Proximal policy optimization

DEEP REINFORCEMENT LEARNING IN PYTHON

Timothée Carayol

Principal Machine Learning Engineer, Komment

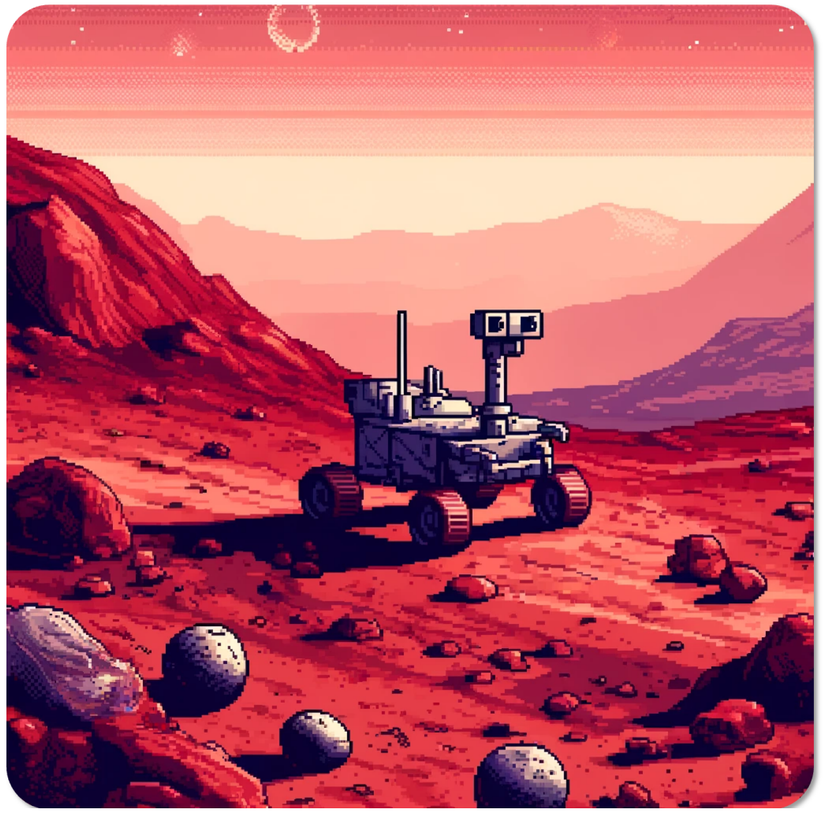
datacamp

A2C

- A2C policy updates:
  - Based on volatile estimates
  - Can be large and unstable
- May harm performance

PPO

- PPO sets limits on the size of each policy update
- Improves stability



The probability ratio

- PPO main innovation: a new objective function
- At its core:

Probability ratio

Ratio between the probability of an action under the new policy and the probability of the same action under the old policy.

New policy

$$r_t(\theta) = \frac{\pi_\theta(a_t|s_t)}{\pi_{\theta_{old}}(a_t|s_t)}$$

Policy from last update

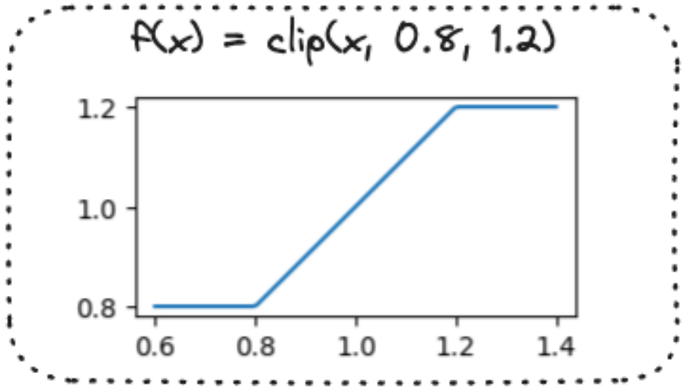
- How much more likely is action a_t with θ than with θ_{old}?

```
ratio = action_log_prob.exp() /
        old_action_log_prob.exp().detach()
# Or equivalently
ratio = torch.exp(action_log_prob -
        old_action_log_prob.detach())
```

- `detach` the denominator to prevent gradient propagation

Clipping the probability ratio

- Clip function:




```
clipped_ratio = torch.clamp(ratio,
                            1-epsilon,
                            1+epsilon)
```

Clipped probability ratio

Probability ratio

$$\text{clip}(r_t(\theta), 1-\varepsilon, 1+\varepsilon)$$

Lower bound

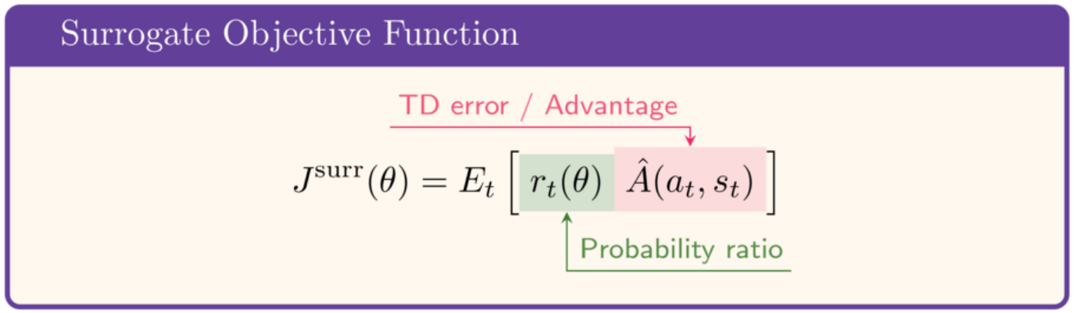
Upper bound

The calculate_ratios function

```
def calculate_ratios(action_log_prob, action_log_prob_old, epsilon):
    prob = action_log_prob.exp()
    prob_old = action_log_prob_old.exp()
    prob_old_detached = prob_old.detach()
    ratio = prob / prob_old_detached
    clipped_ratio = torch.clamp(ratio, 1-epsilon, 1+epsilon)
    return (ratio, clipped_ratio)
```

```
Example with epsilon = .2:
        Ratio: tensor(1.25)
Clipped ratio: tensor(1.20)
```

The PPO objective function

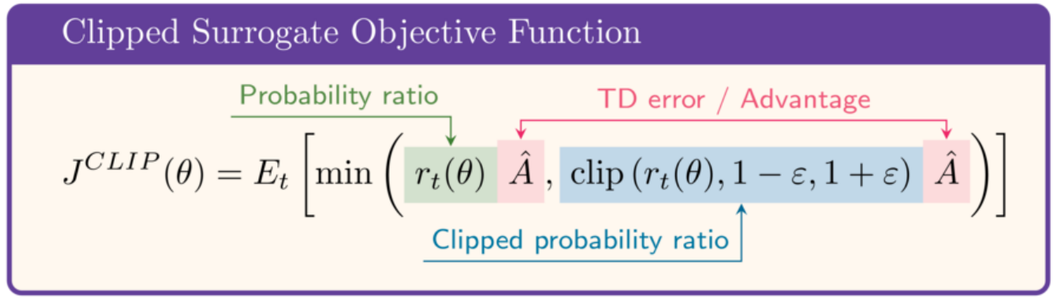



```
surr1 = ratio * td_error.detach()
surr2 = clipped_ratio * td_error.detach()
objective = torch.min(surr1, surr2)
```

- Surrogate with clipped ratio:

$$\text{clip}(r_t(\theta), 1-\varepsilon, 1+\varepsilon)\hat{A}$$

- PPO clipped surrogate objective function:



- More stable than A2C

PPO loss calculation

```
def calculate_losses(critic_network,
                     action_log_prob,
                     action_log_prob_old,
                     reward, state, next_state,
                     done
                     ):

    # calculate TD error (same as A2C)
    value = critic_network(state)
    next_value = critic_network(next_state)
    td_target = (reward +
                 gamma * next_value * (1-done))
    td_error = td_target - value
    ...
```

```
    ...
    ratio, clipped_ratio =
            calculate_ratios(action_log_prob,
                             action_log_prob_old,
                             epsilon)
    surr1 = ratio * td_error.detach()
    surr2 = clipped_ratio * td_error.detach()
    objective = torch.min(surr1, surr2)
    actor_loss = -objective
    critic_loss = td_error ** 2

    return actor_loss, critic_loss
```

Let's practice!

DEEP REINFORCEMENT LEARNING IN PYTHON

Entropy bonus and PPO

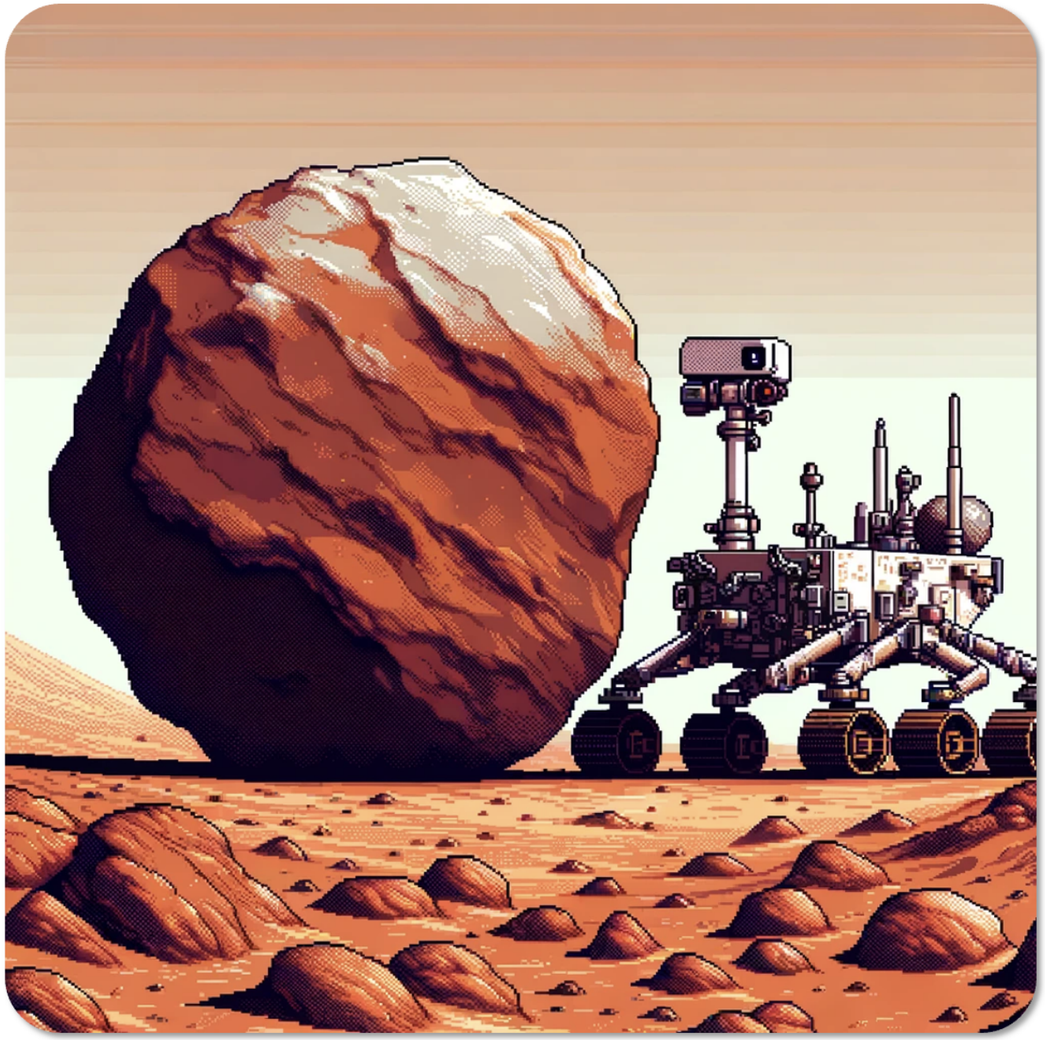
DEEP REINFORCEMENT LEARNING IN PYTHON

Timothée Carayol
Principal Machine Learning Engineer, Komment

datacamp

Entropy bonus

- Policy gradient algorithms may collapse into deterministic policies
- Solution: add entropy bonus
- Entropy measures uncertainty of a distribution!



