Reinforcement Learning: Visual Control

Earl Wong

Visual Control Algorithms

We examine the following algorithms:

- GuidedPolicy Search
- DQN
- DrQ-v2
- Dreamer-v3

DQN

- DQN learned successful policies from high dimensional inputs using end to end reinforcement learning.

- DQN used a deep Q-network, to output actions from pixel based inputs.

- DQN training made use of 1) replay buffers / experience replay and 2) an additional target Q network.

- Paper title: Human-level control through deep reinforcement learning

- Authors: Mnih, et. al.

Foundational “Roots”

- DQN can be viewed as classical (tabular) Q learning, replaced by a deep neural network.
- Like classical Q learning, DQN employed two critics, to address over estimation biases.

DQN

Algorithm 1: deep Q-learning with experience replay.

Initialize replay memory D to capacity N
Initialize action-value function Q with random weights θ
Initialize target action-value function $\hat{Q}$ with weights $\theta^{-} = \theta$
For episode $= 1, M$ **do**
  Initialize sequence $s_1 = \{x_1\}$ and preprocessed sequence $\phi_1 = \phi(s_1)$
  For $t = 1$,T **do**
    With probability ε select a random action a_t
    otherwise select $a_t = \text{argmax}_a Q(\phi(s_t), a; \theta)$
    Execute action a_t in emulator and observe reward r_t and image x_{t+1}
    Set $s_{t+1} = s_t, a_t, x_{t+1}$ and preprocess $\phi_{t+1} = \phi(s_{t+1})$
    Store transition $\left(\phi_t, a_t, r_t, \phi_{t+1}\right)$ in D
    Sample random minibatch of transitions $\left(\phi_j, a_j, r_j, \phi_{j+1}\right)$ from D
    Set $y_j = \begin{cases} r_j & \text{if episode terminates at step j+1} \\ r_j + \gamma \max_{a'} \hat{Q}\left(\phi_{j+1}, a'; \theta^{-}\right) & \text{otherwise} \end{cases}$
    Perform a gradient descent step on $\left(y_j - Q\left(\phi_j, a_j; \theta\right)\right)^2$ with respect to the network parameters θ
    Every C steps reset $\hat{Q} = Q$
  End For
End For

The Details

- DQN made use of two action value functions - the Q function associated with the main network: Q, and the Q function associated with the target network: $\hat{Q}$.

- $\hat{Q}$ was used to compute a target y = $r_j + \gamma \cdot max_{a'} \hat{Q}(\phi_{j+1}, a'; \theta^-)$

- The difference between the target y and Q: $\left(y_j - Q\left(\phi_j, a_j; \theta\right)\right)^2$ was then back propagated into main Q network at each iteration.

- After a finite number of iterations / back propagations, the weights from Q were (then) copied over the existing weights associated with $\hat{Q}$.

- The process then repeated.

The Details

- This training “trick” was instrumental in increasing the training stability of the deep neural network.

- Today, a variation of this trick is used.

- Instead of performing an update (via copy) after a finite number of iterations, the update now occurs at every iteration using a weighted average (polyak averaging).

The Details

- The DQN algorithm sampled from a buffer, consisting of input vectors: $\left(\phi_j, a_j, r_j, \phi_{j+1}\right)$
- ϕ_j consisted of 4 temporally correlated video frames.
- After an action was taken, the environment then output a new video frame x_{t+1}.
- ϕ_j was then updated from $\phi_j = \{x_{t-3}, x_{t-2}, x_{t-1}, x_t\}$ to $\phi_{j+1} = \{x_{t-2}, x_{t-1}, x_t, x_{t+1}\}$
- Valuable temporal information - such as velocity, was contained in the temporally correlated frames.

The Details

- A mini batch is created, by sampling from a replay buffer containing the (previously described) vectors.
- Although the four input frames ϕ_j in each vector are temporally correlated, the vectors in the mini batch are not.
- This is because the vectors are sampled from different “plays” of the same game and / or different time instances.
- i.e. The vectors in the mini batch are approximately IID.
- DQN training was enhanced, by using mini batches with non-temporally correlated vectors.

DrQ-v2

- DrQ-v2 is a model free, off policy, actor-critic RL algorithm, for continuous visual control.
- It is the first model free RL algorithm to solve humanoid locomotion (stand, walk, run) using pixel based data.
- Paper title: Mastering Continuous Visual Control: Improved Data Augmented Reinforcement Learning
- Authors: Yarats, Fergus, Lazaric, Pinto

Foundational “Roots”

- DrQ-v2 uses data augmentation from computer vision, to achieve state of the art results for a suite of Deep Mind Control (DMC) tasks.
- DrQ-v2 uses the DDPG algorithm for it’s actor-critic implementation (vs SAC for DrQ) because it allowed n-step returns to be easily computed.
- DrQ-v2 uses entropy, to keep learning alive.
- DrQ-v2 uses two main Q networks (to reduce over estimation biases) and two target Q networks (to facilitate learning).

