Neural Networks Training for Weapon Selection in First-Person Shooter Games

Stelios Petrakis and Anastasios Tefas

Department of Informatics, Aristotle University of Thessaloniki,
Box 451, 54124, Greece
tefas@aiia.csd.auth.gr

Abstract. First person shooters is probably the most well known genre of the whole gaming industry. Bots in those games must think and act fast in order to be competitive and fun to play with. Key part of the action in a first person shooter is the choice of the right weapon according to the situation. In this paper, we propose a weapon selection technique in order to produce competent agents in the first person shooter game Unreal Tournament 2004 utilizing the Pogamut 2 GameBots library. We propose the use of feedforward neural networks trained with back-propagation for weapon selection and we show that there is a significant increase at the performance of a bot. Moreover, we investigate how the performance of the proposed bot is furthermore improved when trained against more difficult enemies.

Key words: Computational intelligence and Games, Artificial Intelligence and games, First person shooter games, bots, computational intelligence, neural networks

1 Introduction

Computer games have advanced a lot during the course of previous years. The main goal of computer games is simple: fun to play with. This goal is achieved through realism in the environments, rich interactivity, clever opponents and allies and interesting scenarios for the player to get immersed.

Although in previous years, visual realism was nearly the only field most of game developers were interested in, it has later become profound that realism can be achieved not only through life-like graphics and effects, but also through human-like artificial intelligence (AI). Artificial intelligence in computer games is infused into non-playable characters (NPCs), giving the human player the illusion of a clever life-form that he can interact with.

The majority of computer games nowadays, contain some basic AI functionality which makes use of scripted behaviors and finite state machines (FSMs), suffering by this way from a lot of basic gameplay issues, as human players can predict NPC behaviors and easily exploit them.

The need of evolution in bot intelligence for computer games has become clear, so the research is now also focused in computational intelligence meth-

ods, which can produce more reliable and, often, faster results than traditional artificial intelligence techniques.

Computational Intelligence and computer games is a new addition in the field of Artificial Intelligence in general. Despite its youth, there is a huge interest for the advancements on this field, not only from a scientific point of view but also from the economical aspect, as games are a multi-billion dollar industry. Several conferences and competitions are being held throughout the year all around the world, most of them with large prizes for the winners of the Turing tests [11].

With the increase in computational power for consoles and personal computers, industry is finally at the point where there is a lot of horsepower left over for AI. This allows programmers to perform more computationally intensive AI oriented tasks such as collision checks to discover more informations about the environment, decision making or even train neural networks in real time.

Another tough challenge for game AI is to make a bot feel like human, by representing more "human" behaviors, which can often be random and imperfect. Computational intelligence is trying to fill the gap for that need with mimics, memetics, neural networks, fuzzy logic and evolutionary algorithms which can be adjusted and make bots look "human".

In first person shooters, more like any other genres, the fast paced and face to face gaming style, require even more human behavior, especially when the opponents are designed to look like humans, thus fooling the human player to expect them to act with human intelligence.

2 Problem

For any given game moment, the main problem of training of a game agent in a first person shooter is the ability to respond to the sensory data, with ways that make him act like a human player.

This problem can be easily modeled with a back propagation neural network, as neural networks are systems that can be trained in order to learn a certain behavior.

As input vector for those neural networks, we use the sensory data from Pogamut Gamebots API, which practically involves everything an agent can see, hear or sense in any other way (taking damage for example). Input vectors are normalized in $(0, 1)$ space, in order to form a uniform data set and are picked carefully judging from their relevance to the problem each neural network is trying to solve.

The output vector for our neural networks is the action vector, as it will eventually dictate how the bot should act for a given input. Usually action vectors consist of a single value in $(0, 1)$ space.

There are several parts where a neural network can decide how an agent will act. In theory, every single decision a player can make inside the game, can be handled by a separate neural network. From aiming to deciding whether to stay or not in a battle, there are plenty of situations which can be studied separately and then work cooperatively to produce a sophisticated result.

