is because DTQN and the transformer are able to utilize the GPU hardware more effectively than the LSTM in DRQN. Figure 11 shows the success rate of the models in the Gridverse Memory 5x5 environment with respect to environment steps and wall-clock time. DTQN is the most sample efficient in terms of both environment steps and wall-clock time. In Figure 11b, DTQN is the only agent capable of solving the task within the 50 minutes of wall-clock time. To collect these results, we used a PC with Intel Core i7-8700K CPU @ 3.70GHz x 12 and NVIDIA GeForce RTX 2070 Rev. A GPU. Results may vary on other hardware.



(a) Gridverse memory 7x7 (b) Memory Cards (c) Hallway

Figure 10: Study of intermediate Q-value prediction (IQP) while incorporating the gating combine step and identity map reordering from GTrXL Parisotto et al. (2020). The baseline DTQN architecture, which uses IQP with residual skip connection combine step and no identity map reordering, is shown in blue, and is best or competitive in all three domains. Without the intermediate Q-value prediction, the Gate and Identity (pink) agent struggles in Gridverse memory, and completely fails to learn in Memory cards.



(a) Environment Steps (b) Wall-Clock Time

Figure 11: Comparison of model efficiency in Gridverse Memory 5x5 with respect to both environment steps and wall-clock time. DTQN, shown in blue, is more sample efficient than both DRQN (orange) and DQN (purple) with respect to both interactions with the environment and wall-clock time. To collect these results, we used a PC with Intel Core i7-8700K CPU @ 3.70GHz x 12 and NVIDIA GeForce RTX 2070 Rev. A GPU. Results may vary on other hardware.

D ATTENTION VISUALIZATIONS

Figure 12 shows attention weights for a trajectory rolled out by DTQN in the gridverse memory 5x5 domain. The agent first looks for the colored beacons, then turns to navigate toward the flags. Crucially, in its second to last observation (magnified, left, in Figure 12), the agent can only see the green flag but needs to navigate to the blue flag. It therefore attends to both the observation containing the blue beacon as well as the initial observation containing the spatial relationship of the two flags

Table 3: Model comparison on Gridverse Memory 5x5.

Model	# Parameters	Time to 1M steps (minutes)
DTQN (ours)	435,142	158.70
DRQN	155,838	153.12
DQN	23,743	48.972

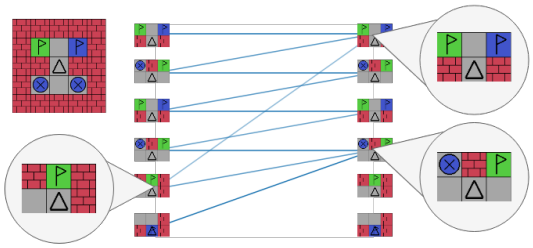


Figure 12: Attention bars for gridverse memory 5x5. The agent uses attention to remember the location of and navigate to the blue flag even when it cannot see the both flags in its second to last observation (magnified, left).

(see magnified observations on right in Figure 12) and knows to move backwards towards the blue flag to achieve its goal.

Similarly, Figure 13 displays attention weights for a trajectory rolled out by a trained DTQN agent in gridverse memory 7x7. The agent takes several actions at the beginning of the episode to localize itself. Notably, the agent encounters the yellow flag before seeing a beacon (magnified, top left in Figure 13). At this point, it does not know which flag is its goal, so it continues until it locates the red beacon (magnified, right). This observation is attended to by all future observations, indicating the model understands this beacon dictates its goal for the episode. Finally, the agent finds the red flag (magnified, bottom left), and correctly navigates to it to successfully complete the episode.

E Positional Encodings

DTQN learns positional encodings, which gives the network information regarding the temporal position of each observation in its history. In Figure 14, we examine the positional encodings DTQN learns in various domains and compare them to the sinusoidal positional encodings from the original transformer (Vaswani et al., 2017). DTQN learns unique positional encoding structures to match its domain. The positional encodings for the gridverse memory domain show high similarity scores, especially for positions 20 through 50, indicating the encodings may not be very valuable for this domain. Indeed, this notion is supported in Table 1 and Figure 7, where we see very similar performance between the DTQN agent with learned positional encodings and the variant without positional encodings. In contrast, DTQN's learned encodings in the memory card domain are very distinct, which we can see in the low similarities everywhere except the symmetric diagonal in Figure 14. This indicates that these unique positional encodings are very valuable in this domain, which is supported by the poor performance of DTQN without any positional encodings on this domain (results shown in Table 1 and Figure 7). We value the flexibility of learning unique positional encodings for each domain as it lets our model adapt to each environment.

