Neuroevolution Strategies for Word Embedding Adaptation in Text Adventure Games

Vivan Raaj Rajalingam
School of Computer Science
and Electrical Engineering
University of Essex
Colchester, Essex, UK
Email: vivraaj@gmail.com

Spyridon Samothrakis
Institute of Analytics and Data Science
University of Essex
Colchester, Essex, UK
Email: ssamot@essex.ac.uk

***Abstract*—Word embeddings have gained popularity in many Natural Language Processing (NLP) tasks. They can encode general semantic relationship between words, and hence provide benefits in many downstream tasks when used as a knowledge base. However, they still suffer from morphological ambiguity as the trained vectors do not share representations at the sub-word level. In this report, we propose a new architecture that uses neuroevolution to fine-tune pre-trained word embeddings for the challenging task of playing text-based games. We fit this architecture into an existing game agent and evaluate its performance on six text-based games. Experimental results show that the proposed fine-tuning architecture may significantly mitigate the effect of morphological ambiguity to enable our game agent to reduce the total number of steps required to generate valid actions and perform well in these games.**

***Index Terms*—neuroevolution, word embedding, reinforcement learning**

I. Introduction

Recently, word embeddings have been explored as a language model to learn word representations. One particularly famous embedding is word2vec which was introduced in [1]. When trained over a large text corpus, word2vec has been proven to reveal certain aspects of similarity using distributed representations. This feature enables it to be successfully adopted in different domains for various NLP tasks such as name entity recognition, sentiment analysis, and word analogy detection.

In this paper, we will focus on the aspect of natural language understanding in the domain of text-based games. Text-based games use natural language to describe the state of the world, accept actions from the player and to generate new states based on the validity of the player's action. In the nomenclature of Reinforcement Learning (RL), this results in the large potentially infinite space for action commands which represents a challenge to existing optimization algorithms such as reinforcement learning. Beyond that, the other properties of the game such as partial observability, stochastic environment, sparse rewards etc. make the games unpredictable and challenging for a game agent.

Word embeddings have been used to build RL agents. For example, in the IEEE The Text-Based Adventure AI 2016 Competition, Fulda et al. [2] built a game playing agent that leveraged word2vec as a common sense knowledge base to guide its agent's exploration on a suite of text-based games. They pre-trained the word2vec model using Wikipedia dataset and found that it was able to encode common sense knowledge within the dataset. This unique feature enabled their agent to effectively use affordance extraction methods to query the embedding to generate valid action commands.

In the same paper, Fulda et al. [2] found that the pre-trained word embeddings suffered from several limitations such as polysemy, neologisms, and semantic extensions. This limitation may have been caused by the biased training corpus which contains imbalance occurrences of individual words. As a result, the embeddings were not able to detect contextual information between these rare words and other words in the corpus. This limitation affected the quality of the affordance extraction which resulted in Fulda et al. [2]'s agent taking many irrelevant or inferior actions that prolong the agent's learning curve.

In this report, we propose a new game agent which is referred to as Neuroagent that is built upon Fulda et al. [2]'s work but introduces a new component which uses neuroevolution to fine-tune the pre-trained word embeddings. Neuroevolution refers to the generation of a function approximator (their weights and/or topology) using an evolutionary algorithm [3]. It can receive inputs of any form and evolve the function approximator's architecture to optimize for any goal. The fine-tuning process functions concurrently by interaction with the game. This concept is also referred to as interaction-based language learning [4]. We will investigate if the addition of the fine-tuning component would recover better word representations in the embeddings that results in more efficient affordance extraction. Then, we will evaluate the performance of our game agent on a suite of text-based games.

The rest of the paper is organized as follows: we first give a overview of related work in Section II and background theory in Section III. In Section IV, we describe the final agent architecture. Section V lists the experimental set-up and the results, and we discuss the findings of the experiments in Section VI. Finally, we list our conclusion and findings in Section VII.

978-1-7281-1884-0/19/$31.00

II. Related Work

In this section, we will describe the related work in the development of artificial intelligence agents that can play text-based games, application of neuroevolution techniques in games and development of knowledge base using word embeddings to handle affordances.

Text-Based Games - In text-based games, the game agent needs to understand the game's state description so that it can generate logical responses while learning how to play the game. As the current state of the art in language understanding is still not able to match human-level performance, other researchers have navigated past this limitation by developing new architectures. For example, Branavan et al.[6] used the game manual to map instructions to these sequences of actions and applied reinforcement learning to learn the game's optimal control policy. The results which were obtained demonstrated that agents with prior knowledge of language representations could navigate a game better compared to agents without this prior knowledge.

