This assumption is not true for most first-person games, in which both players and agents observe the environment from a limited first-person perspective.

To overcome the problem of partial observability, the agent needs to gain a memory and remember previous states. One approach is to stack the last *k* frames and feed them into the network at the same time [7]. A technique that has been used to handle longer temporal context in time series is to introduce recurrent connections in the network. This was done by [9], who used a Deep Recurrent Q-Network (DRQN) with a Long Short-Term Memory (LSTM) layer to estimate the Q-function and play Atari 2600 games with partial observability. A DQN for navigating and a DRQN for action selection was used by [10] to achieve human-level play in a 3D FPS deathmatch scenario. The Asynchronous Advantage Actor-Critic (A3C) algorithm together with an LSTM was used by [11] to train an agent to navigate in randomly generated 3D maze environments only from raw visual input. A stacked LSTM network with an adaptation of the A3C algorithm was used by [12] to teach an agent to navigate in complex 3D maze environments with dynamic elements.

In the present effort, we explore using a DRQN with an LSTM layer for navigating in 3D environments where single observations can be very similar at different points of the environment if not supported by a memory of previous observations. The agent was implemented and tested in a 3D FPS navigation task with partial observability. The model was tested in the ViZDoom scenario *My Way Home*, using the API developed by [8].

In this paper, the background for DRQN will first be presented. Then the model and implementation of the present approach will be presented. We will conclude with some observations of the agent's behaviour in the ViZDoom scenario.

2. Background

Reinforcement Learning [5] is a Machine Learning technique in which an agent deals with learning a policy for behaving in an environment through trial-and-error interaction with the environment. At each interaction, the agent observes a state *s* from the environment, performs an action *a* according to its policy π, and receives a reward *r* from the environment and observes a new state *s'*. The goal of the agent is to find a policy that maximizes its expected return. *Q(s,a)*

Q-Learning [4] is a model-free off-policy algorithm that estimates the action-value function, the value of action *a* given state *s*, by iteratively updating the Q-values towards the observed reward *r* plus the maximum Q-value of the resulting state *s'*. The tabular Q-Learning update is then:

$$Q(s,a) := Q(s,a) + \alpha(r + \gamma \max_{a'} Q(s',a') - Q(s,a)) \quad (1)$$

where α is the learning rate of the update and γ the discount factor weighting future rewards.

Storing an estimate for each state-action pair is not efficient for domains with large or continuous state spaces, such as FPS games. DQN [6] deals with this problem by using a neural network as a non-linear function approximator parameterized by weights and biases θ. Now the parameters θ are updated instead of the individual Q(s,a)-values. The goal is to minimize the average of the loss:

$$L(\theta_t) = (y - Q(s,a;\theta_t))^2 \quad (2)$$

where *t* is the current time-step and *y* is the update target $y = r + \gamma \max_{a'} Q(s',a';\theta_t)$.

The network parameters are updated by following the gradient of the loss function:

$$\theta_{t+1} = \theta_t + \alpha(y - Q(s,a;\theta_t))\nabla_{\theta_t} Q(s,a;\theta_t) \quad (3)$$

Using a neural network as a function approximator for the Q-values has shown unstable behaviour and might lead to divergence [13]. One step for overcoming this problem is to use experience replay [14] in which the agent stores transitions in a replay memory and then samples them uniformly during training. This breaks correlation between successive samples. Another step is to use a target network, identical in structure to the main network, to estimate the Q-values. The parameters of the target network can either be updated gradually towards the parameters of the main network, or frozen in time and updated only every *ith* iteration. The update target then becomes $y = r + \gamma \max_{a'} Q(s',a';\theta'_t)$ where θ'_t are the biases and weights of the frozen network at timestep *t*. A final step for stabilization is to use an adaptive learning rate method such as RMSProp [15]. These steps were all used by [7] and proved to stabilize training of a DQN.