Neural networks can be used to train an agent how to aim with a gun based on the distance and the relative speed, which weapon to select based on the situation, which weapon ammo to pick next, to decide whether to flee or not from a battle in need of health packs and so on.

For weapon selection, the bot could be trained separately for each weapon, having three input values, the angle and distance of the two players (bot and enemy) as well as the velocity of the enemy. The output value of those neural networks (one for each weapon), would calculate the estimated damage that the weapon might do. So, each time, the agent would pick the weapon that would make the maximum damage and fire at the enemy. You can find more about this approach in the following section.

3 Related Work

Using Unreal Tournament 2004 and Gamebots API as a testbed, Ronan Le Hy, et al. [2] were able to apply Bayesian programming to behaviors of bots. They applied probabilistic behavior recognition using a Bayesian model, showing that this method can lead to condensed and easier formalization of finite state machine-like behavior selection, and lend itself to learning by imitation, in a fully transparent way for the player.

Spronck, et al. [5], used dynamic scripting to enable Non-Player Characters (NPCs) to evolve according to different players. NPCs created in different time had different rules and could improve the satisfactory level of human players as their opponents demonstrate a dynamic behavior.

Cook, et al. [6] used graph-based relational learning algorithm to extract patterns from human player graphs and applied those patterns to agents, showing by this way how human knowledge could be transferred.

Kim [1] defined a finite state transition machine to switch behaviors of the agent based on context sensitive stimulations received from the game.

Di Wang, et al. [4], participated in the Bot Prize 2008, using Pogamut library to create bots, implementing FALCON technique, a self-organizing neural network that performed reinforcement learning. Bots were able to learn in real-time without any human intervention.

Hirono et al. [3] developed a bot in Gamebots and Unreal Tournament, winning the second place in Bot Prize 2008, behaving based on Finite State Automaton having two states: the item-collection state (the initial state) and the battle state. Their bot was able to fool some of the judges into believing it was a real person, using heuristic methods.

Unlike the above works, we used neural networks with inputs that affect the score of each weapon, to train bots offline and then apply the trained neural networks in real time situations in order to measure the improvement on their performance.

4 Platform

4.1 Unreal Tournament 2004

Epic Games is one of the top companies in first person shooter games with its 'Unreal Tournament' series. Unreal Tournament is the online spinoff of the original Unreal game series, a fast paced first person shooter. UT (Unreal Tournament) series consist of four games with the two last 2004 and 3 to be amongst the best of the genre. Unreal Tournament 2004 has been probably the most successful game of the series, leveraging the title with new gameplay mechanics and stunning graphics.

Unreal Tournament 2004 is primarily a multiplayer game with single player mode of the game exists as a training base for the multiplayer part. Gamers can train themselves against bots with multiple difficulty levels, in order to be competitive against human opponents. Those bots are using Unrealscript with predefined scripted events in order to achieve victory. They are neither learning from their mistakes nor trying to improve their overall performance, contrary to the bot design we are presenting in this article. There are eight levels of bot skills, each one increasingly more difficult than the previous: novice, average, experienced, skilled, adept, masterful, inhuman and godlike.

Moreover, Unreal Tournament 2004 features a variety of different game types such as Deathmatch, Team Deathmatch, Capture the Flag, Domination, Last Man Standing, Assault and Onslaught. We chose to train and test our bot design in Deathmatch game type, due to its straight forward and single objective nature.

Before analyzing the weapon selection approach it would be better if we first present the weapons our agent was able to used and trained with, in the following process.

Unreal Tournament offers players with a wide range of weapons, that demonstrate different abilities. Some weapons are more effective (do more damage) in a closer range, whereas others are effective at large or very large distances, or have ammunition that can spread throughout the 3D space and cause more damage.

We decided to pick nine from the total 17 weapons of the game, because those nine weapons can be found in nearly every single multiplayer level of the game where the rest are more special weapons.

Those guns are the Lightning Gun, the Assault Rifle, the Bio Rifle, the Shock Rifle, the Minigun, the Link Gun, the Flank Cannon, the Rocket Launcher and the Sniper Rifle.