Entropy of a probability distribution

Entropy

The entropy of a discrete random variable X, measured in bits, is defined as

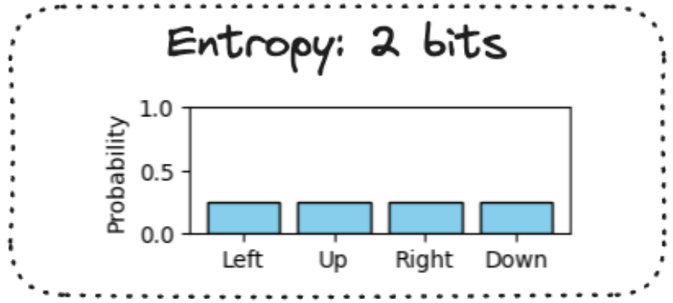
Sum over all possible values

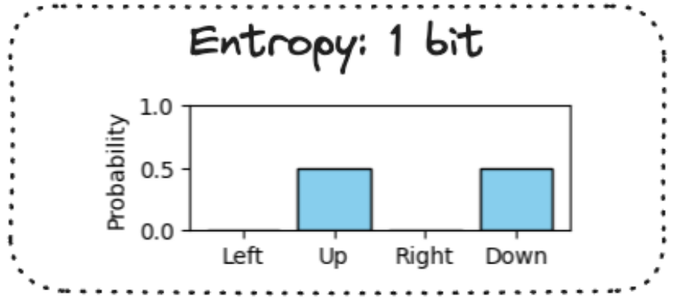
Base 2 logarithm

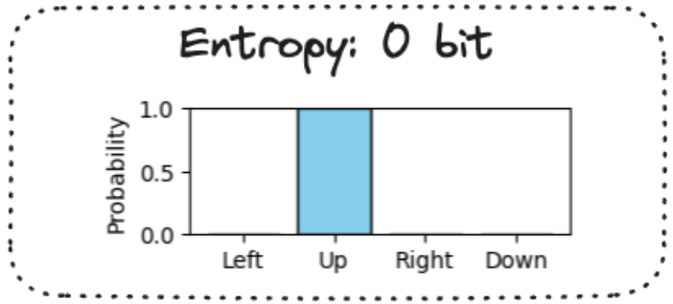
$$H(X) = -\sum_{x} p(x) \log_2 p(x)$$

Probability of value x

- If $\ln$ instead of $\log_2$: result measured in $nats$.
- $1\ nat = \frac{1}{\ln 2}\ bit \approx 1.44\ bit$







Implementing the entropy bonus

```
def select_action(policy_network, state):
  action_probs = policy_network(state)
  action_dist = Categorical(action_probs)
  action = action_dist.sample()
  log_prob = action_dist.log_prob(action)
  # Obtain the entropy of the policy
  entropy = action_dist.entropy()
  return (action.item(),
          log_prob.reshape(1),
          entropy)
```

- Actor loss: `actor_loss -= c_entropy * entropy`
- *Note:* `Categorical.entropy()` is in nats; divide by `math.log(2)` for bits

PPO training loop

```
for episode in range(10):
  state, info = env.reset()
  done = False
  while not done:
    action, action_log_prob, entropy = select_action(actor, state)
    next_state, reward, terminated, truncated, _ = env.step(action)
    done = terminated or truncated
    actor_loss, critic_loss = calculate_losses(critic, action_log_prob, action_log_prob,
                                               reward, state, next_state, done)
    actor_loss -= c_entropy * entropy
    actor_optimizer.zero_grad(); actor_loss.backward(); actor_optimizer.step()
    critic_optimizer.zero_grad(); critic_loss.backward(); critic_optimizer.step()
    state = next_state
```

Towards PPO with batch updates

- Updating at each step: not taking full advantage of PPO objective function
- At each step, θ actually coincides with θ_{old}.
- Full PPO implementations decouple:
  - Parameter updates (minibatches)
  - Policy updates (rollouts)

Let's practice!

DEEP REINFORCEMENT LEARNING IN PYTHON

Batch updates in policy gradient

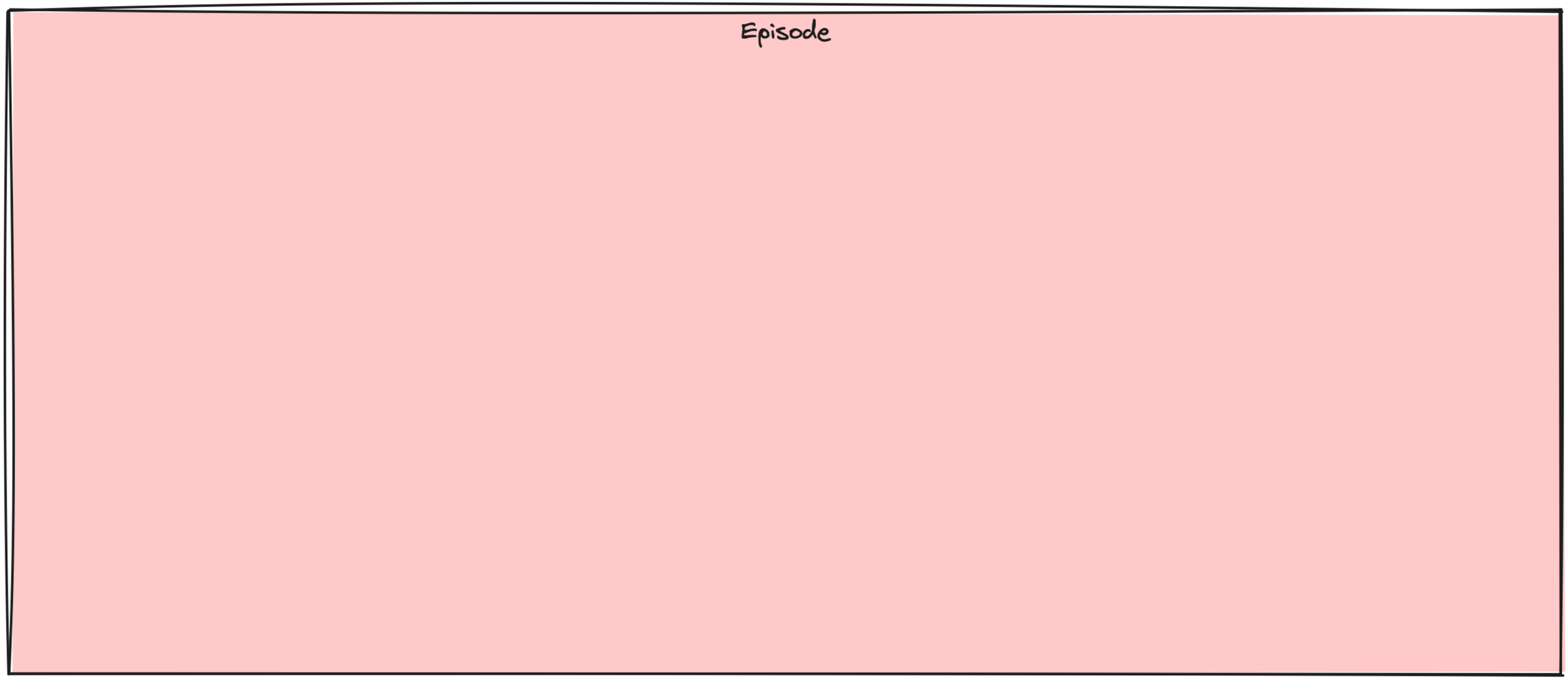
DEEP REINFORCEMENT LEARNING IN PYTHON

Timothée Carayol

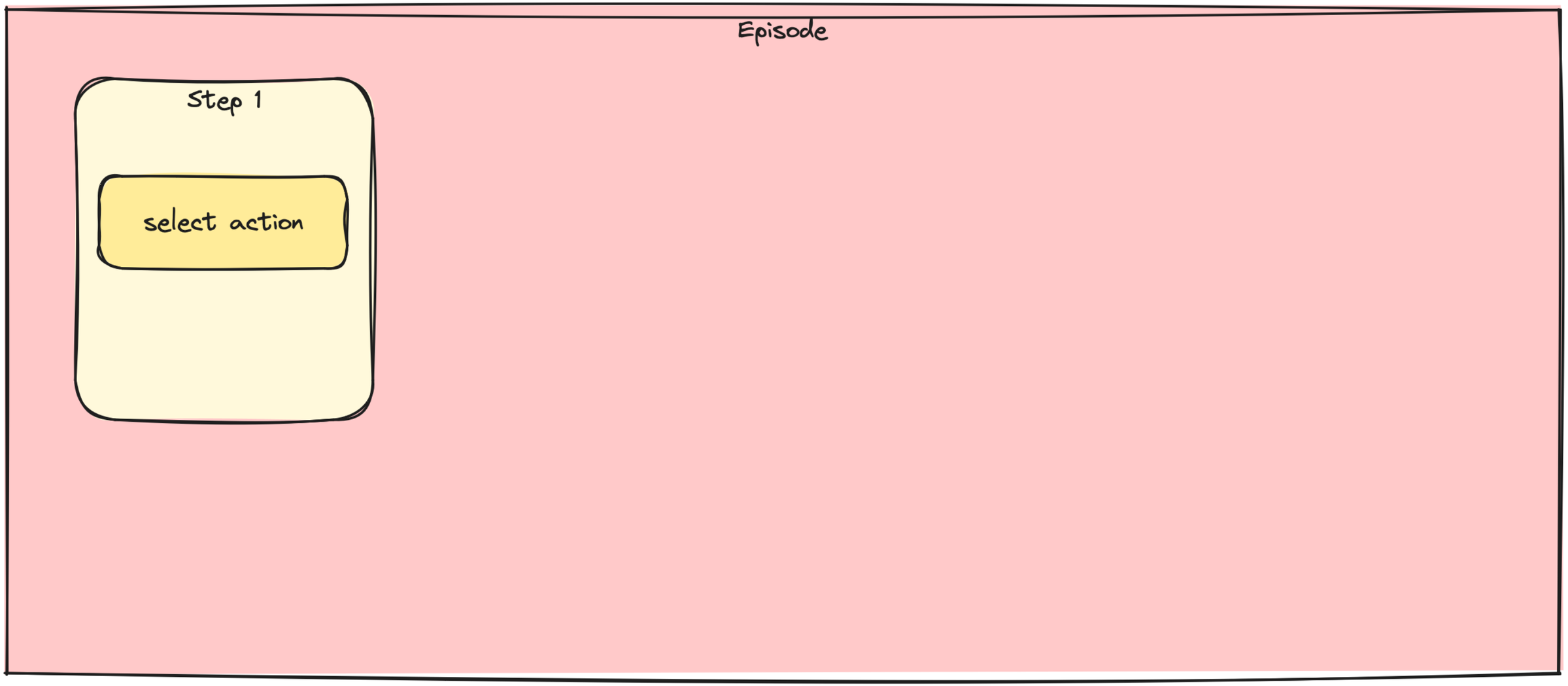
Principal Machine Learning Engineer, Komment

