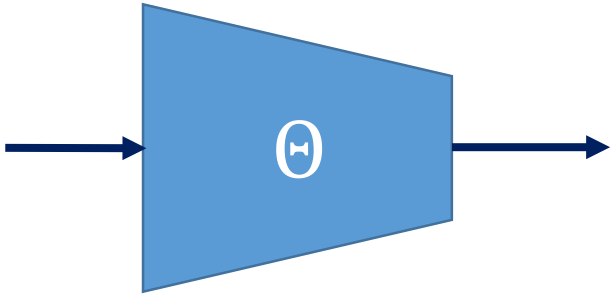
NN to play 3D ball balancing

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

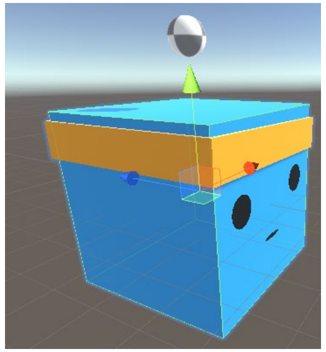
Θ: neural network weights and biases

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



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

8-128-128-2



Calculate GAE

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

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

Δ = reward + expected accumulated reward

gae = Δ + accumulated gae

Return = gae + expected accumulated reward

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

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

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

…

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

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

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

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

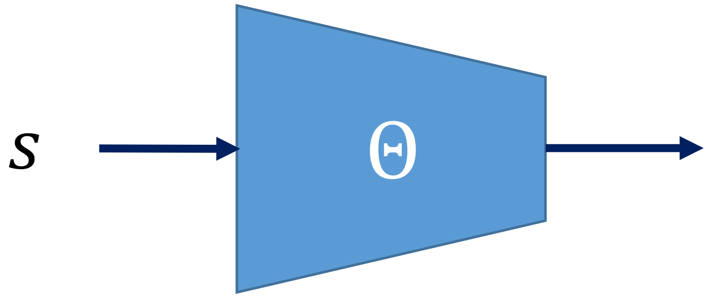
$$gae_{1\sim20} = \Delta_{1} + \gamma * \tau * mask_{1} * gae_{2\sim20}$$

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

Policy gradient to update NN

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

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



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

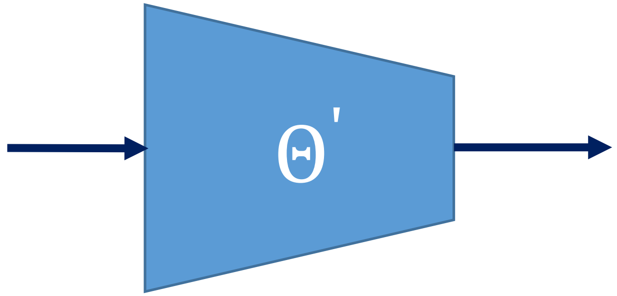
$$\nabla \bar{R}_{\Theta'} \approx \frac{1}{N}\sum_{n=1}^{N}\sum_{t=1}^{T_n}\left(\sum_{t'}^{T_n} \gamma^{t'-t} r_{t'}^{n} - b\right)\nabla \log p_{\Theta'}(a_t^n|s_t^n)$$

Sampling efficiency problem

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

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

s



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

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

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

Important sampling

$$E_{x\sim \mathrm{p}}[f(x)] = \int f(x)p(x)dx = \int f(x)\frac{p(x)}{q(x)}q(x)dx = E_{x\sim \mathrm{q}}\left[f(x)\frac{p(x)}{q(x)}\right]$$

$$Var_{x\sim \mathrm{p}}[f(x)] = E_{x\sim \mathrm{p}}[f(x)^2] - \left(E_{x\sim \mathrm{p}}[f(x)]\right)^2$$

$$\mathrm{VAR}[\mathrm{X}] = \mathrm{E}(\mathrm{X}^2) - (\mathrm{E}[\mathrm{X}])^2$$

$$Var_{x\sim \mathrm{q}}\left[f(x)\frac{p(x)}{q(x)}\right] = E_{x\sim \mathrm{q}}\left[\left(f(x)\frac{p(x)}{q(x)}\right)^2\right] - \left(E_{x\sim \mathrm{q}}\left[f(x)\frac{p(x)}{q(x)}\right]\right)^2$$

$$= E_{x\sim \mathrm{p}}\left[f(x)^2\frac{p(x)}{q(x)}\right] - \left(E_{x\sim \mathrm{p}}[f(x)]\right)^2$$

