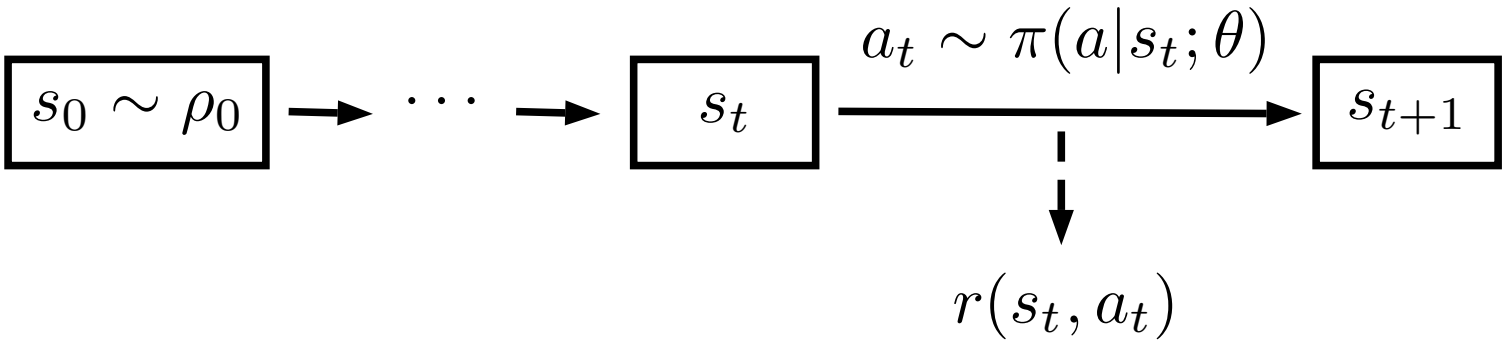
11-695: Competitive Engineering
Implementing REINFORCE

Spring 2018

Reinforcement Learning in a Slide

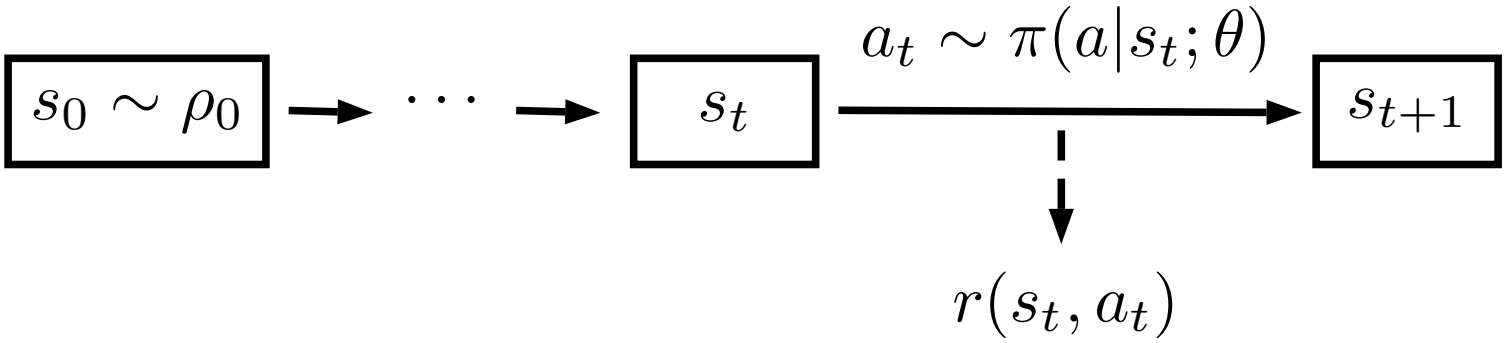
Carnegie Mellon



- Trajectory: $\tau = (s_0, a_0, s_1, a_1, ..., s_T, a_T)$
- Instantaneous reward: $r(s_t, a_t)$
- Discounted reward: $R_{t_0} = \sum_{t=0}^{T} \gamma^t r(s_{t_0+t}, a_{t_0+t})$
- Objective: $\pi^* = \text{argmax}_\pi \, \mathbb{E}_{\tau\sim\pi}[R_0(\tau)]$
- REINFORCE: $\nabla_\theta \mathbb{E}_{\tau\sim\pi}[R_0(\tau)] = \mathbb{E}_{\tau\sim\pi}[R_0(\tau) \cdot \nabla_\theta \log p_\pi(\tau)]$

Coding REINFORCE in `TensorFlow`

Carnegie Mellon



- REINFORCE: $\nabla_\theta \mathbb{E}_{\tau\sim\pi}[R_0(\tau)] = \mathbb{E}_{\tau\sim\pi}[R_0(\tau) \cdot \nabla_\theta \log p_\pi(\tau)]$
- Idea:

$$\nabla_\theta \log p_\pi(\tau) = \sum_{t=0}^{T} \nabla_\theta \log \pi(a_t|s_t;\theta) \tag{1}$$

  - $\log \pi(a_t|s_t;\theta)$ *looks like* cross-entropy
  - Use automatic differentiation to code it!
- Problem: need to collect the trajectories τ's
  - τ does *not* come from a TF computational graph
- Solution: use `tf.placeholder`