In 2015, He et al. [7] introduced the Deep Reinforcement Relevance Network (DRRN) architecture to play two choice-based text-based games. He et al. [7] feed all the words that appear in the game into a bag of words (BOW) model which is passed as an input to a neural network [8]. Even-Dar et al. [9] designed a separate embedding for states and actions which are then combined with Deep Q-Network (DQN) to approximate the Q-function. Finally, He et al. [7] chooses the action with the highest Q-value. Narasimhan et al. [10] developed the LSTM-DQN architecture which is a combination of LSTM and DQN and evaluated its performance on parser-based text-based games. Narasimhan et al. [10] used the LSTM layer to encode the game agent's observation as a sequence of words. The resulting state observation is then used by two subnetworks to predict the Q-values for all verbs and object words independently. The Q-values for the composed actions are then obtained from the mean of both resulting values.

In contrast to the other works which introduced different architectures to learn the control policies of the game, Fulda et al. [2] focused on using affordances as the action elimination method. Fulda et al. [2] used pre-trained word embeddings which are trained using word2vec on Wikipedia dataset to act as a knowledge base. Then, Fulda et al. [2] presented a method for affordance extraction via the inner product of the knowledge base for action selection. This method proved to be successful as it achieved the highest score in the IEEE The Text-Based Adventure AI 2016 Competition.

Neuroevolution - is the combination of neural networks and evolution strategies. Fogel et al. [5] evaluated the performance of evolutionary algorithms on checkers which is a strategy based game by first letting it play games against itself. The algorithm learnt to associate the specific sections of the chess-board over long periods of training resulting in it achieving good performance.

Word Embeddings - Gao et al. [11] most recently proposed to use an autoencoder together with word embeddings to handle the low representation of the syntactic properties and linguistic regularities in the word embedding vectors. The low representation is caused by sparsity which occurs as words in English have different meanings thus, these words can be widely spread over the embedding's vector space. The resulting training showed that the compressed word embedding in the hidden layer of the autoencoder was able to recover the synonym relation between similar word pairs. Their results highlighted the potential of deep neural networks in adjusting word vectors in word embedding's and further inspires our work.

III. Background

This section will describe the background theory of the techniques and tools which was applied in our game agent.

A. Text-Based Games

Text-based games are the type of games that use text characters instead of graphics [12]. The player must type commands in free-form english text to control the game, and the resulting game state is generated via text output. Text-based games offer a perfect environment for artificial intelligence research due to its unique properties such as:

Partial Observability - In text-based games, the states which are generated by the game engines are partially observable. A state may give incomplete information about the current state of the game. Fig. 1 shows an example of a scenario of partial observability in the game 'detective'. The state descriptions in the left box describe the state of the game which is shown to the player while the game engine does not show the hidden state to the player. If the agent inputs an appropriate action command such as 'get all', the game will transition to a different state that describes the effect of the taken action [13].

Infinite Action Spaces - The biggest challenge in text-based games is the huge action space. As the action has to consist of natural language descriptions, there are many combinations of words that can be trialled as an action. For example, given a state description of 'There is a door in front of you. What do you do now?', the right action would be to 'open the door'. However, the game agent has no common-sense knowledge of a word meaning in a given context and thus may try actions such as 'eat door' or 'run door' which do not make sense. By constantly trying a different combination of words, the agent will spend much time being stuck in a particular game state until it may or may not reach the desired action command.

Intrinsic Motivation and Memory - A game agent must not only aim to complete the challenges as it must also explore the game's map to search for paths that will lead to higher game rewards. This problem in RL is known as the balance of exploration and exploitation [14]. The game agent will need to have high 'curiosity' so that it can explore the game efficiently.

B. Word Embeddings

The Skip-Gram model [15] is a neural network architecture that is built upon the word embedding's structure to predict the word in the window from the current word.

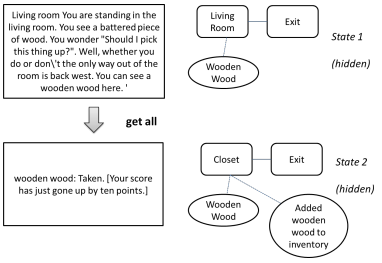

Fig. 1. Example of partial observability in the game 'detective'

Word2vec Analogy Function - Mikolov et al. [15]'s technique for solving word analogies is illustrated in equation 1. Given two words, w_1 and w_2, the similarity measure is given by the angular distance of both words which is calculated by the cosine angle of their respective vectors. Thus, the relationship between both words can be discovered as the offset of their vector embedding [16]. This method results in higher similarity if the angle between the word vectors decreases and lower similarity if the angle increases.

$$sim(w_1, w_2) = cos(w_1, w_2) = \frac{(w_1.w_2)}{||w_1||||w_2||} \tag{1}$$

This method was also successful in assessing whether there is a linguistic relation between the base and gerund form of verbs such as 'debug' and 'debugging' [17] which indicated that these examples of base-to-gerund types are encoded as a particular linear offset in the embedding.

C. Separable Natural Evolution Strategies

Neuroevolution is a combination of a function approximator such as neural networks and natural evolution algorithms. Text-based games can be perceived as a black-box problem due to their partial observability properties. A black-box problem refers to an environment where there is no additional information on the objective function to be optimized besides evaluating the fitness performance at certain points in the parameter space. Thus, the optimization problem is too complex to be modelled directly.