4.2 Pogamut 2 Gamebots

Epic Games has released some tools for Unreal Tournament modding and researchers used it as a testbed for their experiments in artificial intelligence techniques [8]. Gamebots was one of the testbeds created for that particular reason, and was a mod originally developed at the University of Southern California's Information Sciences Institute by Andrew N. Marshal and Gal Kaminka [9]. Pogamut [7] is a direct derivation of Marshal and Kaminka version, ported to

Unreal Tournament 2004 game and it uses the Gamebot message API as an intermediary between itself and the game. It must be noted that UTBots [1] framework was also developed using Gamebots API, but we chose Pogamut for its mature architecture, its tools and the fast learning curve using Java language.

Pogamut is a really mature platform for game bot development which uses Netbeans IDE and Java programming language in order to control, debug and run the experiments. Creators have access to various parts of the game state in every game tick. Those parts involve sensory data from the agent through vision and sound, world state data, memory and some in-agent information such as health, score, weapons and inventory. Researchers using Pogamut library have full control of the game server (speed, players, number of frags and so on) as well as the whole output of the bots inside the game. The can pause the game in realtime and watch every detail of the world, thus having the opportunity to fine tune every single detail.

For our experiments we are using the second version of Pogamut, Pogamut 2. Pogamut 2 has four components: Gamebots, Server, Client, and IDE. Gamebots, as stated before are managing the information from Unreal Tournament 2004 and the execution of commands. Server is running the deathmatch type of game in Unreal Tournament 2004 (described below) and for testing purposes is running in the same machine as the client. Client is handling the connection to the server, parsing messages and providing libraries of atomic actions and basic sensors. The Integrated Development Environment (IDE) used for Pogamut 2 is Netbeans along with the appropriate plug-in which contains features such as server controller and log viewers.

5 Weapon selection

In our approach, we used neural networks for the weapon selection method where we chose to train nine different neural networks, using the back propagation training method.

In order to decide which weapon to pick, our agent must be able to predict the approximate damage every weapon could do based on the situation. We based this assumption on the fact that every player in an online fps game gets to choose the appropriate weapon from his inventory, according to the situation his enemy is in, in relation to him. For example, in case he is far away from the enemy, human player will prefer a weapon with higher ranged damage, or if the enemy is running he might prefer a weapon that can spread the damage. For this reason we created nine neural networks NN^n, one for each weapon n with $n = 1...9$. Each neural network NN^n has an input layer $\boldsymbol{x^n}$ of 3 neurons, where $\boldsymbol{x^n} = (x_1^n, x_2^n, x_3^n)$, with $x_i^n \in [0, 1]$ and x_1^n representing the distance between the two players, x_2^n the angle between them and x_3^n the velocity of the enemy, one hidden layer with 50 neurons and an output layer with one neuron y^n representing the estimated damage of the weapon n. Input values are shown in figure 1.

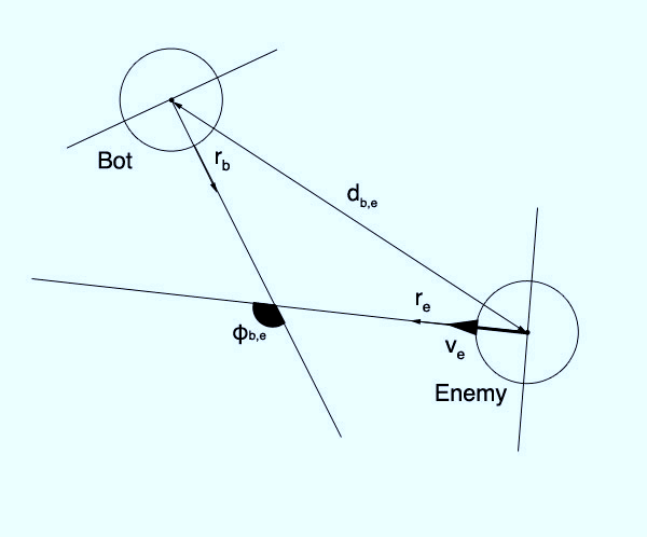

Fig. 1. Diagram of input valus, $d_{b,e}$ is the dinstance, $\phi_{b,e}$ the angle between bot rotation (r_b) and enemy (r_e) and v_e is the velocity of the enemy.

