Analysis and Optimization of Deep Counterfactual Value Networks*

Patryk Hopner and Eneldo Loza Mencía

Knowledge Engineering Group, Technische Universität Darmstadt
patrykhopner@gmail.com, eneldo@ke.tu-darmstadt.de

Abstract. Recently a strong poker-playing algorithm called DeepStack was published, which is able to find an approximate Nash equilibrium during gameplay by using heuristic values of future states predicted by deep neural networks. This paper analyzes new ways of encoding the inputs and outputs of DeepStack's deep counterfactual value networks based on traditional abstraction techniques, as well as an unabstracted encoding, which was able to increase the network's accuracy.

Keywords: Poker, Deep Neural Networks, Game Abstractions

1 Introduction

Poker has been an interesting subject for many researchers in the field of machine learning and artificial intelligence over the past decades. Unlike games like chess or checkers it involves imperfect information, making it unsolvable using traditional game solving techniques. For many years the state of the art approach for creating strong agents for the most popular poker variant of No-Limit Hold'em involved computing an approximate Nash equilibrium in a smaller, abstract game, using algorithms like counterfactual regret minimization and then mapping the results back to situations in the real game. However, those abstracted games are several orders of magnitude smaller than the actual game tree of No-Limit Hold'em. Hence, the poker agent has to treat many strategically different situations as if they were the same, potentially resulting in poor performance.

Recently a work was published, combining ideas from traditional poker solving algorithms with ideas from perfect information games, creating the strong poker agent called DeepStack. The algorithm does not need to pre-compute a solution for the whole game tree, instead it computes a solution during game play. In order to make solving the game during game play computationally feasible, DeepStack does not traverse the whole game tree, instead it uses an estimator for values of future states. For that purpose a deep neural network was created, using several million solutions of poker sub-games as training data, which were solved using traditional poker solving algorithms.

* This is the authors' version of the work retrieved from http://www.ke.tu-darmstadt.de. The final publication is available at Springer at http://dx.doi.org/10.1007/978-3-030-00111-7_26 and appeared in Trollmann F., Turhan AY. (Eds.): *KI 2018: Advances in Artificial Intelligence. 41st German Conference on AI, Berlin, Germany, September 2428, 2018, Proceedings*. Lecture Notes in Computer Science, Vol. 11117. 2018. An extended version appeared in the arXiv Computing Research Repository CoRR:1807.00900 [cs.AI], https://arxiv.org/abs/1807.00900.

It has been proven, that, given a counterfactual value network with perfect accuracy, the solution produced by DeepStack converges to a Nash equilibrium of the game. This means on the other hand, that wrong predictions of the network can result in a bad solution. In this paper we will analyze several new ways of encoding the input features of DeepStack's counterfactual value network based on traditional abstraction techniques, as well as an unabstracted encoding, which was able to increase the network's accuracy. A longer version of this paper additionally analyzes the trade-off between the number of training examples and their quality [7] and many more aspects [6].

2 The Poker-Agent DeepStack

In the popular poker variant No-limit Hold'em for two players (Heads-up) each player receives two private cards which can be combined with five public cards [c.f., e.g. 20]. Players are then betting on whose five cards have the highest rank, according to the rules of the game. The *Counterfactual Regret Minimisation* (CFR) algorithm [20] and its variants [9, 4, 19] are state-of-the-art for finding approximate Nash equilibria [16] in imperfect information games and were the basis for the creation of many strong poker bots [20, 11, 18, 5] such as Libratus [17] which recently won a competition against human professional players. CFR can be used to compute a strategy profile σ and the corresponding *counterfactual values* (CV) at each information set I. The information sets correspond to the nodes in the game tree and the strategy profile assigns a probability to each legal action in an information set. Roughly speaking, the CV $v_i(\sigma, I)$ corresponds to the average utility of player i when both players play according to σ at set I.

Since poker is too large to be solved in an offline manner (the no-limit game tree contains $1.39 \cdot 10^{48}$ information sets) [1, 8], CFR is applied to abstracted versions of the game. The *card abstraction* approach groups cards into buckets for which CFR then computes strategies instead. In addition to the usefulness for creating smaller games, card abstractions can also be used to create a feature set for Deep Counterfactual Value Networks (see next), which is the focus of this work.