However, natural evolution strategies have been shown to solve these problems with good results [18]. Natural evolution strategies (NES) are a black-box optimization framework that uses search gradients to update the parameters of the search distribution. The shortcomings of NES are its limited applicability to high dimensional search spaces [19].

To overcome this shortcoming, Schaul et al. [20] introduced the Separable Natural Evolution Strategies (SNES). This modified algorithm is structured to compensate for some invariance properties of the NES algorithm and to limit the search to diagonal covariance matrices. It functions by restricting the class of search distributions to Gaussians with diagonal covariance matrix which corresponds to restricting a general class of multi-variate search distributions to separable distributions as described in equation 2 where $\widetilde{p}$ is a family of densities on the reals and the parameters of all these distributions are given by $\theta = (\theta_1, ...\theta_d)$.

$$p(x|\theta) = \prod_{k=1}^{d} \widetilde{p}(x_k|\theta_k) \tag{2}$$

These parameters amount to $\theta_k = (\mu_k, \sigma_k)$ where $\mu_k \in R$ is a position and $\sigma_k \in R^+$is a scale parameter such that $z_k = \mu_k + \sigma_k.s_k \sim \widetilde{p}(.|\mu_k, \sigma_k)$ for $s_k \sim \tilde{p}(.|0, 1)$. The modified functions enable the strategy update step to take only linear time, $O(d)$ which makes it computationally efficient and reduced number of parameters in the covariance matrix. Furthermore, the algorithm is able to maintain a reasonable amount of flexibility in the search distributions and faster adaption of its parameters.

D. Neural Networks

The goal of a neural network is to approximate some function, g. In the context of a classifier, $y = g(x)$ maps an input x to a category y and the network defines the mapping $y = g(x|\theta)$ and learns the value of the parameters θ that results in the best function approximation.

Fig. 2 illustrates a basic structure of Multilayer Perceptron (MLP) which is a neural network with hidden layers. In the leftmost layer, the network takes in a set of inputs, x_m where m is the number of inputs. These set of inputs are then mapped to hidden nodes in the next layer. Each input is mapped to every nodes in the hidden layer. The connection between the individual input nodes, hidden nodes and output nodes are termed as weights and these weight values are not fixed but changeable.

All the inputs are multiplied by the weights and summed for each hidden node. Next, the same process repeats for the connections from the hidden layer to the output layer. An activation function controls the amplitude of the output.

IV. Methodology

A. Partially Observable Markov Decision Process

We formulate the objective of learning the control policy of the game as a Partially Observable Markov Decision Process (POMDP) problem. Formally, POMDP is defined by a 7-tuple $< S, T, A, \Omega, R, O, \gamma >$ with the additional parameters; Ω that represents the set of observation, O which represents the set of conditional observation probabilities and γ which is the discount factor. These addition of new information related to past observations now enables the agent to make optimal decisions while playing the game.

Reward Function (R) : Each game has its unique scoring mechanism which enables the agents to obtain points based on their progress in the game. For each action that it takes, the agent receives a reward, $r_t = R(s_t, a_t)$ from the game engine. However, in most text-based games, there are rare instances when the player receives points as they are usually stuck in

a particular state due to invalid action commands. Thus, this score is not useful to our game agent as it will not be able to assess its progress in the game.

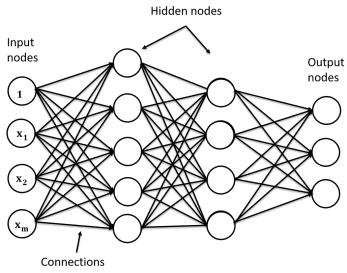

Fig. 2. Structure of a Multilayer Perceptron

Instead of just using the game's scoring mechanism, we introduce a new reward factor called agent reward, rg_t. The agent's reward is calculated based of the game's reward, r_t and predefined values which are determined depending if there are changes in the current state text description compared to the previous state text description. This reward is only visible to the game agent to help improve its decision-making process and thus the overall goal of the agent is to now maximize the discounted sum of rewards received which is given by $E[\sum_t \gamma^t rg_{,t}]$.

B. Neuroagent Architecture

Fig. 3 shows the architecture of our game agent which is referred to as Neuroagent. Neuroagent will generate action commands based on the state text description and reward which are generated by the game engine. We have narrowed down the agent's architecture to three main components as follows.

Learning Algorithm - A game agent must be able to learn the game's control policy by using the rewards which are obtained through continuous interaction. As the game environment is unknown to the agent, the agent does not know how the environment will change in response to its action which is denoted as T and its corresponding reward, R. This type of learning problem is categorized as Model-Free Learning.

Our agent uses Q-learning [21], which is an off-policy algorithm that directly estimates the optimal Q-values of each action in each state based on equation 3 where α is the learning rate, γ is the discount factor and s' is the new state based on action, a.

$$\triangle Q(s,a) = Q(s,a) + \alpha(R(s,a) + \gamma(max_a Q(s',a) - Q(s,a))) \tag{3}$$

This algorithm allows the policy to be updated at every state and the optimal policy, π^* be derived by choosing the action with the highest Q-value in the current state as shown in equation 4.

$$\pi^*(s) = argmax_a Q^*(s,a) \tag{4}$$

[2] As most of the games are generally deterministic with a high percentage of consumable rewards, we set α to 1 to limit the Q-value updates so now the Q-learning algorithm is modified as shown in equation 5.

