A Supplementary Material of The Sensory Neuron as a Transformer: Permutation-Invariant Neural Networks for Reinforcement Learning

A.1 Hyper-parameters

In the table below are the hyper-parameters used for each experiment. We did not employ exhaustive hyper-parameter tuning, but have simply selected parameters that can appropriately size our models to work with training methods such as evolution strategies, where the number of parameters cannot be too large. As mentioned in the discussion section about the limitations, we tested a small range of patch sizes (1 pixel, 4 pixels, 6 pixels), and we find that a patch size of 6x6 works well across tasks.

Table 1: Summary of hyper-parameters.

	CartPole	Ant	CarRacing	Atari Pong
o_t	$\mathcal{R}^5$	$\mathcal{R}^{28}$	$\mathcal{R}^{96\times96\times4}$	$\mathcal{R}^{84\times84\times4}$
$o_t[i]$	$\mathcal{R}^1$	$\mathcal{R}^1$	$\mathcal{R}^{6\times6\times4=144}$	$\mathcal{R}^{6\times6\times4=144}$
N	5	28	$(96/6)^2 = 256$	$(84/6)^2 = 196$
$\lvert A\rvert$	1	8	3	6 (one-hot encoding)
M	16	32	1024	400
W_q	$\mathcal{R}^{8\times32}$	$\mathcal{R}^{8\times32}$	$\mathcal{R}^{8\times16}$	$\mathcal{R}^{8\times32}$
W_k	$\mathcal{R}^{8\times32}$	$\mathcal{R}^{8\times32}$	$\mathcal{R}^{111\times16}$	$\mathcal{R}^{114\times32}$
W_v	I	I	$\mathcal{R}^{144\times16}$	$\mathcal{R}^{144\times32}$
$\sigma(\cdot)$	$tanh$	$tanh$	$softmax$	$softmax$
m_t	$\mathcal{R}^{16}$	$\mathcal{R}^{32}$	$\mathcal{R}^{1024\times16}$	$\mathcal{R}^{400\times32}$

A.2 Description of compute infrastructure used to conduct experiments

For all ES results, we train on Google Kubernetes Engines (GKE) with 256 CPUs (N1 series) for each job. The approximate time, including both training and periodic tests, for the jobs are: 3 days (CartPole), 5 days (PyBullet Ant ES) and 10 days (CarRacing). For BC results, we train with Google Computing Engines (GCE) on an instance that has one V100 GPU. The approximate time, including both training and periodic tests, for the jobs are: 5 days (PyBullet Ant BC), 1 day (Atari Pong).

A.3 Detailed setups for the experiments

A.3.1 PyBullet Ant

In the PyBullet Ant experiment, we demonstrated that a pre-trained policy can be converted into a permutation invariant one with behavior cloning (BC). We give detailed task description and experimental setups here. In `AntBulletEnv-v0`, the agent controls an ant robot that has 8 joints ($|A| = 8$), and gets to see an observation vector that has base and joint states as well as foot-ground contact information at each time step (|O|=28). The mission is to make the ant move along a pre-defined straight line as fast as possible. The teacher policy is a 2-layer FNN policy that has 32 hidden units trained with ES. We collected data from 1000 test roll-outs, each of which lasted for 500 steps. During training, we add zero-mean Gaussian noise ($\sigma = 0.03$) to the previous actions. For the student policy, We set up two networks. The first policy is a 2-layered network that has the AttentionNeuron with output size $m_t \in \mathcal{R}^{32}$ as its first layer, followed by a fully-connected (FC) layer. The second, larger policy is similar in architecture, but we added one more FC layer and expanded all hidden size to 128 to increase its expressiveness. We train the students with a batch size of 64, an Adam optimizer of $lr = 0.001$ and we clip the gradient at maximum norm of 0.5.

A.3.2 Atari Pong

In the Atari game Pong, we append a deep CNN to the AttentionNeuron layer in our agent (student policy). To be concrete, we reshape the AttentionNeuron's output message $m_t \in \mathcal{R}^{400\times32}$ to $m_t \in \mathcal{R}^{20\times20\times32}$ and pass it to the trailing CNN: [Conv(in=32, out=64, kernel=4, stride=2), Conv(in=64, out=64, kernel=3, stride=1), FC(in=3136, out=512), FC(in=512, out=6)]. We use $ReLU$ as the activation functions in the CNN. We collect the stacked observations and the corresponding logits output from a pre-trained PPO agent (teacher policy) from 1000 roll-outs, and we minimize the MSE loss between the student policy's output and the teacher policy's logits. The learning rate and norm clip are the same as the previous experiment, but we use a batch size of 256.

For the occluded Pong experiment, we randomly remove a certain percentage of the patches across a training batch of stacked observation patches. In tests, we sample a patch mask to determine the positions to occlude at the beginning of the episode, and apply this mask throughout the episode.

A.3.3 CarRacing

In AttentionAgent [1], the authors observed that the agent generalizes well if it is forced to make decisions based on only a fraction of the available observations. Concretely, [1] proposed to segment the input image into patches and let the patches vote for each other via a modified self-attention mechanism. The agent would then take into consideration only the top $K = 10$ patches that have the most votes and based on the coordinates of which an LSTM controller makes decisions. Because the voting process involves sorting and pruning that are not differentiable, the agent is trained with ES. In their experiments, the authors demonstrated that the agent could navigate well not only in the training environment, but also zero-shot transfer to several modified environments.

We need only to reshape the AttentionNeuron layer's outputs to adapt for AttentionAgent's policy network. Specifically, we reshape the output message $m_t \in \mathcal{R}^{1024\times16}$ to $m_t \in \mathcal{R}^{32\times32\times16}$ such that it can be viewed as a 32-by-32 "image" of 16 channels. Then if we make AttentionAgent's patch segmentation size 1, the original patch voting becomes voting among the m_t's and thus the output fits perfectly into the policy network. Except for this patch size, we kept all hyper-parameters in AttentionAgent unchanged, we also used the same CMA-ES training hyper-parameters.

Although the simple settings above allows our augmented agent to learn to drive and generalize to unseen background changes, we found the car jittered left and right through the courses. We suspect this is because of the frame differential operation in our $f_k(o_t, a_{t-1})$. Specifically, even when the car is on a straight lane, constantly steering left and right allows $f_k(o_t, a_{t-1})$ to capture more meaningful signals related to the changes of the road. To avoid such jittering behavior, we make m_t a rolling average of itself: $m_t = (1-\alpha)m_t + \alpha m_{t-1}, 0 \leq \alpha \leq 1$. In our implementation $\alpha = g([h_{t-1}, a_{t-1}])$, where h_{t-1} is the hidden state from AttentionAgent's LSTM controller and a_{t-1} is the previous action. $g(\cdot)$ is a 2-layer FNN with 16 hidden units and a $sigmoid$ output layer.

References

[1] Y. Tang, D. Nguyen, and D. Ha. Neuroevolution of self-interpretable agents. In *Proceedings of the Genetic and Evolutionary Computation Conference*, 2020. https://attentionagent.github.io.