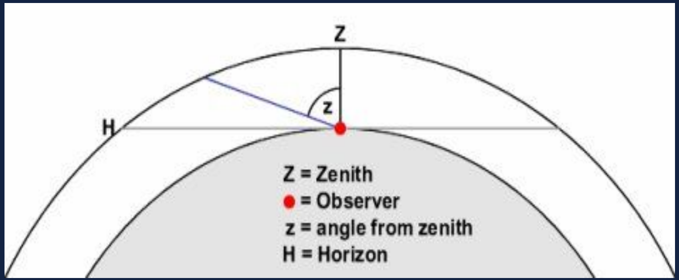
Reinforcement Learning in Telescope Scheduling

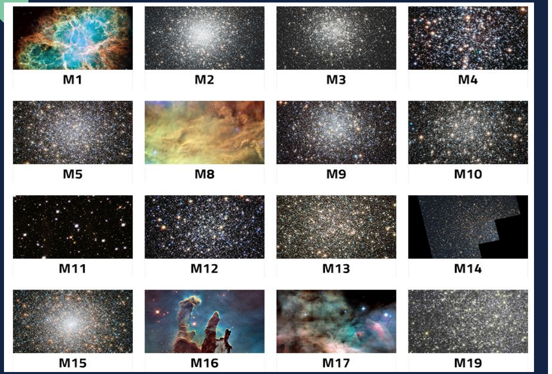
Nicholas Ermolov, Anne Foley

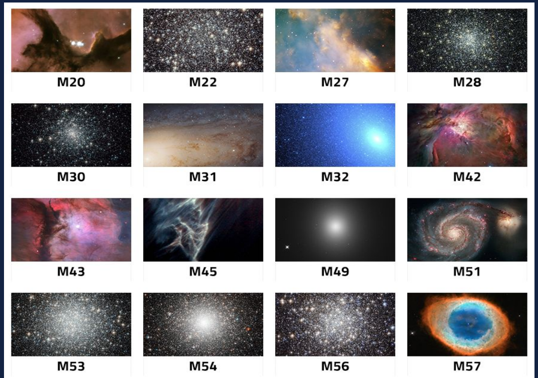
Observing the Night Sky

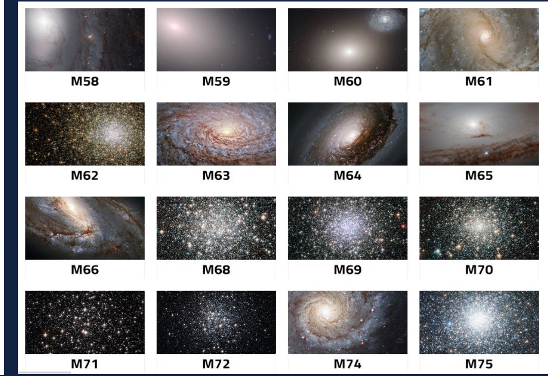
- Logical approach:
  - Obtain high-quality images/data
  - Observe multiple objects
  - Applies even in recreational observations
- Often requires scheduling for efficiency and optimization

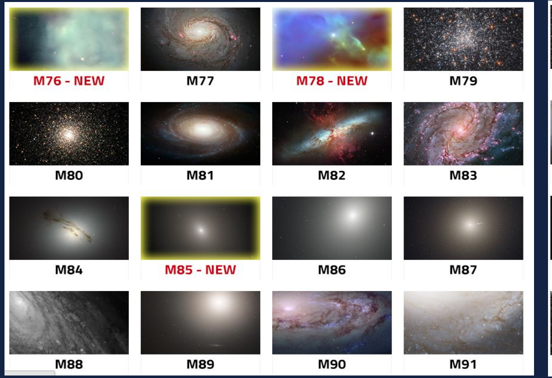


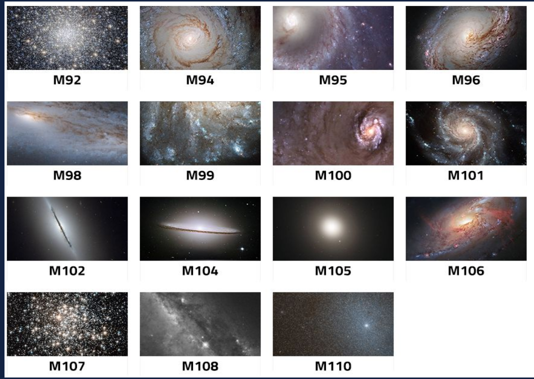
“Messier Marathon”









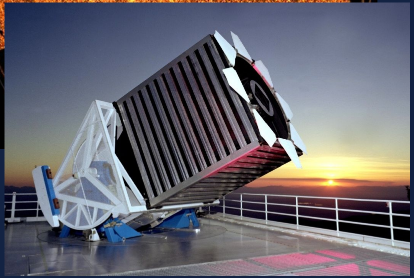


From NASA’s Hubble Telescope

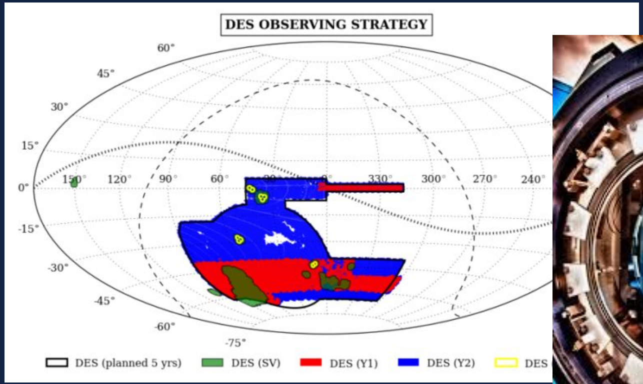
Robotic Telescopes

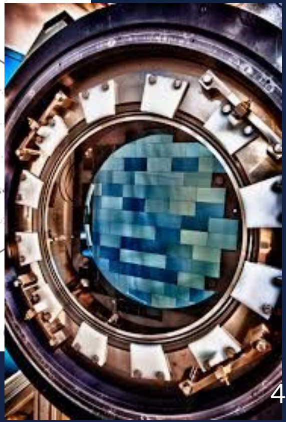
- Sloan Digital Sky Survey (SDSS)
    - Over 300 million objects
    - Manual planning

