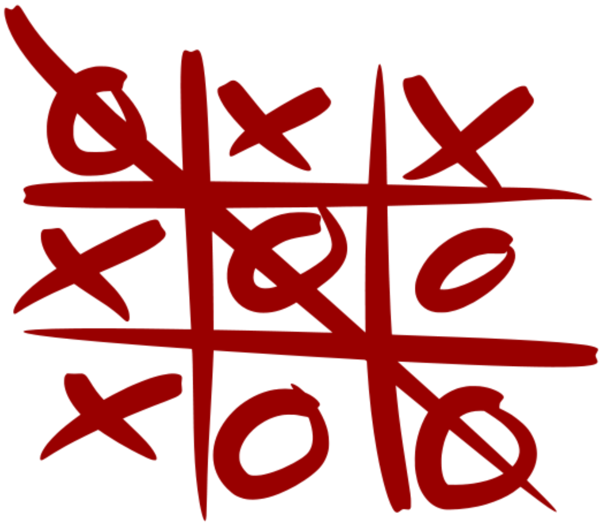
CSC411/2515 Project 4: Tic-Tac-Toe with Policy Gradient



For this project, you will use Reinforcement Learning to train a neural network to play Tic-Tac-Toe.

Starter Code.

The starter code is available at tictactoe.py. You will need to use PyTorch 0.3+ for this project (you can use `torch.__version__` to see which version you are using). The starter code contains the Tic-Tac-Toe environment and other helpful code.

Part 1: Environment (5 pts)

The `Environment` class provides functionality to play the game of Tic-Tac-Toe. Look at the code. How is the grid represented? What do the attributes `turn` and `done` represent?

Create a new environment, and play a game of Tic-Tac-Toe against yourself by calling the `step()`, and `render()` methods. Display the text output in your report.

Part 2: Policy (15 pts)

Part 2(a) (10 pts)

The `Policy` class is incomplete. Complete the implementation so that your policy is a neural network with one hidden layer. Include the new code you wrote in your report.

Part 2(b) (3 pts)

Even though Tic-Tac-Toe has a 3x3 grid, we will represent our state as a 27-dimensional vector. You can see how the 27-dimensional state is generated in the `select_action` function. In your report, state what each of the 27 dimensions mean.

Part 2(c) (2 pts)

The output of this policy should be a 9-dimensional vector. Explain what the value in each dimension means.

Is this policy stochastic or deterministic?

Part 3: Policy Gradient (15 pts)

The function `finish_episode` computes the backward pass for a single episode, in our case a single game of Tic-Tac-Toe.

Note: You may notice that `finish_episode` normalizes the returns to mean 0 and standard deviation 1. This is done to reduce the gradient norm and to train faster.

Part 3(a): computing returns (10 pts)

Implement the `compute_returns` function, which takes a list of rewards r_t and computes the returns

$$G_t = r_t + \gamma r_{t+1} + \gamma^2 r_{t+2} + \ldots.$$

Include your implementation in your report.

Part 3(b) (5 pts)

Note that we pass a list of `saved_rewards` and `saved_logprobs` for the entire episode to `finish_episode`. When using Policy Gradient, we cannot compute the backward pass to update weights until the entire episode is complete. Explain why this is the case: why can we not update weights in the middle of an episode?

Part 4: Rewards (10 pts)

We did not set up the environment to return rewards directly. Instead, when an action is played, the Tic-Tac-Toe environment returns a status. The `get_reward` function determines the reward for each status.

Part 4(a) (5 pts)

Modify `get_reward` function to what you think the reward should be to be able to train the policy to play Tic-Tac-Toe. Include your modified function in your report. For your reference, here are the status meanings:

- `STATUS_VALID_MOVE`: Move was on an empty position, game is not over, opponent's turn.
- `STATUS_INVALID_MOVE`: Move was on a non-empty position and was ignored
- `STATUS_WIN`: Move was a valid, winning move, and the game is over
- `STATUS_TIE`: Move was valid, no more empty positions on board, game is over.
- `STATUS_LOSE`: Opponenet wins the game, game is over
- `STATUS_DONE`: You should never actually see this

Part 4(b) (5 pts)

Explain the choices that you made in 4(a). Are there positive rewards? Negative rewards? Zero rewards? Why? Explain how you chose the magnitude of the positive and negative rewards.

Part 5: Training (30 pts)

After you have completed parts 1-4, you should be able to train your model using the `train()` function. Train your model until your average return stops improving. The average returns can be fairly noisy, so you should train for at least 50000 episodes. You may change any hyperparameters in this function, but clearly indicate in part (a) if you have done so. You may have to return to Part 4 if your choice of rewards was poor.

Note that this function checkpoints learned model weights of the policy throughout training. We will use this in later parts.

Part 5(a) (5 pts)

Plot a training curve: your x-axis should be episodes, and your y-axis should be the average return. Clearly indicate if you have changed any hyperparameters and why.

Part 5(b) (10 pts)

One hyperparameter that you should optimize is the number of hidden units in your policy. Tune this hyperparameter by trying at least 3 values. Report on your findings

Part 5(c) (5 pts)

One of the first things that your policy should learn is to stop playing invalid moves. At around what episode did your agent learn that? State how you got the answer.

Part 5(d) (10 pts)

Use your learned policy to play 100 games against `random`. How many did your agent win / lose / tie? Display five games that your trained agent plays against the random policy. Explain any strategies that you think your agent has learned.

Part 6: Win Rate over Episodes (10 pts)

Use the model checkpoints saved throughout training to explore how the win / lose / tie rate changed throughout training. In your report, include graphs that illustrate this, as well as your conclusions in English.

Part 7: First Move Distribution over Episodes (10 pts)

For your final trained model, show $\pi(a|s_0)$, the learned distribution over the first move. (You may use the `first_move_distr` function provided) What has your model learned? Does the distribution make sense?

Explore how the distribution over the first move has changed throughout training.

Part 8: Limitations (5 pts)

Your learned policy should do fairly well against a random policy, but may not win consistently. What are some of the mistakes that your agent made?

What to submit

The project should be implemented using Python 2 or 3 and should be runnable on the CS Teaching Labs computers. Your report should be in PDF format. You should use LaTeX to generate the report, and submit the .tex file as well. A sample template is on the course website. You will submit at least the following files: `tictactoe.py`, `tictactoe.tex`, and `tictactoe.pdf`. You may submit more files as well. You may submit `ipynb` files in place of `py` files.

Reproducibility counts! We should be able to obtain all the graphs and figures in your report by running your code. Submissions that are not reproducible will not receive full marks. If your graphs/reported numbers cannot be reproduced by running the code, you may be docked up to 20%. (Of course, if the code is simply incomplete, you may lose even more.) Suggestion: if you are using randomness anywhere, use `numpy.random.seed()`.

You must use LaTeX to generate the report. LaTeX is the tool used to generate virtually all technical reports and research papers in machine learning, and students report that after they get used to writing reports in LaTeX, they start using LaTeX for all their course reports. In addition, using LaTeX facilitates the production of reproducible results.

Available code

You are free to use any of the code available from the CSC411 course website.

Readability

Readability counts! If your code isn't readable or your report doesn't make sense, they are not that useful. In addition, the TA can't read them. You will lose marks for those things.

Academic integrity

It is perfectly fine to discuss general ideas with other people, if you acknowledge ideas in your report that are not your own. However, you must not look at other people's code, or show your code to other people, and you must not look at other people's reports and derivations, or show your report and derivations to other people. All of those things are academic offences.