Reinforcement Learning is often considered as a Markov Decision Process (MDP) in which the agent acts in the environment based on states that hold the Markov property [5]. This assumption does not hold in many tasks. This is especially true in a limited first-person view in a 3D world. In this case, the agent partially observes the environment and the problem is then considered a Partially Observable Markov Decision Process (POMDP). A Deep Recurrent Q-Network (DRQN) was introduced by [9] to deal with the problem of partial observability. They showed that introducing recurrence to the network was better at approximating the actual Q-values based on an observation *o*. It was shown by [10] that a DRQN could be used to play 3D FPS games at a high level by using an LSTM layer. The LSTM is a recurrent neural network that is built on memory cells that are able to process time series with the help of an input, output, and forget gate [16]. LSTMs are especially effective at modeling long term dependencies. This applies in games specifically when information was present in previous frames but not in the current frame.

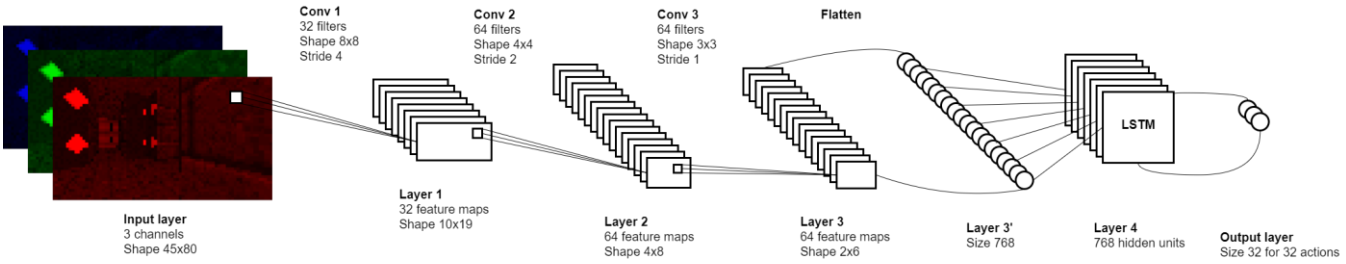

Figure 1. The architecture of the neural network. The network takes a down-sampled RGB image as input and propagates it forward through three convolutional layers and an LSTM layer with 768 hidden units to output 32 action values. Layer 3' is the neurons from layer 3 flattened into one vector of length 768.

3. Experiment

3.1 Model

The model presented in this paper is a DRQN and is based on the DQN model by [7]. The main difference is that the first fully connected layer following the convolutional layers [17] of the DQN model is replaced by an LSTM layer, and the network is only fed one input image at a time, rather than four.

The complete network architecture is shown in Figure 1. From the game, a frame with the original 400x225x3 resolution is downsampled to a 45x80x3 RGB image that serves as the input to the neural network. The input is propagated forward through three hidden convolutional layers, and the third convolutional layer is then flattened and propagated through one LSTM layer before being passed to the output layer in which each unit assigns a Q-value to a different action.

The first convolutional layer has a kernel of size 8x8, a stride of size 4x4, no padding, and 32 feature maps and applies a ReLU [18] activation function. The second convolutional layer has a kernel of size 4x4, a stride of size 2x2, no padding, and 64 feature maps and applies a ReLU activation function. The third convolutional layer has a kernel of size 3x3, a stride of size 1x1, no padding, and 64 feature maps and applies a ReLU activation function. The third convolutional layer is then flattened and fed into an LSTM layer with 768 hidden units. The output of the LSTM layer is finally fed into the output layer, which maps one value to each possible action.

3.2 Training

The agent was interacting with the environment following an ε-greedy policy. With ε probability, pick a random action, otherwise, pick the action with the highest associated Q-value. The ε-greedy policy is popular policy for dealing with the exploration-exploitation trade-off in reinforcement learning [7, 10]. The ε value used in this study was linearly decayed from 1 to 0.1 over 200k actions and then frozen at 0.1.

The agent used a frame-skip technique in which a chosen action was repeated for k frames and, as a result, observations were received and rewards computed every k+1 frames from the environment. The present study used a frame skip of 4 as in [7, 8, 10].

The hidden state of the LSTM was initialized by zero at the beginning of every episode and updated after each selected action by the agent. Transitions by the agent *(s,a,r,s')* were stored in a replay memory. The replay memory stored the last 1 million transitions by the agent.

