# Comparative Analysis of Convolutional and Transformer Architectures in Go Policy Networks

**Antoni Hanke** AH370971@STUDENTS.MIMUW.EDU.PL
**Michał Grotkowski** MG417595@STUDENTS.MIMUW.EDU.PL
*University of Warsaw, Poland*

**Abstract**

In this report we aim to spot shortcomings of using a convolutional architecture as a Go policy network. By comparing it to an equivalently trained Transformer policy and employing XAI methods such as Ceteris Paribus, we can see where each network under and overperforms. This work points in the direction of further research of Transformer architectures in positional games such as Go, where previously it was believed Convolutions were SoTA.

## 1. Introduction

The game of Go has always been one of the benchmarks of Machine Learning. AlphaGo (Silver et al., 2017) employed a convolutional neural network to create a policy of superhuman strength. However, newest advancements in AI show the prowess of transformers. In this report we aim to spot shortcomings of a convolutional Go policy network by employing XAI methods and comparing it to an equivalently trained Transformer policy.

### 1.1 Related Works

Massive success was achieved in AlphaGoZero (Silver et al., 2017), where Google managed to train a superhuman Go engine using with no human knowledge using a convolutional Policy/Value network combined with MCTS. Multiple further enhancements were made in the open source engine KataGo (Wu, 2020), which also uses a Convolutional Policy network and crowd-distributed training.

On the other hand, the transformer architecture shook the world in Attention is All you Need (Vaswani et al., 2017). A Transformer architecture BART (Lewis et al., 2019) was created using multiple encoder and decoder blocks. Research showed prowess of the transformer architecture, especially in tasks such as natural language processing and machine translation.

So far, only a few attempts were made in applying transformers to games such as Chess or Go. Ciolino et al. (2020) and Noever et al. (2020) tried applying NLP transformers to textualized game records to generate new moves. Results of such approach were unimpressive. Czech et al. (2023) tried using a vision transformer architecture to play Chess. Results were slightly below baseline.

## 2. Methodology

### 2.1 Models

We have created two policy networks based on different architectures. We aimed for the comparison to be as fair as possible. Both models have a similar number of parameters and were trained on the same data. The models were trained on the same machine for the same amount of time.

#### 2.1.1 Convolutional Policy

Model 1 is a Convolutional Policy model with 40 residual connection blocks, 256 channels. The architecture is equivalent to the one used in AlphaGoZero, scaled to ∼60M parameters.

#### 2.1.2 Transformer Policy

Model 2 is a BART transformer architecture. The model itself is imported as is from HuggingFace. The exact hyperparameter configuration can be seen in the Appendix. The number of parameters of this model is ∼60M. Tokenization of input and output can be seen in the Appendix.

#### 2.1.3 Datasets and training

We have trained both policies in a supervised manner using behavioral cloning. Our dataset consists of over 500K human master (4dan+) level games gathered from the European Go Server. Both models were trained for 48h on 8xA100 GPUs using the Athena Supercomputer. Both models achieved a level of a human master (3dan), without employing any tree search. The networks are similar in strength, with the Transformer policy achieving a slightly better winrate of 60% against the convolutional network.

#### 2.1.4 Oracle policy

We have downloaded and used a KataGo policy network as an Oracle, helping us in evaluations. It is of superhuman strenght and beats our policy networks confidently in 100% of games played.

### 2.2 Evaluation

#### 2.2.1 Performance

To evaluate the performance of networks we make the following assumptions: (a) The Oracle's best move is the optimal move on the board; (b) If a Policy's best move is the same as the Oracle's best move, it's performance should be equal to 1; (c) Otherwise, it's performance should be proportional to the probability of playing the Oracle's best move divided by its top 1 move.
So finally the following is the formula of performance of a Policy:

$$performacne_p = \frac{p(m_{best})}{\underset{m \in moves}{max}[p(m)]}$$

$p$ – policy, $m_{best}$ – best oracle move, $p(m)$ – probability of policy selecting move m

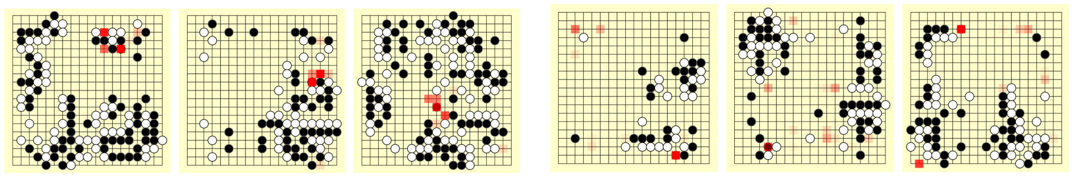

(a) Low dispersity boards (b) High dispersity boards

Figure 1: Boards with different dispersities. Red intensity is Oracle move probability. (b) Shows boards which have multiple independent situations in different regions.

