Exploration-Exploitation Trade-off in Deep Reinforcement Learning

Part 1

Georges Pantalos (pgeorges@ethz.ch)

ETH Zürich

April 2, 2019

Exploration-Exploitation trade-off

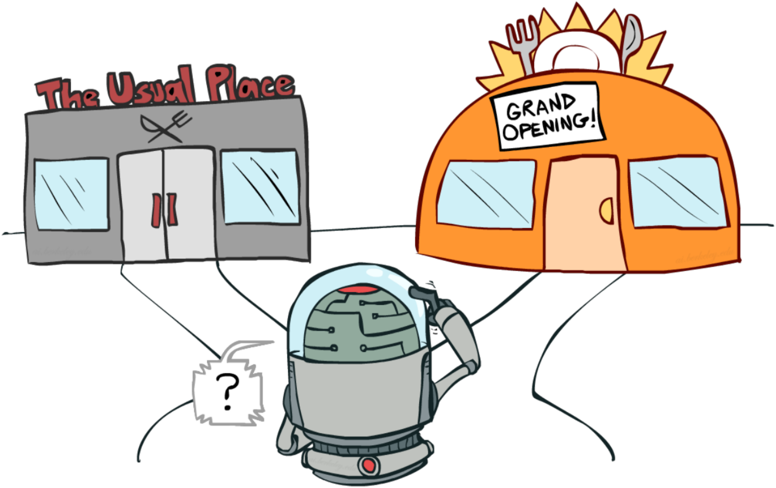


Figure: [UC Berkeley - CS188 Intro to AI]

Exploration-Exploitation trade-off

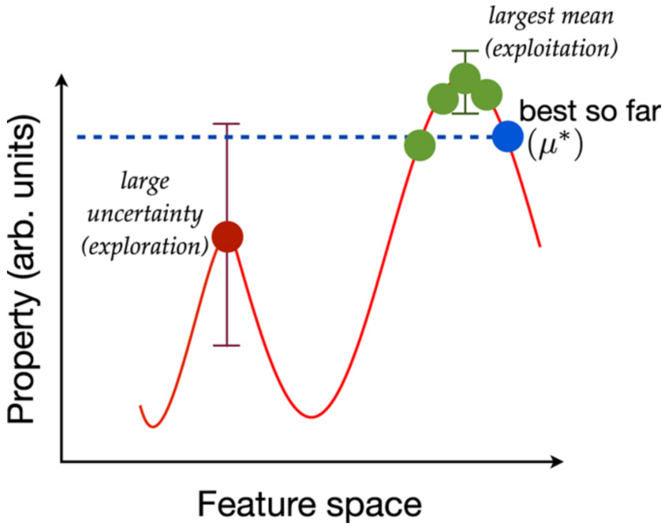


Figure: [researchgate.net]

Curiosity-driven Exploration by Self-supervised Prediction

D. Pathak, A. Efros, T. Darrell – UC Berkeley
ICML, May 2017

Main Idea

Main idea of the paper

The agent receives rewards for finding something unexpected

- Objective: maximize **cumulative reward** $\sum_t r_t$ where

$$r_t = \underbrace{r_t^e}_{\text{extrinsic}} + \underbrace{r_t^i}_{\text{intrinsic}}$$

- Definition:

$$\text{curiosity} := r_t^i$$

Main Idea

Main idea of the paper

The agent receives rewards for finding something unexpected

- Objective: maximize **cumulative reward** $\sum_t r_t$ where

$$r_t = \underbrace{r_t^e}_{\text{extrinsic}} + \underbrace{r_t^i}_{\text{intrinsic}}$$

- Definition:

$$\text{curiosity} := r_t^i$$

Main Idea

Main idea of the paper

The agent receives rewards for finding something unexpected

- Objective: maximize **cumulative reward** $\sum_t r_t$ where

$$r_t = \underbrace{r_t^e}_{\text{extrinsic}} + \underbrace{r_t^i}_{\text{intrinsic}}$$

- Definition:

$$\text{curiosity} := r_t^i$$

Games Tested

1. Super Mario Bros: 2D navigation

2. Viz Doom: 3D navigation

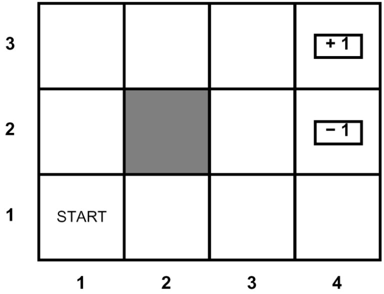
Games Tested

1. Super Mario Bros: 2D navigation

2. Viz Doom: 3D navigation

What are intrinsic rewards?

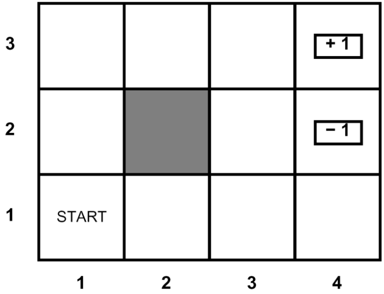
- **Extrinsic** rewards are provided by the environment.



- **Intrinsic** rewards encourage the agent to explore novel states.

What are intrinsic rewards?

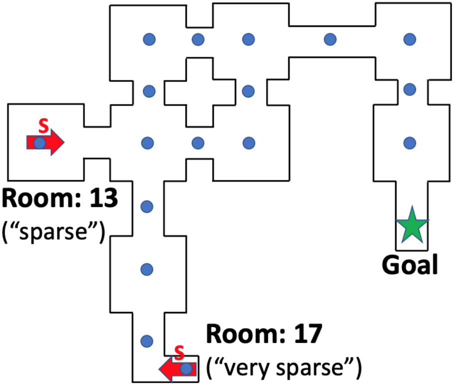
- **Extrinsic** rewards are provided by the environment.



- **Intrinsic** rewards encourage the agent to explore novel states.

Settings investigated

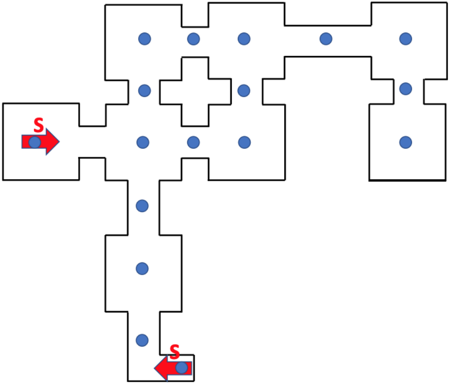
1. **Sparse** extrinsic rewards:



2. **Non existent** extrinsic rewards:
3. Learn **generalised skills** that might be helpful in the future:

Settings investigated

1. **Sparse** extrinsic rewards:
2. **Non existent** extrinsic rewards:



3. Learn **generalised skills** that might be helpful in the future:

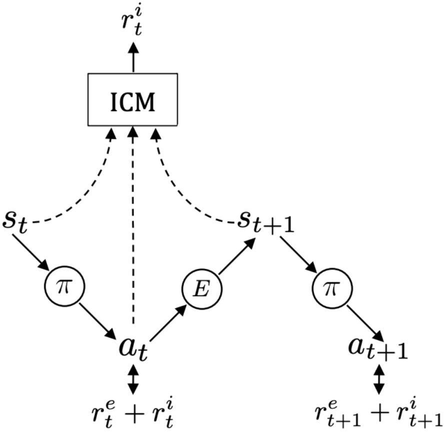
Settings investigated

1. **Sparse** extrinsic rewards:
2. **Non existent** extrinsic rewards:
3. Learn **generalised skills** that might be helpful in the future:

(a) learn to explore in Level-1

(b) explore faster in Level-2

Architecture



At time t

r_t^e	extrinsic reward
r_t^i	intrinsic reward

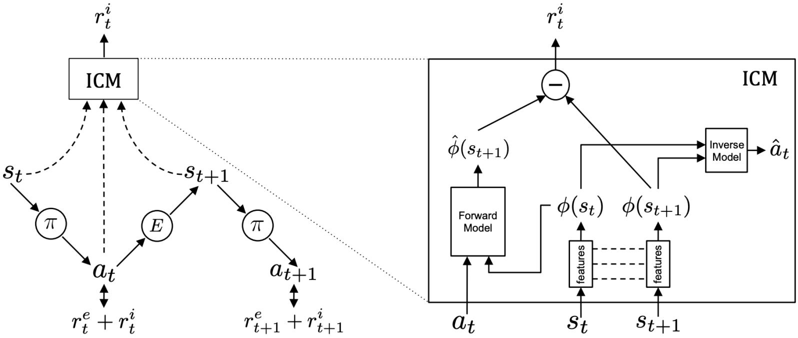
Remember

Intrinsic reward = curiosity!

And $r_t = r_t^e + r_t^i$

ICM = **Intrinsic Curiosity Module**

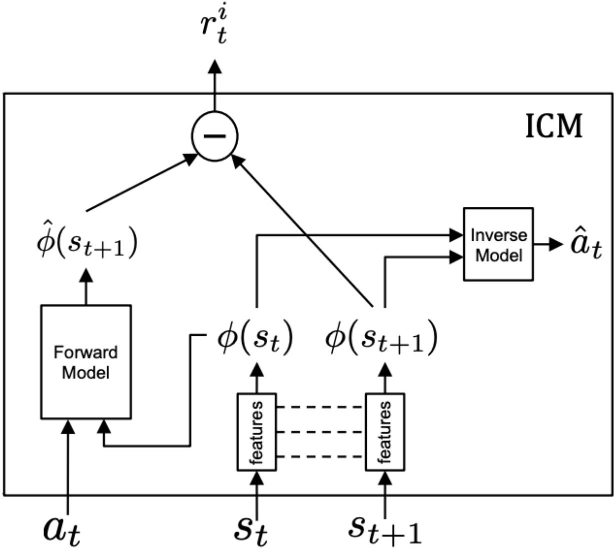
Let's look closer



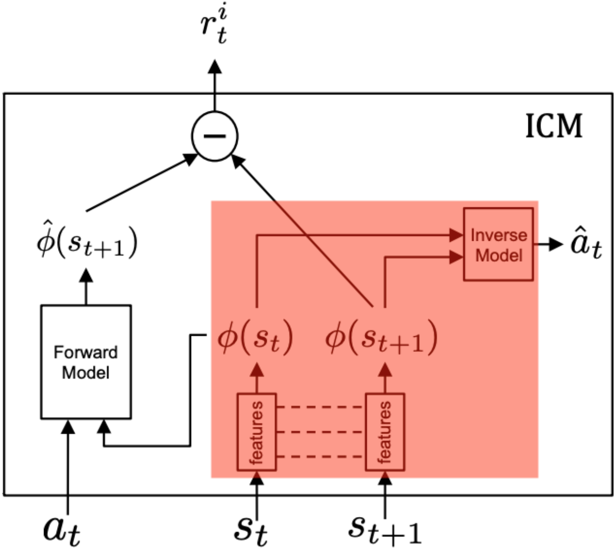
$\phi(\cdot)$ actual feature representation
$\hat{\phi}(\cdot)$ estimated feature representation

Intrinsic Curiosity Module (ICM)

We have two networks

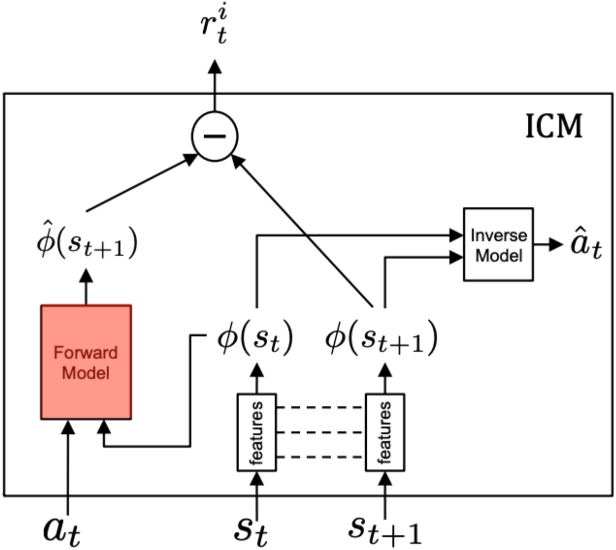


The Inverse Model



Intrinsic Curiosity Module (ICM)

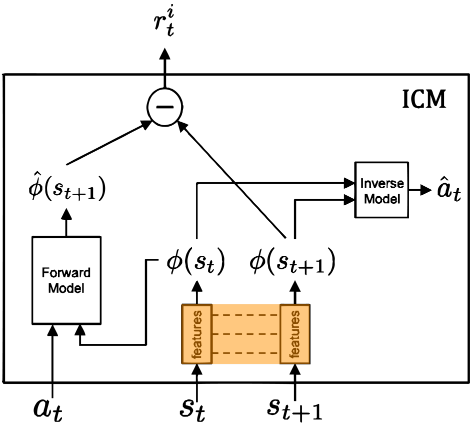
The Forward Model



Learning $\phi(\cdot)$, *i.e.* training the inverse model

Train two sub-modules:

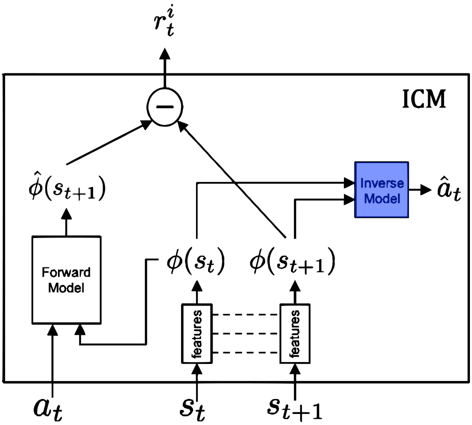
1. The **first** one encodes s_t into a feature vector $\phi(s_t)$
2. The **second** one predicts $\hat{a}_t$ from $\phi(s_t)$ and $\phi(s_{t+1})$



Learning $\phi(\cdot)$, *i.e.* training the inverse model

Train two sub-modules:

1. The first one encodes s_t into a feature vector $\phi(s_t)$
2. The **second** one predicts $\hat{a}_t$ from $\phi(s_t)$ and $\phi(s_{t+1})$



The Inverse Model

Learn function g (*i.e.* the **inverse dynamics model**) defined as:

$$\hat{a}_t = g(s_t, s_{t+1}; \theta_I)$$

where the parameters θ_I are trained to optimize

$$\min_{\theta_I} L_I\left(\hat{a}_t, a_t\right)$$

Implementation (2 submodules)

1. 4 convolution layers, each with 32 filters, stride of 2 and padding of 1. ELU is used after each convolution layer.
2. 2 fully connected layers (288 and 4 units resp.)

