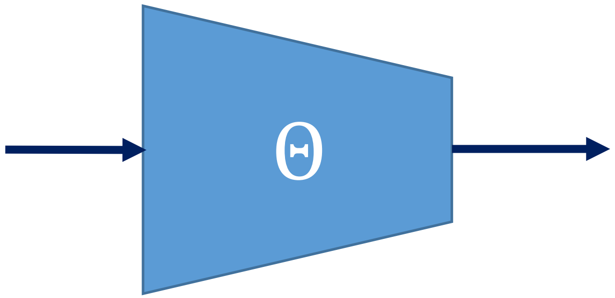
# Design a NN to play 3D ball balancing

2. NN with policy interacts with 3D Ball (MLAgent 10).ipynb

$\Theta$: neural network weights and biases

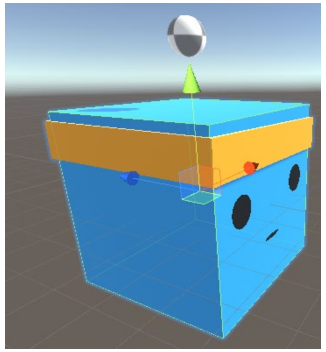
$s = (\theta_x, \theta_z, \Delta_x, \Delta_y, \Delta_z, v_x, v_x, v_x)$




$p_\Theta(\Delta\theta_x|s)$

$p_\Theta(\Delta\theta_z|s)$

8-128-128-2



# Policy gradient

3. NN with policy interacts with 3D Ball to collect training data (MLAgent_10).ipynb

$$\nabla \bar{R}_\theta \approx \frac{1}{N}\sum_{n=1}^{N} \boxed{\sum_{t=1}^{20}\left(\sum\nolimits_{t'}^{20} \gamma^{t'-t} r_{t'}^{n} - b\right)} \nabla \log p_\theta(a_t^n | s_t^n)$$

Δ = reward + expected accumulated reward
gae = Δ + accumulated gae
Return = gae + expected accumulated reward

$\Delta_{19} = r_{19} + (\gamma * v_{20} * mask_{19} - v_{19})$

$gae_{19\sim20} = \Delta_{19} + \gamma * \tau * mask_{19} * gae_{20}$

$return_{19} = gae_{19\sim20} + v_{19}$

...

$\Delta_{20} = r_{20} + (\gamma * v_{21} * mask_{20} - v_{20})$

$gae_{20} = \Delta_{20} + \gamma * \tau * mask_{20} * gae_{initial}$

$return_{20} = gae_{20} + v_{20}$

$\Delta_{1} = r_{1} + (\gamma * v_{2} * mask_{1} - v_{1})$

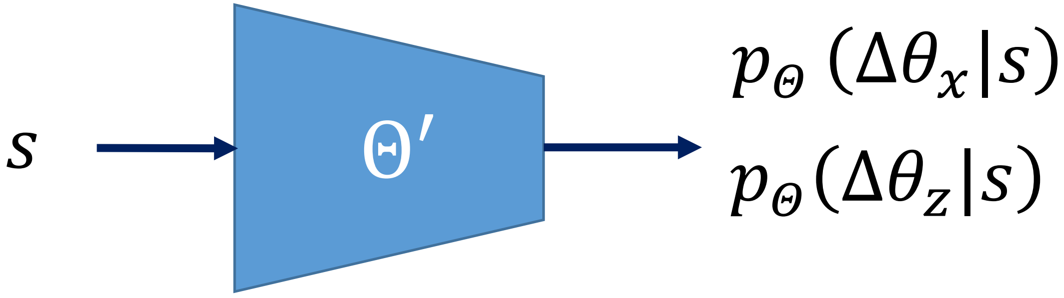
$gae_{1\sim20} = \Delta_{1} + \gamma * \tau * mask_{1} * gae_{2\sim20}$

$return_{1} = gae_{1\sim20} + v_{1}$

# Sampling efficiency problem of policy gradient

$$\nabla \bar{R}_\Theta \approx \boxed{\frac{1}{N}\sum_{n=1}^{N}\sum_{t=1}^{T_n}} \left(\sum_{t'}^{T_n} \gamma^{t'-t} r_{t'}^{n} - b\right) \nabla \log p_\Theta(a_t^n | s_t^n)$$

$$\Theta' \leftarrow \Theta + \eta \nabla \bar{R}_\Theta$$




$$\nabla \bar{R}_{\Theta'} \approx \boxed{\frac{1}{N}\sum_{n=1}^{N}\sum_{t=1}^{T_n}} \left(\sum_{t'}^{T_n} \gamma^{t'-t} r_{t'}^{n} - b\right) \nabla \log p_{\Theta'}(a_t^n | s_t^n)$$

