KU LEUVEN

Behaviourally cloning River Raid agents

Laurens Diels

Thesis submitted for the degree of
Master of Science in Artificial
Intelligence, option Engineering and
Computer Science

Thesis supervisor:
Prof. dr. ir. Johan Driesen

Assessors:
Prof. dr. ir. Luc De Raedt,
Dr. ir. Jeroen Tant

Mentor:
Dr. ir. Hussain Kazmi

Academic year 2020 – 2021

Preface

First and foremost I would like to thank dr. Kazmi for all the help, advice, suggestions, comments and direction he has given throughout this thesis project. Thanks also go out to prof. Driesen for the opportunity of working on this project, all assessors for their time in going through and analysing the dissertation, and any reader in general. Indeed, without you all effort in writing this dissertation would have been pointless.

Secondly I want to thank my friends and family for all the support they have given throughout this academic year.

Finally, I would also like to thank the VSC (Flemish Supercomputer Center) and KU Leuven for allowing me to make use of their servers through introductory credits.

Laurens Diels

Contents

Abstract

In this dissertation we investigate the feasibility and difficulties of behaviourally cloning agents in a simple video game, namely River Raid. We will consider three types of agents: reinforcement learning agents, random agents sampling actions uniformly, and human agents. The behavioural clones' performance will be evaluated on the micro-level through comparison of the state-conditioned and unconditional action distributions via L^1-distance, and on the macro-level by comparing the (cloned) agents' survival time and score per episode. We develop measures for comparing point clouds of such time-score pairs by fitting Gaussian densities with maximum likelihood parameters and comparing these using the KL divergence, and by looking at the p-values of the Kolmogorov-Smirnov two-sample test for equal underlying two-dimensional distributions.

In the case of reinforcement learning agents cloning certainly seems feasible, even without access to absurd amounts of data. We recommend a minimum of 250 training episodes for the clones. But the relation between the number of training episodes and cloning performance is noisy and certainly not monotone.

Dealing with the inherent unpredictability of the random agent poses serious problems at the level of distributions, but they carry over only to a lesser extent to the macro-level. Although also there our measures are clearly able to distinguish between a base agent and its clone, qualitatively they seem similar enough, both simply playing erratically.

Finally these problems are even worse for human players, where additionally we have to deal with limited amounts of training data and the fact that humans improve while playing games. Using our techniques behaviourally cloning human agents decently is not yet possible. We also develop a measure of (early-game) variability of the agents based on the number of different pixels of frames at a fixed time with respect to a medoid frame at that time, and show that the human agents plays the most inconsistent, even more so than the random agent.

List of Figures and Tables

List of Figures

List of Tables

List of Abbreviations and Symbols

Abbreviations

2D	Two-dimensional
AI	Artificial intelligence
ALE	Arcade learning environment
BC	Behavioural clone
Conv	Convolution(al layer)
DF	Down-fire (combination of actions)
DL	Down-left
DLF	Down-left-fire
DQN	Deep Q-network
DR	Down-right
DRF	Down-right-fire
FGKLD	Fitted Gaussian KL divergence
IRL	Inverse reinforcement learning
KL	Kullback-Leibler
KSp	p-value for the (two-dimensional) Kolmogorov-Smirnov two sample test
LF	Left-fire
LZMA	Lempel-Ziv-Markov chain algorithm
MC	Monte Carlo
MDP	Markov decision process
PPO	Proximal policy optimisation
RAM	Random-access memory
RF	Right-fire
RL	Reinforcement learning
TL	Transfer learning
UF	Up-fire
UL	Up-left

ULF	Up-left-fire
UR	Up-right
URF	Up-right-fire
VGG	Visual geometry group (University of Oxford)
VGGQD	VGG16 with only a quarter of the hidden nodes in the classification layers (and added MC dropout layers)

Symbols

Mathematics

$\in$	the elementhood relation
$\subset$	the (not necessarily proper) subset relation
$\{\dots\}$	set defined by the enumeration of its elements
$\{x \in X \mid p(x)\}$	subset of a set X consisting of those elements satisfying property (predicate) p
$\setminus$	set difference
$\times$	Cartesian product of sets (consisting of couples)
$f : X \to Y : x \mapsto f(x)$	a function f with domain X and codomain Y mapping elements $x \in X$ to corresponding elements $y = f(x) \in Y$
$\mathbb{N}$	the set of natural numbers (including 0)
$\mathbb{R}$	the set of real numbers
$[x, y]$	the closed real interval bounded below by x and above by y
$\mathbb{R}^2$	the real plane
$\lvert\cdot\rvert$	the absolute value function
d_1	the L^1-distance function between vectors (or functions with finite domain)
$\lVert\cdot\rVert_2$	the (Euclidian) 2-norm on vectors
$\sum$	summation
$\lim_{t \searrow 0} f(t)$	right-hand limit of a function f (for a positive argument t) approaching zero
max	maximum
min	minimum
argmin	argument of the minimum, the (or an) element achieving the minimum
avg	average (arithmetic mean)
$\log_2$	binary logarithm
$.^T$	transpose of a matrix (or vector)

$(x_n)_{n\geq 1}$	a(n infinite) sequence x indexed by positive natural numbers
$(x_n)_{n=1}^{k}$	a finite sequence x of length $k \in \mathbb{N} \setminus \{0\}$, starting from index 1

Probability theory and statistics

P, Q	probability distributions (measures)
$P(A \mid B)$	conditional probability (according to measure P) of event A given event B
$\mathrm{KL}(P \parallel Q)$	KL divergence of probability distributions P and Q
μ	(population) mean
Σ	(population) covariance matrix
$\widehat{\cdot}$ (e.g. $\widehat{\mu}$)	estimate (e.g. of the mean)
H_0	null hypothesis (to reject) for a hypothesis test
p	p-value of a hypothesis test

Reinforcement learning

$\mathcal{A}$	action space of a Markov decision process
$\mathcal{S}$	state space of a Markov decision process
a	action in the action space
r	reward function of a Markov decision process
s, s'	state in the state space

Miscellaneous

M	million
MB	megabyte
s	score (in River Raid)
t	time (measured in number of frames)

Chapter 1

Introduction

In 2013 researchers at DeepMind wrote a paper titled *Playing atari with deep reinforcement learning* [MKS+13, MKS+15]. As the title suggests, they used reinforcement learning (RL) to train agents to play Atari 2600 games. What was novel about their approach is that the learner was only given access to the pixel data. No semantically important preprocessing occurred. Remarkably the agents they trained using their technique of Deep Q-Networks (DQN) were able to outperform humans in many games.

This sparked a lot of follow-up research where more challenging, more modern games were used. Examples include DeepMind's AlphaStar for playing Starcraft II [VBC+19] and OpenAI's OpenAI Five for Dota 2 [BBC+19]. These systems are able to go head to head with and even defeat the top ranking teams in the respective games.

This has always been the (traditional) goal of reinforcement learning: attaining a superhuman level of performance. For industrial applications this makes sense. Indeed, reinforcement learning can be used for resource allocation (e.g. [BHD13]), scheduling (e.g. [WU05]), assembly tasks (e.g. [IDMM+17]) and energy management (e.g. [KLR+13]), just to name a few. There are clearly benefits to being better at these tasks than humans. This remains true even for more artistic applications such as radio controlled helicopter acrobatics [ACQN07]. Overcoming human limitations such as the occasional crash is obviously a goal worth pursuing.

However, in video games agents having mastered the game to a superhuman level are often undesirable. When playing against bots, players want to have a decent chance of actually winning. Therefore, an alternative goal could be to create models which can play human-like, or even more narrowly, player models which are meant to mimic particular players.

Such player models have many use cases. In [PSM18] we find amongst others

- changing the game environment to adapt to the player's capabilities with the goal of maximising player enjoyment (e.g. by creating personalised challenges),
- (partially) replacing human testers for game balancing,

- detecting cheating and identity fraud (as in these cases the behaviour of the 'player' is very different from normal),
- temporarily substituting players in (cooperative) multiplayer games by player-like bots when the original player's connection drops out,
- playstyle analysis and categorisation (e.g. [HLTY14]).

The first and last points could also be relevant for e-sports. One can imagine creating a personalised training course to improve upon a (professional) player's weaknesses.

Another application domain is opponent bots in multiplayer games. In this case the desired model's actions need not capture a particular individual, but should be indistinguishable from actual human players (passing some kind of Turing test). Previous research has shown that playing against such bots does indeed increase players' engagement compared to playing against bots programmed using the more traditional fuzzy finite state machines [SH08].

There are multiple ways of capturing a player's behaviours in a model. One of these is inverse reinforcement learning (IRL) (e.g. [AN04]). Here the agents are trained using reinforcement learning, where the reward function has been chosen so that performing optimally corresponds to imitating the player behaviour. Of course this shifts the difficulty to learning a good reward function.

A simpler approach is behavioural cloning (BC) (e.g. [BS95, TWS18]). This is straightforward supervised learning where the player we want to imitate provides state-action pairs (i.e. examples of what do to in which situations), and the goal is to learn the mapping from states to actions (the policy). Note that a behavioural clone (or an agent obtained with other strategies for that matter) need not be the end result. This model could also be used as initialisation for a more complicated imitation or learning approach afterwards (e.g. [Sch97]). For example, reinforcement learning algorithms can take quite a while to 'get going', so we might be interested in using a behavioural clone to speed up the learning process.

1.1 The setting in this thesis: River Raid

As the title of the dissertation suggests, we will use behavioural cloning to capture agents playing River Raid. This is a 1982 video game by Carol Shaw for the Atari 2600 which is conveniently emulated in the OpenAI Gym framework [BCP+16] for reinforcement learning. The game uses an output resolution of 160×210 pixels (with 3 colour channels) and updates at 60 frames per second. A single frame is shown in Figure 1.1. The game is a vertical autoscroller controlled by the four main directions (up, left, down, right) and a shoot button. These can be combined to form 18 actions.[1] The up button speeds up the player's aircraft, the down button slows it

[1] There are nine 'movement' directions obtained by combining none, one, or two of the main directions: no movement at all, up, left, down, right, up-left, up-right, down-left and down-right. All of these directions can be combined with the binary choice of shooting or not.

down. The aircraft always stays at the same vertical position in the frames, namely at the bottom of the play area.

There are multiple enemy types: ships, helicopters and planes. Each has its own simple typical movement patterns. These only start after a certain delay, which is randomised. Apart from this, the game is entirely deterministic. Touching an enemy or the terrain will result in the loss of a life. The player starts off with three lives. Upon a death the game restarts at the last checkpoint (bridges which need to be destroyed first). Apart from enemies and increasingly complicated terrain, the player will also encounter fuel depots. These refill the fuel meter, which constantly runs out at a steady pace, irrespective of the player's speed.[2] When the aircraft is out of fuel, the player loses a life.

Rewards (points) are gained by destroying enemy vehicles, checkpoint bridges and fuel depots. Note that the latter makes the game also tactical: you can either choose to destroy the depot to gain points at the cost of no refuelling, or play it safe and refill.[3] After the player has lost all his/her/its lives, the game ends with as final score the sum of all intermediate rewards. The game is episodic in the sense that after the game ends it immediately restarts from the beginning, with full lives and zero score. Consequently we will define a River Raid episode by the period between such complete restarts.

Figure 1.1: A frame from River Raid (emulated). The player controls the yellow aircraft at the bottom of the screen. Here it is moving to the left after having fired a shot which is about to hit the second enemy ship. The current score is 2690 *which will increase after destroying the ship. Further there is a fuel meter which can be refilled by the fuel pad at the bottom left. The player has two remaining lives (not including the current one). The black borders are part of the image and will not be cut off.*

Formally, we can easily fit the game in the RL framework as a Markov decision

[2] Otherwise constantly slowing down the ship would make the game easier.

[3] Obviously there is also the option of ignoring the fuel depot altogether, either due to a suboptimal playstyle, or prioritisation of other short term goals.

process (MDP).[4] The state space $\mathcal{S}$ can be chosen to be the set of all admissible frames (pixel configurations).[5] The action space $\mathcal{A}$ consists of the 18 possible actions. We mentioned how the reward function $r : \mathcal{S} \to \mathbb{N}$ works (note that the rewards do not directly depend on the chosen action). The transition 'function' $\delta : \mathcal{S} \times \mathcal{A} \to \mathcal{S}$ is probabilistic.[6]

Many RL strategies are tested on (some of) the games in the Arcade Learning Environment (ALE) [BNVB13]. DeepMind's DQN achieved an average score of 8316 on River Raid. For comparison a random agent scores 1339 and a professional game tester 13513 [MKS+15]. Another popular RL technique called Proximal Policy Optimization (PPO) [SWD+17] obtains an average score of 8393.6.

Although the state space of River Raid is immensely large, it dwarves in comparison to those of the aforementioned Starcraft II or Dota 2. Also, most if not all of the possible applications of captured player models are not applicable. However, this thesis is meant more as a proof of concept: if (and only if) learning player models is easy (or at least doable) here, we can move on to more interesting cases. Obviously we have neither the resources (including manpower) nor time to rival DeepMind or OpenAI. As we will use only the raw pixel data for our player models, the techniques used will (in principle) easily carry over to other settings.

1.2 Research questions and structure of the thesis

Our main, but rather broad, research question is:

How well can one behaviourally clone an agent playing a simple video game?

As we already indicated, the simple video game in question here will concretely be River Raid. Although our ultimate goal is to behaviourally clone human players we realise that this will be quite difficult and hence we shall start with the simpler case of cloning an RL agent. The main benefits are that we can easily obtain sufficient amounts of data and that the RL agent, although stochastic, should play somewhat predictably. Although we can get as much data as we want, this does not mean we actually need tons of data. That also warrants an investigation. This will be the content of chapter 2, where we attempt to answer the following research questions:

How well can one clone a reinforcement learning agent when there is plenty of training data?

How much data is really needed for that?

[4] We assume the reader is familiar with the basics of reinforcement learning. If not, we refer to [SB18]. However, knowledge of reinforcement learning is not essential for this thesis. Here we just indicate why the game of River Raid is often used for testing reinforcement learning algorithms.

[5] Alternatively the Gym framework allows the use of the Atari 2600's internal RAM of 100 bytes.

[6] To be more precise (and a bit less intuitive) a stochastic transition function is formalised as a map $\delta : \mathcal{S} \times \mathcal{A} \times \mathcal{S} \to [0, 1] : (s, a, s') \mapsto P(s' \mid s, a)$ which assigns to every combination of current state $s \in \mathcal{S}$, chosen action $a \in \mathcal{A}$ and potential future state $s' \in \mathcal{S}$ the probability that s' will indeed be the next state.

In this chapter we will also develop much of the methodology and performance metrics we will be using in the subsequent chapters.

In chapter 3 we will introduce (more) unpredictability into the mix. Therefore we will try to clone a simple agent which for every frame simply samples an action uniformly.

To what extent is behavioural cloning made more difficult when the target agent is unpredictable?

Finally we move on to an actual human agent (namely the author). In this case we have limited amounts of data and the agent is quite inconsistent. We investigate how well behavioural cloning works here. In an attempt to implicitly use more data we will also consider transfer learning from the RL agent's clones. Thus, in chapter 4 we will attempt to answer the following two research questions:

Is cloning a human agent feasible in light of player inconsistency and the limited availability of data?

How much improvement is there to be gained by using transfer learning from a cloned RL agent?

We conclude the dissertation with chapter 5 where we will present and discuss the answers to these research questions. We will also indicate in what directions further research could be conducted.