- Dark Energy Survey (DES)
    - 16,000 pointings, image every 2 minutes
    - Hand-tuned computerized simulations



SDSS telescope (right)
and sky coverage (up)





Dark Energy Camera (right)
and DES ‘footprint’ (up)

Sky Surveying vs. Classic Games

- Surveying the sky: like navigating a 2D plane
- Can be compared to classic video games
- How have computers learned to play games?
    - Reinforcement learning and neural networks

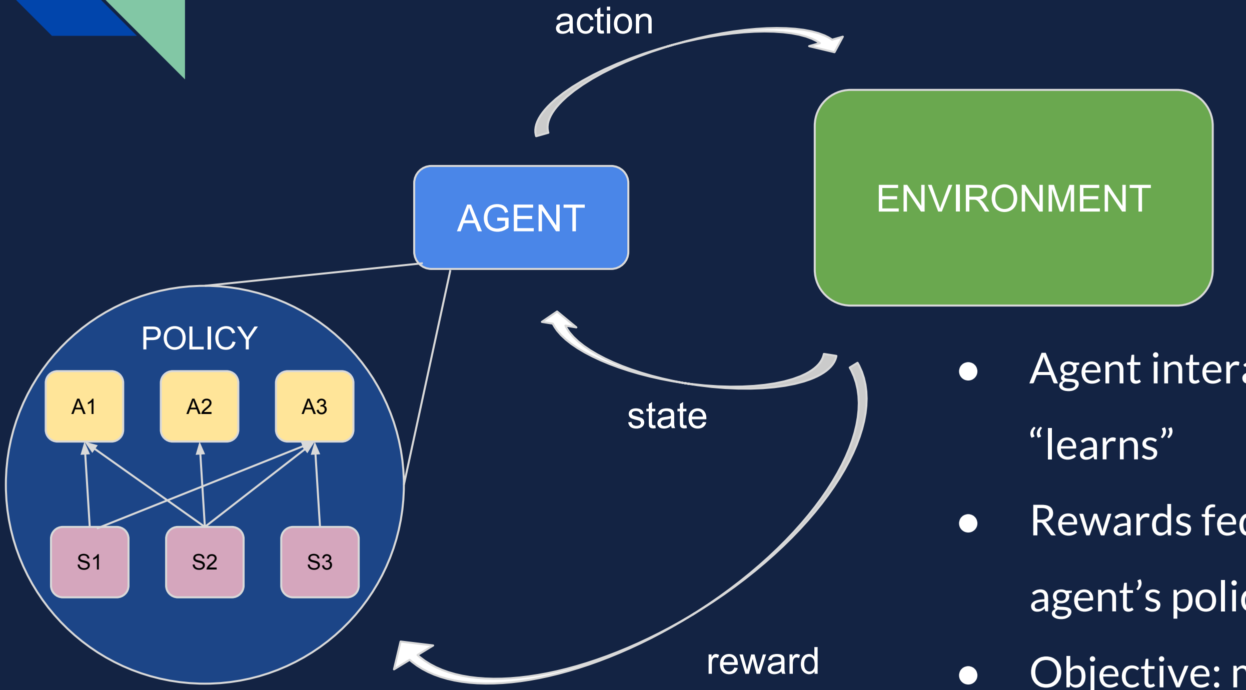
Neural net using reinforcement learning to play *Doom*

Our Project

- Understand the principles of reinforcement learning
- Apply the concepts of reinforcement learning to schedule optimization
- Represent telescope-sky interactions in the context of machine learning
- Implement software that can learn optimal policies on modeled data

Reinforcement Learning

Reinforcement Learning (RL) Concepts

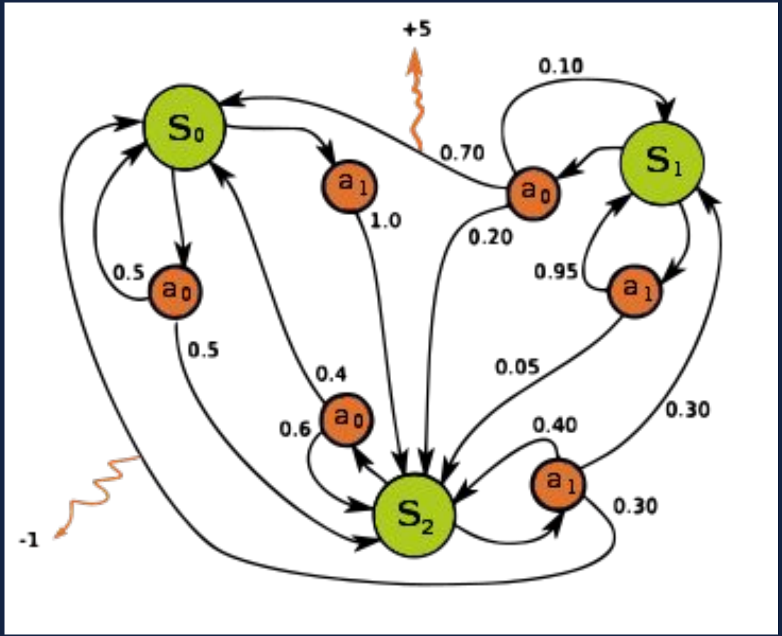


- Agent interacts with its environment and “learns”
- Rewards fed back from environment affect agent’s policy
- Objective: maximise cumulative reward
- Example: Chess