DrQ-v2

Algorithm 1 DrQ-v2: Improved data-augmented RL.

Inputs:
$f_\xi, \pi_\phi, Q_{\theta_1}, Q_{\theta_2}$: parametric networks for encoder, policy, and Q-functions respectively.
aug: random shifts image augmentation.
$\sigma(t)$: scheduled standard deviation for the exploration noise defined in Equation (3).
T, B, α, τ, c: training steps, mini-batch size, learning rate, target update rate, clip value.
Training routine:
for each timestep $t = 1..T$ **do**
    $\sigma_t \leftarrow \sigma(t)$ ▷ Compute stddev for the exploration noise
    $\boldsymbol{a}_t \leftarrow \pi_\phi(f_\xi(\boldsymbol{x}_t)) + \epsilon$ and $\epsilon \sim \mathcal{N}(0, \sigma_t^2)$ ▷ Add noise to the deterministic action
    $\boldsymbol{x}_{t+1} \sim P(\cdot|\boldsymbol{x}_t, \boldsymbol{a}_t)$ ▷ Run transition function for one step
    $\mathcal{D} \leftarrow \mathcal{D} \cup (\boldsymbol{x}_t, \boldsymbol{a}_t, R(\boldsymbol{x}_t, \boldsymbol{a}_t), \boldsymbol{x}_{t+1})$ ▷ Add a transition to the replay buffer
    UPDATECRITIC($\mathcal{D}, \sigma_t$)
    UPDATEACTOR($\mathcal{D}, \sigma_t$)
end for
procedure UPDATECRITIC($\mathcal{D}, \sigma$)
    $\{(\boldsymbol{x}_t, \boldsymbol{a}_t, r_{t:t+n-1}, \boldsymbol{x}_{t+n})\} \sim \mathcal{D}$ ▷ Sample a mini batch of B transitions
    $\boldsymbol{h}_t, \boldsymbol{h}_{t+n} \leftarrow f_\xi(\text{aug}(\boldsymbol{x}_t)), f_\xi(\text{aug}(\boldsymbol{x}_{t+n}))$ ▷ Apply data augmentation and encode
    $\boldsymbol{a}_{t+n} \leftarrow \pi_\phi(\boldsymbol{h}_{t+n}) + \epsilon$ and $\epsilon \sim \text{clip}(\mathcal{N}(0, \sigma^2))$ ▷ Sample action
    Compute $\mathcal{L}_{\theta_1,\xi}$ and $\mathcal{L}_{\theta_2,\xi}$ using Equation (1) ▷ Compute critic losses
    $\xi \leftarrow \xi - \alpha \nabla_\xi (\mathcal{L}_{\theta_1,\xi} + \mathcal{L}_{\theta_2,\xi})$ ▷ Update encoder weights
    $\theta_k \leftarrow \theta_k - \alpha \nabla_{\theta_k} \mathcal{L}_{\theta_k,\xi} \quad \forall k \in \{1, 2\}$ ▷ Update critic weights
    $\bar{\theta}_k \leftarrow (1 - \tau)\bar{\theta}_k + \tau \theta_k \quad \forall k \in \{1, 2\}$ ▷ Update critic target weights
end procedure
procedure UPDATEACTOR($\mathcal{D}, \sigma$)
    $\{(\boldsymbol{x}_t)\} \sim \mathcal{D}$ ▷ Sample a mini batch of B observations
    $\boldsymbol{h}_t \leftarrow f_\xi(\text{aug}(\boldsymbol{x}_t))$ ▷ Apply data augmentation and encode
    $\boldsymbol{a}_t \leftarrow \pi_\phi(\boldsymbol{h}_t) + \epsilon$ and $\epsilon \sim \text{clip}(\mathcal{N}(0, \sigma^2))$ ▷ Sample action
    Compute $\mathcal{L}_\phi$ using Equation (2) ▷ Compute actor loss
    $\phi \leftarrow \phi - \alpha \nabla_\phi \mathcal{L}_\phi$ ▷ Update actor's weights only
end procedure

The Details

- $h_t \leftarrow f_\xi(aug(x_t))$ takes an input image x_t of size 84x84, pads it (using nearest neighbor replication of value 4) and then randomly samples new 84x84 crops from it.