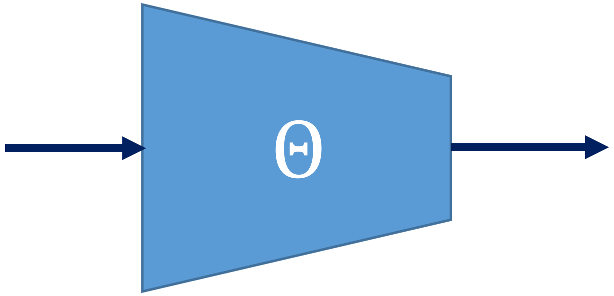
# PPO

4. NN optimization with PPO (MLAgent_10).ipynb

5. PPO (MLAgent_10) .ipynb

$$\Theta' \leftarrow \Theta + \eta \nabla \bar{R}_{\Theta}$$

$s$




$p_{\Theta}(\Delta\theta_x|s)$

$p_{\Theta}(\Delta\theta_z|s)$

$$\max_{\Theta'} \mathrm{PPO2}(\Theta')$$

$$\mathrm{PO2}(\Theta')=$$

$$\sum_{(s_t,a_t)} min\left(\frac{p_{\Theta'}(a_t|s_t)}{p_{\Theta}(a_t|s_t)}A^{\Theta}(s_t,a_t), clip\left(\frac{p_{\Theta'}(a_t|s_t)}{p_{\Theta}(a_t|s_t)}, 1-\varepsilon, 1+\varepsilon\right)A^{\Theta}(s_t,a_t)\right)$$

# Use expected value to reduce sampling variance

$V^{\pi_\theta}(s_t^n)$: Expected long-term return of the current state s under policy π.

$Q^{\pi_\theta}(s_t^n, a_t^n)$: Expected long-term return of the current state s, taking action a under policy π.

$$\boxed{V^{\pi_\theta}(s_t^n)}$$

Expected value of b

$$\nabla \bar{R}_\theta \approx \frac{1}{N}\sum_{n=1}^{N}\sum_{t=1}^{T_n}\left(\boxed{\sum_{t'}^{T_n}\gamma^{t'-t}r_{t'}^n} - \boxed{b}\right)\nabla \log p_\theta(a_t^n|s_t^n)$$

$$\boxed{E[G_t^n] = Q^{\pi_\theta}\ (s_t^n, a_t^n)}$$

Expected value of $G_t^n$

$$G_t^n = \sum_{t'}^{T_n}\gamma^{t'-t}r_{t'}^n$$

unstable when sampling amount is not large enough

Ref: 李弘毅 https://youtu.be/j82QLgfhFiY

# We only need to estimate $V(s)$

$$\nabla \bar{R}_\theta \approx \frac{1}{N}\sum_{n=1}^{N}\sum_{t=1}^{T_n}\left(\boxed{\sum_{t'}^{T_n}\gamma^{t'-t} r_{t'}^n} - \boxed{b}\right)\nabla \log p_\theta(a_t^n|s_t^n)$$

$$\boxed{V^{\pi_\theta}(s_t^n)}$$

$$\boxed{E[G_t^n] = Q^{\pi_\theta}(s_t^n, a_t^n)}$$

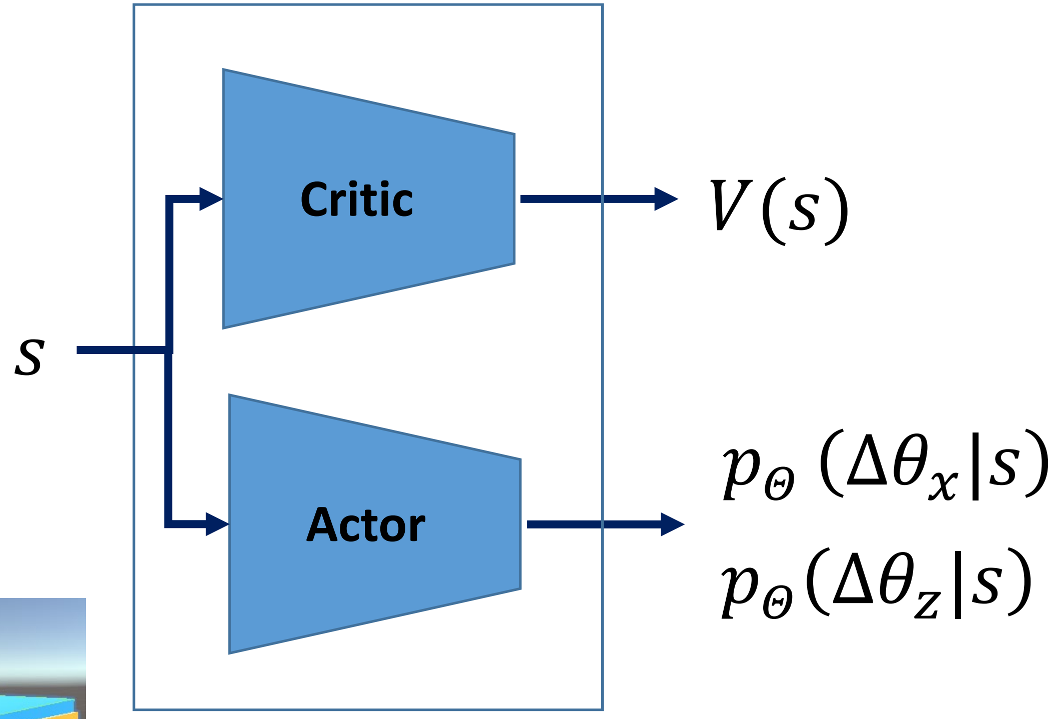
$$Q^{\pi_\theta}(s_t^n, a_t^n) = \mathrm{E}[r_t^n + V^{\pi_\theta}(s_{t+1}^n)] = r_t^n + V^{\pi_\theta}(s_{t+1}^n)$$