General Framework

Carnegie Mellon

tf_rl_training_loop.py

```
1  def train():
2    env = create_environment()
3    g = tf.Graph()
4    with g.as_default():
5      ops = build_tf_graph()
6      with tf.Session() as sess:
7        for eps in range(n_eps):
8          states, actions, rewards = [], [], []
9          state = env.reset()
10         for step in range(200):
11           states.append(state)
12           action = sess.run(ops["actions"], feed_dict={ops["states"]: state})
13           state, reward, done, _ = env.step(action)
14           actions.append(action)
15           rewards.append(reward)
16           if done: break
17         states = np.concatenate(states, axis=0)
18         feed_dict = {ops["states"]: states, ops["rewards"]: rewards,
19                      ops["actions"]: actions}
20         sess.run(ops["train_op"], feed_dict=feed_dict)
```

TensorFlow Graph for RL

Carnegie Mellon

tf_rl_graph.py

```python
def build_tf_graph(inp_shape):
  states = tf.placeholder(tf.float32, [None] + inp_shape, name="states")
  rewards = tf.placeholder(tf.float32, [None], name="rewards")

  # just a feed-forward neural network
  with tf.variable_scope("policy_net"):
    w_hidden = tf.get_variable("w_hidden", [inp_shape[-1], n_hidden])
    w_soft = tf.get_variable( "w_soft", [n_hidden, n_actions])
  hidden = tf.nn.relu(tf.matmul(states, w_hidden))
  logits = tf.matmul(hidden, w_soft)

  # sample actions
  actions = tf.multinomial(logits, num_samples=1)
  actions = tf.to_int32(actions)
  actions = tf.reshape(actions, [-1])

  # log_probs, loss, train_op: next slide
```

- Just write a normal neural network to compute `logits`
- Sample `actions` from your `logits` using `tf.multinomial`

TensorFlow Graph for RL

Carnegie Mellon

tf_rl_graph.py

```python
def build_tf_graph(inp_shape):
  # states, rewards, actions, policy network, etc.: previous slide

  log_probs = tf.nn.sparse_softmax_cross_entropy_with_logits(
    logits=logits, labels=actions)
  loss = tf.reduce_sum(rewards * log_probs)

  tf_vars = [var for var in tf.trainable_variables()
             if var.name.startswith("policy_net")]
  optimizer = tf.train.AdamOptimizer(learning_rate=hparams.lr)
  train_op = optimizer.minimize(
    loss, global_step=global_step, var_list=tf_vars)

  ops = {"states": states, "rewards": rewards, "actions": actions, ...}
  return ops
```

$$\nabla_\theta \mathbb{E}_{\tau\sim\pi}[R_0(\tau)] = \mathbb{E}_{\tau\sim\pi}[R_0(\tau)\cdot\nabla_\theta \log p_\pi(\tau)] \tag{2}$$

$$= \mathbb{E}_{\tau\sim\pi}\left[R_0(\tau)\cdot\sum_{t=0}^{T}\nabla_\theta \log \pi(a_t|s_t;\theta)\right] \tag{3}$$

TensorFlow Graph for RL

Carnegie Mellon

tf_rl_graph.py

```
1  def build_tf_graph(inp_shape):
2    # states, rewards, actions, policy network, etc.: previous slide
3
4    log_probs = tf.nn.sparse_softmax_cross_entropy_with_logits(
5      logits=logits, labels=actions)
6    loss = tf.reduce_sum(rewards * log_probs)
7
8    tf_vars = [var for var in tf.trainable_variables()
9               if var.name.startswith("policy_net")]
10   optimizer = tf.train.AdamOptimizer(learning_rate=hparams.lr)
11   train_op = optimizer.minimize(
12     loss, global_step=global_step, var_list=tf_vars)
13
14   ops = {"states": states, "rewards": rewards, "actions": actions, ...}
15   return ops
```

- Monte Carlo sampling with *one* example

$$\nabla_\theta \mathbb{E}_{\tau\sim\pi}[R_0(\tau)] \approx \hat{\mathbb{E}}_{\tau\sim\pi}\left[R_0(\tau)\cdot\sum_{t=0}^{T}\nabla_\theta \log\pi(a_t|s_t;\theta)\right] \tag{5}$$

REINFORCE with Baseline?

Carnegie Mellon

tf_rl_baseline.py

```
1  def build_tf_graph(inp_shape):
2    # states, rewards, log_probs, etc.
3    baseline = tf.placeholder(tf.float32, None, name="baseline")
4    loss = tf.reduce_sum((rewards - baseline) * log_probs)
5
6  def train():
7    # Create env, graph, ops, session, etc.
8    baseline = 0
9    with tf.Session() as sess:
10     for eps in range(n_eps):
11       # collect states, rewards, actions, etc.
12       baseline = 0.9 * baseline + 0.1 * rewards  # Python code, no TF!
13       feed_dict = {ops["states"]: states, ops["rewards"]: rewards,
14                    ops["actions"]: actions, ops["baseline"]: baseline}
15       sess.run(ops["train_op"], feed_dict=feed_dict)
```

- Baseline

$$\nabla_\theta \mathbb{E}_{\tau\sim\pi}[R_0(\tau)] \approx \hat{\mathbb{E}}_{\tau\sim\pi}\left[(R_0(\tau) - b)\cdot\sum_{t=0}^{T}\nabla_\theta \log\pi(a_t|s_t;\theta)\right] \tag{6}$$

- No problem! At least if you use the moving average baseline

REINFORCE with Temporal Structure?