L_I is an arbitrary loss function

The Inverse Model

Learn function g (*i.e.* the **inverse dynamics model**) defined as:

$$\hat{a}_t = g(s_t, s_{t+1}; \theta_I)$$

where the parameters θ_I are trained to optimize

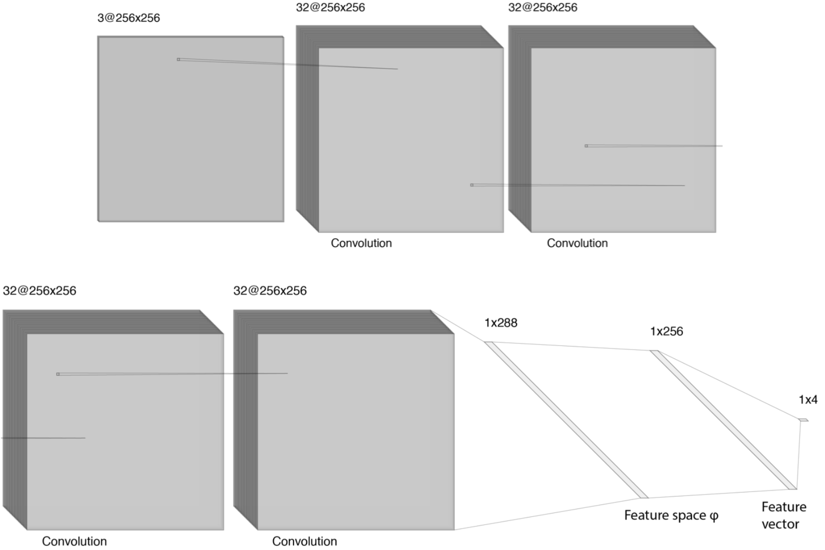
$$\min_{\theta_I} L_I\left(\hat{a}_t, a_t\right)$$

Implementation (2 submodules)

1. 4 convolution layers, each with 32 filters, stride of 2 and padding of 1. ELU is used after each convolution layer.
2. 2 fully connected layers (288 and 4 units resp.)

L_I is an arbitrary loss function

Implementation of the Inverse Model



The Forward Model

Learn function f (*i.e.* the **forward dynamics model**) defined as:

$$\hat{\phi}\left(s_{t+1}\right) = f\left(\phi\left(s_t\right), a_t; \theta_F\right)$$

where the parameters θ_F are trained to optimize

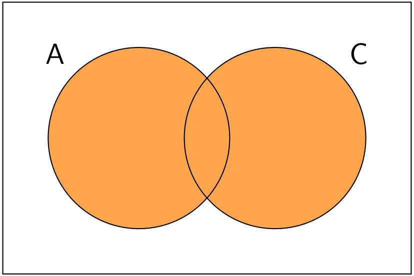
$$\min_{\theta_F} L_F\left(\phi\left(s_{t+1}\right), \hat{\phi}\left(s_{t+1}\right)\right)$$

Implementation

2 fully connected layers.

Here, $L_F = \frac{1}{2}\left\|\hat{\phi}\left(s_{t+1}\right) - \phi\left(s_{t+1}\right)\right\|_2^2$, *i.e.* least squares

Choosing the right $\phi(\cdot)$

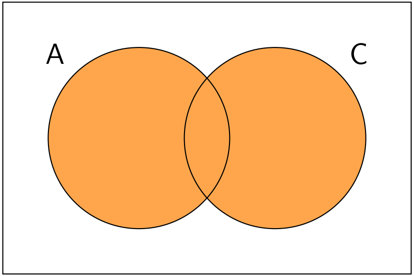


A events that **affect** the agent

C events that can be **controlled** by the agent

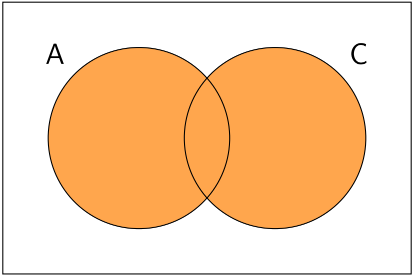
what we want to model

Choosing the right $\phi(\cdot)$



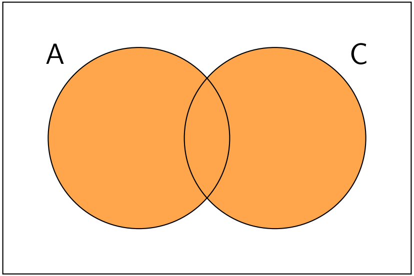
A	events that **affect** the agent
C	events that can be **controlled** by the agent
▢	what we want to model

Choosing the right $\phi(\cdot)$



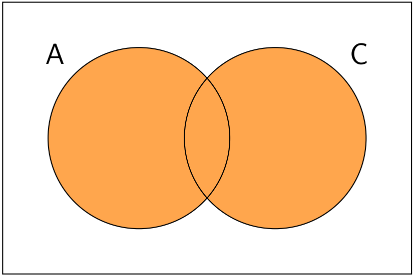
A	events that **affect** the agent
C	events that can be **controlled** by the agent
	what we want to model

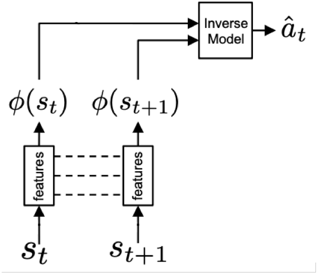
Choosing the right $\phi(\cdot)$



A	events that **affect** the agent
C	events that can be **controlled** by the agent
(orange box)	what we want to model

Choosing the right $\phi(\cdot)$





A events that **affect** the agent
C events that can be **controlled** by the agent
what we want to model

Expression for Curiosity

Curiosity: « *error in the agent's ability to predict the consequences of its own actions* ».

$$r_t^i = \frac{\eta}{2} \left\| \hat{\phi}(s_{t+1}) - \phi(s_{t+1}) \right\|_2^2 = \eta L_F$$

where $\eta > 0$ is a scaling factor.

Expression for Curiosity

Curiosity: « *error in the agent's ability to predict the consequences of its own actions* ».

$$r_t^i = \frac{\eta}{2} \left\| \hat{\phi}\left(s_{t+1}\right) - \phi\left(s_{t+1}\right) \right\|_2^2 = \eta L_F$$

where $\eta > 0$ is a scaling factor.

