

Learning Useful Features For Poker

Arjun Nagineni

December 5, 2018

Supervisor: Risto Miikkulainen

Second Reader: Gordon Novak

Departmental Reader: Robert Van De Geijn

Department of Computer Science

The University of Texas at Austin

Contents

1	Introduction	3
2	Related Work	5
2.1	Poker Algorithms	5
2.2	RNNs	8
3	Modeling Experiments	11
3.1	Cards Model	12
3.2	Action Sequence Model	13
4	Comparison Experiments	17
5	Future Work	21
6	Conclusion	22

Abstract

This thesis focuses on using neural networks to extract useful features from a poker game. These features will be used to exploit opponent behavior in playing poker. The opponent modeling network was built by Xun Li in prior work and this thesis is about exploring a different game state representation in the network. Currently, the input features are hand-made using common poker knowledge. The research is to let the computer figure out which features are important from the raw state of the poker game instead of using the hand-made features. Just as neural networks are proven to work better for extracting visual features for various applications, the project's goal is to prove that for poker game representation. This research focuses on state representation only and actual gameplay (with the new representation) needs to be explored in future work. The model built in this thesis was able to represent most of the poker game state (except bet sizes) by significantly compressing it (through feature learning) while not losing much information. Moreover, the compressed representation is found to be similar yet better than the handcrafted input features because it packs more nuanced information than the latter. In the future, the model can be extended to include bet sizes and tested out in actual poker gameplay.

1. Introduction

Incomplete information games are everywhere in our economy like trading and investing, business and international negotiations, wars, etc. With such use cases, getting better at playing incomplete information games will be quite significant for our society. To study and come up with new algorithms for incomplete information games, one has to start with a well-defined experimental model. Poker is one such example. The rules are well-defined and the information

is incomplete (opponent's cards are unknown). Historically, Nash equilibrium based algorithms like Counterfactual Regret Minimization (CFR) worked best for playing poker [Zinkevich 2007]. In fact, heads up limit poker is solved by these methods [Bowling 2017]. However, CFR plays a defensive strategy that does not maximize winnings and it works only for heads-up (two player) poker while requiring massive compute power and memory (more on this in “related work” section). Therefore, other methods like neural networks are worth exploring to play poker.

Li and Miikkulainen [2018] created a dynamically adapting poker player based on opponent behavior. Li et al.'s poker player is called Adaptive System of Holdem (ASHE). ASHE is found to be more profitable than Nash equilibrium based poker agents against exploitable players while still being competitive against the best Nash equilibrium based players [Li 2018]. ASHE uses Long-Short Term Memory (LSTM) neural network modules to predict opponent behavior and neuro-evolution to train the LSTM modules. However, ASHE also relies on handcrafted features from common poker knowledge as input to the LSTM modules. This research aims to find out whether extracting input features using neural networks from raw poker game states can replace the hand-crafted input features.

One potential benefit of using learned features over hand-crafted features is that ASHE's performance can get better. When using a neural network approach, the goal has long been to learn useful features in the hidden units that will be used to predict the outcome. By giving handcrafted features instead of the raw poker state itself as the input, ASHE is being handicapped by not allowing it to come up with its own innovative set of features. Moreover, these features could be learned from factors derived from the training data that humans may not know about. In other applications, research has shown that extracting features from raw state

using neural networks works better than handcrafted features. For example, researchers have found that using a convolutional neural network on raw speech information is better at extracting human emotions than traditional systems that use “audio-based or image-based hand-crafted features” [Papakostas 2017]. Such results in other domains also give hope that feature learning will be better than human-made features for ASHE.

Another benefit of feature learning is that the model becomes more generalizable. Currently to apply ASHE’s model in other incomplete information games, an expert in the field needs to handcraft features as input to ASHE. Finding this expert and encoding his or her intuitive knowledge will be difficult. If the feature learning model is used, only the raw state of the problem is needed to get results from ASHE. Moreover, neural networks sometimes turn out to be much better than experts which happened in the case of Go [Silver 2017]. So this project will be a success even if ASHE’s performance stays about the same after replacing the handcrafted features.

2. Related Work

The first part of this section discusses the successful poker algorithms in the past. It also discusses some of the newer neural network based approaches and where this thesis fits in. The second part of the section focuses on recurrent neural networks, a key component in this research and compares Long-Short Term Memory networks with Gated Recurrent Units.

2.1 Poker Algorithms

For a long time, researchers have tried to solve poker using game theory techniques like finding Nash equilibria in condensed poker game states. In 2007, Zinkevich et al. introduced a better technique called Counterfactual Regret Minimization (CFR) for playing poker-like games

[Zinkevich 2007]. CFR is essentially a self-play algorithm where the program plays billions of rounds against itself. After each game, the algorithm evaluates which actions would make the program's strategy better over all the previous games (a positive "regret"). These actions are then performed more often in future games. The average strategy over the billions of hands played will converge to a Nash equilibrium for the game [Zinkevich 2007]. CFR is successful at playing poker because it employs a defensive strategy which is nearly impossible to exploit.

