Predicting and Manipulating Game Rules Through Learning in Deep Weight Spaces

Janine Lohse, Kevin Baum, Gerrit Großmann

January 30, 2024

1 Introduction

Learning in deep weight spaces is the process of training models on weight data from other models. This young technique in explainable AI was pioneered by [1, 5] for gaining a better understanding of neural networks and their learning process. In this report, we train a large number of models for playing different variations of a simple card game, and then train meta-models to predict the concrete rule variation from their weight data. This allows us to gain insights about how the game's rules are encoded in the models game data, and how this encoding changes during training. Using this data, we then propose a novel technique for steering a models' behavior by manipulating its weight data using a meta-model. More specifically, we manipulate the models weights for improving its performance in on a specific variation of the game. Surprisingly, intervening with meta-models can lead to a significant improvement of the models' performance, even if the meta-models itself were trained on the weight data of *worse* models.

2 The Card Game

In the beginning of the game, the player obtains 20 cards, each numbered uniformly at random with a number from 1 to 15. There is a *rule matrix* R stating for each pair a, b of card labels with $a \neq b$ whether card a can beat b

or vice versa. ´

$$R_{ab} = \begin{cases} 1 & \text{if card a beats card b} \\ 0 & \text{else} \end{cases}$$

The relation induced by R is not necessarily transitive, i.e. it is possible that a beats b, b beats c, but c beats a. Moreover, $R_{aa} = 0$ and for $a \neq b$, we have that exactly one of the entries R_{ab}, R_{ba} is one.

In each of the 20 rounds, the adversary plays a card and the player has to react with one of their remaining cards. If possible, the player should react with a card that beats the adversary's card. We call a move *legal* if the player plays a card that they still own. We call a move *good* if it is legal and the player plays a card that beats the adversary's card.

For generating game data, we consider a simple strategy: in each round, if a good move is possible, the player chooses a good move among all possible good moves uniformly at random. If no good move is possible, the player uniformly chooses one of their remaining cards to play. We say we *finetune a model for a rule matrix* R if we train the model on game data that was synthesized using the described strategy and rule matrix R. The training objective is to predict the player's next move.

3 Predicting the Rule Matrix

3.1 Weight Data Generation

In order to generate the weight space data for the of, we need to train a large family of models. For being able to compare their weights despite the instability of gradient descent, we first perform a common pre-training on data that is not specific to one rule matrix. That way, the model first learns to predict legal moves, but not yet to predict good moves. This data was generated by randomly choosing a new rule matrix for each game.

Since the full weight data is large, we next identify a subset of the weights to train on. We consider the weights *important* that, on average, change most when finetuning a model for a specific rule matrix. After identifying the important weights, we finetune a large family of models, each model for a different, randomly chosen rule matrix, and extract the important weights after finetuning.

We generated this data for two different model architectures.

	Feedforward	**Transformer**
Legal Moves (pretrained)	98.57%	95.67 %
Good Moves (pretrained)	46.85 %	48.65 %
Accuracy (pretrained)	25.75 %	26.12 %
Legal Moves (finetuned)	99.68 %	95,94 %
Good Moves (finetuned)	92.17 %	55.44 %
Accuracy (finetuned)	42.17 %	28.46 %

Figure 1: The achieved performance of both models on the task of predicting the player's next move. The ratio of good moves is defined as the number of good moves divided by the number of game situations in which a good move was possible. Each move is evaluated individually. For the pretrained model, the ratio of good moves is determined relative to a random rule matrix.

1. The first model is a feedforward neural network with one hidden layer, which, for each move, receives the cards the player had in the beginning and the sequence of cards played so far both by the player and adversary in one-hot-encoding.

2. The second model is a Seq2Seq transformer, whose vocabulary consists of the card types and some meta-tokens. The model gets as input the player's cards and the sequence of cards played by the adversary and has to predict the sequence of cards that were played by the player.

Figure 1 shows the achieved performance of both models on the task of predicting the player's next move. Appendix A shows an overview over the used hyperparameters.

Moreover, we have saved several snapshots of the model's weights during training, creating several datasets of models trained on a different number of games. More specifically, the dataset $D_{a,n}$ contains the important weights of models of architecture a, each trained on n games.

3.2 Results

We trained a linear meta-classifier $M_{a,n}$ on each dataset $D_{a,n}$.