datacamp

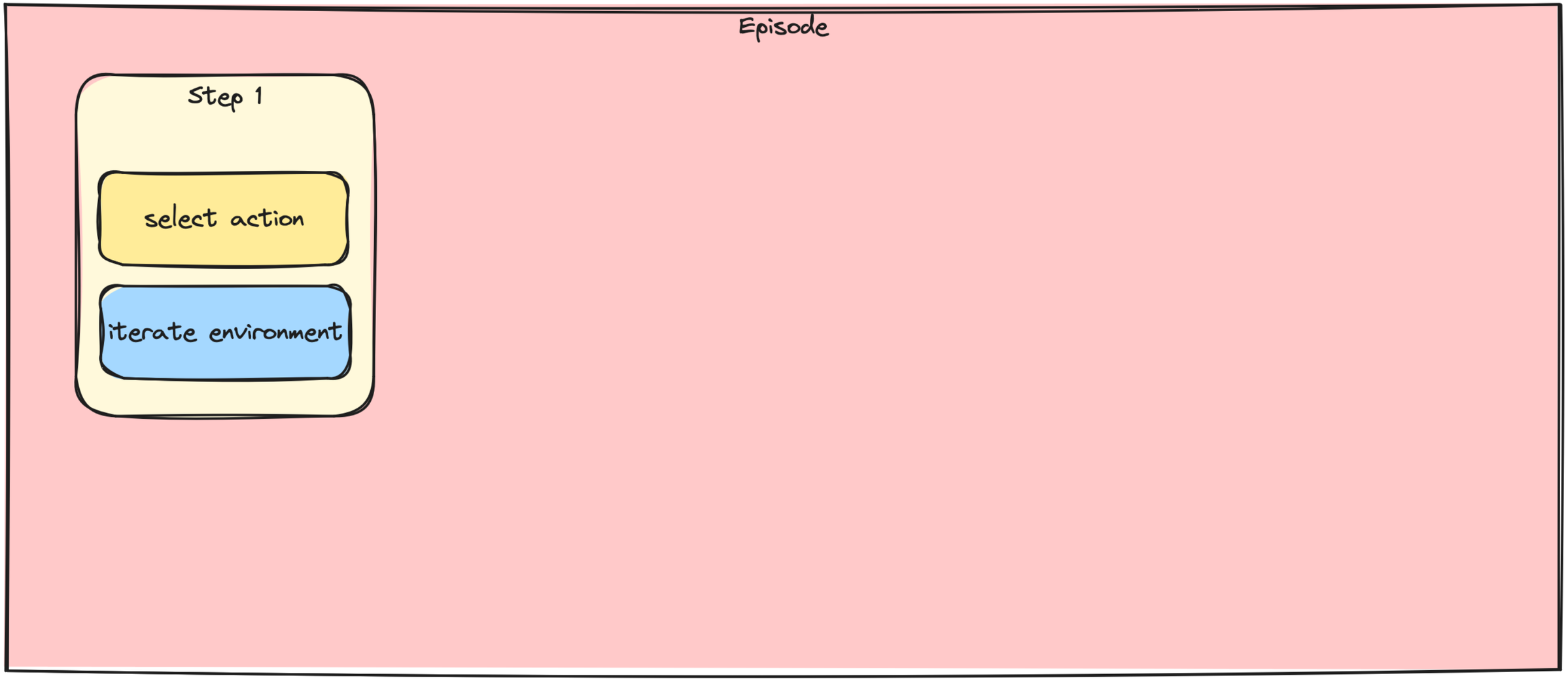
Stepwise vs batch gradient updates



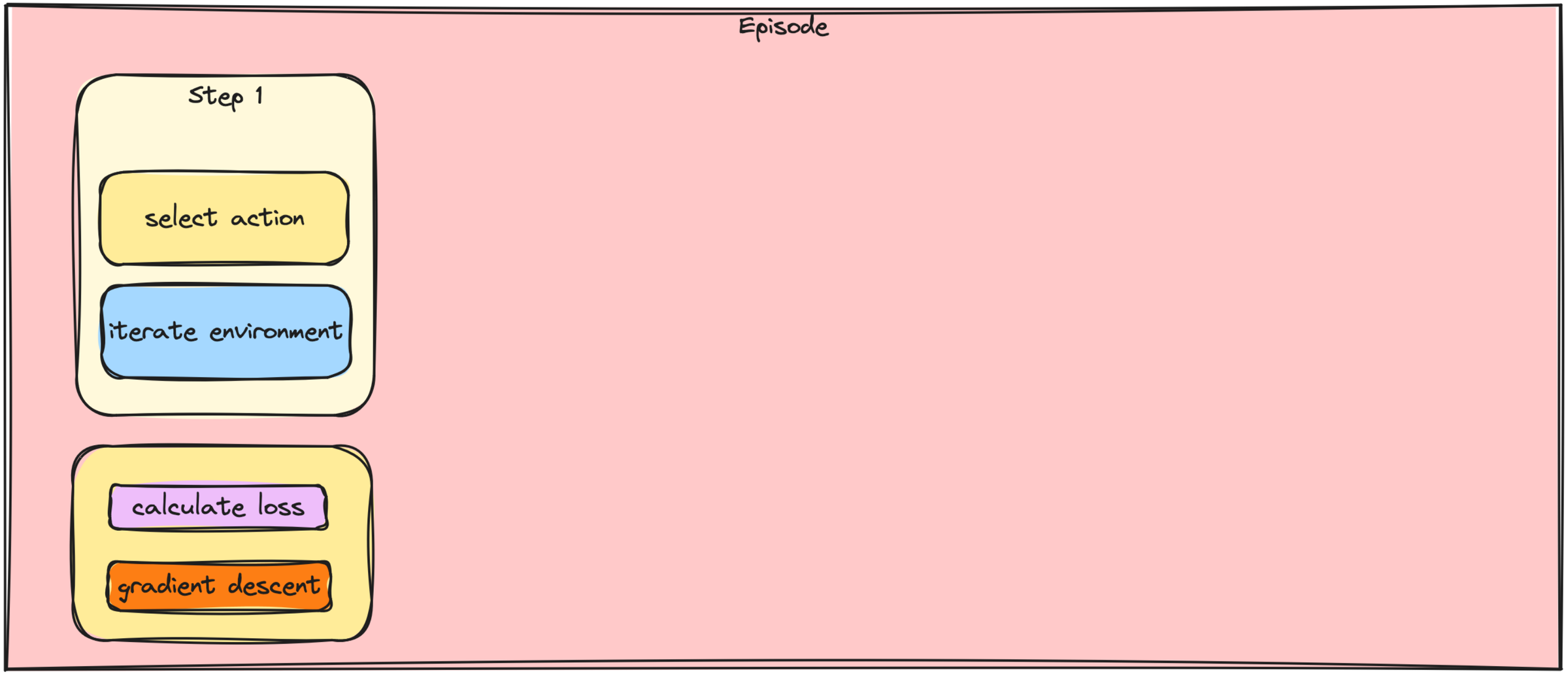
Stepwise vs batch gradient updates



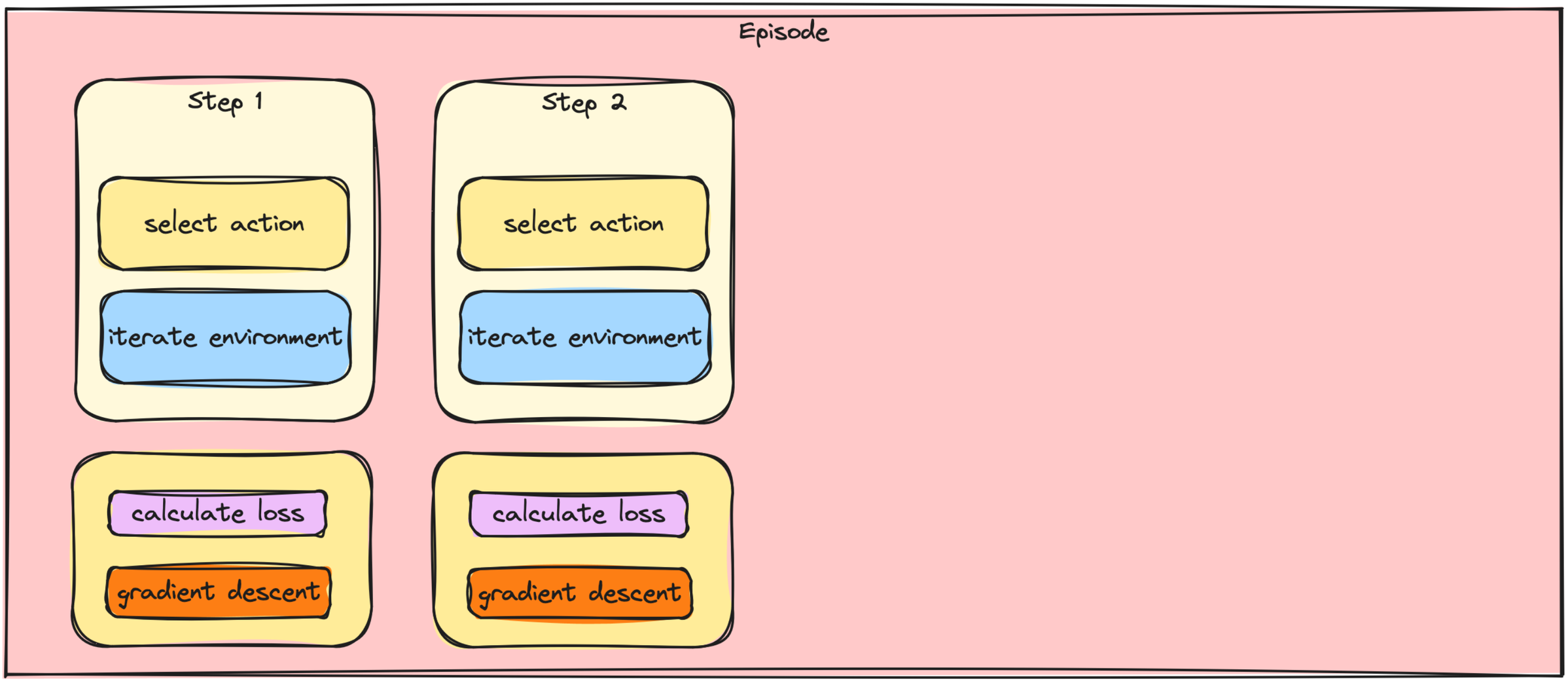
Stepwise vs batch gradient updates



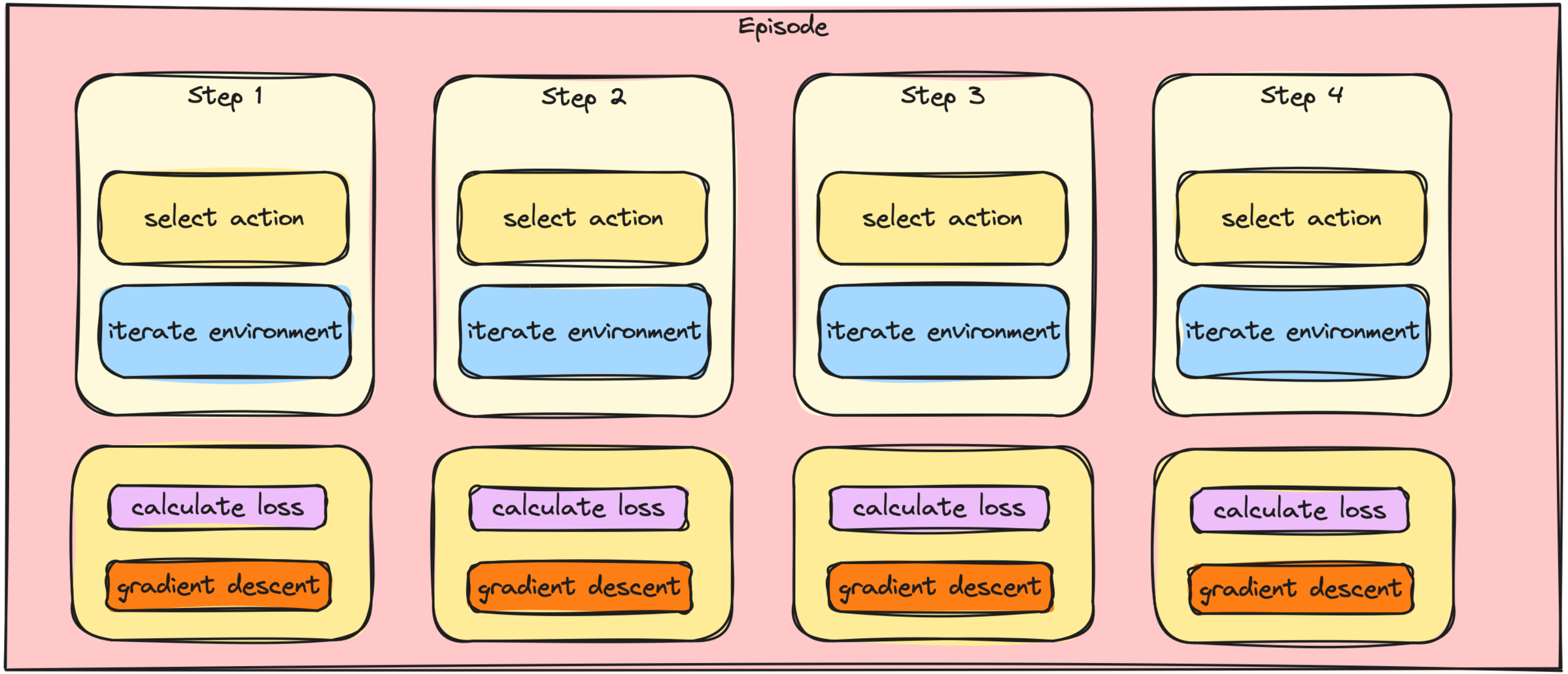
Stepwise vs batch gradient updates