All source code for this thesis is available at `https://github.com/LaurensDiels/Behaviourally-cloning-River-Raid-agents`.

Chapter 2

Cloning a reinforcement learning agent

There are a number of advantages to first trying to clone computer agents and RL agents in particular. Firstly, collecting data is fast and inexpensive. Secondly, also after the data collection phase computer agents can be sampled easily. This makes it possible to compare posterior distributions of the chosen actions given the states, whereas for a human agent we will (almost) never have enough data for this.[1] Thirdly, RL agents (tend to) play more consistently. This means there is less variability to capture, which should make learning a lot easier. Related to this is the fact that the expertise of computer agents can be fixed at a certain level. On the other hand, humans usually automatically improve when playing a game. This makes building player models for human agents even harder. These reasons make cloning RL agents a valid simple stepping stone for the real challenge of cloning human agents. We will use this setting to develop methodologies and performance measures which we can also use when cloning other types of agents.

2.1 Model training methodology

In order to train a model as a behavioural clone of the RL agent, we need to have training data and a model architecture, as well as good hyperparameters. Below we explain our approach to obtain these.

2.1.1 Data collection

Exploiting our first advantage of cloning an RL agent, we started by collecting 2571 episodes (2174108 frames,[2] the equivalent of playing the game non-stop for about

[1] A certain exact frame will with a great degree of certainty appear at most once in the training data. Seeing as our frames are $160 \times 210 \times 3 = 100800$ dimensional, it is not surprising to see the curse of dimensionality rear its head.

[2] For the curious reader: we saved the episodes using LZMA (Lempel-Ziv-Markov chain algorithm) compression (which is lossless), requiring only 265MB of storage space, a size reduction of about a factor 1000 compared to the non-compressed data.

10 hours) of a non-improving RL agent playing River Raid. The RL agent was previously trained for 1000 episodes using PPO in the keras-gym implementation [Hol19]. Internally here we preprocessed the input images by downscaling in both spatial components by a factor of two, by converting to greyscale and by stacking three consecutive frames together. The model architecture used was the default Atari function approximator.[3] The average score it achieves is about 2103.[4]

To avoid having to deal with improving agents (at this point) we made sure the RL agent stopped learning after its 1000 training episodes. Thus all 2571 example episodes were collected from the same fixed RL agent.

2.1.2 Choosing a model architecture

In order to obtain a methodology which also works for other games we want our behaviourally cloned model to use pixel data (game frames) as inputs, and actions as output (one per input frame) for the multiclass classification problem that is behavioural cloning. However, we will not use the raw pixel data as literal input and instead will perform two (generally applicable) preprocessing steps. Firstly we will convert all integer colour intensities in $\{0, 1, 2, \ldots, 255\}$ to floats in $[0, 1]$ by a division by 255. Next we will allow for the stacking of multiple consecutive frames in the 'colour' channels. For example, if we would stack five frames, the input would have dimension $160 \times 210 \times 15$.[5] See Figure 2.1. Such a stack consists of the current frame and a number of previous frames, in chronological order (so the current frame is the last in the stack). In the beginning of episodes, where there might not be sufficiently many previous frames, we will repeat the current frame.

Seeing as we will then be working with visual data it makes sense to use a convolutional neural network. A second important point is that we want our models, like the agents we will clone, to be stochastic. We will achieve this by using a technique called MC dropout [GG16], also known as permanent dropout (or PermaDropout). The idea is to use dropout [SHK+14] also at prediction time, instead of only during training. This means that with MC dropout the output actions for a fixed input frame are no longer deterministic, since in different forward passes different connections might be disabled.

[3] With the exception of the function approximator, all hyperparameters were taken over from the example on Pong with PPO from the keras-gym documentation.

[4] Note that this is significantly higher than a random agent, which as we have seen in chapter 1 attains an average score of about 1339. For the purposes of cloning, the RL agent then performs sufficiently well, in particular acting differently to its random initialisation. Although not required, we have also attempted to increase the RL agent's performance. However, we were encountering the issue that most of the time the RL agents would consistently improve up to an average score of about 2000, after which the score would again begin to decline. We tried to combat this by using a greedy approach: we saved the current state of an RL agent, trained for 25 episodes (as 100 turned out to be too many), compared the new performance to the old one, reloaded the previous state or saved the current, depending on which gave the best results, and repeated the process. Using this strategy we were able to increase the average score from 2103 to 2224, only a modest gain at best.

In any case we will here always use the RL agent with average score of 2103, which as mentioned was just trained over 1000 episodes, without any special tricks (in addition to those built into PPO).

[5] In fact, Keras (and numpy) switch the first two components, giving an input of $210 \times 160 \times 15$.

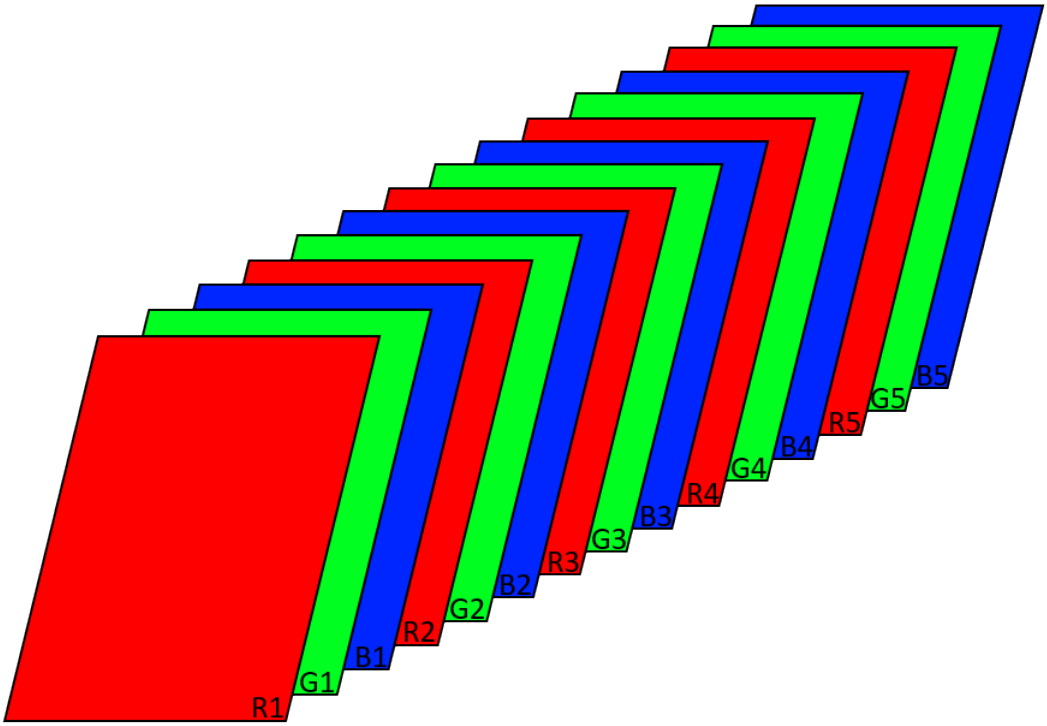


Figure 2.1: Illustration of how we stack frames. This would constitute one input if we stack 5 *frames. Best viewed in colour.*

Speaking of the outputs of our model, we will one-hot encode the 18 possible actions and use such vectors as target values. We will then use a softmax activation function in the final layer of the network, as is usually done for multiclass classification problems. As loss function we will use the categorical cross-entropy loss.

We tried a couple of different architectures in Keras [C+15]. We first considered VGG16 [SZ14] using transfer learning from fixed pretrained weights on ImageNet [DDS+09]. We removed the top classification layers and manually added them back with random small weights and with MC dropout layers with probability 0.5 in between. However, there are three issues with this approach, the first two of which we will discuss immediately and the last at the end of this subsection.

The first concern is that we need to train about 80 million parameters. Considering our inputs are considerably less complex than those from ImageNet, we looked into reducing the amount of nodes in the fully connected (dense) layers by a factor of four. This reduces the number of parameters to train to about 17 million. We will refer to this architecture as VGGQD (where QD stands for "quarter dense"). A second concern is the depth of the network, which will make finetuning afterwards[6] more difficult. To accommodate this problem we tested multiple custom network architectures of various sizes, all inspired by VGG16. We refer to these models as

[6] i.e. unfreezing the ImageNet weights and training further

Model A, Model B, Model C and Model D. Together with VGG16 they are shown in Figure 2.2.

We trained these six networks using 160 episodes from the RL agent. Twenty episodes were used for validation. In both cases and in the entirety of the thesis we only used every third state-action pair, the reason being that two subsequent frames are likely to be extremely similar and thus not very informative. Further we always use a batch size of 32 and use the Nadam optimiser [Doz16] (which is the common Adam optimiser [KB14] combined with Nesterov momentum [SMDH13]) to minimise the categorical cross-entropy loss. In this case we used a learning rate of $1.3 \cdot 10^{-4}$, determined by some earlier tuning using logarithmic grid search and random finetuning. At this point we do not yet stack any frames.

The results in terms of average validation accuracy in the last epoch are shown in Table 2.1. We also present the size of the networks, as expressed through the number of (trainable) parameters and depth.

Architecture	VGG16	VGG16QD	Model A	Model B	Model C	Model D
Trainable parameters	79.78M	16.80M	7.95M	16.31M	3.40M	28690
Depth	26	26	11	10	16	10
Validation accuracy	54.86%	57.57%	64.14%	64.00%	62.57%	62.07%

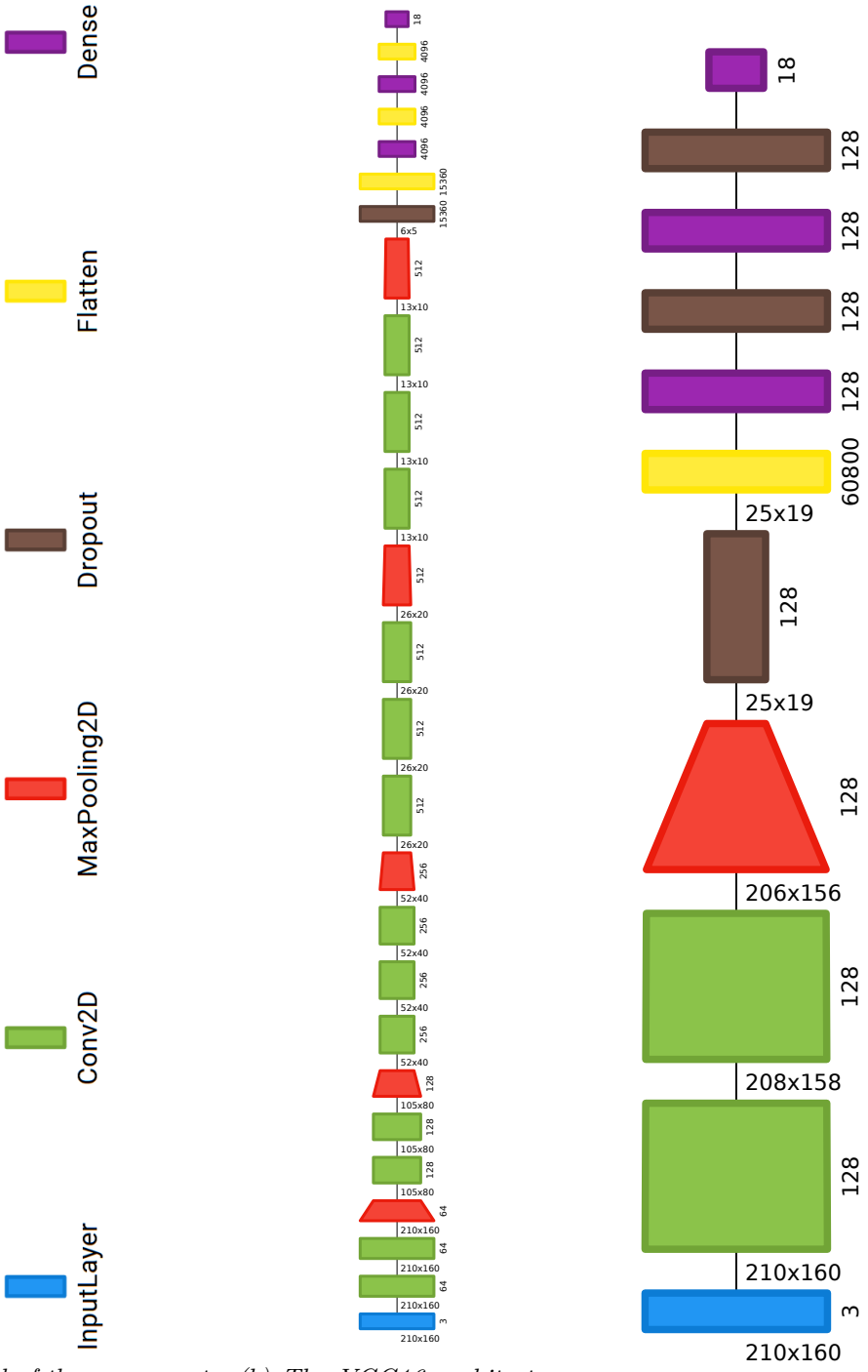
Table 2.1: Comparison of the different model architectures. In the row of the number of trainable parameters, M stands for million.

Of note is that VGG16 (with transfer learning using frozen ImageNet weights) is needlessly large and complex and we actually get better results by taking smaller models. Note also that Model D performs much better than VGG16 although it is tiny.

Based on these results we opted to work further with Model A. Using a greedy approach we checked the effect of L^2-regularisation (weight decay), batch normalisation and frame stacking. Adding L^2-regularisation with a regularisation constant of 0.001 increased performance slightly from 64.14% to 64.43% and should guard against possible overfitting, so from now on we will always use L^2-regularisation.[7] The additional use of batch normalisation before every fully connected layer decreased the validation accuracy to 61.18% and thus we will avoid it. Finally, stacking three frames (while still using regularisation) showed a significant increase in accuracy to 70.96%. Therefore, from now on we will always stack frames.[8] This brings us back to the promised delayed third issue with VGG16 and transfer learning. Since the ImageNet

[7] The value of 0.001 was chosen somewhat arbitrarily. At this point we are not interested in getting the best results possible, just in checking if the addition of e.g. L^2-regularisation would be able to improve performance. If it already does for the value of 0.001, then it will certainly do so for a tuned value.

[8] Earlier we explained why we skip two thirds of the frames (when not stacking), namely because the omitted frames do not contain much extra useful information. This seems at odds with the empirical observation that adding these back through stacking increases performance. But it is important to note that now we are adding a kind of local history to the inputs, which is not equivalent to just using more separate examples.