The results for training the meta-classifiers $M_{F,-}$ for predicting the 105 parameters of the rule matrix are shown in Figure 3, the results for the meta-models $M_{T,-}$ are shown in Figure 2. The best validation accuracies are achieved by $M_{F,30.000}$ for the feedforward model (93.3%) and by $M_{T,50.000}$ for

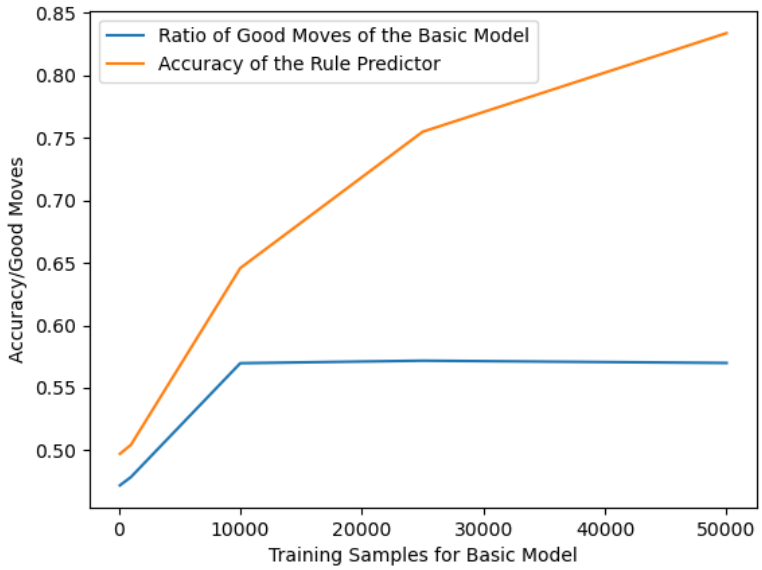

Figure 2: Results for the meta-model trained with weights of transformer models. An orange point (x, y) states that the accuracy of meta-model $M_{T,x}$ is y. A blue point (x, y) states that the feedforward model achieves a ratio of good moves of y after being trained on x games.

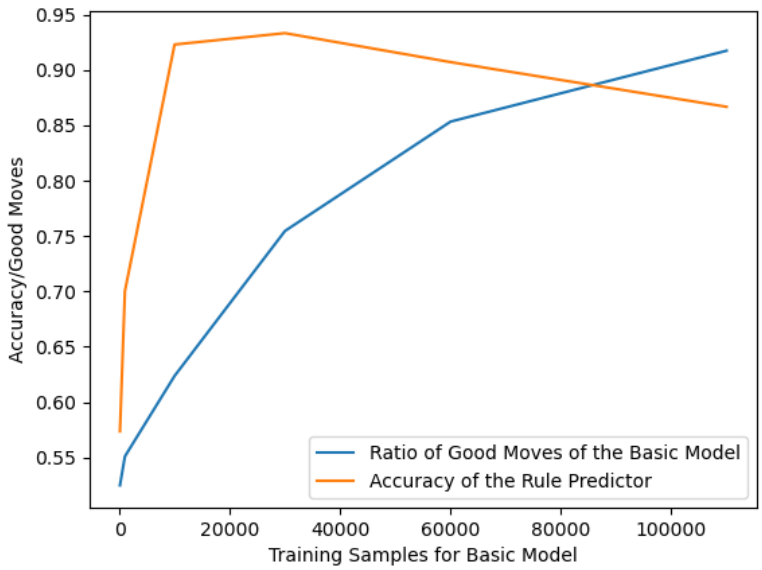

Figure 3: Results for the meta-model trained with weights of feedforward models. An orange point (x, y) states that the accuracy of meta-model $M_{F,x}$ is y. A blue point (x, y) states that the feedforward model achieves a ratio of good moves of y after being trained on x games.

the transformer model (83.5%). This shows that for both models, large parts of the rule matrix can be extracted from a subset of its weights.

Moreover, one can see that even for models that only achieve a low ratio of good moves, it is still possible to achieve a high accuracy when predicting the rule matrix from its weights. This means that large parts the rule matrix are already encoded in the model's weights, even if the models themselves aren't able to use that information for predicting good moves.

Furthermore, one can observe that training meta-models on well-trained feedforward models lead to overfitting - the validation accuracy decreases for the models $M_{F,50.000}$ and $M_{F,110.000}$, while the training accuracy gets close to 1.0.

4 Predicting a Model's Behavior

We have seen that it is possible to extract large parts of the rule matrix, even for models with a low rate of good moves. We are thus predicting something about the model's training data, but not about its behavior. To test if predicting the behavior is also possible, we define the following measure of *preference* for each pair of cards a, b:

$$
\begin{aligned}
n_{ab} &:= \text{number of times } a \text{ was played in reaction to } b \\
n_{ba} &:= \text{number of times } b \text{ was played in reaction to } a \\
p_{a,b} &:= \begin{cases} 0.5 & \text{if } n_{ab} + n_{ba} = 0 \\ \frac{n_{ab}}{n_{ab}+n_{ba}} & else \end{cases}
\end{aligned}
$$

If card a beats card b, n_{ab} should be large and n_{ba} should be small. Otherwise, n_{ba} should be large and n_{ab} should be small. Thus, for a good model, we should roughly have $p_{a,b} \approx R_{a,b}$ if $n_{ab} + n_{ba} > 0$.

