Automatic game playing with DL

Abdullah Al Forkan
Electronic Engineering
Hochschule Hamm-Lippstadt
Lippstadt, Germany
Abdullah-al.forkan@stud.hshl.de

***Abstract*—Automatic game playing with deep learning represents a dynamic and transformative intersection of artificial intelligence and gaming. This field harnesses the power of deep neural networks, often in the form of convolutional and recurrent networks, to train intelligent agents capable of mastering a wide range of games autonomously. The core paradigm employed in this endeavor is reinforcement learning, wherein these agents learn to maximize rewards by making informed decisions within a game environment.**

I. Introduction

Artificial intelligence is the biggest innovation happened in the recent decade, which changed the way of living in the world by introducing an artificial brain implemented in any system. Machine learning is a subset of AI that learns from data using algorithms and Deep learning is a subset of Machine learning that uses large amounts of data showing exceptional performance in such as image and speech recognition, game development, Automation, Natural Language Processing (NLP), etc. In this article, we are mainly focussing on deep learning game development. Though the first innovation started in the 1950s with the 'Ferranti Mark 1' machine that learned to play Checkers and Chess, the proper deep learning implementation came to us with DeepMind's Atari 2600 games developed by Deep Q-Network in 2013. It shocked us when DeepMind's AlphaGo beat the world champion Go player in 2016, showing the power of deep neural networks in strategy games after that deep learning became unstoppable defeating professional players in various video games such as Dota-2 in 2018. There are various reasons developers started using deep learning algorithms to develop games. Designers could not always anticipate possible situations in the game environment, agents and constantly changing gameplay. It is also very difficult to program a realistic gaming experience and every possible requirement for user satisfaction. So, deep learning came to play its role in deploying the neural network which works like our brain cells. It opened the gate to create realistic and immersive game environments through procedural content generation. Developing non-player character behavior became easier and more advanced making them more intelligent in learning player interactions evolving. Natural Language Processing of deep learning is currently used everywhere to create more realistic in-game dialogues and player interactions with virtual guides. Through convolutional neural networks (CNNs) and recurrent neural networks (RNNs), gesture control and object detection can be configured inside a game. Applying deep learning in simulator games has opened another direction for engineering in quality assurance. Like games, real-world machines can be trained and tested in any environment without any delay or risk. Hence, bug detection and identification of issues are now done more efficiently. Manual content creation has come to an ease with the help of Generative Adversarial Networks (GANs) which generate textures, assets, and even the whole game levels. Real-world physics is a difficult development part of the game and deep neural network makes it easy to create object physics such as water flow, wind, realistic fire, and different types of object behaviors. There is always some maths working behind the strategic games like chess, go, tic-tac-toe, etc. Deep learning can handle that math logic.

II. Deep Learning algorithm for games

A. *Convolutional Neural Networks (CNNs)*

Convolutional Neural Networks (CNNs) are a type of deep neural network architecture designed for processing and analyzing structured grid data, such as images and video. They have proven highly effective in tasks like image classification, object detection, and image recognition.

B. *Long Short Term Memory Networks (LSTMs)*

Long Short-Term Memory (LSTM) networks are a type of recurrent neural network (RNN) architecture designed to overcome the limitations of traditional RNNs in capturing long-range dependencies in sequential data. LSTMs are particularly effective in tasks involving time series prediction, natural language processing, and other sequence-related problems.

C. *Recurrent Neural Networks (RNNs)*

Recurrent Neural Network is a category of advanced artificial neural networks. A directed graph with a temporal sequence is what we get when there is a connection between the nodes. Temporal dynamic behavior is exhibited in RNN. Using internal states, RNNs can process inconsistent range sequences of inputs. They are powerful enough because having a distributed hidden state makes them capable of storing huge amounts of information regarding the past and non-linear dynamics, enabling them to revise their hidden state in complex ways

D. Generative Adversarial Networks (GANs)

Generative adversarial network (GAN), a type of deep learning architecture, was invented in 2014 by Ian Goodfellow [36]. Here, two neural networks challenge themselves in a game (initially given a training set). The theory of GAN is to train images that can generate new images that look accurate to human eyes. GANs were originally modeled for unsupervised learning, but they are broadly used in reinforcement learning, semi-supervised learning, and fully supervised learning.

