Proceedings of the Federated Conference on Computer Science and Information Systems pp. 185–188 DOI: 10.15439/2018F365 ISSN 2300-5963 ACSIS, Vol. 15

A Neural Network Approach to Hearthstone Win Rate Prediction

Jan Jakubik
Wrocław University of Science and Technology
Faculty of Computer Science and Management, Department of Computational Intelligence
Wrocław, Poland
Email: jan.jakubik@pwr.edu.pl

***Abstract*—This paper describes a solution to the AAIA'18 data mining challenge, which concerns prediction of win rates for decks in Hearthstone collectible card game. A neural network model assigning win rate to decks is learned based on maximisation of log probability of observed match results. A representation of deck contents is based on a second network, which performs the role of a dual-task encoder. Two tasks learned by the encoding networks are encoding decks in such a way that the full deck can be reconstructed, and encoding individual cards so that their specific properties can be decoded. Shared representation for these tasks allows the knowledge of individual cards to be taken into account.**

I. Introduction

COMPETETIVE multiplayer video games have been a thriving market in recent years, posing new challenges in areas of artificial intelligence and data analysis. In online game communities, the search for optimal tactics has lead to the development of "metagaming" - an entire layer of strategy related to the knowledge of certain playing styles and changes in their popularity on a community level.

The collectible card game (CCG) landscape, which includes popular and highly profitable games such as Hearthstone and Magic: The Gathering, naturally leads to the development of a particular type of metagame. In these games, players compete using decks which consist of a limited number of cards selected from a much larger pool of cards possible to collect. In a realistic scenario, the flow of knowledge between players leads to rapid development of popular deck types, with players often directly copying known well-performing decks. Any practical approach to applying artificial intelligence methods in such an environment has to consider not only the typical issues of moment-to-moment gameplay but also the metagame information.

In this paper, we explore the problem of predicting win rates for Hearthstone decks within a particular metagame environment. The proposed approach was developed as a submission to the AAIA'18 data mining challenge organised by Silver Bullet Labs and Knowledge Pit as part of the FedCSIS 2018 conference. The paper is arranged as follows: Section II describes the competition and available challenge data, Section III explains the use of external data not provided by organisers. Neural networks which serve as core components of the proposed approach are described in Sections IV and V. Section VI describes how ensembling was used to improve the results and Section VII summarises the conclusions.

II. Competition Description

The competition posed the task of predicting deck win rates for a set of 200 decks based on the record of 300000 games between another set 400 decks. The decks were played by four distinct AI agents, with the AI choice influencing win rates significantly. The goal of the prediction model was to compute win rates for all possible AI-deck pairs in the test dataset. This equates to 800 test samples. The training dataset included:

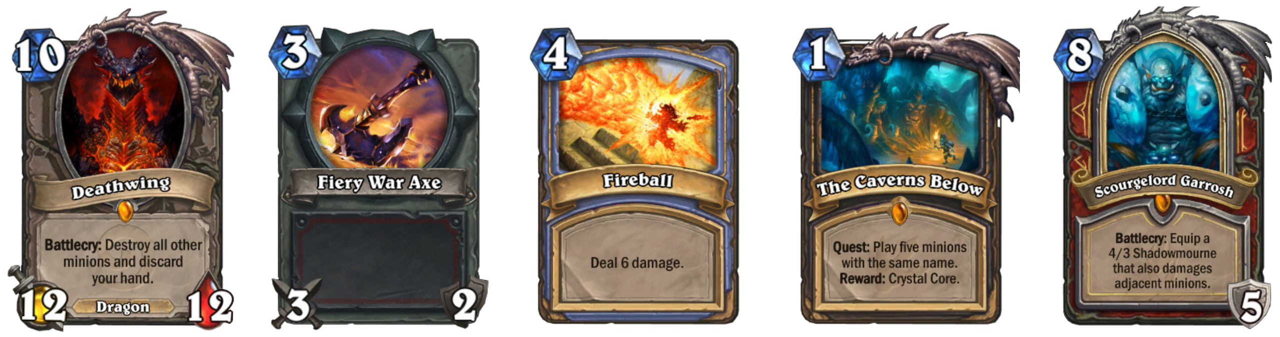
- name and the number of copies for each card present in the deck
- basic description of games including AIs playing, decks being played and the winning player
- detailed description of games - a recorded data of all turns, including actions taken by respective players

The proposed solution uses the basic descriptions of games while utilising an external dataset to represent cards present in the test, but not training set.

