Reinforcement Learning for Turkish card game called “Batak”

Metehan Çekiç
Can Bakışkan

High Level Info

- Teaching a network to play Turkish card game “Batak” by using Reinforcement Learning methods.
- Asynchronous Advantage Actor Critic (A3C) algorithm is implemented for learning.
- Game environment for agents is written from scratch.
- LSTM based network model is used to capture temporal information.

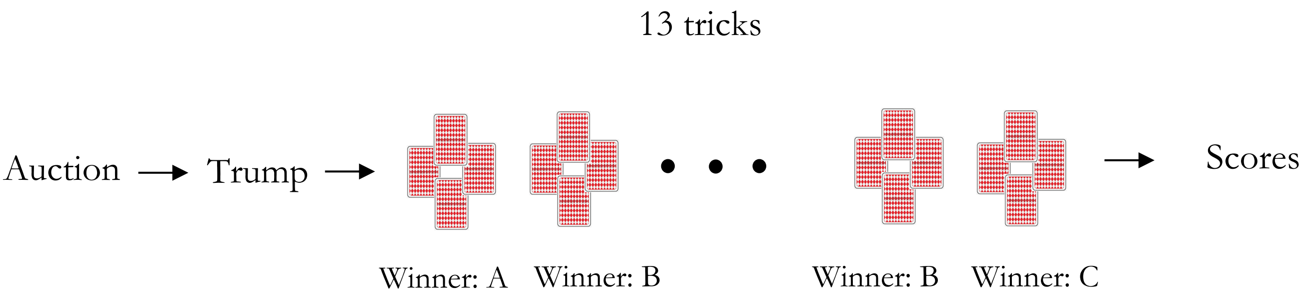
Batak Description

- Like “watered down” *bridge* or positive hands of *king*
- 4 person table
- Dealer deals 13 cards to each
- Each player bids on how many *tricks* they’re going to take in that *play* (consists of 13 tricks)
- If no one bids, first player wins the auction by 5 by default
- Winner determines *trump suit*
- In each trick players try to up the cards on the table by following suit

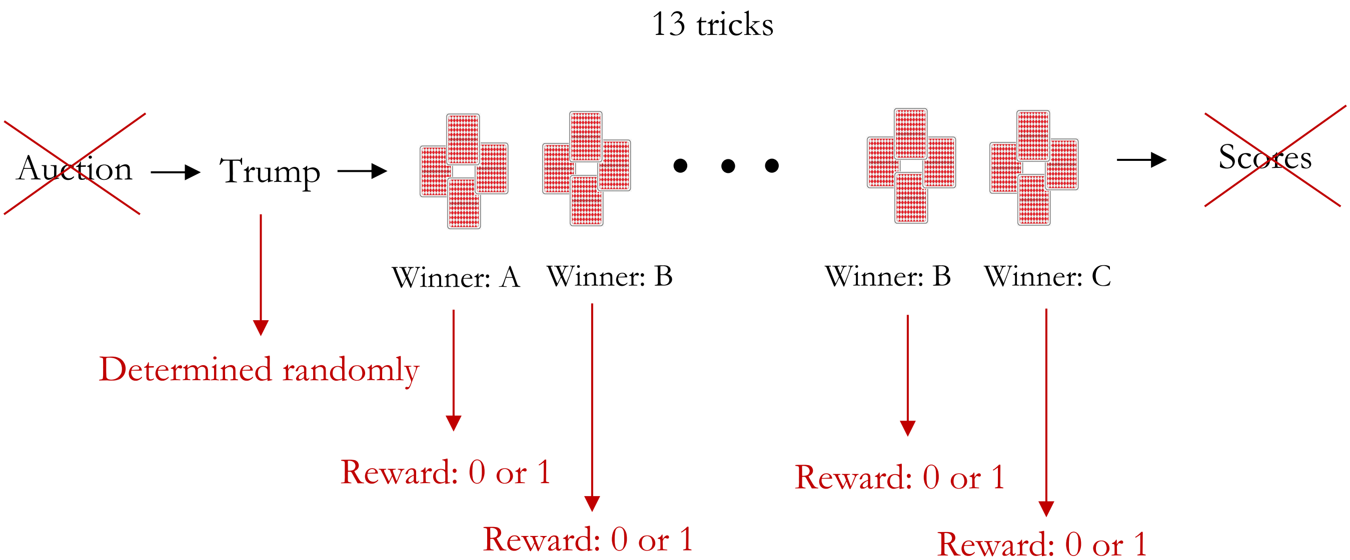
Batak Description (Continued)

- If player doesn’t have the suit they play from the trump suit.
- If trump cards have been played in a trick, largest trump card wins the trick.
- Trump suit cannot be played until someone “dropped” it i.e. a player played it on a different suit because they didn’t have that suit.
- Players have to raise the preceding card if they can.