We trained a feedforward meta regression model with two layers for predicting the a transformer's preference given the important weights. We achieve a validation R^2-Score of 0.424, which means that roughly 42% of the variance in the models' preferences can be explained by the meta-model trained on their weights.

Training Samples for Basic Model	0	100	1000	10000	25000	50000
Improvement by $M_{T,100}$	7.03%	7.08%	6.47%	1.86%	2.42%	2.57%
Improvement by $M_{T,1000}$	6.18%	6.69%	6.13%	2.51%	1.94%	2.27%
Improvement by $M_{T,10000}$	8.06%	6.70%	7.00%	2.26%	1.87%	2.77%
Improvement by $M_{T,25000}$	5.63%	4.44%	4.82%	2.12%	1.90%	2.26%
Improvement by $M_{T,50000}$	3.54%	3.30%	3.53%	1.31%	1.63%	1.50%

Figure 4: Average improvement (percent points) in the ratio of good moves by intervening using different meta-models and a different number of training samples for the basic model. All results are statistically significant (p-value less than 0.05).

5 Steering a Model's Behavior using the Meta-Classifier

To examine whether the meta classifier learnt a causal relationship between the model's weights and the rule matrix, we used the classifier to intervene on the model's weights. These interventions were inspired by [2], who show that it is possible to change a model's internal game state representations by changing the neurons' activations. In an intervention, we optimize a model's important weights to obtain a certain rule matrix as the output of our meta-classifier, and then evaluate the updated model. The strength of the intervention is chosen such that the resulting ratio of good moves is as high as possible - if the intervention is too weak, its effect is too small, but if its too strong, it is possible to destroy the model's ability to make legal moves (which also reduces the ratio of good moves, since good moves are always legal)

Figure 4 shows the average improvement achieved by the intervention for different meta-models trained on transformers' weights. Surprisingly, the improvement is generally larger when using meta-models that were trained on weight data of models that itself were trained on only a few samples. Intervening with the meta-models $M_{T,100}, M_{T,1000}$ and $M_{T,10000}$ leads to a improvement that is similar to the improvement achieved by finetuning a model on 10.000 games or more. Moreover, all meta-models are able to significantly improve the performance of already fully finetuned models. It is remarkable that intervening with meta-models can lead to an significant improvement, even through the meta-models itself were trained on the weight data of *worse* models.

6 Related Work

Learning in deep weight spaces is a young research area that was pioneered by [1] and [5].

In [1] Eilertsen et al. train a large number of models on varying datasets, with varying optimization procedures and different architectures. Next, a meta-classifier is trained on either a random selection of consecutive weights of those models, or on statistical measures of all weights. The meta-classifier is trained to predict the concrete training variation. They find that in their setting, training on weight statistics generally leads to a higher accuracy than training directly on a (subset of) the models' weights.

Underthiner et al. [5] train a meta-classifier to predict a model's accuracy. For generating the weight data, they consider CNNs trained with different hyperparameters on standard image classification datasets. Just as [1], their experiments show that training on weight statistics leads to better results than directly training on the models' weights. Notably, they can show that their meta-classifier, despite being trained on small CNNs only, can also rank large ResNet models.

Both experiments do, in contrast to ours, not involve common pretraining, which makes their meta-classifier more broadly applicable. However, they also do not consider interventions as discussed in Section 5

A recent line of work [3, 6] focuses on creating architectures explicitly for training in deep weight spaces, either by exploiting known symmetries of the weights of feedforward networks [3], or by adapting the transformers' attention mechanism [6].

Finally, [4] addresses the problem of data generation for training in deep weight spaces. They study how to augment existing data in order to reduce the number of models that have to be trained.

7 Future Work

- We can not yet fully explain the results of the interventions. Why do the meta-models that are trained on the weights of less trained models achieve better results when intervening? Examining this behavior in more, different settings could help in gaining a better understanding of this phenomenon - and possibly deep learning in general.

- The related work found it to be more successful to train on weight

statistics instead of on the weights itself. It would be interesting to test whether this is also true in our setting. We would suspect that the rule matrix, in contrast to accuracy or learning hyperparameters, is not well predictable from the weight statistics.

- In order to reduce the number of models to train for generating the weight data, one might use, for the meta-classifier, a model architecture that was designed for being trained on weight spaces [3, 6]

- It would be interesting to further explore the behavioural meta classification/regression. We only tested one way for measuring the behaviour/preferences of a model, but it might not be the most natural choice. Maybe there is a measure that can better capture the model's behaviour.