Overall Optimisation Problem

Overall Optimisation Problem

$$\min_{\theta_P,\theta_I,\theta_F} \left(-\lambda \underbrace{\mathbb{E}_{\pi(s_t;\theta_P)}\left[\sum_t r_t\right]}_{\text{policy gradient}} + (1-\beta)\underbrace{L_I}_{\text{inverse}} + \beta \underbrace{L_F}_{\text{forward}} \right)$$

subject to

$$0 \leq \beta \leq 1$$
$$0 < \lambda$$

Maximise Expected Cumulative Reward

Overall Optimisation Problem

$$\min_{\theta_P,\theta_I,\theta_F} \left(\underbrace{-\lambda \mathop{\mathbb{E}}_{\pi(s_t;\theta_P)} \left[\sum_t r_t \right]}_{\text{policy gradient}} + (1-\beta) \underbrace{L_I}_{\text{inverse}} + \beta \underbrace{L_F}_{\text{forward}} \right)$$

subject to

$$0 \leq \beta \leq 1$$
$$0 < \lambda$$

Learn the Feature Representation

Overall Optimisation Problem

$$\min_{\theta_P,\theta_I,\theta_F} \left(\underbrace{-\lambda \mathop{\mathbb{E}}_{\pi(s_t;\theta_P)} \left[\sum_t r_t \right]}_{\text{policy gradient}} + (1-\beta) \underbrace{L_I}_{\text{inverse}} + \beta \underbrace{L_F}_{\text{forward}} \right)$$

subject to

$$0 \leq \beta \leq 1$$
$$0 < \lambda$$

Overall Optimisation Problem

Minimise Curiosity

Overall Optimisation Problem

$$\min_{\theta_P, \theta_I, \theta_F} \left(\underbrace{-\lambda \, \mathbb{E}_{\pi(s_t;\theta_P)} \left[\sum_t r_t \right]}_{\text{policy gradient}} + (1-\beta) \underbrace{L_I}_{\text{inverse}} + \beta \underbrace{L_F}_{\text{forward}} \right)$$

subject to

$$0 \leq \beta \leq 1$$
$$0 < \lambda$$

Overall Optimisation Problem

Overall Optimisation Problem

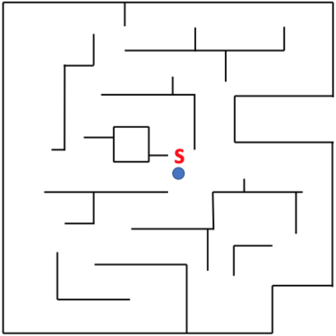
$$\min_{\theta_P, \theta_I, \theta_F} \left(\underbrace{-\lambda \mathop{\mathbb{E}}_{\pi(s_t;\theta_P)} \left[\sum_t r_t \right]}_{\text{policy gradient}} + (1-\beta) \underbrace{L_I}_{\text{inverse}} + \beta \underbrace{L_F}_{\text{forward}} \right)$$

subject to

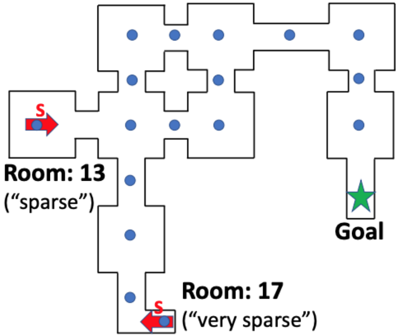
$$0 \leq \beta \leq 1$$
$$0 < \lambda$$

Remember

- Curiosity $= r_t^i = \eta L_F$
- Total reward at time t is $r_t = r_t^e + r_t^i$



(a) Train Map Scenario

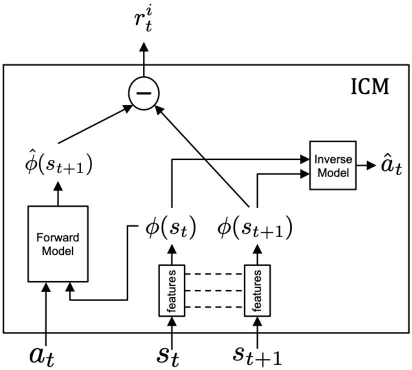


(b) Test Map Scenario

Sparse Reward Setting

We compare 3 setups:

1. **ICM + A3C:** full algorithm which combines ICM with A3C
2. **A3C:** vanilla A3C with ε-greedy exploration
3. **ICM-pixels + A3C:** variant of ICM without the inverse model



Sparse Reward Setting

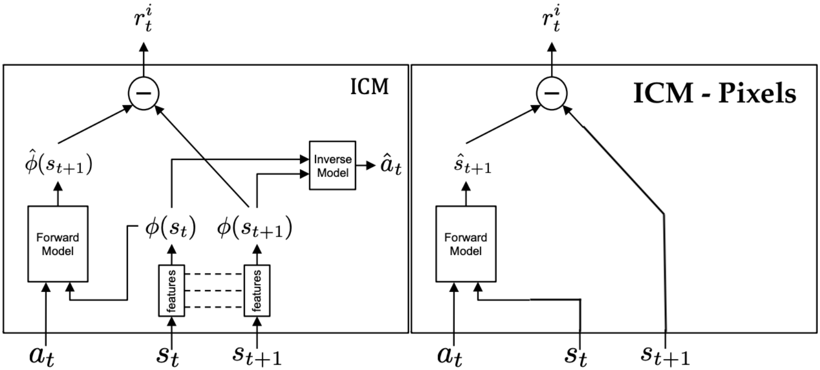
We compare 3 setups:

1. **ICM + A3C:** full algorithm which combines ICM with A3C
2. **A3C:** vanilla A3C with ε-greedy exploration
3. **ICM-pixels + A3C:** variant of ICM without the inverse model

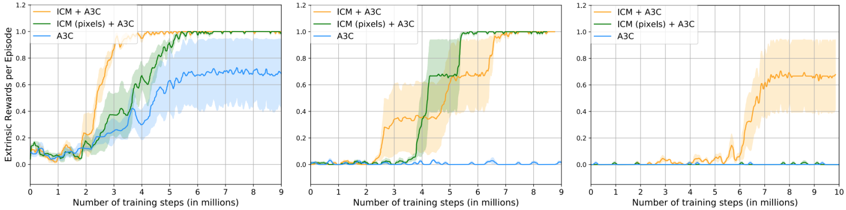
Sparse Reward Setting

We compare 3 setups:

1. **ICM + A3C:** full algorithm which combines ICM with A3C
2. **A3C:** vanilla A3C with ε-greedy exploration
3. **ICM-pixels + A3C:** variant of ICM without the inverse model



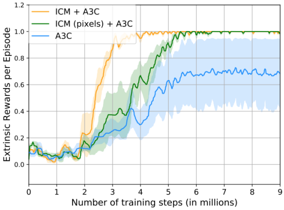
Performance



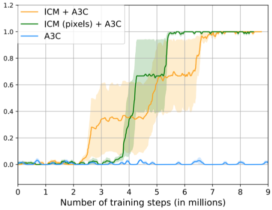
(a) “dense reward” setting (b) “sparse reward” setting (c) “very sparse reward” setting