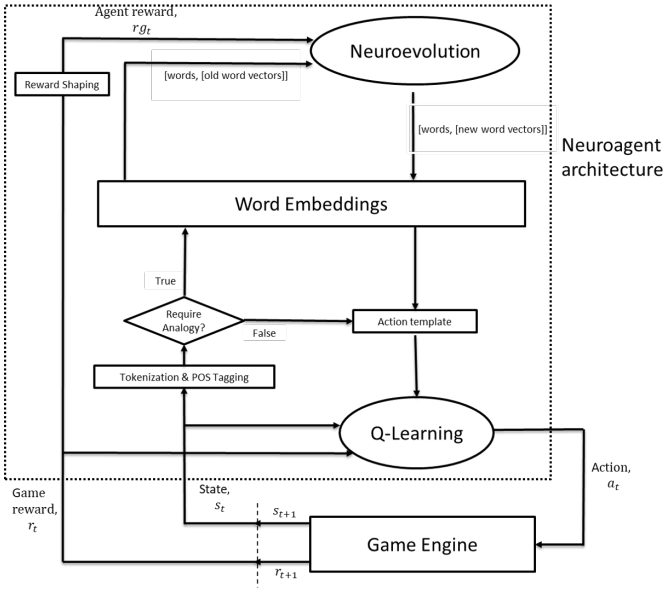

Fig. 3. Neuroagent architecture

$$Q^{'}(s,a) = max(Q(s,a), Q(s,a) \triangle Q(s,a)) \tag{5}$$

Word Embeddings - Our approach is based on Fulda et al. [2]'s work which used pre-trained word embedding as a knowledge base. Fulda et al. [2] used the English Wikipedia dataset dump as a knowledge base. The reader is referred to [2] for further information on the method of which the knowledge base is designed and action template for action selection. These words that are considered relevant are stored into an object list. The agent will utilize each word in the object list and generate an action based on the action template which is described in equation 6. The actions are a combination of verb/object pairs where V is the set of all English-language verbs and O is the set of English-language nouns.

$$a = v + \text{" "} + o, v \in V, o \in O \tag{6}$$

Fulda et al. [2] utilizes word2vec's built-in analogy method to fully leverage the semantic structure which is encoded in the knowledge base to find appropriate verb for a given noun. Fulda et al. [2] defined a set of canonical vectors which illustrates the relationships that are intended to be extracted from the knowledge base. As an example, she used a combination of $vector['wear'] - vector['coat']$ and $vector['attack'] - vector['enemy']$ as exemplars rather than as descriptors. She describes these relationship as verb/noun pair $(\overrightarrow{v}, \overrightarrow{n})$ which act as affordant to the extent that $(\overrightarrow{n} + \overrightarrow{a} = \overrightarrow{v})$ where $\overrightarrow{a}$ is the affordant vector which is desired. To satisfy these equation, the knowledge base will return the n closest verbs $(\overrightarrow{v_{c1}}, ..., \overrightarrow{v_{cn}})$ to the point $\overrightarrow{n} + \overrightarrow{a}$.

Neuroevolution - Fig. 4 illustrates the architecture of the neuroevolution component which is composed of a MLP as

the function approximator to the SNES evolutionary algorithm. The FNN is made of one hidden layer with 100 neurons.We chose 100 neurons for the hidden layer as it represents the number of dimensions in the word vectors. [21] We then used tanh as the activation function in the hidden layer so that the output values range between [-1, 1]. There are 10,100 weights in this MLP architecture. The weights of the neural network and the corresponding agent reward rg_t are passed as input to the SNES algorithm.

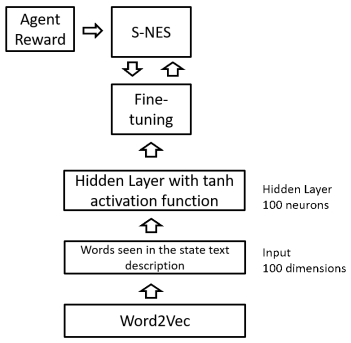

Fig. 4. The neuroevolution structure which is implemented in our game agent

The process of fine-tuning the word embeddings are as follows; First, we will track each word in the game's state text description and then perform pre-processing methods such as lower-casing all the words and concatenating them with '_' and their corresponding POS tag.

Next, we will retrieve their index position and their corresponding vector of 300 dimensions from the word2vec model [22]. We convert the list that contains the word vectors into an array so that it can be passed as an input to the MLP. Beyond that, we will obtain the agent reward, rg_t as a result of the game reward which is transformed using the reward shaping formula. The weights of the neural network and the agent reward, rg_t for the corresponding state are then passed as inputs to the SNES algorithm.

