Learning to Play Starcraft II by Mimicking the Pros: Style Extraction in Adversarial Games

Chunya Hua
chunya25@stanford.edu

David Rendleman
rendled@stanford.edu

Abstract

Can the style of a player's gameplay be extracted in the same way that the style of a painting or song be extracted from its content? We introduce a dataset of 956 professional de-anonymized Starcraft II games played by 86 players over the course of three tournaments in early 2018. We also present an approach to discriminating player identity as the first step of our ultimate goal, to transfer the gaming styles of pro-players to an agent. We achieve good accuracy on race classification (f1=0.92) and introduce baseline performance for name classification (Top-3 f1=0.48) on this dataset.

1 Introduction

Style in competitive play is a very nebulous and difficult to define concept compared to how it manifests in music or painting, but it is arguably still the most prominent aspect of a player's individuality. For lovers of the competitive real-time strategy game Starcraft, just look at Lee "Flash" Young-ho's rigid and mechanical macro style, Kim "Bisu" Taek-yong's surgical precision, or Lee Jae-dong's overwhelming "Sauron zerg" onslaught! Starcraft's new player accessibility and high skill-ceiling have made it a spectator sport for millions of fans around the globe, but nowhere more than South Korea.

The goal of this project is to demonstrate that style as a concept may be extracted from the gameplay of a player. We've chosen a domain where human performance still far exceeds that of the best AI solutions: we aim to apply style transfer to the domain of algorithmic game playing in Starcraft II. Starcraft II itself is a partially observable, semi-stochastic adversarial strategy game with a continuous state and action space released back in 2010, and has gained popularity as one of the most demanding computer games in terms of player speed and tactics required. Players use one of three armies known as races (Terran, Zerg, Protoss) to attempt to wrest control of a map and destroy all of an enemy's buildings. The three armies are asymmetric in their abilities, requiring different strategies to play. The Zerg army is cost efficient, numerous, but individually weak - with players needing to take on enemies in open areas where they can fully exploit their greater numbers. The Terran army is very versatile but easy to destroy, requiring favorable engagements to win battles. The Protoss army is the most powerful but also the most expensive, requiring players to preserve their numbers over many engagements.

2 Related work

Starcraft II has received a lot of attention in the reinforcement learning space due to strong developer support from producer Blizzard-Activision [5], availability of a machine-learning API developed by deepmind [4], open-ended AI competitions [7], and availability of large amounts of human data [8]. Machine learning in Starcraft II isn't new, but learned methods have yet to challenge the world's best

players.
Deepmind's initial approach [2] applied the same ML algorithm used in playing Atari games to either the raw video output or a set of human-engineered spatial features (henceforth the screen and minimap feature maps) produced by the game, but neither approach reached much more than basic performance. [2] explains that the game's action space is large enough that even trivial operations like collecting resources is hard to master. As a baseline, [2] used supervised learning to regress the action performed for every state from a dataset of anonymized high-level ranked games. This approach outperformed Deepmind's RL-based models, but was still unable to match the performance of the players it was trained with. Learning the average action taken by human players in a given scenario meant that some basic behavior would be repeated ad-nauseum because it exists as a component in many strategies, but not that any higher level strategy would actually be learned.
The authors of [3] extend on this by introducing a hierarchy of non-spatial features (need for additional territory, progress toward specific technologies, opponent's weaknesses ascertained, etc) to simplify the exploration process, as it is difficult to encode the sequence of low-level actions necessary to implement a coherent strategy. Leveraging this human-engineered hierarchy of behaviors was sufficient to beat the game's built-in rule-based AI.
On the style-transfer side, [1] generalizes the concept of style-transfer to arbitrary problems by formulating style-transfer as a GAN-type unsupervised learning problem, and applying it to imitating chess-playing style. This approach takes as input game-states and attempts to compare the action produced by a generator to actions performed by a target player. A discriminator differentiates between generated and real actions for each game state.

3 Dataset and Features

Data released from Blizzard in [6] for the purposes of machine learning has been anonymized, meaning that only the race and skill level of the player is known at training time but not the player's identity. For this reason, we did not use publically available datasets such as [4], despite containing hundreds of thousands of games. Instead, the WCS Montreal and IEM Katowice 2018 tournaments are used as training data (consisting of 884 total games played by 86 individual players, 5,633,478 training examples, 2.5 TB of data uncompressed), while the 2018 WCS Global Finals (consisting of 72 games played by 16 players, 294,875 training examples) is used as our test set. Tournament play is performed on a set of 7 maps, each requiring different strategies to master. Maps are chosen by the losing player, causing an imbalance towards maps which are harder to defend (the loser wants a quick victory). Due to players ranking higher in the tournament playing more games, a significant dataset imbalance exists toward finalists and semi-finalists, along with the races they play. Data is collected by replaying the game from the vantage point of each player and sampling the game-state once every 8 timesteps (twice a second). We did not constrain episode length, so players which favored long macro-style strategies are overrepresented to those who prefer short micro-oriented strategies.