Applications of RL

Why RL is Useful for our Project

- RL is powerful in learning Markov Decision Processes
  - Learning best actions between different states
  - Works for stochastic or deterministic
- Our environment can be modeled as an episodic MDP



Bellman Equations

State value representation:

$$V^{\pi}(s) = \mathbb{E}_{\pi}\left[R_t | s_t = s\right] \Rightarrow V^{\pi}(s) = \sum_{a} \pi(s,a) \sum_{s'} \mathcal{P}^{a}_{ss'} \left[\mathcal{R}^{a}_{ss'} + \gamma V^{\pi}(s')\right]$$

State-action pair value:

$$Q^{\pi}(s,a) = \mathbb{E}_{\pi}\left[R_t | s_t = s, a_t = a\right] \Rightarrow Q^{\pi}(s,a) = \sum_{s'} \mathcal{P}^{a}_{ss'} \left[\mathcal{R}^{a}_{ss'} + \gamma \sum_{a'} Q^{\pi}(s',a')\right]$$

Optimal policy:

$$\pi^{*} = \arg\max_{\pi} \mathbb{E}\left[\sum_{t \geq 0} \gamma^{t} r_t | \pi\right]$$

Using a statistical representation to navigate a MDP

From Equations to Code

- Want to estimate the values of states
  - Used as a baseline for moving forward in our code
- Use ‘advantage’ to estimate relative values of actions

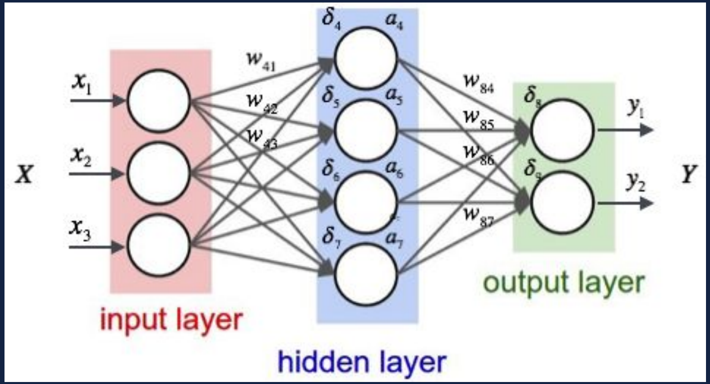
$$A^{\pi}(s_t, a_t) = Q^{\pi}(s_t, a_t) - V^{\pi}(s_t)$$

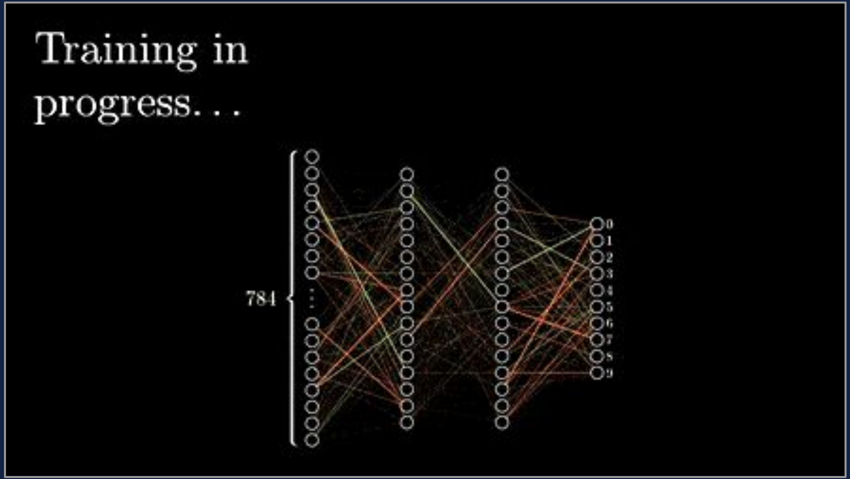
```
with tf.name_scope("losses"):
    self.pg_loss = self.pg_scalar * tf.reduce_mean((self.q_value - self.state_value) *
                   tf.nn.sparse_softmax_cross_entropy_with_logits(logits = self.outputs, labels = self.action_holder), name = "pg_loss")
    self.value_loss = self.value_scalar * tf.reduce_mean(tf.square(self.q_value - self.state_value), name = "value_loss")
    self.total_loss = self.pg_loss + self.value_loss
```

Neural Networks

Neural Networks (NN)