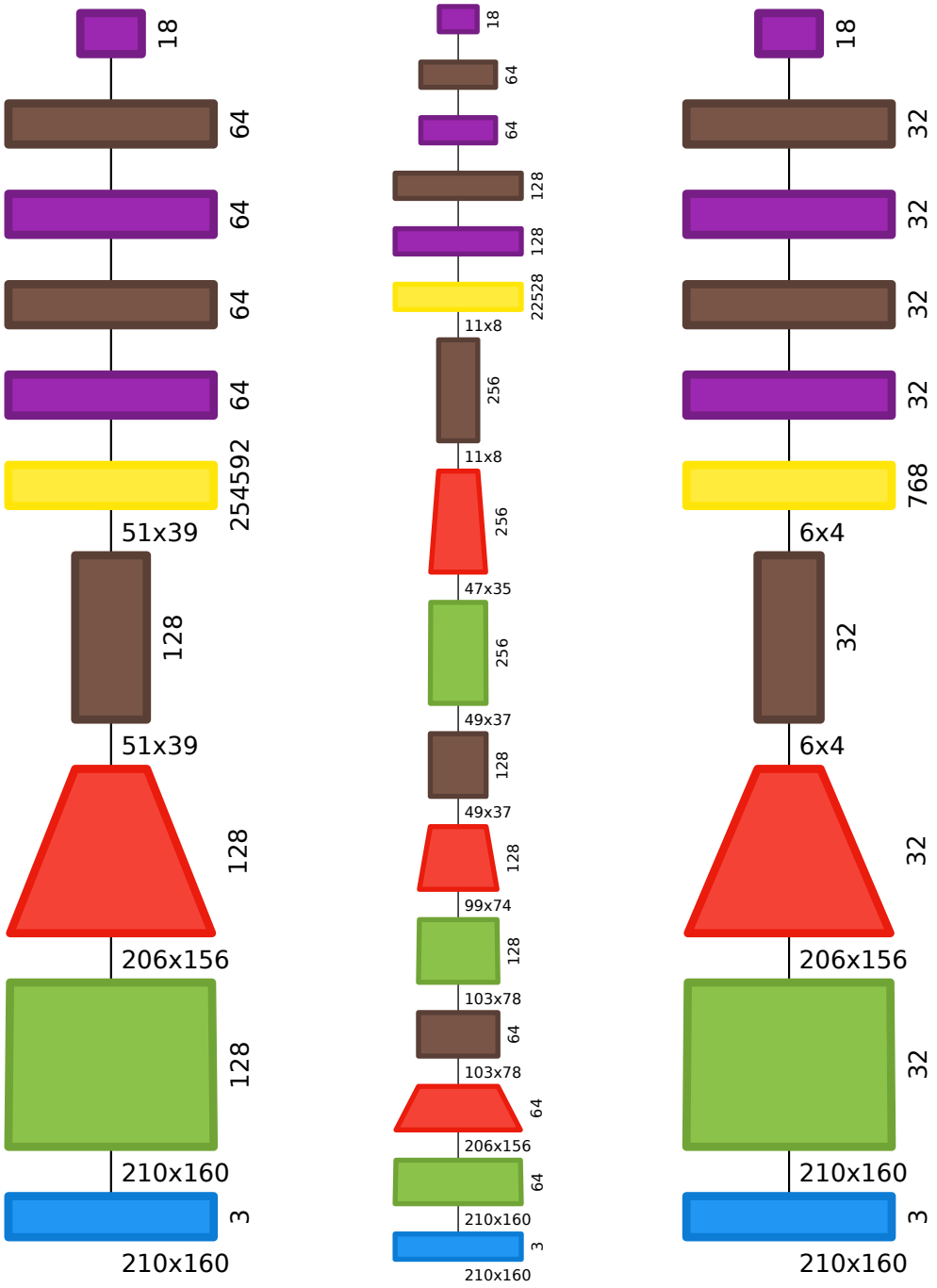


(a) Legend of the components in the convolutional neural networks.

(b) The VGG16 architecture with added MC dropout layers in the classification part.

(c) Our Model A architecture.

Figure 2.2: Schematic summaries of the different model architectures under consideration. Horizontal numbers refer to the size of a feature (scalar if omitted), vertical numbers to the number of output features. All dropout layers will remain active at prediction time. Best viewed in colour. Created using net2vis [BR19].



(d) Our Model B architecture. (e) Our Model C architecture. (f) Our Model D architecture.

Figure 2.2: Schematic summaries of the different model architectures under consideration (continued).

images use three colour channels, the network input shapes are incompatible with frame stacking.

To conclude, from this architecture comparison we decided to move forward with Model A with no batch normalisation, and with three hyperparameters to tune: the learning rate for the Nadam optimiser, the regularisation constant for L^2-regularisation, and the number of consecutive frames to stack.

2.1.3 Tuning the hyperparameters

To tune the three hyperparameters, we used the hyperopt package [BYC13] using the tree-structured Parzen estimator approach (TPE) [BBBK11], which is a form of Bayesian optimisation. We used a loguniform distribution between 10^{-5} and 10^{-3} for the learning rate, a loguniform distribution between 10^{-4} and 10^{-2} for the regularisation constant and a uniform distribution on $\{2, 3, \ldots, 25\}$ for the number of frames to stack. We let the tuning algorithm run for 50 combinations of hyperparameters, evaluated using the average validation accuracy over the last training epoch of our model trained with 50 episodes from the RL agent and using 25 episodes for validation. Each time the model was trained for 10 epochs.

The best hyperparameters found were a learning rate of about $2.114 \cdot 10^{-4}$, a regularisation constant of $7.120 \cdot 10^{-3}$, and the stacking of two consecutive frames. These yielded a validation accuracy of 66.15%.[9,10]

2.1.4 Training behavioural clones

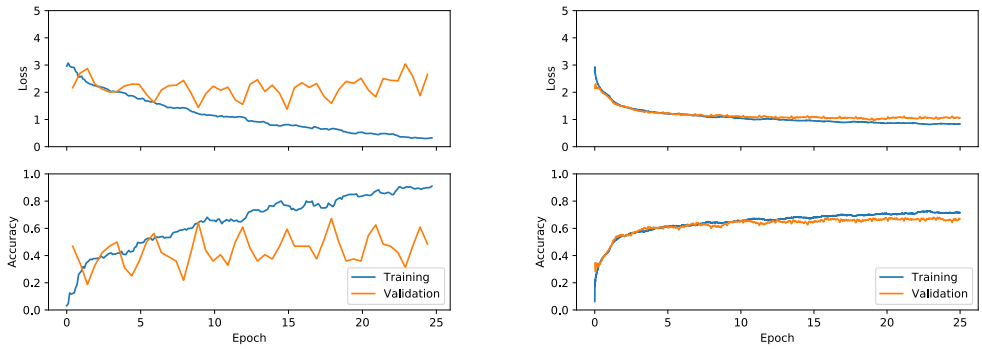
We then trained models using these tuned hyperparameters. We used 1, 2, 5, 10, 25, 50, 125, 250, 500, 1000 and 2000 training episodes and trained for 25 epochs. The number of validation episodes was a quarter of the number of training episodes, rounded up. We made sure that the training sets were nested, e.g. the 25 episodes used to train the corresponding model were also used (in addition to 25 new ones) for the model using 50 episodes. The same is true for the validation sets.[11]

When using only one training episode, 25 epochs was not sufficient for convergence of the model (see Figure 2.3a). But the validation loss was no longer improving,

[9] For comparison, the worst performing combination of hyperparameters gave a validation accuracy of 37.27%. However, the average over the 50 runs was significantly better at 59.82%. The standard deviation was 5.767%.

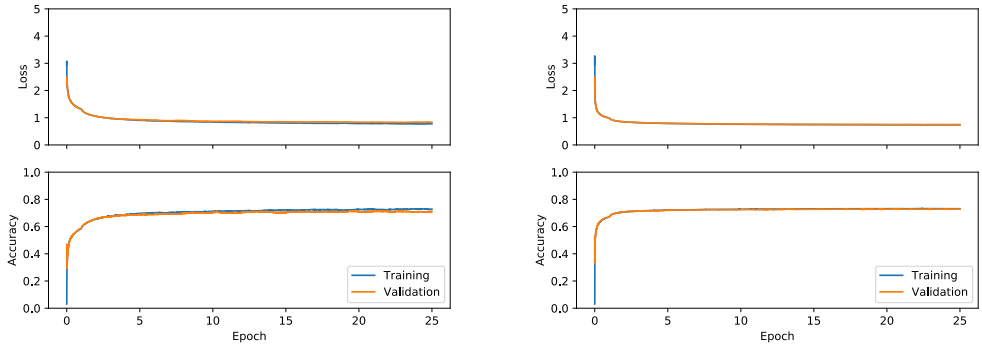
[10] The attentive reader might have noticed that this validation accuracy (66.15%) is lower than the one we found when we were investigating the usefulness of frame stacking (70.96%). Seeing as we are now using tuned hyperparameters this seems counter-intuitive. But keep in mind that we now only use 50 training episodes compared to the previous 160 (to reduce the computational requirements and allow us to test 50 combinations of hyperparameters). Below, in Table 2.2, we will see that using more episodes does increase validation accuracy (to some extent), as is to be expected.

[11] Note that this makes the validation performance less noisy than when we would have just used different (random) validation episodes for each of the different models. In hindsight, for the fairest (least noisy) comparison we just should have always used the same validation sets, even though this might lead to totally mismatched sizes of the training and validation data. But seeing as in section 2.2 we will not evaluate our clones using any validation information, all of this does not matter much.



(a) For 1 training episode.

(b) For 25 training episodes.



(c) For 250 training episodes.

(d) For 2000 training episodes.

Figure 2.3: Loss and accuracy learning curves for the behavioural clones of the RL agent, trained on various numbers of episodes. All curves were smoothed using a backwards simple moving average with epoch window size.[12]

nor was the accuracy. For 2000 training episodes there were already only marginal improvements after the first couple of epochs (see Figure 2.3d). In this situation early stopping would have helped considerably in terms of required computation time. But we can only draw this conclusion precisely because we trained for sufficiently long (which we are allowed to do because of the dropout layers and L^2-regularisation). Therefore for the remainder of the thesis we will keep training models for 25 epochs, even though this is excessive most of the time.

In terms of validation loss and accuracy the clones' performance is listed in Table 2.2. Although the relevance is disputable, we see that generally validation loss goes down and accuracy up with the number of training episodes. But already from 250 episodes onwards, there are no longer any major gains in validation accuracy.

[12] If $(x_n)_{n\geq 1}$ is a real sequence, then its backwards simple moving average with window size $w \in \mathbb{N} \setminus \{0\}$ is the sequence $(y_n)_{n\geq 1}$ where $y_n = \operatorname{avg}\{x_{n-w+1}, x_{n-w+2}, \ldots, x_n\}$, i.e. the average of last w values, including the current one. Of course this formula only makes sense when $n \geq w$. For $n < w$ we use $y_n = \operatorname{avg}\{x_1, \ldots, x_n\}$.

The relation between number of training episodes and loss or accuracy is also not perfectly monotonic.

2.2 Evaluating the clones

So far we have used validation accuracy as performance metric. However, our goal is not to obtain a clone which plays exactly the same as the RL agent, but rather a clone that plays similarly. We do not care (much) if it decides to move left one frame later than the RL agent would have. But this behaviour will be punished in terms of accuracy. Therefore, we devise four alternative performance metrics: two based on the action distribution, and two based on the number of frames and scores in episodes.

2.2.1 Comparing distributions

Seeing as our models are stochastic, in every state we (normally) have a non-degenerate conditional distribution of actions given that state. There is also the global distribution of actions. To find these and compare them between the clone-agents and the original RL agent, we used an episode from a human player, extracted every 10th frame (stack) from it and sampled the models 100 times for each of those frames. This resulted in 115 empirical conditional distributions for every agent, based on these 100 samples. By aggregating all 11500 actions we get an idea of the global action distribution.

To compare (conditional) distributions P and Q one often uses the KL divergence

$$\mathrm{KL}\left(P \parallel Q\right) = \sum_x P(x) \log_2 \frac{P(x)}{Q(x)}.$$

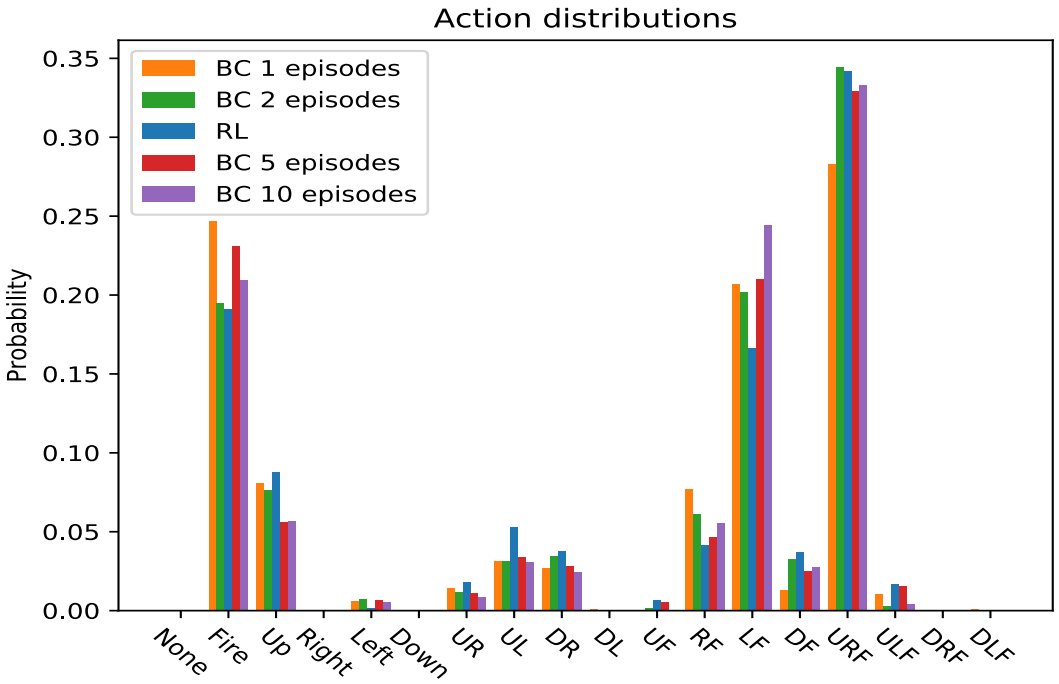
Here the summation is taken over all possible values x of the discrete distributions. However, if for some value x we have $Q(x) = 0$ while $P(x) > 0$,[13] then $\mathrm{KL}\left(P \parallel Q\right) = +\infty$. In our case we have to sum over the 18 possible actions x and $P(x)$ is the quotient of the number of times we sampled action x for an agent π_P, and the number of samples in total, i.e. 100. However, since it does happen that clones (with distribution Q) never use a certain action which the RL agent (with distribution P) does use occasionally, we will often have $\mathrm{KL}\left(P \parallel Q\right) = +\infty$ which is not very informative. This means that the KL divergence is not a good measure in this case.

Therefore we simply use the L^1-distance

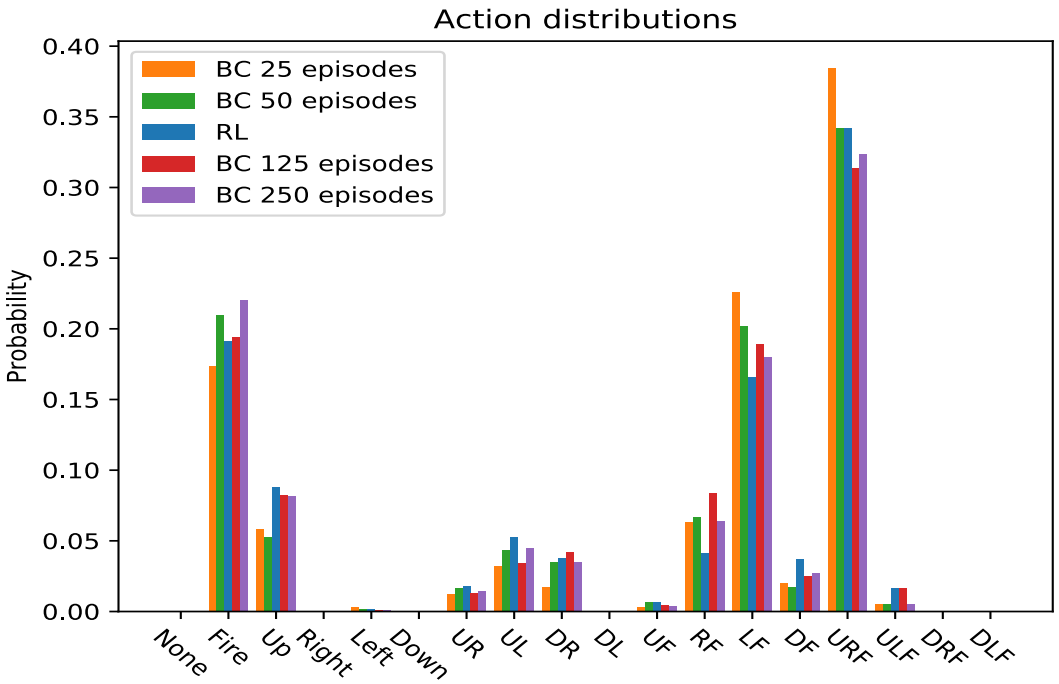
$$d_1(P, Q) = \sum_x \left|P(x) - Q(x)\right|.$$

For the global distribution we can use this as is, but for the conditional distributions we would get 115 distances for every clone. Thus we aggregate these 115 values simply by taking the mean. This results in the mean conditional (or local) L^1-distance.