5.1 Neural Network Initialization

During our experiments, we used neural networks with a specific structure. Hecht-Nielsen [10] proved that one hidden layer is sufficient to approximate a bound continuous function to an arbitrary accuracy. However, one hidden layer may result in an excessive number of neurons used in the hidden layer. That is why we chose to model our neural network with one hidden layer having 50 neurons.

As an activation function for the neural network we used the hyperbolic tanget $\varphi(\nu) = \tanh(\frac{\nu}{2}) = \frac{1-\exp(-\nu)}{1+\exp(-\nu)}$ and the learning rate η of the neural network was set to 0.5.

After the initial training of the neural network, which is explained in the training section, we iteratively trained our neural networks with the same data for 1000 times, in order to minimize the mean squared error, keeping at the same time the iterations number to a moderate value in order to prevent overlearning and thus having an overfitted curve. This phase is explained in the offline training section.

5.2 Training phase

Our agent was trained by implementing a scripted bot whose solely behavior was to shoot at the enemy every time he was visible. By using Pogamut functionality, we made our bot invulnerable throughout the training phase, providing it with unlimited ammo, in order to collect data fast and for many different inputs. We

were also able to speed up the training by forcing Pogamut server to run ten times faster than its original speed.

For each encounter, our bot was facing one Unreal Tournament pre-made bot, created by the game developers of Epic Games using Unrealscript which was handling its scripted behavior. This bot, that will be known as '*enemy*' from now on, was configured for two different tests, in the first (novice) and the seventh (inhuman) level of difficulty.

Training phase collected approximately 5000 input states for each weapon, which were used to initially train 9 different neural networks. Those neural networks were initialized randomly and every time the bot was damaging the enemy, we used: the resulting damage normalized in $[0, 1]$ space by diving it to 100 as the desired output z of the neural network and the variables: distance x_1, velocity x_2 and angle x_3 of that particular moment, as input variables.

As we said before those input parameters were calculated relatively to the position of our bot. More specifically, distance was the calculated euclidian distance between the position of our bot in the map (3D space coordinates) and the position of the enemy. This distance parameter was normalized in the range $[0, 1]$ space using a maximum value of 1500 game units, which was picked up empirically. Following the same way of thinking, we used the relative angle between our bot and the enemy by calculating the angle in radians between the vector our bot was facing and the one the enemy was facing, normalized by the maximum value of 2 radians and finally we calculated the normalized velocity by applying a maximum velocity value of 500 game units.

We chose those three values as input of our neural networks, by considering which world variables can affect the weapon damage in a particular moment. The distance of two players plays a key part in a fight, as well where each player is facing. Finally, speed does matters in a fast paced game, as players who stand still are always easy targets, and players who run a lot cannot always aim right.

Using the back propagation method we were able to teach neural networks that when they were facing those inputs, the desired output was the damaged caused. All inputs and the output were normalized at $[0, 1]$. We collected approximately 30.000 states for each weapon, having for each state four values: *distance*, *velocity*, *angle* and *reward* which was the damage done to the enemy player normalized as already described.

5.3 Offline training phase

After the data collection process was complete for every weapon, we collected two files for each experiment: One with all the input and output values of the training set and one with the weights of the first generation neural networks.

Having nine pairs of files, we used some offline training to further train the neural networks for 1000 epochs. After the offline learning process, the neural networks were ready for testing.

As a first test, we passed one last time the same data to the mature neural networks, comparing their output to the real damage caused each time. The results shown that neural networks were able to fit the original data, with each

weapon having its own behavior in different values of distance, velocity and angle.

5.4 Testing phase

During the testing phase, we initiated those nine trained neural networks with the structure exported from the offline training and we integrated them into our bot decision making process. Our bot had a pretty simple instruction set: In idle mode it is running around the map collecting any health, weapon and armor power up but when it sees an enemy who is close by, it calculates which weapon of his inventory could do the maximum damage, according to the input values of that particular moment. This time, our bot is neither invulnerable nor has all the weapons in his disposal, so it must decide among the weapons of his inventory which are a subset of the nine weapons we presented before.