With the focus on playing safe, CFR does not maximize winnings against a typical player. So there is room for another poker playing algorithm that focuses on winning big by exploiting opponent behavior. As mentioned in the introduction, Xun et al. are aiming to do just that with ASHE. Moreover the CFR self-play algorithm is quite compute intensive for massive state space games like Texas hold 'em poker (3×10^{14} information sets for 2 player limit hold 'em). So to play complex games, the successful CFR algorithms create an abstraction of the game like grouping "many different card dealings into buckets" [Gibson 2011]. Even with the abstractions, CFR only works for heads-up (two player) poker and is still quite memory intensive during actual gameplay. For example, CMU's Tartarian that statistically tied with pro poker players had a 2 TB online lookup table [Yakovenko 2016]. A neural network approach (like ASHE) could be used to significantly reduce the memory and compute requirements at runtime. For example, a poker playing convolutional neural network designed by Yakovenko et al. [2015] fits in a 1MB file. Such low compute and memory requirements during gameplay mean that neural network approaches like opponent modeling can also be extended to multiple players.

There have been attempts to model and exploit opponent behavior in poker and other contexts. Most of these experiments used neural network approaches. One such example is

University of Alberta’s poker agent Poki. They have progressed from using a generic opponent model (one size fits all approach) to a specific one (treating each opponent as distinct) that tracked the opponent hand strength based on a “table of betting frequencies for various stages during the hand” [Davidson 2000]. They also used a neural network to identify the most relevant features in predicting opponent behavior. Those features turned out to be previous action and previous amount to call. Although Poki has made considerable progress in opponent modeling and achieved close to the average winnings of a professional poker player, as Davidson et al. said: “the topic is far from being well-solved” [Davidson 2000]. There is still a need for better opponent modeling in poker because the results are not yet as decisive as results of the CFR approach for playing heads up no limit poker.

This thesis aims to improve opponent modeling in general by using feature extraction without relying on hand-crafted features. Research has shown that neural networks are much better at feature extraction than hand-made features in fields like computer vision. Almost three decades ago, neural networks were used to extract features from raw image data that worked much better than “engineered feature vectors” [LeCun 1989]. The goal for this project is to get to a similar conclusion with poker game state representation.

Recently, advances have been made by a company called DeepStack to use neural networks along with CFR (Nash equilibrium based) algorithms to create better approximations [Moravcik 2017]. However, the input to the neural network is a ratio of pot size to opponent’s stacks and encoding opponent hand probabilities into 1,000 buckets based on the community cards. This representation is nowhere close to the raw state of a poker game. If this research

leads to a better set of features that are built from looking at the entire raw state of the game, the following network could also be used to improve such Nash equilibrium approximation methods.

Researchers like Yakovenko et al. [2015] came very close to representing the raw poker game state as input to a convolutional neural network. They used a 3D sparse array (31 x 17 x 17) that captures cards, pot size, previous betting rounds, button (dealer in the game). Although this is a very comprehensive approach to representing the raw state, there are some unaccounted details like the order of bets in each betting round (like check-raises) which is very important in understanding opponent behavior.

2.2 Recurrent neural networks

Recently, recurrent neural networks (RNNs) have gained popularity for specific use cases and can be explored for poker game representation too. Unlike regular feedforward networks, recurrent networks have a loop or cycle in the connections between units. This loop allows recurrent networks to work with data represented as an arbitrarily long sequence where order matters. This feature of RNNs is where feedforward networks lack because they assume each training example is independent with a fixed length [Lipton 2015]. Among all the recurrent neural networks, Long Short-Term Memory networks (LSTMs) and Gated Recurrent Units (GRUs) are the most popular mainly due to their success in natural language processing and machine translation respectively [Hochreiter 1997; Cho 2014]. In this research, both LSTMs and GRUs are explored to model the action sequence of a poker game. For this use case, GRUs were found to provide better accuracy than LSTMs (more on this in the next section).

LSTMs were designed to solve the vanishing gradient problem that occurs in recurrent networks due to repeated backpropagation (or simply multiplication) of gradients less than one.

Such repeated multiplication leads to a gradient very close to zero with increasing time steps and the network will not be able to learn from long sequences. LSTMs solve the vanishing gradient problem by maintaining an internal state that is added (instead of multiplied) to the processed state and by forgetting unimportant states (through a concept called gating). Figure 1 describes the structure of an LSTM cell.