The SNES algorithm will evolve the weights of the neural network structure by using the agent reward's values as feedback. These new weights are then adapted back to the MLP's structure. Then, the original array that contains the previous word vectors will be passed through the MLP in a feedforward manner. The resulting output is the new word vectors. The analogy function will then query the word2vec to extract the new query list based on the new position of the word in the word2vec vector space.

V. Results and Experiments

Our experiments have two main objectives which are to build an effective game agent and investigate if the addition of the neuroevolution component influenced our game agent's performance.

A. Textplayer Framework

We evaluated the agent on six text-based games using the Textplayer framework. These games are part of Multi-User Dungeon (MUD) genre and can be accessed from the Interactive Fiction Archive (IF) library.

Textplayer[1] is a python interface to run the games which are compatible with Infocom's Z-machine using Frotz. It also provides our baseline agent, on top of which Neuragent is built. All the games in the framework differ by their properties. For example, games such as 'Zork1' represent fictional world situations whereas 'Detective' represent business scenarios.

B. Parameters

Table 1 describes the final parameters for the Neuroagent which were tuned through trial and error. The MLP was built using Keras's[2] Dense layer. The SNES algorithm codes were adapted with permission from Samothrakis et al. [23].

TABLE I
THE FINAL PARAMETERS FOR THE NEUROAGENT

Layer	Parameter	Values
Multilayer Perceptron (MLP)	No. of Layers	One Layer with 100 neurons
	Activation Function	tanh
	Number of weights to be evolved	10,100
SNES	No. of Population	50
	No. of Generations	500

As the Neuroagent is built upon the work done by Fulda et al. [2], we will use Fulda et al. [2]'s agent as a baseline. We will run the baseline agent over 25,000 iterations to ensure uniformity among the measured results. Then, we will group the baseline scores in increasing increments of 50 iterations to represent one generation which would result in 500 generations.

C. Comparison of Results

Table 2 shows the comparison of the final scores which were achieved by the Neuroagent and baseline at the end of 500 generations over five runs when evaluated on the six text-based games in the textplayer framework.

TABLE II
COMPARISON OF THE FINAL SCORES WHICH ARE ACHIEVED BY NEUROAGENT AND THE BASELINE

Game	Maximum score of the game	Final score of the agents	
		Baseline	NeuroAgent
detective	360	0	**70**
omniquest	50	5	**10**
deephome	300	1	1
zork1	350	10	10
Balances	51	5	**10**
Advent	350	36	36

The bold values in the table indicate the agent that obtained the highest score for each game. The results show that the Neuroagent outperformed the baseline in three out of the six games. There are three games in which both agents achieved similar scores.

[1]https://github.com/danielricks/textplayer
[2]https://github.com/keras-team/keras

Besides using the score as a benchmark for performance, we plot the learning trajectories [23] of the agents in Fig. 5. These figures were generated using seaborn's lineplot which aggregates the different measurement runs at each generation by plotting the mean and the 95% confidence interval around the mean.

We can observe that the Neuroagent reaches convergence faster compared to the baseline in the games of 'detective' and 'Balances'. In the game 'detective', it also exhibited a faster learning curve as it continues to achieve higher reward as the game progresses. This is in contrast to the baseline's performance which decreases as it gets killed early on during the gameplay. In the game 'Balances', the baseline's performance stagnates from the generation no. 150 onwards.

The Neuroagent's performance in the game 'omniquest' is peculiar as it has a more 'bumpy' learning curve compared to the baseline which reaches its highest score early on during the game. However, the Neuroagent eventually outperformed the baseline after generation no. 350 onwards whereas the baseline was stuck in the same score for the rest of the game.

In the games 'deephome' and 'Advent', both agents achieved a similar score, and their learning behaviour are almost identical. The only notable difference is that the Neuroagent gets killed in half of the runs towards the end of the 'deephome' game in contrast to the baseline that managed to stay alive for the rest of the game.

We can focus specifically on the learning behaviour of the agents based on the game reward that is received during gameplay. Fig. 6 illustrates the game milestones where the agents either receive a positive or negative reward as the game progresses. The peaks in the graph represent these milestones. The general observation is that both agents receive zero rewards for the majority of the game and there are only a few instances when the agent receives a reward. Besides that, in most of the games, the Neuroagent have more instances of game milestones compared to the baseline which indicates that the Neuroagent can take more relevant actions that warrant a response from the game interpreter.

VI. Discussion

Overall, the results show that Neuroagent is capable of exhibiting a faster learning curve to either outperform or match the performance of the baseline in every game. It could also obtain game rewards faster than the baseline and produce a more stable policy as the game progresses [24]. The results also suggest that Neuroagent is a more efficient agent compared to the baseline. In this section, we will discuss the effects that are caused by the implementation of the neuroevolution component in the Neuroagent. We will analyze the results which are obtained from the game 'detective' as points for discussion.