Depth Limited Continual Resolving *DeepStack* is a strong poker AI [14] which combines traditional imperfect game solving algorithms, such as CFR and endgame solving, with ideas from perfect information games, while remaining theoretically sound. In contrast to previous approaches using endgame solving [3, 2], which use a precomputed strategy before reaching the endgame, the authors of DeepStack propose to always re-solve the sub-tree, starting from the current state, after every taken action. However, on the early rounds of the game DeepStack does not traverse the full game tree since this would be computationally infeasible. Instead, it uses deep neural networks as an estimator of the expected CV of each hand on future rounds for its resolving step, resulting in the technique referred to as *depth limited continual resolving*.

Deep Counterfactual Value Networks DeepStack used a deep neural network to predict the player's counterfactual values on future betting rounds, which would otherwise be obtained by applying CFR. Consequently, the *deep counterfactual value network* (DCVNN) is trained with examples consisting of representations of poker situations as input and the counterfactual values of CFR as output. More specifically, the

network was fed with 10 million random poker situations and the corresponding counterfactual values obtained by applying CFR on the resulting sub-games [15]. For every situation a public board, private card distributions for both players and a pot size were randomly sampled. From this CFR is able to compute two counterfactual value vectors $\mathbf{v}_i = (v_i(j, \sigma))_j$ with $j = 1 \dots 1326$ for each possible private hand combination and for each player $i = 1, 2$. Note that $I = j$ represents the first level of the game tree starting from the given public board.

The input to the network is given by a representation of the players' private card distributions and the public cards. Hence, before the training of the neural network starts, DeepStack creates a potential aware card abstraction with 1000 buckets (cf. Sec. 3). For each training example the probabilities of holding certain private hands are then mapped to probabilities of holding a certain bucket by accumulating the probabilities of every private hand in said bucket. After the training of the model is completed, the CV for each bucket in a distribution can be mapped back to CV of actual hands by creating a reverse mapping of the used card abstraction. Fig. 1 depicts the general process, Sec. 3 describes it in more detail.

DeepStack was able to solve many issues associated with earlier game solving algorithms, such as avoiding the need for explicit card abstraction. However, DCVN introduce their own potential problems. For instance, the incorrect predictions caused by encoding of the player distributions as well as the counterfactual value outputs could potentially result in a highly exploitable strategy. The distributions and outputs are encoded using a potential aware card abstraction, potentially leading to similar problems as traditional card abstraction techniques, which is something we will call *implicit card abstraction*.

3 Distribution Encoding

While DeepStack never uses explicit card abstraction during its re-solving step, the encoding of inputs and outputs of counterfactual value networks is based on a card abstraction, which introduces potential problems. Because the input player distributions get mapped to a number of buckets prior to training, the training algorithm is not aware of the exact hand distributions, but only of the distribution of bucket probabilities. Because this is a many to one mapping, the algorithm might not be able to distinguish different situations, thus not being able to perfectly fit the training set. The second problem stems from the encoding of the output values. Counterfactual values of several hands are aggregated to a counterfactual value of a bucket, potentially losing precision. Both problems are visualized in Figure 1 which also depicts the basic architecture of DeepStack's counterfactual value estimation.

While the problem is similar for inputs and outputs, we will focus on the loss of accuracy of counterfactual value outputs. We will call the difference between the original counterfactual values of hands, as computed by the CFR solver, and the counterfactual values after an abstraction based encoding was used, the *encoding error*. The difference between the original counterfactual values and the bucket counterfactual values will be measured using the mean squared error as well as the Huber loss (with $\delta = 1$) averaged over all private hands and test examples, as proposed by [14]. For instance,

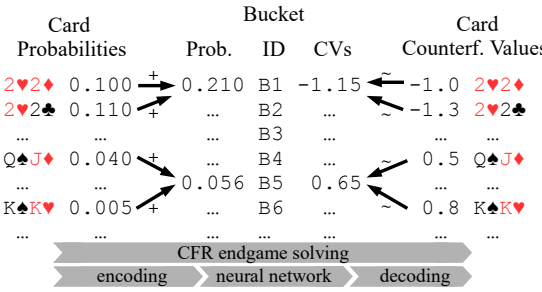

Fig. 1. Diagram depicting the 1) encoding from private cards to buckets (depicted by black arrows) 2) mapping from private card distributions to bucket distributions (the summing of probabilities symbolized by $+$) 3) mapping from private card counterfactual values to buckets CVs (averaging symbolized by $\sim$) 4) pipeline of CFR mapping private card distributions to respective CVs 6) replicating DeepStack pipeline consisting of a) encoding b) estimating of the buckets' CVs by a neural network c) decoding back the estimated buckets' CVs to the cards' CVs.

in Figure 1 we would apply the loss functions on the differences $|-1.0-(-1.15)|$, $|-1.3-(-1.15)|, \ldots$.

