Proceedings of the Federated Conference on Computer Science and Information Systems pp. 131–134 DOI: 10.15439/2017F561 ISSN 2300-5963 ACSIS, Vol. 11

Helping AI to Play Hearthstone using Neural Networks

Łukasz Grad
University of Warsaw
Email: lg334481@students.mimuw.edu.pl

***Abstract*—This paper presents a winning solution to the AAIA'17 Data Mining Challenge. The challenge focused on creating an efficient prediction model for digital card game *Hearthstone*. Our final solution is an ensemble of various neural network models, including convolutional neural networks.**

I. Introduction

H*EARTHSTONE: Heroes of Warcraft* is a digital collectible card game that gained huge interest from players and AI researchers over the last couple of years. Although the rules of the game are simple, creating an AI model that would succesfully challenge an experienced human opponent is a difficult task, mainly due to inherent stochasticity and partial information. The goal of AAIA'17 Data Mining Challenge: Helping AI to Play Hearthstone was to design a model that can accurately evaluate arbitrary intra-game states, by predicting likelihood of winning the game by the first player.

In this paper, we present a winning solution that utilizes both neural network and convolutional neural network architectures. The proposed method requires only minimal domain knowledge and relies on basic feature preprocessing with no feature selection.

The rest of the paper is organized as follows. In section II we describe the details of the AAIA'17 challenge. Section III provides the description of features extracted from the data. Finally in section IV we present the details and results of our solution.

II. The Challenge

A. Game description

Hearthstone [1] is a turn based game between two players with custom built decks of thirty cards. Each player starts with thirty health points and zero mana crystals. At the start of each turn, player gains an additional mana crystal, draws a random card and his mana crystals are refreshed. Each card costs mana to play. Some cards are creatures, also called minions, that stay on the game board as long as their health is above 0. Players can make arbitrary number of actions during their turns, limited only by the mana crystals left. The goal of a game is to reduce the opponent's health to zero.

B. Problem statement

The given problem is an instance of binary classification task. Given a detailed representation of an arbitrary intra-game state, the goal is to predict the likelihood of winning the game by the first player, assuming it is his turn to play. The game state need not be a beginning state of a current turn. The predicted values do not need to be in particular range, however higher values should indicate a higher chance of winning. The accuracy of a model is defined as an *area under the ROC curve* (AUC). AUC can be interpreted as a probability that a classifier will rank a randomly chosen winning state higher than a randomly chosen losing state.

C. Data

Data sets provided in competition were extracted from large collection of play outs between weak AI players. Play outs were generated using all available nine classes and decks assembled from basic set of cards. The data was split into seven training sets and three test sets. In total, the training set consisted of 3 250 000 observations for which the correct decision label was provided. The test set had 750 000 game states in total and was missing the true labels. Game states contained in the test set were extracted from different play outs. Competitors were asked to submit their predictions on the whole test set.

Furthermore, data sets were provided in two different formats. The main set is a collection of JSON records containing a detailed description of each game state. Each record in JSON file contained:

- information about player and opponent heroes
- detailed description of all played creatures for both player and opponent
- detailed description of cards in player's hand
- current turn number

However, there was no information about previously played cards neither by player nor the opponent. The remaining cards in the player's deck were also unknown. The second data set was available in a simpler, tabular format. It contained the most important features extracted from JSON format and a handful of additional columns that aggregated information from several JSON fields.

D. Evaluation

Preliminary leaderboard was available for all competitors, based on a randomly chosen 5% subset of the test data set. This subset was the same for all participants and was known only to the organizers. There was no hard limit on the number of submisions available. Each team could select only one submission as their final solution that was evaluated on the remaining 95% of the data set.