- Curious A3C agents are superior in all cases
- ICM-pixels < ICM because the rooms have different textures

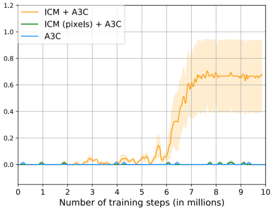
Performance



(a) “dense reward” setting



(b) “sparse reward” setting

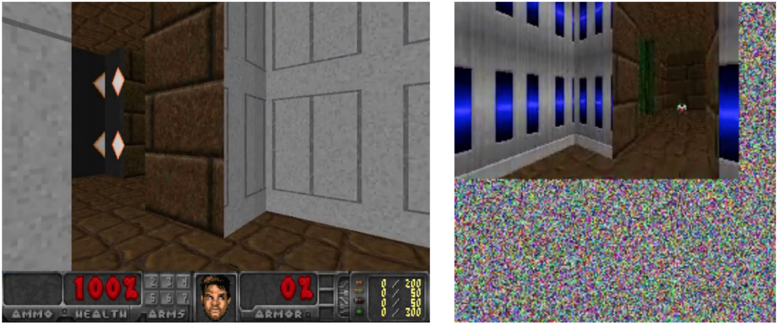


(c) “very sparse reward” setting

- Curious A3C agents are superior in all cases
- ICM-pixels $<$ ICM because the rooms have different textures

Robustness to Noise Inputs

Replace 40% of the input by white noise.



(a) Input snapshot in VizDoom (b) Input w/ noise

Robustness to Noise Inputs

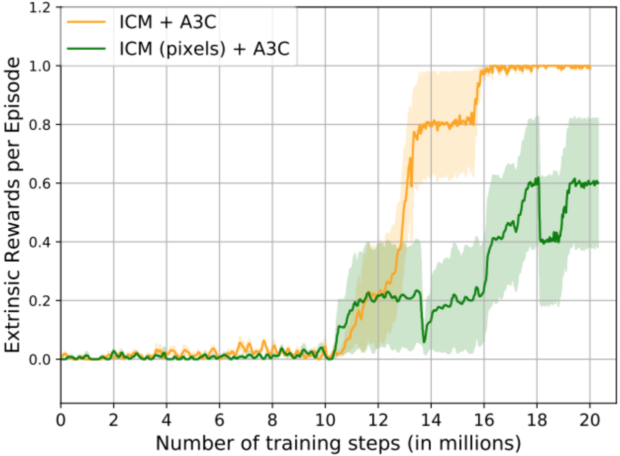


Figure: Results to noise input in the « sparse reward » setting

No Reward Setting

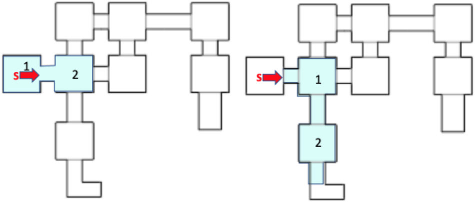


Figure: Random exploration

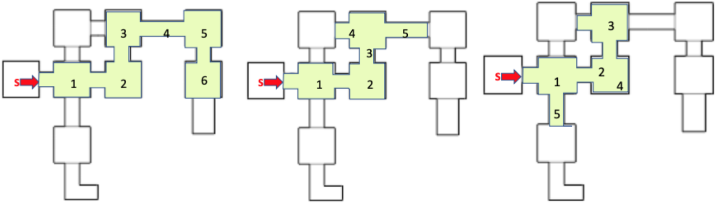


Figure: Curiosity driven exploration

No Reward Setting

- The agent now only survives so that he can explore more!
- First time in literature that learning from pixels occurs **without rewards**!

No Reward Setting

- The agent now only survives so that he can explore more!
- First time in literature that learning from pixels occurs **without rewards**!

Generalisation to Unseen Scenarios

3 settings are investigated:

1. Evaluate the learned policy **as is**

Use the policy learned in level 1 directly in level 2

2. Adapt the policy by **fine-tuning with curiosity** reward
3. Adapt the policy by **fine-tuning with extrinsic** reward

Generalisation to Unseen Scenarios

3 settings are investigated:

1. Evaluate the learned policy **as is**
2. Adapt the policy by **fine-tuning with curiosity** reward

Fine-tune the policy learned in level 1 using intrinsic rewards also in level 2

3. Adapt the policy by **fine-tuning with extrinsic** reward

Generalisation to Unseen Scenarios

3 settings are investigated:

1. Evaluate the learned policy **as is**
2. Adapt the policy by **fine-tuning with curiosity** reward
3. Adapt the policy by **fine-tuning with extrinsic** reward

Fine-tune the policy learned in level 1 by adding extrinsic rewards in level 2

Results showing generalisation

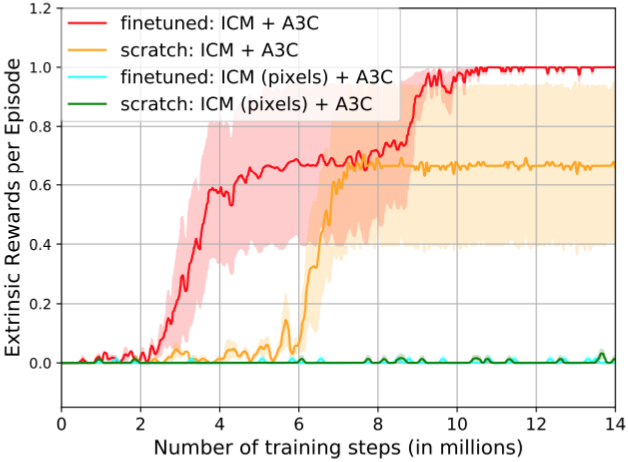


Figure: Test set of *VizDoom* in the « very sparse » reward case and fine-tuned on extrinsic rewards

Demo

https://www.youtube.com/watch?v=J3FHOyhUn3A

Noisy Networks for Exploration,

M. Fortunato, M. Azar, B. Piot et al. – Deepmind
ICLR, Feb 2018

Heuristic Approaches for Exploration

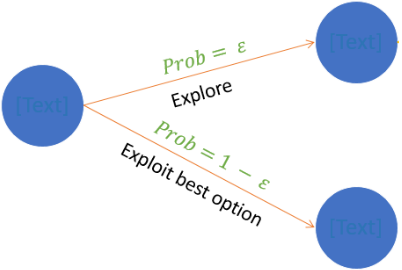
1. **Entropy regularisation**: entropy bonus to the loss function.
2. ε-**greedy**:

Problem

Local perturbation methods

Heuristic Approaches for Exploration

1. **Entropy regularisation**: entropy bonus to the loss function.
2. ε-**greedy**:

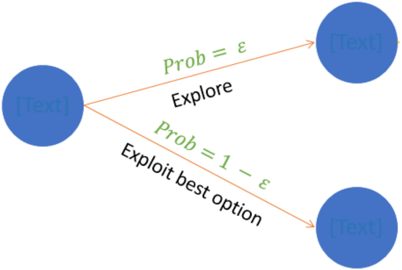


Problem

Local perturbation methods

Heuristic Approaches for Exploration

1. **Entropy regularisation**: entropy bonus to the loss function.
2. ε-**greedy**:



Problem

Local perturbation methods

More Heuristics for Exploration

3. **Optimism in the face of uncertainty**: theoretical guarantees on performance

Problem

Small state-action spaces and linear function approximations

4. **Intrinsic Motivation term** : *e.g.* curiosity

Problems

Separate generalisation from exploration. Many interactions needed.

