Watching a Language Model Learning Chess

Andreas Stöckl
University of Applied Sciences Upper Austria
Digital Media Department
Hagenberg, Austria
`andreas.stoeckl@fh-hagenberg.at`

Abstract

We analyse how a transformer-based language model learns the rules of chess from text data of recorded games. We show how it is possible to investigate how the model capacity and the available number of training data influence the learning success of a language model with the help of chess-specific metrics. With these metrics, we show that more games used for training in the studied range offers significantly better results for the same training time. However, model size does not show such a clear influence. It is also interesting to observe that the usual evaluation metrics for language models, predictive accuracy and perplexity, give no indication of this here. Further examination of trained models reveals how they store information about board state in the activations of neuron groups, and how the overall sequence of previous moves influences the newly-generated moves.

1 Introduction

Language models are now used for a variety of applications that are not, or not directly, related to *Natural Language Processing* tasks, and process data that is not text Parmar et al. (2018); Huang et al. (2018); Dhariwal et al. (2020). Lu et al. (2021) used a so-called Frozen Pretrained Transformer (FPT) to study finetuning on a variety of sequence classification tasks spanning numerical computation, vision, and protein fold prediction.

In this article, we use state-of-the-art methods for language models in an area that at first glance does not seem to be an application area for them, namely the area of computer chess. This is because clear rules determine what happens here, rather than the ambiguities and vagueness that characterise language.

Brown et al. (2020) demonstrated, among other things, that a language model with increasing model capacity is able to learn the rules of arithmetic to a certain degree by training it with the data of crawled websites. In the process, elementary operations were learned in a certain number space, but not beyond. Is this limitation due to the lack of capacity of the model, insufficient training time or training data that did not contain sufficient information?

In Nogueira et al. (2021), it was demonstrated that regardless of the number of parameters and training examples, *Transformer* Vaswani et al. (2017) models are unable to learn addition rules that are independent of the length of the numbers seen during training.

To test the ability of language models to learn rules, and assess the influence of model size, training time and available training data, we use the commonly-stressed field of "computer chess" as an example. We investigate whether a language model is able to learn the rules of chess only from the records of games played by humans.

The test area is well suited for studying the training process of language models, since training data is available in large quantities here thanks to the recording of chess games on the booming internet chess servers. Another reason is that the quality of the language model can not only be assessed with the usual evaluation metrics for language models such as perplexity, but also with the help of the chess rules to check whether correct games are generated.

Chess as an AI testing ground has also been very popular for decades, and it regained strong focus a few years ago, thanks to the work of Deepmind[1] Silver et al. (2018, 2017); Tomašev et al. (2020) together with the game of GO. There, only starting from the rules with new techniques of reinforcement learning, a chess engine was created that sur-

[1] https://www.deepmind.com/

passed everything that had previously existed in terms of playing strength.

In Schrittwieser et al. (2020), as a continuation of the previous work, knowledge of the rules is now not even assumed. When evaluating Go, Chess and Shogi, without knowledge of the rules of the game, the new algorithm *MuZero* achieved the superhuman performance of *AlphaZero*[2] of the earlier work, which was trained with the rules of the game.

We will use a completely different approach as a starting point to create a system that discovers the rules of chess itself. Only transcripts of games played will be used to train language models. We will then inspect the models to ascertain how the system has learned the rules of chess.

2 Formulation of the Problem

We formulate the problem of learning to play chess within the framework of the usual methodology for language models. A language model is made capable of writing texts by training it with data comprising natural language. It achieves this by completing given passages of text, inserting word by word or part of word by part of word that is likely to be next.

In recent years, major progress has been made in this field thanks to the use of neural networks. These very powerful models are not only able to form syntactically correct sentences, but also to keep the context correct across several paragraphs, thus producing texts that are almost indistinguishable from those written by humans.

This has become possible because the new model architectures called *Transformers* Vaswani et al. (2017); Alammar (2018) are able to capture dependencies in the texts over long distances, and sufficient training material is also available from the WWW. In order to have a supervised training setting, the systems are either fed with text as input to guess the next word (casual language modelling), or words in a whole sentence are masked, which the model then has to reconstruct (masked language modelling). The best-known representative of the first type is the family of GPT models Radford et al. (2018, 2019); Brown et al. (2020) and for that of the second type BERT Devlin et al. (2018) and its many relatives.

We will use the *GPT2* model in different model sizes as a basis and train them with chess data. This data is often in the so-called Portable Game Notation (PGN) format[3]. These are text files containing some metadata, such as the names of the players, the date, the ELO rating[4] and more, and the transcription of the actual game in Standard Algebraic Notation[5]. This part is a string that can be seen like a sentence of a natural language. The individual moves form the "words" of the sentence.

Example:

"d4 d5 Nf3 Nf6 e3 Bf5 Nh4 Bg6 Nxg6 hxg6 Nd2 e6 Bd3 Bd6 e4 dxe4 Nxe4 Rxh2 Ke2 Rxh1 Qxh1..."

Adding a new word to the *SAN* string is equivalent to making a chess move. The context that a language model has in the form of the preceding words for prediction contains all of the information needed to generate the state of the chessboard. Accordingly in principle it should be possible to create a model that predicts all legal moves of a position with positive probability and all illegal moves with probability near 0.

3 Data and Pre-Processing

For training the language model, we need a large amount of game data containing legal moves. We can download these from the internet chess server *Lichess*[6], for example. All games played on the server since 2013 are offered there, grouped by month. A compressed PGN file is available for each month. Overall, over 400 GB of compressed data with over 1.7 billion games played.

This is a sufficiently large amount of data, even if the amount becomes much smaller after pre-processing (e.g. removing metadata). Considering that the original GPT2 language model was trained with 40GB of internet text, we have sufficient room to experiment with different amounts of data during training.