We will examine three abstraction based encodings, including the potential aware encoding, which was used by DeepStack, as well as an unabstracted encoding. We will then compare the encoding error of each encoding, as well as the accuracy of the resulting networks.

When measuring the accuracy of the model, we have two possible perspectives. The first is to look at the prediction error with both inputs and outputs encoded with a card abstraction. The second way is to map the predictions of buckets back to predicted counterfactual values of private hands and compare them to the unabstracted counterfactual values of the test examples. When measuring the error using encoded inputs and outputs, we will refer to the test set as *abstract test set*. In Figure 1 this would correspond to the error between the bucket CVs column (after mapping from the actual private privat card CVs) and the predicted bucket CVs. When we are measuring the prediction error for unabstracted private hands, we will call the dataset the *unabstracted test set*, which in Figure 1 corresponds to comparing to the card CVs column after decoding the predicted bucket CVs. We will use the same logic for the training set.

$E[HS^2]$ Abstraction On the last betting round the *hand strength* (HS) value of a hand is the probability of winning against a uniform opponent hand distribution. On earlier rounds the *expected hand strength squared* ($E[HS^2]$) [11] is calculated by averaging the square of the HS values over all possible card roll outs.

The $E[HS^2]$ abstraction uses the $E[HS^2]$ values in order to group hands into *buckets*. There are several ways to map hands to a bucket, including percentile bucketing, which creates equally sized buckets, clustering of hands with an algorithm such as k-Means [12] or by simply grouping hands together, that differ only by a certain threshold in their $E[HS^2]$ values.

Nested Public Card Abstraction A *nested public card abstraction* first groups public boards into *public buckets* and those buckets are later subdivided according to some metric which takes private card information into account, such as $E[HS^2]$.

In this work boards were clustered according to two features, the *draw value* and the *highcard value*. The draw value of a turn board was defined as the number of straight and flush combinations, which will be present on the following round. The highcard

value is the sum of the ranks of all turn cards, with the lowest card, a deuce, having a rank of zero and an ace having a rank of 12.

Potential Aware Card Abstraction The *potential aware card abstraction* [10] tries to not only estimate a hand's current strength, but also its potential on future betting rounds. It does that by first creating a probability distribution of future HS values for each hand and then clustering hands using the *k-Means* [12] algorithm and the *earth mover's distance* [10].

Abstraction-free Direct Encoding Instead of using a card abstraction in order to aggregate private hand distributions to bucket distributions and private hand CVs to bucket CVs, this encoding uses the private hand data directly. The input distributions are represented as a vector of probabilities of holding one of the 1326 possible card combinations. The boards are represented using one hot encoded vectors where each of the 52 dimensions represents whether a specific card is present on the public board.

4 Evaluation

In order to compare the encodings, first a version of each card abstraction described in the previous section was created. Like in the original DeepStack implementation, the potential aware card abstraction used 1000 buckets. The $E[HS^2]$ abstraction used 1326 buckets based on a equal width partition of the value interval $[0, 1]$. The public nested card abstraction was created by first clustering the public boards into 10 public clusters according to their draw and highcard value and subdividing each public cluster into 100 $E[HS^2]$ buckets, resulting in a total of 1000 buckets. For the analysis of the encoding error, the CVs of each training example were then encoded using each of the three card abstractions, meaning that they were aggregated to a CV of their bucket. Those bucket CVs were then compared with the original CVs of the hands in said bucket and the average error over all available training examples was computed.