More Heuristics for Exploration

3. **Optimism in the face of uncertainty**: theoretical guarantees on performance

Problem

Small state-action spaces and linear function approximations

4. **Intrinsic Motivation term** : *e.g.* curiosity

Problems

Separate generalisation from exploration. Many interactions needed.

More Heuristics for Exploration

3. **Optimism in the face of uncertainty**: theoretical guarantees on performance

Problem

Small state-action spaces and linear function approximations

4. **Intrinsic Motivation term** : *e.g.* curiosity

Problems

Separate generalisation from exploration. Many interactions needed.

More Heuristics for Exploration

3. **Optimism in the face of uncertainty**: theoretical guarantees on performance

Problem

Small state-action spaces and linear function approximations

4. **Intrinsic Motivation term** : *e.g.* curiosity

Problems

Separate generalisation from exploration. Many interactions needed.

Implementation on a Linear Layer

- **Linear layer:**

$$y = wx + b$$

- Corresponding **noisy linear layer**:

$$y := \underbrace{(\mu^w + \sigma^w \odot \varepsilon^w)}_{w} x + \underbrace{\mu^b + \sigma^b \odot \varepsilon^b}_{b}$$

  - μ^w, μ^b, σ^w and σ^b are learnable
  - ε^w and ε^b are noise RVs.

$\odot$ denotes element-wise multiplication

Implementation on a Linear Layer

- **Linear layer:**

$$y = wx + b$$

- Corresponding **noisy linear layer**:

$$y := \underbrace{(\mu^w + \sigma^w \odot \varepsilon^w)}_{w} x + \underbrace{\mu^b + \sigma^b \odot \varepsilon^b}_{b}$$

  - μ^w , μ^b , σ^w and σ^b are learnable
  - ε^w and ε^b are noise RVs.

$\odot$ denotes element-wise multiplication

Implementation on a Linear Layer

- **Linear layer:**

$$y = wx + b$$

- Corresponding **noisy linear layer**:

$$y := \underbrace{(\mu^w + \sigma^w \odot \varepsilon^w)}_{w} x + \underbrace{\mu^b + \sigma^b \odot \varepsilon^b}_{b}$$

  - μ^w , μ^b , σ^w and σ^b are learnable
  - ε^w and ε^b are noise RVs.

$\odot$ denotes element-wise multiplication

Implementation on a Linear Layer

- **Linear layer:**

$$y = w x + b$$

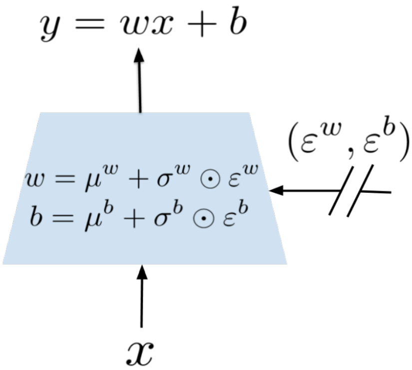
- Corresponding **noisy linear layer**:

$$y := \underbrace{(\mu^{w} + \sigma^{w} \odot \varepsilon^{w})}_{w} x + \underbrace{\mu^{b} + \sigma^{b} \odot \varepsilon^{b}}_{b}$$

  - μ^{w} , μ^{b} , σ^{w} and σ^{b} are learnable
  - ε^{w} and ε^{b} are noise RVs.

$\odot$ denotes element-wise multiplication

Graphical Representation of the noisy linear layer



Dimensions

For p inputs and q outputs:

$$x \in \mathbb{R}^p$$
$$y, \mu^b, \sigma^b, \varepsilon^b \in \mathbb{R}^q$$
$$\mu^w, \sigma^w, \varepsilon^w \in \mathbb{R}^{q \times p}$$

How do we choose ε?

1. **Independent Gaussian Noise**: Sample each variable $\varepsilon \sim \mathcal{N}(0,1)$ for *every weight* in a layer *independently*.

$$\varepsilon^w = \begin{bmatrix} \varepsilon^w_{11} & \cdots & \varepsilon^w_{1p} \\ \vdots & \ddots & \vdots \\ \varepsilon^w_{q1} & \cdots & \varepsilon^w_{qp} \end{bmatrix} \quad \text{and} \quad \varepsilon^b = \begin{bmatrix} \varepsilon^b_1 \\ \vdots \\ \varepsilon^b_q \end{bmatrix}$$

For p inputs and q outputs, $pq + q$ variables to samples.

Noise is completely uncorrelated

How do we choose ε?

1. **Independent Gaussian Noise**: Sample each variable $\varepsilon \sim \mathcal{N}(0,1)$ for *every weight* in a layer *independently*.

$$\varepsilon^w = \begin{bmatrix} \varepsilon^w_{11} & \cdots & \varepsilon^w_{1p} \\ \vdots & \ddots & \vdots \\ \varepsilon^w_{q1} & \cdots & \varepsilon^w_{qp} \end{bmatrix} \quad \text{and} \quad \varepsilon^b = \begin{bmatrix} \varepsilon^b_1 \\ \vdots \\ \varepsilon^b_q \end{bmatrix}$$

For p inputs and q outputs, $pq + q$ variables to samples.

Noise is completely uncorrelated

How do we choose ε?

1. **Independent Gaussian Noise**: Sample each variable $\varepsilon \sim \mathcal{N}(0, 1)$ for *every weight* in a layer *independently*.

$$\varepsilon^w = \begin{bmatrix} \varepsilon^w_{11} & \cdots & \varepsilon^w_{1p} \\ \vdots & \ddots & \vdots \\ \varepsilon^w_{q1} & \cdots & \varepsilon^w_{qp} \end{bmatrix} \quad \text{and} \quad \varepsilon^b = \begin{bmatrix} \varepsilon^b_1 \\ \vdots \\ \varepsilon^b_q \end{bmatrix}$$

For p inputs and q outputs, $pq + q$ variables to samples.

Noise is completely uncorrelated

How do we choose ε?

2. **Factorised Gaussian Noise**: Sample p variables $\varepsilon_i \sim \mathcal{N}(0,1)$ and q variables $\varepsilon_j \sim \mathcal{N}(0,1)$ independently.

$$\begin{cases} \varepsilon_{ij}^{w} = f(\varepsilon_i) f(\varepsilon_j) \\ \varepsilon_{j}^{b} = f(\varepsilon_j) \end{cases}$$

where $f(x) = \operatorname{sgn}(x)\sqrt{|x|}$

For p inputs and q outputs, $p + q$ variables to sample.

Noise is correlated

How do we choose ε?