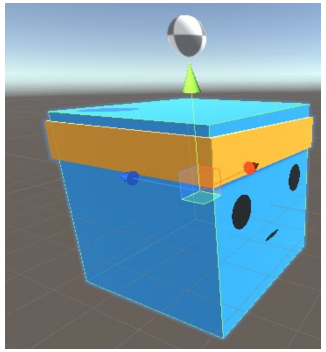
$$Q^{\pi_\theta}(s_t^n, a_t^n) - V^{\pi_\theta}(s_t^n) = r_t^n + V^{\pi_\theta}(s_{t+1}^n) - V^{\pi_\theta}(s_t^n)$$

$$A^\theta(s_t, a_t) = \left(r_t^n + V^{\pi_\theta}(s_{t+1}^n) - V^{\pi_\theta}(s_t^n)\right)$$

# A2C (Advantage Actor Critic)

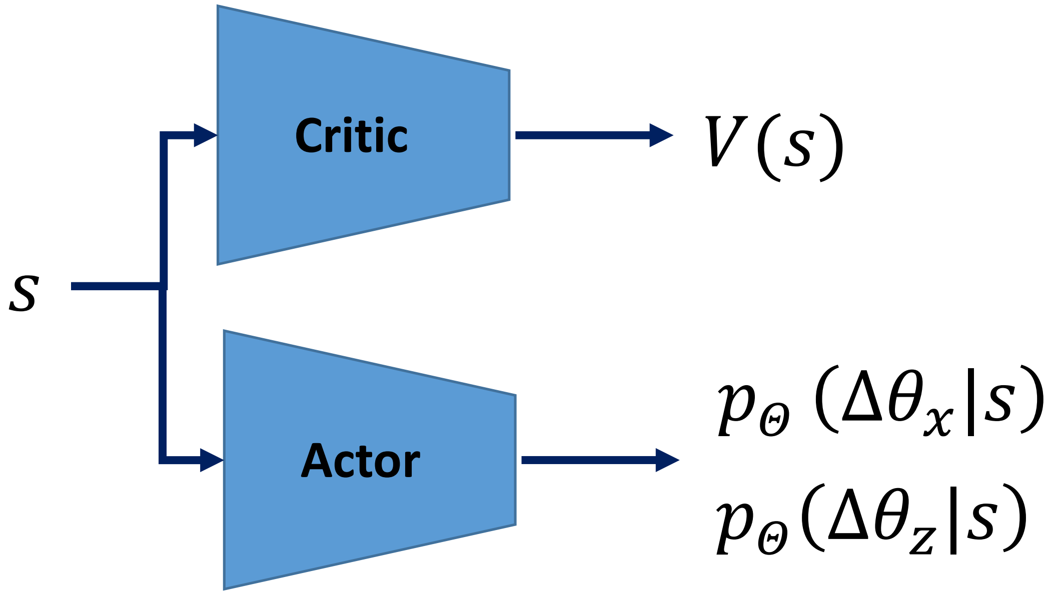
Actor – Learns the best actions (that can have maximum long-term rewards)
Critic – Learns the expected value of the long-term reward.



8-128-128-2

# We need to know the true answer of $V(s)$




$$\nabla \bar{R}_\theta \approx \frac{1}{N}\sum_{n=1}^{N}\sum_{t=1}^{T_n}\left(\sum\nolimits_{t'}^{T_n}\gamma^{t'-t} r_{t'}^{n} - b\right)\nabla \log p_\theta(a_t^n|s_t^n)$$

$$A^{\theta}(s_t, a_t) = \left(r_t^n + V^{\pi_\theta}(s_{t+1}^n) - V^{\pi_\theta}(s_t^n)\right)$$

# Use temporal difference to calculate $V(s)$

$$\nabla \bar{R}_\theta \approx \frac{1}{N}\sum_{n=1}^{N}\sum_{t=1}^{T_n}\left(\sum_{t'}^{T_n}\gamma^{t'-t} r_{t'}^n - b\right)\nabla \log p_\theta(a_t^n | s_t^n)$$