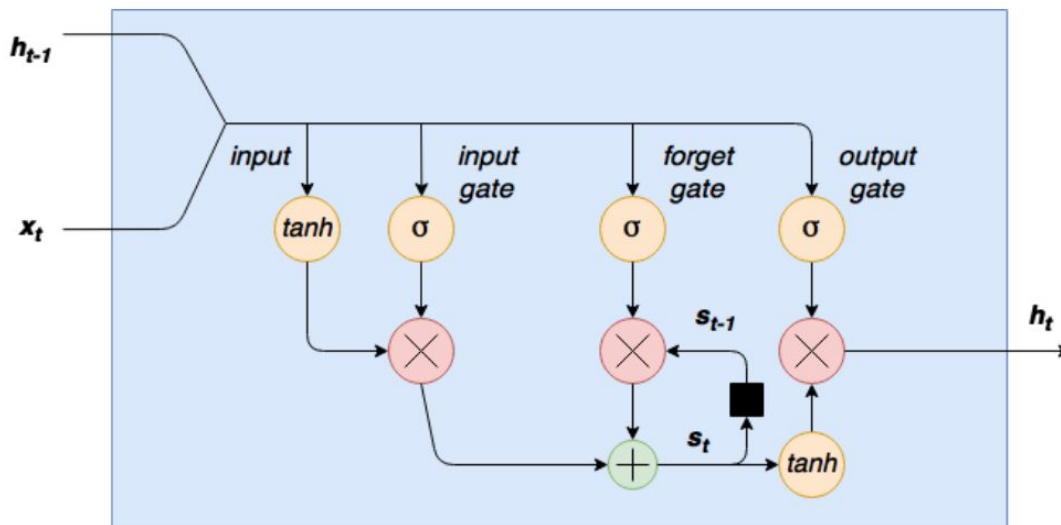


Figure 1: LSTM Cell Structure [Andy 2017]. The diagram shows the flow of x_t , a new element in the sequence (e.g. a word in a paragraph) and h_{t-1} , the previous output to generate the output at current time step, h_t . This diagram helps visualize the inner workings of an LSTM cell.

An LSTM cell (described in figure 1) has an input gate which is a sigmoid activation that rates the importance of the input (0 means switch off input values, 1 means pass through). This gate helps with only “remembering” important input. S_t is the internal state variable which is delayed by a time step and added to the output from the input gate (this addition is important to deal with the vanishing gradient problem). The forget gate and output gate have a similar filtering as the input gate. The final output, h_t is an element-wise multiplication (denoted by the ‘X’) of $\tanh(S_t)$ and output of output gate. From the figure, it is evident that an LSTM cell offers a lot of

flexibility with what is set as input, what is “remembered” in the internal state and what is set as output.

GRUs were recently introduced by Cho et al. [2014] to perform neural machine translation. GRU is like a simplified LSTM which still solves the vanishing gradient problem with two gates instead of three and without an internal state S_t .