[13] If $P(x) = 0$, then since the right-hand limit $\lim_{t \searrow 0} t \log_2 t = 0$, we will equate the term $P(x) \log_2 \frac{P(x)}{Q(x)}$ to 0, even if $Q(x) = 0$.

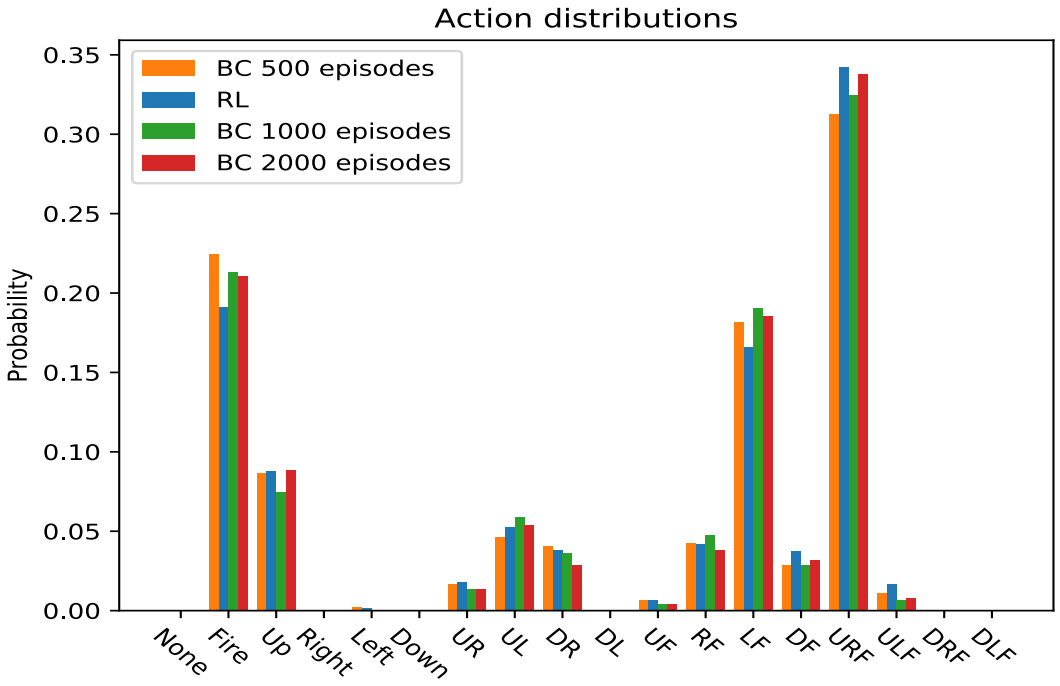


(a) For 1, 2, 5 *and* 10 *episodes.*



(b) For 25, 50, 125 *and* 250 *episodes.*

Figure 2.4: Histograms of the global action distributions of the RL agent and its behavioural clones trained using varying numbers of episodes. For the labels of combined actions we abbreviated Up by U, Right by R, Down by D, Left by L, and Fire by F.



(c) For 500, 1000 *and* 2000 *episodes.*

Figure 2.4: Histograms of the global action distributions of the RL agent and its behavioural clones (continued).

In Table 2.3 we list the performance of all clones with respect to these measures. For the global action distributions we also supply histograms in Figure 2.4. To get some frame of reference for the quantitative values, we present the results of an agent sampling actions uniformly, irrelevant of the current frame (stack). This agent qualitatively behaves very different from the RL agent. The L^1-distance between the global action distributions of the RL agent and such a uniform agent was 1.131, the mean conditional L^1-distance 1.794. So the clones at least outperform such an agent by a considerable margin (as one might hope).

Generally speaking the distance between the global action distributions of the RL agent and the clones goes down (improves) as the number of episodes increases. The same holds for the average local distances. However, it should be noted that in both cases this is not monotonic behaviour. Additionally, the global and local results are not perfectly correlated. In any case there seems to be no substantial improvement from 250 episodes onwards. Recall we came to the same conclusion when comparing validation accuracies.

2.2.2 Time-score scatter plots

The second group of performance metrics can also be used when we cannot sample the agent (as is the case for human agents). Here we just play episodes and at the end consider the time (number of frames) the agent survived and the score it has

obtained. By repeating this multiple times, we gain a scatter plot of time-score such as Figure 2.5. Note that such scatter plots contain relevant information about the playstyle of agents, such as whether it plays slowly (possibly lower scores, but longer survival times) or aggressively (relatively high scores, possibly lower survival times). Further note that we are here using information of the agents actually playing the game of River Raid. In the distribution comparison we sampled actions in states that might potentially never be reached. For example, in the past we have encountered situations where an agent and its clone had very similar distributions, but upon visual inspection it turned out that the clone actually always immediately steered into a wall, whereas the base agent moved close to the wall, but did not crash. Only a small difference in distributions might then still lead to a significant change in the actual observed playstyle, something which for this example would be immediately clear from the time-score scatter plots.

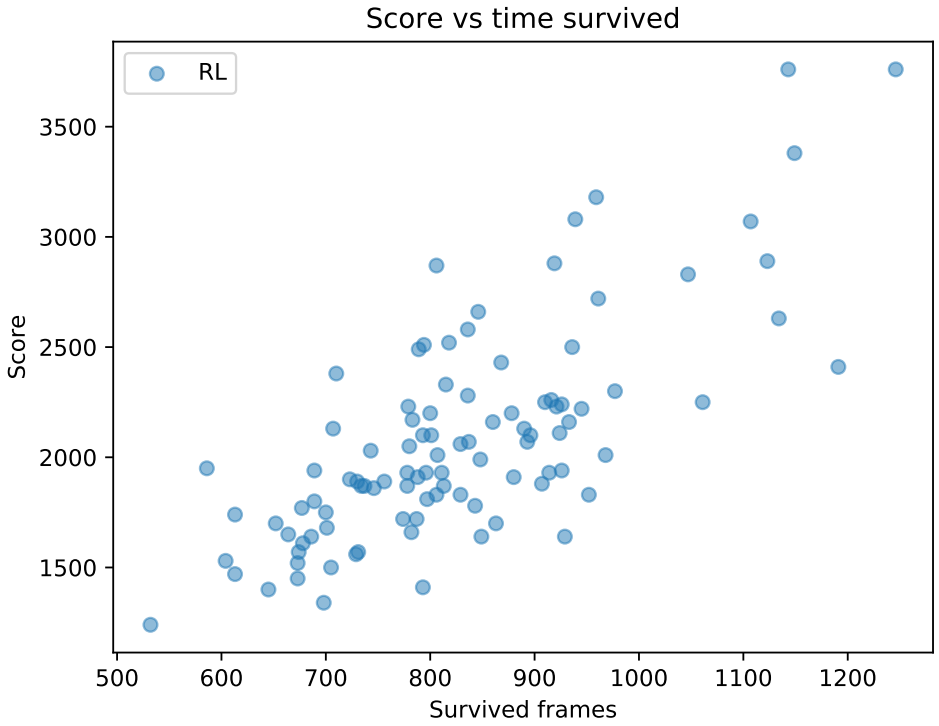


Figure 2.5: Score versus time (in frames) scatter plot of the reinforcement learning agent. Every one of the 100 *blue points corresponds to one episode played by the RL agent. Its x-coordinate is the number of frames the agent managed to survive in this episode. The y-coordinate is the total score the agent was able to obtain in this time.*

Similar agents should have similar scatter plots and therefore we need to find dissimilarity measures[14] between scatter plots. We will use the following two.

[14]None of the measures we will present are actual distance metrics, but that is not really important.

KL divergence between fitted Gaussians (FGKLD) The scatter plots are just sets of $N \in \mathbb{N}$ pairs (t_i, s_i), where $t_i \in \mathbb{N}$ is the number of frames the agent survived (until all lives ran out) and $s_i \in \mathbb{N}$ is the obtained score.[15] Here $i \in \{1, 2, \ldots, N\}$. We can then fit a two-dimensional Gaussian distribution using the maximum likelihood parameters

$$\widehat{\mu} = \frac{1}{N} \sum_{i=1}^{N} (t_i, s_i)$$

and

$$\widehat{\Sigma} = \frac{1}{N} \sum_{i=1}^{N} \left((t_i, s_i) - \widehat{\mu}\right) \left((t_i, s_i) - \widehat{\mu}\right)^T .$$

These estimate the mean and covariance matrix, respectively. In Figure 2.6 we plot the fitted Gaussian to the time-score scatter plot of Figure 2.5. The KL divergence can then be used to compare such fitted normal distributions. We call the resulting measure the *fitted Gaussian KL divergence FGKLD*.

Kolmogorov-Smirnov two-sample test p-values (KSp) There are two downsides to the previous dissimilarity measure. Firstly, this only makes sense when the collected data points are (roughly) normally distributed. Although this seems in agreement with Figure 2.6 it is nevertheless an extra assumption we need to make. And secondly, the KL divergence is not normalised. It is not clear if a KL divergence of 1 is good or bad.

To remedy this first issue and to an extent the second, we additionally use the following approach. We will perform a hypothesis test of whether the two point clouds have been sampled from the same underlying distribution or not. One possibility is the Kolmogorov-Smirnov two-sample test, generalised to two-dimensional data [FF87]. This test is non-parametric, and in particular does not assume Gaussian data. The p-value under the null hypothesis of same underlying distribution then gives a normalised similarity measure. If it is 0, then we can be certain that the two sets of data follow different distributions. If it is 1, then (based on the data at hand) they have the same distribution.

The usual interpretation is that we should reject the hypothesis of equal distribution if $p < 0.05$ (or the chosen significance level). This gives a binary decision. Using the actual p-values for a more gradual decision comes at a risk though. When

Conversely, having an actual distance metric does not mean it is a satisfactory dissimilarity measure for this context. For example the map D given by

$$D(X, Y) = \max_{x \in X} \min_{y \in Y} \|x - y\|_2^2 + \max_{y \in Y} \min_{x \in X} \|x - y\|_2^2$$

for compact (in particular finite) sets $X, Y \subset \mathbb{R}^2$ can be checked to be a distance metric. However, it is clearly not robust to outliers.

[15] Scores in River Raid are always natural numbers. For other games they might be (non-natural) reals. This makes no difference for the analysis here. The presented measures can equally well be applied to other games.

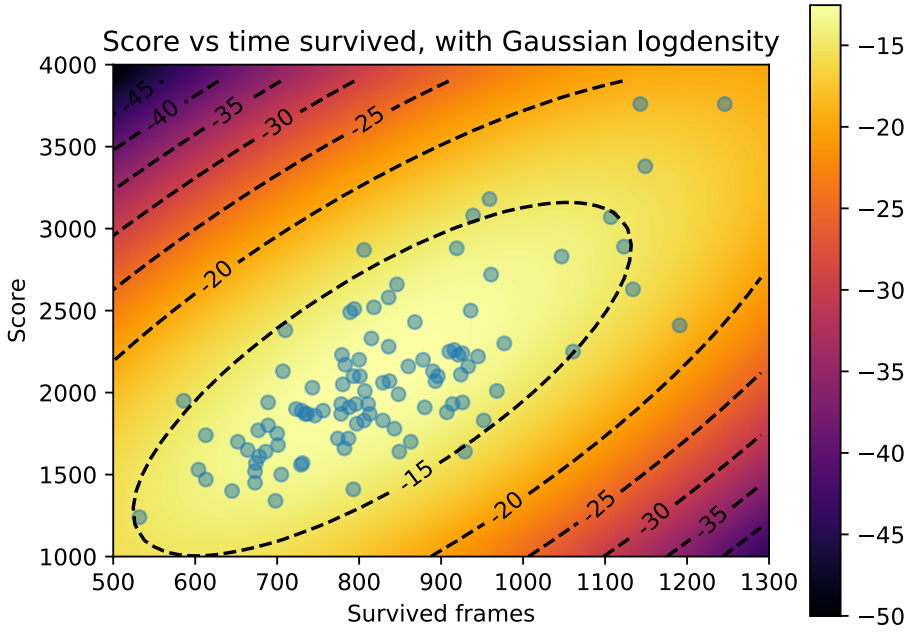


Figure 2.6: The time-score scatter plot from Figure 2.5 with overlaid Gaussian density and contour lines. The background colours, quantified in the colour bar at the right, indicate the logdensity of the Gaussian fitted using maximum likelihood estimation. Also shown are some of its contour lines.

comparing points sampled from the same distribution, we should not expect $p = 1$. In fact, as a test we used 100 time-score points from the RL agent and split them randomly into two groups of 50. Comparing these two point clouds gave wildly varying p-values, from higher than 0.95 to lower than 0.01.[16] Note that in the latter case we would even have rejected the hypothesis of equal distribution, even though it is actually valid here.

Results The behavioural clones are evaluated with respect to these measures in Table 2.4. The scatter plots are shown in Figure 2.7. First of all, note that the FGKLD and KSp are roughly inversely correlated: when the KL divergence is high, the p-value is small, and vice versa. But this relation is certainly not absolute. For example, according to the FGKLD the behavioural clone trained on 125 episodes does a better job at capturing the RL agent's behaviour than the clone from 50 episodes. However, according to the KSp it is the other way around.

[16] In our tests the mean p-value was about 0.41 with a standard deviation of around 0.23.

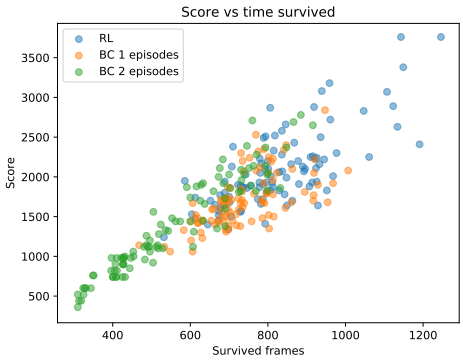
The results are very chaotic: there is no clear connection between the number of training episodes and the similarity of the resulting scatter plots. According to both metrics the clone with access to the most data (2000 episodes) performs worse than all clones with at least 25 episodes at their disposal. In fact, from these clones it is also the only one for which the Kolmogorov-Smirnov test at a significance level of 0.05 would reject the null hypothesis and declare the clone's and RL agent's scatter plots to be sampled from different distributions. The best clone is the one trained on 10 episodes. Visually its scatter plot is indeed very similar to the one from the RL agent (see Figure 2.7b). We also looked directly at episodes played by the model from 10 episodes and were not able to distinguish them from the ones from the real RL agent. But the same was true for the clone of 2000 episodes. We want to point out that for the clone trained on 5 episodes (for example) we were able to distinguish it from the real RL agent as it tended to die early more often, as can also be seen in the scatter plot (see Figure 2.7b).

