universidade de aveiro

TENSORFLOW AGENTS

Daniel Emídio (108986)
David Palricas (108780)
Márcio Tavares (108096)

Professor
Luís Seabra Lopes
(lsl@ua.pt)

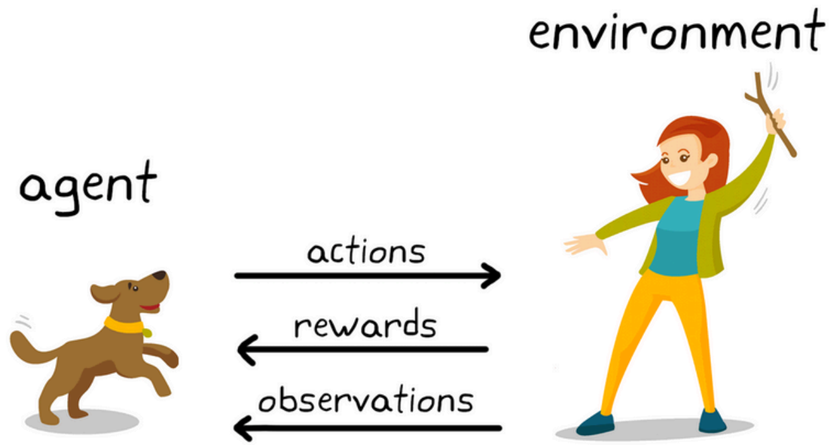
INTRODUCTION

Reinforcement Learning with TF-Agents

Language:

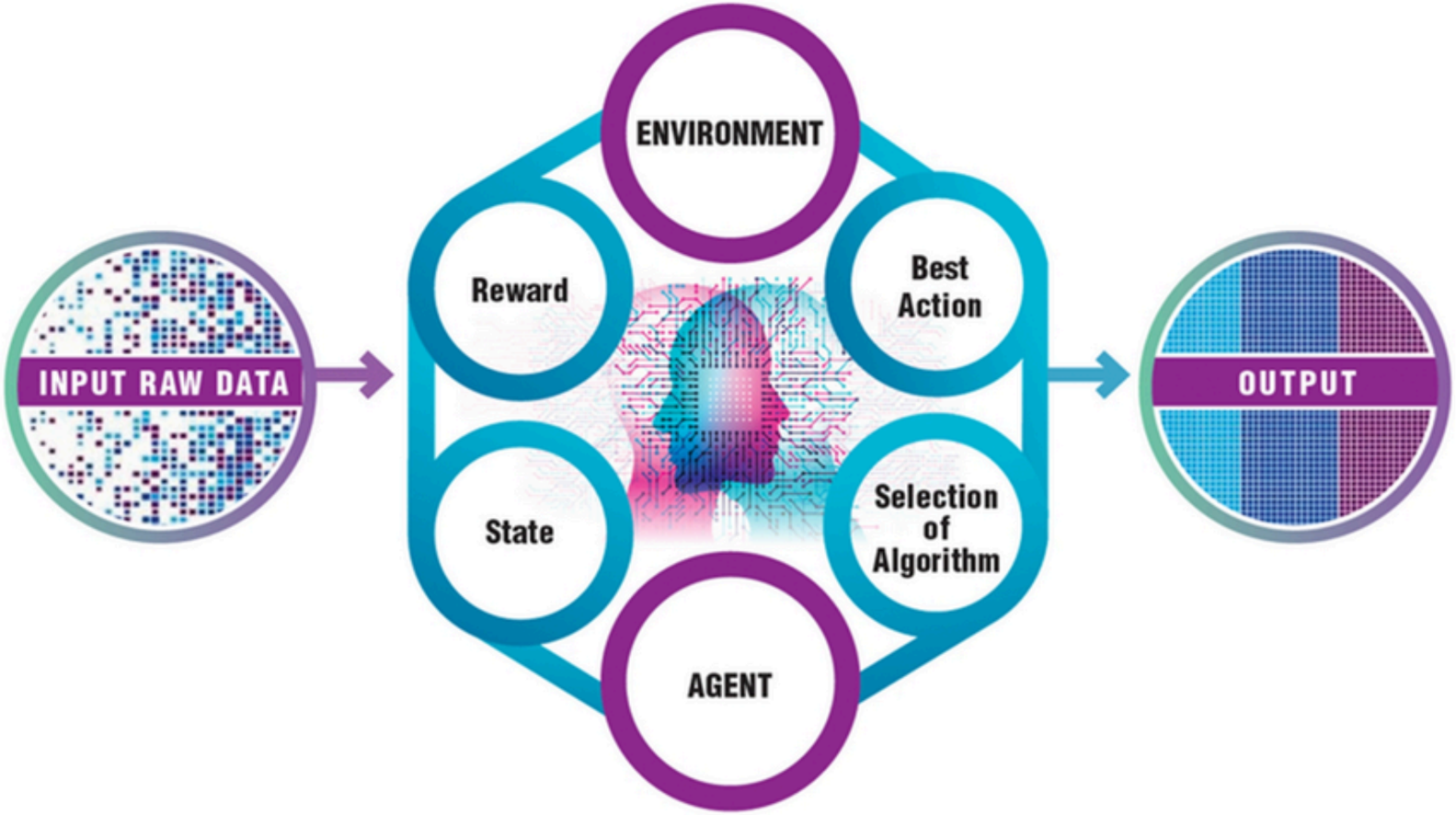
python™

Reinforcement Learning:



BASE PRINCIPLES

1. **Agent**
2. **Environment**
3. **State**
4. **Action**
5. **Reward**
6. **Policy**



USE CASES

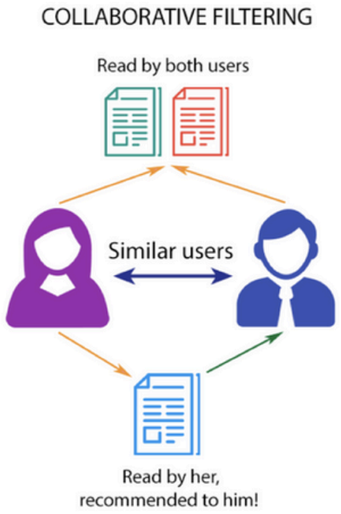
Used By:

Google

Uber

AMERICAN EXPRESS

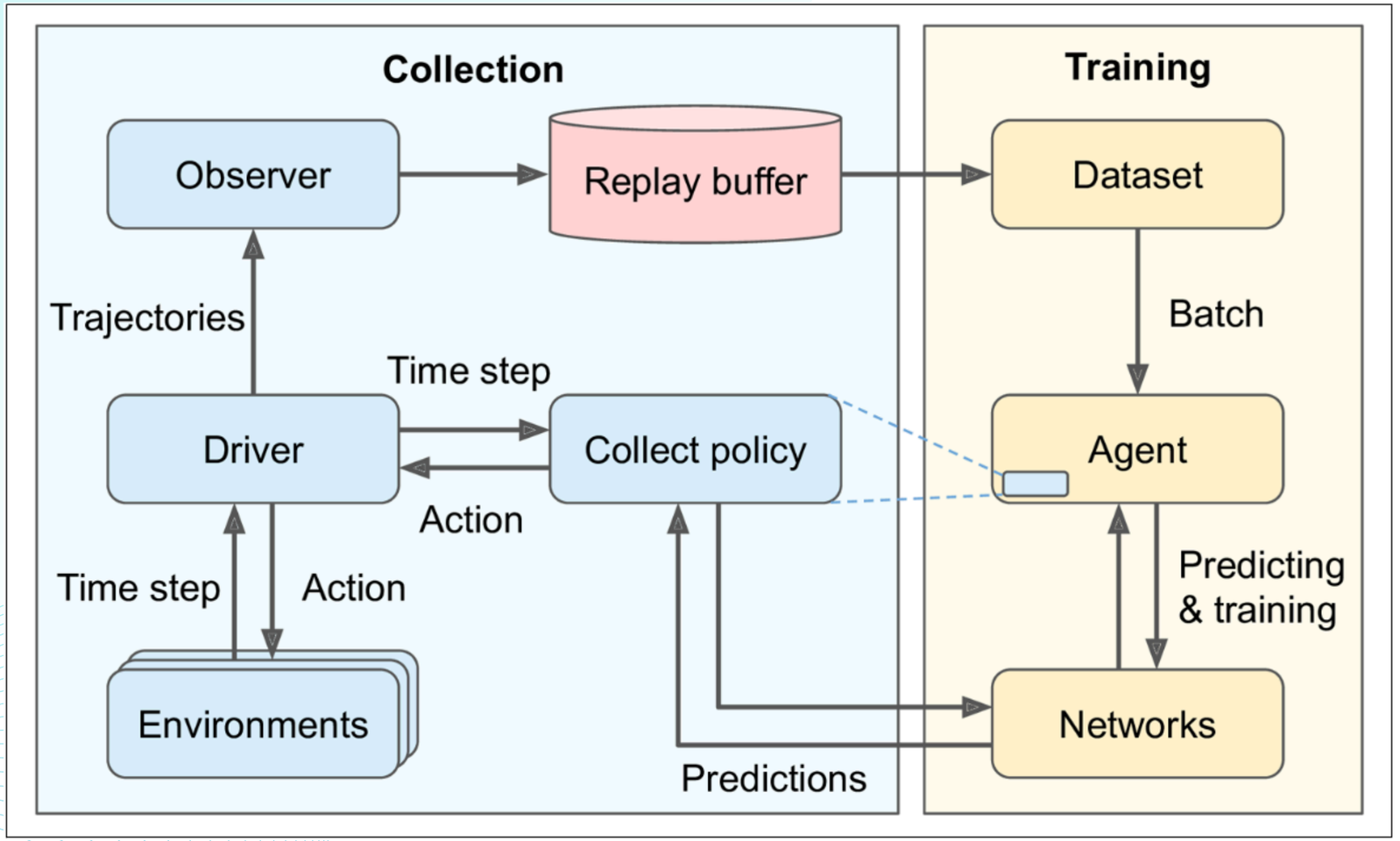
Recommendation Systeams



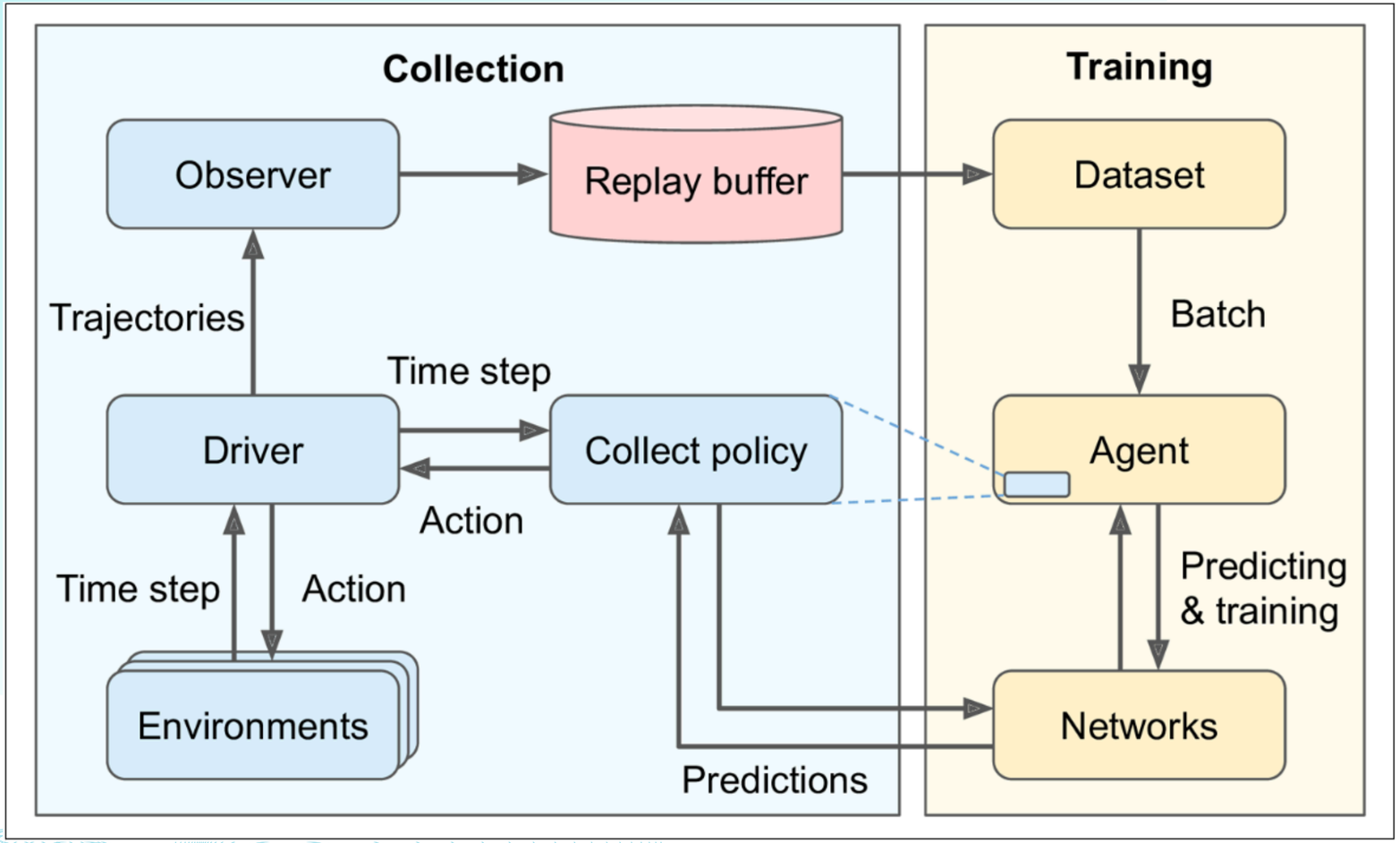
Medical Diagnostics

Image Recognition

ARQUITECTURE



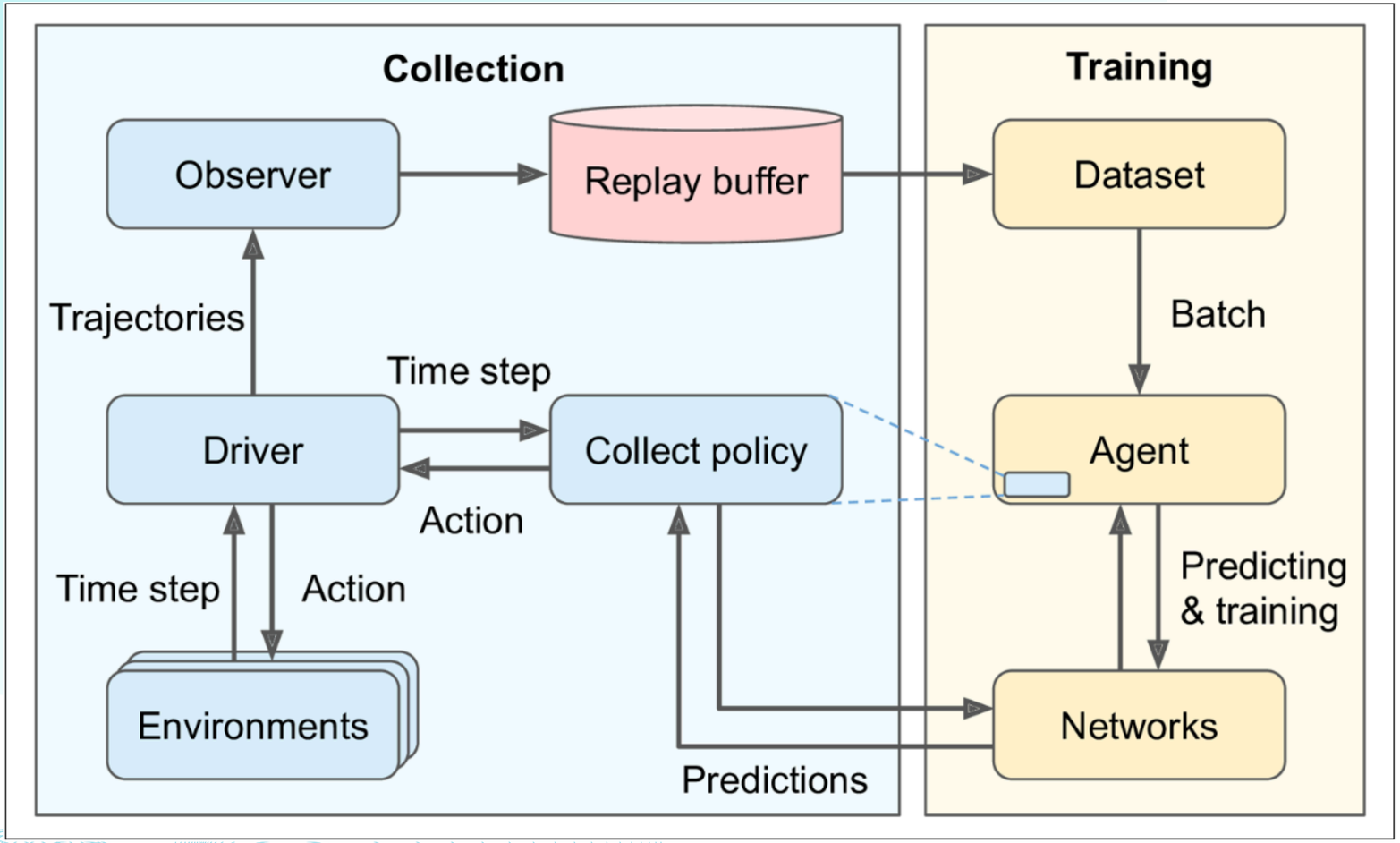
ARQUITECTURE



Collection

- **Environments:**
  - Provides time step after each action;
- **Driver:**
  - Executes actions and receives feedback;
- **Collection Policy:**
  - Used policy during data collection;
- **Observer:**
  - Collects Trajectories;
- **Replay Buffer:**
  - Stores Trajectories.

ARQUITECTURE



Training

- **Dataset**
  - Generates Batches;
- **Agent:**
  - Uses collected data to adjust the policy and networks;
  - Supports several RL algorithms;
- **Networks:**
  - Represent the models used for predictions.

ENVIRONMENTS

- **Observation:** This is the part of the environment state that the agent can observe to choose its actions at the next step.

- **Reward:** The agent is learning to maximize the sum of these rewards across multiple steps

- **step_type**: The first, intermediate or last time step in a sequence

- **discount:** how much to weight the reward at the next time step relative to the reward at the current time step

-

```python
import abc

class PyEnvironment(object):

  def reset(self):
    """Return initial_time_step."""
    self._current_time_step = self._reset()
    return self._current_time_step

  def step(self, action):
    """Apply action and return new time_step."""
    if self._current_time_step is None:
        return self.reset()
    self._current_time_step = self._step(action)
    return self._current_time_step

  def current_time_step(self):
    return self._current_time_step

  def time_step_spec(self):
    """Return time_step_spec."""

  @abc.abstractmethod
  def observation_spec(self):
    """Return observation_spec."""

  @abc.abstractmethod
  def action_spec(self):
    """Return action_spec."""

  @abc.abstractmethod
  def _reset(self):
    """Return initial_time_step."""

  @abc.abstractmethod
  def _step(self, action):
    """Apply action and return new time_step."""
```