The quality of the games played plays a subordinate role in this research, as it is initially only about learning the rules. However, it would be possible to filter games via the metadata of the ELO values of the players and examine the influence on the playing strength. However, in order to avoid games that were abandoned early on, sometimes after only one move on the server, we will use a minimum length

[2]https://deepmind.com/blog/article/alphazero-shedding-new-light-grand-games-chess-shogi-and-go algorithm

[3]http://www.saremba.de/chessgml/standards/pgn/pgn-complete.htm

[4]https://en.wikipedia.org/wiki/elo_rating_system

[5]https://www.chessprogramming.org/Algebraic_Chess_Notation

[6]https://database.lichess.org/

for filtering. The pre-processing of the batch of data is undertaken with a command line programme for manipulating the PGN files, which masters almost all of the required steps. *pgnextract*[7] can perform the required transformations in a reasonable time even with large amounts of data.

In these cleansings, all move numbers, results, comments, variations, etc. are removed from the games to obtain only the pure string with the SAN notation. One line per game is written to a file. All games with fewer than 20 moves are also filtered.

4 Related Work

A very similar approach was followed in Noever et al. (2020). With slightly different pre-processing, a model also based on GPT2 was trained from game data (11,000 games and 2.19 million games). After 30,000 training steps, plausible looking games could be generated, but with about 10 percent illegal moves in the games. It was also clearly observed that fewer errors occurred in the early stages of the game, which is obvious due to the lower context required. In our experiments, we will investigate positions after different number of moves from the starting position.

Jhamtani et al. (2018) use text generation not to generate the games themselves, but rather to generate comments in text form. A specially created data set and an LSTM (Hochreiter and Schmidhuber, 1997)-based neural network are used for training.

In our work, as in Kaplan et al. (2020); Henighan et al. (2020) we will also investigate the relationship between the development of test loss in language models as a function of model size, computing capacity and data volume. In their work, power laws were found to be observed very precisely over seven orders of magnitude, showing that the result benefits from a scale up as long as the sizes are increased simultaneously and there is no bottleneck at one. If the model sizes in terms of the number of parameters are increased by eight times, the amount of data has to be increased by eight times. We will investigate the relationships with chess-specific metrics, such as the number of correctly-generated moves.

Before we start fine-tuning the model with the games data, we would like to test whether the GPT2 models of different sizes trained with English language files are inherently capable of continuing chess games. Since the training texts comprising web crawls certainly contained chess games in algebraic notation, this could be possible.

Sequences generated with the GPT2 small, for example, if started with “e4 d5”, look like this: “e4 d5 e5 e8 8 f8) e5 e3 f8 e9 f8 f8) 9 0-0-0-0-0-”. While this looks a little like a game of chess, but they are hardly correct moves.

Games generated with the medium GPT2 model look like this, for example: e4 d5 18. f6 Nc5 19. Nf3 Nd4 20. Rg1 Nd6 This looks more like chess notation, but it still does not contain correct moves. This example also shows a problem that the model has with the notation, given that games are published very differently on the web and therefore also in the GPT2 training data. They are partly with numbering and partly without, which is also reflected in the example.

The *Large Model* and the *XL Model* do not give better results. We suspect that this is due to insufficient and inconsistent lot data in the training data set. A larger model capacity does not bring any progress here.

5 Training and Evaluation of the Models

The hardware used for the training was a Kubernetes cluster Brewer (2015) with NVIDIA RTX 3090 GPUs, each with 24 GB of video RAM and 256 GB of main memory.

For the implementation, the *Transformers* package Wolf et al. (2020) from *HuggingFace*[8] based on Pytorch Paszke et al. (2019) was used.

The learning rate search was conducted according to Smith (2018). We start fine-tuning the model as the learning rate increases from very low to very high, and stop when the loss starts to truly become out of control.

Fastai Howard and Gugger (2020) was used for the training, using the *1-cylce-policy* Smith and Topin (2019). [9]

With batch sizes just fitting on the GPU memory, the models were saved after some epochs of training for the evaluations.

When language models generate sequences of words, the same sequence will always emerge if the word with the highest probability is always chosen next. Furthermore, the models often tend to repeat sequences of words. This also applies to the

[7]https://www.cs.kent.ac.uk/people/staff/djb/pgn-extract/

[8]https://huggingface.co/transformers/

[9]https://sgugger.github.io/the-1cycle-policy.html

generation of chess games here. Therefore, random mechanisms such as *top-k sampling* Fan et al. (2018) and *top-p sampling* Holtzman et al. (2019) are used to generate the games. These techniques reduce the tendency of repetition, although it can still occur, as Welleck et al. (2020) have investigated.

To evaluate the models, games are generated in different ways:

- From a list of typical opening positions after two moves.
- From positions of games from a game data set after a given number of moves.
- From randomly-generated positions after a given number of moves

For all of these games, the average number of correct moves generated is counted. These three chess-specific metrics for assessing the generated moves pose different challenges to the language model. For the first evaluation criterion, it is easiest to generate legal moves, since all test positions were included in a large number of the training data games, and therefore it is sufficient for the model to remember the data. A generalization in the form that the rules of chess were actually learned is only necessary for very long generated move sequences.

The second method presents more of a challenge, increasing as the length of the given number of moves increases. Since the game data set used for the test is not included in the training data, as the length of the given moves increases, increasingly more positions will appear that the model has never seen before. Therefore, the model has to learn the rules to generate valid moves.

The third metric uses starting positions generated by a random sequence of moves. A large proportion of these moves have therefore never appeared in human games, nor in the test data set. Furthermore, the move patterns that appear are very different from those in human games, as well as from those in conventional chess programs. It is therefore very difficult for the model to generate regular moves for these sequences. Even for humans, handling such random positions is very difficult. Chase and Simon (1973) has found in experiments with chess grandmasters and amateurs that while good chess players can easily remember typical positions, they have problems with random positions.