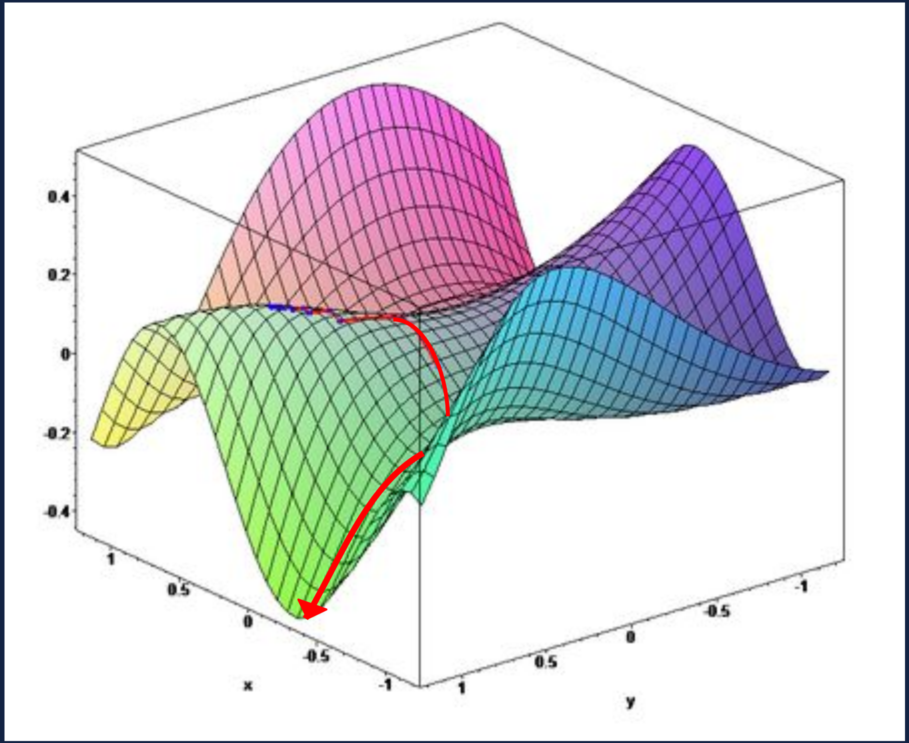
- Common approach to machine learning
- Input, hidden, and output layers
  - Path from one layer to the next consists of weights
- A series of linear transformations
- Variables within the network can be adjusted





Loss and Policy Gradients

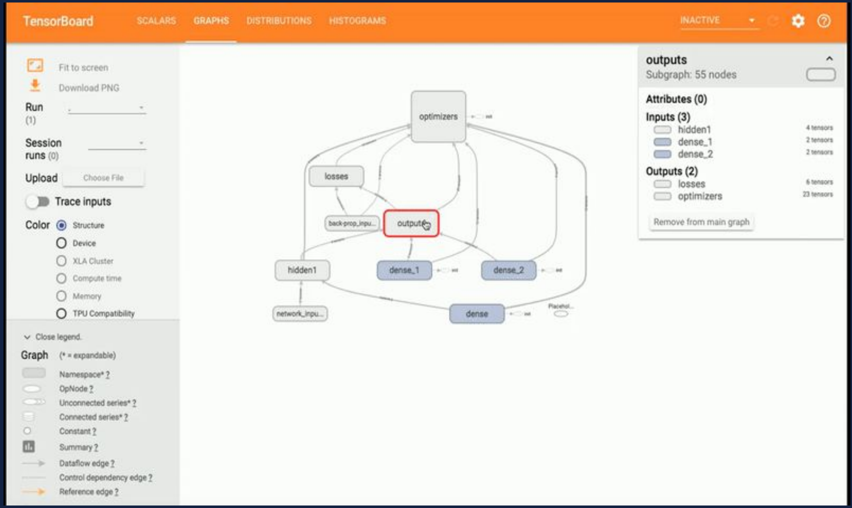
- Use gradients descent to adjust the neural network (i.e. using chain rule)
- Goal is to minimize a defined loss
  - Along ‘dimensions’ that represent weights and biases



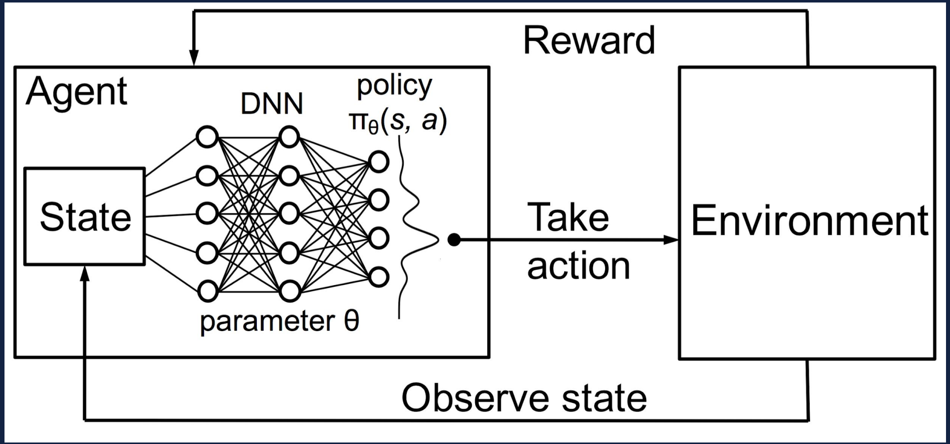
Building NN's Using TensorFlow

- TensorFlow library
    - Creates nodes and layers
    - Calculates loss gradients
- Tensorboard for visualization

```
# our input consists of our state
#   change to a one-hot format to make it easier for our network to recognize
with tf.name_scope("network_inputs"):
    self.state = tf.placeholder(shape = [None,1],dtype = tf.int32)
    self.state_one_hot= tf.one_hot(self.state, obs_space)

# keep track of our previous layer to feed to the next layer
previous= self.state_one_hot

# create our hidden layers using the new_hidden_layer method
for i in range(hidden_layers):
    with tf.name_scope("hidden"+str(i+1)):
        new_hidden_layer = self.__new_dense_layer(self.hidden_nodes,self.kernel_init,self.bias_init,previous)
        previous = new_hidden_layer

with tf.name_scope("outputs"):
    self.state_value = self.__new_dense_layer(1,self.kernel_init,self.bias_init,previous)
    self.outputs = self.__new_dense_layer(self.action_space,self.kernel_init,self.bias_init,previous)
    self.outputs = tf.squeeze(self.outputs)
```