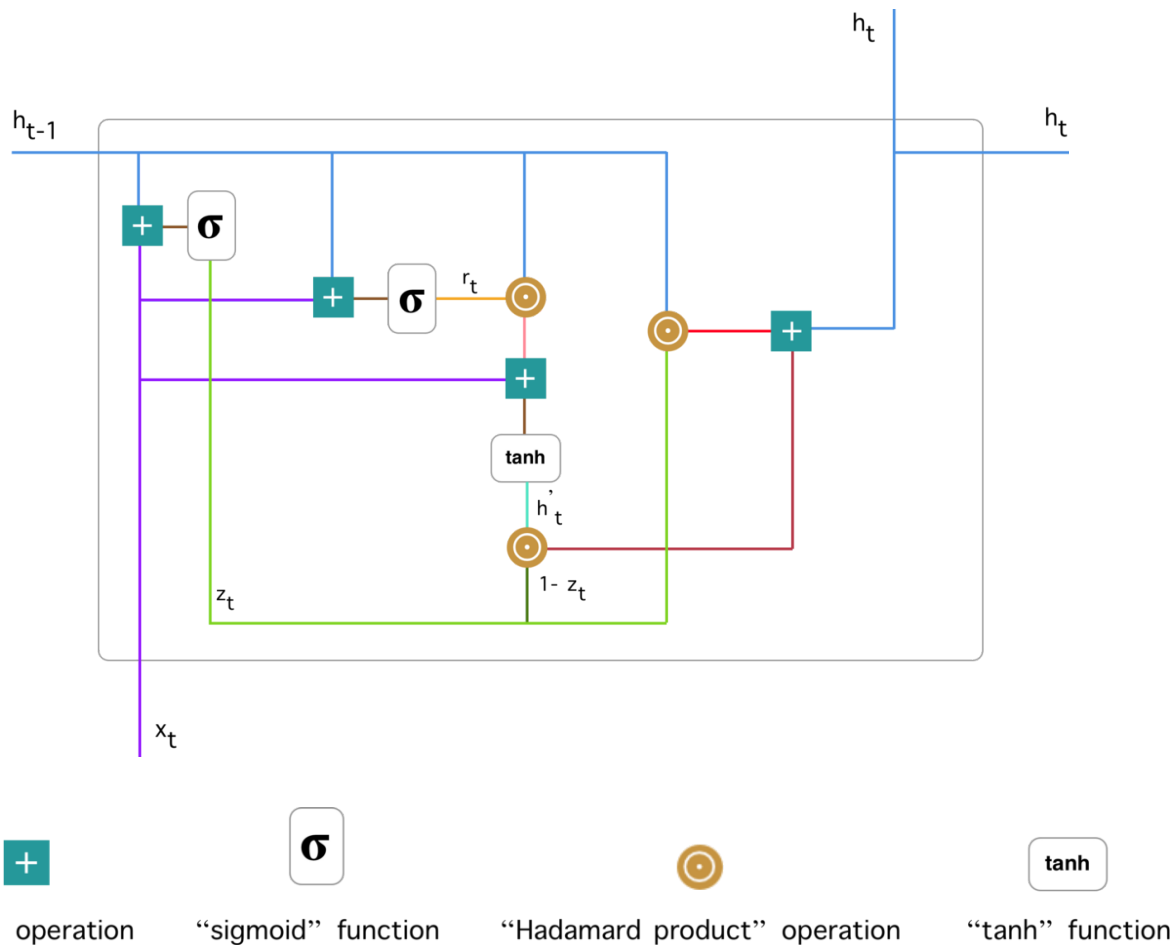


Figure 2: GRU Cell Structure [Kostadinov 2017]. The diagram shows the flow of x_t , a new element in the sequence (e.g. a word in a paragraph) and h_{t-1} , the previous output to generate the output at current time step, h_t . This diagram helps visualize the inner workings of a GRU cell. As shown in figure 2, there are two gates in a GRU cell: the update gate (left sigmoid function) and the reset gate (right sigmoid function). The update gate determines “how much the unit updates its activation (h_t)” [Chung 2014]:

$$z_t = \sigma(\mathbf{W}_z x_t + \mathbf{U}_z h_{t-1}). \quad (1)$$

where \mathbf{W} and \mathbf{B} are weight matrices. The reset gate determines how much of the previous state to forget:

$$r_t = \sigma(\mathbf{W}_r x_t + \mathbf{U}_r h_{t-1}). \quad (2)$$

If r_t is close to 0, the unit acts “as if it is reading the first symbol of an input sequence” [Chung 2014]. An intermediate memory state is computed (similar to S_t in the LSTM description above) using the reset gate output:

$$h'_t = \tanh(\mathbf{W} x_t + r_t \odot \mathbf{U} h_{t-1}). \quad (3)$$

The final output from the GRU cell is given by:

$$h_t = z_t \odot h_{t-1} + (1 - z_t) \odot h'_t. \quad (4)$$

There are similarities between the internal workings of LSTM and GRU cells especially with “the additive component of their update from t to $t + 1$ ” which helps avoid the vanishing/exploding gradient problem [Chung 2014]. The next section shows how RNNs are used in this research and how LSTMs and GRUs compare against each other for this use case.

3. Modeling Experiments

Ultimately, this project is successful if the learned features make ASHE more generalizable or improve ASHE’s performance. However, training ASHE on each variation of the feature extractor model in order to check the performance would be too time consuming (given the time available for this thesis). So there needed to be an intermediate sanity check. This check was to extract a handful of features from the raw representation of the game and be able to reconstruct the game state back from it. This would ensure that the learned features are a condensed and more robust version of the entire game state. Once this check is achieved, the

learned features can be used as input to a real poker agent like ASHE to check the performance. This thesis shows results where the intermediate check is achieved for most of the poker game state and the extracted features are similar yet more powerful than ASHE's hand-crafted features.

To model the intermediary check, an autoencoder approach was used where the hidden layers have fewer units than the input layer and the output layer is same as the input layer. The hidden layer with least number of units will be the learned features. First the poker game state was split into cards (the community and hole cards) and action sequence (all the actions along with bet sizes that lead to the current round). Features for each of them were extracted in separate neural networks because both have distinct input representations that required different models.

3.1 Cards Model

For the cards part of the poker game state, the model took seven cards (five community cards and two hole cards) as input, each represented as a 52 element one-hot vector. So the total number of input features for the cards model was 364. Some of these 52-element cards could be all zeros which indicated that the card has not been drawn yet. The undrawn cards were important to model because not all community cards are known in a poker game until the final betting round. The card input then went through three densely connected layers where the feature set was compressed from 364 to 100 to 50 to 20 features. This architecture is the encoding part of the network. To decode back to the input, the 20 learned features were passed through another three densely connected layers that went from 20 to 50 to 100 to 364 features. A sigmoid activation function was applied at the output layer to introduce nonlinearities and to restrict the output between 0 and 1 which made it consistent to compare with the one-hot encoded input

vectors. A crossentropy loss was used between the predicted input (the output) and the actual input to improve the model.

The card model was trained on input batches of 32 on a data set with one million randomly generated card sets. The accuracy of the model was measured on a validation dataset (20% of the original dataset) by checking if each card was predicted correctly in the output rather than checking every single feature. The model yields a 99.9%+ accuracy for 20 extracted features. The accuracy drops for fewer learned features. For example, the accuracy was around 70% for 15 learned features. The relation between accuracy and extracted features is shown better in figure 3. The accuracy fell drastically as extracted features were reduced.

