Reinforcement Learning in Games

Trevor Clelland

Simon Fraser University

PROBLEM

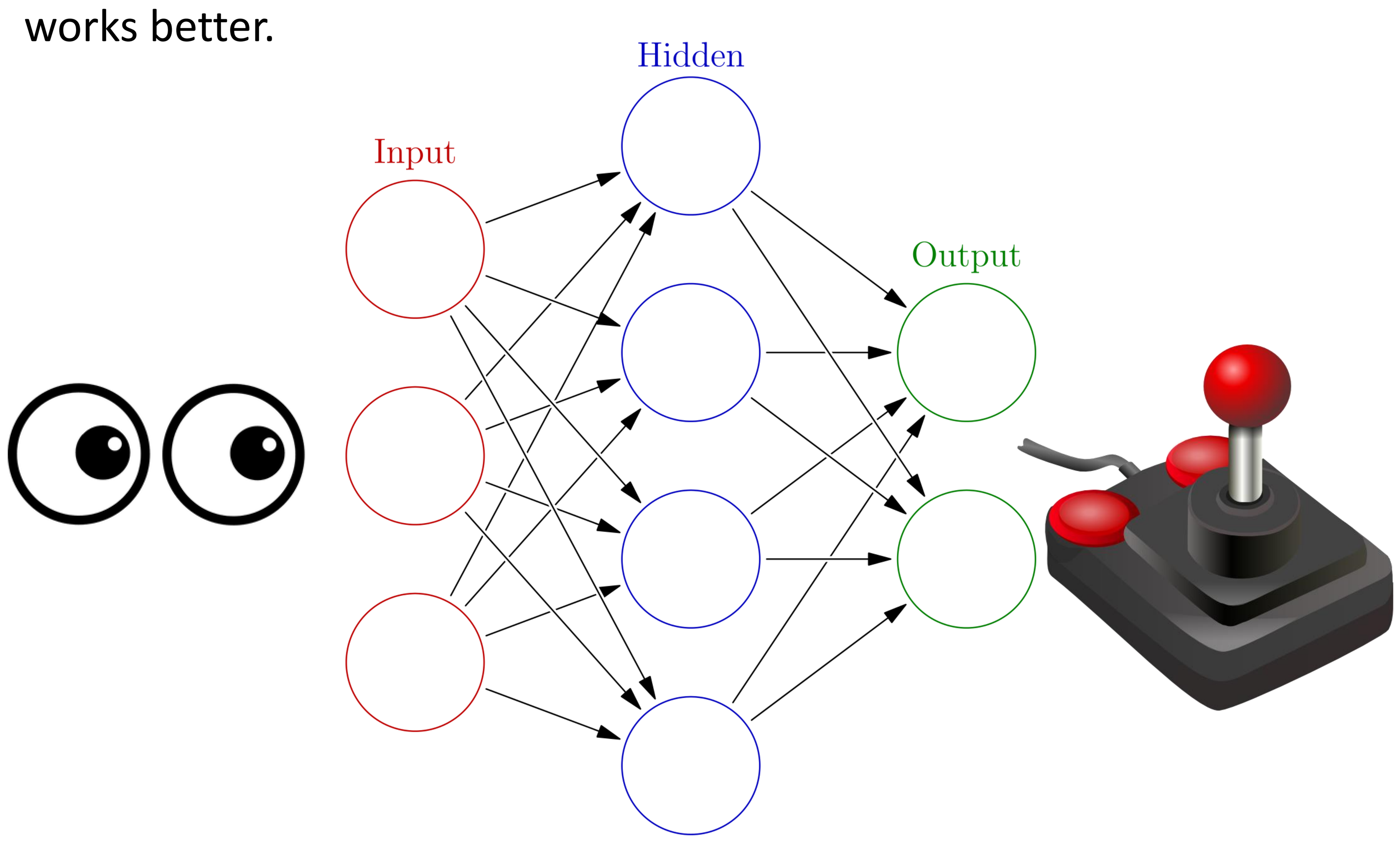
Teach agents to play a simulated game of Soccer using
– **Reward Functions** to signal to an agent whether actions it performs are good or bad
– **Curriculum Learning** where agents learn through a series of increasingly complex scenarios to overcome sparse reward signals

– **Simulated Environment** in Unity game engine where agents can take actions.
– **MLAgents** Unity plugin bridges the gap between Unity and Machine Learning by providing an implementation of OpenAI's Proximal Policy Optimization algorithm

Contribution: A Novel application of reinforcement learning. Reward functions and curriculum scenarios are tailored specifically for agents to learn to play soccer. Similar concepts could be applied to other games.

MODEL FORMULATION

Standard **Feed-forward** Neural Networks (NN) and **Long Short Term Memory** (LSTM) networks are used. Both are tested to see which one works better.