#### 2.2.2 Board Dispersity

Some positions in Go show characteristics of multiple independent regions, where playing in any of them is a good move. Other positions show a single situation which is globally the most important.
We try to quantify with the help of probabilities of our Oracle Policy Network:

$$dispersity(b) = \sum_{m \in moves} dist(m_{best}, m) \cdot p_{oracle}(m)$$

#### 2.2.3 Ceteris Paribus Probability Difference Dispersity

We also want to evaluate if removing a single stone from the board affects the probabilities of squares close-by or far away.

$$probabilities\_dispersity_p(b, m_{removed}) = \frac{\sum_{s \in board\_squares} |p_b(s) - p_{b'}(s)| \cdot dist(s, m_{removed})}{\sum_{s \in board\_squares} |p_b(s) - p_{b'}(s)|}$$

$p$ – policy, $m_{removed}$ – stone removed from board b, $b$ – original board,
$b'$ – board with removed stone, $p_b(m)$ – probability of policy selecting move m on board b

## 3. Experimental results

### 3.1 Policy Performance against Board Dispersity

Tests performed on 1000 random positions show a negative correlation between board dispersity and how much Transformers outperforms Convolution. (Figure 2)
The result could be explained by the shape of Convolution architecture - it specializes in analyzing regions locally, which helps when there are multiple small independent regions. Transformers on the other hand are not limited by kernel sizes and can attend to any place on the board, which helps in single large global situations.

### 3.2 Ceteris Paribus Probability Difference Dispersity

100 random positions show that removing a stone makes the transformer architecture shift probabilities far away from the stone removed (Figure 3a). This is desirable, as the Oracle shifts probabilities very far away (Table 1). This also can be explained by global attention. Further results in Appendix show this global attention mechanism.

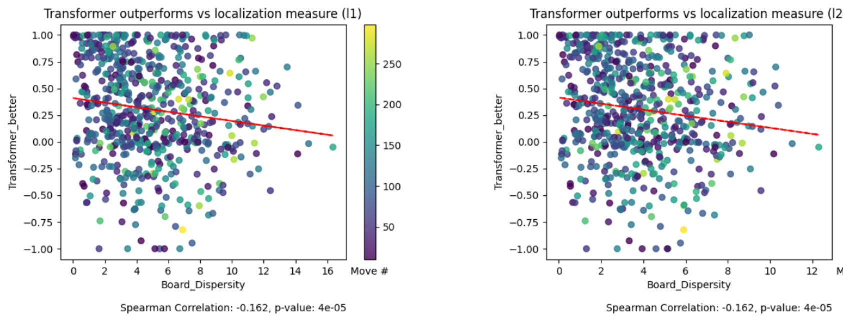


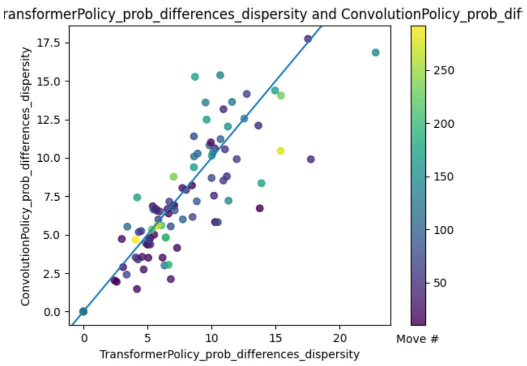
Figure 2: The less disperesed the board is, the more Transformer outperforms Convolution

## 4. Conclusion

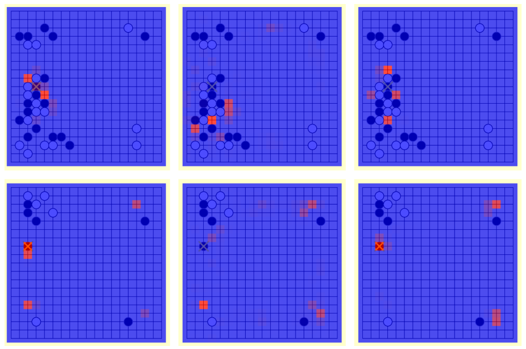
Our experiments show the limitations of convolutional policy networks. They are great at capturing local features, however limited in attending to global phenomena. In some tasks, like the game of Go, this might be detrimental. Transformers on the other hand are able to seamlessly incorporate both the local and global understanding via its flexible attention mechanism. This points in the direction of future work on implementing Transformers as policy networks for large positional games.

| **Convolution** | **Transformer** | **Oracle** |
|---|---|---|
| 7.23 | 7.77 | 8.72 |

Table 1: Average probability change dispersity. Oracle shows very high dispersity.



(a) Probability difference disparity. (x=y) Wilcoxon signed-rank test p-value: 0.039



(b) Probability Differences of respectively Convolution, Transformer, Oracle

Figure 3: More situations have higher probability difference dispersities for Transformers

## References

Matthew Ciolino, David Noever, and Josh Kalin. The go transformer: Natural language modeling for game play, 2020.

Johannes Czech, Jannis Blüml, and Kristian Kersting. Representation matters: The game of chess poses a challenge to vision transformers, 2023.