III. Feature Engineering

Since the main data sets were given in a raw JSON format, a crucial first step in the competition was to extract meaningful features. Moreover, usage of external knowledge bases about cards was allowed, as long as it was publically available. In general, we can divide created features into three groups:

- played minion features
- hero features
- aggregating features

The following features were extracted from JSON data for each creature:

- `attack` - attack value
- `health` - current health value
- `can_attack` - whether minion can attack this turn
- `forgetful` - whether minion has 50% chance to miss a target
- `taunt` - whether minion has taunt
- `hp_max` - maximum health value

Other features extracted from JSON data:

- all player and opponent hero information
- all aggregating features from tabular data
- hero and opponent class in one-hot encoding

In addition, the following variables were added to the set:

- for each minion played: `aura` - whether a creature provides an active bonus for other minions
- `effective_health` - difference between total health of hero and total attack value of enemy minions
- `hand_power` - sum of marginal player hand card values based on *Heartharena Tierlist* [2]
- `hand_aoe` - total damage of 'area of effect' spells in hand
- `hand_answers` - number of 'hard removal' spells (that neutralize arbitrarily strong minions)

IV. Solution

A. Preprocessing

In all models, we normalize the data using Min-Max scaling. All features are scaled down to a fixed range from 0 to 1. Min-Max scaling proved to give slightly better results on holdout test sets than standarization with regard to mean and standard deviation.

In both neural network and convolutional neural network models, we also decided to include square and logarithm transformed features for all variables, excluding minion features and one-hot encoded hero class. This increased the total number of features to 260. In terms of bias-variance tradeoff, we want to decrease the bias even at the cost of increased variance of a single model.

B. Evaluation

Given a very large dataset we decided to evaluate our models using standard random train and test split, with 70% and 30% size respectively.

C. Initial models

To better understand the difficulty of the problem, we decided to train several standard linear and non-linear classifiers. We utilized Python's **scikit-learn** machine learning package (ver. 0.18.1) [3], [4]. For all models, if not explicitly stated, we used default parameters. Optimal hyperparameters where found using basic grid search approach on a small, random subsample of data.

- **Logistic Regression** fitted using Stochastic Average Gradient [5] solver with penalty parameter $C = 2.0$ and L2 regularization which resulted in 0.79321 score on local test set.
- **Support Vector Machine (SVM)** with RBF kernel and penalty parameter $C = 35.0$ trained on a random sample of size 50000 achieved a score of 0.78835 on local test set subsample of size 25000.
- **Random Forest** with 500 trees, minimum number of samples to split a node equal to 5 and maximal depth of 30 which resulted in 0.83494 score on local test set, around 0.784 on online preliminary test set.

We can see from the above results that a decent result can be achieved with a simple Logistic Regression model. However, an SVM trained on a very small data sample achieved only a slightly worse result. This tells us that the problem is highly non-linear and more complex models should perform better. On the other hand, the discrepancy between local and preliminary results for a Random Forest model is a clear sign of overfitting. The final test set has different characteristics and a good generalization is the key to win the competition.

D. Neural network

Feedforward neural network satisfies all requirements stated in section IV-C. The model can have arbitrary complexity, depending on the number of neurons and hidden layers, is very flexible and provides many techniques to reduce overfitting. Neural networks can also successfully be trained on very large datasets, as opposed to SVMs with non-linear kernels. On the downside, neural network training is highly sensitive to parameter initialization and can be hard to reproduce.

Both neural networks and convolutional networks were implemented in **Tensorflow** (ver. 1.1.0) [6] framework, a library for numerical computations using data flow graphs.

Network architecture consists of two hidden layers with dense connections and ReLU as an activation function [7]. Each hidden layer is followed by a Batch Normalization layer. Batch Normalization can speed up learning and reduce the *exploding gradient* problem [8]. We use L2 regularization of weights with $\lambda = 0.0002$. λ was set to a maximal value that did not hinder the network learning performance on local test set.

We trained many models with different number of hidden layer neurons. Best single network consisted of 100 neurons in first hidden layer and 50 neurons in second. It scored 0.7980 on the preliminary leaderboard.