Carnegie Mellon

tf_rl_baseline.py

```
1  def build_tf_graph(inp_shape):
2    # just focus on this line!
3    loss = tf.reduce_sum(rewards * log_probs)
4
5  def train():
6    # Create env, graph, ops, session, etc.
7    with tf.Session() as sess:
8      for eps in range(n_eps):
9        # collect states, rewards, actions, etc.
10       discount = np.pow(discount, np.arange(0, np.size(rewards)))
11       rewards = np.array(rewards, dtype=np.float32)
12       rewards *=  discount
13       rewards = np.cumsum(rewards[::-1])[::-1] / discount
14       sess.run(ops["train_op"], feed_dict=feed_dict)
```

- Temporal Structure is another form of REINFORCE:

$$\nabla_\theta \mathbb{E}_{\tau\sim\pi}[R_0(\tau)] \approx \hat{\mathbb{E}}_{\tau\sim\pi}\left[\sum_{t=0}^{T} \gamma^t R_t(\tau) \nabla_\theta \log \pi(a_t|s_t;\theta)\right] \tag{7}$$

REINFORCE with Temporal Structure + Baseline?

Carnegie Mellon

tf_rl_baseline.py

```
1  def build_tf_graph(inp_shape):
2    # just focus on this line!
3    loss = tf.reduce_sum((rewards - baseline) * log_probs)
4
5  def train():
6    # Create env, graph, ops, session, etc.
7    with tf.Session() as sess:
8      baseline = 0
9      for eps in range(n_eps):
10       # collect states, rewards, actions, etc.
11       discount = np.pow(discount, np.arange(0, np.size(rewards)))
12       rewards = np.array(rewards, dtype=np.float32)
13       rewards *=  discount
14       rewards = np.cumsum(rewards[::-1])[::-1] / discount - baseline
15       sess.run(ops["train_op"], feed_dict=feed_dict)
```

- Temporal Structure is another form of REINFORCE:

$$\nabla_\theta \mathbb{E}_{\tau\sim\pi}[R_0(\tau)] \approx \hat{\mathbb{E}}_{\tau\sim\pi}\left[\sum_{t=0}^{T} \gamma^t (R_t(\tau) - b) \nabla_\theta \log \pi(a_t|s_t;\theta)\right] \tag{8}$$

REINFORCE with Neural Baseline?

Carnegie Mellon

tf_rl_neural_baseline.py

```
1  def build_tf_graph(inp_shape):
2    # just a feed-forward neural network
3    with tf.variable_scope("baseline_net"):
4      bl_w_hidden = tf.get_variable("w_hidden", [inp_shape[-1], n_hidden])
5      bl_w_pred = tf.get_variable("w_pred", [n_hidden, n_actions])
6    hidden = tf.nn.relu(tf.matmul(states, bl_w_hidden))
7    baseline = tf.matmul(hidden, bl_w_pred)
8
9    rl_loss = tf.reduce_sum((rewards - baseline) * log_probs)
10   bl_loss = tf.reduce_sum(tf.pow(baseline - rewards, 2.0))
11   bl_train_op = optimizer.minimize(bl_loss, [bl_w_hidden, bl_w_pred])
12
13 def train():
14   # Create env, graph, ops, session, etc.
15   with tf.Session() as sess:
16     baseline = 0
17     for eps in range(n_eps):
18       # collect states, rewards, actions, etc.
19       sess.run([ops["train_op"], ops["bl_train_op"]], feed_dict=feed_dict)
```

- Just train your baseline network together
- This method is *roughly* known as *Actor-Critic*

Shixiang Gu, Timothy P. Lillicrap, Zoubin Ghahramani, Richard E. Turner, and Sergey Levine. Q-prop: Sample-efficient policy gradient with an off-policy critic. In *ICLR*, 2017.

Volodymyr Mnih, Adri Puigdomnech Badia, Mehdi Mirza, Alex Graves, Timothy P. Lillicrap, Tim Harley, David Silver, and Koray Kavukcuoglu. Asynchronous methods for deep reinforcement learning. In *ICML*, 2016.

John Schulman, Sergey Levine, Philipp Moritz, Michael Jordan I., and Pieter Abbeel. Trust region policy optimization. In *ICML*, 2015.

Richard S. Sutton, David A. McAllester, Satinder P. Singh, and Yishay Mansour. Policy gradient methods for reinforcement learning with function approximation. In *NIPS*, 1999.

Ronald J. Williams. Simple statistical gradient-following algorithms for connectionist reinforcement learning. *Machine Learning*, 1992.