Genetic Algorithms and Competitive Self-Play in Training Deep Neural Networks Agents

Matthew Baas 20786379@sun.ac.za

Abstract

In reinforcement learning, agents represented by deep neural networks are usually trained via gradient-based learning algorithms. Recent work, however, has shown that gradient-free genetic algorithms can competitively train complex neural networks to accomplish tasks. In this work, we combine genetic algorithms with competitive self-play to train an agent with an autoregressive policy to play a tower-defense game.

Game Data

The game used is a Tower Defense game by Entelect software. It consists of a grid where at each turn, both agents take an action which consists of both *what building to build* and *where on the grid to place it*. There is an in-game economy, with some buildings acting as defences, 'income-generators', or offensive structures. The goal is to reduce the opponent's base's life to 0 and agents can only build on their side of the grid. An example a 4x8 map with indicated grid coordinates and bases:

Player A Home Base	0,0	1,0	2,0	3,0	3,0	2,0	1,0	0,0	Player B Home Base
	0,1	1,1	2,1	3,1	3,1	2,1	1,1	0,1	
	0,2	1,2	2,2	3,2	3,2	2,2	1,2	0,2	
	0,3	1,3	2,3	3,3	3,3	2,3	1,3	0,3	

Preprocessing

All the game information is processed to give an observation tensor of shape 8x8x38 (channel at end), containing both map info and embeddings of non-spatial data. This is the input given to each agent.

Policy Representation

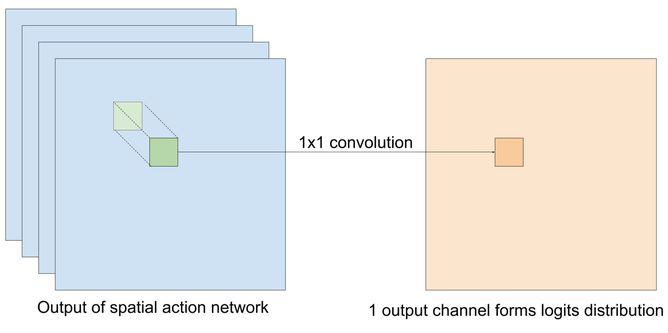
Since the building must first be chosen before knowing what spatial actions are available, an autoregressive style policy π is used:

$$\pi(a|s) = \pi(building|s) \cdot \pi(coordinate|building, s)$$

Where *a* and *s* is the full action and state, respectively.

So the following is performed in order:

1. The agent infers the logits for the buildings and then samples from that distribution to decide on what to build.
2. The sampled action is then *embedded* in a one-hot vector and *broadcast over the width and height of the spatial information*.
3. The agent then, using the embedded base action and other spatial data, infers the logits for the coordinates and the chosen coordinate is taken as a sample from this distribution.



To preserve spatial information, a 1x1 convolution is used on the final stack of convolution channels and then flattened to obtain the logit distribution for the coordinates.

Methods

Genetic algorithm

- Initialize population of models and a queue of reference bots
- For each generation,
  - Sample reference agent at random from reference agent queue
  - Play collective rollout of all agents in population (each against the same reference agent) to determine total discounted rewards for each agent
  - Truncate population to a few of the agents with the highest rewards
  - Evaluate truncated population for several more episodes to find true best agent at current generation
  - Repopulate the agents by mutating the best agents
- Every generation pop an agent off the reference agent queue, and push one of the better agents at the current generation to the queue.

Mutation algorithm

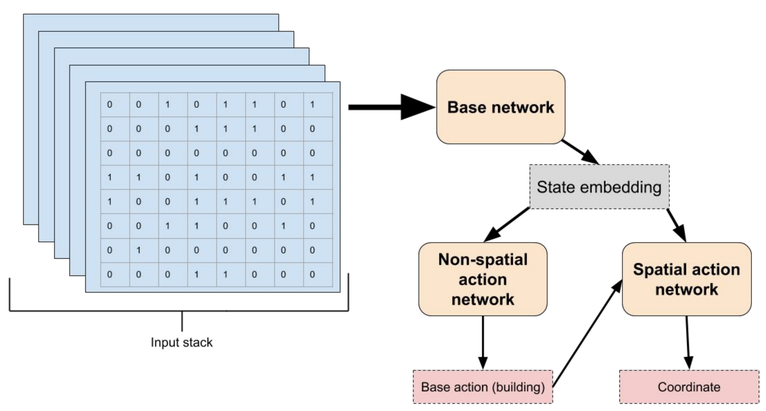
Given a deep neural network parameterized by a vector **θ**, a mutated network is produced by adding gaussian noise to θ:

$$\theta_{new} = \theta + \sigma \cdot \mathcal{N}(0, 1)$$

Where σ is a hyperparameter to scale the noise. For a network with ~2.1 million parameters, we found σ = 0.0021 to be the best fit.