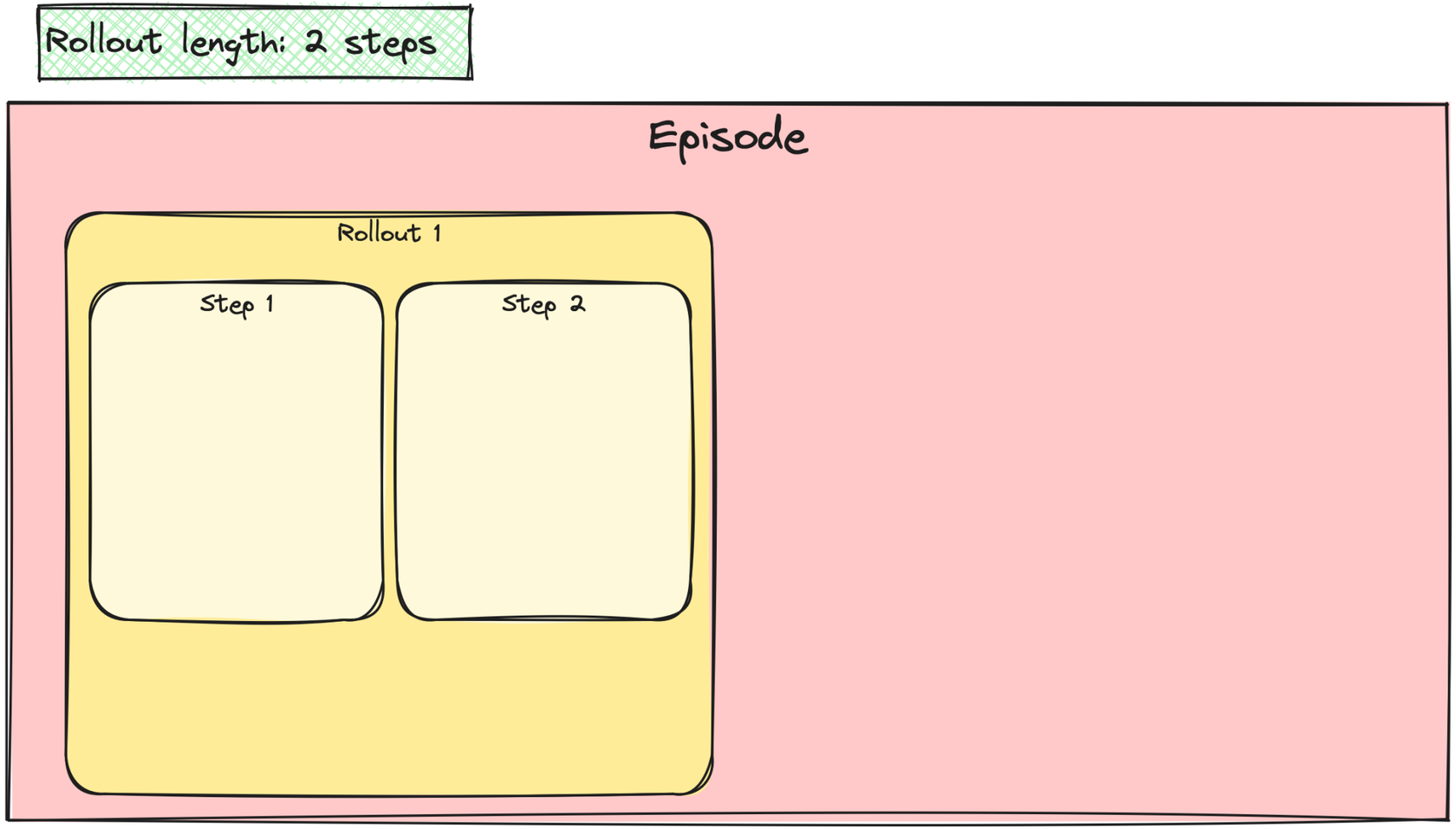
Stepwise vs batch gradient updates



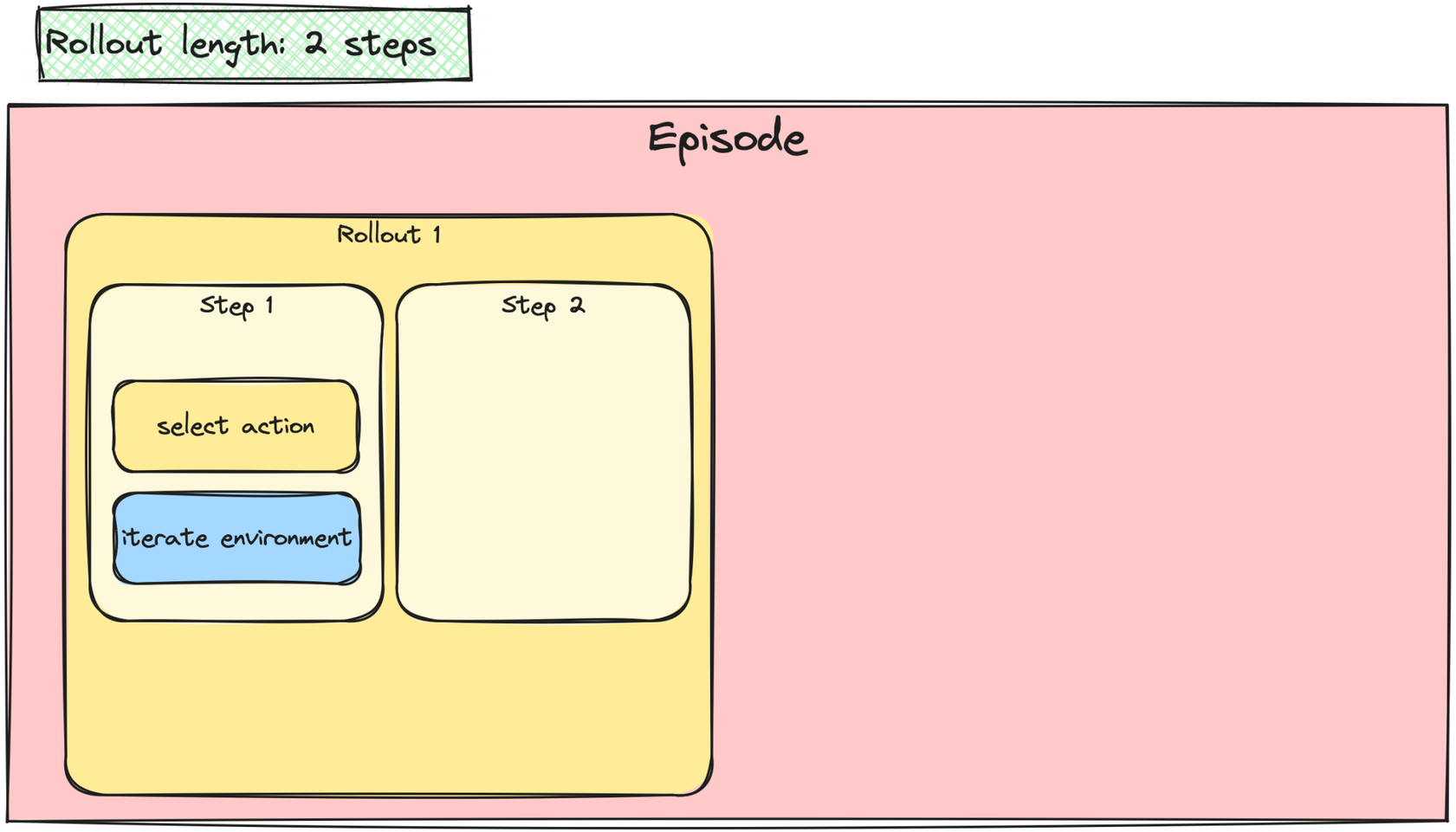
Stepwise vs batch gradient updates



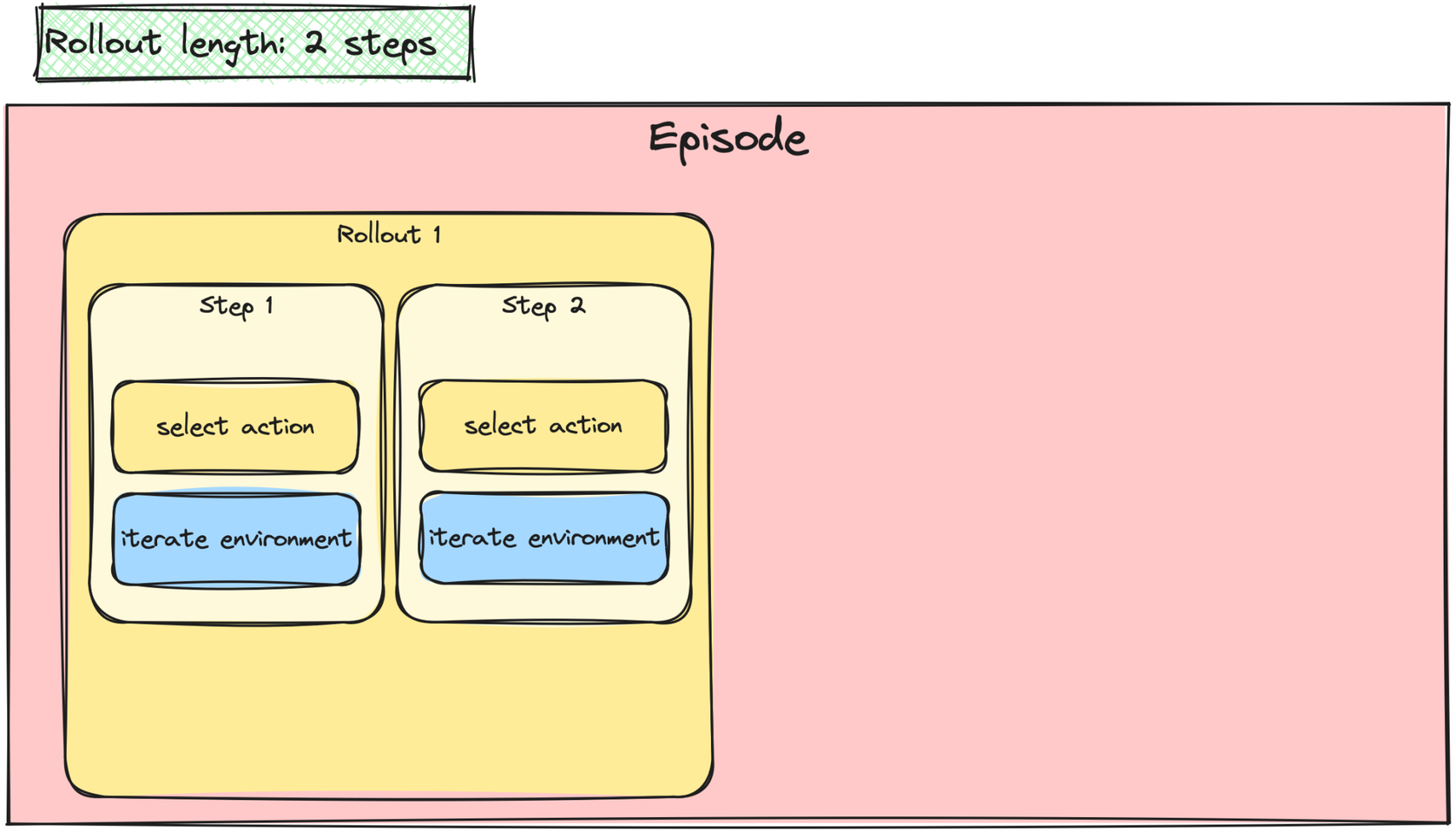
Batching the A2C / PPO updates



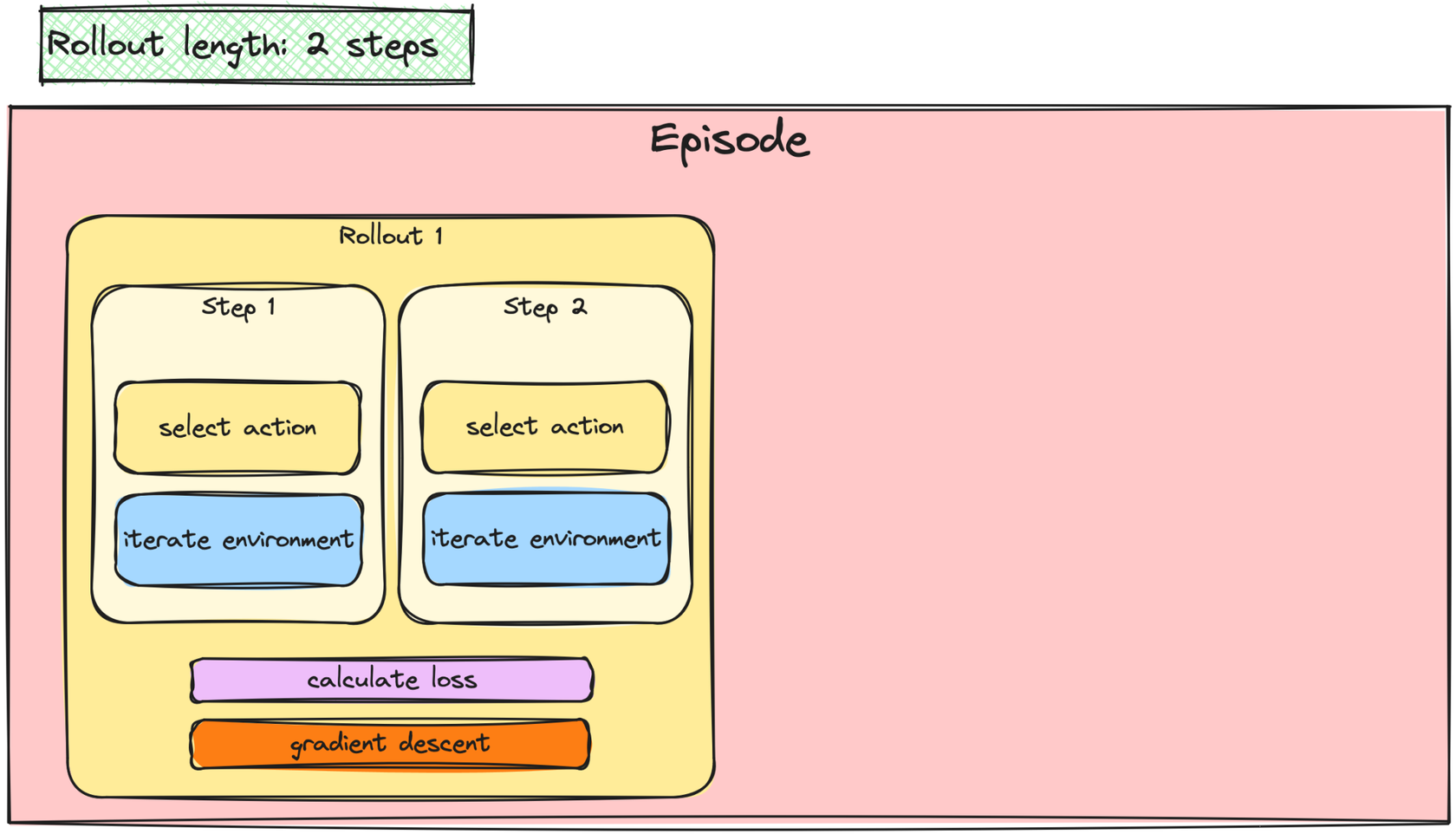
Batching the A2C / PPO updates



