Appendix

More Atari Results

We provide more experimental results for Atari games. Table S3 compares different evaluation schemes for stochastic activation A3Cs. Figure S2 compares the training curves among the stochastic activation A3C variants from the main text with the baseline A3C-LSTM.

Pseudocode

Pesudo-code for baseline A3C-LSTM is at Algorithm 1, for HPA3C is at Algorithm 2, ACER at Algorithm 3, and SACER at Algorithm 4.

Atari 2600 Environment

See Table S1.

CarRacing Environment

CarRacing is a simple driving environment from the Box2D module of OpenAI Gym. The observations consist of 96×96 RGB top-view of the race car and a black bar containing information regarding speed, ABS sensor outputs per each wheel, steering wheel position and gyroscope. During training, we only receive the pixel-valued observations. The observations are normalized by subtracting mean and dividing by standard deviation and then resized to 80×80. The beginning of the game shows a top view of the entire track and slowly zooms in. As this period is irrelevant to our task, we set the zoom to a static value. The original action space consists of 3 continuous values, namely breaking (0 to 1), steering wheel left to right (-1 to 1), and acceleration (0 to 1). We discretize the continuous space to 4 general categories, namely breaking, acceleration, turning right and turning left. For each category, we further provide 3 levels of intensity in applying the action, namely soft, medium, and full-throttle, in total yielding 12 discrete action options. The maximum total score possible, if without any negative rewards, is 1000. Every tile the car clears is given $1000/N$ points where N is the total number of tiles on the track and every second used for the game is given -5 points. We also reduce the FPS from 50 to 12.5 and repeat the action 4 times, to simulate the effect of (deterministic) frame-skipping, while making the game computationally more efficient. The environment is considered to be solved when the agent consistently achieves more than 900 points and the OpenAI oracle agent scores 837 averaging over 100 games. We test on 10 randomly generated games with different random seeds outside of the training seeds at each iteration.

More on Architecture and Hyperparameters

We document the complete details on model architecture and hyperparameters in this section.

Model Architecture. All models start with a CNN encoder. Most environments also use an LSTM module to incorporate longer time dependencies, except for DDPG where we find that the training is much more stable with a single CNN and stacked frames (4 frames). DDPG also uses separate encoders on the observations for the actor and critic whereas the other models share the same CNN encoder. The CNN or CNN-LSTM part of model architecture is documented in Table S4.

The CNN encoders are a generic composition of convolutional layers and LeakyReLU nonlinearities. Two types of CNN encoders are used for Atari games: one with fewer parameters for simpler games, namely Breakout, Boxing, and Freeway, and the other with more parameters for the rest of the games. The LSTM module has 512 hidden units. DDPG uses the same CNN encoder as in [S4] but without the LSTM module. ACER shares the same architecture as A3C-LSTM for Seaquest, except that the value network now outputs 12 (number of actions for CarRacing) state-action value estimations instead of a single state value estimation.

Normalization of HPA3C. Since HPA3C needs to learn σ_t, more variance is introduced to the gradients and the resulting stochastic activations require further normalization. Because LeakyReLU allows the activations to concentrate below 0 in avoidance of the noisy gradients from variance learning, the effects of stochastic units are diminished. After trial-and-error with techniques such as Batch [S6] and Layer [S1] Normalization, we pick the most effective option—concatenated ReLU [S10] as used in [S5]. In theory, normalization techniques such as BatchNorm and LayerNorm can also adjust the output distribution but in practice they are not effective. We note that with the right normalization HPA3C in fact exhibits more stable and consistent training than A3C-LSTM and the other stochastic activation A3Cs.

Normalization of DDPG. In [S9], the authors achieve meaningful perturbation from a uniform spherical Gaussian noise distribution by normalizing the noisy layer's activations. We also discover that without proper normalization, the additional randomness from either weights or activations can catastrophically hamper optimization. We conjecture that the lack of $\tanh$ non-linear activation from LSTM can contribute to this issue. Therefore we apply layer normalization after the linear layers with stochastic weights or activations in DDPG.