NEURAL NETWORKS

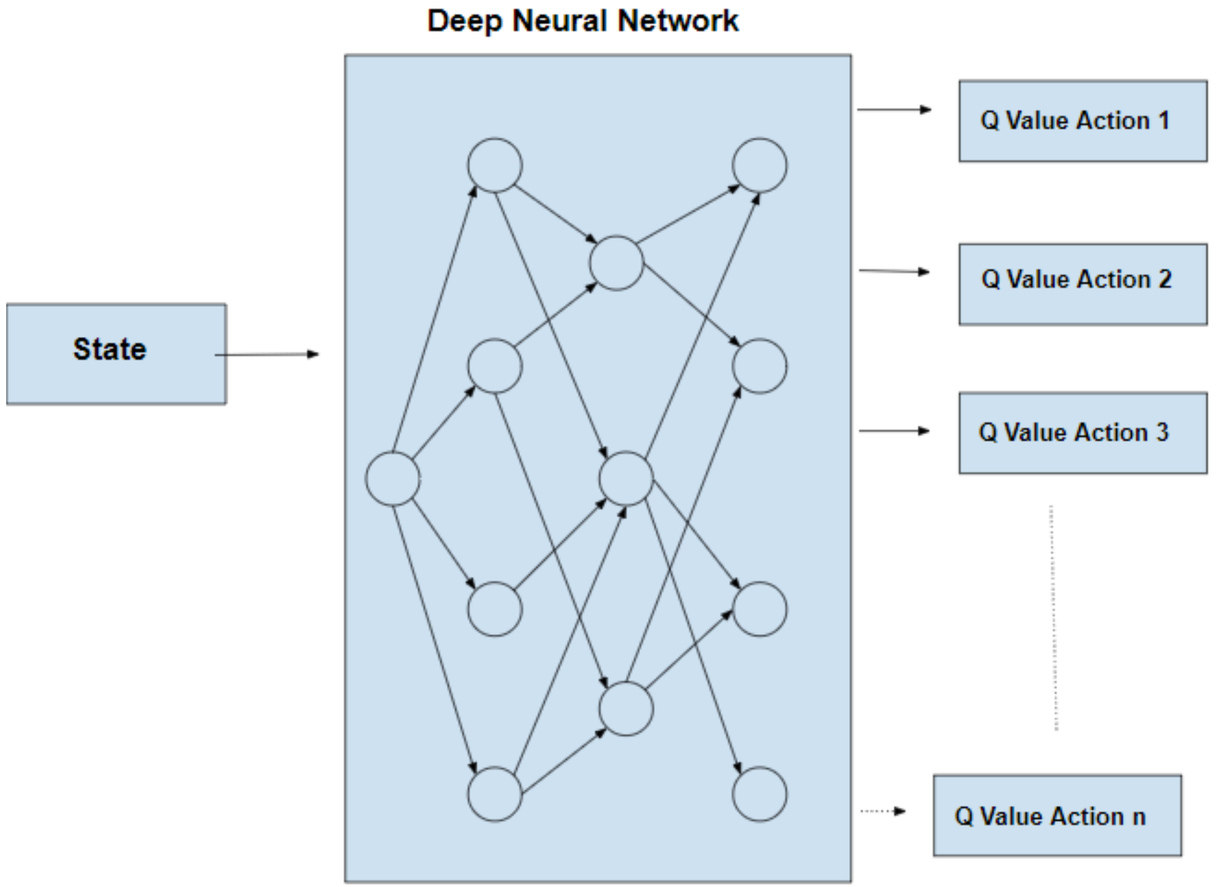
Tensor FLow Main Networks : QNetwork, CriticNetworks, ActorNetworks, ActorDistributionNetworks

