1. main()

Learning rate, discount factor, GAE lambda, num of processes, max episode length, num of steps, environment name, optimizer with shared momentum

yes

optimizer

SharedOptimizer()

Shared Momentum?

train()

. . . .

model, optimizer

model, optimizer

model, optimizer

train()

train()

model

ActorCritic()

1. ActorCritic() : \_\_init\_\_()

Initialize weights for actor and critic

Create Layers

model

1. ActorCritic : forward()

Critic o/p

hx-1

cx-1

Dense (256)

hx

Inputs

LSTM

conv2d

32x3x3

conv2d

32x3x3

conv2d

32x3x3

conv2d

32x3x3

Actor o/p

. .

hx

cx

Dense (256)

1. SharedAdam() : \_\_init\_\_()

Store params

lr, betas, eps, weight\_decay

1. SharedAdam(): share\_memory()

Store params (step, exp\_avg, exp\_avg\_sq) in shared memory

1. SharedAdam(): step()

for group in params\_group

for param in group

1. train()

Shared optimizer

Shared model

Create local actor and critic models:

model = ActorCritic()

create\_atari\_env() and initialize with seed

Create local optimizer

Done == False

Copy local model with shared model state and parameters

Forward Pass

For num\_steps

episode\_len += 1

value, logit, (hx,cx) = model.forward()

prob = softmax(logit)

log\_prob = log\_softmax(logit)

entropy = -sum(log\_prob \* prob)

action = multinomial(prob)

state, reward, done = env.step(action)

store state, value, log\_prob, reward

done = done or episode\_len > max\_episode\_len

R, \_, \_ = model.forward()

Store R in values list

For each reward in rewards list

Back propagation

R = gamma\*R + reward[i]

advantage = R – values[i]

value\_loss = value\_loss + 0.5\*advantage^2

delta\_t = rewards[i] + gamma\*values[i+1] – values[i]

delta\_t = rewards[i] + args.gamma\*values[i + 1] - values[i]

gae = gae \* gamma \* gae\_lambda + delta\_t

policy\_loss = policy\_loss - log\_probs[i] \* gae - entropy\_coef \* entropies[i]

Update shared model: ensure\_shared\_grads()

optimizer.step()

Backpropagate using (policy\_loss + args.value\_loss\_coef \* value\_loss)

1. create\_atari\_env()

Create Gym env

Env ID

Normalized env

NormalizedEnv

AtariRescale42x42

1. AtariRescale42x42 : \_\_init()

Create observation space of size 42x42

1. AtariRescale42x42: \_observation()

Processed observation

Input frame/ observation

\_process\_frame42()

1. \_process\_frame42()

processed observation

Input frame/ observation

Rearrange channels, height and width

Normalize the values

Crop to 160x160

Resize to 80x80

Resize to 42x42

1. NormalizedEnv: \_\_init\_\_()

Initialize all the values to default

state\_mean = 0

state\_std = 0

alpha = 0.9999

num\_steps = 0

1. NormalizedEnv: \_observation()

Input frame/ observation

Find running std deviation

state\_std = state\_std\*alpha + obs\_std\*(1-alpha)

Find running mean

state\_mean = state\_mean\*alpha + obs\_mean\*(1-alpha)

unbiased\_std = state\_std/ (1 – alpha^num\_steps)

unbiased\_mean = state\_mean/ (1 – alpha^num\_steps)

Normalized observation = (observation – unbiased\_mean)/ (unbiased\_std + 1e^-8)

Normailzed frame/ observation

1. ensure\_shared\_grads()

Store param grads in shared memory

1. test():

Shared optimizer

Shared model

Create local actor and critic models:

model = ActorCritic()

create\_atari\_env() and initialize with seed

Create local optimizer

Done == False

Copy local model with shared model state and parameters

episode\_len += 1

value, logit, (hx,cx) = model.forward()

prob = softmax(logit)

action = max(prob)

state, reward, done = env.step(action)

reward\_sum += reward

Store action in list

If episode\_len > max\_episode\_len, done =1

If len(actions) > max\_actions, done =1

For each reward in rewards list