3.1 Data preprocessing

The Py-SC2 low-level API exposes featuremaps which represent the game-state visible to the agent: the agent's location on the map, any nearby hostile units, explored terrain, available resources, etc. Two main modalities exist: the minimap, which represents a bird's eye view of the play field, and the main screen, which acts as a magnifying glass that the player may use to focus on some particular area of the play field. Since the minimap does not shift perspective with the player's display, it is considered the easier input to learn. Categorical feature maps such as unit type (there are around 200 distinct unit types) and player allegiance (ally, enemy, neutral, etc) are represented as one-hot matrices, whereas real-valued feature maps are normalized from 0..1.

4 Methods

4.1 Dual-pathway network

A dual-pathway network design is used to take both the minimap as well as screen data as input. Since the one-hot matrices used as input can be quite large, Each pathway consists of an encoding

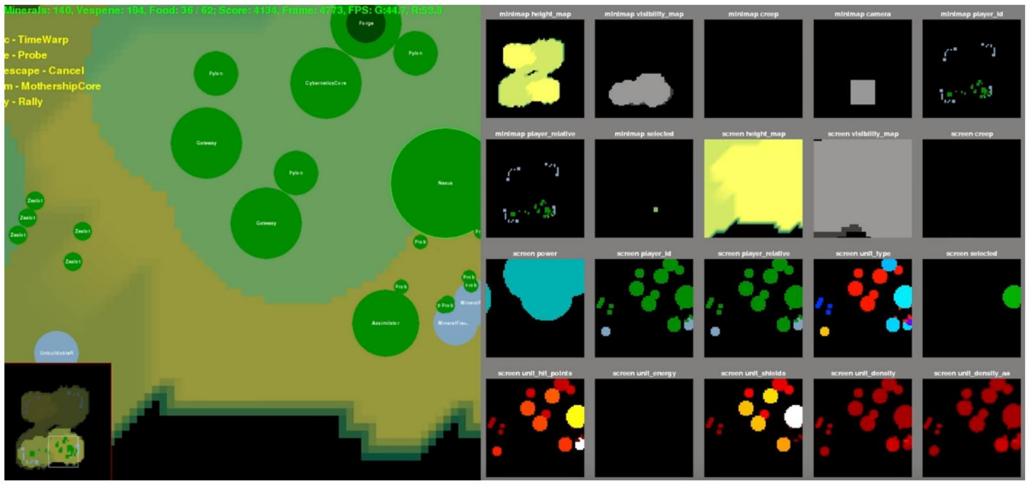

Figure 1: Featuremaps collected on game step 736. Both minimap and Screen featuremaps are used in our experiment.

layer (one-hot input and a 1x1 convolution) to reduce the number of channels that the network sees, along with 4 3x3 convolutions which each down-stride by 2 to reduce the size of the input's spatial dimensions. The two channels are concatenated in a late-fusion pattern and fed into a dense network with a hidden layer of 50 neurons. Dropout and batch normalization are used as regularization and training stability, respectively. The network's two heads, race and player-id, then branch off of this hidden layer - enforcing that these 50 neurons encode all available information about objectives mentioned.

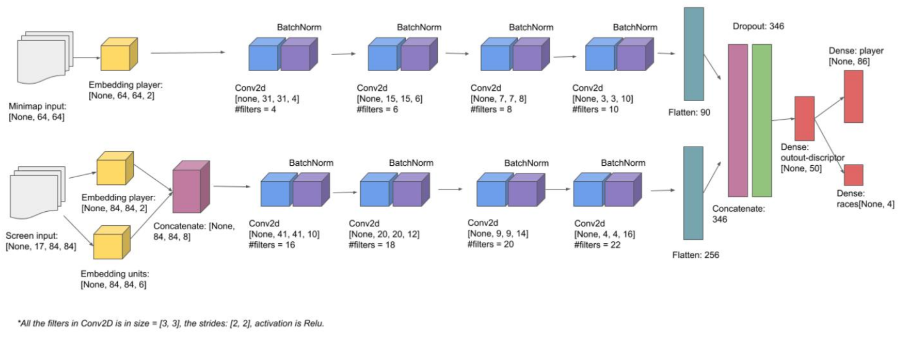
Minimap input: [None, 64, 64]
Embedding player: [None, 64, 64, 2]
BatchNorm
Conv2d [none, 31, 31, 4] #filters = 4
Conv2d [None, 15, 15, 6] #filters = 6
Conv2d [None, 7, 7, 8] #filters = 8
Conv2d [None, 3, 3, 10] #filters = 10
Flatten: 90
Dropout: 346
Dense: player [None, 86]
Dense: outout-discriptor [None, 50]
Dense: races[None, 4]
Concatenate: 346
Screen input: [None, 17, 84, 84]
Embedding player: [None, 84, 84, 2]
Embedding units: [None, 84, 84, 6]
Concatenate: [None, 84, 84, 8]
Conv2d [None, 41, 41, 10] #filters = 16
Conv2d [None, 20, 20, 12] #filters = 18
Conv2d [None, 9, 9, 14] #filters = 20
Conv2d [None, 4, 4, 16] #filters = 22
Flatten: 256
*All the filters in Conv2D is in size = [3, 3], the strides: [2, 2], activation is Relu.