We cannot draw many strong conclusions from these results. One would be that using fewer than 10 episodes is insufficient. Another that using many episodes might not necessarily be required. Finally, the FGKLD and KSp seem to be conservative estimators of the real cloning performance. If they indicate that a clone performs well, this is likely to be confirmed upon manual investigation of the playstyle. But the converse does not hold: even if they indicate that a clone performs badly, it might still be indistinguishable from the real RL agent, at least according to our eyes.

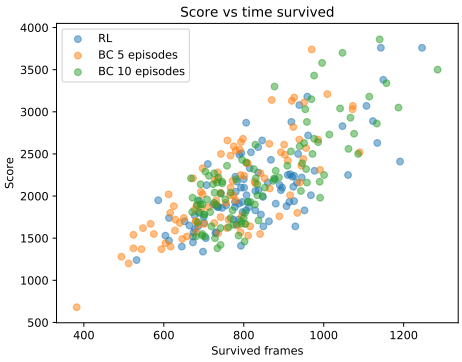
2.3 Conclusion

How well can one clone an RL agent? The answer will depend on what properties of the clone we are looking for. If we want clones that have similar posterior distributions in all states (given the immediate history), or have similar global distributions, then our answer would be: quite well. In Table 2.3 and Figure 2.4c we have seen that the clone trained on 2000 episodes from the RL agent is able to capture these distributions quite closely. If we want clones to have a similar playstyle, then the answer is again positive and hopeful. Visual inspection showed that the clone from 2000 episodes played similarly compared to the RL agent. However, quantitatively our measures of FGKLD (fitted Gaussian KL divergence) and KSp (Kolmogorov-Smirnov two-sample test p-value) based on the time-score scatter plots of the (cloned) agents, are conservative and need not agree with our intuitive findings. But even when we solely use these measures, the behavioural clone trained on 10 episodes performed very well. In terms of the action distributions it does work somewhat differently compared to the RL agent, though. The clone of 1000 episodes is quite good at capturing both the distribution and the overall playstyle (excluding some early deaths (see Figure 2.7e) which do not seem to bother the KSp).

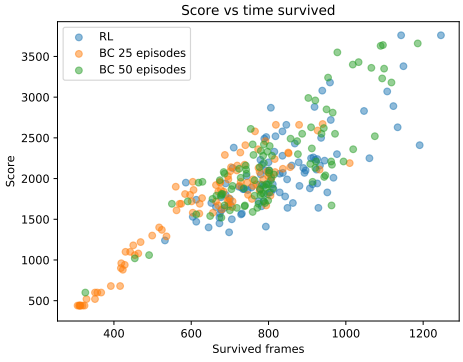
How many episodes do we need? We do not have a clear answer for this question. To better capture the distributions, more episodes are generally useful. We advise to use at least 250 training episodes. But it is not automatically guaranteed that



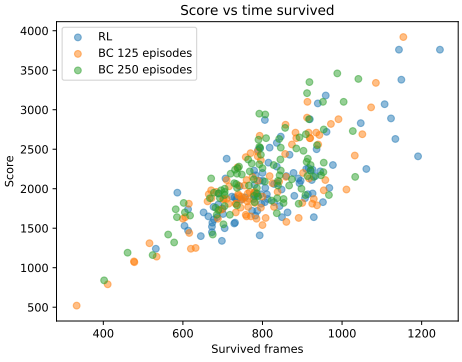
(a) BCs from 1 *and* 2 *episodes.*



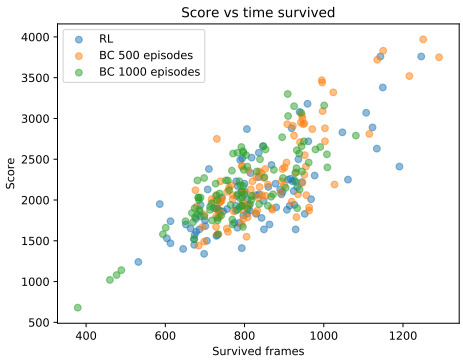
(b) BCs from 5 *and* 10 *episodes.*



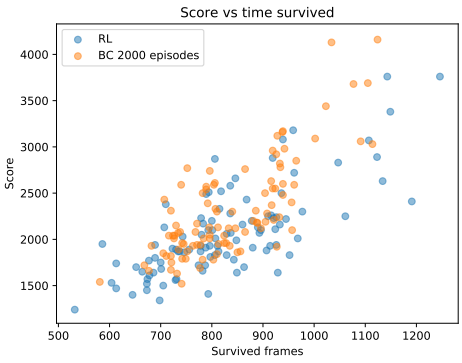
(c) BCs from 25 *and* 50 *episodes.*



(d) BCs from 125 *and* 250 *episodes.*



(e) BCs from 500 *and* 1000 *episodes.*



(f) The BC from 2000 *episodes.*

Figure 2.7: Time-score scatter plots of the reinforcement learning agent and its behavioural clones (BCs). The plots are created by letting all agents play for 100 *episodes.*

increasing the training data will decrease the L^1-distance between action distributions. Random factors such as the initialisations of the models are likely also at play. For capturing the time-score scatter plots the results are even less clear-cut. Here we can only say that using 5 training episodes or fewer is a bad idea. But even with 10 episodes we can get very good results. And conversely, at least according to our measures, we can also get poor results using many episodes. We should again point out, however, that this is mainly due to the conservative nature of these measures.

Training episodes	1	2	5	10	25	50	125	250	500	1000	2000
Validation loss	2.655	1.468	1.525	1.235	1.057	1.012	1.232	0.8360	0.8793	0.7616	0.7490
Validation accuracy	48.44%	56.25%	57.19%	60.62%	66.92%	67.80%	65.01%	71.00%	70.17%	72.50%	72.86%

Table 2.2: Validation loss and accuracy of the behavioural clones of the reinforcement learning agent, trained on various numbers of episodes.

Training episodes	1	2	5	10	25	50	125	250	500	1000	2000
Global L^1-distance	0.2774	0.1341	0.1897	0.2287	0.2522	0.1595	0.1449	0.1297	0.1064	0.1167	0.07930
Mean local L^1-distance	1.099	0.8290	0.7313	0.7183	0.6496	0.6003	0.6365	0.4518	0.4979	0.4266	0.4534

Table 2.3: Evaluation of the behavioural clones of the reinforcement learning agent in terms of capturing the RL agent's action distributions, as measured by the L^1-distance between the global distribution of a clone and the RL agent, and by the mean L^1-distance between the local distributions. Lower is better. For a point of comparison, an agent sampling actions uniformly has a global L^1-distance of 1.131 *and an average local L^1-distance of* 1.794*.*

Training episodes	1	2	5	10	25	50	125	250	500	1000	2000
FGKLD	1.218	4.324	0.3273	0.08024	1.882	0.1838	0.1056	0.2620	0.2131	0.4118	0.5033
KSp	$2.002 \cdot 10^{-6}$	0	0.02921	0.2670	0	0.1509	0.09667	0.07243	0.05845	0.1134	0.01617

Table 2.4: Evaluation of the behavioural clones of the reinforcement learning agent in terms of capturing the RL agent's time-score scatter plot, as measured by the fitted Gaussian KL divergence (FGKLD, lower is better) and the p-value for the two-sample Kolmogorov-Smirnov test for equal (two-dimensional) distributions (KSp, higher is better). Values of zero are due to limited numerical precision.

Chapter 3

Cloning a random agent

The RL agent plays quite consistently (e.g. it always starts off by moving to the right). This should make behavioural cloning easier. Because humans are likely to use a more varied playstyle, it is worth investigating how well we can capture inherent randomness. Therefore, we now try to behaviourally clone a simple random agent which selects its actions uniformly, without taking the state information into account at all.

3.1 Training methodology

We collected 511 episodes of a random agent playing River Raid. This sits comfortably above the minimum number of 250 episodes we recommended for cloning the RL agent.

We used the same model architecture (Model A) as for the RL agent. We again used Bayesian hyperparameter optimisation over 50 runs,[1] with the same prior distributions. And again each model was trained during 10 epochs with 50 training episodes and 25 episodes for validation. This process yielded optimised hyperparameters of a learning rate of $1.841 \cdot 10^{-5}$, a regularisation constant of $1.462 \cdot 10^{-3}$ and the stacking of 10 consecutive frames, giving a validation accuracy of 6.204%. Since there are 18 possible actions which the random agent samples uniformly, the expected accuracy should be $1/18 \approx 5.556\%$. The fact that these hyperparameters gave better results can only be due to luck, or some overfitting on the validation episodes. But it is worth noting that the training accuracy was a bit higher at 8.708%. This indicates that we are not just sampling the same action over and over again (which would equally result in an accuracy of 1/18).[2]

The worst validation accuracy encountered in the tuning process was 4.909%, the mean 5.570% and the standard deviation 0.2456%. From this we conclude that

[1] Because of practical reasons some of these were performed locally, i.e. not on the VSC servers. Sadly when the number of frames to stack was too high (around 15 or more), we would sometimes run out of memory and crash. In the end the local tuning performed 15 successful runs. The results will then be somewhat biased towards lower numbers of frames to stack.

[2] The loss when always sampling the same action should also be higher. Therefore a clone of the random agent is unlikely to ever learn such a policy.

the hyperparameter tuning really makes almost no difference in terms of validation accuracy. This is not unexpected from a theoretical point of view, but was worth confirming empirically nevertheless. In particular we could probably stack fewer frames without any performance degradation, but with a noticeable improvement in execution time and memory usage. Regardless, in following suit with the methodology established in chapter 2 we will keep working with these tuned hyperparameters.

Next we trained a behavioural clone of the random agent using 25 epochs of 400 episodes for training and 100 for validation.[3] The learning curves for loss and accuracy are shown in Figure 3.1. Clearly we need not train for this long: one epoch already more than suffices.[4] Like in chapter 2 early stopping would be useful.

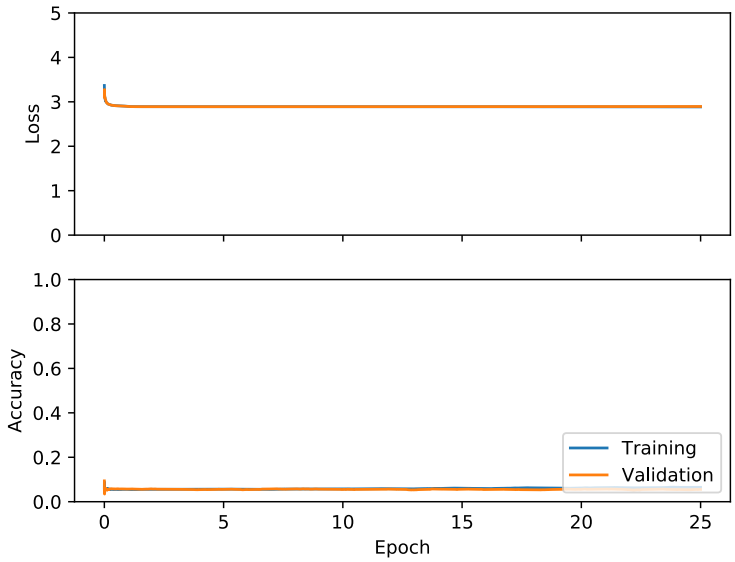


Figure 3.1: Loss and accuracy learning curves for the behavioural clone of the random agent. All four curves were smoothed using a backwards simple moving average with epoch window size.

3.2 Evaluating the clone

We again compare action distributions and time-score scatter plots.

3.2.1 Distributions

The L^1-distance between the global action distributions of the random agent and its clone was 0.3023. In Figure 3.2 we also present a histogram of both agents' global

[3] As was the case for the RL agent, we can easily generate more episodes, so that we need not set apart a test set in advance.

[4] The training accuracy keeps increasing ever so slightly, but this is not very relevant.

action distributions. It is clear that the clone's distribution is considerably less uniform than the random agent's action distribution.[5]

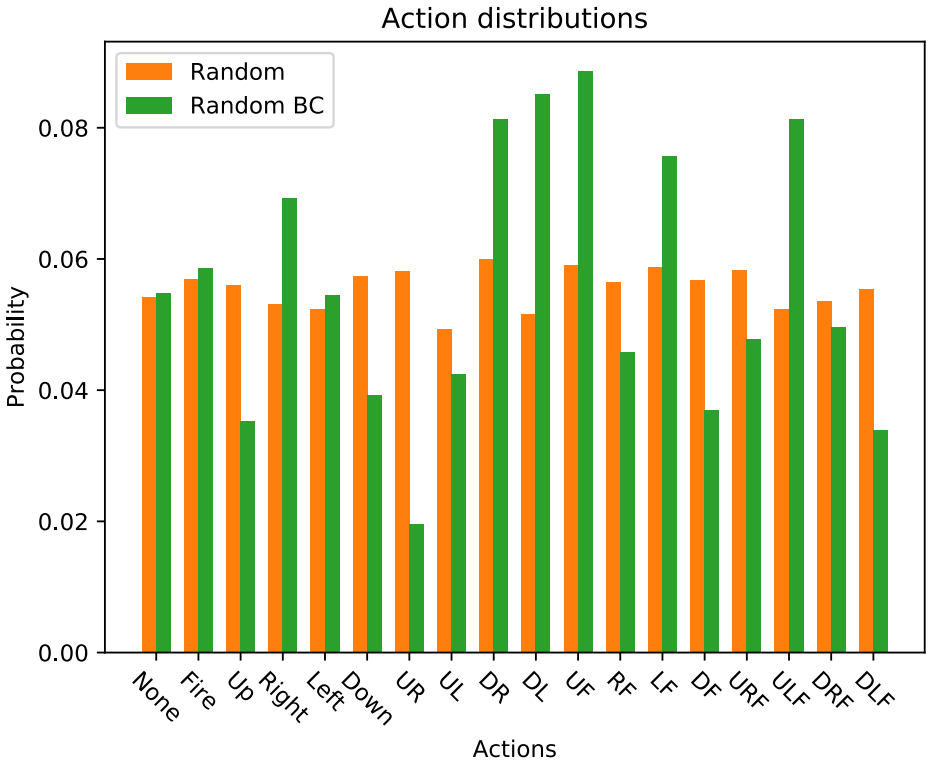


Figure 3.2: Histograms of the global action distributions of the random agent and its behavioural clone.

The mean conditional (local) L^1-distance was 1.050. It should be noted that for consistency with the RL agent case we used only newly sampled[6] empirical distributions for the random agent, even though in this case we have complete knowledge of the underlying distributions (being uniform). Potentially this might inflate this number. For example, if we compare the conditional distributions of the random agent to a perfect uniform distribution, the mean conditional L^1-distance is 0.3332. If we were to (considerably) increase the number of samples per state up from 100, this quantity should reduce to close to zero. Consider Figure 3.3 for an example of a sampled local distribution. Note that indeed it does deviate considerably from a perfect uniform distribution.

In conclusion, we did not capture the distributions of the random agent very well. In fact these results are comparable to the results when comparing the RL agent to

[5]The L^1-distance between the random agent's global action distribution and a perfect uniform distribution is 0.04471. This is not precisely 0 since we are working with only finitely many (11500) samples.

