Witam gitas

Kamil Kozioł, Magdalena Sadowska, Wiktor Leszczyński

1 Introduction

This project intends to compare 4 different AI implementations which can be used when making bots that play games

1.1 Podsekcja

ziut DAWAJ TEKST

2 Conditions

We have now added a title, author and date to our first LaTeX document!

2.1 Podsekcja

ziut

3 Genetic algorithm

Genetic Algorithm is a machine learning technique that uses principles of natural selection to find the optimal solution to a problem.

We can determine how good individual is doing by calculating his fitness.

3.1 Algorithm

Each of these steps is repeated untill conditions are met and then the current generation value is increased

3.1.1 Initialization

The first step is initializing a population using random neural networks, where each solution represents a possible strategy for playing the game.

Done only once at the first generation

3.1.2 Evaluation

Each solution is evaluated by playing the game using the corresponding neural network. Fitness score of the solution is calculated based on its performance in the game. The fitness score is a measure of how well the solution performs in achieving the objective of the game.

3.1.3 Selection

Then propabilities of picking each individual are made. Those propabilities range from $\langle 0, 1 \rangle$

3.1.4 Mutation

To introduce diversity into the population, picked by random individuals undergo mutation, where random changes are made to their weights and biases. This process helps to prevent the population from converging too quickly to a local optimum.

Every weight and bias has a change of being mutated based on $mutationRate$

If mutated those are modified using this formula:

$$x = x + randomGaussian(initMean, initStdev) * mutationPower$$

Then being kept at $\langle minValue, maxValue \rangle$

3.1.5 Crossover

The selected solutions are then combined through crossover, where random pairs of individuals exchange weights and biases of neural network to create new offspring solutions. This process mimics the natural process of sexual reproduction, where genes from two parents combine to produce offspring with a mix of genetic traits.

3.1.6 Elites

Some of the best-performing solutions from the previous generation are also added to the new generation

3.2 Fitness function

Fitness function is calculated by this formula:

$$fitness = \begin{cases} age^2 * (2^{apples}) & apples < 10 \\ age^2 * (2^{10}) * (apples - 9) & apples \geq 10 \end{cases}$$

Each snake in the game has a hunger variable, denoted as H. Upon consumption of an apple, H is replenished to a maximum value of $H_{\max}$. At every time step, H decreases and once it reaches 0, the snake perishes.

3.3 Hyperparameters

Hyperparameters can be modified in file *settings.json*.

3.3.1 Neural network

Neural network used to train the snakes is described by this model:

$$Model = \begin{array}{l} InputLayer(28, linear) \\ DenseLayer(20, relu) \\ DenseLayer(12, relu) \\ OutputLayer(4, softmax) \end{array}$$

Description:

$$Layer(neurons, activationFunction)$$

4 NEAT

We have now added a title, author and date to our first LaTeX document!

4.1 Podsekcja

ziut

5 DeepQ

The purpose of this report is to outline the implementation of Deep Q-learning (DQL) used to play the game Snake. DQL is a reinforcement learning technique that combines Q-learning with deep neural networks that approximates the action-value function.

5.1 Implementation

The implementation includes a DeepQAgent which encapsulates the deep-q netrwork, the replay buffer(with past experiences), the epsilon policy and the training loop. The QNetwork class constructs the neural network that is used to approximate the action-value function. For each input layer and ouptut layer there is a hidden layer. DeepQAgentConfig is a class with configuration parameters for the used agent. DeepQSnake is a class that inherits from the Snake class and is used to play the game.

5.2 Initialization

The agent constructs the deep Q-network and initializes the replay buffer with the initial epsilon value, which is used to balance exploration and exploitation.

6 Training

The training consists of a loop that runs for a specified number of epochs. Each epoch includees a sequence of game steps and training steps. In each steps the agent makes a decision based on the epsilon, performs this decision and observes the next state, which leads to the potential reward. The agent then stores this experience in the replay buffer and samples a batch of experiences from the replay buffer. The agent then trains the network on the batch of experiences. The epsilon is decreased at the end of each epoch.

6.1 Bellman Equation

Bellman Equation is used to update the Q-values, which determine the expected return for state-action pairs. The Q-values are updated by the following formula:

$$Q(s, a) = r + \gamma \max_{a'} Q(s', a')$$

where:

- $Q(s, a)$ represents the expected return of taking action a in state s and following a fixed policy thereafter.
- r is the reward for taking action a in state s.
- γ is the discount factor which determines the present value of future rewards.
- s' is the next state after taking action a in state s.
- a' is the action taken in the next state s'.

However in this implementation the Q-values are updated by the following formula:

$$q_{\text{target}} = r + \gamma \max_{a'} Q(s', a')$$

The implementation takes advantage of the deep Q-network to approximate the Q-values. This allows the algorithm to work in large action spaces where calculating the maximum Q-value is computationally expensive.

Besides that, during the training the goal of agent's actions is to minimalize the mean square error between the Q-values calculated using the Bellman Equation and the predicted Q-values.

6.1.1 Neural Network

Neural network used to train is described by this model:

$$\begin{aligned} & InputLayer(state\_size) \\ & DenseLayer(hidden\_sizes[0], relu) \\ Model = \ & ... \\ & DenseLayer(hidden\_sizes[n], relu) \\ & OutputLayer(action\_size, linear) \end{aligned}$$

Here, $state\_size$ is the size of the state representation, $hidden\_sizes$ is a list defining the number of neurons in each hidden layer, $action\_size$ is the number of possible actions, and n is the index of the last hidden layer.

$$Layer(neurons, activationFunction)$$

Each $InputLayer$ and $DenseLayer$ is fully connected to the next layer. The $relu$ (Rectified Linear Unit) activation function is used for all layers except for the output layer, which uses a linear activation function.

7 Supervised Classification

We have now added a title, author and date to our first LaTeX document!

7.1 Podsekcja

ziut

8 Summary

This project intends to compare 4 different AI implementations which can be used when making bots that play games

8.1 Podsekcja

ziut DAWAJ TEKST