- Each pixel in the crop is blurred, using the average of it's four nearest neighbors.

- The blurred / averaged image is then encoded (via neural network) into a low dimensional latent space vector h_t

The Details

- DrQ-v2 uses N step returns, to improve training efficiency.

- To accomplish this, DrQ-v2 changed from a soft actor critic (SAC) backbone to a deep deterministic policy gradient (DDPG) backbone.

- In DDPG, exploration was no longer built into the objective function (like in SAC), but added as a noise term.

The Details

- DrQ-v2 added a “twist” to the original DPPG exploration construct, by making the noise time dependent.
- i.e. $a_t \leftarrow \pi_\phi(f_\xi(x_t)) + \epsilon$ where $\epsilon \sim N(0,\sigma^2(t))$
- Now, exploration was a based on a time dependent schedule.

The Details

- DrQ-v2 was computationally efficient and computationally competitive, because it was re-engineered to have a fast replay buffer and to perform fast data augmentation.

- DrQ-v2 performed well on hard DMC tasks (=tasks with large initial state distributions) by significantly increasing the size of the replay buffer.

Dreamer-v3

- Dreamer-v3 is a model based reinforcement learning (MBRL) algorithm.
- Dreamer-v3 learns a policy using an actor-critic algorithm trained on “latent imagination” produced by the Dreamer world model.
- Dreamer-v3 is a “generalist” algorithm, that was successfully trained on over 150 diverse visual tasks (Atari, ProcGen, DMLab, VisualControl, BSuite), outperforming many specialized models.
- Dreamer-v3 achieved this using 1) a fixed architecture, 2) fixed loss functions and 3) fixed hyper parameter settings.

Dreamer-v3

- Paper title: Mastering Diverse Control Tasks Through World Models
- Authors: Hafner, Pasukonis, Ba, Lillicrap

Foundational “Roots”

- The learned world model consisted of a recurrent state space model (RSSM) / latent dynamics model, a rewards predictor and a reconstruction predictor.

- Instead of interacting with the environment or using buffered data / experience replay, a policy was now learned using the world model environment / outputs from the world model.

- i.e. Learning a policy from rollouts / imagined trajectories in the latent space, where the world model supplies the next latent states and rewards.

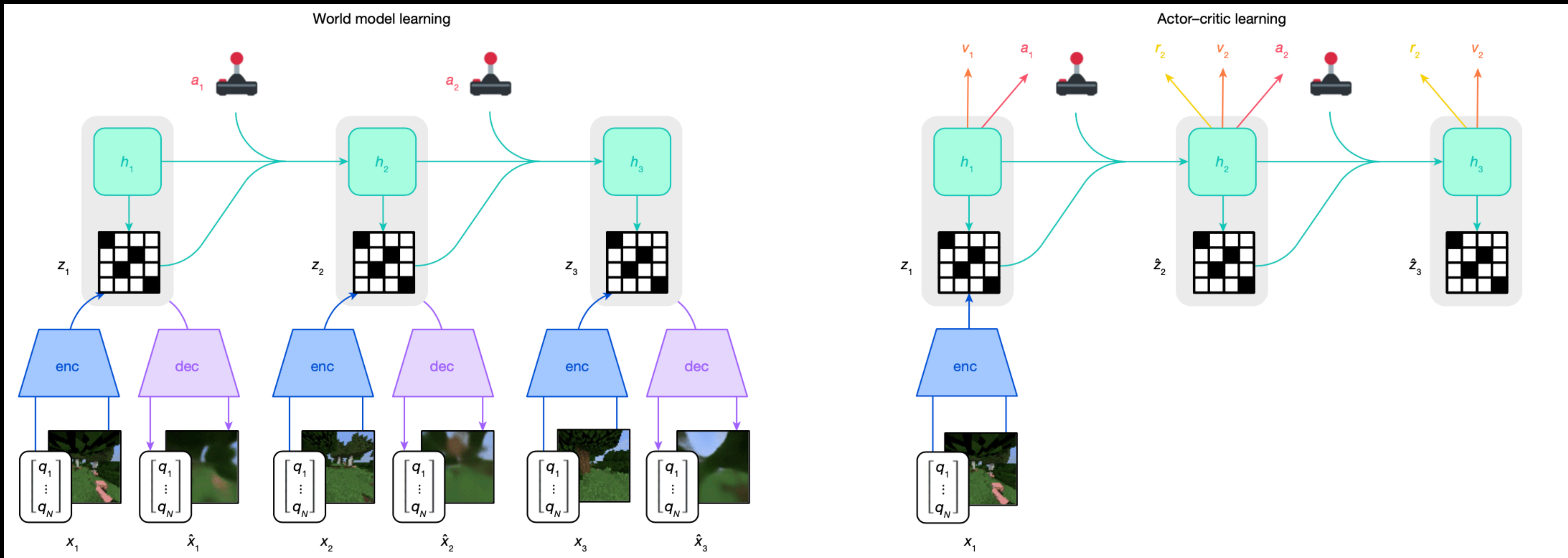
Foundational “Roots”