[6]in this way again creating an on-demand test set

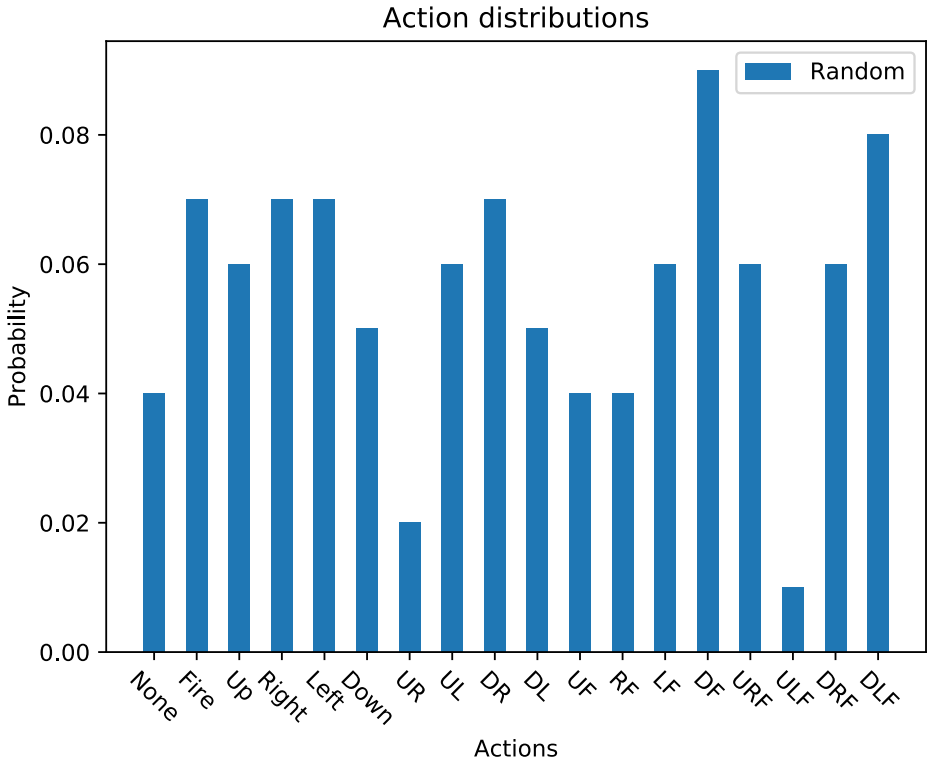


Figure 3.3: Example of a local (state-conditioned) distribution for the random agent.

the clone trained on just one episode.

3.2.2 Time-score scatter plots

The obtained time-score scatter plots of the random agent and its clone are shown in Figure 3.4. It is clear that the clone plays differently from the target random agent. More specifically, it regularly plays noticeably worse. The random agent moves essentially straight ahead, since its movement actions mostly even each other out. This seems to be less the case for the behavioural clone. We should point out, however, that when manually inspecting the clone's playstyle it is passable for the real random agent, its actions being very brief and erratic.

But the numbers do not lie. From the scatter plot it is clear that the clone plays worse than the real random agent. This is also picked up by the KSp, which is quite low at 0.0621 (though not low enough to conclude at a significance level of 0.05 that the scatter plots were sampled from different distributions). Interestingly the FGKLD seems to disagree, being low at only 0.1969.[7]

[7] This is comparable to the FGKLD between the RL agent and its clone of 50 episodes (Figure 2.7c). Note however that in this scatter plot we have a much higher range of scores. This might (unfairly) make the visual comparison between the random agent and its clone seem inferior.

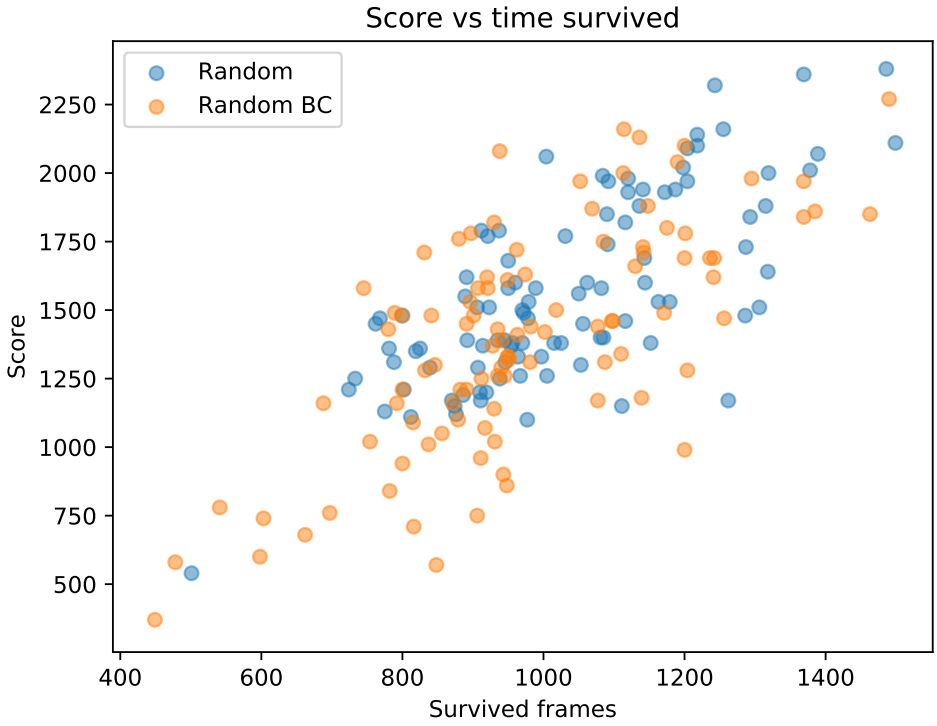


Figure 3.4: Time-score scatter plots of the random agent and its behavioural clone (BC). The plots are again created by letting both agents play for 100 *episodes.*

We conclude that there is a measurable difference between the playstyle of the random agent and its clone. But intuitively upon visual inspection the playstyles do still seem quite similar.

3.3 Conclusion

Although basing our conclusions upon just a single clone means we can only cautiously draw limited conclusions, from the tests we have done it certainly seems that cloning unpredictable random agents is more difficult. At the same time, at least to our eyes, it is hard to distinguish between different erratic playstyles. Therefore the random agent and its clone look quite similar. Depending on the use case of clones the difficulty in capturing the specific kind of unpredictable behaviour may be then more or less of a problem.

Chapter 4

Cloning a human agent

Now we come to what is surely the most exciting and relevant question: how well can we clone a human agent? Indeed, cloning RL agents and random agents is not a goal in and of itself. Those were merely stepping stones and comparison material to the useful human context. But cloning humans will be significantly more difficult than cloning computer agents. As mentioned numerous times already, humans tend to play less consistently, which makes the problem inherently harder. Related to this is that their playstyle also simply changes (hopefully improves) over time. But there are additional compounding factors. Computer agents can be sampled indefinitely and rapidly. Collecting episodes from human players is much slower and/or more expensive.[1]

4.1 Methodology

We collected 254 episodes of the author playing River Raid. This constitutes 531102 frames, or (the equivalent of) about 2.5 hours of non-stop playing. We used the same model architecture (Model A) as before and again used Bayesian hyperparameter tuning, this time only comparing 44 configurations due to practical considerations. As always we trained our model for 10 epochs, using 50 episodes for training and 25 episodes for validation. The best found hyperparameters were a learning rate of $2.083 \cdot 10^{-4}$, an L^2-regularisation constant of $3.107 \cdot 10^{-4}$ and the stacking of four consecutive frames. These resulted in a validation accuracy of 65.79%.[2]

Using these hyperparameters we then trained our model for 25 epochs using 80% (203) of the episodes for training and 10% (25) for validation. The remaining 10%

[1] For the RL agent we were able to play the game at an average of 188.0 frames per second. The random agent played at 524.6 frames per second. This means that the RL agent plays more than 3 times faster than a human player (at 60 frames per second), and the random agent almost 9 times faster.

[2] As per usual we provide some reference material. The worst configuration of hyperparameters encountered gave a validation accuracy of 27.66%. The average found validation accuracy was 56.29% and the standard deviation was 9.103%. Therefore, hyperparameter tuning seems more important now than it was for the cloning of the RL agent.

(26) will be used for testing the model. In figures and tables we will refer to the resulting clone as *BC Scratch*.

For cloning the RL agent and the random agent we used 20% of the episodes for validation. The reason why we are now halving that is because we lack episodes to begin with. Therefore, we want to maximally use the episodes for training. Given the computational complexity of training these deep neural networks, more data-efficient techniques such as cross-validation are out of the question. Additionally, this time we need to explicitly set apart a test set. Note that even now we are still below the 250 episodes we advised for cloning an RL agent, which moreover presumably already requires fewer episodes, being less complex.

Because of the lack of data we also train additional models using transfer learning from the behavioural clone of the RL agent with access to 2000 training episodes. Surely the features learnt in the intermediate layers are useful also for cloning human agents. Usually in transfer learning there is a more significant difference between the base model and the target model (which uses parts of the base model). Indeed, if for example we want to perform real-life object recognition we might use ImageNet weights. But it is likely that we will not want to use the same exact classes as in ImageNet. In our River Raid situation, however, we do have the same output labels (actions) in the RL agent case as in the human agent case. Therefore we might wonder if simply initialising the human clone model with the RL clone weights and continuing training from that point onwards would be a good idea. The resulting clone will be called *BC TL Init*.

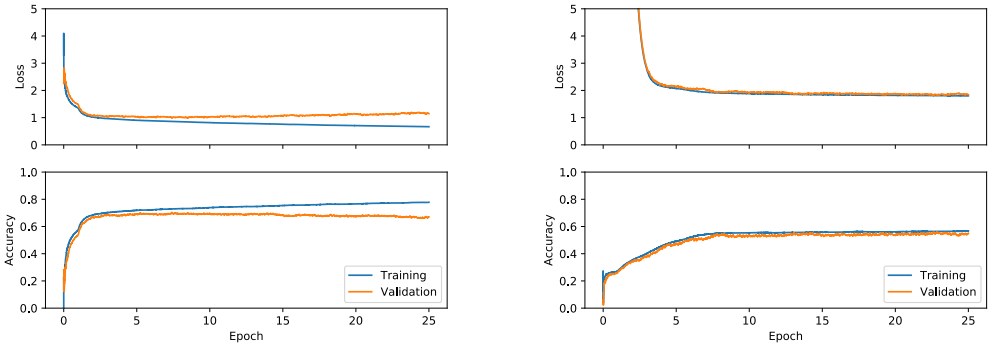
We will also test the more common technique of only copying the weights from the non-top, i.e. apart from the classification layers. Recall our model architecture (Figure 2.2c). About all of the weights are contained in the fully connected layers, in particular in the first one. Since we want to recover at least some weights, we will therefore consider only the last two fully connected layers (and the dropout layer in between) as the classification top. Thus all other weights will be copied over from the RL agent clone of 2000 episodes, and these last layers will be retrained from scratch. We will first freeze the RL weights when training for 25 epochs, resulting in *BC TL New top*. Afterwards we will unfreeze them and finetune for another 25 epochs, yielding the clone *BC TL New top Finetune*.

An important note to make is that the choice of number of frames to stack affects the concrete architecture, altering the shape of the inputs. Therefore, in all the transfer learning models we will have to stack only two frames, as we did for the RL clones.

The loss and accuracy learning curves are shown in Figure 4.1, the final validation losses and accuracies in Table 4.1. A first thing to note is that none of the transfer learning methods were able to improve upon the model trained from scratch. It is also clear that the RL clone model is not very compatible with the human player, as evidenced by the extremely high losses at the start of training, or just in general for the model where only the top was not frozen (BC TL New top). Seeing how much the finetuning helped, the representations learnt by the convolutional part of the model for the RL agent do not seem too useful in this situation.

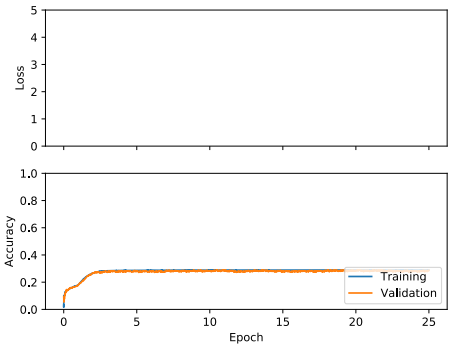
Finally, note that the model trained from scratch started overfitting slightly, the

validation loss increasing from a minimum of 0.9828 to 1.149 and the validation accuracy decreasing from a maximum of 70.37% to 67.00% at the end. As in all previous training processes, early stopping would have helped in reducing the computational load, but now additionally also in combatting overfitting.

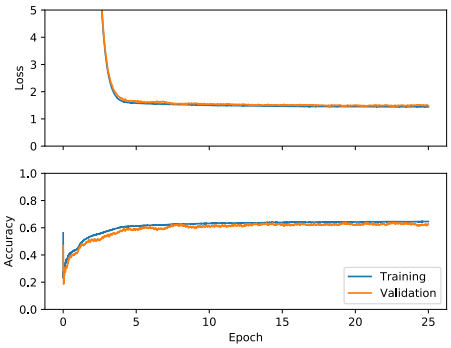


(a) For BC Scratch.

(b) For BC TL Init.



(c) For BC TL New top.



(d) For BC TL New top Finetune.

Figure 4.1: Loss and accuracy learning curves for the behavioural clones of the human agent. The minimum (training as well as validation) loss in case (c) was about 2331(!), which explains why no curve is visible. As always, these curves were smoothed using a backwards simple moving average with epoch window size.

Clone name	BC Scratch	BC TL Init	BC TL New top	BC TL New top Finetune
Validation loss	1.149	1.844	2331	1.482
Validation accuracy	67.00%	54.97%	28.62%	62.70%

Table 4.1: Validation loss and accuracy of the behavioural clones (BCs) of the human player.

4.2 Evaluating the clones

Since we cannot sample the human agent, comparisons of local distributions are out of the question. But global action distributions can still be obtained.

4.2.1 Global distribution

The distribution collection process will necessarily be different from that in the previous chapters, as there we obtained global distributions by aggregating local distributions. Here we will let all agents play for a number of episodes and collect the chosen actions. For the human agent we have 26 test episodes for this purpose, for the other agents we can run 26 new episodes. The action distribution histograms are shown in Figure 4.2.

