PPO Implementation Details

- sample from Gaussian distribution:
  Use Normal(mean, std) instead of MultivariateNormal(mean, cov) or Normal(0,1).
  If sampling from z~Normal(0,1) first then multiplied std, make sure z has the same dimension as std, otherwise all action dimensions depend on the same random variable z, which is not desired in most environments. However, for some special environments like Ant-v2, etc, depending on a single variable may help with the stability of actions in exploration. But in general, just use Normal(mean, std).

  Normal(0,1):
  ```
  normal = Normal(0, 1)
  z      = normal.sample()
  a = mean + std*z
  logprob = Normal(mean, std).log_prob(a)
  logprob = logprob.sum(dim=-1, keepdim=True)  # reduce dim
  ```
  Normal(mean, std):
  ```
  normal = Normal(mean, std)
  a = normal.sample()
  logprob = normal.log_prob(a).sum(-1)
  ```
  MultivariateNormal(mean, cov):
  ```
  cov = torch.diag_embed(var)
  dist = MultivariateNormal(mean, cov)
  a = dist.sample()
  logprob = dist.log_prob(a)
  ```

- log_std depends on state or not:
  in most case, just use
  self.log_std = nn.Parameter(torch.zeros(1, np.prod(env.action_space.shape)))
  instead of letting log_std depending on the state or feature, even if you want to do so, do not let the gradient of log_std backpropagate to every layers in the policy, just take a two-head structure for the policy to output mean and log_std, and the log_std does not BP to the shared feature layers but only its own layers.

- value loss:
  F.smooth_l1_loss vs MSEloss, no big difference

- do not update only with episodic data:
  The data buffer can be the same as DQN (just smaller), samples from different episodes can be put in order into the same buffer.
  The Done signals will handle this properly.

- If using entropy regularization (boosting exploration), be sure to clamp the log_std in a proper range (-20, 2), to avoid entropy (std of Gaussian) blowing up.

- gradient clip:
  nn.utils.clip_grad_norm_(self.parameters, self.max_grad_norm)
  no big effects in general

- learning rate annealing can make some difference

- v_loss_clip: no big difference

- for value function normalization:
  only do advantage normalization, no other value function normalization

- for reward normalization:
  use the gym.wrappers.NormalizeReward(env), dividing the std but not subtracting the mean
  reward normalization can be critical for PPO to work on some envs like Hopper-v2

- shape error:
  This is one of the most common error (hiding in code!), mostly caused by the automatic shape broadcasting.
  Avoid to having (N), (N,1) or (1,N) shapes operated together, this may lead to (N, N), which will not report error but give wrong results!
  Use Assert to ensure the shape of variables operated together to be the same.

- torch.tensor of list is super slow (this triggers a warning):
  do not take torch.tensor(dx) if dx is list, make it array first by torch.tensor(np.array(dx))

- The advantage and target_v_s can be calculated before the epoch updates for the whole batch, so this should be before the epoch update loop, and this should not track any gradients so use with torch.no_grad(): to wrap it.

- minibatch SGD:
  it may speed up training, but no critical difference is made, full batch SGD also learns.