Analysis of Neuroagent's performance in the game 'detective' - In the game 'detective', the Neuroagent achieves a higher score compared to the baseline. We will use the term 'milestones' to define the game states in which the agent collects a nonzero reward. The leftmost diagram in Fig. 6 describe the milestones in the game 'detective' for both agents.

Although both agents use the same pretrained word embeddings as a knowledge base, they generate different actions. This observation is caused by the neuroevolution component of the Neuroagent.

The neuroevolution algorithm guides the direction of the word vectors shifts in the word2vec vector space by using the agent reward as feedback. The leftmost diagram in Fig. 7 shows the progress of the agent reward throughout the game. If a particular action does not result in any reward, the reward shaping function will transform the agent reward to a negative value, and this value will then be passed as an input to the neuroevolution algorithm. As most of the action which are taken in the game eventually results in zero game reward, thus the neuroevolution algorithm will continually shift the word vectors around the word2vec vector space. The agent's reward act as an intrinsic reward which guides the agent's exploration of the search space.

We can further demonstrate the word vectors shift by using the example of words that were seen throughout the 'detective' game which can be queried using word2vec's analogy function. We visualize the projection of these words vector representation at the start of the game onto a two-dimensional space using t-SNE algorithm [25] as shown in Fig. 8 (a). The left most diagram in Fig. 8 (a) represents these word's original vector position in the pre-trained word embedding vector space. The next diagram then shows the shift in the position of these words at generation no. 400. Finally, we can again observe that the word vectors shifts again at the end of the game at the rightmost diagram. Furthermore, all the word vectors tend to be more compact and better clustered to each other as the game progresses [26].

By constantly shifting the word vector position around in the vector space, word2vec's analogy function will generate a different query list. The nearest neighbour for this word is more spread across several parts of speech during the gameplay [27].

The neuroevolution algorithm continually tunes the word embeddings so that it would improve the affordances extraction to ensure that the agent avoids getting stuck at suboptimal positions [21]. One of the limitations of the baseline agent was that it generates many invalid actions. This situation can be attributed to the fixed position of the word vectors in the word2vec vector space thus the analogy function will always generate the same query list throughout the game. The drawbacks of a fixed vector position mean that the agent is not able to trial new verb and noun pair combinations even though the agent may have discovered that the entire list of verbs in the existing query list does not result in any reward.

Limitations of the Neuroagent - As the Neuroagent is composed of evolution learning algorithms, the training time increases as the state text descriptions in games grew. Besides that, the analysis of the embeddings which are learnt by the agent in the 'Omniquest' game are illustrated in Fig. 8 (b) shows that there is no defined cluster which was obtained at

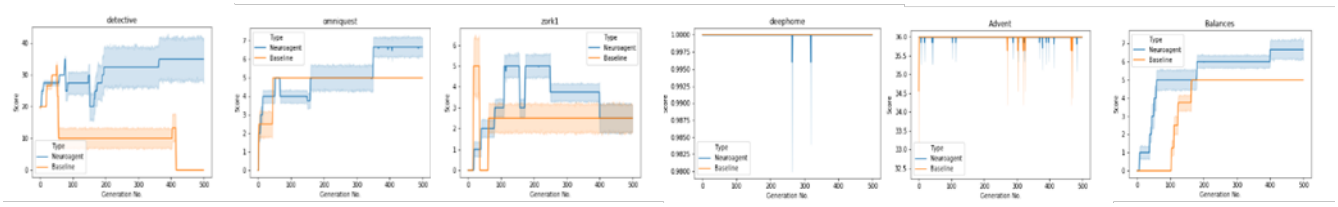

Fig. 5. The progression of the learning trajectories of the agents during 500 generations

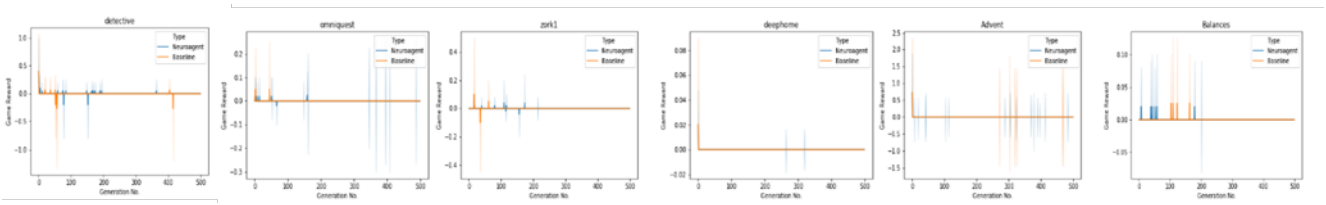

Fig. 6. The milestones in the game where the agents collect a nonzero reward

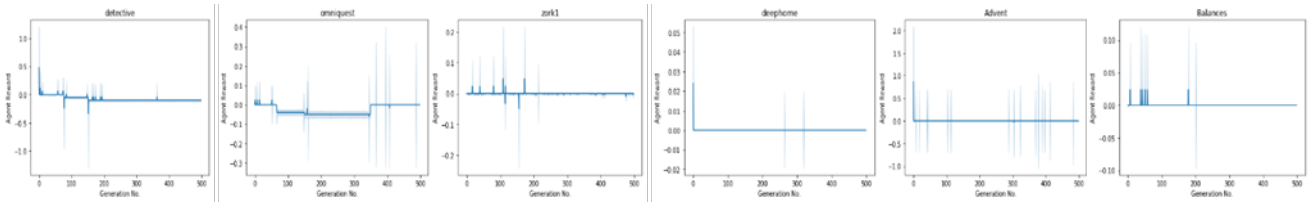

Fig. 7. The progression of the agent reward which is passed as input to the neuroevolution component of the Neuroagent over 500 generations

the end of the game. Thus, the learning behaviour might have been random.