Hyperparameters. Our models are optimized with Adam [S7] through gradient descent using mini-batches of size 64, thus spawning 64 asynchronous agents—except for CarRacing, where we use 8 agents. For Atari, hyperparameters are tuned on Seaquest A3C-LSTM baseline and then transferred to other games, with only small adjustments if necessary. We inherit all common hyperparameters from A3C-LSTM to stochastic activation A3Cs and only tune the additional ones, namely the variance for SA3C and FSA3C, the variance prior and the KL term weight for HPA3C. For other environments, hyperparameters are tuned on the baseline algorithm then transferred to other related models, such as with parametric noise and stochastic activations.

For Adam optimizer, initial learning rate for all Atari games is 0.0005, $\epsilon = 0.001, \beta_1 = 0.9, \beta_2 = 0.999$; gradients over 40 are clipped to 40. Discount rate γ for return is

	Exploration Type	Remarks
Seaquest-v4	easy, score exploit	Players drive around a submarine to eliminate enemies and save divers.
Boxing-v4	easy, human optimal	Players hit the opponent in a boxing game. max score: 100
BeamRider-v4	easy, score exploit	Players pilot a fixed ship to clear enemy sectors in the outer space.
Breakout-v4	easy, human optimal	Players bounce a traveling ball to hit and eliminate bricks. max score: 864
MsPacman-v4	hard, dense reward	Players navigate Pacman through a 2D maze to eat pellets and avoid ghosts.
Qbert-v4	hard, dense reward	Players need to alter the cube color in a pyramid by making Qbert hop around while avoiding obstacles and enemies.
Freeway-v4	hard, sparse reward	Players control chickens to run across highway filled with traffic to reach the other side.

Table S1: Selected Atari Game Descriptions.

	Seaquest	BeamRider	MsPacman	Boxing	Breakout	Qbert	Freeway
MAP	24601±4614	9886±1936	3665±1942	99.7±0.9	427±19	14772±2257	21.7±2.3
1	18749±6386	7331±1135	3368±1540	99.5±1.2	410±30	13535±2886	5.1±1.1
5	26550±8240	7826±1519	3344±1788	100.0±0.0	495±152	14772±1033	10.2±1.0
50	24552±6911	8580±1824	4267±1962	99.9±0.3	448±159	15227±290	19.8±0.9

Table S2: Dropout stochastic units tested on 10 random seeded games with MAP, and voting from 1, 5, and 50 stochastic policies.

0.95 except for Breakout where $\gamma = 0.99$. The objective for value estimation is weighted with $\lambda = 5$ except for Freeway[1] where $\lambda = 0.5$. The entropy term is weighted with $\beta = 0.01$. The rewards are clipped between -1 and 1 except for Freeway, where reward is not clipped. The maximum rollout for LSTM is 20 time steps except for Freeway's 30 steps. The trace decay parameter for the generalized advantage estimator is $\tau = 1.0$ except for Freeway $\tau = 0.92$. For the variance, we set $\log(\sigma^2) = -6$ for Seaquest, MsPacman and Breakout and $\log(\sigma^2) = -4$ for the rest. The variance prior for HPA3C is set in the same way and the KL term is weighted with $\psi = 0.0001$. The maximum training iteration is $150K$ but training is stopped early if model performance reaches a plateau or the score is maximized. In addition, we apply variance scaling based on the stochastic activation norm as explained in Proposition **??** for Breakout.

In our ACER experiments, we follow most of the hyperparameters used for Seaquest and train 10K iterations. Additional ACER hyperparameters include the replay ratio to be 1, replay buffer size 15000 and replay start when buffer reaches 5000, trust decay for TRPO 0.99, trust threshold 1, model averaging ratio 0.99, and importance weight truncation 10.

In our DDPG experiments, we use batch size 128, initial learning rate 0.0005. For parametric noise, the initial noise standard deviation 0.05, the distance threshold is 0.3 and the adapt coefficient is 1.05. For stochastic activation, we set $\log(\sigma^2) = -5$.