- Dreamer-v3 used KL divergence to ensure that the world model latent state outputs matched the latent state outputs computed from real visual information.

- Dreamer-v3 was able to learn policies for 150 diverse tasks (=good generalization) because it performed variable normalization.

- Specifically, normalization was applied to the input frames, the predicted output frames, the predicted rewards and the computed value function.

Dreamer-v3



1) **The Dreamer world model was realized using a recurrent state space model (RSSM), a rewards predictor and a reconstruction predictor.**

2) **An actor-critic algorithm was trained to determine the best actions, using future latent states and rewards produced by the world model.**

Dreamer-v1 (Original)

Algorithm 1: Dreamer

Initialize dataset $\mathcal{D}$ with S random seed episodes.
Initialize neural network parameters θ, ϕ, ψ randomly.
while *not converged* **do**
  for *update step* $c = 1..C$ **do**
    `// Dynamics learning`
    Draw B data sequences $\{(a_t, o_t, r_t)\}_{t=k}^{k+L} \sim \mathcal{D}$.
    Compute model states $s_t \sim p_\theta(s_t \mid s_{t-1}, a_{t-1}, o_t)$.
    Update θ using representation learning.
    `// Behavior learning`
    Imagine trajectories $\{(s_\tau, a_\tau)\}_{\tau=t}^{t+H}$ from each s_t.
    Predict rewards $\mathrm{E}\big(q_\theta(r_\tau \mid s_\tau)\big)$ and values $v_\psi(s_\tau)$.
    Compute value estimates $\mathrm{V}_\lambda(s_\tau)$ via Equation 6.
    Update $\phi \leftarrow \phi + \alpha \nabla_\phi \sum_{\tau=t}^{t+H} \mathrm{V}_\lambda(s_\tau)$.
    Update $\psi \leftarrow \psi - \alpha \nabla_\psi \sum_{\tau=t}^{t+H} \frac{1}{2} \big\| v_\psi(s_\tau) - \mathrm{V}_\lambda(s_\tau) \big\|^2$.
  `// Environment interaction`
  $o_1 \leftarrow$ `env.reset()`
  for *time step* $t = 1..T$ **do**
    Compute $s_t \sim p_\theta(s_t \mid s_{t-1}, a_{t-1}, o_t)$ from history.
    Compute $a_t \sim q_\phi(a_t \mid s_t)$ with the action model.
    Add exploration noise to action.
    $r_t, o_{t+1} \leftarrow$ `env.step(`a_t`)`.
  Add experience to dataset $\mathcal{D} \leftarrow \mathcal{D} \cup \{(o_t, a_t, r_t)_{t=1}^{T}\}$.

Model components

Representation	$p_\theta(s_t \mid s_{t-1}, a_{t-1}, o_t)$
Transition	$q_\theta(s_t \mid s_{t-1}, a_{t-1})$
Reward	$q_\theta(r_t \mid s_t)$
Action	$q_\phi(a_t \mid s_t)$
Value	$v_\psi(s_t)$

Hyper parameters

Seed episodes	S
Collect interval	C
Batch size	B
Sequence length	L
Imagination horizon	H
Learning rate	α

Evolution

- Dreamer-v2 and Dreamer-v3 did not provide a “cut and paste” algorithm descriptions.

- This was (probably) attributed to the “evolution details” for each new paper.

- However, the underlying methodology remained constant.

- 1) Use real data (image, action, reward) to train a world model.

- 2) Freeze the world model, and then train the actor-critic model using imagined, latent space trajectories.

Evolution

- Discrete action outputs were supported in v2.
- Loss functions definitions were modified between v1, v2 and v3.
- Normalizations were applied in Dreamer-v3, enabling Dreamer-v3 to train on 150 tasks without the need for retuning, etc.

The Details

- The Dreamer-v3 algorithm has the networks: the world model, the actor and the critic.

- The world model consisted of the RSSM (sequence model, encoder and dynamics predictor), the reward predictor, the decoder and the continue predictor.