During the contest, it was possible to upload solutions and receive an evaluation of RMSE on an evaluation subset of 10% test samples (i.e. 80 AI-deck pairs randomly sampled from all possible 800). This influenced the chosen approach, as "over-tuning" parameters to increase fitness on the evaluation subset of the test set was possible. In fact, the order of top 4 results on the competition leaderboard was reversed in the final results, suggesting multiple submissions including the one described here were over-tuned to some extent. The solution described in this paper was in 2nd place on the competition leaderboard when the submissions closed but placed 3rd in the final evaluation. Possible causes are discussed in the Conclusions section.

III. External Data Use

The proposed solution employs a set of data from the hearthstonejson database [1]. This database contains information on all cards present in the training and test decks. All numerical properties such as life and attack of minions, weapon durability etc. are accounted for. Keywords such as Battlecry, Taunt, Adapt etc. are recognized as binary variables (whether the card has a keyword or does not). Full card text


10
Deathwing
Battlecry: Destroy all other minions and discard your hand.
12
Dragon
12
3
Fiery War Axe
3
2
4
Fireball
Deal 6 damage.
1
The Caverns Below
Quest: Play five minions with the same name. Reward: Crystal Core.
8
Scourgelord Garrosh
Battlecry: Equip a 4/3 Shadowmourne that also damages adjacent minions.
5

(a) Minion (b) Weapon (c) Spell (d) Quest (e) Hero

Fig. 1: Types of cards available in Hearthstone CCG [2]

for every hearthstone card present in the challenge datasets is also available. The dataset also contains certain conditions that need to be met to play the cards (such as "there needs to be a valid minion target").

IV. Encoding Network

To build an encoding of hearthstone decks, first, an encoding of a card is created. The goal of this encoding is to represent cards present in the test, but not training data. There are 330 unique cards in all training decks, and 18 unique cards in test decks that do not appear in training set. These cards include both spells and minions.

Types of cards available in Hearthstone CCG are shown in Fig. 1. Note that minions (a) and weapons (b) have informative numerical properties of attack, health, and durability. However, even when these properties are accounted for, card text can still have a significant effect on the gameplay. In the case of the presented minion, Deathwing, its battlecry ability drastically alters the game state by destroying all minions on the game board. In case of spells (c), the only available information is the card text. Finally, quest (d) and hero (e) cards can alter the overarching game strategy of the entire deck by replacing the player's hero or offering a powerful reward for fulfilling the quest condition. For these cards, even card text does not offer a sufficient explanation. However, no quests or heroes that are not in the training data appear in the test dataset.

Taking this knowledge into account, we build the representation of a card as a concatenation of two vectors. First contains numeric properties, mechanics and other data available from hearthstonejson.com. Each numeric property is encoded as a continuous variable, each keyword is encoded as a binary variable, and all conditions required to play a card are encoded as binary variables. We use all properties available in hearthstonejson descriptions, as long they actually occur in training and test dataset.

The second vector is a word occurrence vector based on the card text. Card text is cleaned by removing punctuation, after which we build a dictionary of all strings that occur in the dataset (separated by whitespace characters) and count their occurrences in each card's text. Word occurrence serves as a simplified way to contain information regarding card function. Terms such as "destroy" or "heal" describe the interactions of a card to some extent, and can be relevant to the AI's ability to efficiently use the card. Without actually simulating the game logic, this is an easy way to represent effects such as the Deathwing battlecry mentioned above.

The representation of decks is then built by a neural network trained on all 600 training and test decks. The encoding network's loss function is defined as a sum of two terms, representing two distinct tasks. First is a standard autoencoder [3], i.e., the loss is based on the network's ability to reconstruct exact input vectors from a lower-dimensional encoding in the hidden layer. Inputs used in optimising this objective are decks from both training and test sets, represented as simple card occurrence vectors - each dimension in the input space represents the count of a particular unique card in the deck.

The second task is to learn an encoding which makes it possible to decode the properties of each card. For this purpose, we use the matrix of card properties C, in which each row represents hearthstonejson information of a single card. The assumption here is that encoding a single card in the same space as full decks can be decoded as card's specific properties. Given shared encoding Enc, autoencoder decoding Dec_1 and card property decoding Dec_2, the combined loss for both tasks can be calculated as:

$$\|Dec_1(Enc(X)) - X\|_F^2 + \|Dec_2(Enc(I)) - C\|_F^2 \tag{1}$$

Where X is the matrix of deck vectors, C is the matrix of card property vectors and I is an identity matrix of a size corresponding to the number of cards. Combining both tasks ensures the network encodes decks in a way that retains full information, but also encodes similar cards in a similar way.