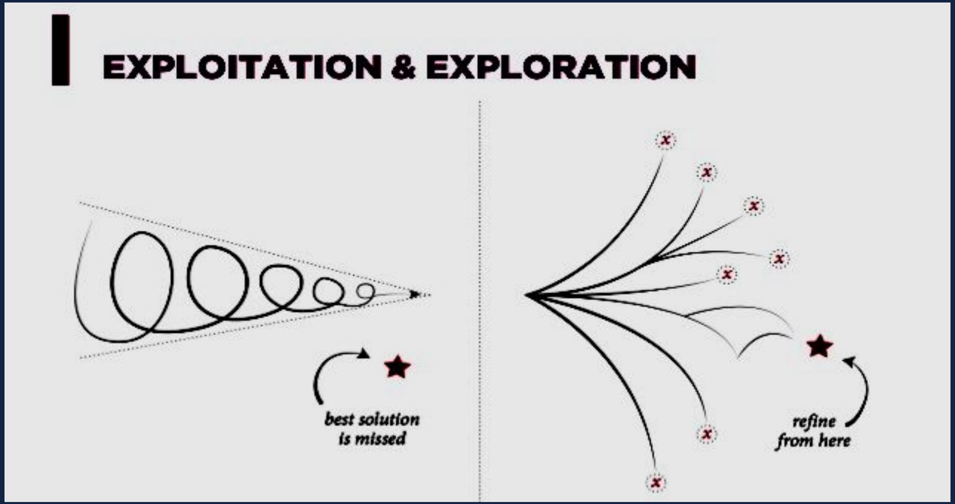
Reinforcement Learning w/ Neural Networks

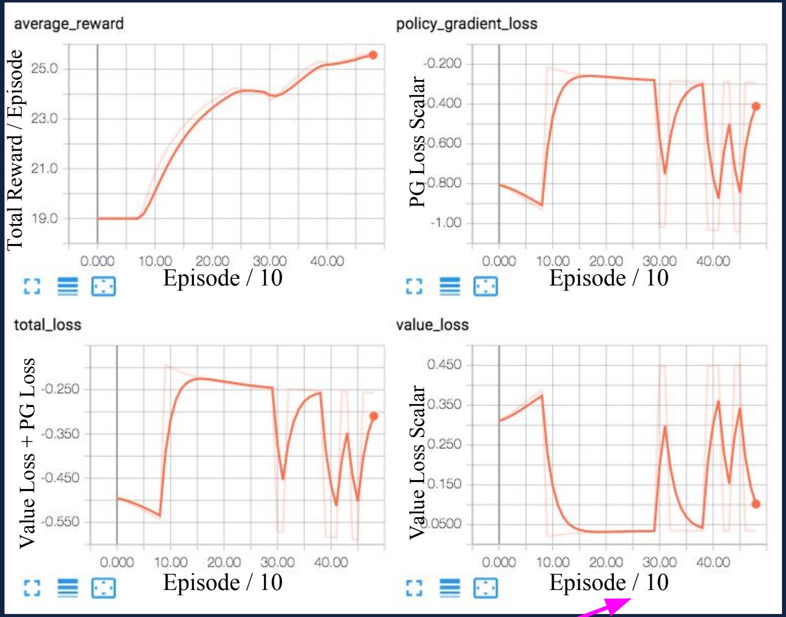


- Neural network becomes part of the agent
  - Input is the state, outputs an action
  - Reward adjusts the weights (policy)
  - Eventually becomes fine-tuned for optimization

Challenges of Reinforcement Learning

- Difficult to define a loss function
    - Policy gradient adjustment method
    - Actor-critic method
- ‘Exploitation’ versus ‘Exploration’



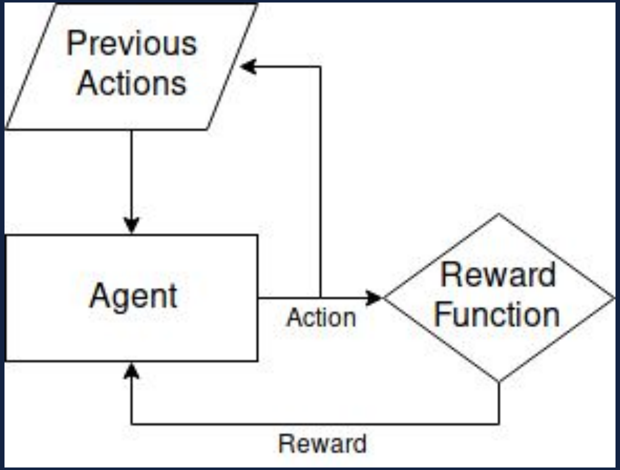


Plotted data once every 10 episodes

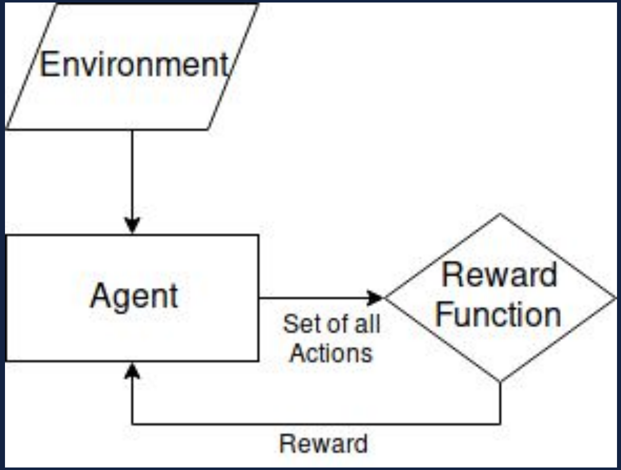
Observation Planning using Reinforcement Learning