Let N_l^k+ be the number of frags of a bot trained versus an enemy of level k, when it faces an enemy level l in the simulated environment. We can draw the performance curve of such a bot, based on the ratio of the number of frag of this bot to the number of frags of the enemy: N_l^k-, $P_l^k = \frac{N_l^k+}{N_l^k-}$. In this way, when $P_l^k > 1$, bot performs better than the enemy with level l.

We tested our bot performance against Unreal Tournament bots with different level of difficulty each time.

Having trained our bot not only with Unreal Tournament enemies of $k = 1$ (easy skill), but also with $k = 7$ (godlike skill) we were able to see some major performance improvement.

When our bot was trained with an enemy of level $k = 7$, it can perform better against enemies of maximum level $l = 2$, whereas when it is trained with an enemy of level $k = 1$ it can only perform well against enemies of same level. It must be noted that just by applying the weapon selection method in a bot which does nothing else other than randomly moving through the map, we observed an immediate major improvement of the bot score, compared with a random weapon selection bot.

In the graph at Figure 2, the dotted line represents the frag ratios P_l^R of a random bot (no neural network attached) versus different levels of the enemy ($l = 1...7$), the dashed line represents the frag ratios P_l^1 of our neural network bot trained with an enemy of level $k = 1$, where as the solid line represents the frag ratios P_l^7 of our neural network bot trained with an enemy of level $k = 7$.

We can clearly see that when the bot is trained with a much more difficult enemy ($k = 7$), it is not only possible to outnumber an enemy with level $l = 2$, but can also improve its score with the level $l = 1$ enemies.

So, in conclusion a bot trained to face more difficult enemies, can perform better in battle from a bot that has been trained to face easier enemies. So the measure of performance of such a bot versus enemies of level l is the frag ratio $P_l^k = \frac{N_l^k+}{N_l^k-}$.

As for what weapons our bot ends up using, we plotted the weapon n usage percentages P_n of our bot in a single match (trained versus an enemy level $k = 7$,

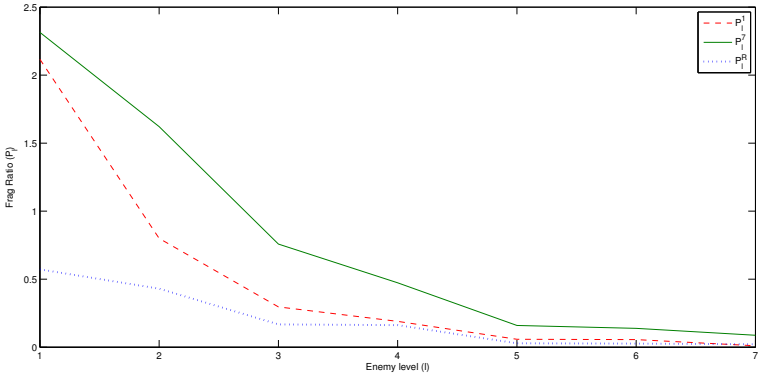

Fig. 2. Frag ratio P_l for a random bot P_l^R / a bot trained with an enemy level $k = 1$ P_l^1 / a bot trained with an enemy level $k = 7$ P_l^7 for $l = 1...7$

when facing an enemy level $l = 1$), where $P_n = \frac{m_n}{N}$ for weapons $n = 1...9$, with N total uses and m_n the uses of weapon n during the battle.

As we can see, our bot prefers mostly weapons like Assault Rifle ($n = 2$) and Bio Rifle ($n = 3$), uses also weapons like Minigun ($n = 5$) and Link Gun ($n = 6$) as well as Sniper Rifle ($n = 9$) and doesn't prefer weapons like Lightning Gun ($n = 1$), Shock Rifle ($n = 4$), Flank Cannon ($n = 7$) and Rocket Launcher ($n = 8$).