E. *Convolution layer rationale*

From bayesian perspective, we can think of convolutional layer as a fully connected layer with an infinitely strong prior over some of its weights [9]. An infinitely strong prior places zero probability on parameters, making them forbidden, regardless of how much support the data assigns to those parameters. In case of convolution, this prior states that the layer should only learn local interactions and be equivariant to translation. Such prior results in sparse connections and parameter sharing, that significantly reduces the parameter space, when compared to fully connected layer of the same size.

The overall performance of convolutional network depends on whether our prior beliefs are reasonably accurate. If we are not correct, the network will likely underfit. On the other hand, if our prior is acceptable, we can expect the convolutional model to perform similarly, or even better then the original fully connected network, while having much less parameters.

F. *Convolution layer in detail*

Recall from section III that for each played minion we extracted 7 features. Each player can have up to 7 minions in play, giving a total of 98 variables. Since we already included features that describe the overall state of the game board, from the detailed minion features we want to extract information about how well they perform against each other. We state our belief that such performance should be measured independently of the position of a minion. Let p_i be the i'th player minion feature vector and o_i be the i'th opponent minion feature vector. We have $p_i, o_i \in \Re^7$, and $p_i, o_i = \vec{0}$ whenever there is no minion at the position i.

We can reshape the input vector as a $[7, 2, 7]$ tensor (multidimensional array), see Fig. 1a. We now introduce a *partial cyclic shift* (PCS) operation on such tensor, that applies a row-wise shift of player minion features, while leaving opponent minion features intact, Fig. 1b. We apply PCS 7 times with shift from 0 to 6. We then reshape each resulting tensor to $[7, 14]$, so that i'th row contains features of both i'th player and i'th opponent minions. Finally the tensors are stacked along third dimension, Fig. 1c shows a single row of final tensor (indices modulo 7). We want a convolution kernel to process the effectiveness of *all player minions againt a single opponent minion*.

p_0	o_0
p_1	o_1
p_2	o_2
p_3	o_3
p_4	o_4
p_5	o_5
p_6	o_6

(a) Minion features

p_1	o_0
p_2	o_1
p_3	o_2
p_4	o_3
p_5	o_4
p_6	o_5
p_0	o_6

(b) PCS with shift 1

p_i	o_i
p_{i+1}	o_i
p_{i+2}	o_i
p_{i+3}	o_i
p_{i+4}	o_i
p_{i+5}	o_i
p_{i+6}	o_i

(c) Final i'th row

Fig. 1

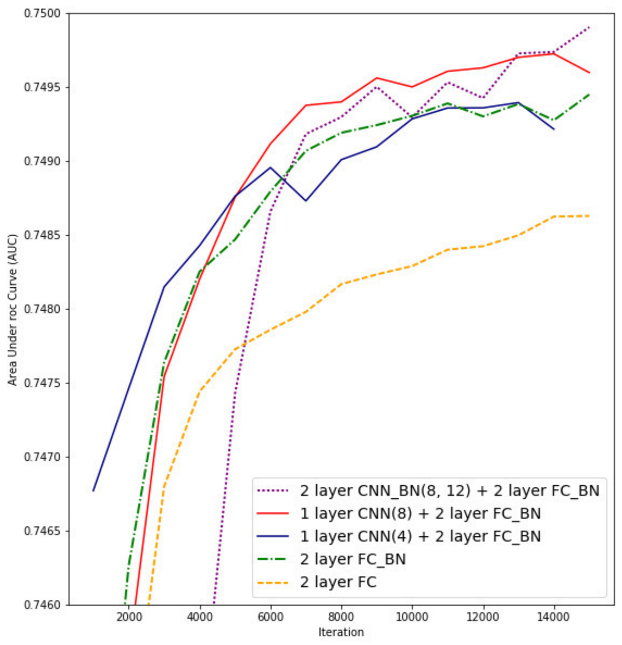

Fig. 2: Test set AUC scores during learning using **minion features only**. FC - fully connected layer, BN - batch normalization, CNN(d_1) - single convolution with depth d_1, CNN(d_1, d_2) - double convolution with depths d_1, d_2.