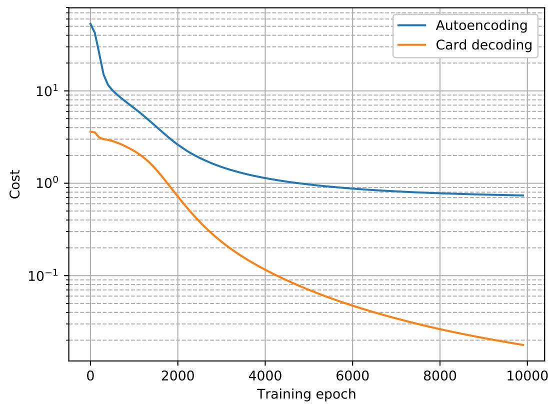


Fig. 2: Training curves of two objectives in the encoding network

The latter is relevant for the cards that do not appear in training data.

Each of the functions: Enc, Dec_1, Dec_2, is learned by a single neural network layer, resulting in a network with one hidden layer and two separate output layers. The encoding layer uses ReLU nonlinearity [4], and decoders are linear layers. The dimensionality of encoding was set to 200 after preliminary tests. The number of training epochs for encoding network was set to 10000, and the network was trained with a gradient-based method Adadelta [5] with $\rho = 0.9$. Using a Theano [6] implementation, training this network takes approximately 3.5 seconds per 1000 epochs on a Nvidia GTX970 GPU.

Training curves of the encoding network are presented in Fig.1. While autoencoding objective reaches a visible plateau, card decoding objective could still be trained beyond 10000 epochs. However, we found this did not improve the prediction network results.

V. Prediction Network

Features encoded by the encoding network are used as and input to the prediction network. Prediction network is a standard feedforward neural network [7] with three hidden layers, respectively 300, 200 and 100 neurons. ReLU nonlinearity is used for activation in hidden layers. The network is optimized with Adadelta, using $\rho = 0.9$. To avoid errors (loss function can return NaN values if the output is outside of $(0, 1)$ interval), final layer activation was implemented as:

$$\sigma(x) = 1 - ReLU(1 - ReLU(x)) \tag{2}$$

However, in practice, outputs do not exceed 1.0 or 0.0 during optimisation if the hyperparameters are tuned for the task, i.e., the bias of final layer is initialised to 0.5 and other parameters to very small values. Therefore, the final layer effectively works as linear.

A basic loss function over all outputs, where o_i denotes the prediction for i-th deck that approximates a known win rate y_i, is defined as:

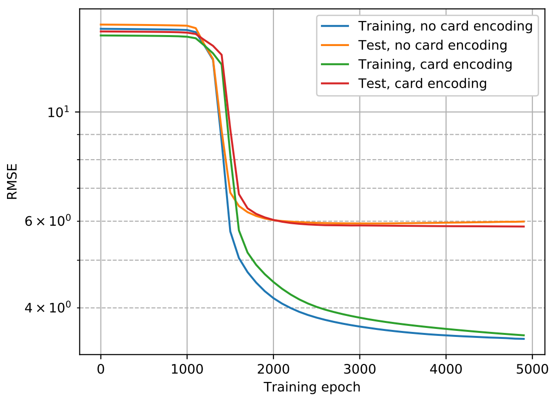


Fig. 3: Training curves of the prediction network, with and without using the card decoding objective in the encoding network

$$\sum_i -d_i(y_i \log(o_i) + (1 - y_i) \log(1 - o_i)) \tag{3}$$

where d_i represents the number of observations based on which the y_i win rate was calculated. In other words, the solution maximises log probability of the observed sequence of games, assuming all games by any particular deck can be modelled as a win-loss boolean random variable. Note that this completely ignores the two-sided nature of games and the essential property of decks having varying win rate against specific enemies. However, attempts to estimate matchup-specific win rates resulted in worse performance, possibly caused by to overfitting due to an insufficient number of games for each specific matchup.

Additionally, decks from the test set were employed in training of the network to counteract the positive win rate bias which appeared when using training data. When the number of games per deck is not distributed uniformly, it is possible for the average win rate to be over or under 0.5, thus creating an unwanted bias in the model. In the training set for the competition, the average deck has a win rate of 0.517, leading models trained on the data to overestimate win rates of test decks. Using prediction for test games, this bias can be removed. Assuming u_j is the output for j-th test set game, the full loss is defined as:

$$\sum_i -d_i(y_i \log(o_i)+(1-y_i) \log(1-o_i))+(\lambda-\sum_j \frac{u_j}{200})^2 \tag{4}$$

The second term leads the average predicted win rate over the test set to be close to λ. This also works as regularisation for training. The λ value was chosen experimentally to maximize performance on leaderboard evaluation. We tested a range of values from 0.48 to 0.51, with 0.005 step size, and set $\lambda = 0.49$.