- Inputs = world perceptions
- Network = LSTM or standard feed-forward NN
- Outputs = Game controls; actions in the world

Using Proximal Policy Optimization

Policy Gradient Methods

"Run a policy for a while. See what actions led to high rewards. Increase their probability." – Andrej Karpathy

Comparison with Supervised Learning:
- Goal is to maximize some **Objective Function**
- "Vanilla" supervised learning:
- Maximize $\sum_i \log p(y_i \mid x_i)$ where Yi = labels, or expected output, Xi = input
- "Vanilla" Policy Gradient Method
  Maximize $\sum_i A_i \log p(y_i \mid x_i)$
- Yi is no longer expected output. We have no labels!
- Yi is the output of the current policy
- A = **Advantage Function**, measures success of action sequence
  - Common choice: **Sum of Discounted Rewards**
  - Could be as simple as {1 = success, -1 = failure}

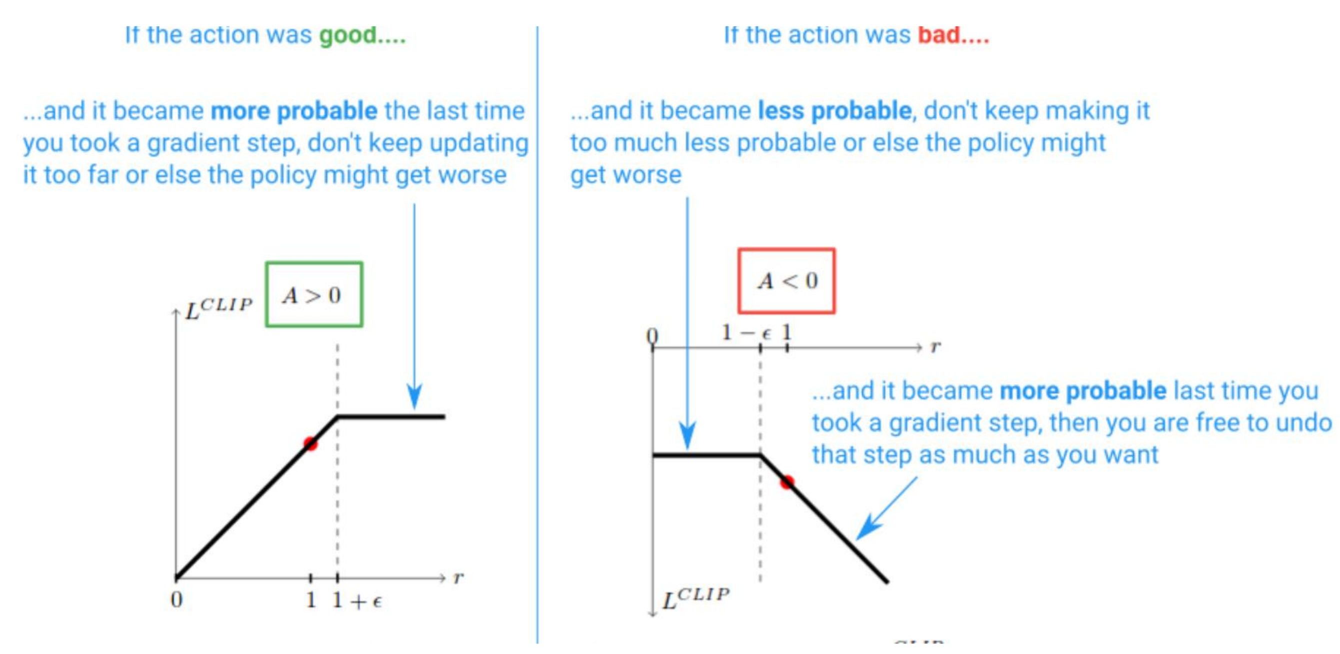
Actions that lead to rewards get likelihood increased
Actions that lead to failure get likelihood decreased

Proximal Policy Optimization

Advantage Function:
$\hat{A}_t$ = R_t – **Estimated rewards baseline**
where $R_t = \sum_{k=0}^{\infty} \gamma^k r_{t+k}$ i.e. **Sum of discounted Rewards**:
and **Estimated rewards baseline** is a separate supervised NN which predicts the "expected" reward
Advantage function result: determines whether an action is **better or worse than expected**

Objective Function:
Replace $\log p(y_i \mid x_i)$ with $r_t(\theta) = \frac{\pi_\theta(a_t|s_t)}{\pi_{\theta\,old}(a_t|s_t)}$, > 1 if new policy more likely

$$L^{CLIP}(\theta) = \hat{E}_t\left[\min(r_t(\theta)\hat{A}_t, \mathrm{clip}(r_t(\theta), 1-\varepsilon, 1+\varepsilon)\hat{A}_t)\right]$$



Advantages
- Simpler than other methods
- Prevents policy from changing too much in one update
- Gives ability to "undo" bad updates
- Shown to produce good results ex: OpenAI's Five