李弘毅: https://youtu.be/OAKAZhFmYoI

Off-policy to improve sampling efficiency

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

Notice: My interpretation about Θ and Θ' here is different from Lee, Hung-yi

$$\nabla \bar{R}_{\Theta'} = E_{\tau \sim p_{\Theta}(\tau)}[\frac{p_{\Theta'}(a_t|s_t)}{p_{\Theta}(a_t|s_t)}R(\tau)]$$

$$E_{x\sim p}[f(x)] = E_{x\sim q}\left[f(x)\frac{p(x)}{q(x)}\right]$$

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

$$\nabla \bar{R}_{\Theta'} = E_{(s_t,a_t)\sim \Theta}\left[\frac{p_{\Theta'}(a_t|s_t)}{p_{\Theta}(a_t|s_t)}A^{\Theta}(s_t,a_t)\nabla \log p_{\Theta}(a_t^n|s_t^n)\right]$$

From gradient to objective function

$$\nabla \bar{R}_{\Theta'} = E_{(s_t,a_t)\sim\Theta}\left[\frac{p_{\Theta'}(a_t|s_t)}{p_{\Theta}(a_t|s_t)} A^{\Theta}(s_t,a_t)\nabla \log p_{\Theta}(a_t^n|s_t^n)\right]$$

$$\max_{\Theta'} \bar{R}_{\Theta'}$$

$$\bar{R}_{\Theta'} = E_{(s_t,a_t)\sim\Theta}\left[\frac{p_{\Theta'}(a_t|s_t)}{p_{\Theta}(a_t|s_t)} A^{\Theta}(s_t,a_t)\right]$$

$$\nabla f(x) = f(x)\nabla \log f(x)$$

Proximal policy optimization (PPO)

$$\max_{\Theta'} \left(\bar{R}_{\Theta'} - \beta KL(\Theta', \Theta)\right) \qquad \bar{R}_{\Theta'} = E_{(s_t,a_t)\sim\Theta}\left[\frac{p_{\Theta'}(a_t|s_t)}{p_{\Theta}(a_t|s_t)} A^{\Theta}(s_t, a_t)\right]$$

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

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

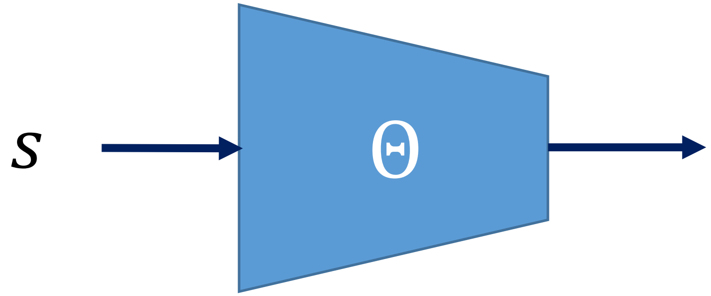
$$\sum_{(s_t,a_t)} min\left(\frac{p_{\Theta'}(a_t|s_t)}{p_{\Theta}(a_t|s_t)} A^{\Theta}(s_t, a_t), clip\left(\frac{p_{\Theta'}(a_t|s_t)}{p_{\Theta}(a_t|s_t)}, 1-\varepsilon, 1+\varepsilon\right) A^{\Theta}(s_t, a_t)\right)$$

Schulman, J., Wolski, F., Dhariwal, P., Radford, A., & Klimov, O. (2017). Proximal policy optimization algorithms. arXiv preprint arXiv:1707.06347.