For each trained model, the training loss, validation loss, accuracy predicting the moves in the data and perplexity are also calculated.

6 Results

For encoding, we use byte pair encoding, and therefore a typical chess game of 50 moves from both sides requires about 200 tokens for encoding the whole game. However, a game can be much longer. We use a maximum sequence length of 256 and cut of the rest of the moves.

We trained different model sizes of GPT2 (small, medium, large) with different numbers of games (99,604, 577,202, 2,163,417 games) to investigate the influence of the two factors on the learning process. To assess the results, the models were each subjected to an evaluation after a few epochs, using the evaluation metrics described in the previous section. Appendix A-1 shows how the predictive accuracy of the language model evolves with the number of GPU training hours.

The small amount of training data leads to a strong increase in accuracy for all three model sizes after only a few days of training. With more data, no model shows this increase. The different models seem to learn at about the same rate, with the small model being slightly slower.

Alternatively, if we look at perplexity as an evaluation measure, the same picture emerges. All models with a small training data set lead to a faster drop in perplexity, which indicates a better prediction of the language model. All other combinations of model size and amount of data seem to perform the same.

We now want to investigate whether the models with little data are able to learn the chess rules faster and whether it truly makes no difference with the other combinations.

With the chess-specific metrics, we can get to the root of this, and look at the performance of the models as a function of training time.

From five typical opening positions after one move by both sides, five games were generated with the models with top-p sampling (p=0.92), and then it was checked how many moves were correct until the first incorrect move was made. The average of these 25 games was calculated. Appendix A-2 shows how this performance for the respective models developed with the training time.

Top-p sampling is used to check the models' ability to produce not only the most likely move, but also other valid moves. In a chess position, more than one move is usually possible.

For each combination of models and data set, the result was plotted over the training time and a logarithmic fit was drawn. No data are available for the GPT2 large model with the larger amount of data because hardly any models could be trained in the available time period due to the long training time per epoch.

In contrast to the accuracy, this measure shows that the models with a small amount of data perform significantly worse.

The best values are delivered by the medium model with medium data volume. This is in sync with the observation in Kaplan et al. (2020) that model size and data volume should be increased together for good results.

The second metric is a more demanding task, as the games are not generated from a typical starting position after one move, but rather from positions after ten moves, taken at random from games played. These games were not from the training data set.

Games are generated from 100 random positions using top-p sampling (p=0.92), and then tested to see how many moves were correct from this position, whereby the mean value is then calculated. Appendix A-3 shows how this performance for the respective models developed with the training time. Again, a logarithmic fit was drawn.

For this task, all models benefit from more training data, although there are no differences in model size.

As a third metric, we have chosen a task that is even more challenging, and it should make it clearer whether the task was solved based on learned rules rather than pure memorisation of variants.

Again, starting with top-p sampling (p=0.92), 100 games were generated from one position after ten random moves, and then checked to see how many moves were correct from this position, whereby the mean value was then calculated. Appendix A-4 again shows how this performance for the respective models developed with the training time.

It can be seen that this task is much more difficult, as the models now only manage to correctly execute sequences of a few moves on average. However, the same effect can be seen as with the two previous metrics, namely that all models benefit from a larger amount of data. In terms of model size, there are few differences.

7 How Is Chess Knowledge Stored in the Model?

So far, we have explored how language models benefit from more parameters in the model and more training data when learning chess rules. Now we want to ascertain whether there are patterns in how the information of the chess rules is stored in the parameters of the models. We will use different visualisations for this purpose.

For this purpose, different approaches help to visualise language models. On the one hand, we look at the influence of different inputs on the generated words/moves, as shown in Arrieta et al. (2020); Li et al. (2015), and on the other hand, we can look at the activation of the different neurons in the models, as shown in Karpathy et al. (2015); Poerner et al. (2018); Dalvi et al. (2019).

When properly visualised and studied, neuron activations can reveal the roles played by individual neurons and groups of neurons. We use the *Ecco* library[10] Alammar (2021) for analysis.

In order to combine the groups of neurons involved in the same tasks, factorisation methods for matrices are used, whereby the library used employs NMF for this purpose.

Let us look at the inner workings of a trained model for moves from some sample positions. The first position is from the opening phase and it is a special situation where only one legal move is possible (Fig. 1).

If we look at the influence of the individual parts of the sequence in Fig. 2, the moves that led to position Fig. 1, on the new move, we see that the last parts have the strongest influence, but otherwise the entire sequence also influences the output.

The colour code shows the strength of the influence, and alternatively we can also show the influence of the individual parts as a percentage (Fig. 3).

How certain is the model that the generated move is a correct one?

For this purpose, we look at the probabilities that the model assigns to the individual possible tokens at the end of the last layer of the generator part. The move is generated in two parts, and it shows a high probability of the generated move for both

[10]https://www.eccox.io/

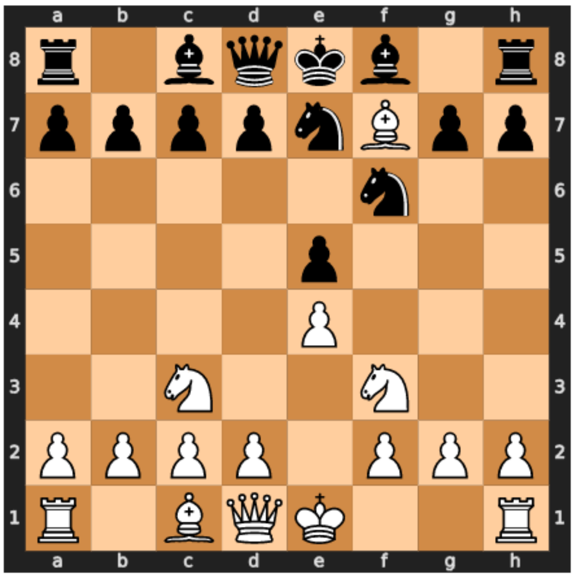


Figure 1: Position from the opening pair with exactly one possible move.