- We assumed that the weights that change most during finetuning are most relevant for predicting the rule matrix. While this is a natural assumption, it would be great to actually test whether this is really true by comparing our results to the accuracy values when training on a random selection of weights, a random selection of consecutive weights, or the weights that change least during finetuning.

- Instead of training on a subset of the weights, principal component analysis could be another promising way of reducing the dimension of the weight space, a way that potentially leads to less information loss than just selecting a subset of the weights.

- So far, we only used interventions to steer the behaviour of a model that was not finetuned or finetuned for the rule matrix that we want to have. It would be interesting to test if it is possible to 'change' the rule matrix of a model that was already finetuned.

- We assumed that we can only compare the model's weights if we do a common pretraining. It would be interesting to test if this is really the case. If analyzing the weight spaces is even possible without common pretraining, this might open up new possibilities for analyzing existing models.

- Our experiment was conducted using synthetic data only. It would be very interesting to try a similar experiment on real-world data. Here

is a potential setup: Take data from Amazon reviews that is labeled with at least two kinds of labels. Train a large number of language models (can be something pretrained, but should be small), each only on reviews with a specific label combination. Then, try to predict the label combination from the weight data. Each individual label should appear at least twice in the training data.

If such an experiment works, this would mean that we can find out something about the kind of text data a model was trained on just by looking at its weights, which would be very exciting.

References

[1] Gabriel Eilertsen, Daniel Jönsson, Timo Ropinski, Jonas Unger, and Anders Ynnerman. Classifying the classifier: Dissecting the weight space of neural networks. In Giuseppe De Giacomo, Alejandro Catalá, Bistra Dilkina, Michela Milano, Senén Barro, Alberto Bugarín, and Jérôme Lang, editors, *ECAI 2020 - 24th European Conference on Artificial Intelligence, 29 August-8 September 2020, Santiago de Compostela, Spain, August 29 - September 8, 2020 - Including 10th Conference on Prestigious Applications of Artificial Intelligence (PAIS 2020)*, volume 325 of *Frontiers in Artificial Intelligence and Applications*, pages 1119–1126. IOS Press, 2020.

[2] Kenneth Li, Aspen K. Hopkins, David Bau, Fernanda B. Viégas, Hanspeter Pfister, and Martin Wattenberg. Emergent world representations: Exploring a sequence model trained on a synthetic task. In *The Eleventh International Conference on Learning Representations, ICLR 2023, Kigali, Rwanda, May 1-5, 2023*. OpenReview.net, 2023.

[3] Aviv Navon, Aviv Shamsian, Idan Achituve, Ethan Fetaya, Gal Chechik, and Haggai Maron. Equivariant architectures for learning in deep weight spaces. In Andreas Krause, Emma Brunskill, Kyunghyun Cho, Barbara Engelhardt, Sivan Sabato, and Jonathan Scarlett, editors, *International Conference on Machine Learning, ICML 2023, 23-29 July 2023, Honolulu, Hawaii, USA*, volume 202 of *Proceedings of Machine Learning Research*, pages 25790–25816. PMLR, 2023.

[4] Aviv Shamsian, David W. Zhang, Aviv Navon, Yan Zhang, Miltiadis Kofinas, Idan Achituve, Riccardo Valperga, Gertjan J. Burghouts, Efstratios Gavves, Cees G. M. Snoek, Ethan Fetaya, Gal Chechik, and Haggai Maron. Data augmentations in deep weight spaces. *CoRR*, abs/2311.08851, 2023.

[5] Thomas Unterthiner, Daniel Keysers, Sylvain Gelly, Olivier Bousquet, and Ilya O. Tolstikhin. Predicting neural network accuracy from weights. *CoRR*, abs/2002.11448, 2020.

[6] Allan Zhou, Kaien Yang, Yiding Jiang, Kaylee Burns, Winnie Xu, Samuel Sokota, J. Zico Kolter, and Chelsea Finn. Neural functional transformers. *CoRR*, abs/2305.13546, 2023.

A Hyperparameter Overview

Feedforward		**Transformer**	
Batch Size	8	Batch Size	64
Learning Rate	0.001	Learning Rate	0.001
Hidden Dimension	400	Embedding Size	28
Number of Layers	2	Number of Layers	2
		Attention Heads	1
Num. Games for Pretraining	10,000	Num. Games for Pretraining	200,000
Num. Games for Finetuning	110,000	Num. Games for Finetuning	50,000
Num. Models in Train Set	1,000	Num. Models in Train Set	10,000
Num. Models in Test Set	100	Num. Models in Test Set	100
Num. Important Weights	10,000	Num. Important Weights	10,000

The meta-classifier is trained for 500 epochs with a batch size of 4.