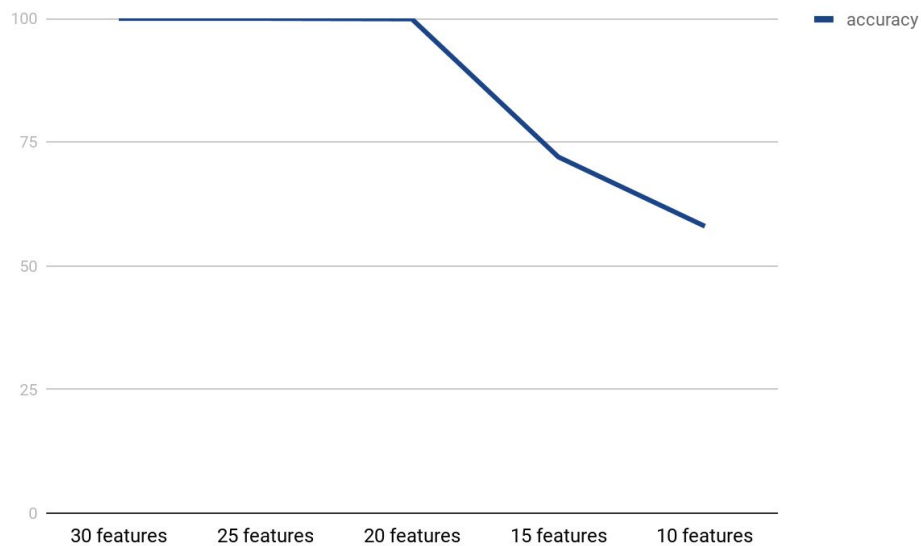


Figure 3: Cards Model: Accuracy vs Extracted Features. A graph that shows the relation between accuracy of the cards model and the number of extracted features.

3.2 Action Sequence Model

For the action sequence part of the poker game state, another model was built that was quite different from the first one. In the card model, the cards were represented as 52 element

one-hot vectors. This was a straightforward representation but the action sequence was trickier because of its variable length (there could be one action (i.e. a fold) to tens of actions in a single poker game). Hence, there have been multiple iterations to find the model that works best and some of them are discussed below. As mentioned in the related work section, feedforward networks could not be used for variable length sequences and RNNs (LSTMs and GRUs) were used instead. To keep the model simple for initial experiments, the action sequence data only contained actions (check, call, bet, raise, fold to be precise) without accompanying bet sizes. Variable length action sequences were generated strictly based on poker rules for training the model where each action was a 5-element one hot encoded vector.

The first model built was a simple two layer LSTM network (one layer was the encoder and the other was the decoder) shown in figure 4.

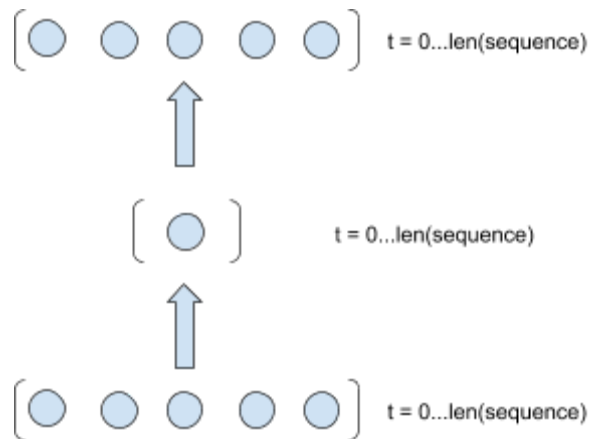


Figure 4: Initial Action Sequence Model. Each circle indicates input/output from an LSTM layer and arrows indicate data flow through an LSTM layer. This diagram shows the simplicity of the first model built.

An action was fed into the encoder per time step which compressed the five features into one extracted feature. Then, the output at each time step was fed into the decoder to get the predicted input. The decoder had a sigmoid activation instead of the usual tanh activation in a LSTM cell

to normalize the predicted input between 0 and 1. The accuracy for this model was measured by checking if each action (5 element one-hot encoded vector) was predicted correctly. This model achieved a 99.9%+ action accuracy while reducing the action state space by five times. However, the main problem with this model was that it only learned features per time step (per action) instead of learning from the entire sequence. A good poker agent needs to learn from the entire sequence because it has information about betting behavior like check-raises.

So the next model was built to learn from the entire sequence (shown in figure 5). As the encoder, this model had an RNN layer that took in a game sequence (batch size of one with variable time steps) and gave out a 50-element vector. This vector was passed through a densely connected feedforward layer which also output a 50-element vector and it constituted the learned features. As the decoder, the learned features were passed through another RNN layer length-of-input-sequence times. The output at each time step from the decoder RNN was the five-element action in the input sequence. The model was trained on 100,000 poker game sequences with a 80-20 training and validation split.