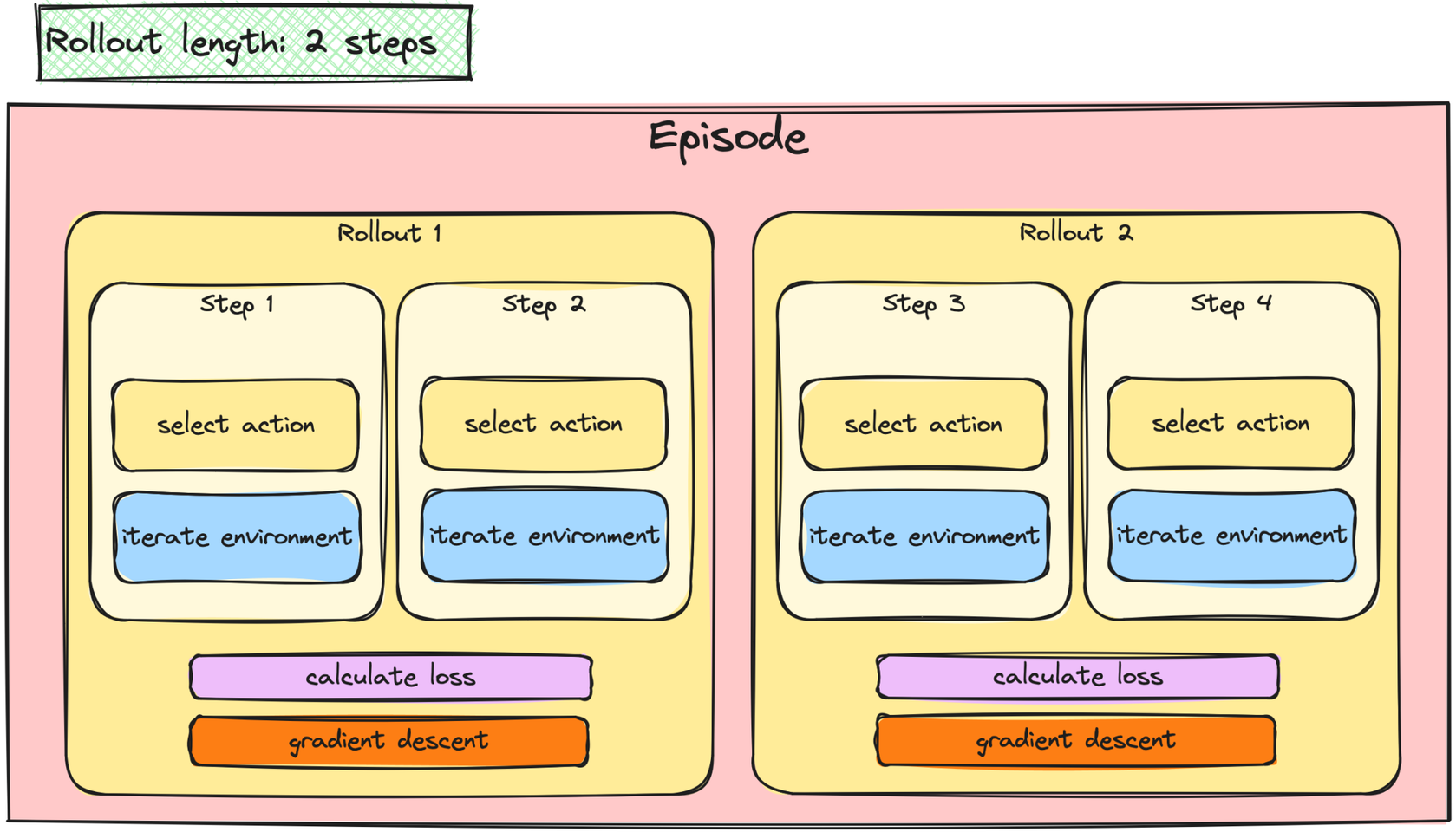
Batching the A2C / PPO updates



Batching the A2C / PPO updates



Batching the A2C / PPO updates



The A2C training loop with batch updates

```
# Set rollout length
rollout_length = 10
# Initiate loss batches
actor_losses = torch.tensor([])
critic_losses = torch.tensor([])
```