Sequence model:	$h_t = f_\phi(h_{t-1}, z_{t-1}, a_{t-1})$
Encoder:	$z_t \sim q_\phi(z_t \mid h_t, x_t)$
Dynamics predictor:	$\hat{z}_t \sim p_\phi(\hat{z}_t \mid h_t)$
Reward predictor:	$\hat{r}_t \sim p_\phi(\hat{r}_t \mid h_t, z_t)$
Continue predictor:	$\hat{c}_t \sim p_\phi(\hat{c}_t \mid h_t, z_t)$
Decoder:	$\hat{x}_t \sim p_\phi(\hat{x}_t \mid h_t, z_t)$

- All of the components in the world model were trained concurrently.

The Details

- The world model loss had 3 different loss terms:

$$\mathcal{L}(\phi) \doteq E_{q_\phi}\left[\sum_{t=1}^{T}(\beta_{\text{pred}}\mathcal{L}_{\text{pred}}(\phi) + \beta_{\text{dyn}}\mathcal{L}_{\text{dyn}}(\phi) + \beta_{\text{rep}}\mathcal{L}_{\text{rep}}(\phi))\right],$$

$$\begin{aligned}
\mathcal{L}_{\text{pred}}(\phi) &\doteq -\log p_\phi(x_t|z_t,h_t) - \log p_\phi(r_t|z_t,h_t) - \log p_\phi(c_t|z_t,h_t)\\
\mathcal{L}_{\text{dyn}}(\phi) &\doteq \max(1, \mathrm{KL}[\mathrm{sg}(q_\phi(z_t|h_t,x_t))\|p_\phi(z_t|h_t)])\\
\mathcal{L}_{\text{rep}}(\phi) &\doteq \max(1, \mathrm{KL}[q_\phi(z_t|h_t,x_t)\|\mathrm{sg}(p_\phi(z_t|h_t))])
\end{aligned}$$

- pred lumped together the reward, continue and decoder loss.
- dyn handled the difference between the predicted latent space output for the world model and the latent space output trained using input data.
- rep should be for the sequence model? (check code)

The Details

- The symlog function was applied to the encoder inputs and the reconstruction outputs.
- The symlog function compressed large positive and negative values, while simultaneously preserving symmetry.
- The network was trained to produce compressed, reconstruction outputs.
- symexp was then applied, to produce the true reconstruction outputs.

$$\mathcal{L}(\theta) \doteq \frac{1}{2}(f(x,\theta) - \mathrm{symlog}(y))^2 \qquad \hat{y} \doteq \mathrm{symexp}\,(f(x,\theta))$$

$$\mathrm{symlog}(x) \doteq \mathrm{sign}(x)\log(|x|+1)$$
$$\mathrm{symexp}(x) \doteq \mathrm{sign}(x)(\exp(|x|)-1)$$

The Details

- The reward and value function scalers were computed using 2 hot encoding.
- First, a compressed, output range was determined - [min, max].
- Then, a fixed number of histogram bins were used to subdivide this range.
- i.e. Each bin was assigned a specific value in the range.

The Details

- symlog was then applied, to compress the scalar values to lie within the [min, max] range.
- By construction, each compressed scalar value would lie between two consecutive bins.
- These adjacent bins were then assigned probabilities (summing to 1), based on their closeness to the scalar value.

The Details

- At the same time, the network produced a sequence of logit values associated with each bin location.
- 2 hot cross entropy was then applied, to compute the resulting loss.

$$\mathcal{L}(\theta) \doteq -\operatorname{twohot}(y)^T \log \operatorname{softmax}(f(x, \theta))$$

$$\hat{y} \doteq \operatorname{softmax}(f(x))^T B \qquad B \doteq \operatorname{symexp}([-20 \quad \ldots \quad +20])$$

- The authors claimed that this was better for propagating gradients with large targets with large variance.
- i.e. The process was more stable than traditional regression.

Example

- The range is [-20, 20].
- 9 bins are assigned to this range.
- Following symbol compression, the scalar value maps to -19.
- This scalar lies between bin0 and bin1, with assigned values of -20 and -16.
- As a result, bin0 receives a probability of 0.75 while bin1 receives a probability of 0.25.

Example

- The network outputs 9 logits for the predicted scalar quantity.
- These logits are reprocessed using the softmax function.
- Suppose the result is [.20, .15, .01, .30, .11, .20, .01, .01, .01]
- 2 hot cross entropy loss is then performed: .75*.20 + .25*.15
- This loss is then back propagated into the network.