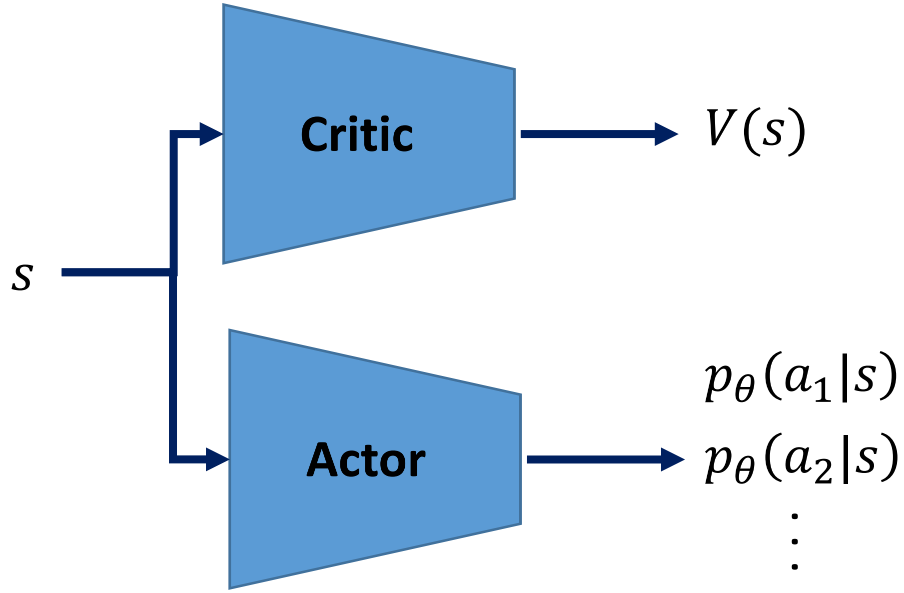
$$G_t^n = \sum_{t'}^{T_n}\gamma^{t'-t} r_{t'}^n$$

$V^{\pi_\theta}(s_a) \leftrightarrow G_a$ Monte-Carlo approach

$V^{\pi_\theta}(s_t) + r_t = V^{\pi_\theta}(s_{t+1})$ Temporal-difference approach

$V^{\pi_\theta}(s_t) - V^{\pi_\theta}(s_{t+1}) \leftrightarrow r_t$

# Define loss function to optimize the Actor-Critic networks



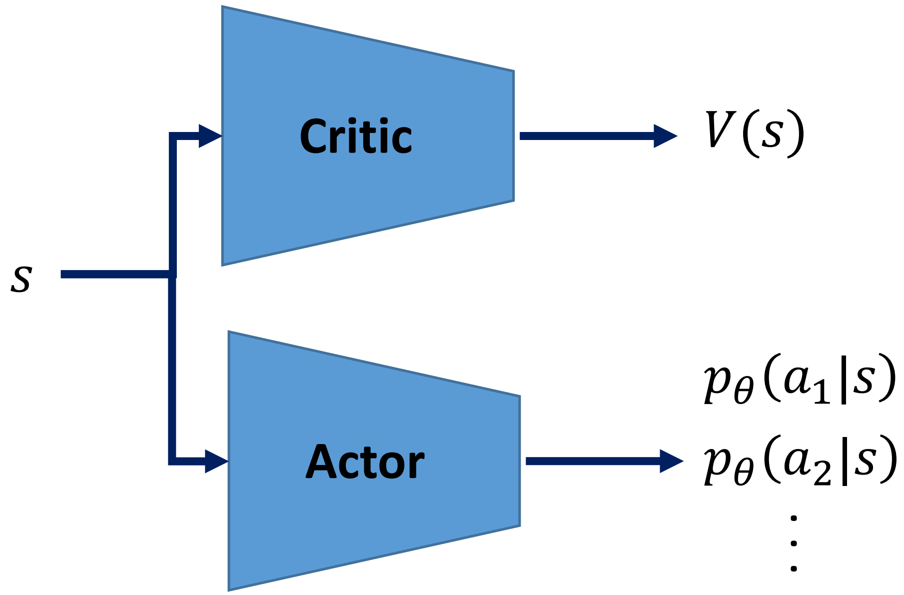

$$L = L_\pi + c_v L_v$$

$$Loss_\pi = \sum_{(s_t,a_t)} min\left(\frac{p_\theta(a_t|s_t)}{p_{\theta'}(a_t|s_t)} A^{\theta'}(s_t,a_t), clip\left(\frac{p_\theta(a_t|s_t)}{p_{\theta'}(a_t|s_t)}, 1-\varepsilon, 1+\varepsilon\right) A^{\theta'}(s_t,a_t)\right)$$

$$A^\theta(s_t,a_t) = r_t^n + \gamma V^{\pi_\theta}(s_{t+1}^n) - V^{\pi_\theta}(s_t^n)$$

$$Loss_v = \left(r_t^n + \gamma V^{\pi_\theta}(s_{t+1}^n) - V^{\pi_\theta}(s_t^n)\right)^2$$