We introduce new hyperparameter d_1 - the depth of convolution result. After the preprocessing we run two convolutions in parallel creating a layer similar to inception layer [10]. First one, with kernel shape $[1, 14, 7, d_1]$, measures performance of player minions against a single oppponent minion. Second one with kernel shape $[2, 14, 7, d_1]$ that can take into account cooperation against 2 adjacent opponent minions. All convolutions are without padding, resulting in tensors with shapes $[7, 1, d_1]$ and $[6, 1, d_1]$ for first and second convolutions. We again use ReLU as activation function.

We also tested running additional convolutions with kernel $[1, 1, d_1, d_2]$ on top of the resulting tensors described above. Applying such operation creates the same d_2 linear combinations from d_1 features for each spatial location, see Fig. 3.

We then tested the performance of our convolution layer on **minion features only** with different d_1, d_2 hyperparameters, merging and flattening the resulting tensors and using a double layer fully connected network. We compared the results with double layer dense networks with raw minion features as input. Results are presented in Fig. 2.

We see that there is a lot of information contained only in minion features about the depending variable. Also, the convolutional layers can extract features that lead to similar classification performance as the raw inputs, despite the im-

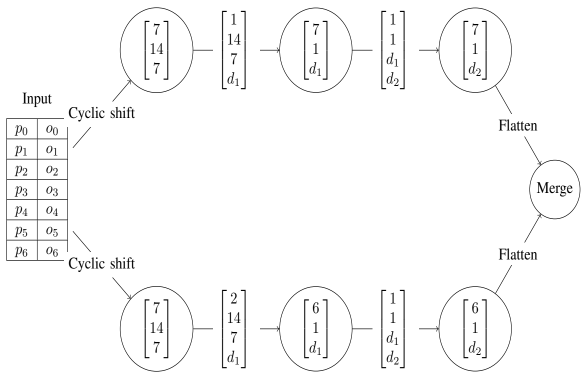

Fig. 3: Overview of the convolutional layer architecture. Node vectors represent tensor shapes at each step of computation. Edge vectors represent convolution kernel shapes.

posed prior. We believe that those features will allow a network achieve better generalization performance.

G. Convolutional network

We create a convolutional network by flattening the output from the convolutional layer and concatenating it with the original features, including raw minion features. We then use 3-layer feedforward network with ReLU activation and batch normalization after each hidden layer, without scaling factors. We did not include batch normalization in convolutional layer. We again use L2 regularization with $\lambda = 0.0002$. In some models, we substituted L2 regularization with dropout [11] with 0.5 probability, applied only to the last hidden layer. We present two best convolutional architectures in Table I.

H. Training

All networks were trained using Adam [12] optimizer with initial learning rate of 0.0001 or 0.0002, without learning rate decay and with cross entropy loss function. Network weights were initialized by sampling from truncated normal distribution with 0 mean and 0.1 standard deviation. Larger convolutional networks were also initialized using lower standard deviation of 0.05. Biases were initialized with 0.1 constants.

TABLE I: CNN with L2 regularization on the left and with dropout regularization on the right

[7x14x1] MINION INPUT	
[7x14x7] CYCLIC_SHIFT(MINION INPUT)	
[7x1x12] CONVOLUTION [1x14]	[6x1x12] CONVOLUTION [2x14]
[7x1x24] CONVOLUTION [1x1]	[6x1x24] CONVOLUTION [1x1]
[312 + 260] MERGE WITH INPUT	
[300] FULLY CONNECTED	
[300] BATCH NORM	
[60] FULLY CONNECTED	
[60] BATCH NORM	
[1] FULLY CONNECTED	

[7x14x1] MINION INPUT	
[7x14x7] CYCLIC_SHIFT(MINION INPUT)	
[7x1x12] CONVOLUTION [1x14]	[6x1x12] CONVOLUTION [2x14]
[7x1x24] CONVOLUTION [1x1]	[6x1x24] CONVOLUTION [1x1]
[312 + 260] MERGE WITH INPUT	
[300] FULLY CONNECTED	
[300] BATCH NORM	
[120] FULLY CONNECTED	
[120] BATCH NORM + DROPOUT	
[1] FULLY CONNECTED	

We used stochastic batch gradient descent with batch size of 320 and trained final models for around 16000 iterations on whole dataset, roughly 1.5 epochs. Such early stopping method was chosen empirically, basing on preliminary test results, since the holdout test scores proved to be highly unreliable.