Model Architecture

The model consists of a base network, which processes the observation tensor into a state embedding, which is then used by the non-spatial action network and spatial action network to infer the two actions for each step.



The architecture of each network is as follows:

Base		
Type	Channels/units	Filter
conv	32	3x3
conv	32	3x3
conv	64	3x3
inception-4 resnet A module		
inception-4 resnet A module		

Non-spatial network		
Type	Channels/units	Filter
inception-4 resnet B module		
inception-4 resnet B module		
conv	8	1x1
dense	256	
dense	6	
sample categorical		

Spatial network		
Type	Channels/units	Filter
conv	64	3x3
inception-4 resnet B module		
inception-4 resnet B module		
conv	1	1x1
sample categorical		

Remarks:

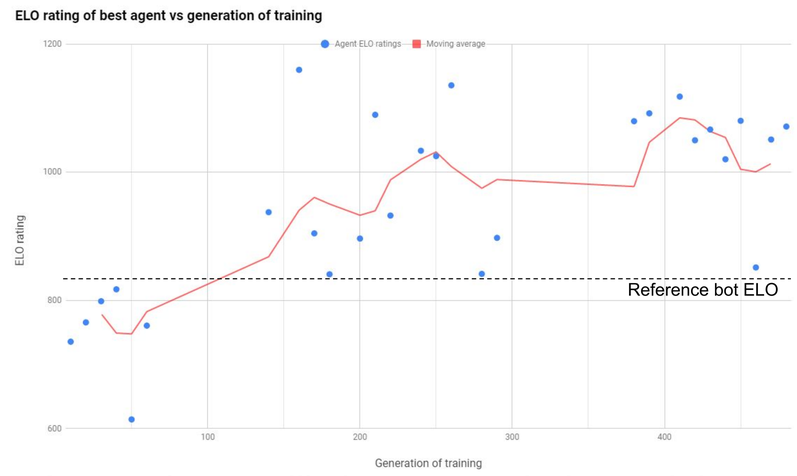
- All activations are relu except for the final dense layers.
- The agents in the population do not share parameters.
- There is no dilation for the conv layers.
- The agent's reward was annealed from the provided in-game score at the beginning of training to a binary win/lose reward at the end.

Results

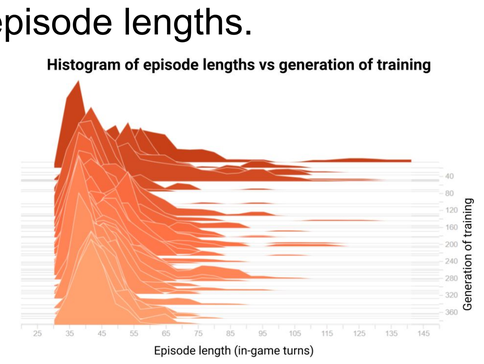
Training was done with the following parameters:

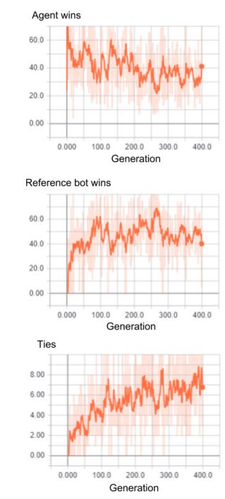
Training Parameters	
Generations	480
Population size	88
Truncation size	8
Mutation scalar σ	0.0021
Additional episodes for truncated population	8

At the end, agent checkpoints from several generations were evaluated via a round-robin matchup to generate standard ELO scores.



- Agent elo steadily increases during training, but improvement appears to diminish later in training.
- Agent strategy becomes more refined over training, with lower variance in elo and episode lengths.





Agents and reference bot each generation maintain balanced win/loss ratio, yielding stable training.

Conclusions

- Genetic algorithms can be combined with self-play to achieve convincing results in training a complex network to play a moderately complex game.
- Achieving stability and low-variance in training with genetic self-play is non-trivial and requires additional methods to perform well.
- Genetic algorithms can successfully train agents with autoregressive policy representations.

Future Work

- Compare results with baseline deep RL training methods.
- Experiment with more compute resources with bigger populations and more generations.
- Explore effects of training different network sizes

References

1. Such, F.P., Madhavan, V., Conti, E., Lehman, J., Stanley, K.O. and Clune, J., 2017. Deep neuroevolution: genetic algorithms are a competitive alternative for training deep neural networks for reinforcement learning. *arXiv preprint arXiv:1712.06567*.
2. Bansal, T., Pachocki, J., Sidor, S., Sutskever, I. and Mordatch, I., 2017. Emergent complexity via multi-agent competition. *arXiv preprint arXiv:1710.03748*.
3. Szegedy, C., Ioffe, S., Vanhoucke, V. and Alemi, A.A., 2017, February. Inception-v4, inception-resnet and the impact of residual connections on learning. In *AAAI* (Vol. 4, p. 12).