In Fig. 3, the training curve for the prediction network is shown. We compare the prediction network's performance

TABLE I: Results of the competition - top 5 submissions

Team	**RMSE**
hieuvq	5.57339852
amy	5.6482014
jj	**5.66759451**
dymitrruta	5.696228
amorgun	5.8473786

using two different encodings of training data. First is using the dual-task encoder described in Section IV, while the second one is encoded by a standard autoencoder, with no card decoding objective (i.e. the loss function is equal to the first term in Eq. 1). Training-test split for these experiments was the same, using 300 decks for training and 100 for tests. It is noticeable that the prediction network starts on a plateau and requires more than 1000 epochs to escape it, then rapidly improves the results. Past 2000 epochs little improvement is seen in test results although the minimum on the training set is not yet reached. Because of this, we set the early stopping point at 5000 epochs. Training time for this network was approximately 8 seconds per 1000 epochs on a GTX970 GPU.

Moreover, the improvement from using card decoding objective can also be seen in Fig.3. Prediction network performs better on training set but worse on test set when the card encoding objective is ignored. This indicates worse generalization without using the card decoding objective.

VI. Ensembling

The best performing single network achieved RMSE of approximately 5.0 on the 10% of the test data used to calculate leaderboard results. This result was further improved by ensembling, averaging results over multiple deep network models. Since throughout the competition we uploaded multiple results, models for the final ensemble were chosen from these according to their leaderboard evaluation results. The parameters given in sections IV and V describe the best single-network model. Other models in the final ensemble were variants of the described one with minor alterations, previously tested during parameter tuning: one with a larger number of training epochs (100000 for the encoder, 10000 for prediction network), one with added l2 regularization term in loss function (0.01 weight), and one with card matrix ignoring properties of cards other than word occurrence. These four best single-network models were averaged to obtain the final submission, resulting in approximately 0.2 RMSE improvement on the evaluation subset of test data.

VII. Conclusions

The final result placed the proposed solution as third in the competition. Results for other top submissions can be seen in Table I. It is worth noting that during the competition, two top solutions on the evalutation leaderboard reached RMSE below 5.0.

It can be argued that the chosen approach to representing Hearthstone decks was not sufficient to represent all intricacies of cards present in the test, but not training data. However, the change between results on the evaluation subset and final leaderboard suggests another explanation of the results, namely, that the described approach (along with some other top submissions) was over-tuned for the evaluation subset of test data.

While identifying the exact cause of this over-tuning is not possible without extensive tests on full data, the most likely explanation lies in the chosen approach to reducing positive win rate bias. As mentioned in Section V, the bias reduction is achieved by explicitly forcing the average win rate over test data to be close to an experimentally chosen value. The value 0.49 was set to maximise the performance according to the leaderboard. This means an implicit assumption was made that the 10% evaluation subset provides an accurate estimate of mean win rate for the entire test set.

Additionally, the choice of models to build an ensemble was based on the evaluation leaderboard, further contributing to the exact fitting of the model to evaluation subset of test data. A more refined ensemble building strategy would likely improve the results.

Regarding future work, the possibilities of the proposed approach are somewhat limited. The chosen model predicts win rates given a specific opponent distribution while using an unrealistic assumption that the test decks themselves are not part of the metagame. In a practical setting, due to fluctuating opponent distribution, a model that computes win rate for a matchup between two specific decks should be more applicable. We attempted to build such a model during the competition, however, since its training was significantly more computationally expensive than the prediction of win rates, the described approach was chosen instead. Nevertheless, predicting matchup-specific win rates remains a possible goal for further development that would require designing a new prediction network. The encoding network, on the other hand, can be potentially re-used without changes for any task that requires deck representation.

Acknowledgements

We would like to thank Silver Bullet Labs and Knowledge Pit for providing the simulation data and a platform for the competition.

References

[1] https://hearthstonejson.com/docs/cards.html

[2] https://hearthpwn.com

[3] Coates, Adam, Andrew Ng, and Honglak Lee. "An analysis of single-layer networks in unsupervised feature learning." Proceedings of the fourteenth international conference on artificial intelligence and statistics, 2011

[4] R Hahnloser, R. Sarpeshkar, M A Mahowald, R. J. Douglas, H.S. Seung, "Digital selection and analogue amplification coexist in a cortex-inspired silicon circuit" Nature, 405, pp. 947-951, 2000

[5] Zeiler, Matthew D. "ADADELTA: an Adaptive Learning Rate Method." Computing Research Repository, 2012

[6] Theano Development Team, "Theano: A Python framework for fast computation of mathematical expressions", arXiv:1605.02688 (2016).

[7] Deng, Li. "A tutorial survey of architectures, algorithms, and applications for deep learning." APSIPA Transactions on Signal and Information Processing 3, 2014