E. Radial Basis Function Networks (RBFNs)

RBFNs are special types of feedforward neural networks that use radial basis functions as activation functions. They have an input layer, a hidden layer, and an output layer and are mostly used for classification, regression, and time-series prediction.

F. Multilayer Perceptrons (MLPs)

MLPs belong to the class of feedforward neural networks with multiple layers of perceptrons that have activation functions. MLPs consist of an input layer and an output layer that is fully connected. They have the same number of input and output layers but may have multiple hidden layers and can be used to build speech recognition, image recognition, and machine-translation software.

G. Self Organizing Maps (SOMs)

SOMs enable data visualization to reduce the dimensions of data through self-organizing artificial neural networks. Data visualization attempts to solve the problem that humans cannot easily visualize high-dimensional data. SOMs are created to help users understand this high-dimensional information.

H. Deep Belief Networks (DBNs)

A deep belief network, a type of deep learning architecture, is a probabilistic generative graphical model. They are constructed from several layers of stochastic, latent variables with binary values called hidden layers or feature identifiers.

I. Restricted Boltzmann Machines(RBMs)

Developed by Geoffrey Hinton, RBMs are stochastic neural networks that can learn from a probability distribution over a set of inputs. This deep learning algorithm is used for dimensionality reduction, classification, regression, collaborative filtering, feature learning, and topic modeling. RBMs constitute the building blocks of DBNs.

J. Autoencoders

Autoencoder is a distinctive category of artificial neural network utilized in learning efficient data coding through unsupervised techniques. Autoencoders target to learn an encoding for a set of given data and to reduce dimensions in data by training the given network to ignore “noise.” They can probably encode a provided input to represent smaller dimensions, being a data-compression model. Later, decoders can then reconstruct the input back from the encoded version.

K. Deep Reinforcement Learning

Deep reinforcement learning is a combination of reinforcement learning and Q-learning where deep neural networks are implemented. Reinforcement learning is a machine learning algorithm where an agent makes sequential decisions by interacting with an environment and receiving rewards or penalties as feedback. The feedback is calculated based on its actions and goals to follow an algorithm that increases the overall reward over time. Q-learning is a model-free reinforcement learning algorithm to learn the value of an action in a particular state and handle problems with stochastic transitions getting rewards without requiring adaptation. Deep reinforcement learning is in between supervised and unsupervised learning as it is not quite dependent on large amounts of data. This algorithm learns by solving multi-layer problems by trial and error training itself by taking the sequence of decisions and receiving rewards or penalties towards the goal of maximum reward.

III. Deep learning techniques

A. Dropout

B. Rectified linear unit

C. Stochastic gradient descent

D. Batch normalization

E. Minimax theorem

F. Autoencoders

G. Nash equilibrium

H. Self Organizing Maps (SOMs)

IV. My Game

A. Game

B. Players

C. Strategies

D. Payoffs

E. Best response

F. Shapley function

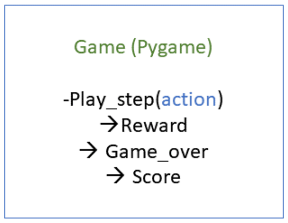
V. Application of Deep Learning in My Game

As we know already, RL is an area of machine learning concerned with how software agents ought to take actions in an environment to maximize the notion of cumulative reward. Or RL is teaching a software agent how to behave in an environment by telling it how good it’s doing. In our game, we are going to use Q-value-based Deep Learning which is also called Deep Q Learning that extends reinforcement learning by using a neural network to predict the actions.

A. Implementing the environment

We have developed a snake game using Python which is a basic human controllable version. We used Pygame which is a cross-platform set of Python modules designed for writing video games. It includes computer graphics and sound libraries designed to be used with the Python programming language. Our game environment consists of three main play step functions which are reward, Game over, and score which are considered as the agent's actions as shown in Figure 1.

[]



```
Game (Pygame)

-Play_step(action)
→Reward
→ Game_over
→ Score
```

Fig. 1. Game environment

1) Reward, Game over, Score: The reward is implemented as if the snake eats the food, it gets +10 points. If there is a collision happens either crashing the wall or eating itself, it's game over and gets -10 points. Otherwise, the reward is 0. Following this instruction, the gameplay step is created and the score shows all the points collected in an attempt after eating food.