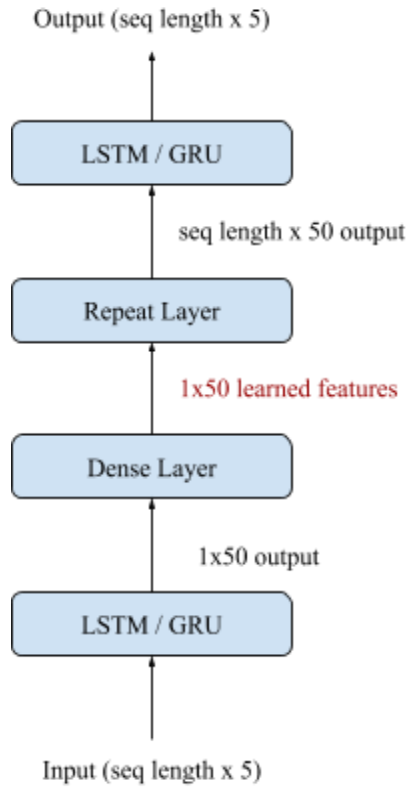
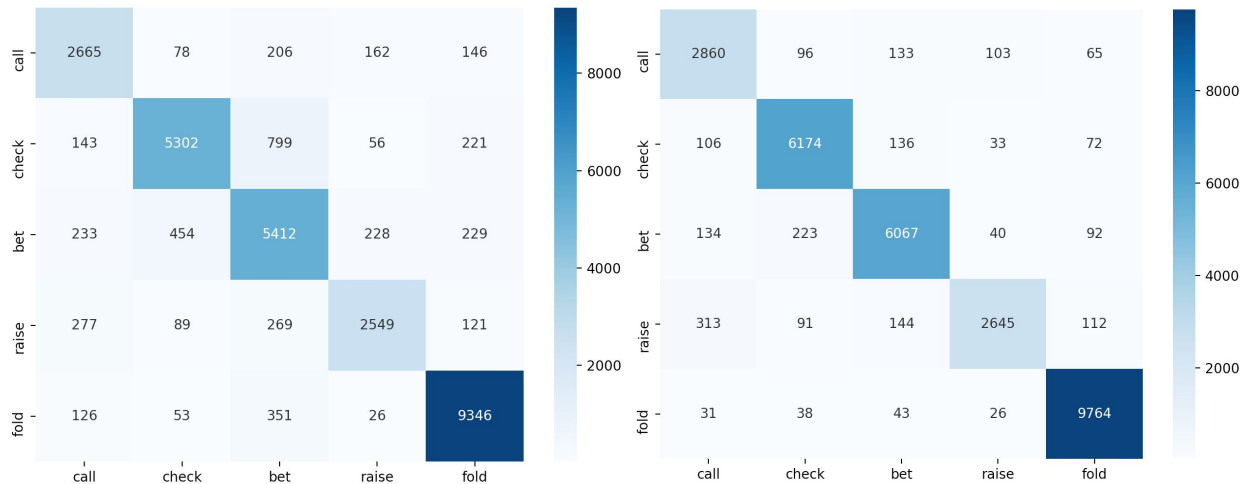


Figure 5: Final Action Sequence Model. The model took one game sequence at a time as input and encoded it into a 50-element vector. This 50-element vector was turned into a sequence length \times 50 element vector (in the Repeat Layer) by simply copying it sequence-length times. This output was finally fed into the decoder RNN to get the original game sequence back.

The reported accuracy for this model was measured from a separate test dataset with 10,000 game sequences and a total of 29,541 actions. The accuracy was just a percentage of correctly predicted actions over the total number of actions. While using LSTMs, the model achieved an accuracy of 85.6% whereas the model using GRUs yielded an accuracy of 93.1% (figure 6 gives the confusion matrices for both the models). Moreover, the model with GRUs took 20% less training time than the model with LSTMs (633 seconds vs 796 seconds per epoch). These results were in line with what Chung et al. [2014] found in their comparison between LSTMs and GRUs.



(a) LSTM Model's Confusion Matrix

(b) GRU Model's Confusion Matrix

Figure 6: Confusion Matrix Comparison. This comparison shows a more uneven distribution of errors (especially between check and bet) in LSTM model than the GRU model.

Although GRUs are simpler than LSTMs, it is interesting to note that they performed significantly better than LSTMs. GRUs were introduced to perform neural machine translation which uses a similar encoder-decoder (sequence-to-sequence) model as the action sequence model [Cho 2014]. The machine translation use case could explain why GRUs were better suited for this model than LSTMs.

4. Comparison Experiments

As mentioned in the beginning of this section, the ultimate goal with the extracted features is to see if they are better than ASHE's hand-crafted features. Due to time and resource constraints, the extracted features from the above two models could not be incorporated in ASHE within this thesis project for a direct comparison with the hand-crafted features. However, they could still be compared by seeing how much information they contain and how they are correlated. To do this comparison, hand-crafted and extracted features were generated for a

sample poker game sequence and then a principal component analysis (PCA) was performed on both the feature sets to reduce the dimensionality to two. Then, the principal components were compared against each other through clustering. Table 1 contains the poker game sequence used for comparison.


















Action	Hole Cards	Community Cards	Round
1. Opponent: bet			Pre-flop
2. Agent: call			Flop
3. Opponent: bet			Flop
4. Agent: raise			Flop
5. Opponent: raise			Flop
6. Agent: raise			Flop
7. Opponent: call			Turn
8. Opponent: bet			Turn
9. Agent: raise			Turn

Table 1: Poker Game Sequence. This table shows a game sequence used for comparison between extracted (learned) features and handcrafted features. This sequence is not complete but it is just long enough to see the differences between the two feature sets.

Before going into the results, first let us look at ASHE’s hand-crafted features. ASHE uses a total of ten features as input to its opponent modeling networks out of which six are

related to the current game state. Two of these six hand-crafted features are ASHE’s total bet and opponent’s total bet [Li 2018]. Since the extracted features did not consider bet sizes, these two features were excluded from the comparison. Therefore, the considered handcrafted features were narrowed down from ten to four (explained in table 2):

Feature Name	Definition
Flush and Straight Draw	Probability of a random hand hitting flush or straight given the board.
Pair(s)	0: no pair on board, 0.5: one pair on board, 1.0: two pairs on board.
Betting Round	One-hot encoding of the betting rounds i.e. preflop, flop, turn and river.
Raw Hand Strength	Probability of ASHE’s hand beating a random hand given the board

Table 2: ASHE’s Handcrafted Features. These are the descriptions of ASHE’s handcrafted features considered in this comparison.