- Initiate loss batches
- Iterate through episodes and steps as usual

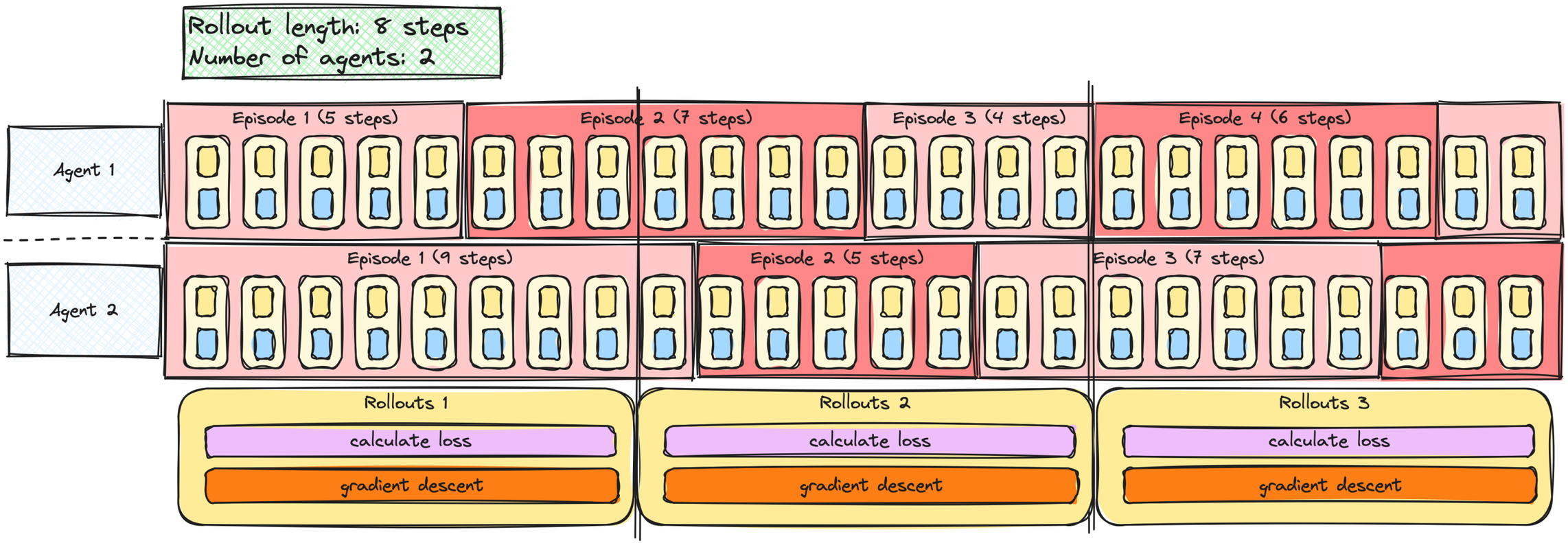
```
for episode in range(10):
  state, info = env.reset()
  done = False
  while not done:
    action, action_log_prob = select_action(actor,
                                            state)
    next_state, reward, terminated, truncated, _ = (
                                        env.step(action))
    done = terminated or truncated
    actor_loss, critic_loss = calculate_losses(
        critic, action_log_prob,
        reward, state, next_state, done)
    ...
```

The A2C training loop with batch updates

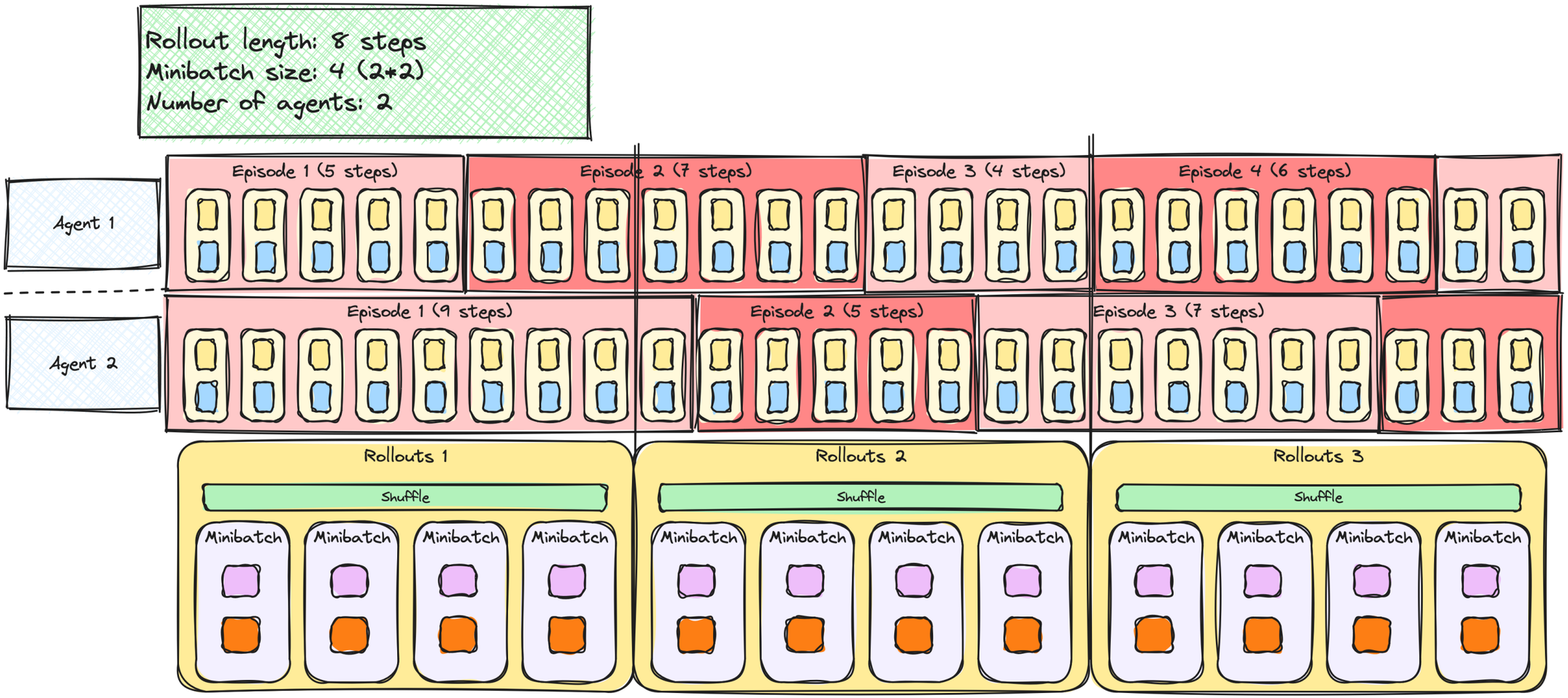
```
...
actor_losses = torch.cat((actor_losses, actor_loss))
critic_losses = torch.cat((critic_losses, critic_loss))

# If rollout is full, update the networks
if len(actor_losses) >= rollout_length:
  actor_loss_batch = actor_losses.mean()
  critic_loss_batch = critic_losses.mean()
  actor_optimizer.zero_grad()
  actor_loss_batch.backward()
  actor_optimizer.step()
  critic_optimizer.zero_grad()
  critic_loss_batch.backward()
  critic_optimizer.step()
  actor_losses = torch.tensor([])
  critic_losses = torch.tensor([])

state = next_state
```

- Append step loss to the loss batches
- When rollout is full:
  - Take the batch average loss with `.mean()`
  - Perform gradient descent
  - Reinitialize the batch losses

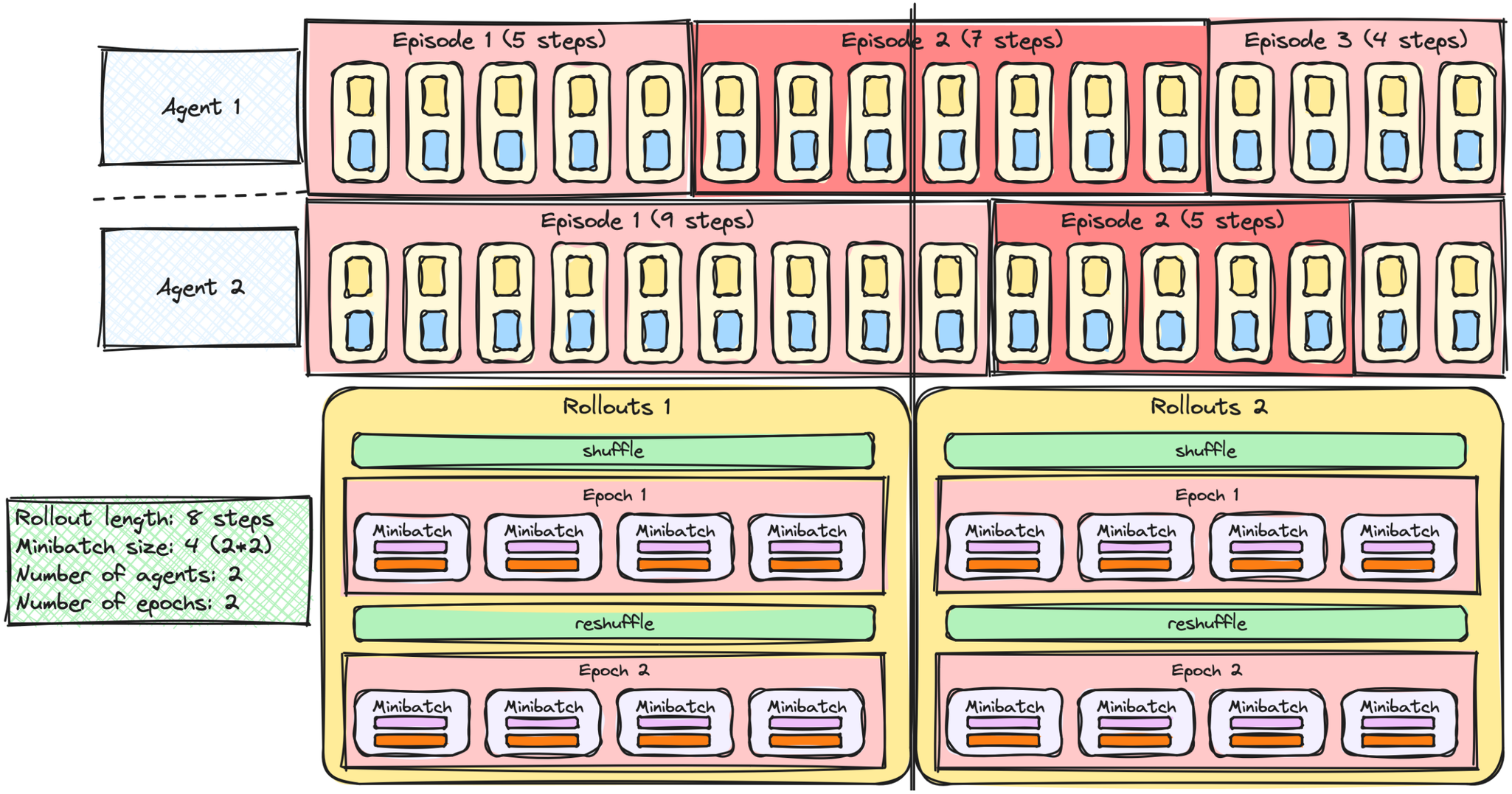
A2C / PPO with multiple agents



Rollouts and minibatches



PPO with multiple epochs



Let's practice!