# Add entropy-based regularization




$$L = L_\pi + c_v L_v + c_{reg} L_{reg}$$

# PyTorch implementation of A2C

6. A2C (MLAgent_10).ipynb

# 3DBall.yaml

```yaml
behaviors:
  3DBall:
    trainer_type: ppo
    hyperparameters:
      batch_size: 64
      buffer_size: 12000
      learning_rate: 0.0003
      beta: 0.001        c_reg
      epsilon: 0.2       ε
      lambd: 0.99        τ
      num_epoch: 3
      learning_rate_schedule: linear
    network_settings:
      normalize: true
      hidden_units: 128
      num_layers: 2
      vis_encode_type: simple
    reward_signals:
      extrinsic:
        gamma: 0.99      γ
        strength: 1.0
    keep_checkpoints: 5
    max_steps: 50000
    time_horizon: 1000
    summary_freq: 5000
    threaded: true
```

```python
N_STATES  = 8
N_ACTIONS =2

N_AGENTS = 3   #My Unity

BATCH_SIZE = 64
BUFFER_SIZE = 12000
LEARNING_RATE = 0.0003
BETA = 0.001
EPSILON = 0.2
LAMBD = 0.99
N_EPOCH = 3

GAMMA = 0.99

MAX_STEPS = 5000 #50000
TIME_HORIZON = 50 #1000
```

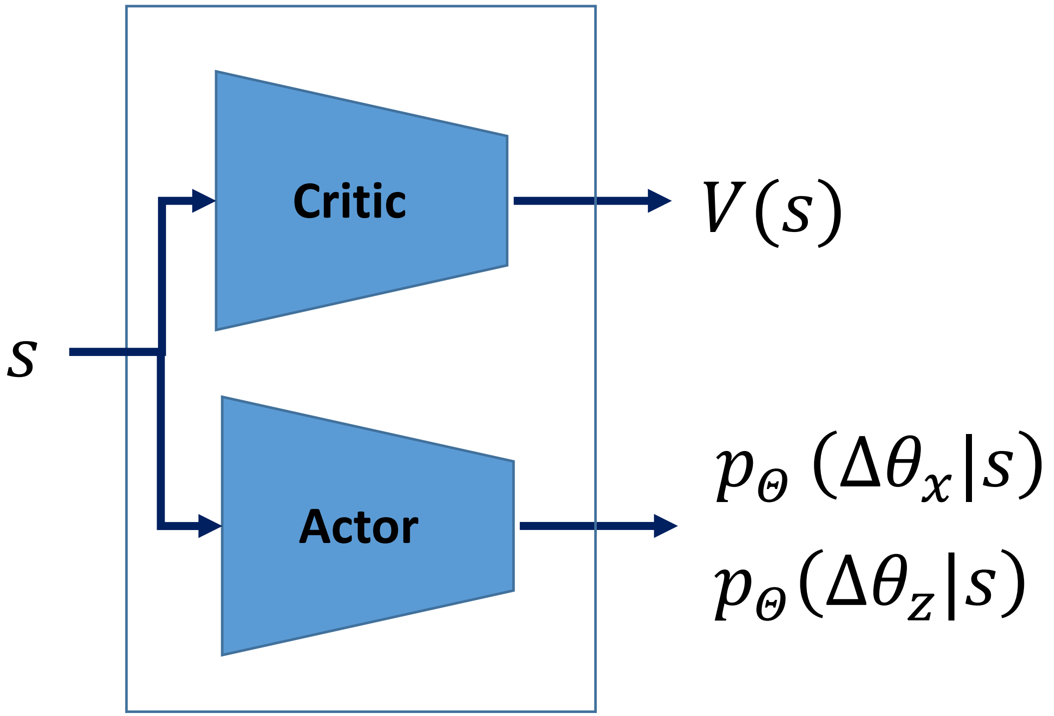
# PyTorch implementation of A2C

```python
class Net(nn.Module):
    def __init__(self, ):
        super(Net, self).__init__()

        self.critic = nn.Sequential(
            nn.Linear(N_STATES, 128),
            nn.LayerNorm(128),
            nn.Linear(128, 128),
            nn.LayerNorm(128),
            nn.Linear(128, 1)
        )

        self.actor = nn.Sequential(
            nn.Linear(N_STATES, 128),
            nn.LayerNorm(128),
            nn.Linear(128, 128),
            nn.LayerNorm(128),
            nn.Linear(128, N_ACTIONS)
        )
        self.log_std = nn.Parameter(torch.ones(1,
        self.apply(init_weights)

    def forward(self, x):
        value = self.critic(x)
        mu    = self.actor(x)
        std   = self.log_std.exp().expand_as(mu)
        dist  = Normal(mu, std)
        return dist, value
```