One can easily observe that these features only changed between betting rounds (all of them depend on cards on the table). So without bet sizes, the hand-crafted features did not provide additional information on actions within a betting round (like checks, bets and raises). This can be clearly seen in the PCA comparison between the two feature sets for the above game sequence (shown in table 3).

Action	Round	Hand-crafted PCA	Extracted PCA
1. Opponent: bet	Pre-flop	[0.1412, 1.101]	[16.49, 0.4526]
2. Agent: call	Flop	[-0.5835, -0.1144]	[-1.065, -2.365]
3. Opponent: bet	Flop	[-0.5835, -0.1144]	[-1.133, -2.306]
4. Agent: raise	Flop	[-0.5835, -0.1144]	[-1.143, -2.175]
5. Opponent: raise	Flop	[-0.5835, -0.1144]	[-1.157, -2.105]

6. Agent: raise	Flop	[-0.5835, -0.1144]	[-1.169, -2.05]
7. Opponent: call	Turn	[0.9255, -0.1762]	[-1.677, 0.612]
8. Opponent: bet	Turn	[0.9255, -0.1762]	[-1.672, 0.631]
9. Agent: raise	Turn	[0.9255, -0.1762]	[-1.669, 0.632]

Table 3: Principal Components Of Feature Sets. This table shows the two principal components yielded from a PCA of the handcrafted features and the extracted features of the sample poker game sequence in table 1. Doing a PCA simplifies comparing the two feature sets due to lowered dimensionality.

The hand-crafted features were reduced from seven dimensions (betting round has four dimensions) and the extracted features were reduced from 70 dimensions (50 from action sequence model and 20 from card model) to two principal components. A PCA comparison in this context was just a comparison of the clusters formed in principal components (like measuring distances within a cluster) of both the feature sets. Due to the low dimensionality and sample size, the clusters for these principal components could be seen without applying any clustering algorithms. Table 3 and figure 7 show that the extracted features were clustered in a similar way as the hand-crafted features (based on the betting round). However, the extracted features offered some variability within a cluster (or betting round) whereas the hand-crafted features converged to the same point within a betting round. This showed that extracted features capture more fine-grained information than the hand-crafted features. This finer grained information was coming from the action sequence model.

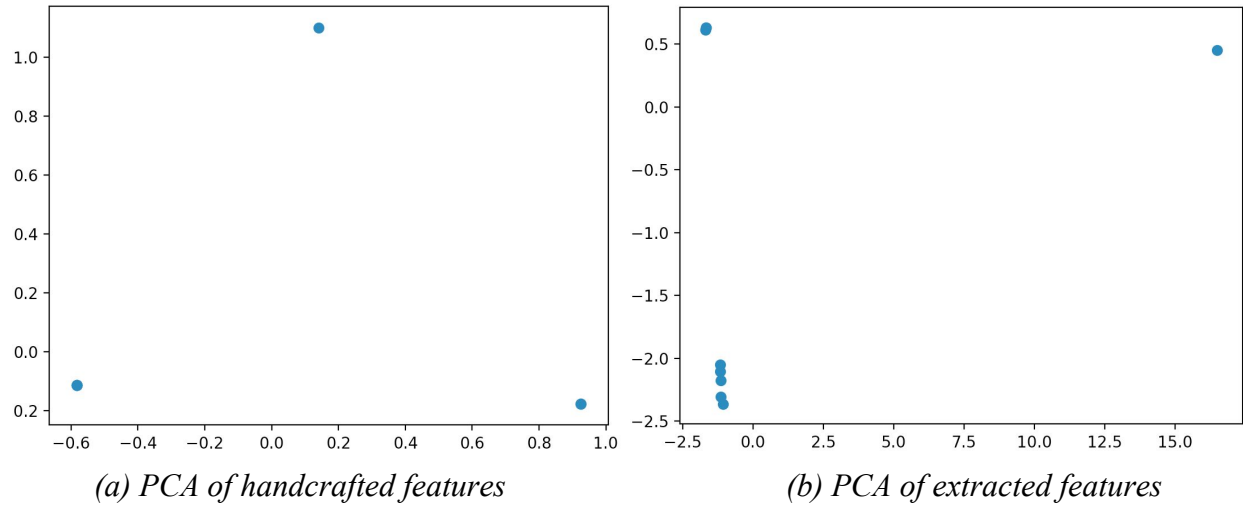


Figure 7: PCA Comparison. Shows the principal components of both the feature sets plotted on a 2D graph. This graph helps to see the clusters and the variability within a cluster.

This comparison shows that the extracted features are meaningful because they are clustered on betting round (just like the hand-crafted features) and they are also better than the hand-crafted features because they contain more information within each cluster or betting round.