The parameters of the main network were updated once for every four selected actions. The parameters were updated using the RMSProp [15] optimization algorithm with a learning rate of 0.0025. The update followed the Bootstrapped Random Updates method [9], where a minibatch of size 32 of experiences, each experience consisting of 8 timesteps, were selected uniformly from the experience replay. The target values were computed by the target network. The parameters of the target network were gradually updated towards the parameters of the main network by a factor $\tau = 0.001$ after each network update:

$$\theta'_t = \tau\theta_t + (1-\tau)\theta'_t \quad (4)$$

3.3 Scenario

The model was trained and tested in the ViZDoom environment *My Way Home* [8]. The goal of the agent was to learn to navigate a labyrinth-like environment and find a green vest in one of the rooms. The map was a series of interconnected rooms and one corridor with a dead end. Each room had a different colour. The agent was spawned in a random room facing a random direction and the vest was always in the same room. The agent had five available binary buttons: *turn left, turn right, move forward, move left, move right*. The agent thus had 32 different actions – one for each possible combination of buttons. The agent received a reward of 1 for reaching the vest, and otherwise a reward of -0.0001 for every timestep. Each episode ended after 2100 environment steps or when the agent reached the vest. OpenAI Gym [19] has a wrapper for the *My Way Home*

environment[2] and they define the scenario as solved if the agent reaches an average reward of 0.5 or more over 100 consecutive episodes.

The agent of the present study was trained and evaluated for 200 epochs. Each epoch consisted of 10k training steps. A training step was defined as a step where the agent picked an action. The agent was evaluated for 10 episodes after each ended epoch. The agent followed a greedy policy for testing in which the perceived best action was always chosen. The training and testing was completed in 15 hours on three NVIDIA Titan X Pascal GPUs.

4. Results and discussion

Informal observations in the video of the gameplay[3]. indicate that the agent learned how to find doors and navigate through the corridors. It also indicates that the agent had an implicit goal of finding a corner in the corridor next to the life vest, rather than the explicit goal of finding the life vest. This indicates that the agent had a general idea of where to go to get a good reward, but did not know the exact location of the reward.

A reason for why the agent did not find the vest is likely that the environment is rather complex, and that the agent did not get to explore it enough. Indeed, the exploration rate ε of the present study was decayed very fast compared to [7], leading to the agent not discovering the reward of the life vest enough to learn to go there. The exploration rate was set low to account for the few training steps to promote the agent exploiting its knowledge of the environment for most of training.

A big limitation of the present study was that the agent was trained for very few steps compared to [7, 10, 12]. The video of the agent suggesting that the agent learned to navigate the environment indicates that the agent did improve and may have found a better policy given more training time and possibly complete the goal. It would therefore be interesting to train the agent for longer to see if this assumption is indeed correct.

Another future direction of research would be to evaluate the DRQN model against a standard DQN model used by [10], as well as an A3C model used by [12].

5. Conclusion

In this work, we proposed a Deep Reinforcement Learning model based on a Deep Recurrent Q-Network for teaching an autonomous agent to navigate in a 3D environment from a first-person perspective with partial observability of the environment. Our experiment indicates that the agent might not have been trained long enough to solve the complex challenge, but that it was able to learn how to find doors and pass through corridors. Our work supports literature [7, 12] in end-to-end reinforcement learning, indicating that agents can learn to act in an environment from raw sensory input. We see a promising future for using reinforcement learning to model agent behaviour in commercial games but also acknowledge with our results that there is still some more research to be done within the field.

[2] https://gym.openai.com/envs/DoomMyWayHome-v0
[3] https://youtu.be/GUsnVaL4Y54

Acknowledgements

We would like to thank NVIDIA for donating a Titan X Pascal GPU card which was used in the training.

References

[1] G. N. Yannakakis and J. Togelius, “A Panorama of Artificial and Computational Intelligence in Games,” *IEEE Transactions on Computational Intelligence and AI in Games*, pp. 317-355, 4 December 2015.

[2] B. Auslander, S. Lee-Urban, C. Hogg and H. Munõz-Avila, “Recognizing the Enemy: Combining Reinforcement Learning with Strategy Selection Using Case-Based Reasoning,” *European Conference on Case-Based Reasoning*, pp. 59-73, 2008.

[3] M. Smith, S. Lee-Urban and H. Muñoz-Avila, “RETALIATE: Learning Winning Policies in First-Person Shooter Games,” *Proceedings of the Twenty-Second AAAI Conference on Artificial Intelligence*, pp. 1801-1806, July 2007.