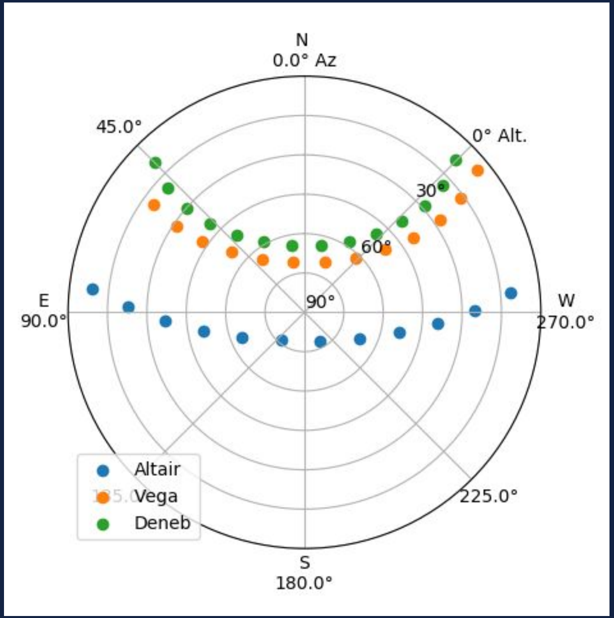
How to Represent the States and Actions

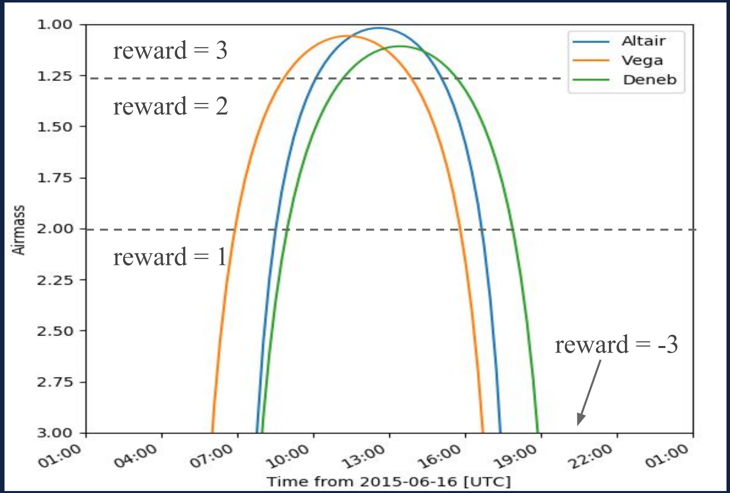
- Agent takes the current time as input
- Neural network learns the environment
- Must be trained on every new environment



- Environment as the input
- Outputs entire schedule
- Network is reusable for different environments