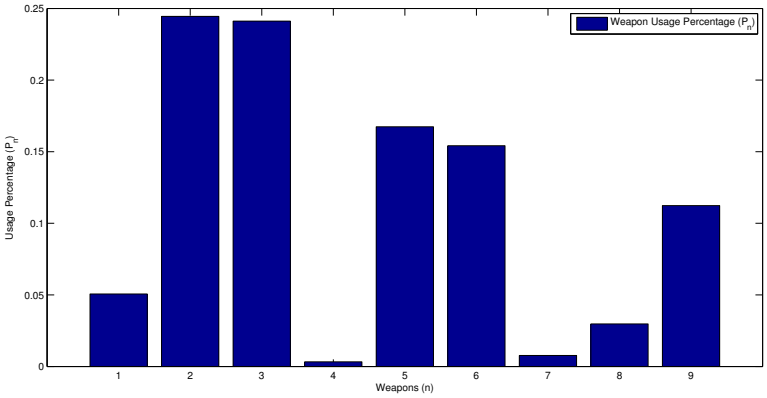

Fig. 3. Weapon usage percentages P_n for weapons n of a bot trained with an enemy level $k = 7$ when facing an enemy level $l = 1$

6 Conclusions

In this paper we proposed a computational intelligence method using neural networks, in order to develop a bot which can be trained while playing Unreal Tournament 2004 using Pogamut 2 Gamebots library, trying by this way to achieve higher performance than the average dummy bot. Our methods involved a weapon selection module, where the bot was trained to choose the weapon to fire depending on the condition it was in. We have shown that by training the bot for the weapon selection module using single-hidden layer feedforward neural networks, led to a significant increase in the performance of the bot. We were also able to observe the increase of bot's performance while it was facing even more difficult enemies.

References

1. In-Cheol Kim, UTBot: A Virtual Agent Platform for Teaching Agent System Design, Journal of Multimedia, pages 48-53 (2007)
2. Le Hy R., A. Arrigioni, P. Bessiere and O. Lebeltel, Teaching Bayesian behaviours to video game characters, Robotics and Autonomous Systems 47, pages 177-185 (2004)
3. Daichi Hirono and Ruck Thawonmas, Implementation of a Human-Like Bot in a First Person Shooter: Second Place Bot at BotPrize 2008, Proc. Asia Simulation Conference 2009 (JSST 2009) (2009)
4. Wang D., Subagdja B., Tan A. and Ng. G, Creating Human-like Autonomous Players in Real-time First Person Shooter Computer Games, Proc. IAAAI -09 (2009)
5. Spronck P.; Ponsen M.; Sprinkhuizen-Kuyper I.; and Postma E, Adaptive Game AI with Dynamic Scripting, Machine Learning, pages 217–248 (2006)
6. Cook D. J.; Holder L. B.; and Youngblood, Graph-Based Analysis of Human Transfer Learning Using a Game Testbed, IEEE Transaction on Knowledge and Data Engineering 19, pages 1465-1478 (2007)
7. Kadlec R.; Gemrot J.; Burkert O.; Bida M.; Havlicek Brom C., POGAMUT 2 - A platform for Fast Development of Virtual Agents Proceedings of CGAMES 07, La Rochelle, France (2007)
8. M. Dawes and R. Hall, Towards using first-person shooter computer games as an artificial intelligence testbed, Proceedings of the 9th International Conference on Knowledge-Based Intelligent Information and Engineering Systems (KES '05), pages 276-282 (2005)
9. Rogelio Adobbati and Andrew N. Marshall and Andrew Scholer and Sheila Tejada, Gamebots: A 3D Virtual World Test-Bed For Multi-Agent Research, In Proceedings of the Second International Workshop on Infrastructure for Agents, MAS, and Scalable MAS (2001)
10. Hecht-Nielsen, R., Theory of backpropagation neural network, Proc of the Int. Conf of Neural Networks, I, pages 593-611 (1989)
11. Hingston, P., Computational Intelligence and AI in Games, IEEE Transactions on, A Turing Test for Computer Game Bots, pages 169-186 (2009)