I. Ensembling

We retrained each model a couple of times and selected top networks, based on preliminary results. Choosing models solely on preliminary results certainly could lead to overfitting, thus we created ensembles of top scoring predictors, with manually adjusted weights. All submitted ensembles scored far better than single models, see Table II. Final submission contained 11 models and won the competition with 0.80185 AUC score.

TABLE II: Excerpts of preliminary results

Model	Best model AUC
CNN L2	0.8012
CNN Dropout	0.8005
NN Ensemble	0.80
CNN Ensemble	**0.8041**
Final Ensemble	0.8037

References

[1] Blizzard Entertainment. (2017) Hearthstone official game site. [Online]. Available: https://eu.battle.net/hearthstone/en/

[2] HearthArena. (2017) Heartharena's hearthstone arena tierlist. [Online]. Available: http://www.heartharena.com/tierlist

[3] F. Pedregosa, G. Varoquaux, A. Gramfort, V. Michel, B. Thirion, O. Grisel, M. Blondel, P. Prettenhofer, R. Weiss, V. Dubourg *et al.*, "Scikit-learn: Machine learning in python," *Journal of Machine Learning Research*, vol. 12, no. Oct, pp. 2825–2830, 2011.

[4] L. Buitinck, G. Louppe, M. Blondel, F. Pedregosa, A. Mueller, O. Grisel, V. Niculae, P. Prettenhofer, A. Gramfort, J. Grobler *et al.*, "Api design for machine learning software: experiences from the scikit-learn project," *arXiv preprint arXiv:1309.0238*, 2013.

[5] M. Schmidt, N. Le Roux, and F. Bach, "Minimizing finite sums with the stochastic average gradient," *Mathematical Programming*, vol. 162, no. 1-2, pp. 83–112, 2017. doi: 10.1007/s10107-016-1030-6. [Online]. Available: http://dx.doi.org/10.1007/s10107-016-1030-6

[6] M. Abadi, A. Agarwal, P. Barham, E. Brevdo, Z. Chen, C. Citro, G. S. Corrado, A. Davis, J. Dean, M. Devin *et al.*, "Tensorflow: Large-scale machine learning on heterogeneous distributed systems," *arXiv preprint arXiv:1603.04467*, 2016.

[7] V. Nair and G. E. Hinton, "Rectified linear units improve restricted boltzmann machines," in *Proceedings of the 27th international conference on machine learning (ICML-10)*, 2010, pp. 807–814.

[8] S. Ioffe and C. Szegedy, "Batch normalization: Accelerating deep network training by reducing internal covariate shift," *arXiv preprint arXiv:1502.03167*, 2015.

[9] I. Goodfellow, Y. Bengio, and A. Courville, *Deep Learning*. MIT Press, 2016, http://www.deeplearningbook.org.

[10] C. Szegedy, W. Liu, Y. Jia, P. Sermanet, S. Reed, D. Anguelov, D. Erhan, V. Vanhoucke, and A. Rabinovich, "Going deeper with convolutions," in *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, 2015. doi: 10.1109/CVPR.2015.7298594 pp. 1–9. [Online]. Available: http://dx.doi.org/10.1109/CVPR.2015.7298594

[11] N. Srivastava, G. Hinton, A. Krizhevsky, I. Sutskever, and R. Salakhutdinov, "Dropout: A simple way to prevent neural networks from overfitting," *The Journal of Machine Learning Research*, vol. 15, no. 1, pp. 1929–1958, 2014.

[12] D. Kingma and J. Ba, "Adam: A method for stochastic optimization," *arXiv preprint arXiv:1412.6980*, 2014.