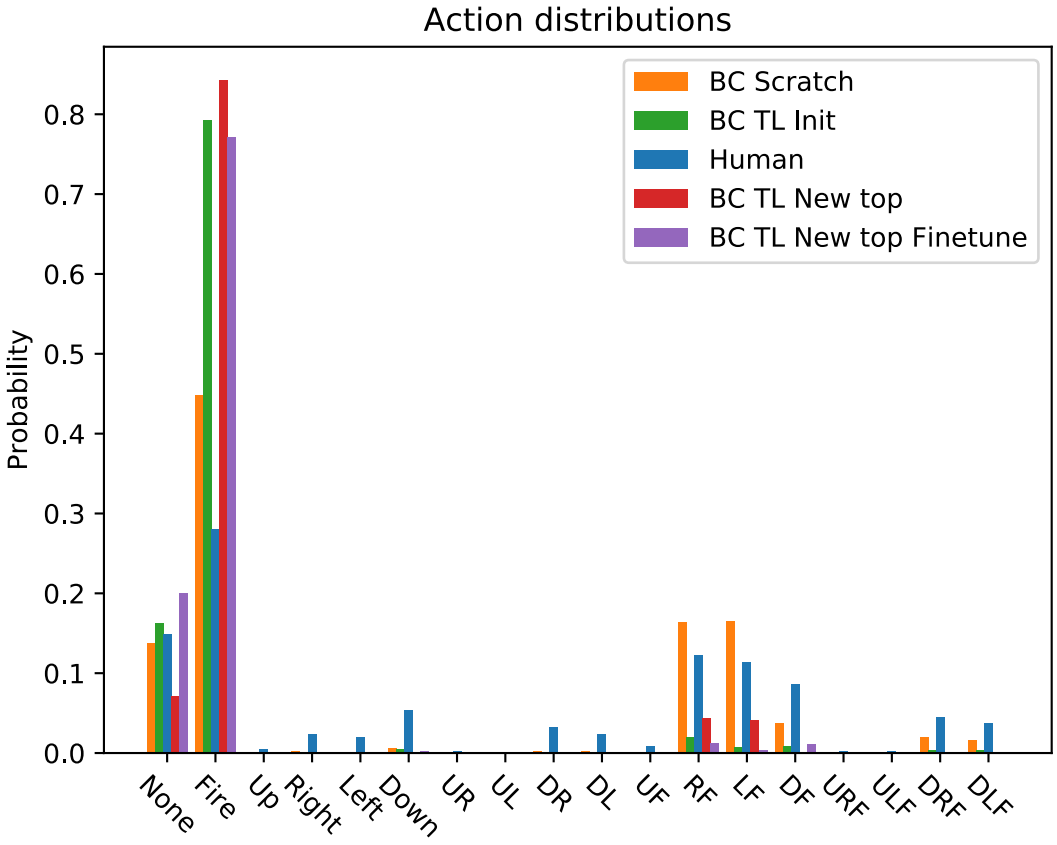


Figure 4.2: Histograms of the global action distributions of the human player and its various behavioural clones.

Clearly none of the clones have been able to capture the action distribution. This is also evidenced by the L^1-distances, listed in Table 4.2. For a point of comparison, the distance between the global action distribution of the human agent and a uniform distribution is 0.9448. This is in fact better (lower) than all of the clones using

transfer learning. Again, the model trained from scratch using only human episodes performs best, although it still leaves a lot to be desired.

Clone name	BC Scratch	BC TL Init	BC TL New top	BC TL New top Finetune
Global L^1-distance	0.5211	1.054	1.126	1.087

Table 4.2: Evaluation of the behavioural clones of the human agent in terms of capturing the human player's global action distribution, as measured by the L^1-distance between the human's global distribution and the clones'. Lower is better.

4.2.2 Time-score scatter plots

The time-score scatter plots can be found in Figure 4.3, the quantitative results in terms of the FGKLD and KSp in Table 4.3.

Clone name	BC Scratch	BC TL Init	BC TL New top	BC TL New top Finetune
FGKLD	24.97	303.1	834.5	186.8
KSp	0	0	0	0

Table 4.3: Evaluation of the behavioural clones of the human agent in terms of capturing the human player's time-score scatter plot, as measured by the fitted Gaussian KL divergence (FGKLD, lower is better), and the p-value for the two-sample Kolmogorov-Smirnov test for equal (two-dimensional) distributions (KSp, higher is better). KSp values of exactly zero are due to limited numerical precision.

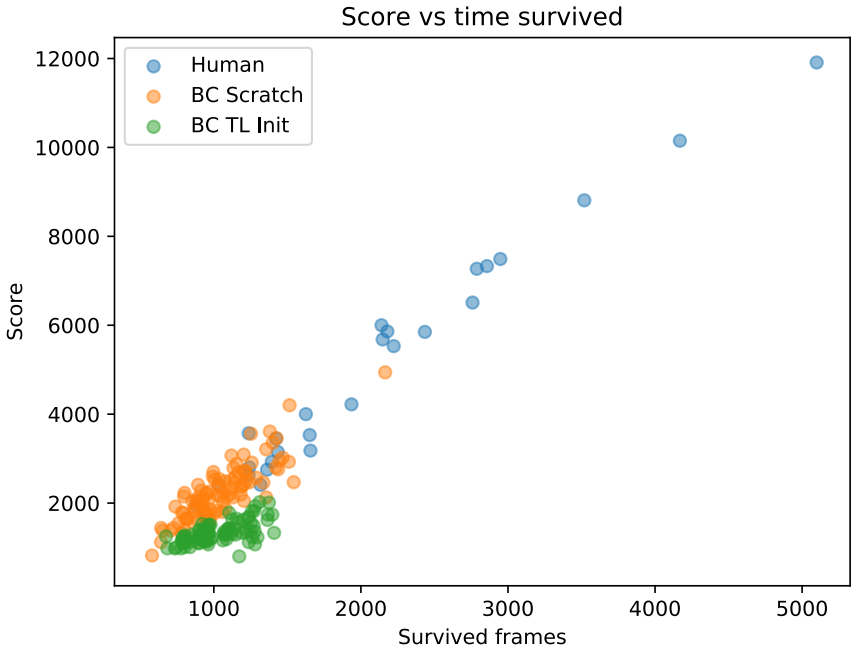
Clearly the KSp is not very informative. The FGKLD at least gives an ordering, namely: the model trained from scratch outperforms all others by a wide margin, the finetuned transfer learning model performs the second best, but much worse compared to the scratch model. But as can be seen in Figure 4.1a also this clone is far from optimal. Note however that it is the only clone that seems to have correctly captured the correlation between the number of survived frames (time) and score. The other clones just accumulate fewer points in the same time span.

4.3 Explaining the poor performance

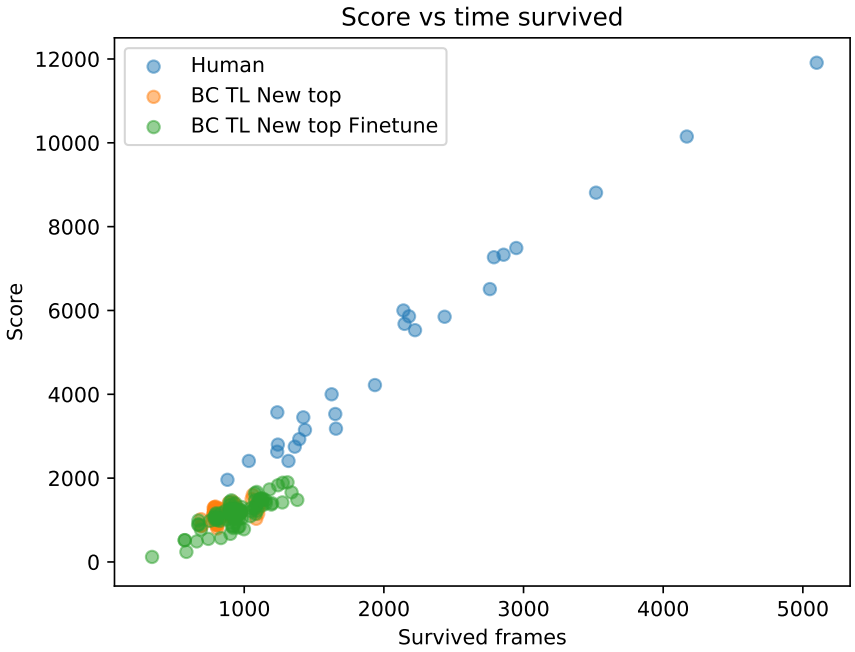
Note in particular that these results are far worse than the ones for the RL agent, or even for the random agent. We see two reasons for this. Firstly, humans improve gradually over time, updating their implicit policy for the better. We explicitly made sure this did not happen with both base computer agents. Secondly, the human player plays more inconsistently than the others, as we will quantify below.

4.3.1 Human player improvements

Practice makes perfect. By playing more, humans will (normally) get better. Certainly when doing something for the first few times the gains should be considerable.



(a) Of the clones initialised from scratch (BC Scratch) and from the weights of the RL clone of 2000 *episodes (BC TL Init).*



(b) Of the clones using transfer learning via the weights of the RL clone of 2000 *episodes, where the top was reset, pre- (BC TL New top) and post-finetuning (BC TL New top Finetune).*

Figure 4.3: Time-score scatter plots of the human agent and its behavioural clones. For the points from the human agent we used the 26 *test episodes. The clones played for* 100 *episodes.*

The human episodes were collected from the author, who had almost no prior experience with the game. In Figure 4.4 we can clearly see that he improves over time.

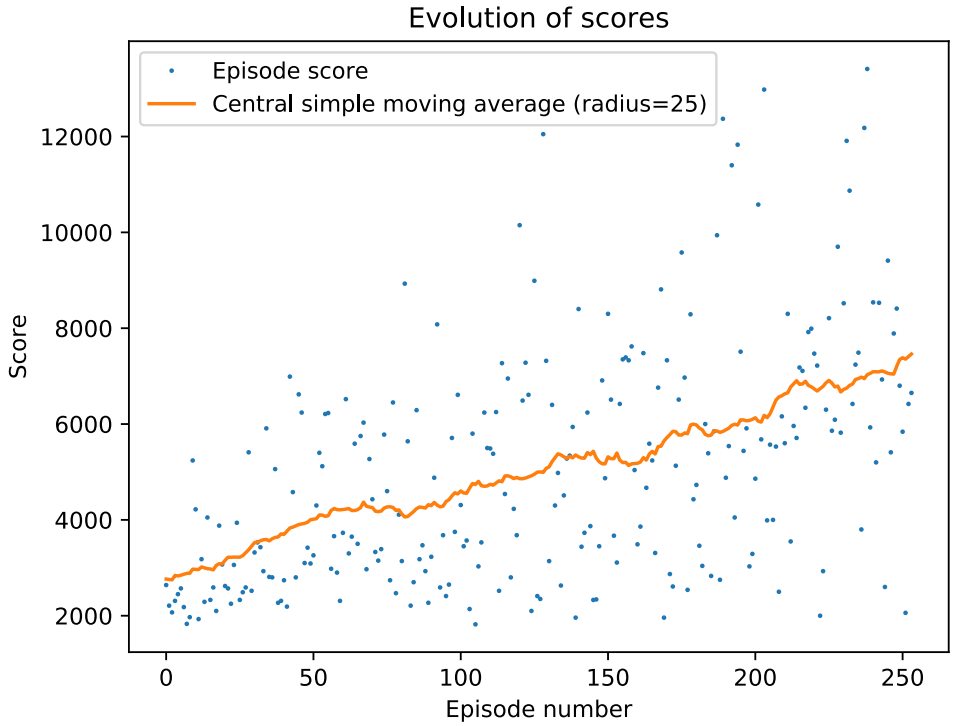


Figure 4.4: Improvement of the scores of the human player over time. The orange line is the central simple moving average of the sequence of scores (blue dots), for a smoothing radius of 25.[3]

We ignored this improving trend and tried to even it out by making sure that our training (as well as validation and testing) episodes were selected in a random order. If we could find a way to effectively model this improvement, we would presumably be able to capture the (improving) playstyle(s) better. Learning from improving supervisors is not completely unexplored territory, as there do exist algorithms such as *Follow the improving teacher* (FIT) [BTL+19]. But this is an on-policy imitation

[3]For a radius $r \in \mathbb{N} \setminus \{0\}$ the central simple moving average of a sequence $(x_n)_{n=1}^k$ of length $k \in \mathbb{N} \setminus \{0\}$ is another sequence $(y_n)_{n=1}^k$ where

$$y_n = \operatorname{avg}\{x_{n-r}, x_{n-r+1}, \ldots, x_{n-1}, x_n, x_{n+1}, \ldots, x_{n+r-1}, x_{n+r}\},$$

that is, we replace x_n by the average of its $2r+1$ 'neighbours' (including x_n itself). At least when $r < n \leq k - r$. In the boundary cases for $1 \leq n \leq r$ or $k - r < n \leq k$ we use

$$y_n = \operatorname{avg}\{x_i \mid \max\{1, n-r\} \leq i \leq \min\{k, n+r\}\},$$

where we replace x_n by the *existing* neighbours within a radius of r.

learning algorithm, requiring the supervisor (i.e. human in our case) to label on request. We do not know of any algorithms for off-policy imitation learning (such as behavioural cloning) dealing with improving supervisors which cannot later be sampled.

4.4 Quantifying the variability of the base agents

The RL agent more closely follows a fixed strategy than our human player. The same is true for the random agent (since playing randomly is indeed just a strategy). Also in terms of areas the agent's ship has visited, the difference is large. For the RL agent this is again because it does not deviate too much from a fixed trajectory. For the random agent it is because the agent tends to move mostly in a straight line, the small horizontal movements cancelling each other out. At least in the beginning of episodes this makes the random agent's position quite predictable. In particular, it is extremely improbable that the random agent would immediately move completely to the left at the start of the episode. For the human agent this is actually quite common.

We will now quantify this variability. First consider the relationship between mean and variance. The mean μ of a numerical data set X is precisely that constant c which minimises the total (or mean) squared error between the data points and this constant:

$$\mu = \operatorname*{argmin}_c \sum_{x \in X} \|x - c\|_2^2 .$$

The resulting mean squared error is precisely the (biased sample) variance σ^2:

$$\sigma^2 = \operatorname{avg} \left\{ \|x - \mu\|_2^2 \;\middle|\; x \in X \right\} .$$

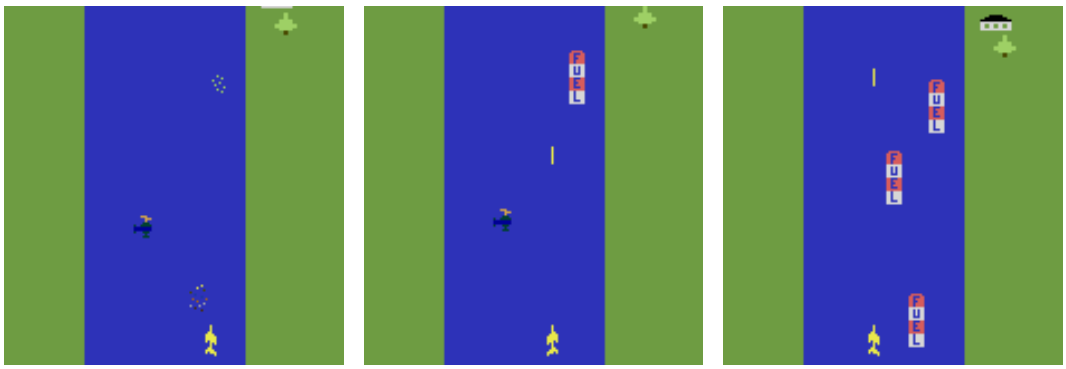
To abstract a bit, we had a distance measure (squared error) and sought the centroid (the mean) by looking at all possible vectors and taking the one for which the total distance of the data points to it is minimal. The corresponding mean distance is then a measure for the variability. We can also use medoids, i.e. only consider elements in the data itself for the role of centroid.

Therefore, to find a good variability measure, we only need to find a relevant distance measure. As data set X we will take a set of 60th frames from episodes of a River Raid agent. This is close to the beginning of the episodes, so that these frames should still be quite similar, but after the moment where the agent's actions can have a noticeable impact on the location of the playable aircraft. We could use more complicated distance measures such as cross-correlation (using sliding windows), but by sliding we would actually lose valuable information. Indeed, sliding vertically means we ignore knowledge about how far the agent has made it in the game, and horizontally about the location of the aircraft on the x-axis. Of course there would also be practical considerations such as which filter we should use to slide (e.g. a crop of the frame), and if and how we should pad.