Proximal policy optimization (PPO)

4. NN optimization with PPO (MLAgent_10).ipynb

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



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

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

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

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

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

Combine data collected from different agents

Use PPO to update NN weights and biases

```
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)
advantage = returns - values
```

```
print(len(returns), returns[0].shape)
print(len(log_probs), log_probs[0].shape)
print(len(values), values[0].shape)
print(len(states), states[0].shape)
print(len(actions), actions[0].shape)
print(len(advantage), advantage[0].shape)
```

```
60 torch.Size([1])
60 torch.Size([2])
60 torch.Size([1])
60 torch.Size([8])
60 torch.Size([2])
60 torch.Size([1])
```

N: no. of agents
K: time horizon

$$\begin{bmatrix} \vec{s}_{1,step1} \\ \vdots \\ \vec{s}_{N,step1} \\ \vdots \\ \vec{s}_{1,stepk} \\ \vdots \\ \vec{s}_{N,stepk} \end{bmatrix} \begin{bmatrix} \vec{a}_{1,step1} \\ \vdots \\ \vec{a}_{N,step1} \\ \vdots \\ \vec{a}_{1,stepk} \\ \vdots \\ \vec{a}_{N,stepk} \end{bmatrix} \quad \begin{bmatrix} v_{1,step1} \\ \vdots \\ v_{N,step1} \\ \vdots \\ v_{1,stepk} \\ \vdots \\ v_{N,stepk} \end{bmatrix} \begin{bmatrix} return_{1,step1} \\ \vdots \\ return_{N,step1} \\ \vdots \\ return_{1,stepk} \\ \vdots \\ return_{N,stepk} \end{bmatrix} \begin{bmatrix} gae_{1,step1} \\ \vdots \\ gae_{N,step1} \\ \vdots \\ gae_{1,stepk} \\ \vdots \\ gae_{N,stepk} \end{bmatrix}$$

Sampling a batch of data to train NN

```
batch_size = states.size(0)
for _ in range(batch_size // mini_batch_size):
    rand_ids = np.random.randint(0, batch_size, mini_batch_size)
    break
print(rand_ids)
print(states[rand_ids, :].shape)
print(actions[rand_ids, :].shape)
print(log_probs[rand_ids, :].shape)
print(returns[rand_ids, :].shape)
print(advantage[rand_ids, :].shape)
```

```
[39 52 11  8 45]
torch.Size([5, 8])
torch.Size([5, 2])
torch.Size([5, 2])
torch.Size([5, 1])
torch.Size([5, 1])
```

$$\begin{bmatrix} \vec{s}_{1,step1} \\ \vdots \\ \vec{s}_{N,step1} \\ \vdots \\ \vec{s}_{1,step20} \\ \vdots \\ \vec{s}_{N,step20} \end{bmatrix} \begin{bmatrix} \vec{a}_{1,step1} \\ \vdots \\ \vec{a}_{N,step1} \\ \vdots \\ \vec{a}_{1,step20} \\ \vdots \\ \vec{a}_{N,step20} \end{bmatrix} \quad \begin{bmatrix} return_{1,step1} \\ \vdots \\ return_{N,step1} \\ \vdots \\ return_{1,step20} \\ \vdots \\ return_{N,step20} \end{bmatrix} \begin{bmatrix} gae_{1,step1} \\ \vdots \\ gae_{N,step1} \\ \vdots \\ gae_{1,step20} \\ \vdots \\ gae_{N,step20} \end{bmatrix}$$

Select one batch of train data to optimize NN

$$\mathrm{PO2}(\Theta')=\sum_{(s_t,a_t)} min\left(\frac{p_{\Theta'}(a_t|s_t)}{p_{\Theta}(a_t|s_t)}A^{\Theta}(s_t,a_t), clip\left(\frac{p_{\Theta'}(a_t|s_t)}{p_{\Theta}(a_t|s_t)},1-\varepsilon,1+\varepsilon\right)A^{\Theta}(s_t,a_t)\right)$$

select one batch and perform PPO optimization

```
for batch_state, batch_action, batch_old_log_probs, batch_re
    break
```

```
print(batch_state.shape, batch_action.shape)
```

```
torch.Size([5, 8]) torch.Size([5, 2])
```

```
dist = net(batch_state.to(device))
print(dist)
```

```
Normal(loc: torch.Size([5, 2]), scale: torch.Size([5, 2]))
```