[4] C. J. C. H. Watkins, “Learning from Delayed Rewards,” May 1989. [Online]. Available: http://www.cs.rhul.ac.uk/~chrisw/new_thesis.pdf. [Accessed 20 May 2017].

[5] R. S. Sutton and A. G. Barto, Reinforcement Learning: An Introduction, Cambridge, MA, USA: MIT Press, 1998.

[6] V. Mnih, K. Kavukcuoglu, D. Silver, A. Graves, I. Antonoglou, D. Wierstra and M. A. Riedmiller, “Playing Atari with Deep Reinforcement Learning,” 19 December 2013. [Online]. Available: http://arxiv.org/abs/1312.5602. [Accessed 4 April 2017].

[7] V. Mnih, K. Kavukcuoglu, D. Silver, A. A. Rusu, J. Veness, M. G. Bellemar, A. Graves, M. Riedmiller, A. K. Fidjeland, G. Ostrovski, S. Petersen, C. Beattie, A. Sadik, I. Antonoglou, H. King, D. Kumaran, D. Wierstra, S. Legg and D. Hassabis, “Human-level control through deep reinforcement learning,” *Nature*, pp. 529-533, 26 February 2015.

[8] M. Kempka, M. Wydmuch, G. Runc, J. Toczek and W. Jaśkowski, “ViZDoom: A Doom-based AI Research Platform for Visual Reinforcement Learning,” *Proceedings of IEEE Conference of Computational Intelligence in Games 2016*, pp. 341-348, September 2016.

[9] M. Hausknecht and P. Stone, “Deep Recurrent Q-Learning for Partially Observable MDPs,” 23 July 2015. [Online]. Available: https://arxiv.org/abs/1507.06527. [Accessed 20 May 2017].

[10] G. Lample and D. S. Chaplot, “Playing FPS Games with Deep Reinforcement Learning,” 18 September 2016. [Online]. Available: https://arxiv.org/abs/1609.05521. [Accessed 20 May 2017].

[11] V. Mnih, A. P. Badia, M. Mirza, A. Graves, T. P. Lillicrap, T. Harley, D. Silver and K. Kavukcuoglu, "Asynchronous Methods for Deep Reinforcement Learning," 4 February 2016. [Online]. Available: https://arxiv.org/abs/1602.01783. [Accessed 20 May 2017].

[12] P. Mirowski, R. Pascanu, F. Viola, H. Soyer, A. J. Ballard, A. Banino, M. Denil, R. Goroshin, L. Sifre, K. Kavukcuoglu, D. Kumaran and R. Hadsell, "Learning to Navigate in Complex Environments," 11 November 2016. [Online]. Available: https://arxiv.org/abs/1611.03673. [Accessed 4 June 2017].

[13] J. N. Tsitsiklis and B. V. Roy, "An Analysis of Temporal-Difference Learning with Function Approximation," *IEEE Transactions On Automatic Control*, May 1997.

[14] L.-J. Lin, "Reinforcement Learning for Robots Using Neural Networks," Carnegie Mellon University, Pittsburgh, PA, USA, 1992.

[15] G. Hinton, "Lecture 6e rmsprop: Divide the gradient by a running average of its recent magnitude," 2012. [Online]. Available: http://www.cs.toronto.edu/~tijmen/csc321/slides/lecture_slides_lec6.pdf. [Accessed 4 June 2017].

[16] S. Hochreiter and J. Schmidhuber, "Long Short-Term Memory," *Neural Computing*, pp. 1735-1780, 15 November 1997.

[17] Y. Lecun, L. Bottou, Y. Bengio and P. Haffner, "Gradient-based learning applied to document recognition," *Proceedings of the IEEE*, pp. 2278-2324, November 1998.

[18] A. Krizhevsky, I. Sutskever and G. E. Hinton, "ImageNet Classification with Deep Convolutional Neural Networks," *Advances in Neural Information Processing Systems 25*, 2012.

[19] G. Brockman, V. Cheung, L. Pettersson, J. Schneider, J. Schulman, J. Tang and W. Zaremba, "OpenAI Gym," 5 June 2016. [Online]. Available: https://arxiv.org/abs/1606.01540. [Accessed 1 June 2017].