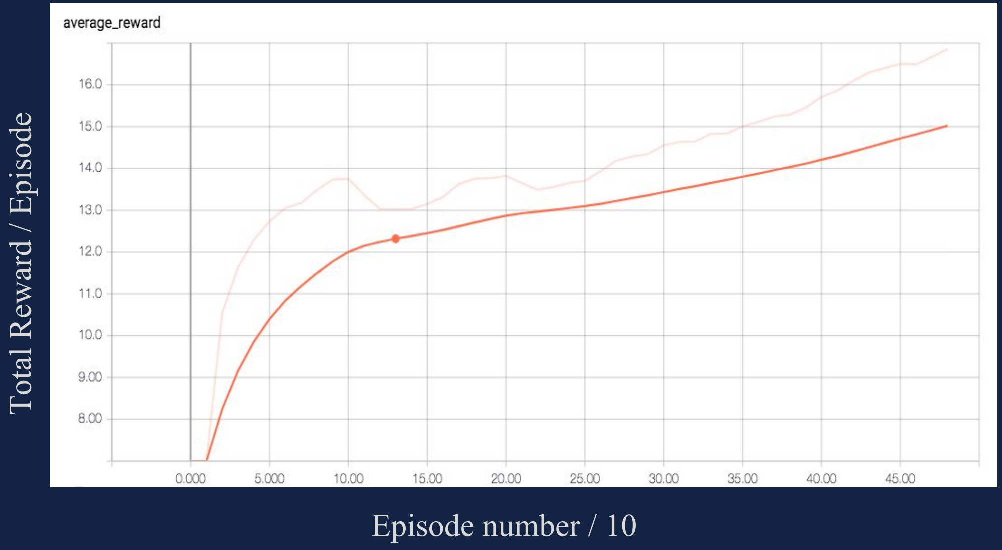


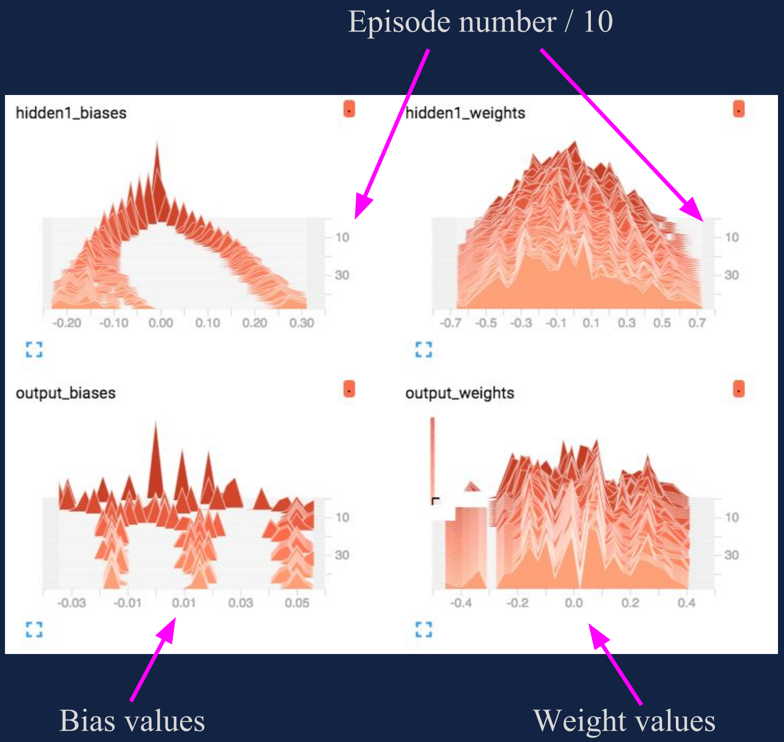





```
Telescope's Finalized Schedule:
        Time: 2015-06-16 06:00:00.000   Star: Vega
        Time: 2015-06-16 07:00:00.000   Star: Altair
        Time: 2015-06-16 08:00:00.000   Star: Vega
        Time: 2015-06-16 09:00:00.000   Star: Vega
        Time: 2015-06-16 10:00:00.000   Star: Vega
        Time: 2015-06-16 11:00:00.000   Star: Altair
        Time: 2015-06-16 12:00:00.000   Star: Deneb
        Time: 2015-06-16 13:00:00.000   Star: Deneb
        Time: 2015-06-16 14:00:00.000   Star: Deneb
        Time: 2015-06-16 15:00:00.000   Star: Deneb
        Time: 2015-06-16 16:00:00.000   Star: Altair
        Time: 2015-06-16 17:00:00.000   Star: Altair
        Time: 2015-06-16 18:00:00.000   Star: Deneb
        Time: 2015-06-16 19:00:00.000   Star: Deneb
        Time: 2015-06-16 20:00:00.000   Star: Deneb
```

Summer Triangle

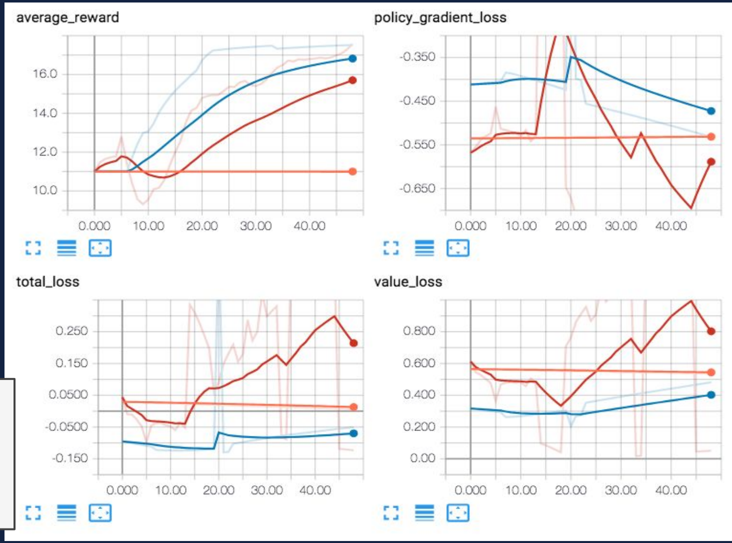




Tensorboard for Summer Triangle

Neural Network and Training Parameters

- Cannot just ‘put together’ a neural network
- Need to adjust hyper-parameters
  - Setting up the neural network
  - Training the network
- Evaluate reward and loss




```python
def __init__(self, obs_space, action_space,
             hidden_layers = 1,              # the number of hidden layers
             hidden_nodes = 20,              # the nodes per hidden layer
             kernel_init = "variance_scaling", # the weights initializer for each layer
             bias_init = "zeros",            # the bias initailizer for each layer
             pg_scalar = 1,                  # magnitude of policy gradient loss
             value_scalar = 1,               # magnitude of value loss
             learning_rate = 1e-2,           # learning rate for optimizer
             optimizer = "Adam",             # type of optimizer
             directory = None):              # directory name for tensorboard
```

```python
def train_agent(agent, data,
                pre_train = True,           # whether or not to pre-train state values
                display = True,             # whether or not to display/print
                plot = True,                # whether or not to plot to tensorboard
                pre_episodes = 50,          # how many pre-training episodes to conduct
                episodes = 500,             # how many episodes we want to run
                batch_size = 10,            # number of episodes per update for our agent
                display_rate = 50,          # how often to display/print our status
                plot_rate = 10,             # how often to write to tensorboard
                rewards_discount = 0.9,      # how much to discount our rewards by
                final_reward_subtraction = 2 # how much to subtract for each object we didn't pick
                ):
```