Figure 2: Discriminator architecture.

The architecture employs multi-objective learning to force the network to encode the different facets of the game at the same time. This is done as players typically define their playstyle with only a specific race (f.ex: Nerchio plays only Zerg, InControl plays only Protoss. TheLittleOne plays all three, but each very differently). As a whole the network contains a little over 30k parameters, 72% of which belong to the final dense layers. We are experimenting with even smaller network architectures.

4.2 dataset separation

We separate 90% of the WCS Montreal and IEM Katowice games mentioned in the above setting for training, shuffled and in batches of 2048 examples, and leave a randomly sampled 10% out for validation. 100% of WCS Global Finals games are left out as a test set.

4.3 loss function

The model's objective is to predict the player's ID (N=86) as well as the race of the player's army (N=3) for every discrete time-step using compound loss. Both objectives use the categorical cross-entropy loss function, with a heavier weighting toward player ID than race (0.7 and 0.3). Inverse-frequency sample weighting is used to normalize the effect of more commonly seen players versus less common players.

4.4 hyperparameters

The network is trained using the ADAM optimizer with a learning rate of 5e-4, with L2 regularization (lambda=1e-4) applied to all kernels (but not biases) to address overfitting. Heavy dropout (rate=0.9) is used after concatenation of the minimap and screen pathways to force the network not to rely upon the much more stable minimap input. Any less, and the network is quickly able to learn that players do not often move their haphazardly-placed buildings once they are built, and so can memorize the locations of buildings (and thus the player and race) for each match played in the training set. We train for 20 epochs, taking 9.5 hours on a Geforce 1080 paired with an i7-2700K.

5 Experiments/Results/Discussion

An F1 score of 0.92 is achieved on the top-1 race classification objective, along with an F1 score of 0.48 on the top-3 player identity objective. The training set consists of 39% zerg players, 34% protoss players, and 26% terran players, with random chance on the order of 1.2% per player. Since the test set contains less individual players than the training set, we map all but the 16 players present in the test set to a meta-class of "UNKNOWN" prior to evaluation, and exclude the row from the confusion matrix. We also remove the 8 players with the lowest amount of time played from our stats tables for the sake of brevity. Looking at the identities of players in a lower-dimensional space, we see that the separation (and assumedly, the style) between some players (Has, ShoWTimE) is quite small, whereas other players with very distinctive styles (SpeCial, Serral) have more obvious clusters.
Real-world evaluation was performed by running the discriminator on the WCS grandfinal match between a zerg player Serral and a protoss player Stats. We observe that the network is not able to distinguish early game styles between the players as the actions performed here are largely the same across all games. However, signature strategies such as Serral's "3-base roach-ling all-in", obvious because of the quick third base which feigns a defensive posture but is instead used to launch an assault, produce a much stronger activation.

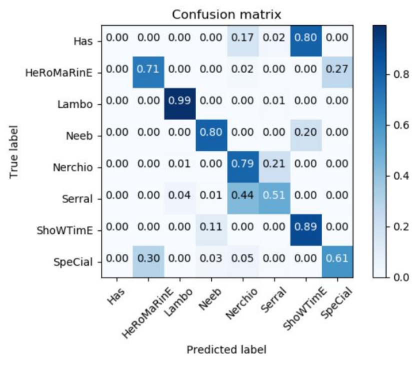

Figure 3: Test-set player classification confusion matrix.

Player	Precision	Recall	F1-Score
Has	0.00	0.00	0.00
HeRoMaRinE	0.31	0.71	0.43
Lambo	0.50	0.99	0.67
Neeb	0.11	0.80	0.19
Nerchio	0.09	0.79	0.16
Serral	0.64	0.51	0.57
ShoWTimE	0.33	0.89	0.48
SpeCial	0.56	0.61	0.58

Table 1: Test-set Top-3 evaluation performance for player classification.