Dropout Stochastic Units

Existing works [S11, S3] have interpreted hidden units equipped with stochastic dropout, traditionally a means to prevent overfitting, in a Bayesian point of view. We test this alternative by replacing the Gaussian units in SA3C with dropout units of dropout rate 0.25. During evaluation, we test with no dropout (MAP), a single forward pass with dropout, and averaging over 5/50 forward passes with dropout. Dropout can improve upon baseline A3C-LSTM in some of the games but does not outperform the best Gaussian stochastic activation model (Table S3). Additionally, comparing to Gaussian units, dropout units are less flexible in design. However, it is a worthwhile future direction to more closely investigate the link with, and potential benefits of, a formal Bayesian treatment of stochastic activations with the aid of dropout stochastic units.

More on NoisyNet A3C-LSTM

As there is no existing published implementation of NoisyNet to A3C-LSTM frameworks, we experimented with different configurations as shown in Figure S1. NoisyNet-v1 only randomize the value and policy networks (in red). As we have additional FC layers after LSTM before the policy and value networks, NoisyNet-v2 attempts to randomize those connecting FC layers only. Finally, NoisyNet-v3 randomize both connecting FC layers and the policy and value networks, following the protocol of the A3C-NoisyNet in [S2].

For fairness, all models are evaluated using the MAP protocol. We test the 3 NoisyNet configurations on Seaquest. NoisyNet-v2 and v3 results in performance at the level of random moves, indicating that no meaningful training happened and randomizing the connecting FC layer is detrimental to optimization. NoisyNet-v1 indeed learns up to some degree but the performance (894 ± 313) is significantly worsened from the baseline A3C-LSTM model (13922 ± 4920); however, since it is the most promising configurations out of all possibilities, throughout the main text, we apply this architecture

[1] Freeway is a sparse reward game hence its hyperparameters are additionally tuned.

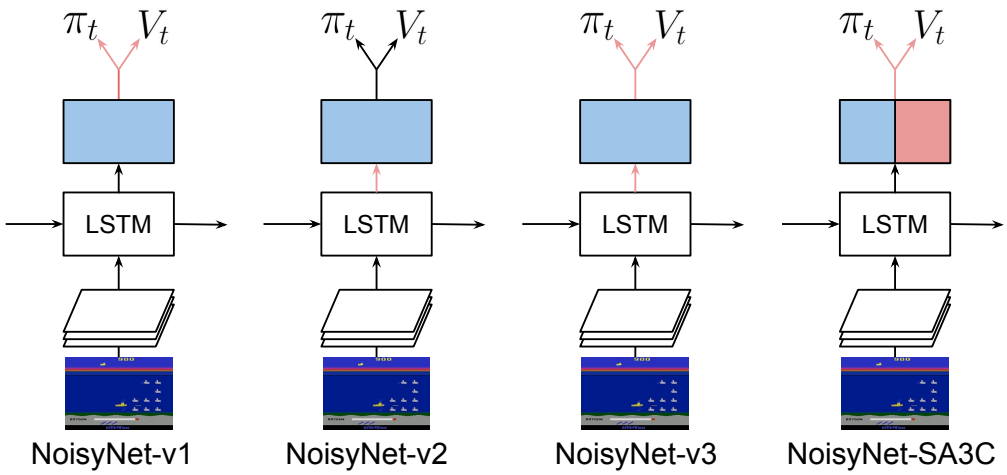

Figure S1: Different variations of NoisyNet that we test for A3C-LSTM as well as an integration with SA3C. Stochastic connections and activations are in red.

for any NoisyNet related experiment.

Lastly, we investigate how weight randomization affects stochastic activations and integrate NoisyNet to SA3C, forming NoisyNet-SA3C. NoisyNet-SA3C achieves 16037 ± 7438, better than NoisyNet-A3C-LSTM or A3C-LSTM, but substantially worse than SA3C (27695 ± 9096). Indeed, stochastic activations can help alleviate the difficulty of taming stochastic weights to some degree but the two are not necessarily complementary and stochastic weights can lower the performance of stochastic activations.

References

[S1] J. L. Ba, J. R. Kiros, and G. E. Hinton. Layer normalization. *arXiv*, 2016.