Batak Description



Batak Description

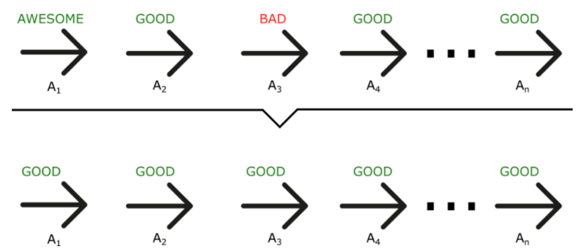


Background Information

Value Based Methods Vs. Policy Based Methods

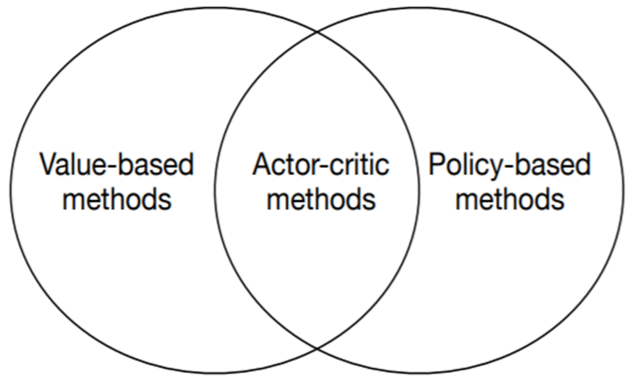
- Tries to approximate Q function or Value Function
- Finds policy by selecting actions that maximize Q
- More sample efficient when it works

- Directly optimizes policies
- Can work well for continuous action spaces
- Tends to be more popular these days



Advantage Actor-Critic (A2C) Agents

- Actor-Critic combines the benefits of both value based and policy based approach.
- A2C algorithm estimates both a value function ***V(s)*** (how good a certain state is to be in) and a policy ***π(s)*** (a set of action probability outputs).



Advantage Actor-Critic (A2C) Diagram

Value function:

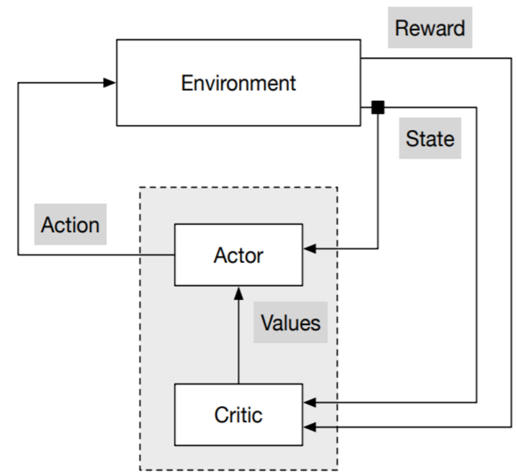
$$V_\pi(s) = \mathbb{E}_\tau(R_\tau \mid s_0 = s, \pi).$$

Q function:

$$Q_\pi(s, a) = \mathbb{E}_\tau(R_\tau \mid s_0 = s, a_0 = a, a_t \sim \pi).$$

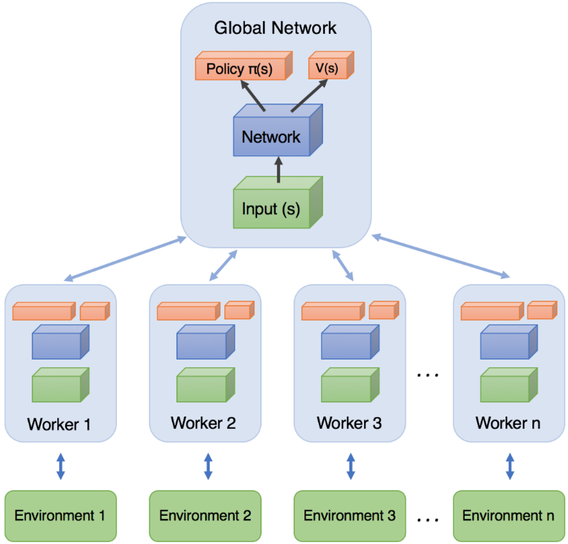
Advantage function:

$$A_\pi(s, a) = Q_\pi(s, a) - V_\pi(s).$$



Asynchronous Advantage Actor-Critic (A3C)

- In addition to stabilizing learning, using multiple parallel actor-learners has multiple practical benefits.
- First, it reduces training time that is roughly linear in the number of parallel actor-learners.
- Second, since we no longer rely on experience replay for stabilizing learning we are able to use actor-critic algorithm.