VII. Conclusion and Future Work

In this report, we have proposed a new architecture to fine-tune word embeddings so that an Artificial Intelligence (AI) agent can enable language-based interaction learning in any domain. We evaluated this architecture on the domain of text-based games by modifying the architecture of last year's Text-Based Adventure AI competition winner to incorporate the new component which uses neuroevolution algorithm to perform the fine-tuning tasks. We found that the addition of the neuroevolution algorithm enabled our game agent which is referred as Neuroagent to outperform the baseline in three out of the six games which were tested. Based on these results, we also found that the neuroevolution component managed to compress the game's word vector which resulted in better verb-noun pair combinations as action selection [11] and enabled the agent to improve its learning curve with more effective exploration as the game progresses.

Although the results are promising, it is still inconclusive to claim that the Neuroagent is a more efficient agent compared to the baseline due to the limited number of games in which the agents were tested. We were unable to test the agents on more games as we encountered problems such as memory error and not being able to obtain any reward in other games within the textplayer framework. However, we can deduce that the neuroevolution algorithm can guide the exploration of word embeddings vector space by using game rewards as feedback so that the game agent does not get stuck in suboptimal solutions.

There are multiple future work directions that we would like to point out. First, we could experiment with different function approximators in the neuroevolution algorithm. We can use more advanced activation functions such as Exponential Linear Units (ELU) which could help to speed up the neural network's learning curve as they bring gradient closer to the unit natural gradient which fits well with the S-NES algorithm. Besides that, we can compare the performance of the game agents by replacing the neuroevolution component with a standard neural network layer with backpropagation - this however would require chancing he training regime of the agent. Furthermore, we can evaluate the generalization ability of the fine-tuned word embedding by first training the word embedding on one game and then testing its performance on other unseen games. The results of this approach can serve as a guide to further research in zero-shot learning ability of neuroevolution based language learning methods.

References

[1] Mikolov, T., Chen, K., Corrado, G. and Dean, J., 2013. Efficient estimation of word representations in vector space. *arXiv preprint arXiv:1301.3781.*

[2] Fulda, N., Ricks, D., Murdoch, B. and Wingate, D., 2017. What can you do with a rock? Affordance extraction via word embeddings. arXiv preprint arXiv:1703.03429.

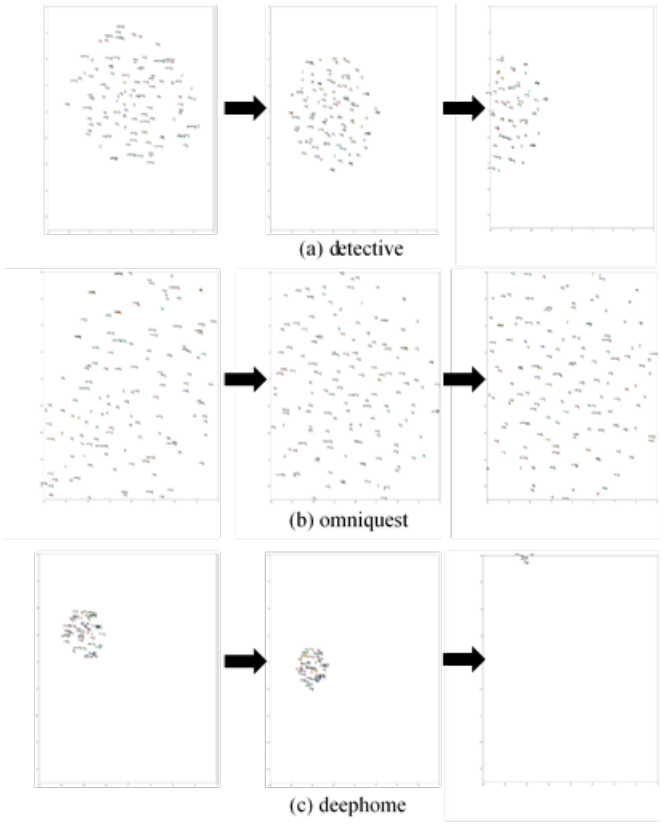

Fig. 8. t-SNE visualization of the entire words seen in the games as the game progresses