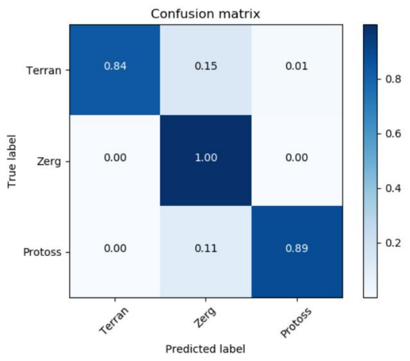

Figure 4: Test-set race classification confusion matrix.

Race	Precision	Recall	F1-Score
Terran	1.00	0.84	0.91
Zerg	0.82	1.00	0.90
Protoss	0.99	0.89	0.93

Table 2: Test-set Top-1 evaluation performance for race classification.

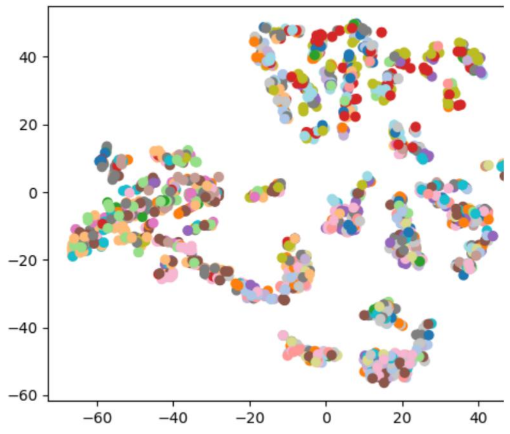

Figure 5: Validation set T-SNE embedding of player identities.

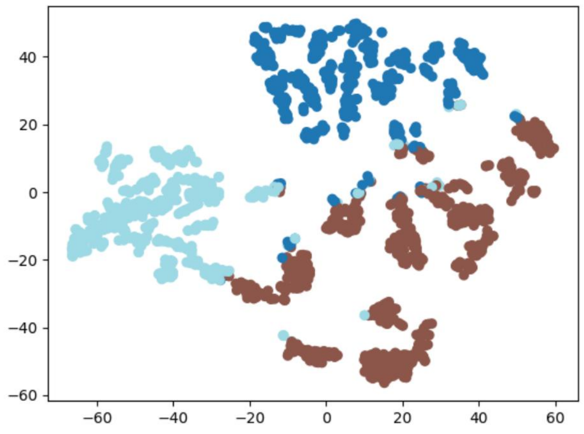

Figure 6: Validation set T-SNE embedding of player races.

6 Conclusion/Future Work

A single timestep of information is unlikely to encode the more interesting characteristics of a player's style, such as their most favored strategies. Can we leverage temporal information via a sequential model to encode strategy as well as style?
Once a network which can appreciably separate players based on strategy is built, we can then close the loop by constructing a generator policy network as described by [1]. This network would receive a penalty whose magnitude was proportional to how far away its playstyle appears from the target player's playstyle.

7 Contributions

Our code is on GitHub at `https://github.com/ubermouser/sc2_styletransfer`
David:

- Dual-pathway model
- Data Collection
- Evaluation framework
- Training framework
- Paper, text

Chunya:

- Temporal Model
- Starcraft II shim
- Cloud training
- Poster
- Paper, figures & tables

References

[1] Chidambaram, Muthuraman, & Yanjun Qi (2017). *Style transfer generative adversarial networks: Learning to play chess differently.* arXiv preprint arXiv:1702.06762.

[2] Vinyals, O., Ewalds, T., Bartunov, S., Georgiev, P., Vezhnevets, A. S., Yeo, M., ... & Quan, J. (2017). *Starcraft ii: A new challenge for reinforcement learning.* arXiv preprint arXiv:1708.04782.

[3] Sun, P., Sun, X., Han, L., Xiong, J., Wang, Q., Li, B., ... & Zhang, T. (2018). *Tstarbots: Defeating the cheating level builtin ai in starcraft ii in the full game.* arXiv preprint arXiv:1809.07193.

[4] Wu, H., Zhang, J., & Huang, K. (2017). *MSC: A Dataset for Macro-Management in StarCraft II.* arXiv preprint arXiv:1710.03131.

[5] *PySC2 - A Starcraft 2 Learning Environment* https://github.com/deepmind/pysc2/

[6] *Starcraft 2 Machine Learning API* https://github.com/Blizzard/s2client-proto

[7] *The Starcraft 2 AI Ladder* https://www.sc2ai.net/

[8] *Starcraft 2 Replay Hosting* http://sc2rep.ru/