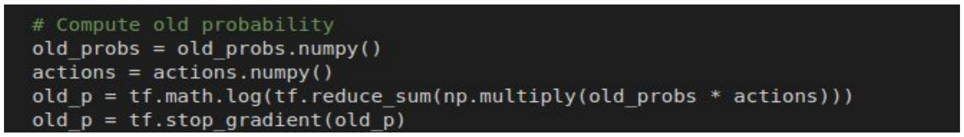
Policy gradient is a form of model free reinforcement learning in which the objective is to maximize the rewards using a parameterized policy. This technique can be found in reinforcement learning algorithms such as actor-to-critic (A2C) and proximal policy optimization (PPO) where a neural network is used to represent the policy. This policy has a set of parameters that will be used in order to train the policy and calculate the maximum rewards. In the case of A2C, the policy selects an action based on a distribution of probabilities that given the agent's current state. Then the parameters of the policy is updated so that an optimal policy can be reached. This will optimize the actions that are taken by the agent.

Policy gradient updates the policy's parameters using gradient descent. Gradient descent algorithms come in a variety of different forms. However, gradient descent in its most rudimentary form can be represented as a linear equation. This linear equation defines the new policy in which the new parameters of the policy are equal to the current parameters of the policy plus its derivative in respect to the parameters multiplied by a coefficient. In the case of PPO and Trust Region Policy Optimization (TRPO), it uses a variation of gradient descent in order to limit the size of the policy update after each full iteration. Because reinforcement algorithms that use A2C require computing the expected value from a set of probabilities, other variations of gradient descent. An example of the implementation is in Figure 1.



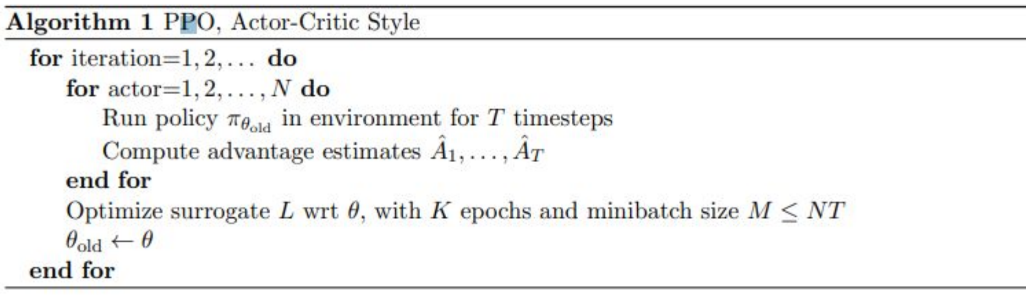
```
# Compute old probability
old_probs = old_probs.numpy()
actions = actions.numpy()
old_p = tf.math.log(tf.reduce_sum(np.multiply(old_probs * actions)))
old_p = tf.stop_gradient(old_p)
```

Figure 1: Compute probabilities via Gradient Descent algorithm

This variation of gradient descent computes the probabilities denoted by old_probs. It then takes the logarithm of the summation of all of those probabilities in order to compute the probability. This is found in A2C, PPO, and TRPO.

A2C algorithms have been integrated into both PPO and TRPO using gradient descent. The crux of A2C involves the implementation of two neural networks denoted as the actor and critic hence its name. The actor network defines the policy updates whereas the critic computes the value estimates given the current state. In the case of PPO and TRPO, the critic network computes value estimates in order to ensure that the action taken by the current policy i.e actor network is good/bad. In the case of TRPO, it goes a step further by computing an advantage through an advantage function which relies on value estimates provided by the critic network. As such, the advantage can also be used to determine if an action is good or bad.

The actor network is also used in order to compute the probabilities before and after a policy update in order to compute a probability ratio. This probability ratio is used in order to compute the loss function. The critic network is used to perform value estimates which can be used as part of the loss function and compute the advantages as is the case with TRPO. Both networks also perform their own iterations which results in a nested for loop. The inner loop handles optimizing the policy so the agent will take an action that better optimizes the rewards. The outer loop attempts to optimize the loss and update the parameters of the policy. An illustration is provided below.



```
Algorithm 1 PPO, Actor-Critic Style
for iteration=1, 2, ... do
    for actor=1, 2, ..., $N$ do
        Run policy $\pi_{\theta_{\text{old}}}$ in environment for $T$ timesteps
        Compute advantage estimates $\hat{A}_1, \ldots, \hat{A}_T$
    end for
    Optimize surrogate $L$ wrt $\theta$, with $K$ epochs and minibatch size $M \leq NT$
    $\theta_{\text{old}} \leftarrow \theta$
end for
```

Figure 2: Implementation of PPO which integrates A2C