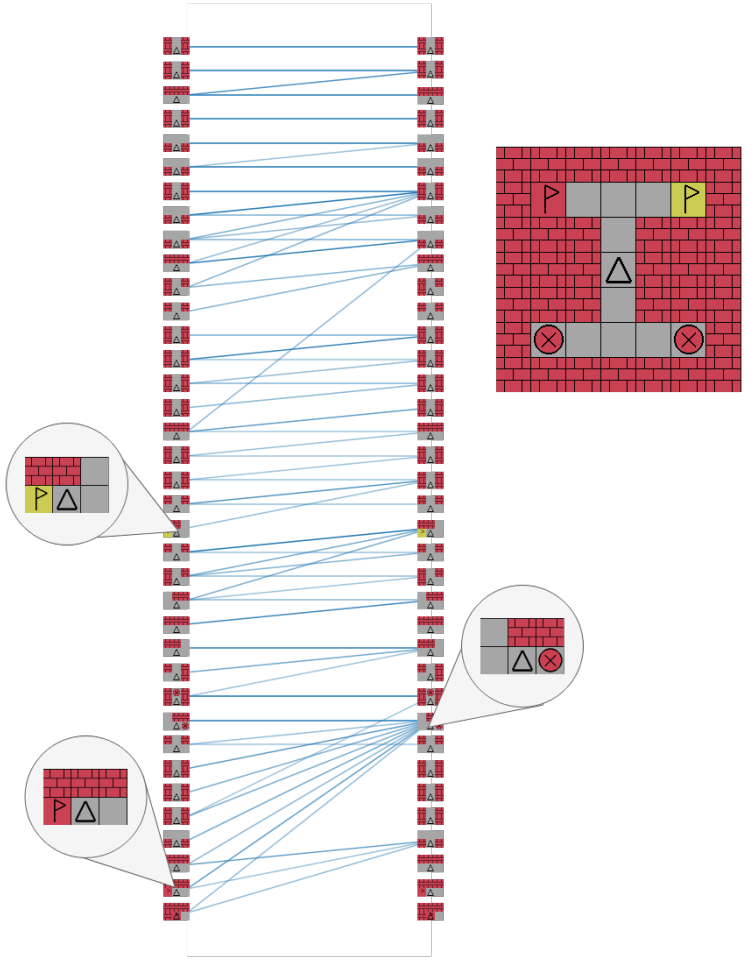


Figure 13: Attention bars for gridverse memory 7x7. Bars go from left to right, and observations go top to bottom (i.e. the second observation attended to the first and second observation). The observation containing the red beacon (magnified, right) is attended to strongly by all future observations, indicating the agent understands the beacon's importance in achieving its goal. Attention weights below 0.2 have been removed for visibility.

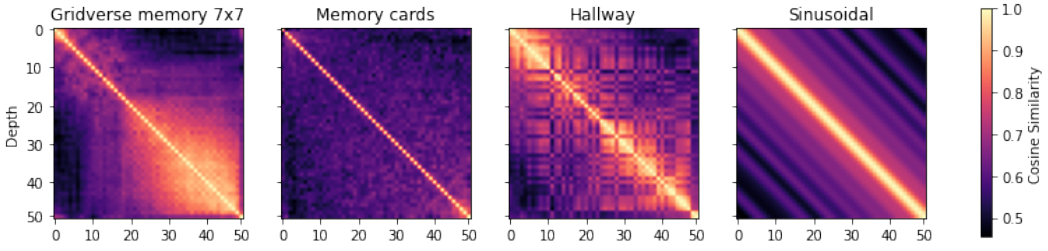


Figure 14: Cosine similarity of positional encodings from a trained DTQN agent across three domains (left), and sinusoidal positional encodings (right). The brightness at point (i, j) indicates the similarity between the ith and jth positional encoding. DTQN learns it's positional encodings uniquely for each domain.