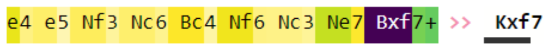
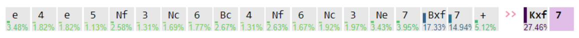


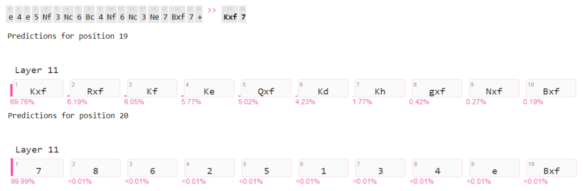
Figure 2: Sequence of moves leading to position in Fig. 1



```
e      4      e      5      Nf     3      Nc     6      Bc     4      Nf     6      Nc     3      Ne     7      Bxf    7      +      >> Kxf    7
3.48%  1.82%  1.82%  1.13%  2.58%  1.31%  1.69%  1.77%  2.67%  1.31%  2.63%  1.67%  1.92%  1.97%  3.43%  3.95%  17.33% 14.94% 5.12%     27.46%
```

Figure 3: Sequence of moves leading to position in Fig. 1 with percentages

parts, but still far from 100 percent for the first part (Fig. 4).



```
e 4 e 5 Nf 3 Nc 6 Bc 4 Nf 6 Nc 3 Ne 7 Bxf 7 + >> Kxf 7

Predictions for position 19

 Layer 11
1 Kxf    2 Rxf   3 Kf    4 Ke    5 Qxf   6 Kd    7 Kh    8 gxf   9 Nxf   10 Bxf
69.76%   6.19%   6.05%   5.77%   5.02%   4.23%   1.77%   0.42%   0.27%   0.19%
Predictions for position 20

 Layer 11
1 7      2 8     3 6     4 2     5 5     6 1     7 3     8 4     9 e     10 Bxf
99.99%   <0.01%  <0.01%  <0.01%  <0.01%  <0.01%  <0.01%  <0.01%  <0.01%  <0.01%
```

Figure 4: Predicted tokens for moves in position Fig. 1

Let us now look at the activations of the neurons and try to see how the model stores information about the chessboard from the move sequences. In order to generate valid moves, the language model needs a representation of the chessboard and its pieces in the states of the neurons.

If we look at the activation of the neurons by group, we see that one group (red in Fig. 5) is active on the parts of the text responsible for the row information on the chessboard, and another for the column and piece information (blue in Fig. 5). The other two are active at the beginning or end of the sequence.

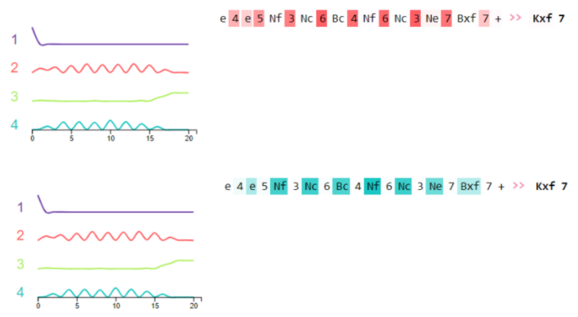


Figure 5: Activations of groups of neurons in position Fig. 1

Let us now look at another position, which comes from the so-called "Game of the Century"[11]. In the complicated position from the middle game in Fig. 6 with many possible moves, the model reaches its limits. It recognises that a piece on the d-file was captured last and wants to capture it back with Rxd or Qxd. However, there is no valid move for this in this position. The representation on the board was not correctly mapped in the neurons here, so no valid moves are generated.

The probability for possible next tokens in the last layer of the network is also distributed over

[11]https://en.wikipedia.org/wiki/The_Game_of_the_Century_(chess)

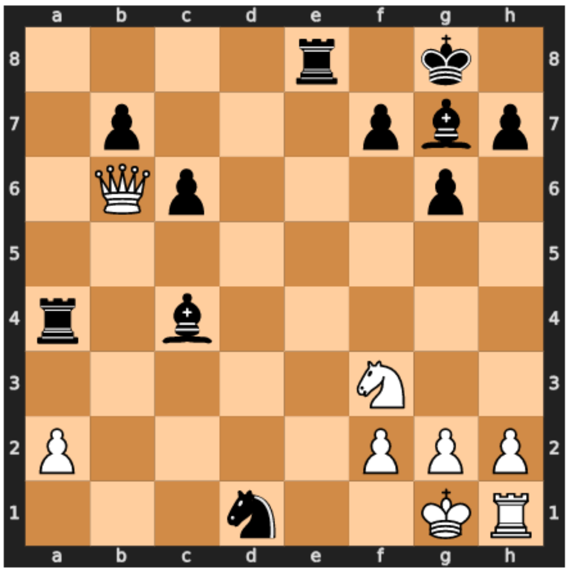


Figure 6: Position from the middle game.

many candidates (Fig. 7).

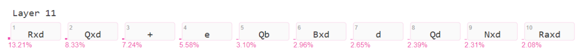
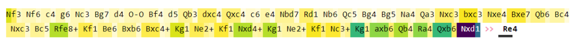


Figure 7: Predicted tokens for moves in position Fig. 6.