2. **Factorised Gaussian Noise**: Sample p variables $\varepsilon_i \sim \mathcal{N}(0,1)$ and q variables $\varepsilon_j \sim \mathcal{N}(0,1)$ independently.

$$\begin{cases} \varepsilon_{ij}^{w} = f(\varepsilon_i) f(\varepsilon_j) \\ \varepsilon_{j}^{b} = f(\varepsilon_j) \end{cases}$$

where $f(x) = \mathrm{sgn}(x)\sqrt{|x|}$

For p inputs and q outputs, $p + q$ variables to sample.

Noise is correlated

How do we choose ε?

2. **Factorised Gaussian Noise**: Sample p variables $\varepsilon_i \sim \mathcal{N}(0,1)$ and q variables $\varepsilon_j \sim \mathcal{N}(0,1)$ independently.

$$\begin{cases} \varepsilon_{ij}^{w} = f\left(\varepsilon_i\right) f\left(\varepsilon_j\right) \\ \varepsilon_j^{b} = f\left(\varepsilon_j\right) \end{cases}$$

where $f(x) = \operatorname{sgn}(x)\sqrt{|x|}$

For p inputs and q outputs, $p + q$ variables to sample.

Noise is correlated

Loss Function of a Noisy Network

Loss Function

$$\overline{L}(\zeta) = \mathbb{E}_{\varepsilon}[L(\theta)]$$

where $\zeta := (\mu, \Sigma)$ and $\theta := \mu + \Sigma \odot \varepsilon$.

Gradient of the Loss Function

$$\nabla \overline{L}(\zeta) = \nabla \mathbb{E}_{\varepsilon}[L(\theta)] = \mathbb{E}_{\varepsilon}[\nabla L(\mu + \Sigma \odot \varepsilon)] \overset{\text{MC}}{\approx} \nabla L(\mu + \Sigma \odot \xi)$$

where ξ is sampled at each step of the optimisation.

$\odot$ denotes element-wise multiplication

Loss Function of a Noisy Network

Loss Function

$$\overline{L}(\zeta) = \mathop{\mathbb{E}}_{\varepsilon}[L(\theta)]$$

where $\zeta := (\mu, \Sigma)$ and $\theta := \mu + \Sigma \odot \varepsilon$.

Gradient of the Loss Function

$$\nabla \overline{L}(\zeta) = \nabla \mathop{\mathbb{E}}_{\varepsilon}[L(\theta)] = \mathop{\mathbb{E}}_{\varepsilon}[\nabla L(\mu + \Sigma \odot \varepsilon)] \overset{\text{MC}}{\approx} \nabla L(\mu + \Sigma \odot \xi)$$

where ξ is sampled at each step of the optimisation.

$\odot$ denotes element-wise multiplication

Loss Function of a Noisy Network

Loss Function

$$\overline{L}(\zeta) = \mathop{\mathbb{E}}_{\varepsilon}[L(\theta)]$$

where $\zeta := (\mu, \Sigma)$ and $\theta := \mu + \Sigma \odot \varepsilon$.

Gradient of the Loss Function

$$\nabla \overline{L}(\zeta) = \nabla \mathop{\mathbb{E}}_{\varepsilon}[L(\theta)] = \mathop{\mathbb{E}}_{\varepsilon}[\nabla L(\mu + \Sigma \odot \varepsilon)] \overset{\text{MC}}{\approx} \nabla L(\mu + \Sigma \odot \xi)$$

where ξ is sampled at each step of the optimisation.

$\odot$ denotes element-wise multiplication

The changes made are:

1. ε-greedy no longer used. The policy **always** optimises the action-value function Q.
2. Fully connected layers of value network $\rightarrow$ noisy network with **factorised** gaussian noise.

NoisyNet Versions of DQN and Dueling

The changes made are:

1. ε-greedy no longer used. The policy **always** optimises the action-value function Q.
2. Fully connected layers of value network $\rightarrow$ noisy network with **factorised** gaussian noise.

Loss Function of the *NoisyNet* DQN

- Original DQN:

$$L(\theta) = \mathop{\mathbb{E}}_{(x,a,r,y)\sim D}\left[\left(r + \gamma \max_{b\in A} Q\left(y, b; \theta^{-}\right) - Q(x, a; \theta)\right)^2\right]$$

- *NoisyNet* DQN:

$$\overline{L}(\zeta) = \mathop{\mathbb{E}}_{\varepsilon,\varepsilon'}\left[\mathop{\mathbb{E}}_{(x,a,r,y)\sim D}\left[r + \gamma \max_{b\in A} Q\left(y, b, \varepsilon'; \zeta^{-}\right) - Q(x, a, \varepsilon; \zeta)\right]^2\right]$$

Loss Function of the *NoisyNet* DQN

- Original DQN:

$$L(\theta) = \mathbb{E}_{(x,a,r,y)\sim D}\left[\left(r + \gamma \max_{b\in A} Q\left(y, b; \theta^{-}\right) - Q(x, a; \theta)\right)^2\right]$$

- *NoisyNet* DQN:

$$\overline{L}(\zeta) = \mathbb{E}_{\varepsilon,\varepsilon'}\left[\mathbb{E}_{(x,a,r,y)\sim D}\left[r + \gamma \max_{b\in A} Q\left(y, b, \varepsilon'; \zeta^{-}\right) - Q(x, a, \varepsilon; \zeta)\right]^2\right]$$

NoisyNet Version of Dueling DQN

- Original Dueling DQN:

$$L(\theta) = \mathop{\mathbb{E}}_{(x,a,r,y)\sim D}\left[\left(r + \gamma Q\left(y, b^*(y); \theta^-\right) - Q(x, a; \theta)\right)^2\right]$$

$$\text{s.t.} \quad b^*(y) = \arg\max_{b\in\mathcal{A}} Q(y, b; \theta)$$

- *NoisyNet* Dueling DQN:

$$\overline{L}(\zeta) = \mathop{\mathbb{E}}_{\varepsilon,\varepsilon'}\left[\mathop{\mathbb{E}}_{(x,a,r,y)\sim D}\left[r + \gamma Q\left(y, b^*(y), \varepsilon'; \zeta^-\right) - Q(x, a, \varepsilon; \zeta)\right]^2\right]$$

$$\text{s.t.} \quad b^*(y) = \arg\max_{b\in\mathcal{A}} Q\left(y, b(y), \varepsilon''; \zeta\right)$$

NoisyNet Version of Dueling DQN

- Original Dueling DQN:

$$L(\theta) = \underset{(x,a,r,y)\sim D}{\mathbb{E}} \left[\left(r + \gamma Q\left(y, b^*(y); \theta^-\right) - Q(x, a; \theta) \right)^2 \right]$$

$$\text{s.t.} \quad b^*(y) = \arg\max_{b \in \mathcal{A}} Q(y, b; \theta)$$

- *NoisyNet* Dueling DQN:

$$\overline{L}(\zeta) = \underset{\varepsilon,\varepsilon'}{\mathbb{E}} \left[\underset{(x,a,r,y)\sim D}{\mathbb{E}} \left[r + \gamma Q\left(y, b^*(y), \varepsilon'; \zeta^-\right) - Q(x, a, \varepsilon; \zeta) \right]^2 \right]$$

$$\text{s.t.} \quad b^*(y) = \arg\max_{b \in \mathcal{A}} Q\left(y, b(y), \varepsilon''; \zeta\right)$$

NoisyNet Version of A3C

The changes made are:

- the **entropy bonus** of the policy loss is removed:

$$\nabla_\theta L^\pi(\theta) = -\mathbb{E}^\pi \left[\sum_{i=0}^{k} \nabla_\theta \log \left(\pi \left(a_{t+i} | x_{t+i}; \theta \right) \right) A \left(x_{t+i}, a_{t+i}; \theta \right) + \beta \sum_{i=0}^{k} \nabla_\theta H(\pi(\cdot | x_{t+i}; \theta)) \right]$$

- Fully connected layers of policy network $\rightarrow$ noisy layers.