We could also simply again use the (squared) Euclidian distance. But that might give issues with the colours. For example, if our yellow aircraft moves slightly

to the left, yellow pixels will have become blue (now being water) and vice versa. But destroying a (black) helicopter would change the colours less. This is not the desired behaviour for our distance measure. Therefore we just consider the number of different pixels between two images. Adding 10 points to the score might still make a large pixelwise difference, so we crop the image to only show the playable area, ignoring the score, number of lives and fuel meter.

Now that we have decided on a distance measure, we can look for a medoid as described above. We will simply try out every single frame $x \in X$ as a candidate-medoid, compute the total pixelwise difference between the other frames and this one, and finally take that frame achieving the minimum total distance. We extracted these medoids for the RL agent, random agent and human agent (Figure 4.5). These also give a visual indication of the agents' playstyle.[4] For the human player we used the 26 60th frames from the test episodes. For the other base agents we used 100 newly sampled episodes and extracted the 60th frame of each one.



(a) For the RL agent. *(b) For the random agent.* *(c) For the human agent.*

Figure 4.5: The medoid 60th frames for the RL, random and human agents.

In Table 4.4 we show the mean number of different pixels with respect to the medoids for the three base agents, not only for the 60th frame, but also for the 50th, 55th, 65th and 70th.[5] This way we can check that our somewhat arbitrary choice of 60 was justified.

We see that the RL agent always plays the most consistent (according to this measure), followed by the random agent and finally the human agent. This conforms to our expectations.

To explain why a large variability makes cloning harder, consider the extreme opposite situation where the variability would be zero. In this case the agent would always perform the same actions in the same states. Since at first River Raid is completely deterministic, this means that the sequence of agent actions is the same across all episodes (at least, until the first moving enemy is encountered a bit later

[4] Note for example that only the human agent has destroyed the enemy helicopter.

[5] From each of the 100 episodes per agent we extracted these five frames. Therefore each time these frames were not taken independently of each other.

	RL agent	Random agent	Human agent
50th frame	455.9	776.4	994.5
55th frame	459.3	613.3	730.1
60th frame	519.5	595.4	757.4
65th frame	562.9	660.2	667.2
70th frame	647.3	707.7	718.0

Table 4.4: Mean number of different pixels with respect to the medoids for the three base agents and for different choices of fixed frame number.

on).[6] We could then perfectly clone the agent by simply memorising the sequence of actions. Having a large variability means the sequences of frames will be wildly different across episodes. Obviously the same then holds for the sequences of actions. Simple memorisation is no longer a viable strategy.

4.5 Conclusion

Behaviourally cloning a human agent turns out to be very hard. We were not able to capture the action distributions, nor were we able to get similar time-score scatter plots. There are a number of reasons for these poor results. Firstly, a practical reason: we simply cannot get the same amounts of data as we used for the RL agent. The amount of training episodes we had to use was lower than the minimum we recommended for the RL agent. Our attempts at transfer learning from a clone from the RL agent were unsuccessful. Secondly, humans improve over time, something our current models cannot capture. Thirdly, human agents play considerably less predictably than the RL agent, or even the random agent. Of course, it is harder (fundamentally unattainable) to predict the random agent's next move. But in the grand scheme of things it tends to play somewhat consistently, due to its actions cancelling out. This is not the case for the human agent, where most actions are intentional and thus unlikely to be immediately undone. We were able to verify this theoretical reasoning empirically by quantifying the variability in terms of the mean number of different pixels at a certain time with respect to the medoid at that time.

[6] Enemies at the beginning of the game are stationary.

Chapter 5

Conclusion

We started from the very general question of whether it would be possible to satisfactorily behaviourally clone an agent playing a simple video game. The answer is obviously yes, also for the particular case of River Raid, but it does strongly depend on the type of agent in question. Indeed, cloning a simple reinforcement learning agent turns out to be doable. This was confirmed both qualitatively and quantitatively, although only to a lesser degree. We were able to copy action distributions quite accurately (in terms of L^1-distance) and also on a less granular level the scores and survival times were (sometimes) comparable (as measured by the fitted Gaussian KL divergence, and Kolmogorov-Smirnov two-sample test p-values). As for the question of how many episodes we need to clone our RL agent well, the answer is less clear-cut. As was expected using very little training data yielded poor results. But even the use of 10 training episodes was able to give good results for the time-score scatter plots. However, based primarily on the capturing of action distributions, we recommended a minimum of 250 training episodes. The results were far from monotonic, though: increasing the number of training episodes was no guarantee for better cloning performance.

What about inherent unpredictability? Does this make behavioural cloning harder? The answer is yes, considerably harder, at least according to the metrics we considered. The distribution is harder to copy, and our clone seemed to die significantly faster in game. However, qualitatively when observing both the random agent and its clone in action, we found it difficult to distinguish between them. They both simply appeared erratic. The fact that one performed worse at the game was not noticeable when not explicitly paying attention to this facet.

For our most interesting and relevant research question "*Is cloning a human agent feasible?*" our results were negative. Of course this by no means implies it is impossible, just that we were not able to overcome the many inherent difficulties in this case. Our attempts at solving the issue of limited availability of training data by leveraging the power of transfer learning from the RL agent setting proved fruitless. Making do with the (admittedly not extremely small) training set proved the most effective. But even in this case there are at least two other major hurdles we did not overcome. We did not tackle the problem of improving players. And

somewhat related to this, we showed that human agents on the macro-scale play very unpredictably, even more so than a random agent uniformly sampling actions.

Although not an explicit research question per se, we also established in all cases that we need not train the cloning models for very long. Since the learning curves were always roughly convex for the loss (and concave for the accuracy), early stopping is advisable, not only to ease the computational load,[1] but also to reduce the risk of overfitting.

5.1 Outlook

There are still many questions that remain open. Although we conjecture that the lack of episodes for a human player will be problematic, we did not explicitly check this. The results for cloning the RL agent were mixed, so perhaps (though we judge it to be unlikely) we might still get comparable results with fewer episodes from the human player. Therefore, we could repeat the tests but now with clones trained on fewer episodes.

In our approach we infused the models' inputs with some local history by stacking consecutive frames. We noticed that this significantly boosted performance. It would then certainly be interesting to check whether recurrent neural networks might be able to achieve better results.

We did not model the natural learning of humans. In fact, we explicitly made sure that the RL agent we wanted to clone was not allowed to learn. Obviously that is not possible, nor desirable for human players. So in this situation there is no way around it: we should take the fact that humans improve while playing games into account. Mimicking the structure of this thesis, we could first again try to capture the improvements in RL agents.[2] Perhaps using reinforcement learning on a clone might help to push it in the same direction as the base agent.

Finally, the quantitative measures we developed for checking whether a clone is decent at imitating the base agent are not perfect. Indeed, we noted that the FGKLD and KSp are conservative in the sense that even if they indicate that a clone is poor, visually we might not be able to distinguish between it and the base agent. We might be able to use machine learning to obtain better measures, although this would likely require unrealistically large amounts of training data of human judgements of whether episodes have come from the same player. Alternatively we could just repeatedly present two episodes to a classifier and again ask it to make a binary prediction of whether the two episodes come from the same agent, where we e.g. sample the episodes from the RL agent and a clone. However, we expect this classifier will start to pick up on properties a human might not notice. That

[1] For example, training the behavioural clone of the RL agent with 2000 training episodes for 25 epochs took over 15 hours on an Nvidia Tesla P100 graphics card from the VSC servers. The hyperparameter tuning for the human agent was interrupted after 100 hours, having only finished 44 of the 50 planned runs.

[2] In fact, we attempted to do this, but had trouble in getting the RL agent to actually improve, as explained in footnote 4 of chapter 2. Due to time constraints we had to give up on this research direction for this thesis.

said, it might be possible to incorporate this judgement into the loss function while training clones, in order to obtain better quality clones. To make this approach feasible for human cloning, we would need to have access to many more episodes than we currently do.

Bibliography

[ACQN07] Pieter Abbeel, Adam Coates, Morgan Quigley, and Andrew Y Ng. An application of reinforcement learning to aerobatic helicopter flight. In *Advances in neural information processing systems*, pages 1–8, 2007.

[AN04] Pieter Abbeel and Andrew Y Ng. Apprenticeship learning via inverse reinforcement learning. In *Proceedings of the twenty-first international conference on Machine learning*, page 1, 2004.

[BBBK11] James S Bergstra, Rémi Bardenet, Yoshua Bengio, and Balázs Kégl. Algorithms for hyper-parameter optimization. In *Advances in neural information processing systems*, pages 2546–2554, 2011.

[BBC+19] Christopher Berner, Greg Brockman, Brooke Chan, Vicki Cheung, Przemysław Dębiak, Christy Dennison, David Farhi, Quirin Fischer, Shariq Hashme, Chris Hesse, et al. Dota 2 with large scale deep reinforcement learning. *arXiv preprint arXiv:1912.06680*, 2019.

[BCP+16] Greg Brockman, Vicki Cheung, Ludwig Pettersson, Jonas Schneider, John Schulman, Jie Tang, and Wojciech Zaremba. OpenAI Gym. *arXiv preprint arXiv:1606.01540*, 2016.

[BHD13] Enda Barrett, Enda Howley, and Jim Duggan. Applying reinforcement learning towards automating resource allocation and application scalability in the cloud. *Concurrency and Computation: Practice and Experience*, 25(12):1656–1674, 2013.

[BNVB13] Marc G Bellemare, Yavar Naddaf, Joel Veness, and Michael Bowling. The arcade learning environment: An evaluation platform for general agents. *Journal of Artificial Intelligence Research*, 47:253–279, 2013.

[BR19] Alex Bäuerle and Timo Ropinski. Net2vis: Transforming deep convolutional networks into publication-ready visualizations. *arXiv preprint arXiv:1902.04394*, 2019.

[BS95] Michael Bain and Claude Sammut. A framework for behavioural cloning. In *Machine Intelligence 15*, pages 103–129, 1995.

[BTL+19] Ashwin Balakrishna, Brijen Thananjeyan, Jonathan Lee, Arsh Zahed, Felix Li, Joseph E Gonzalez, and Ken Goldberg. On-policy imitation learning from an improving supervisor. 2019.

[BYC13] James Bergstra, Daniel Yamins, and David Daniel Cox. Making a science of model search: Hyperparameter optimization in hundreds of dimensions for vision architectures. 2013.

[C+15] François Chollet et al. Keras. https://keras.io, 2015.

[DDS+09] J. Deng, W. Dong, R. Socher, L.-J. Li, K. Li, and L. Fei-Fei. ImageNet: A large-scale hierarchical image database. In *CVPR09*, 2009.

[Doz16] Timothy Dozat. Incorporating Nesterov momentum into Adam. 2016.

[FF87] Giovanni Fasano and Alberto Franceschini. A multidimensional version of the Kolmogorov–Smirnov test. *Monthly Notices of the Royal Astronomical Society*, 225(1):155–170, 1987.

[GG16] Yarin Gal and Zoubin Ghahramani. Dropout as a Bayesian approximation: Representing model uncertainty in deep learning. In *international conference on machine learning*, pages 1050–1059, 2016.

[HLTY14] Christoffer Holmgard, Antonios Liapis, Julian Togelius, and Georgios N Yannakakis. Generative agents for player decision modeling in games. 2014.

[Hol19] Kristian Holsheimer. keras-gym. https://github.com/KristianHolsheimer/keras-gym, 2019.

[IDMM+17] Tadanobu Inoue, Giovanni De Magistris, Asim Munawar, Tsuyoshi Yokoya, and Ryuki Tachibana. Deep reinforcement learning for high precision assembly tasks. In *2017 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, pages 819–825. IEEE, 2017.

[KB14] Diederik P Kingma and Jimmy Ba. Adam: A method for stochastic optimization. *arXiv preprint arXiv:1412.6980*, 2014.

[KLR+13] Elizaveta Kuznetsova, Yan-Fu Li, Carlos Ruiz, Enrico Zio, Graham Ault, and Keith Bell. Reinforcement learning for microgrid energy management. *Energy*, 59:133–146, 2013.

[MKS+13] Volodymyr Mnih, Koray Kavukcuoglu, David Silver, Alex Graves, Ioannis Antonoglou, Daan Wierstra, and Martin Riedmiller. Playing Atari with deep reinforcement learning. *arXiv preprint arXiv:1312.5602*, 2013.

[MKS+15] Volodymyr Mnih, Koray Kavukcuoglu, David Silver, Andrei A Rusu, Joel Veness, Marc G Bellemare, Alex Graves, Martin Riedmiller, Andreas K Fidjeland, Georg Ostrovski, et al. Human-level control through deep reinforcement learning. *Nature*, 518(7540):529–533, 2015.

[PSM18] Johannes Pfau, Jan David Smeddinck, and Rainer Malaka. Towards deep player behavior models in MMORPGs. In *Proceedings of the 2018 Annual Symposium on Computer-Human Interaction in Play*, pages 381–392, 2018.

[SB18] Richard S Sutton and Andrew G Barto. *Reinforcement learning: An introduction*. MIT press, 2018.

[Sch97] Stefan Schaal. Learning from demonstration. In *Advances in neural information processing systems*, pages 1040–1046, 1997.

[SH08] Bhuman Soni and Philip Hingston. Bots trained to play like a human are more fun. In *2008 IEEE International Joint Conference on Neural Networks (IEEE World Congress on Computational Intelligence)*, pages 363–369. IEEE, 2008.

[SHK+14] Nitish Srivastava, Geoffrey Hinton, Alex Krizhevsky, Ilya Sutskever, and Ruslan Salakhutdinov. Dropout: a simple way to prevent neural networks from overfitting. *The journal of machine learning research*, 15(1):1929–1958, 2014.

[SMDH13] Ilya Sutskever, James Martens, George Dahl, and Geoffrey Hinton. On the importance of initialization and momentum in deep learning. In *International conference on machine learning*, pages 1139–1147, 2013.

[SWD+17] John Schulman, Filip Wolski, Prafulla Dhariwal, Alec Radford, and Oleg Klimov. Proximal policy optimization algorithms. *arXiv preprint arXiv:1707.06347*, 2017.

[SZ14] Karen Simonyan and Andrew Zisserman. Very deep convolutional networks for large-scale image recognition. *arXiv preprint arXiv:1409.1556*, 2014.

[TWS18] Faraz Torabi, Garrett Warnell, and Peter Stone. Behavioral cloning from observation. *arXiv preprint arXiv:1805.01954*, 2018.

[VBC+19] Oriol Vinyals, Igor Babuschkin, Junyoung Chung, Michael Mathieu, Max Jaderberg, Wojtek Czarnecki, Andrew Dudzik, Aja Huang, Petko Georgiev, Richard Powell, Timo Ewalds, Dan Horgan, Manuel Kroiss, Ivo Danihelka, John Agapiou, Junhyuk Oh, Valentin Dalibard, David Choi, Laurent Sifre, Yury Sulsky, Sasha Vezhnevets, James Molloy, Trevor Cai, David Budden, Tom Paine, Caglar Gulcehre, Ziyu Wang, Tobias Pfaff, Toby Pohlen, Dani Yogatama, Julia Cohen, Katrina McKinney, Oliver Smith, Tom Schaul, Timothy Lillicrap, Chris Apps, Koray Kavukcuoglu, Demis Hassabis, and David Silver. AlphaStar: Mastering the real-time strategy game StarCraft II. https://deepmind.com/blog/alphastar-mastering-real-time-strategy-game-starcraft-ii/, 2019.

[WU05] Yi-Chi Wang and John M Usher. Application of reinforcement learning for agent-based production scheduling. *Engineering Applications of Artificial Intelligence*, 18(1):73–82, 2005.