DEEP REINFORCEMENT LEARNING IN PYTHON

Hyperparameter optimization with Optuna

DEEP REINFORCEMENT LEARNING IN PYTHON

Timothée Carayol

Principal Machine Learning Engineer, Komment

datacamp

What are hyperparameters

- Large number of hyperparameters in DRL algorithms
- Can have large effect on performance
- Search complexity grows with number of hyperparameters

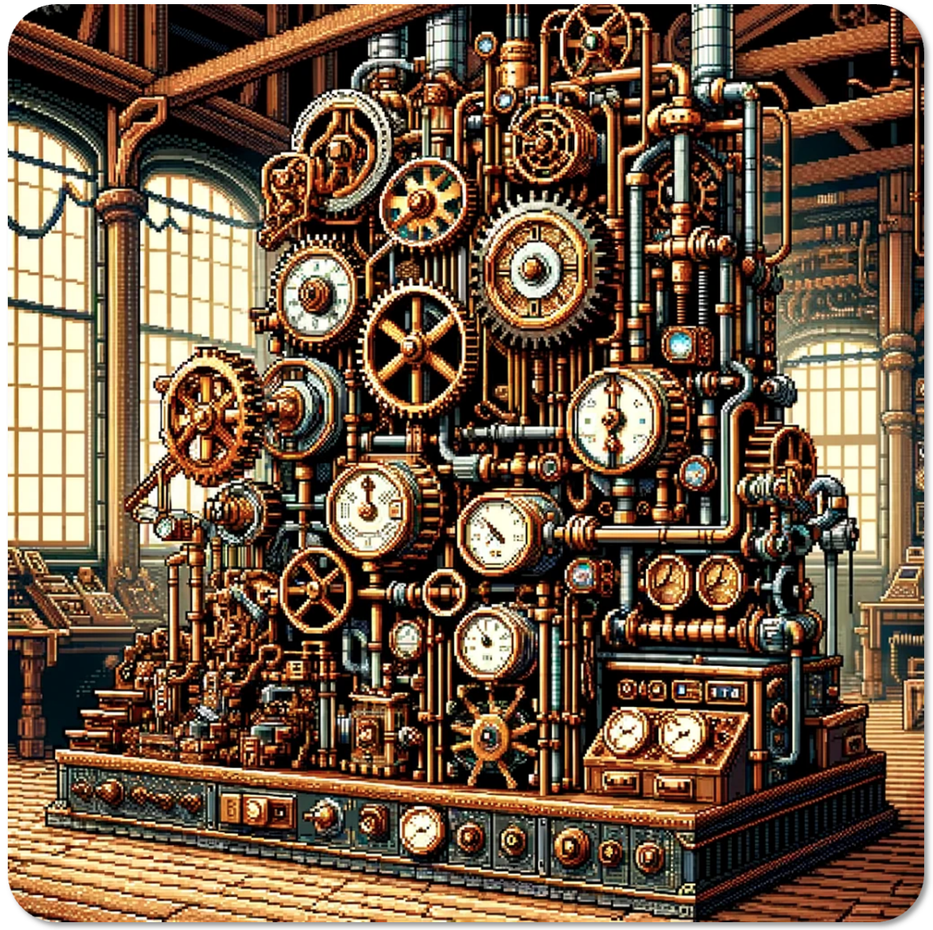
Examples
Discount rate
PPO: clipping epsilon, entropy bonus
Experience replay: buffer size, batch size
Decayed epsilon greediness schedule
Fixed Q-targets: τ
Learning rate
Number of layers, nodes per layer...

How to choose hyperparameter values

Objective: average cumulative rewards

Hyperparameter search techniques:

- Trial and error by hand
- Grid search
- Random search
- Dedicated algorithms



Optuna workflow:

- Define an objective function
- Instantiate an Optuna `study`
- Let Optuna iterate over trials

```
import optuna
def objective(trial):
    ...
study = optuna.create_study()
study.optimize(objective, n_trials=100)
```

```
study.best_params
```

```
{'learning_rate': 0.001292481, 'batch_size': 8}
```

Specifying the objective function

In the objective function:

- Define the hyperparameters of interest
- Define the metric(s) to optimize

Offers full flexibility on hyperparameter specification:

- float
- integer
- categorical

```
def objective(trial: optuna.trial.Trial):
    # Hyperparameters x and y between -10 and 10
    x = trial.suggest_float('x', -10, 10)
    y = trial.suggest_float('y', -10, 10)
    # Return the metric to minimize
    return (x - 2) ** 2 + 1.2 * (y + 3) ** 2
```

The optuna study

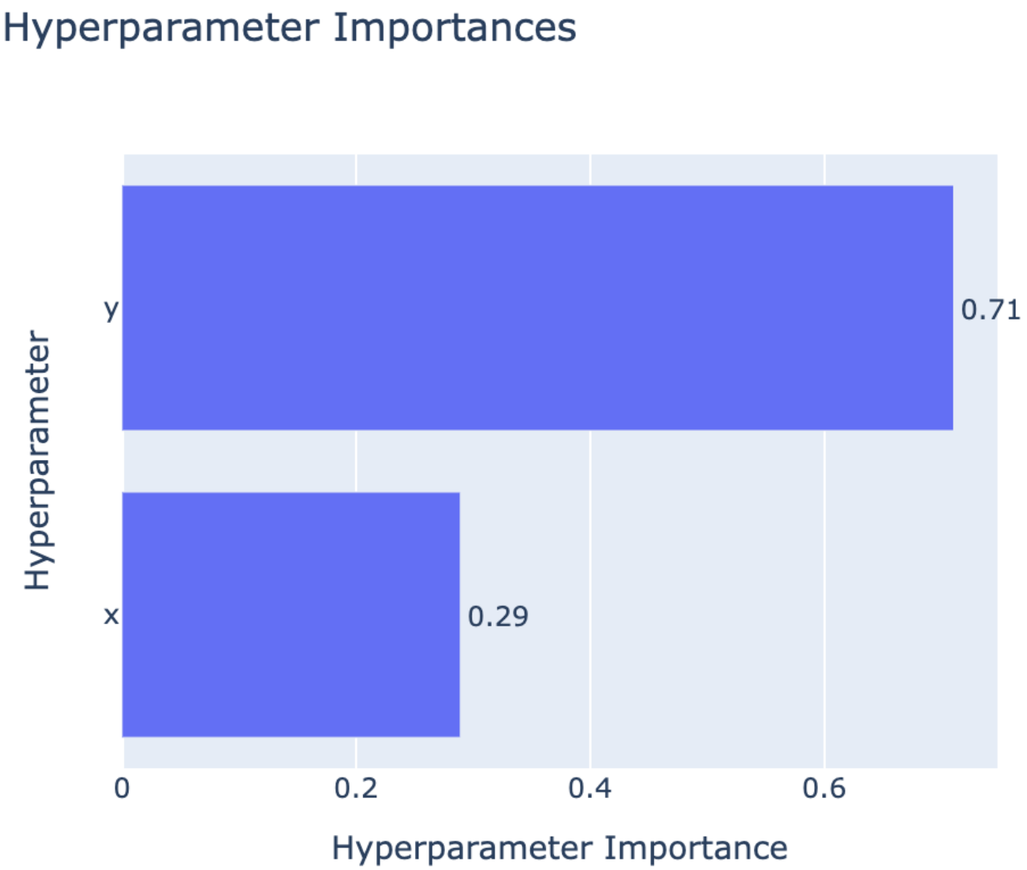
- Use sqlite to save the study
- Sample `n_trials` with default sampler (TPE)
  - Picks hyperparameters at random at first
  - Then focus more on promising regions
- If `n_trials` omitted: run trials until interrupted
- Can load study from database later

```
import sqlite
study = optuna.create_study(
                  storage="sqlite:///DRL.db",
                  study_name="my_study")

study.optimize(objective, n_trials=100)
```

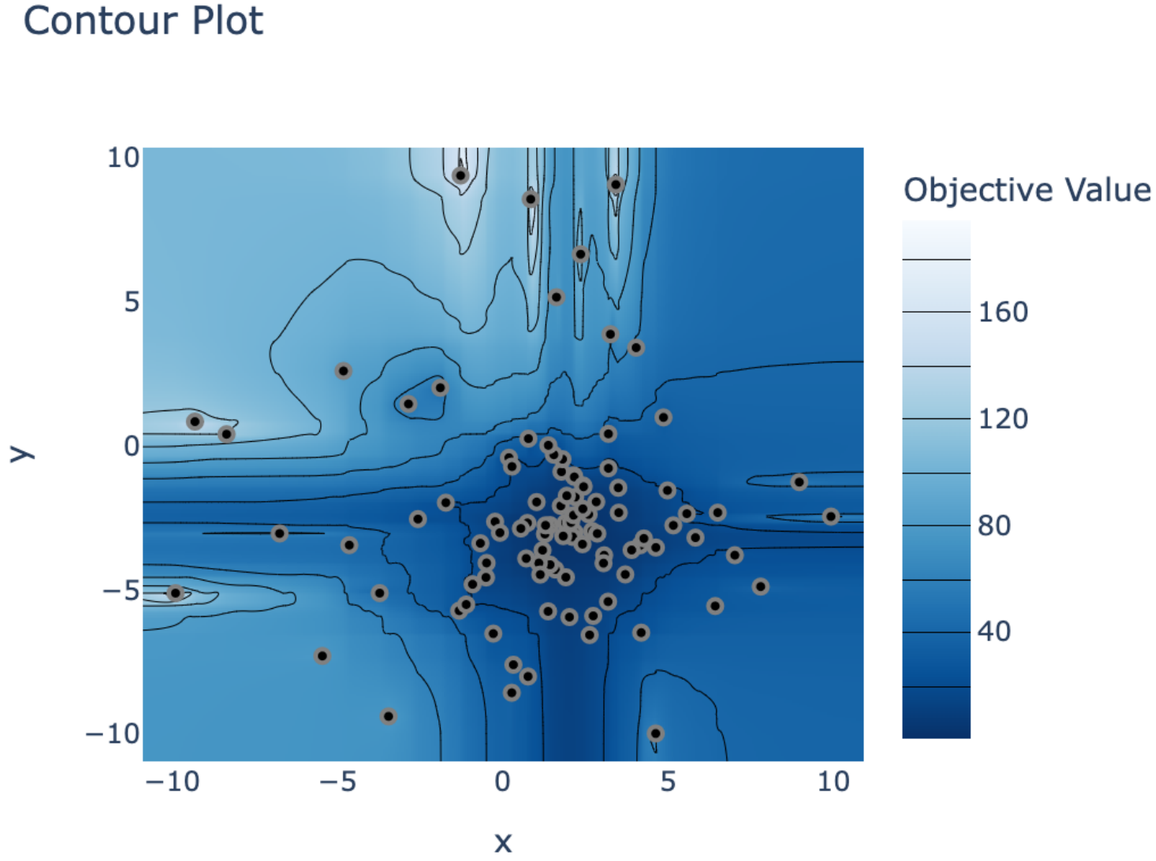
```
loaded_study = optuna.load_study(
                           study_name="my_study",
                           storage="sqlite:///DRL.db")
```

Exploring the study results

```
optuna.visualization.plot_param_importances(study)
```




```
optuna.visualization.plot_contour(study)
```



Let's practice!