As in position 1, virtually the entire sequence of previous moves has an influence on the new move to be generated, as can be seen in Fig. 8.



```
Nf3 Nf6 c4 g6 Nc3 Bg7 d4 O-O Bf4 d5 Qb3 dxc4 Qxc4 c6 e4 Nbd7 Rd1 Nb6 Qc5 Bg4 Bg5 Na4 Qa3 Nxc3 bxc3 Nxe4 Bxe7 Qb6 Bc4
 Nxc3 Bc5 Rfe8+ Kf1 Be6 Bxb6 Bxc4+ Kg1 Ne2+ Kf1 Nxd4+ Kg1 Ne2+ Kf1 Nc3+ Kg1 axb6 Qb4 Ra4 Qxb6 Nxd1 >> Re4
```

Figure 8: Sequence of moves leading to position in Fig. 6.

In this example, again the specialisation of groups of neurons on the row information and the figure and column information can be seen. However, the activations are lower at the beginning of the sequence.

8 Conclusion and Future Work

In the two example positions, it can be seen that the whole sequence has an influence on the generated move, which is necessary to generate correct moves.

By looking at the activation of neurons, we could see that the information about the row, column and type of figure is stored in different groups of neurons. Thus, the model seems to organise the storage of the information necessary to represent the state of the board. So the training could benefit from a longer string representation of the games such as the long algebraic notation since the mechanics of moves are more directly accessible. Because rows, columns, and pieces are all represented as separate tokens.

In order to study the learning process for larger models and larger data sets we would like to use distributed training with Falcon (2019) with higher computational capacity, and the optimizations proposed in Rajbhandari et al. (2020); Rasley et al. (2020); Zhang and He (2020); Ren et al. (2021); Tang et al. (2021); Rajbhandari et al. (2021); Li et al. (2021).

We have only investigated the learning of the rules of the game here. Accordingly, investigating the possible playing strength of the language models as a function of the training data quality would be an interesting extension of the investigations. The influence of the selection of the training data according to the playing strength of the players involved and the model size would have to be considered here.

References

J Alammar. 2021. Finding the words to say: Hidden state visualizations for language models.

Jay Alammar. 2018. The illustrated transformer. *The Illustrated Transformer–Jay Alammar–Visualizing Machine Learning One Concept at a Time*, 27.

Alejandro Barredo Arrieta, Natalia Díaz-Rodríguez, Javier Del Ser, Adrien Bennetot, Siham Tabik, Alberto Barbado, Salvador García, Sergio Gil-López, Daniel Molina, Richard Benjamins, et al. 2020. Explainable artificial intelligence (xai): Concepts, taxonomies, opportunities and challenges toward responsible ai. *Information Fusion*, 58:82–115.

Eric A Brewer. 2015. Kubernetes and the path to cloud native. In *Proceedings of the sixth ACM symposium on cloud computing*, pages 167–167.

Tom B Brown, Benjamin Mann, Nick Ryder, Melanie Subbiah, Jared Kaplan, Prafulla Dhariwal, Arvind Neelakantan, Pranav Shyam, Girish Sastry, Amanda Askell, et al. 2020. Language models are few-shot learners. *arXiv preprint arXiv:2005.14165*.

William G Chase and Herbert A Simon. 1973. Perception in chess. *Cognitive psychology*, 4(1):55–81.

Fahim Dalvi, Avery Nortonsmith, Anthony Bau, Yonatan Belinkov, Hassan Sajjad, Nadir Durrani, and James Glass. 2019. Neurox: A toolkit for analyzing individual neurons in neural networks. In *Proceedings of the AAAI Conference on Artificial Intelligence*, volume 33, pages 9851–9852.

Jacob Devlin, Ming-Wei Chang, Kenton Lee, and Kristina Toutanova. 2018. Bert: Pre-training of deep bidirectional transformers for language understanding. *arXiv preprint arXiv:1810.04805*.

Prafulla Dhariwal, Heewoo Jun, Christine Payne, Jong Wook Kim, Alec Radford, and Ilya Sutskever. 2020. Jukebox: A generative model for music. *arXiv preprint arXiv:2005.00341*.

WA Falcon. 2019. Pytorch lightning. *GitHub. Note: https://github.com/PyTorchLightning/pytorch-lightning*, 3.

Angela Fan, Mike Lewis, and Yann Dauphin. 2018. Hierarchical neural story generation. *arXiv preprint arXiv:1805.04833*.

Tom Henighan, Jared Kaplan, Mor Katz, Mark Chen, Christopher Hesse, Jacob Jackson, Heewoo Jun, Tom B Brown, Prafulla Dhariwal, Scott Gray, et al. 2020. Scaling laws for autoregressive generative modeling. *arXiv preprint arXiv:2010.14701*.

Sepp Hochreiter and Jürgen Schmidhuber. 1997. Lstm can solve hard long time lag problems. In *Advances in neural information processing systems*, pages 473–479.

Ari Holtzman, Jan Buys, Li Du, Maxwell Forbes, and Yejin Choi. 2019. The curious case of neural text degeneration. *arXiv preprint arXiv:1904.09751*.

Jeremy Howard and Sylvain Gugger. 2020. Fastai: A layered api for deep learning. *Information*, 11(2):108.

Cheng-Zhi Anna Huang, Ashish Vaswani, Jakob Uszkoreit, Noam Shazeer, Ian Simon, Curtis Hawthorne, Andrew M Dai, Matthew D Hoffman, Monica Dinculescu, and Douglas Eck. 2018. Music transformer. *arXiv preprint arXiv:1809.04281*.

Harsh Jhamtani, Varun Gangal, Eduard Hovy, Graham Neubig, and Taylor Berg-Kirkpatrick. 2018. Learning to generate move-by-move commentary for chess games from large-scale social forum data. In *Proceedings of the 56th Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pages 1661–1671.

Jared Kaplan, Sam McCandlish, Tom Henighan, Tom B Brown, Benjamin Chess, Rewon Child, Scott Gray, Alec Radford, Jeffrey Wu, and Dario Amodei. 2020. Scaling laws for neural language models. *arXiv preprint arXiv:2001.08361*.