NoisyNet Version of A3C

The changes made are:

- the **entropy bonus** of the policy loss is removed:

$$\nabla_\theta L^\pi(\theta) = -\,\mathbb{E}^\pi\left[\sum_{i=0}^{k} \nabla_\theta \log\left(\pi\left(a_{t+i}|x_{t+i};\theta\right)\right) A\left(x_{t+i}, a_{t+i};\theta\right) \right.$$
$$\left. + \cancel{\beta \sum_{i=0}^{k} \nabla_\theta H(\pi(\cdot|x_{t+i};\theta))}\right]$$

- Fully connected layers of policy network $\rightarrow$ noisy layers.

Metric

- **Human Normalised Score** (HNS):

$$100 \times \frac{\text{Score}_{\text{Agent}} - \text{Score}_{\text{Random}}}{\text{Score}_{\text{Human}} - \text{Score}_{\text{Random}}}$$

- HNS $= 0 \rightarrow$ as good as **random**
- HNS $= 100 \rightarrow$ **human** performance
- HNS $> 100 \rightarrow$ **superhuman** performance

Metric

- **Human Normalised Score** (HNS):

$$100 \times \frac{\text{Score}_{\text{Agent}} - \text{Score}_{\text{Random}}}{\text{Score}_{\text{Human}} - \text{Score}_{\text{Random}}}$$

- HNS $= 0 \rightarrow$ as good as **random**
- HNS $= 100 \rightarrow$ **human** performance
- HNS $> 100 \rightarrow$ **superhuman** performance

Metric

- **Human Normalised Score** (HNS):

$$100 \times \frac{\text{Score}_{\text{Agent}} - \text{Score}_{\text{Random}}}{\text{Score}_{\text{Human}} - \text{Score}_{\text{Random}}}$$

- $\text{HNS} = 0 \rightarrow$ as good as **random**
- $\text{HNS} = 100 \rightarrow$ **human** performance
- $\text{HNS} > 100 \rightarrow$ **superhuman** performance

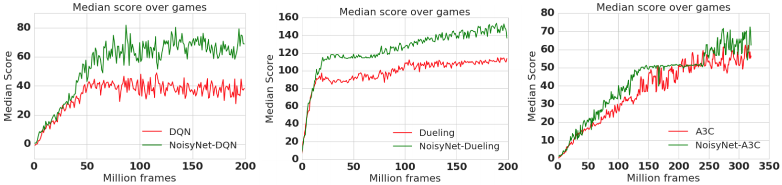
Metric

- **Human Normalised Score** (HNS):

$$100 \times \frac{\text{Score}_{\text{ Agent}} - \text{Score}_{\text{ Random}}}{\text{Score}_{\text{ Human}} - \text{Score}_{\text{ Random}}}$$

- HNS $= 0 \rightarrow$ as good as **random**
- HNS $= 100 \rightarrow$ **human** performance
- HNS $> 100 \rightarrow$ **superhuman** performance

Results



NoisyNets always produce superior performance in learning process.

Analysis of Learning in Noisy Layers

- $\bar{L}(\zeta)$ is a positive and continuous function of ζ.
- Always exist a **deterministic** optimiser for it.

Hypothesis

Learn to **discard noise** entries by pushing σ^w and σ^b to 0.

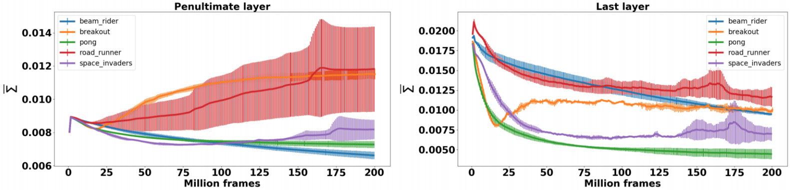


Figure: Learning of average noise parameters $\bar{\Sigma}$ in a NoisyNet-DQN

Analysis of Learning in Noisy Layers

- $\bar{L}(\zeta)$ is a positive and continuous function of ζ.
- Always exist a **deterministic** optimiser for it.

Hypothesis

Learn to **discard noise** entries by pushing σ^w and σ^b to 0.

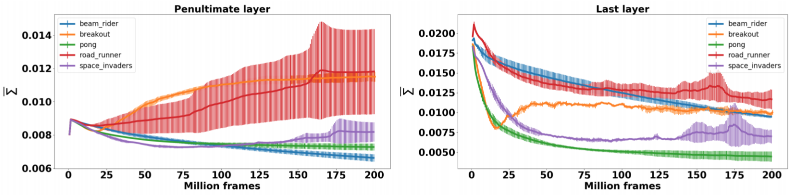


Figure: Learning of average noise parameters $\bar{\Sigma}$ in a NoisyNet-DQN

Analysis of Learning in Noisy Layers

- $\bar{L}(\zeta)$ is a positive and continuous function of ζ.
- Always exist a **deterministic** optimiser for it.

Hypothesis

Learn to **discard noise** entries by pushing σ^w and σ^b to 0.

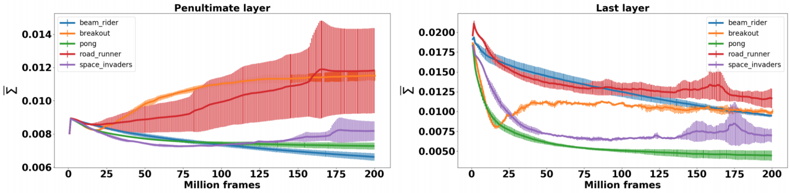


Figure: Learning of average noise parameters $\bar{\Sigma}$ in a NoisyNet-DQN

References

Noisy Networks for Exploration
M. Fortunato et al.

Curiosity-driven Exploration by Self-supervised Prediction
Deepak Pathak et al.

https://pathak22.github.io/noreward-rl/

Reinforcement Learning: An Introduction
Richard S. Sutton, Andrew G. Barto

www.cis.upenn.edu/~cis519/fall2015/lectures/14_ReinforcementLearning.pdf

An Introduction to Deep Reinforcement Learning
Bellemare et al.

Average Noise Parameters

Let σ_i^w denote the i^{th} weight of a noisy layer. Then,

$$\bar{\Sigma} := \frac{1}{N_{\text{weights}}} \sum_i |\sigma_i^w|$$

provides a **measure of the stochasticity of the Noisy layer**.

Factorised vs. Independent Noise in A3C