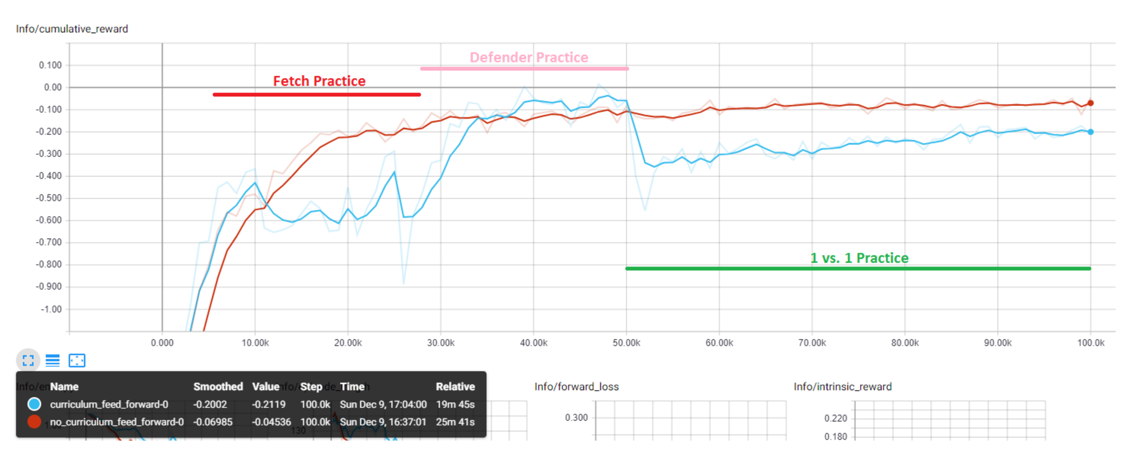
EXPERIMENTS

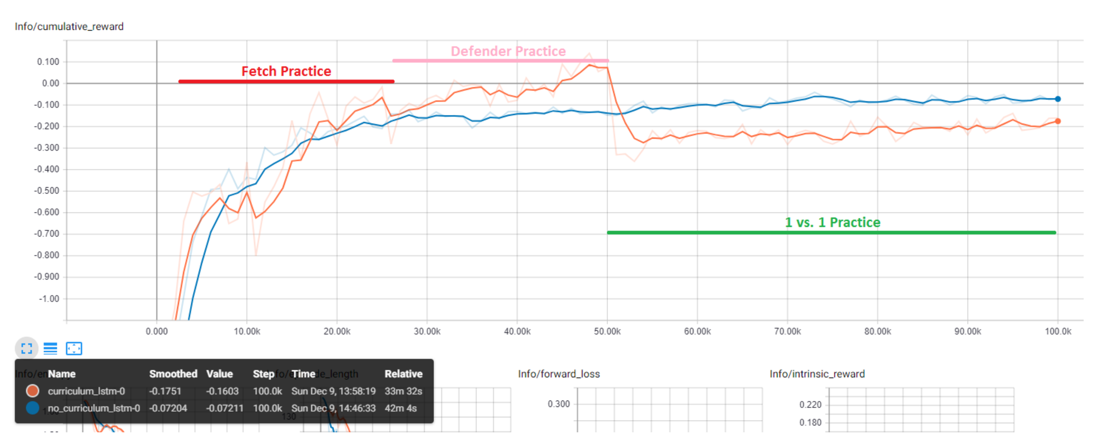
Different Lessons in Curricula:
- Grab
- Dribble Practice: grab ball and avoid obstacles
- 1v1
- Teams

Different Reward Functions:
- Is changing rewards for each lesson effective?
- Try to provide rewards for making progress

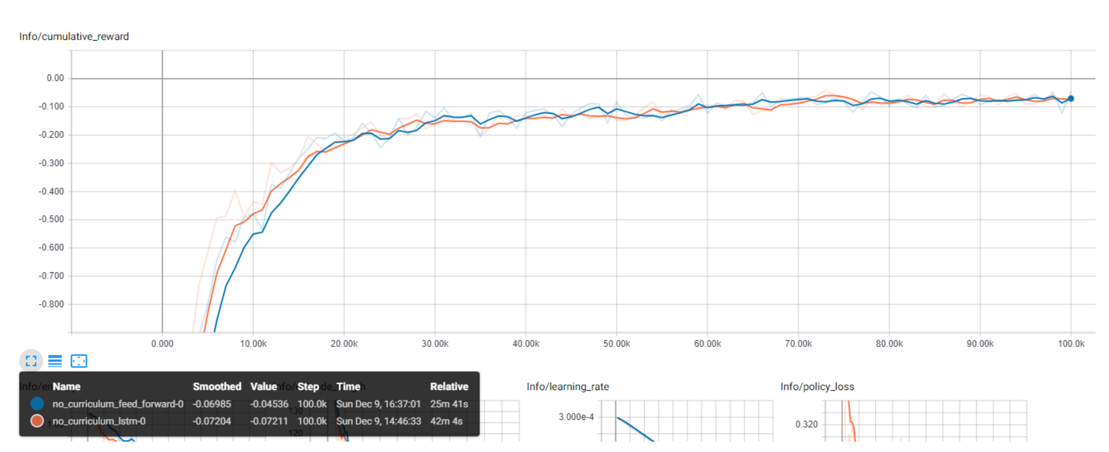
Tune Hyperparameters
- Gamma value controls discount of rewards
- Change network structure: layers, recurrent, etc…

Curriculum Learning: Feed-forward and Recurrent





Feed-forward vs. Recurrent



Hyperparameter Tuning