How A3C works?

We choose our actions using a conditional probability distribution $P(a|x)$ over the possible actions, given the observation.

$\pi(a|x;\theta) = P(a|x;\theta)$ Policy

Trajectory:

$$\tau = (x_0, a_0, r_1, x_1, \ldots, x_{T-1}, a_{T-1}, r_T, x_T).$$

! Each worker has its own trajectory

How A3C works?

- Empirical Reward: $R = r_{t_0} + \gamma r_{t_1} + \gamma^2 r_{t_2} \cdots$
- Advantage : $\sum (R - V(s;\theta))^2$
- Value Loss : $A_\pi(s, a; \theta) = R - V(s;\theta)$
- Policy Loss : $\log\left(\frac{1}{\pi(a|s;\theta_p)}\right) * A_\pi(s, a; \theta_v) - \beta H(\pi)$

1. Compute gradient descent of losses for each worker neural net.
2. Accumulate gradient descent from each step neural network.
3. Update Master neural network according to accumulated updates
4. Update all worker networks with master network

A3C Algorithm

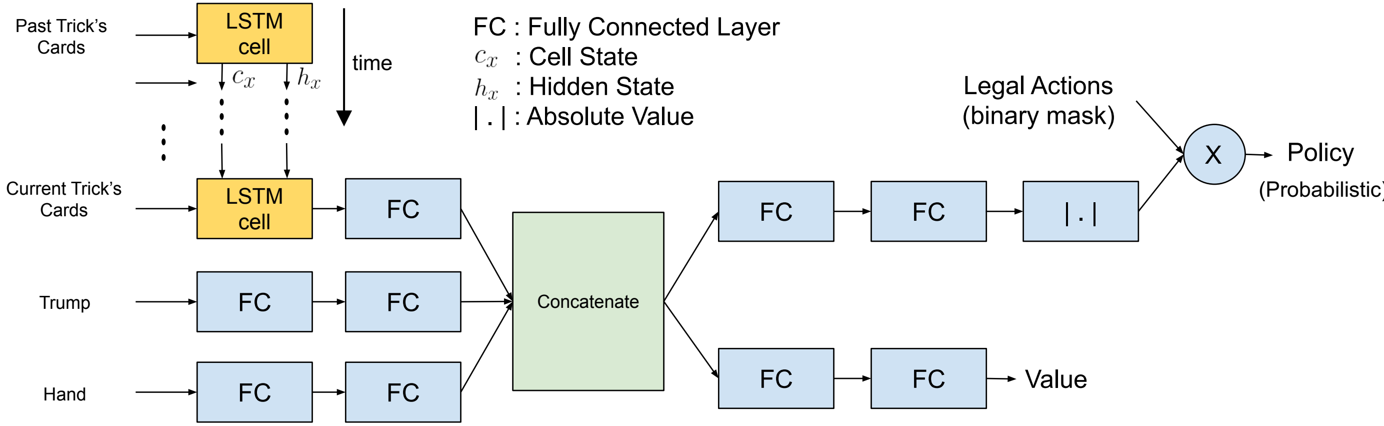
1. Play the game in each worker environment with current policies
2. Compute gradient descent of losses for each worker neural net at the end of game.
3. Accumulate gradient descent from each step neural network.
4. Update Master neural network according to accumulated updates
5. Update all worker networks with master network
6. Calculate expected reward.
7. Go back to step 1 until convergence

A3C Algorithm

Algorithm S3 Asynchronous advantage actor-critic - pseudocode for each actor-learner thread.