Andrej Karpathy, Justin Johnson, and Li Fei-Fei. 2015. Visualizing and understanding recurrent networks. *arXiv preprint arXiv:1506.02078*.

Conglong Li, Ammar Ahmad Awan, Hanlin Tang, Samyam Rajbhandari, and Yuxiong He. 2021. 1-bit lamb: Communication efficient large-scale large-batch training with lamb's convergence speed. *arXiv preprint arXiv:2104.06069*.

Jiwei Li, Xinlei Chen, Eduard Hovy, and Dan Jurafsky. 2015. Visualizing and understanding neural models in nlp. *arXiv preprint arXiv:1506.01066*.

Kevin Lu, Aditya Grover, Pieter Abbeel, and Igor Mordatch. 2021. Pretrained transformers as universal computation engines. *arXiv preprint arXiv:2103.05247*.

David Noever, Matt Ciolino, and Josh Kalin. 2020. The chess transformer: Mastering play using generative language models. *arXiv preprint arXiv:2008.04057*.

Rodrigo Nogueira, Zhiying Jiang, and Jimmy Li. 2021. Investigating the limitations of the transformers with simple arithmetic tasks.

Niki Parmar, Ashish Vaswani, Jakob Uszkoreit, Lukasz Kaiser, Noam Shazeer, Alexander Ku, and Dustin Tran. 2018. Image transformer. In *International Conference on Machine Learning*, pages 4055–4064. PMLR.

Adam Paszke, Sam Gross, Francisco Massa, Adam Lerer, James Bradbury, Gregory Chanan, Trevor Killeen, Zeming Lin, Natalia Gimelshein, Luca Antiga, et al. 2019. Pytorch: An imperative style, high-performance deep learning library. In *Advances in neural information processing systems*, pages 8026–8037.

Nina Poerner, Benjamin Roth, and Hinrich Schütze. 2018. Interpretable textual neuron representations for nlp. *arXiv preprint arXiv:1809.07291*.

Alec Radford, Karthik Narasimhan, Tim Salimans, and Ilya Sutskever. 2018. Improving language understanding by generative pre-training.

Alec Radford, Jeffrey Wu, Rewon Child, David Luan, Dario Amodei, and Ilya Sutskever. 2019. Language models are unsupervised multitask learners. *OpenAI blog*, 1(8):9.

Samyam Rajbhandari, Jeff Rasley, Olatunji Ruwase, and Yuxiong He. 2020. Zero: Memory optimizations toward training trillion parameter models. In *SC20: International Conference for High Performance Computing, Networking, Storage and Analysis*, pages 1–16. IEEE.

Samyam Rajbhandari, Olatunji Ruwase, Jeff Rasley, Shaden Smith, and Yuxiong He. 2021. Zero-infinity: Breaking the gpu memory wall for extreme scale deep learning. *arXiv preprint arXiv:2104.07857*.

Jeff Rasley, Samyam Rajbhandari, Olatunji Ruwase, and Yuxiong He. 2020. Deepspeed: System optimizations enable training deep learning models with over 100 billion parameters. In *Proceedings of the 26th ACM SIGKDD International Conference on Knowledge Discovery & Data Mining*, pages 3505–3506.

Jie Ren, Samyam Rajbhandari, Reza Yazdani Aminabadi, Olatunji Ruwase, Shuangyan Yang, Minjia Zhang, Dong Li, and Yuxiong He. 2021. Zero-offload: Democratizing billion-scale model training. *arXiv preprint arXiv:2101.06840*.

Julian Schrittwieser, Ioannis Antonoglou, Thomas Hubert, Karen Simonyan, Laurent Sifre, Simon Schmitt, Arthur Guez, Edward Lockhart, Demis Hassabis, Thore Graepel, et al. 2020. Mastering atari, go, chess and shogi by planning with a learned model. *Nature*, 588(7839):604–609.

David Silver, Thomas Hubert, Julian Schrittwieser, Ioannis Antonoglou, Matthew Lai, Arthur Guez, Marc Lanctot, Laurent Sifre, Dharshan Kumaran, Thore Graepel, et al. 2018. A general reinforcement learning algorithm that masters chess, shogi, and go through self-play. *Science*, 362(6419):1140–1144.

David Silver, Julian Schrittwieser, Karen Simonyan, Ioannis Antonoglou, Aja Huang, Arthur Guez, Thomas Hubert, Lucas Baker, Matthew Lai, Adrian Bolton, et al. 2017. Mastering the game of go without human knowledge. *nature*, 550(7676):354–359.

Leslie N Smith. 2018. A disciplined approach to neural network hyper-parameters: Part 1–learning rate, batch size, momentum, and weight decay. *arXiv preprint arXiv:1803.09820*.

Leslie N Smith and Nicholay Topin. 2019. Super-convergence: Very fast training of neural networks using large learning rates. In *Artificial Intelligence and Machine Learning for Multi-Domain Operations Applications*, volume 11006, page 1100612. International Society for Optics and Photonics.

Hanlin Tang, Shaoduo Gan, Ammar Ahmad Awan, Samyam Rajbhandari, Conglong Li, Xiangru Lian, Ji Liu, Ce Zhang, and Yuxiong He. 2021. 1-bit adam: Communication efficient large-scale training with adam's convergence speed. *arXiv preprint arXiv:2102.02888*.