[S2] M. Fortunato, M. G. Azar, B. Piot, J. Menick, I. Osband, A. Graves, V. Mnih, R. Munos, D. Hassabis, O. Pietquin, et al. Noisy networks for exploration. *ICLR*, 2018.

[S3] Y. Gal and Z. Ghahramani. Dropout as a Bayesian approximation: Representing model uncertainty in deep learning. In *ICML*, 2016.

[S4] D. Griffis. Continuous action space A3C. `https://github.com/dgriff777/a3c_continuous`, 2018.

[S5] A. Gruslys, M. G. Azar, M. G. Bellemare, and R. Munos. The reactor: A sample-efficient actor-critic architecture. *arXiv*, 2017.

[S6] S. Ioffe and C. Szegedy. Batch normalization: Accelerating deep network training by reducing internal covariate shift. In *ICML*, 2015.

[S7] D. Kingma and J. Ba. Adam: A method for stochastic optimization. In *ICLR*, 2014.

[S8] V. Mnih, K. Kavukcuoglu, D. Silver, A. A. Rusu, J. Veness, M. G. Bellemare, A. Graves, M. Riedmiller, A. K. Fidjeland, G. Ostrovski, et al. Human-level control through deep reinforcement learning. *Nature*, 2015.

[S9] M. Plappert, R. Houthooft, P. Dhariwal, S. Sidor, R. Y. Chen, X. Chen, T. Asfour, P. Abbeel, and M. Andrychowicz. Parameter space noise for exploration. *ICLR*, 2018.

[S10] W. Shang, K. Sohn, D. Almeida, and H. Lee. Understanding and improving convolutional neural networks via concatenated rectified linear units. In *ICML*, 2016.

[S11] S. Wang and C. Manning. Fast dropout training. In *ICML*, 2013.

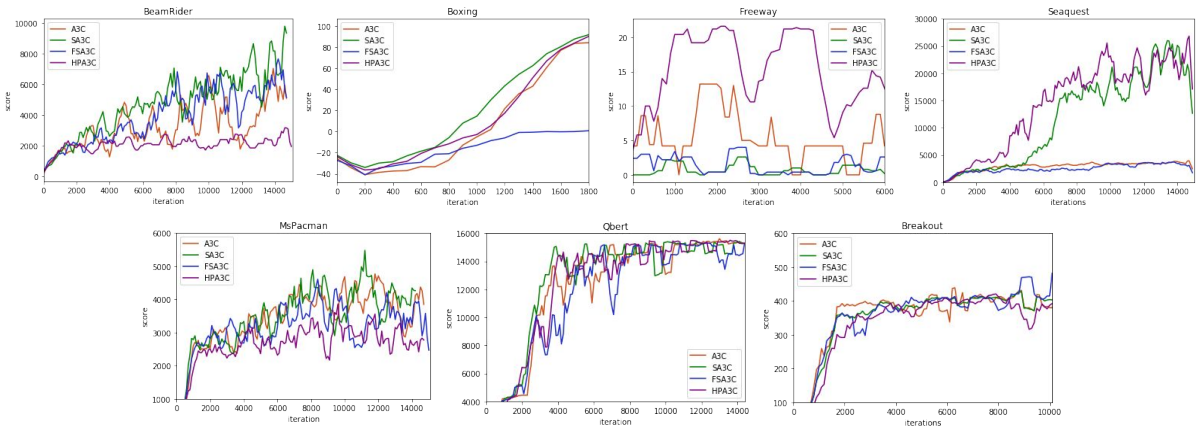

Figure S2: Compare different stochastic activation A3C with baseline. Training curves over 3 runs (median).