Our computational resource only allowed us to create 300,000 endgame solution-sinstead of the 10 million available to DeepStack. All 300,000 training examples were used for testing the encoding error of each abstraction. For the second comparison the DCVN were trained using each of the 3 abstraction based encodings, as well as the unabstracted encoding. The training set consisted of 80% of the total 300,000 endgame solutions, while the test set consisted of 20%. The networks were trained for 350 epochs using the Adam Gradient descent [13] and the Huber Loss.[1]

Encoding and Prediction Errors Table 1 shows the encoding error of the abstraction based encodings. Table 2 reports the errors of the trained neural networks. Remember that the abstraction-free encodings do not produce any encoding error, therefore, their performance is also the same on the abstracted and unabstracted sets. Note also that the errors on the abstracted sets are not directly comparable to each other due to the different encoding.

[1] As in DeepStack, the inputs to the networks with 7 layers with 500 nodes each using parametric ReLUs and an outer network ensuring the zero-sum property are the respective encodings.

Table 1. Encoding error of different encoding schemes on the turn.

Encoding Approach	$E[HS^2]$	Public Nested	Potential Aware
Huber loss	**0.0240**	0.0406	0.0258
MSE	**0.0509**	0.0886	0.0544

Table 2. Prediction error of neural network using different input encodings on the abstracted and unabstracted train and test sets, on the turn.

Encoding Approach	$E[HS^2]$	Public Nested	Potential Aware	Abstraction–Free
Abstracted Train	0.0254	0.0080	**0.0052**	0.0102
Unabstracted Train	0.0387	0.0436	0.0267	**0.0102**
Abstracted Test	0.0330	0.0161	**0.0102**	0.0143
Unabstracted Test	0.0434	0.0478	0.0297	**0.0143**

We can observe that the $E[HS^2]$ abstraction introduces a smaller encoding error than the potential aware card abstraction, although not by a big margin. However, it is outperformed in terms of the accuracy of the neural networks. The potential aware abstraction performed better in its own abstraction, as well as after mapping the counterfactual values of buckets back to counterfactual values of cards.

A contrary behaviour can be observed for the public nested encoding. Whereas it has major difficulties in encoding, the resulting encodings carry enough information for the network to predict relatively well on the bucketed CVs. However, mapping the CVs back to the actual hands strongly suffers from the initial encoding problems.

However, the most noteworthy (and surprising) result is the performance of the abstraction-free encoding. Whereas the potential aware encoding was able to produce a lower Huber Loss in its own abstraction, the abstraction-free encoding outperformed the abstraction on the unabstracted training set and the unabstracted test set. The direct encoding was therefore better than the potential aware encoding at predicting counterfactual values of actual hands instead of buckets, which is the most important measure in actual game play. These results suggest that the neural network was able to generalize among the public boards even though no explicit or implicit support was given in this respect. Note that this was possible even though we only used a small number of training instances compared to DeepStack.

5 Conclusions

In this paper we have analyzed several ways of encoding inputs and outputs of deep counterfactual value networks. We have introduced the concept of the encoding error, which is a result of using an encoding based on lossy card abstractions. An encoding based on card abstraction can lower the accuracy of training data by averaging counterfactual values of multiple private hands, introducing an error before the training of the neural network even started. We have observed that the encoding error can have a substantial impact on the accuracy of the trained network, as observed in the case of the

public nested card abstraction which performed well on its abstract test set but lost a lot of accuracy when the counterfactual values of buckets were mapped back to hands.

The potential aware card abstraction produced the best results of all the abstraction based encodings, which corresponds to the results achieved by the abstraction in older algorithms, where it is the most successful abstraction at this point. However, the unabstracted encoding produced the lowest prediction error. While a good result on the training set was expected, it was unclear if the neural network would generalize well to unseen test examples. This result again shows the importance of minimizing the encoding error when designing a deep counterfactual value network.

References

[1] Bowling, M., Burch, N., Johanson, M., Tammelin, O.: Heads-up limit hold'em poker is solved. Commun. ACM 60(11), 81–88 (Oct 2017)

[2] Burch, N., Bowling, M.: CFR-D: solving imperfect information games using decomposition. CoRR abs/1303.4441 (2013), http://arxiv.org/abs/1303.4441

[3] Ganzfried, S., Sandholm, T.: Endgame solving in large imperfect-information games. In: Proceedings of the 2015 International Conference on Autonomous Agents and Multiagent Systems. pp. 37–45. AAMAS '15, Richland, SC (2015)

[4] Gibson, R.: Regret Minimization in Games and the Development of Champion Multiplayer Computer Poker-Playing Agents. Ph.D. thesis, University of Alberta (2014)