A Less Stringent Approach

- Method used until now
  - Using the timestep (i.e. hour of night) as input state
  - Choosing one object per timestep
  - Jumping from object to object
- Robotic telescopes
  - Capture entire images
  - Impractical to frequently redirect

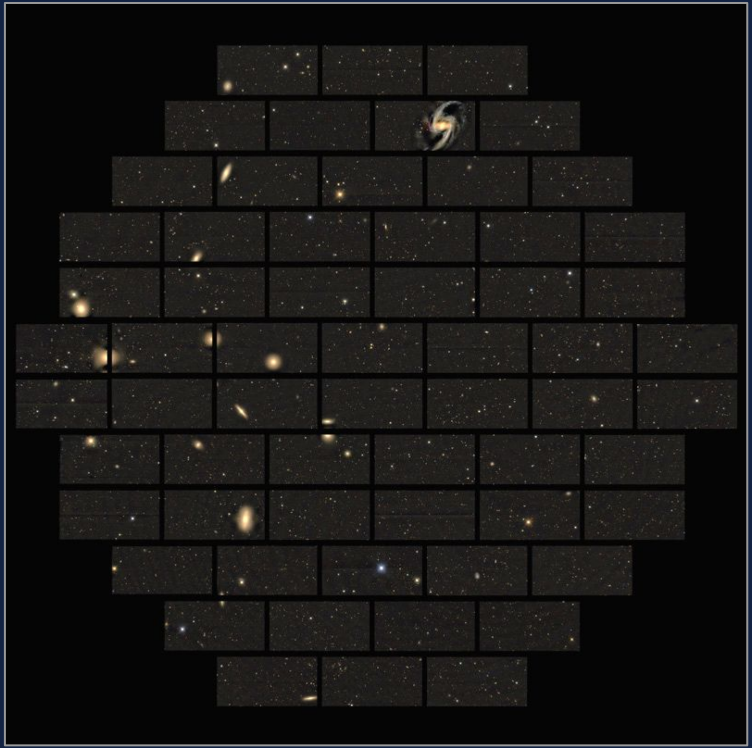
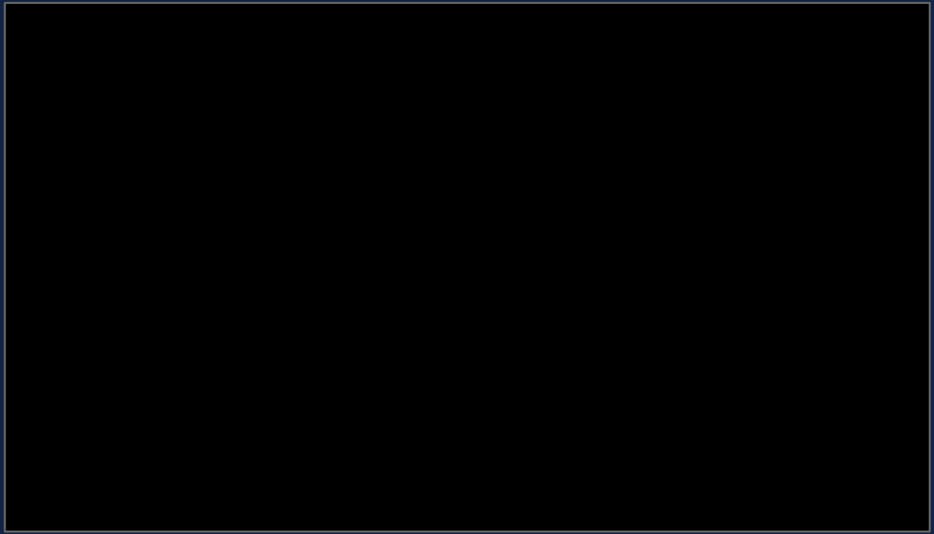


Image from Dark Energy Camera

Closer to an Atari Game



- Map celestial sphere onto 2D plane
- Reward function: define optimal and poor observation conditions
    - Distance from horizon, number of objects in frame
- Advantage: can account for more factors
- Disadvantage: less deterministic, more processing power

Next Steps using Reinforcement Learning

Accounting for More Factors

- Add more complexity to environment simulations
  - Weather
  - Multiple observations of each object
  - Apply correct wavelength filters

Crab Nebula

LSST and More

- Mostly set survey strategy
  - Multiple objectives and classes of objects
  - Arguably more complex than SDSS and DES
  - Combination of multiple projects
- However, open for white papers on ‘mini surveys’
  - 10-20% of survey

Key Takeaways

- Machine learning, neural networks, and reinforcement learning are simply iterative algorithms that utilize linear algebra and statistics to achieve certain goals
- Reinforcement learning is a powerful tool for finding an optimal policy to a Markov Decision Process
- Neural networks can provide the foundation for reinforcement learning algorithms
- Reinforcement learning is a fitting approach for optimizing telescope survey strategies

Our Code and More Resources

Our code on GitHub

Neural Networks and Deep Learning, by Michael Nielsen

TensorFlow Tutorials, from the TensorFlow website

Reinforcement Learning, an Introduction, by Richard S. Sutton

Gym by OpenAI, toolkit for reinforcement learning

Special Thanks

Jim Annis: SCD Scientist

Tom Carter: COD, QuarkNet Sponsor

George Dzuricsko: QuarkNet Teacher

Joshua Einstein-Curtis: AD Engineer

Angela Fava: ND Scientist, QuarkNet Mentor

Eric Neilsen: SCD Scientist

Brian Nord: SCD Scientist, Project Mentor

Gabe Perdue: SCD Scientist