	Seaquest	BeamRider	MsPacman	Boxing	Breakout	Qbert	Freeway
SA3C							
MAP	27695±9096	9970±3428	4816±1569	99.4±1.2	497±139	15530±480	22.4±1.1
1	27081±6817	9648±2623	4819±1569	99.5±0.8	524±180	15330±146	22.4±1.1
5	28876±4270	9319±2125	4830±1595	99.2±1.6	493±135	15570±462	21.3±0.6
50	28341±9839	10540±3176	5096±1283	99.9±0.3	552±162	15530±480	21.2±0.6
FSA3C							
MAP	4287±171	11105±3402	5590±1521	99.4±1.0	409±14	14095±1762	21.2±2.3
1	5090±1099	10108±2858	4464±1638	98.7±2.0	422±27	14592±1943	24.2±0.6
5	4453±592	13779±3650	5304±1920	99.6±0.8	412±18	14017±1935	21.2±0.6
50	4794±523	9401±2207	3988±1697	99.4±1.8	423±89	14412±1652	21.3±0.6
HPA3C							
MAP	25474±8067	3349±2171	4515±2116	98.9±1.4	437±197	15302±2535	21.2±0.6
1	24475±4765	4338±3017	4287±1765	99.6±1.2	564±167	13160±147	23.9±1.3
5	29656±5317	4951±2171	4131±1345	100±0.0	596±197	12035±4306	21.3±0.6
50	28341±9839	5278±2234	3988±1697	99.2±1.6	378±192	12030±4923	21.2±0.6

Table S3: We compare testing on 10 random seeded games with MAP, and voting from 1, 5, and 50 stochastic policies.

Initialize network parameters θ;
for $k = 0, 1, 2, \cdots$ **do**
  Clear gradients $d\theta \leftarrow 0$;
  Simulate under current policy π_{t-1} until $t_{\max}$ steps are obtained, where, $h_t = f_{\text{LSTM}}(\text{CNN}(o_t), h_{t-1})$, $w_t = f_{\text{FC1}}(h_t), k_t = f_{\text{FC2}}(h_t), V_t = f_v(w_t, k_t), \pi_t = f_p(w_t, k_t), t = 1, \cdots t_{\max}$;
  $R = \begin{cases} 0, \text{ if terminal} \\ V_{t_{\max}+1}, \text{otherwise} \end{cases}$;
  for $t = t_{\max}, \cdots 1$ **do**
    $R \leftarrow r_t + \gamma R$;
    $A_t \leftarrow R - V_t$;
    Accumulate gradients from value loss: $d\theta \leftarrow d\theta + \lambda \frac{\partial A_t^2}{\partial \theta}$;
    $\delta_t \leftarrow r_t + \gamma V_{t+1} - V_t$;
    $\hat{A}_t \leftarrow \gamma \tau \hat{A}_{t-1} + \delta_i$;
    Accumulate policy gradients with entropy regularization: $d\theta \leftarrow d\theta + \nabla \log \pi_t(a_t) \hat{A}_t + \beta \nabla H(\pi_t)$;
  end
end

Algorithm 1: A3C-LSTM

Initialize network parameters θ;
Fix variance σ^2;
for $k = 0, 1, 2, \cdots$ **do**
  Clear gradients $d\theta \leftarrow 0$;
  Simulate under current policy π_{t-1} until $t_{\max}$ steps are obtained, where, $h_t = f_{\mathrm{LSTM}}(\mathrm{CNN}(o_t), h_{t-1})$, $\mu_t^p = f_{\mathrm{mean}}^p(z_{t-1})$,$\mu_t = f_{\mathrm{mean}}(h_t), \sigma_t^2 = f_{\mathrm{var}}(h_t), k_t = f_{\mathrm{d}}(h_t), z_t \sim \mathcal{N}(\mu_t, \sigma_t^2), V_t = f_v(z_t, k_t), \pi_t = f_p(z_t, k_t)$, $t = 1, \cdots t_{\max}$;
  $R = \begin{cases} 0, \text{ if terminal} \\ V_{t_{\max}+1}, \text{otherwise} \end{cases}$;
  for $t = t_{\max}, \cdots 1$ **do**
    $R \leftarrow r_t + \gamma R$;
    $A_t \leftarrow R - V_t$;
    Accumulate gradients from value loss: $d\theta \leftarrow d\theta + \lambda \frac{\partial A_t^2}{\partial \theta}$;
    $\delta_t \leftarrow r_t + \gamma V_{t+1} - V_t$;
    $\hat{A}_t \leftarrow \gamma\tau\hat{A}_{t-1} + \delta_i$;
    Accumulate policy gradients with entropy regularization:
    $d\theta \leftarrow d\theta + \nabla \log \pi_t(a_t)\hat{A}_t + \beta\nabla H(\pi_t) + \phi D_{KL}(\mathcal{N}(\mu_t, \sigma_t^2)||\mathcal{N}(\mu_t^p, (\sigma^p)^2))$;
  end
end