[5] Gilpin, A., Sandholm, T., Sørensen, T.B.: A heads-up no-limit texas hold'em poker player: Discretized betting models and automatically generated equilibrium-finding programs. In: Proceedings of the 7th International Joint Conference on Autonomous Agents and Multiagent Systems - Volume 2. pp. 911–918. AAMAS '08, International Foundation for Autonomous Agents and Multiagent Systems, Richland, SC (2008)

[6] Hopner, P.: Analysis and Optimization of Deep Counterfactual Value Networks. Bachelor's thesis, Technische Universität Darmstadt (2018), http://www.ke.tu-darmstadt.de/bibtex/publications/show/3078

[7] Hopner, P., Loza Mencía, E.: Analysis and optimization of deep counterfactual value networks (2018), http://arxiv.org/abs/1807.00900

[8] Johanson, M.: Measuring the size of large no-limit poker games. CoRR abs/1302.7008 (2013), http://arxiv.org/abs/1302.7008

[9] Johanson, M., Bard, N., Lanctot, M., Gibson, R., Bowling, M.: Efficient nash equilibrium approximation through monte carlo counterfactual regret minimization. In: Proceedings of the 11th International Conference on Autonomous Agents and Multiagent Systems - Volume 2. pp. 837–846. AAMAS '12, International Foundation for Autonomous Agents and Multiagent Systems, Richland, SC (2012)

[10] Johanson, M., Burch, N., Valenzano, R., Bowling, M.: Evaluating state-space abstractions in extensive-form games. In: Proceedings of the 2013 International Conference on Autonomous Agents and Multi-agent Systems. pp. 271–278. AAMAS '13, International Foundation for Autonomous Agents and Multiagent Systems, Richland, SC (2013)

[11] Johanson, M.B.: Robust Strategies and Counter-Strategies: From Superhuman to Optimal Play. Ph.D. thesis, University of Alberta (2016), http://johanson.ca/publications/theses/2016-johanson-phd-thesis/2016-johanson-phd-thesis.pdf

[12] Kanungo, T., Mount, D.M., Netanyahu, N.S., Piatko, C.D., Silverman, R., Wu, A.Y.: An efficient k-means clustering algorithm: Analysis and implementation. IEEE Trans. Pattern Anal. Mach. Intell. 24(7), 881–892 (Jul 2002), http://dx.doi.org/10.1109/TPAMI.2002.1017616

[13] Kingma, D.P., Ba, J.: Adam: A method for stochastic optimization. CoRR abs/1412.6980 (2014), http://arxiv.org/abs/1412.6980
[14] Moravcík, M., Schmid, M., Burch, N., Lisý, V., Morrill, D., Bard, N., Davis, T., Waugh, K., Johanson, M., Bowling, M.H.: Deepstack: Expert-level artificial intelligence in no-limit poker. CoRR abs/1701.01724 (2017), http://arxiv.org/abs/1701.01724
[15] Moravcík, M., Schmid, M., Burch, N., Lisý, V., Morrill, D., Bard, N., Davis, T., Waugh, K., Johanson, M., Bowling, M.H.: Supplementary materials for deepstack: Expert-level artificial intelligence in no-limit poker (2017), https://www.deepstack.ai/
[16] Nash, J.: Non-cooperative games. Annals of Mathematics 54(2), 286–295 (1951)
[17] Noam Brown, T.S.: Libratus: The superhuman ai for no-limit poker. In: Proceedings of the Twenty-Sixth International Joint Conference on Artificial Intelligence, IJCAI-17. pp. 5226–5228 (2017)
[18] Schnizlein, D.P.: State Translation in No-Limit Poker. Master's thesis, University of Alberta (2009)
[19] Tammelin, O.: Solving large imperfect information games using CFR+. CoRR abs/1407.5042 (2014), http://arxiv.org/abs/1407.5042
[20] Zinkevich, M., Johanson, M., Bowling, M., Piccione, C.: Regret minimization in games with incomplete information. In: Platt, J.C., Koller, D., Singer, Y., Roweis, S.T. (eds.) Advances in Neural Information Processing Systems 20, pp. 1729–1736. Curran Associates, Inc. (2008), http://papers.nips.cc/paper/3306-regret-minimization-in-games-with-incomplete-information.pdf