DEEP REINFORCEMENT LEARNING IN PYTHON

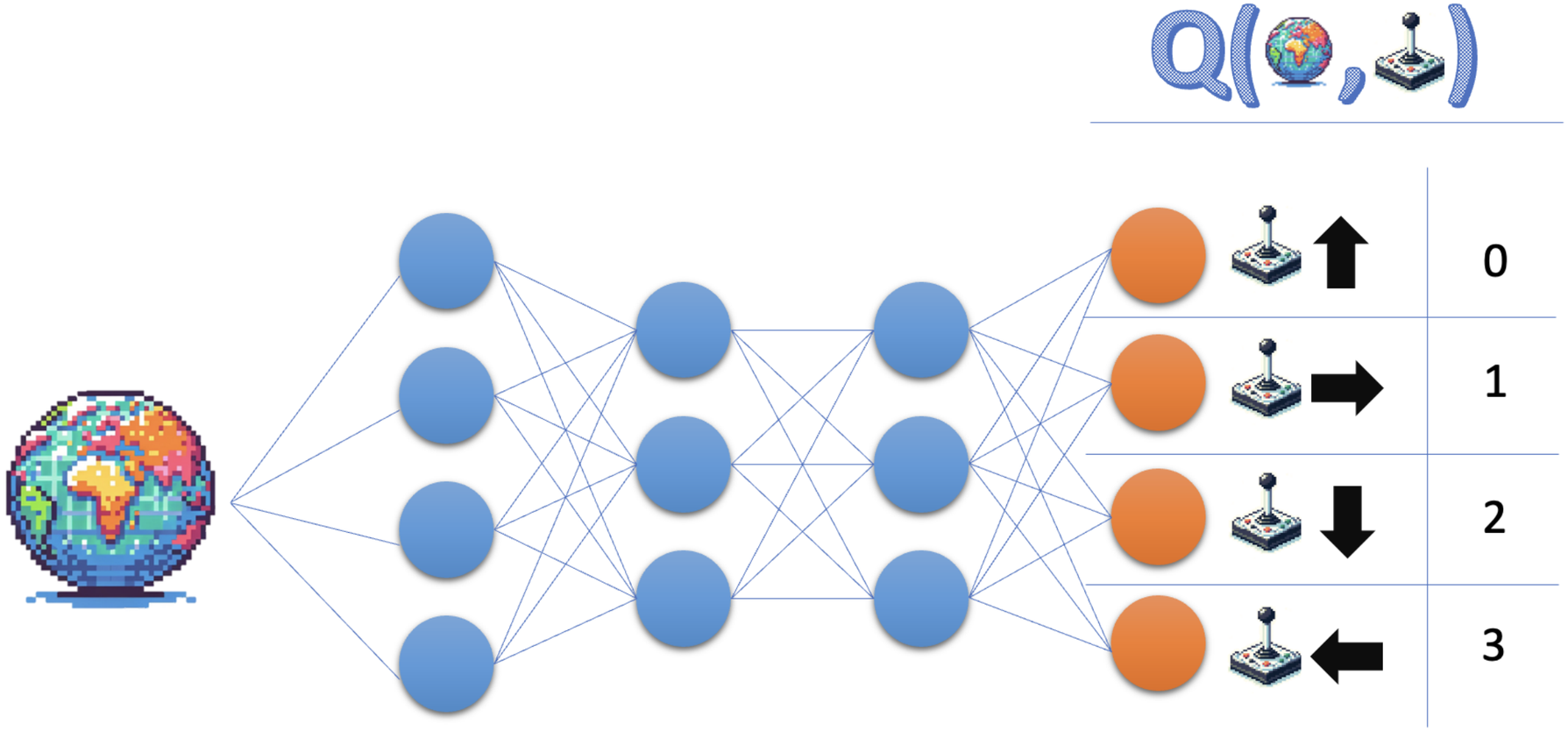
Congratulations!

DEEP REINFORCEMENT LEARNING IN PYTHON

Timothée Carayol

Principal Machine Learning Engineer, Komment

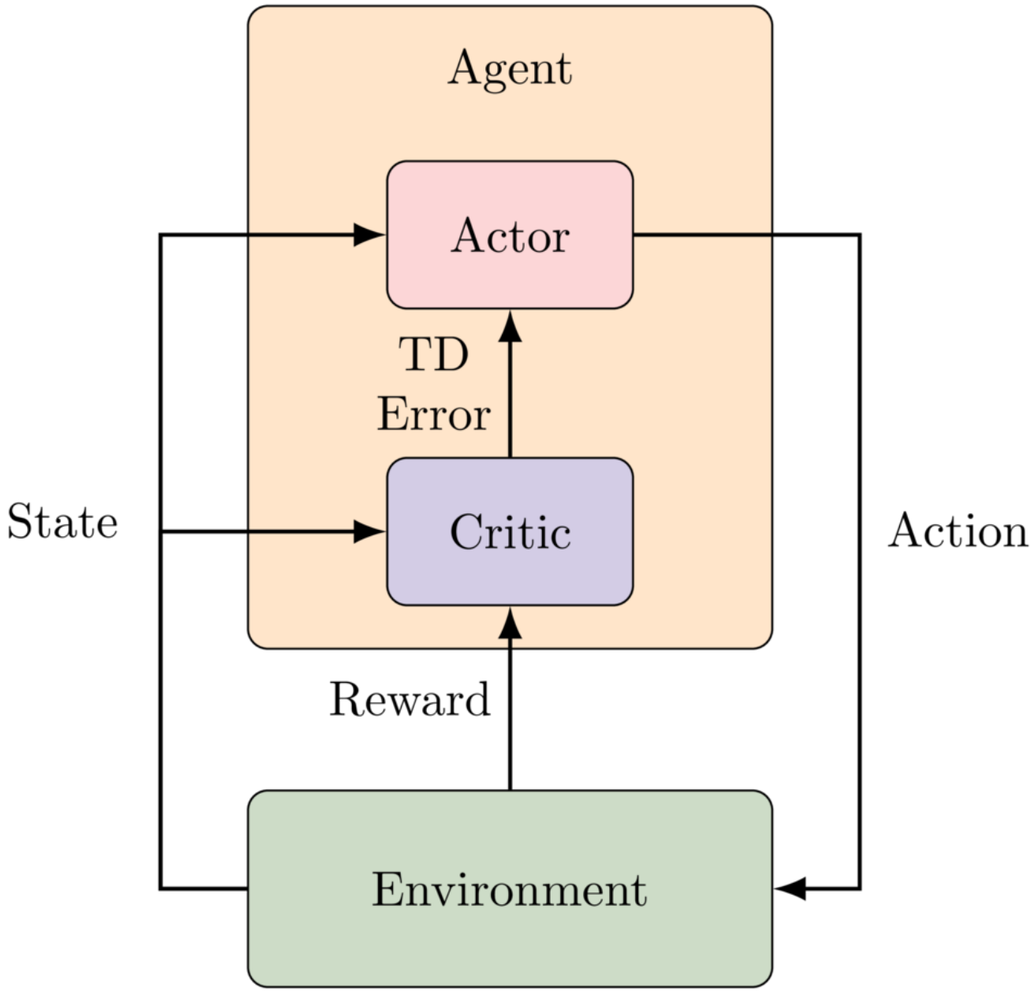
Chapter 1



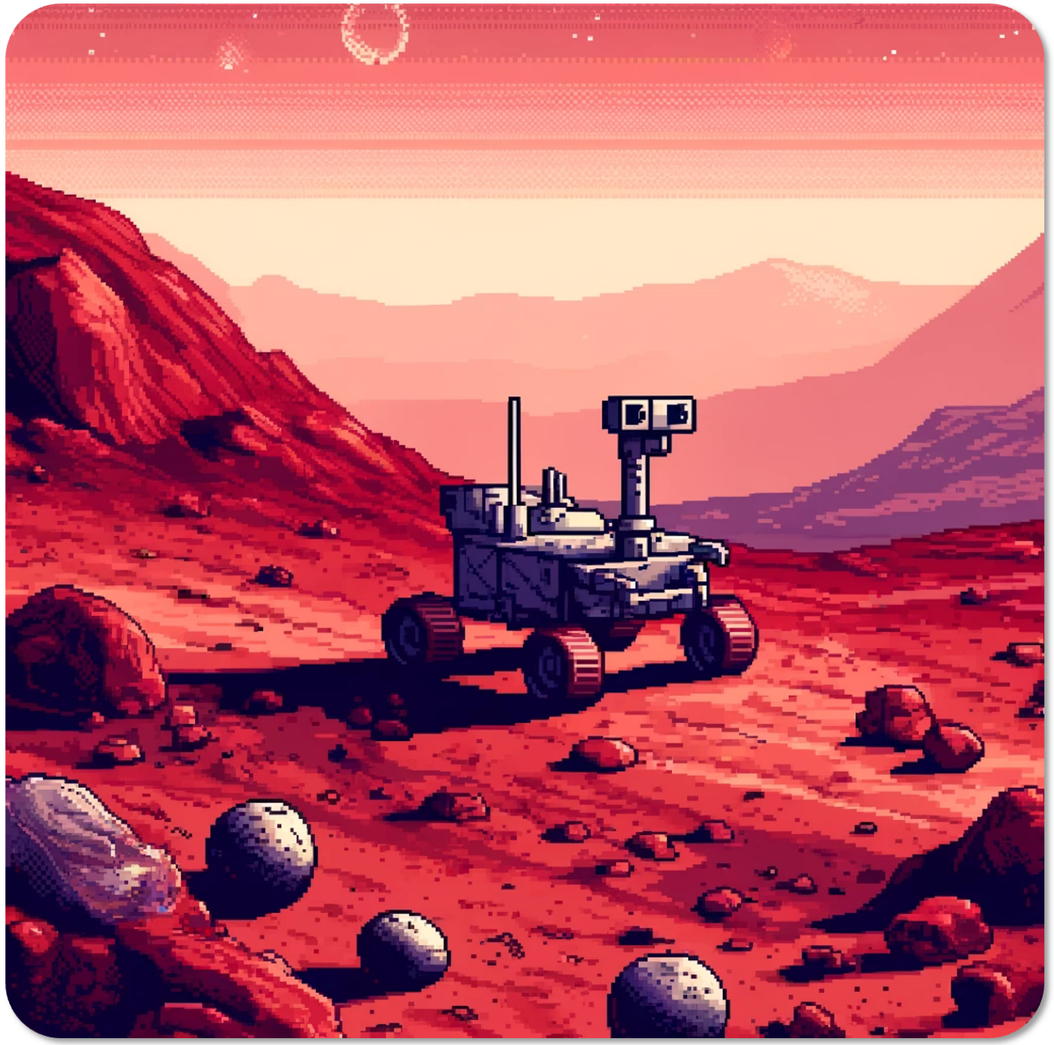
Chapter 2


class ReplayBuffer:
def __init__(self, capacity):
self.memory = deque(maxlen=capacity)

Chapter 3


Agent
Actor
TD
Error
Critic
State
Action
Reward
Environment

Chapter 4



What next?

Well done!

DEEP REINFORCEMENT LEARNING IN PYTHON