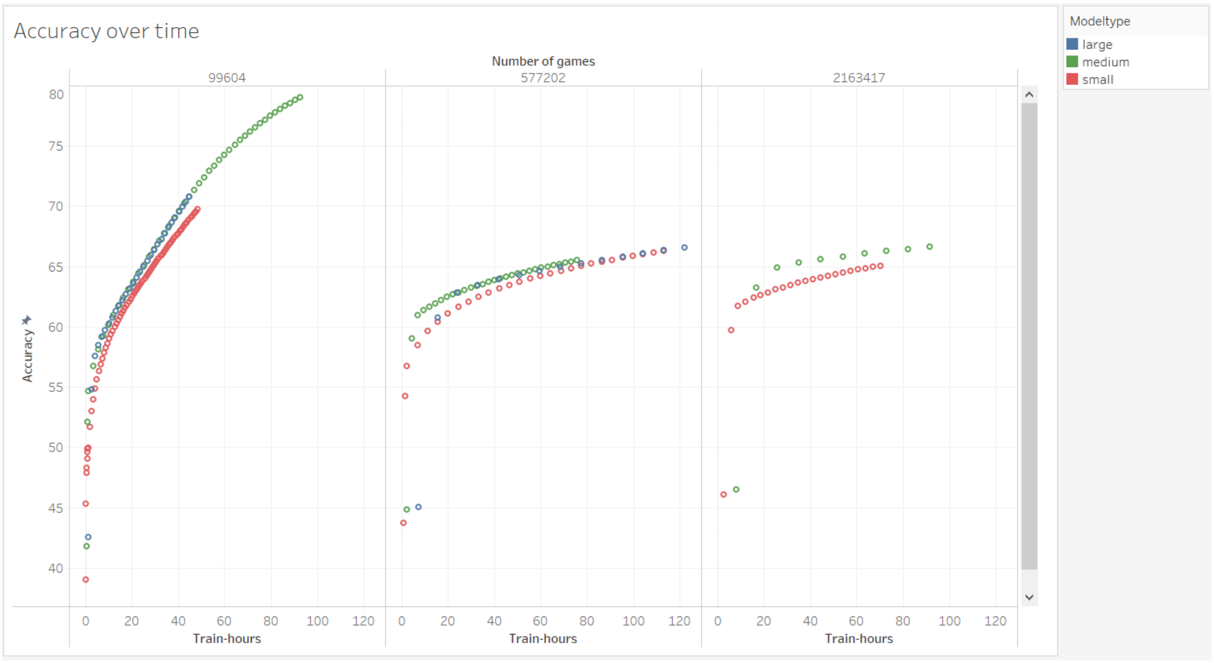
Nenad Tomašev, Ulrich Paquet, Demis Hassabis, and Vladimir Kramnik. 2020. Assessing game balance with alphazero: Exploring alternative rule sets in chess. *arXiv preprint arXiv:2009.04374*.

Ashish Vaswani, Noam Shazeer, Niki Parmar, Jakob Uszkoreit, Llion Jones, Aidan N Gomez, Łukasz Kaiser, and Illia Polosukhin. 2017. Attention is all you need. In *Advances in neural information processing systems*, pages 5998–6008.

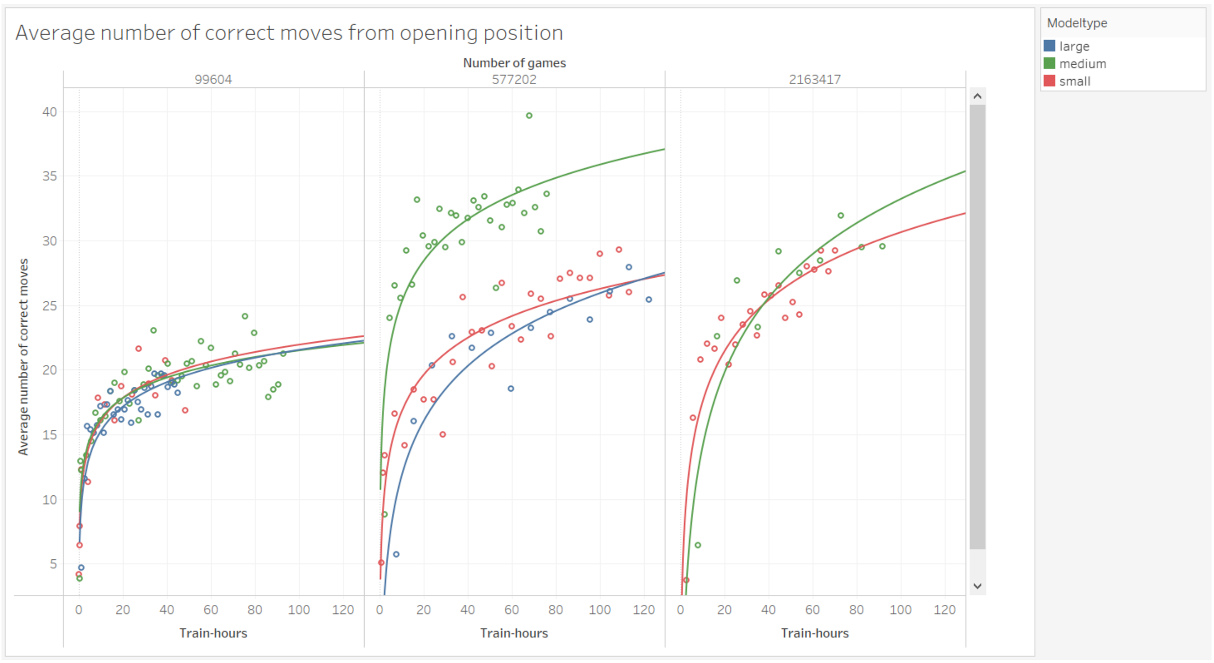
Sean Welleck, Ilia Kulikov, Jaedeok Kim, Richard Yuanzhe Pang, and Kyunghyun Cho. 2020. Consistency of a recurrent language model with respect to incomplete decoding. *arXiv preprint arXiv:2002.02492*.

Thomas Wolf, Julien Chaumond, Lysandre Debut, Victor Sanh, Clement Delangue, Anthony Moi, Pierric Cistac, Morgan Funtowicz, Joe Davison, Sam Shleifer, et al. 2020. Transformers: State-of-the-art natural language processing. In *Proceedings of the 2020 Conference on Empirical Methods in Natural Language Processing: System Demonstrations*, pages 38–45.