// Assume global shared parameter vectors θ and θ_v and global shared counter $T = 0$
// Assume thread-specific parameter vectors θ' and θ'_v
Initialize thread step counter $t \leftarrow 1$
repeat
    Reset gradients: $d\theta \leftarrow 0$ and $d\theta_v \leftarrow 0$.
    Synchronize thread-specific parameters $\theta' = \theta$ and $\theta'_v = \theta_v$
    $t_{start} = t$
    Get state s_t
    repeat
        Perform a_t according to policy $\pi(a_t|s_t;\theta')$
        Receive reward r_t and new state s_{t+1}
        $t \leftarrow t + 1$
        $T \leftarrow T + 1$
    until terminal s_t **or** $t - t_{start} == t_{max}$
    $R = \begin{cases} 0 & \text{for terminal } s_t \\ V(s_t, \theta'_v) & \text{for non-terminal } s_t \text{// Bootstrap from last state} \end{cases}$
    for $i \in \{t-1, \ldots, t_{start}\}$ **do**
        $R \leftarrow r_i + \gamma R$
        Accumulate gradients wrt θ': $d\theta \leftarrow d\theta + \nabla_{\theta'} \log \pi(a_i|s_i;\theta')(R - V(s_i;\theta'_v))$
        Accumulate gradients wrt θ'_v: $d\theta_v \leftarrow d\theta_v + \partial\left(R - V(s_i;\theta'_v)\right)^2 / \partial\theta'_v$
    end for
    Perform asynchronous update of θ using $d\theta$ and of θ_v using $d\theta_v$.
until $T > T_{max}$