Select one batch of train data to optimize NN

$$\text{PO2}(\Theta')=\sum_{(s_t,a_t)} min\left(\frac{p_{\Theta'}(a_t|s_t)}{p_{\Theta}(a_t|s_t)}A^{\Theta}(s_t,a_t), clip\left(\frac{p_{\Theta'}(a_t|s_t)}{p_{\Theta}(a_t|s_t)},1-\varepsilon,1+\varepsilon\right)A^{\Theta}(s_t,a_t)\right)$$

```
batch_action = dist.sample()
batch_new_log_probs = dist.log_prob(batch_action)
print(batch_new_log_probs.shape)
```

```
torch.Size([5, 2])
```

```
ratio = (batch_new_log_probs - batch_old_log_probs.to(device)).exp()
print(ratio)
```

```
tensor([[1.2427, 0.6327],
        [0.4962, 1.2191],
        [0.8360, 1.0056],
        [1.4339, 0.3089],
        [1.3568, 0.3191]], device='cuda:0', grad_fn=<ExpBackward>)
```

Select one batch of train data to optimize NN

$$\text{PO2}(\Theta')=\sum_{(s_t,a_t)} min\left(\frac{p_{\Theta'}(a_t|s_t)}{p_{\Theta}(a_t|s_t)}A^{\Theta}(s_t,a_t), clip\left(\frac{p_{\Theta'}(a_t|s_t)}{p_{\Theta}(a_t|s_t)}, 1-\varepsilon, 1+\varepsilon\right)A^{\Theta}(s_t,a_t)\right)$$

```
surr1 = ratio * batch_advantage.to(device)
print(surr1)
```

```
tensor([[0.0606, 0.0309],
        [0.0242, 0.0595],
        [0.0408, 0.0491],
        [0.0699, 0.0151],
        [0.0662, 0.0156]], device='cuda:0', grad_fn=<MulBackward0>)
```

```
clip_param=0.2
surr2 = torch.clamp(ratio, 1.0 - clip_param, 1.0 + clip_param) * batc
print(surr2)
```

```
tensor([[0.0585, 0.0390],
        [0.0390, 0.0585],
        [0.0408, 0.0491],
        [0.0585, 0.0390],
        [0.0585, 0.0390]], device='cuda:0', grad_fn=<MulBackward0>)
```

```
actor_loss  = - torch.min(surr1, surr2).mean()
print(actor_loss)
```

```
tensor(-0.0410, device='cuda:0', grad_fn=<NegBackward>)
```

Select one batch of train data to optimize NN

$$PO2(\theta')=\sum_{(s_t,a_t)} min\left(\frac{p_{\Theta'}(a_t|s_t)}{p_{\Theta}(a_t|s_t)}A^{\Theta}(s_t,a_t), clip\left(\frac{p_{\Theta'}(a_t|s_t)}{p_{\Theta}(a_t|s_t)},1-\varepsilon,1+\varepsilon\right)A^{\Theta}(s_t,a_t)\right)$$

```
net = Net().to(device)
```

```
optimizer = optim.Adam(net.parameters(), lr=0.001)
```

```
actor_loss  = - torch.min(surr1, surr2).mean()
print(actor_loss)
```

```
tensor(-0.0410, device='cuda:0', grad_fn=<NegBackward>)
```

```
optimizer.zero_grad()
actor_loss.backward()
optimizer.step()
```

3DBall.yaml

```
behaviors:
  3DBall:
    trainer_type: ppo
    hyperparameters:
      batch_size: 64
      buffer_size: 12000
      learning_rate: 0.0003
      beta: 0.001
      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
```

Proximal policy optimization (PPO)

5. PPO (MLAgent_10) .ipynb