5. Future Work

The learned features took the entire poker game state into account except for bet sizes. Learning from bet sizes could give important information like does the opponent want the agent to call or fold, whether it is an all-in, etc. The action sequence model can be modified to include bet sizes. However, making this modification will be challenging because calculating loss for bet sizes (a regression problem) will be different from calculating loss for actions (a classification problem). One possible solution could be to group bet sizes into buckets that are multiples of the pot size.

After the entire game state is represented in the extracted features, the model can be tested out with real poker gameplay. This evaluation can be done either by switching out the handcrafted features in ASHE with the new model or by building a simple poker agent that can be tested with and without the model. The former may take longer than building a simple poker agent due to the training required to integrate this model into ASHE's LSTMs. However, testing on ASHE would give more decisive results than on a simple poker agent.

6. Conclusion

This research is about learning useful features from raw poker game state. These learned features have the potential to improve the performance of ASHE and other neural network approaches to playing poker. Moreover, it can make ASHE more generalizable to use in other incomplete information games. The results show that it is possible to extract a handful of features without losing much information about the game. The results also show that the extracted features are similar to hand-crafted features but they also contain more fine grained information than the latter. This shows that the extracted features are more useful than the hand-crafted features. Although this conclusion is a sign of progress, the model still needs to be tested on actual poker gameplay to see if it works. The finish line is not crossed yet.

Acknowledgements

I would like to thank my thesis supervisor, Dr. Miikkulainen for all the brainstorming and planning sessions which were instrumental in guiding this thesis in the right direction. I would also like to thank Xun Li for spending countless hours on building my foundational knowledge about neural networks, poker and ASHE. Finally, I would like to thank my thesis committee, Dr. Novak and Dr. Van De Geijn for their time and feedback on my tech report.

References

- Li, X., & Miikkulainen, R. (2018). Dynamic adaptation and opponent exploitation in computer poker (unpublished doctoral dissertation). University of Texas, Austin, TX.
- Zinkevich, M., Johanson, M., Bowling, M., & Piccione, C. (2007). Regret minimization in games with incomplete information. *Conference on Neural Information Processing Systems*.
- Bowling, M., Burch, N., Johanson, M., & Tammelin, O. (2017). Heads-up limit hold'em is solved. *Communications of the ACM*, 65(11), 81-88.
- Silver, D., Schrittwieser, J., Simonyan, K., Antonoglou, I., Huang, A., Guez, A., . . . Hassabis, R. (2017). Mastering the game of go without human knowledge. *Nature*, 550, 354-359.
- Papakostas, M., Spyrou, E., Giannakopoulos, T., Siantikos, G., Sgouropoulos, D., Mylonas, P., & Makedon, F. (2017). Deep visual attributes vs. hand-crafted audio features on multidomain speech emotion recognition. *Computation*, 5(2), 26.
- Gibson, R., Szafron, D. (2011). On strategy stitching in large extensive form multiplayer games. *Conference on Neural Information Processing Systems*.
- Yakovenko, N. (2016). Poker and AI: Reporting from the 2016 Annual Computer Poker Competition. *Poker News*. Retrieved from <https://www.pokernews.com/strategy/poker-ai-2016-annual-computer-poker-competition-24246.htm>
- Andy. (2017). Recurrent neural networks and LSTM tutorial in Python and TensorFlow. *Adventures in Machine Learning*. Retrieved from <http://adventuresinmachinelearning.com/recurrent-neural-networks-lstm-tutorial-tensorflow/>

- Davidson, A., Billings, D., Schaeffer, J., Szafron, D. (2000). Improved opponent modeling in poker. *International Conference on Artificial Intelligence*, 1467-1473.
- Lipton, Z. C., Berkowitz, J., Elkan, C. (2015). A critical review of recurrent neural networks for sequence learning. *University of Cornell Library (arxiv)*.
- Hochreiter, S., Schmidhuber, J. (1997). Long short-term memory. *Neural Computation*, 9(8), 1735-1780.
- LeCun, Y., Boser, B. E., Denker, J. S., Henderson, D., Howard, R. E., Hubbard, W. E., & Jackel, L. D. (1989). Handwritten digit recognition with a back-propagation network. *Conference on Neural Information Processing Systems*.
- Moravcik, M., Schmid, M., Burch, N., Lisy, V., Morrill, D., Bard, N., . . . Bowling, M. (2017). DeepStack: Expert-level artificial intelligence in heads-up no-limit poker. *Science*, 355(6320).
- Yakovenko, N., Cao, L., Raffel, C., & Fan, J. (2015). Poker-CNN: A pattern learning strategy for making draws and bets in poker games. *AAAI'16 Proceedings of the Thirtieth AAAI Conference on Artificial Intelligence*, 360-367.
- Cho, K., Merriënboer, B. V., Bahdanau, D., & Bengio, Y. (2014). On the properties of neural machine translation: Encoder-decoder approaches. *Eighth Workshop on Syntax, Semantics and Structure in Statistical Translation*.
- Chung, J., Gulcehre, C., Cho, K., & Bengio, Y. (2014). Empirical evaluation of gated recurrent neural networks on sequence modeling. *University of Cornell Library (arxiv)*.
- Kostadinov, S. (2017). Understanding GRU networks. *Towards Data Science*. Retrieved from <https://towardsdatascience.com/understanding-gru-networks-2ef37df6c9be>