discount factor = 0.90

$$Q(s,a) = r + \gamma max_{a'} Q(s',a')$$

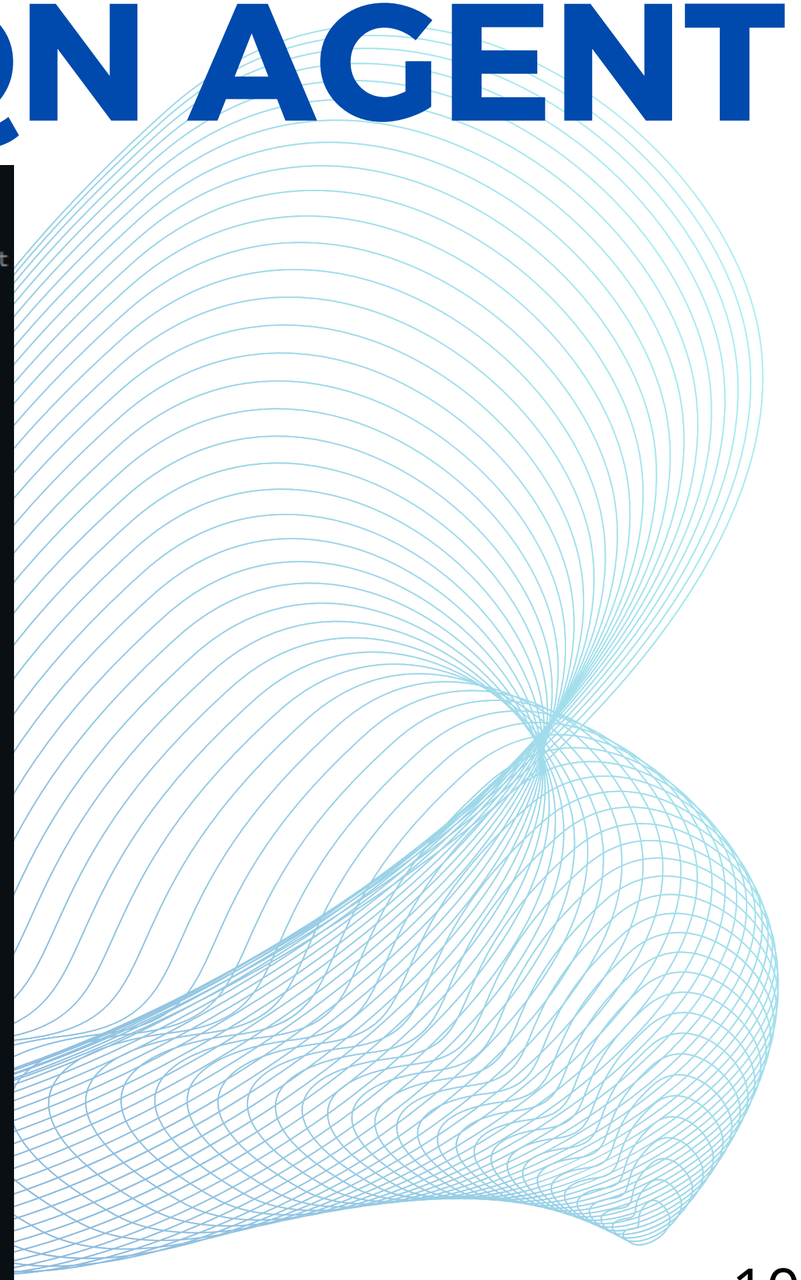
immediate reward

future reward



SETTING UP THE DQN AGENT

```
# 1. Configure the environment
env_name = "CartPole-v1"  # Environment name
train_env = TFPyEnvironment(suite_gym.load(env_name))  # Training environment
eval_env = suite_gym.load(env_name)  # Evaluation environment (non-TF)

# 2. Create the Q network and agent
fc_layer_params = (100,)  # Neural network parameters

q_net = QNetwork(
    train_env.observation_spec(),
    train_env.action_spec(),
    fc_layer_params=fc_layer_params
)

optimizer = tf.keras.optimizers.Adam(learning_rate=1e-3)

agent = DqnAgent(
    train_env.time_step_spec(),
    train_env.action_spec(),
    q_network=q_net,
    optimizer=optimizer,
    td_errors_loss_fn=tf.keras.losses.Huber(reduction="none"),
    train_step_counter=tf.Variable(0)
)
agent.initialize()

# 3. Configure the replay buffer and data collection
replay_buffer = TFUniformReplayBuffer(
    data_spec=agent.collect_data_spec,
    batch_size=train_env.batch_size,
    max_length=10000
)
```



METRICS

Plot Graphs:


```
# Define the metrics + TensorBoard writer
train_metrics = [
    tf_metrics.NumberOfEpisodes(),
    tf_metrics.EnvironmentSteps(),
    tf_metrics.AverageReturnMetric(),
    tf_metrics.AverageEpisodeLengthMetric(),
]
train_summary_writer = tf.summary.create_file_writer(log_dir)
train_summary_writer.set_as_default()
```

DEMO

THANKS FOR LISTENING

Do you have any question?