[3] Risi, S. and Togelius, J., 2017. Neuroevolution in games: State of the art and open challenges. IEEE Transactions on Computational Intelligence and AI in Games, 9(1), pp.25-41.
[4] Ansari, G.A., Chandar, S. and Ravindran, B., 2018. Language Expansion In Text-Based Games. arXiv preprint arXiv:1805.07274.
[5] Fogel, D.B., Hays, T.J., Hahn, S.L. and Quon, J., 2004. A self-learning evolutionary chess program. Proceedings of the IEEE, 92(12), pp.1947-1954.
[6] Branavan, S.R., Chen, H., Zettlemoyer, L.S. and Barzilay, R., 2009, August. Reinforcement learning for mapping instructions to actions. In Proceedings of the Joint Conference of the 47th Annual Meeting of the ACL and the 4th International Joint Conference on Natural Language Processing of the AFNLP: Volume 1-Volume 1 (pp. 82-90). Association for Computational Linguistics.
[7] He, J., Chen, J., He, X., Gao, J., Li, L., Deng, L. and Ostendorf, M., 2015. Deep reinforcement learning with a natural language action space. arXiv preprint arXiv:1511.04636.
[8] Côté, Marc-Alexandre, et al. "TextWorld: A Learning Environment for Text-based Games." arXiv preprint arXiv:1806.11532 (2018).
[9] Even-Dar, E. and Mansour, Y., 2003. Learning rates for Q-learning. Journal of Machine Learning Research, 5(Dec), pp.1-25.
[10] Narasimhan, K., Kulkarni, T. and Barzilay, R., 2015. Language understanding for text-based games using deep reinforcement learning. arXiv preprint arXiv:1506.08941.
[11] Gao, X. and Ichise, R., 2017. Adjusting Word Embeddings by Deep Neural Networks. In ICAART (2) (pp. 398-406).
[12] Wikipedia. (2018). Text-based game. [online] Available at: https://en.wikipedia.org/wiki/Text-based_game [Accessed 28 Jun. 2018].
[13] Narasimhan, K.R., 2017. Grounding natural language with autonomous interaction (Doctoral dissertation, Massachusetts Institute of Technology).
[14] Aytar, Y., Pfaff, T., Budden, D., Paine, T.L., Wang, Z. and de Freitas, N., 2018. Playing hard exploration games by watching YouTube. arXiv preprint arXiv:1805.11592.
[15] Mikolov, T., Sutskever, I., Chen, K., Corrado, G.S. and Dean, J., 2013. Distributed representations of words and phrases and their compositionality. In Advances in neural information processing systems (pp. 3111-3119).
[16] Gladkova, Anna, Aleksandr Drozd, and Satoshi Matsuoka. "Analogy-based detection of morphological and semantic relations with word embeddings: what works and what doesn't." Proceedings of the NAACL Student Research Workshop. 2016.
[17] Linzen, T., 2016. Issues in evaluating semantic spaces using word analogies. arXiv preprint arXiv:1606.07736.
[18] Wierstra, D., Schaul, T., Peters, J. and Schmidhuber, J., 2008, June. Natural evolution strategies. In Evolutionary Computation, 2008. CEC 2008.(IEEE World Congress on Computational Intelligence). IEEE Congress on (pp. 3381-3387). IEEE.
[19] Schaul, T., 2012, July. Benchmarking separable natural evolution strategies on the noiseless and noisy black-box optimization testbeds. In Proceedings of the 14th annual conference companion on Genetic and evolutionary computation (pp. 205-212). ACM.
[20] Schaul, T., Glasmachers, T. and Schmidhuber, J., 2011, July. High dimensions and heavy tails for natural evolution strategies. In Proceedings of the 13th annual conference on Genetic and evolutionary computation (pp. 845-852). ACM.
[21] Cruz, F., Magg, S., Weber, C. and Wermter, S., 2016. Training agents with interactive reinforcement learning and contextual affordances. IEEE Transactions on Cognitive and Developmental Systems, 8(4), pp.271-284.
[22] Feng, W., Zhuo, H.H. and Kambhampati, S., 2018. Extracting Action Sequences from Texts Based on Deep Reinforcement Learning. arXiv preprint arXiv:1803.02632.
[23] Samothrakis, S., Perez-Liebana, D., Lucas, S.M. and Fasli, M., 2015, August. Neuroevolution for general video game playing. In Computational Intelligence and Games (CIG), 2015 IEEE Conference on (pp. 200-207). IEEE.
[24] Haroush, M., Zahavy, T., Mankowitz, D.J. and Mannor, S., 2018. Learning How Not to Act in Text-based Games.
[25] Maaten, L.V.D. and Hinton, G., 2008. Visualizing data using t-SNE. Journal of machine learning research, 9(Nov), pp.2579-2605.
[26] Ansari, G.A., Chandar, S. and Ravindran, B., 2018. Language Expansion In Text-Based Games. arXiv preprint arXiv:1805.07274.
[27] Peters, M.E., Neumann, M., Iyyer, M., Gardner, M., Clark, C., Lee, K. and Zettlemoyer, L., 2018. Deep contextualized word representations. arXiv preprint arXiv:1802.05365.