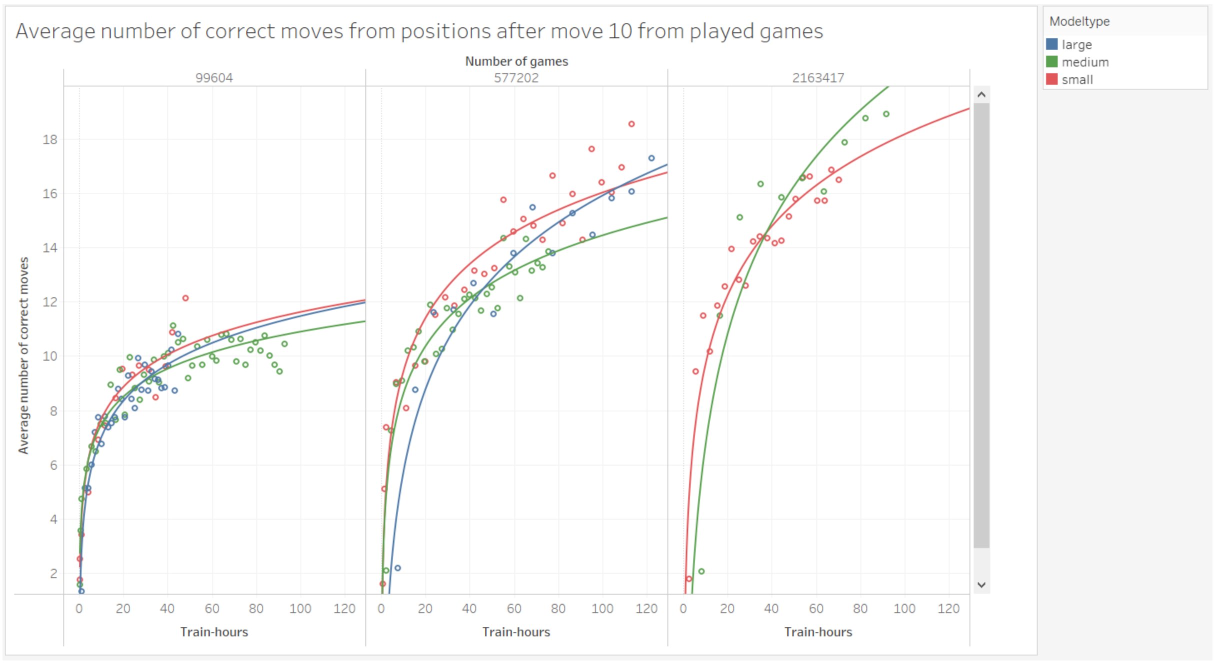
Minjia Zhang and Yuxiong He. 2020. Accelerating training of transformer-based language models with progressive layer dropping. *arXiv preprint arXiv:2010.13369*.



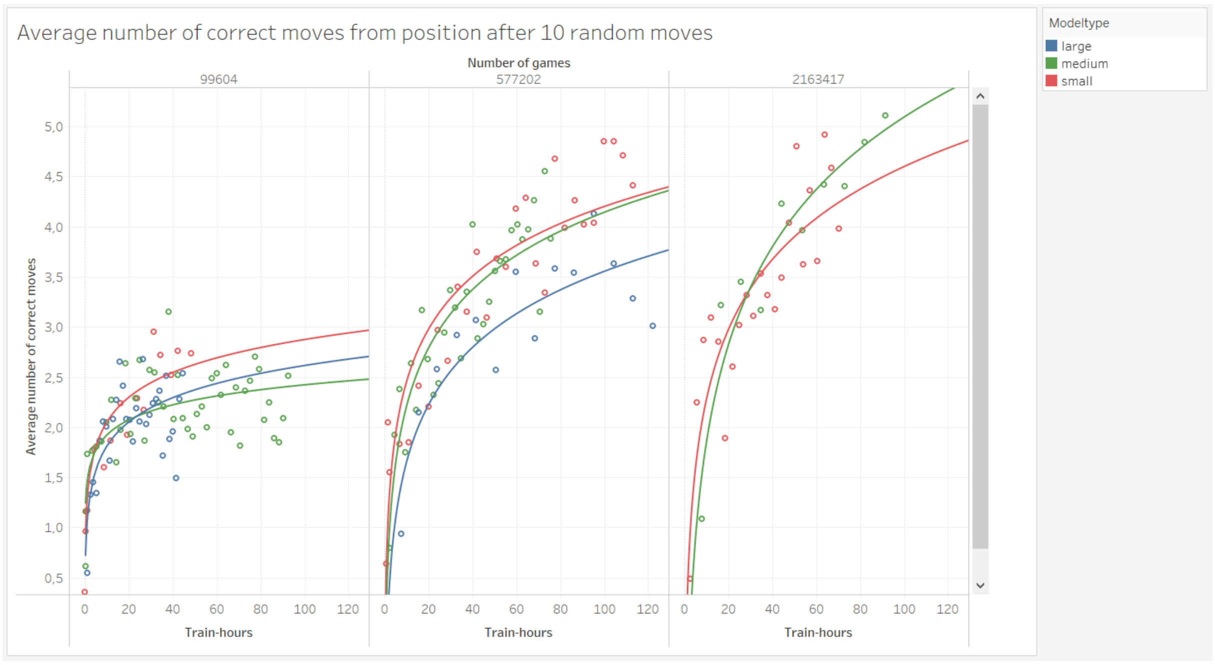
Appendix A-1: Predictive accuracy over trainingstime



Appendix A-2: Average number of correct moves from an opening position



Appendix A-3: Average number of correct moves from positions after move 10 from games played



Appendix A-4: Average number of correct moves from position after ten random moves