Algorithm 2: HPA3C

Initialize network parameters θ;
Initialize average network parameters θ_a;
for $k = 0, 1, 2, \cdots$ **do**
  Clear gradients $d\theta \leftarrow 0$;
  if *on policy* **then**
    Simulate under current policy π_{t-1} until $t_{\max}$ steps are obtained, where, $h_t = f_{\mathrm{LSTM}}(\mathrm{CNN}(o_t), h_{t-1})$, $w_t = f_{\mathrm{FC1}}(h_t), k_t = f_{\mathrm{FC2}}(h_t), Q_t = f_v(w_t, k_t), \pi_t = f_p(w_t, k_t), V_t = Q_t \cdot \pi_t, t = 1, \cdots t_{\max}$, set $\bar{\rho}_t = 1$;
  else
    Retrieve experience $(o_{1\cdots t_{\max}}, r_{1\cdots t_{\max}}, a_{1\cdots t_{\max}}, \mu_{1\cdots t_{\max}})$ from the memory buffer, compute $h_i = f_{\mathrm{LSTM}}(\mathrm{CNN}(o_t), h_{t-1}), w_t = f_{\mathrm{FC1}}(h_t), k_t = f_{\mathrm{FC2}}(h_t), Q_t = f_v(w_t, k_t), \pi_t = f_p(w_t, k_t), V_t = Q_t \cdot \pi_t$, $\bar{\rho}_t = \pi_t/\mu_t$,$t = 1, \cdots t_{\max}$.
  end
  $R = \begin{cases} 0, \text{ if terminal} \\ V_{t_{\max}+1}, \text{otherwise} \end{cases}$;
  for $t = t_{\max}, \cdots 1$ **do**
    $R \leftarrow r_t + \gamma R$;
    $A_t \leftarrow R - V_t$;
    $\mathcal{L}_V \leftarrow \frac{1}{2}(R - Q_t(a_t))^2$;
    Calculate advantage policy gradients: $g \leftarrow \nabla \log \pi(a_t)\hat{A}_t$;
    Calculate KL gradients: $k \leftarrow \nabla D_{KL}(\pi_t^a||\pi_t)$;
    Accumulate trust region policy gradients with entropy regularization:
    $d\theta \leftarrow d\theta + \nabla\theta(g - \max(0, \frac{k^T g - \delta}{||k||_2^2})k) + \beta\nabla H(\pi_t)$;
    Accumulate gradients from value loss: $d\theta \leftarrow d\theta + \lambda \frac{\partial \mathcal{L}_V}{\partial \theta}$;
    Update Retrace target: $R \leftarrow \rho_t(R - Q_t(a_t)) + V_t$
  end
  Update average model parameter: $\theta_a \leftarrow 0.99 * \theta_a + 0.01 * \theta$;
end