8-128-128-2

# PyTorch implementation of A2C

```
dist, value = net(batch_state.to(device))
critic_loss = (batch_return.to(device) - value).pow(2).mean()
entropy = dist.entropy().mean()
batch_action = dist.sample()
batch_new_log_probs = dist.log_prob(batch_action)
ratio = (batch_new_log_probs - batch_old_log_probs.to(device)).exp()
surr1 = ratio * batch_advantage.to(device)
surr2 = torch.clamp(ratio, 1.0 - clip_param, 1.0 + clip_param) * batc
actor_loss  = - torch.min(surr1, surr2).mean()
loss = 0.5 * critic_loss + actor_loss - 0.001 * entropy
```

$$L = L_\pi + c_v L_v + c_{reg} L_{reg}$$

$$Loss_v = \left(r_t^n + \gamma V^{\pi_\theta}(s_{t+1}^n) - V^{\pi_\theta}(s_t^n)\right)^2$$

$$Loss_\pi = \sum_{(s_t, a_t)} min\left(\frac{p_\theta(a_t|s_t)}{p_{\theta'}(a_t|s_t)} A^{\theta'}(s_t, a_t), clip\left(\frac{p_\theta(a_t|s_t)}{p_{\theta'}(a_t|s_t)}, 1-\varepsilon, 1+\varepsilon\right) A^{\theta'}(s_t, a_t)\right)$$

# PyTorch implementation of A2C

```
while (frame_idx < MAX_STEPS):
    print("\nframe idx = ", frame_idx)
    print("Interacts with Unity to collect training data")
    states, actions, log_probs, values, rewards, masks, next_state = collect_training_data (N_AGEN
    _, next_value = net(next_state.to(device))

    print("Compute GAE of these training data set")
    returns = compute_gae(TIME_HORIZON, next_value, rewards, masks, values, GAMMA, LAMBD)

    returns   = torch.cat(returns).detach()
    log_probs = torch.cat(log_probs).detach()
    values    = torch.cat(values).detach()
    states    = torch.cat(states)
    actions   = torch.cat(actions)
    advantages = returns - values

    print("Optimize NN with PPO")
    critic_loss, actor_loss = ppo_update(N_EPOCH, BATCH_SIZE, states, actions, log_probs, returns,
    CriticLossLst.append(critic_loss)
    ActorLossLst.append(actor_loss)

    frame_idx += TIME_HORIZON
```

# Use Tensorboard to visualize loss

6.1. A2C use Tensorboard (MLAgent_10).ipynb

# Use Tensorboard to visualize loss

```
!echo The current directory is %CD%
```

```
The current directory is C:\Users\ADMIN\Google 雲端硬碟\0
```

```
# Run python terminal window
# Change directory to this ipython notebook directory
# tensorboard --logdir=runs
```

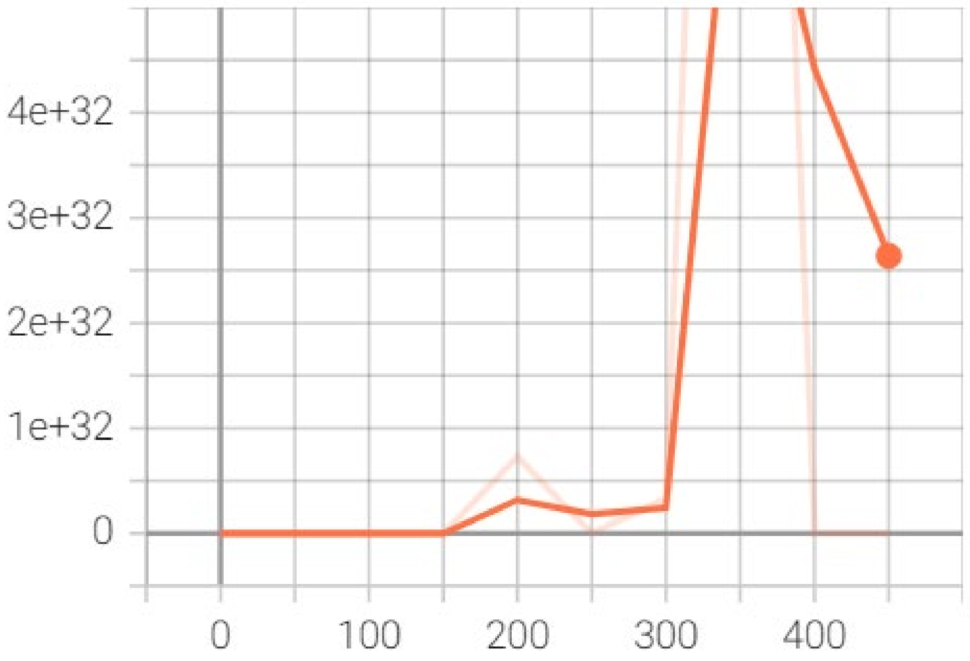
```
# Web browser:  localhost:6006
```

```
from torch.utils.tensorboard import SummaryWriter
writer = SummaryWriter()
```