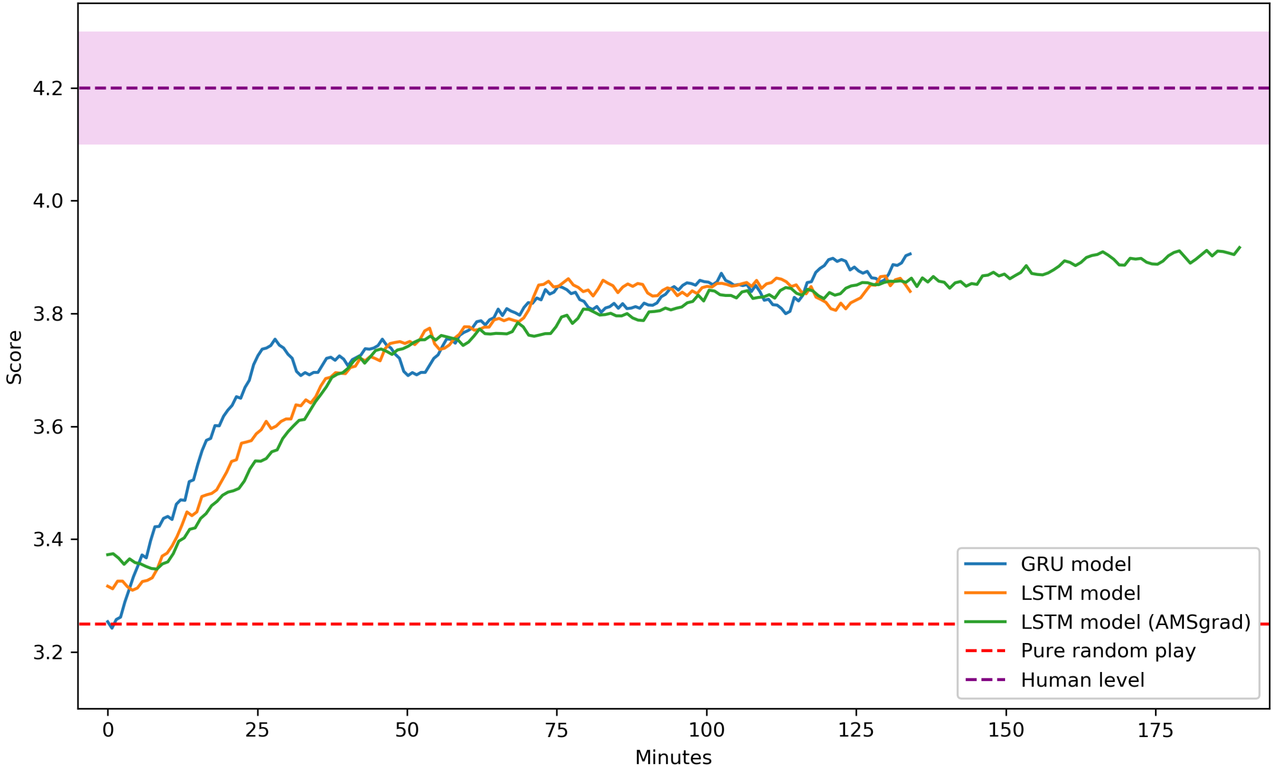
Our Model Architecture



Details

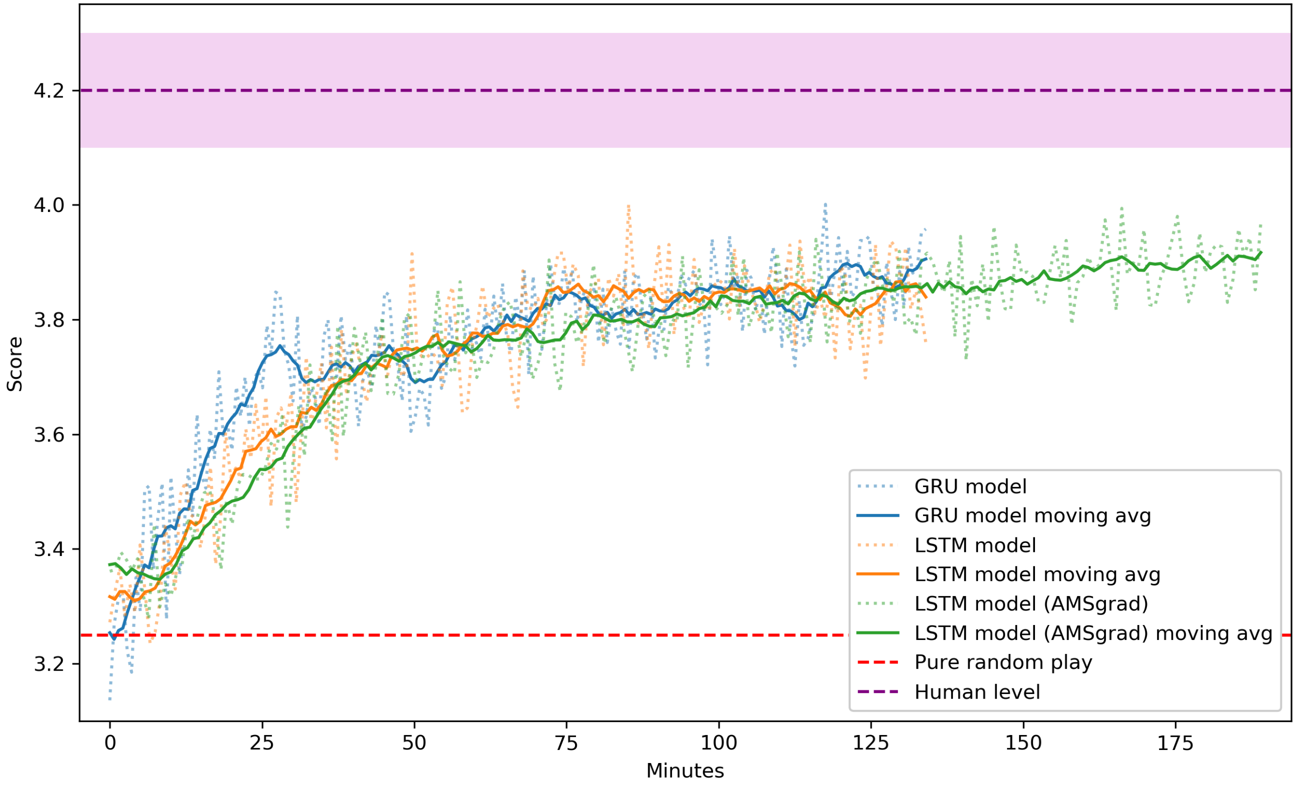
- Each agent trains in a table with other players playing at random.
- Test accuracy is averaged over 1000 plays
- We trained three different versions

Results



4.2
4.0
3.8
3.6
3.4
3.2
Score
0
25
50
75
100
125
150
175
Minutes
GRU model
LSTM model
LSTM model (AMSgrad)
Pure random play
Human level

Results



Score
Minutes
4.2
4.0
3.8
3.6
3.4
3.2
0
25
50
75
100
125
150
175
GRU model
GRU model moving avg
LSTM model
LSTM model moving avg
LSTM model (AMSgrad)
LSTM model (AMSgrad) moving avg
Pure random play
Human level

Future work:

- Self play
- Full batak game
- Other card or multiagent games

References

1. Asynchronous Methods for Deep Reinforcement Learning, Deepmind 2016
2. Schulman John et al. 2015, High-Dimensional Continuous Control Using Generalized Advantage Estimation
3. Amsgrad: Reddi S. et al. 2018, On The Convergence Of Adam And Beyond
4. Parts of code from: https://github.com/dgriff777/rl_a3c_pytorch
5. Bridge terms from: https://en.wikipedia.org/wiki/Glossary_of_contract_bridge_terms

Thank You for Listening