Algorithm 3: ACER

Initialize network parameters θ;
Fix variance σ^2;
Initialize average network parameters θ_a;
for $k = 0, 1, 2, \cdots$ **do**
  Clear gradients $d\theta \leftarrow 0$;
  if *on policy* **then**
    Simulate under current policy π_{t-1} until $t_{\max}$ steps are obtained, where, $h_t = f_{\mathrm{LSTM}}(\mathrm{CNN}(o_t), h_{t-1})$, $\mu_t = f_{\mathrm{mean}}(h_t), k_t = f_{\mathrm{d}}(h_t)$, $z_t \sim \mathcal{N}(\mu_t, \sigma^2)$, $Q_t = f_v(z_t, k_t)$, $\pi_t = f_p(z_t, k_t)$, $V_t = Q_t \cdot \pi_t$, $t = 1, \cdots t_{\max}$;
  else
    Retrieve experience $(o_{1\cdots t_{\max}}, r_{1\cdots t_{\max}}, a_{1\cdots t_{\max}}, \mu_{1\cdots t_{\max}})$ from the memory buffer, compute $h_t = f_{\mathrm{LSTM}}(\mathrm{CNN}(o_t), h_{t-1})$, $\mu_t = f_{\mathrm{mean}}(h_t), k_t = f_{\mathrm{d}}(h_t)$, $z_t \sim \mathcal{N}(\mu_t, \sigma^2)$, $Q_t = f_v(z_t, k_t)$, $\pi_t = f_p(z_t, k_t)$, $V_t = Q_t \cdot \pi_t$, $\bar{\rho}_t = \pi_t / \mu_t$,$t = 1, \cdots t_{\max}$.
  end
  $R = \begin{cases} 0, \text{ if terminal} \\ V_{t_{\max}+1}, \text{otherwise} \end{cases}$;
  for $t = t_{\max}, \cdots 1$ **do**
    $R \leftarrow r_t + \gamma R$;
    $A_t \leftarrow R - V_t$;
    $\mathcal{L}_V \leftarrow \frac{1}{2}(R - Q_t(a_t))^2$;
    Calculate advantage policy gradients: $g \leftarrow \nabla \log \pi(a_t) \hat{A}_t$;
    Calculate KL gradients: $k \leftarrow \nabla D_{KL}(\pi_t^a || \pi_t)$;
    Accumulate trust region policy gradients with entropy regularization: $d\theta \leftarrow d\theta + \nabla\theta(g - \max(0, \frac{k^T g - \delta}{||k||_2^2})k) + \beta \nabla H(\pi_t)$;
    Accumulate gradients from value loss: $d\theta \leftarrow d\theta + \lambda \frac{\partial \mathcal{L}_V}{\partial \theta}$;
    Update Retrace target: $R \leftarrow \rho_t(R - Q_t(a_t)) + V_t$
  end
  Update average model parameter: $\theta_a \leftarrow 0.99 * \theta_a + 0.01 * \theta$;
end

Algorithm 4: SACER

layer	input	output size	parameters
Default			
conv1	observation	32×80×80	32, 5, 1, 2
conv2	conv1	32×40×40	32, 3, 2, 1
conv3	conv2	32×40×40	32, 5, 1, 2
conv4	conv3	32×20×20	32, 3, 2, 1
conv5	conv4	64×20×20	64, 3, 1, 1
conv6	conv5	64×10×10	64, 3, 2, 1
conv7	conv6	64×10×10	64, 3, 1, 0
conv8	conv7	64×5×5	64, 3, 2, 1
lstm	conv8	512	1024
[FC1, FC2]	lstm	1024	
Slimmer			
conv1	observation	32×80×80	16, 5, 1, 2
conv2	conv1	32×40×40	32, 3, 2, 1
conv3	conv2	16×40×40	32, 5, 1, 2
conv4	conv3	32×20×20	32, 3, 2, 1
conv5	conv4	64×20×20	32, 3, 1, 1
conv6	conv5	64×10×10	64, 3, 2, 1
conv7	conv6	32×10×10	64, 3, 1, 0
conv8	conv7	64×5×5	64, 3, 2, 1
lstm	conv8	512	1024
[FC1, FC2]	lstm	1024	
1D			
conv1	observation	32×24	32, 3, 1, 1
conv2	conv1	32×24	32, 3, 1, 1
conv3	conv2	64×25	64, 2, 1, 1
conv4	conv3	64×25	64, 1, 1, 0

Table S4: The encoder and recurrent modules for A3C-LSTM and ACER. The parameter tuple for convolutional layers corresponds to number of filter, kernel size, stride size and padding size and for LSTM corresponds to number of hidden units. After the conv layers in the default CNN follows LeakyReLU activation function. Similarly with the slimmer CNN, unless if the number of channels doubled at output stage, then CReLU is applied.