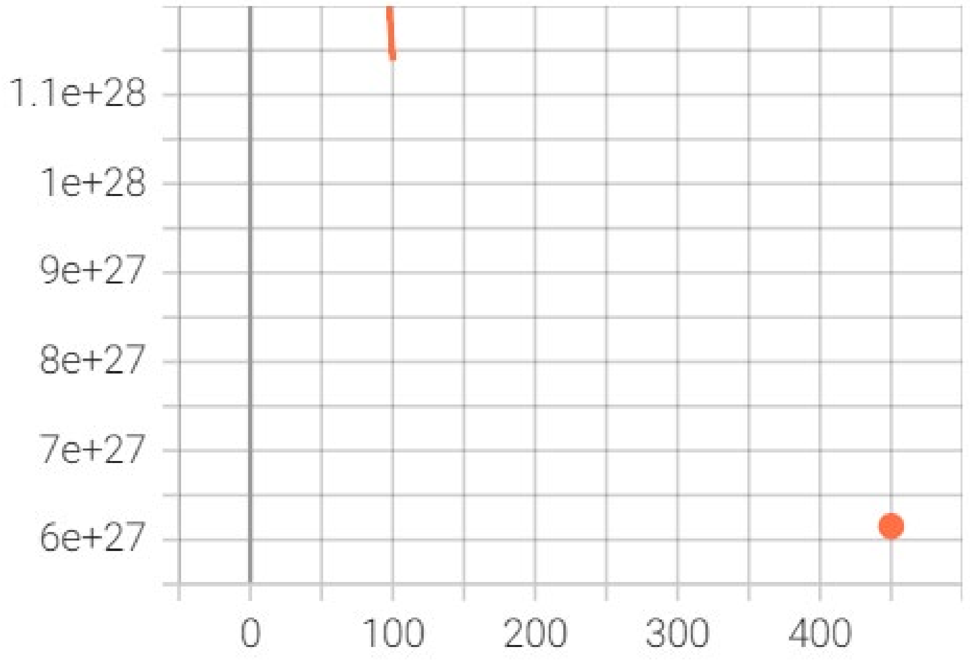
```
critic_loss, actor_loss = ppo_update(N_EPOCH, BATCH_SIZE,
writer.add_scalar("Actor loss", actor_loss, frame_idx)
writer.add_scalar("Critic loss", critic_loss, frame_idx)
```