2) Actions: We have 4 actions that our snake can perform which are up, down, left, and right. But we design 3 actions. Because of two directions up and down which results in a 180-degree turn, we can immediately die. So, the better decision is to design three actions as shown below.

- Straight – [1, 0, 0]
- Right turn – [0, 1, 0]
- Left turn – [0, 0, 1]

Here, [1, 0, 0] means the current direction which means if we go right, we stay right. If we go left, we stay left. So, if we are going right, by taking another right [0, 1, 0], we go down, and by taking a left [0, 0, 1] turn we can go up.

3) States: We have 11 states in total.

- Danger zones = [danger straight, danger right, danger left]
- Direction states = [left, right, up, down]
- Food direction = [left, right, up down]

These states are represented as the binary locations that tells our agent where to go.

4) Scenarios: First scenario (Figure-2): []

According to the binary state representation,

- Danger zones = [danger straight, danger right, danger left] = [0, 0, 0]
- Direction states = [left, right, up, down] = [0, 1, 0, 0]
- Food direction = [left, right, up, down] = [0, 1, 0, 0]

As our snake is ahead of the food and going straight there is still no danger. Thus all the danger states are 0.

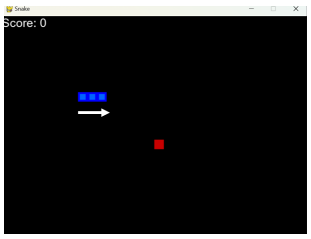


Fig. 2. First Scenario

The direction state right is set to 1, as we have to go right after a certain distance.

Now our food is located on the right side of the snake, so the food right state is true.

Second scenario(Figure-3): []

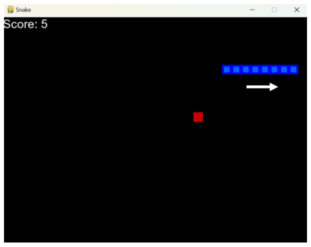


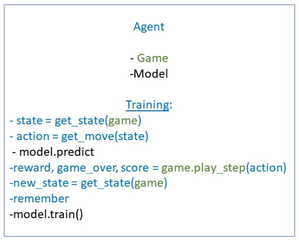
Fig. 3. Second Scenario

- Danger zones = [danger straight, danger right, danger left] = [1, 0, 0]
- Direction states = [left, right, up, down] = [0, 1, 0, 1]
- Food direction = [left, right, up, down] = [0, 1, 0, 1]

Here our danger straight is true. Our direction will be right and to get the food we have to go down and right.

B. Implimenting the agent

[]



```
Agent

- Game
-Model

Training:
- state = get_state(game)
- action = get_move(state)
 - model.predict
-reward, game_over, score = game.play_step(action)
-new_state = get_state(game)
-remember
-model.train()
```

Fig. 4. Agent

C. Implimenting the model

[]

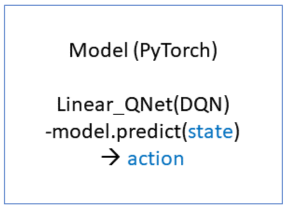


Fig. 5. Model

VI. Training and testing

A. Figures and Tables

B. challenges

C. Results

Conclusions

References

[1]. [2]. [3]
[4]

References

[1] Silver, D., Schrittwieser, J., Simonyan, K., et al. (2017). ”Mastering Chess and Shogi by Self-Play with a General Reinforcement Learning Algorithm.” Nature, 551(7676), 354-359.

[2] Silver, D., Huang, A., Maddison, C. J., et al. (2016). ”Mastering the Game of Go with Deep Neural Networks and Tree Search.” Nature, 529(7587), 484-489.

[3] Mnih, V., Kavukcuoglu, K., Silver, D., et al. (2015). ”Human-level Control through Deep Reinforcement Learning.” Nature, 518(7540), 529-533.

[4] Schulman, J., Wolski, F., Dhariwal, P., et al. (2017). ”Proximal Policy Optimization Algorithms.” arXiv preprint arXiv:1707.06347.