# Use Tensorboard to visualize loss

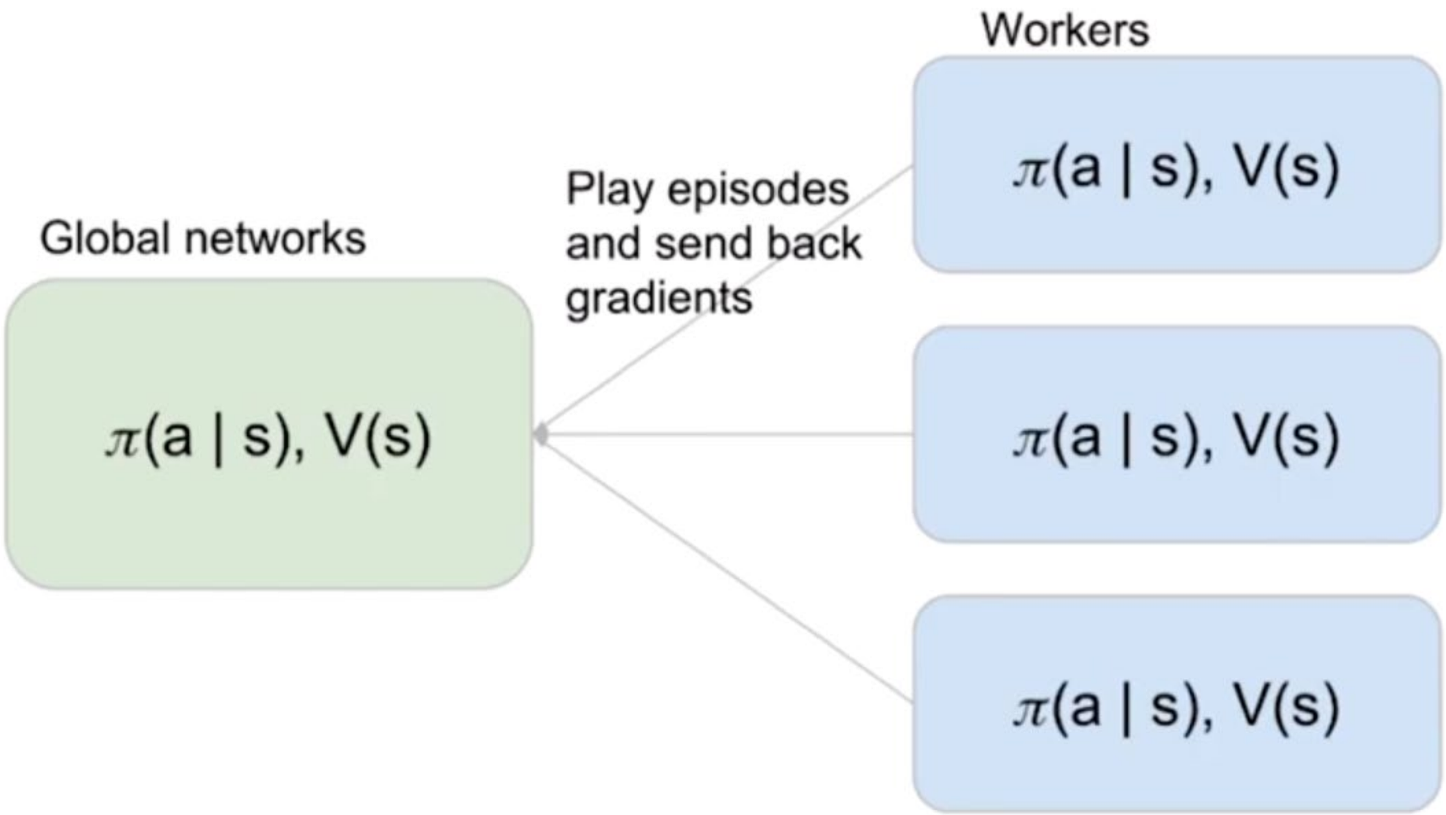
Actor loss




Critic loss



# A3C (Asynchronous Advantage Actor Critic)

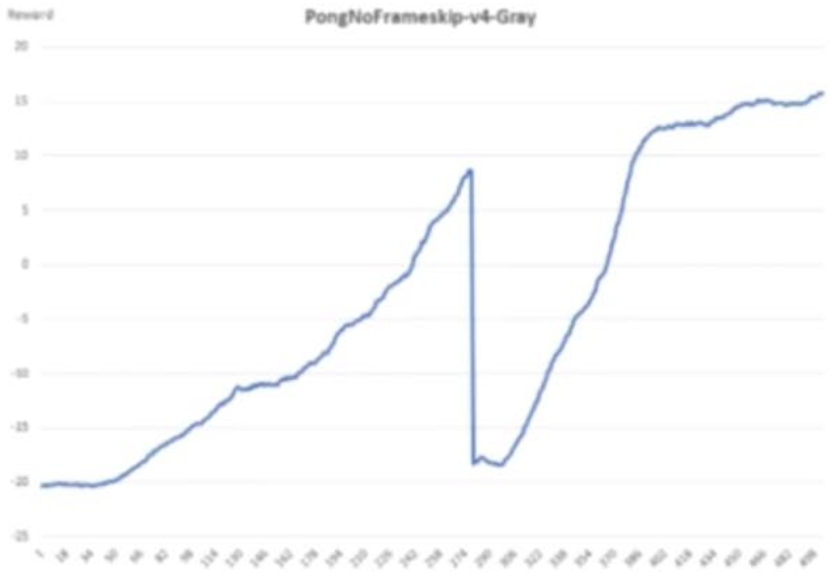



Reference: https://youtu.be/iCV3vOl8IMk

Mnih, V., Badia, A. P., Mirza, M., Graves, A., Lillicrap, T., Harley, T., ... & Kavukcuoglu, K. (2016, June). Asynchronous methods for deep reinforcement learning. In International conference on machine learning (pp. 1928-1937).

# A3C

- Each episode will progress randomly
- Each action is sampled probabilistically
- Occasionally, performance of agent can drop off due to bad update
    - Well, this can still happen with A3C so don't think you are immune




Reference: [https://youtu.be/iCV3vOl8IMk](https://youtu.be/iCV3vOl8IMk)

# A3C

- DQN is also interested in stabilizing learning
- Techniques:
  - Freezing target network
  - Experience replay buffer
- Use experience replay to look at multiple examples per training step
- A3C simply achieves stability using a different method (parallel agents)
- Both solve the problem: how to make neural networks work as function approximators in classic RL algorithms?

Reference: https://youtu.be/iCV3vOl8IMk

# A3C

- Remember: the theory part is not new, just need to create multiple parallel agents and asynchronously update/copy parameters
- 3 files:
  - main.py (master file; global policy and value networks)
    - Create and coordinate workers
  - worker.py (contains local policy and value networks)
    - Copy weights from global nets
    - Play episodes
    - Send gradients back to master
  - nets.py
    - Definition of policy and value networks

Reference: https://youtu.be/iCV3vOl8IMk

# A3C

## main.py

Instantiate global policy and value networks

Check # CPUs available, create threads and workers

Initialize global thread-safe counter, so every worker knows when to quit (when # of total steps reaches a max.)

Reference: [https://youtu.be/iCV3vOl8IMk](https://youtu.be/iCV3vOl8IMk)

# A3C

## worker.py

```
def run():
  in a loop:
    copy params from global nets to local nets
    run N steps of game (and store the data - s, a, r, s')
    using gradients wrt local net, update the global net
```

Conceptually, it's like:

$$1) \quad g_{local} = \frac{\partial L(\theta_{local})}{\partial \theta_{local}}$$

$$2) \quad \theta_{global} = \theta_{global} - \eta g_{local}$$

But in reality, we'll use RMSprop

Reference: https://youtu.be/iCV3vOl8IMk

# PPO-A3C

- PG $\nabla\bar{R}_\theta$
- Tips to reduce bias and variance in estimating $\nabla\bar{R}_\theta$
- Off-policy to improve efficiency of calculating $\nabla\bar{R}_\theta$
- Proximal policy optimization (PPO)
- Actor-critic strategy to reduce sampling variance in calculating $\nabla\bar{R}_\theta$
- Use temporal difference to calculate V(s)
- A3C