Mike Lewis, Yinhan Liu, Naman Goyal, Marjan Ghazvininejad, Abdelrahman Mohamed, Omer Levy, Ves Stoyanov, and Luke Zettlemoyer. Bart: Denoising sequence-to-sequence pre-training for natural language generation, translation, and comprehension. 2019.

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

D. Silver, J. Schrittwieser, and K. Simonyan. Mastering the game of go without human knowledge. *Nature*, 550:354–359, 2017.

Ashish Vaswani, Noam Shazeer, Niki Parmar, Jakob Uszkoreit, Llion Jones, Aidan N. Gomez, Lukasz Kaiser, and Illia Polosukhin. Attention is all you need, 2017.

David J. Wu. Accelerating self-play learning in go, 2020.

## Appendix A. Transformer implementation details

### A.1 Hyperparameters

The BART Transformer model is taken straight as is from the Huggingface portal: `https://huggingface.co/docs/transformers/model_doc/bart`.

Below is the hyperparameter configuration used for training:

```
"BartConfig.vocab_size": 512,
"BartConfig.max_position_embeddings": 512,
"BartConfig.encoder_layers": 8,
"BartConfig.decoder_layers": 8,
"BartConfig.encoder_attention_heads": 8,
"BartConfig.decoder_attention_heads": 8,
"BartConfig.decoder_ffn_dim": 2048,
"BartConfig.encoder_ffn_dim": 2048,
"BartConfig.d_model": 512,
"BartConfig.dropout": 0.1,
```

### A.2 Tokenization details

The input board was encoded as a sentence of tokens $(t_0, t_1, ...t_{361})$.

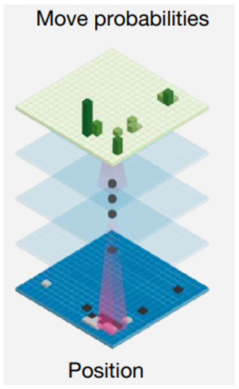
$$t_0 = \begin{cases} 0 & \text{if it is White's turn} \\ 1 & \text{if it is Blacks's turn} \end{cases}$$

$$t_{x+19*y+1} = \begin{cases} 2 & \text{if there is a White stone on square (x, y)} \\ 3 & \text{if there is a Black stone on square (x, y)} \\ 4 & \text{if there is no stone on square (x, y)} \\ 5 & \text{if there is a Ko on square (x, y)} \end{cases}$$

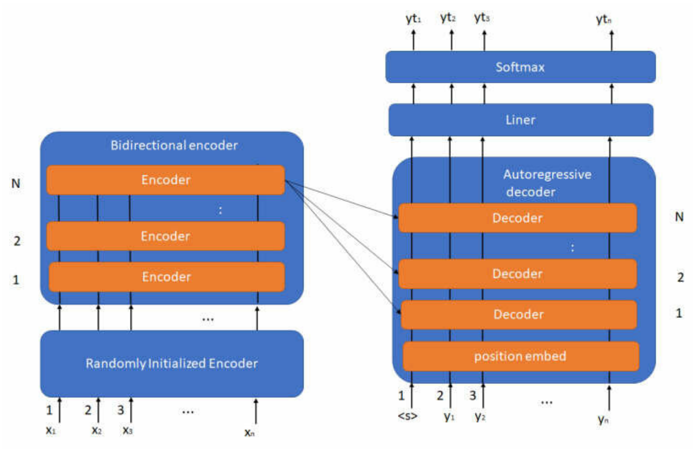
The output is encoded in a single token being one of $(6, 7, ..., 367)$ - which indicates the square where the next move was played.
The transformer always recieves as an input the 362 long token sequence describing the board and always tries to predict the 363th token.

## Appendix B. Additional results

Sample boards were analyzed using Ceteris Paribus to see the logit differences of the last layer (Figure 5, 6). The results exhibit an interesting phenomena: Convolution shows a smooth gradual change in logits near the removed stone, whereas transformer attends to single points potentially far away from it.
Further research could be performed to understand this more.

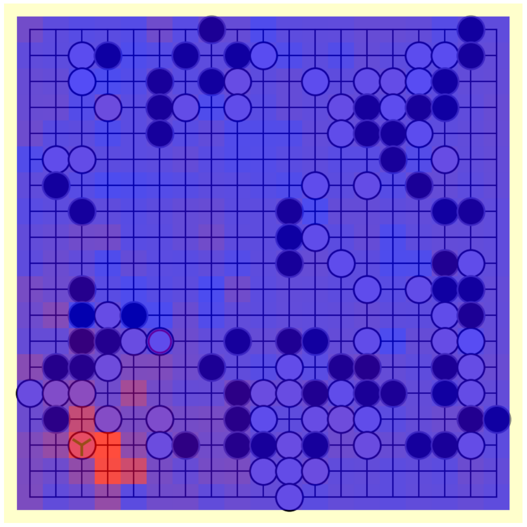


(a) AlphaGo policy

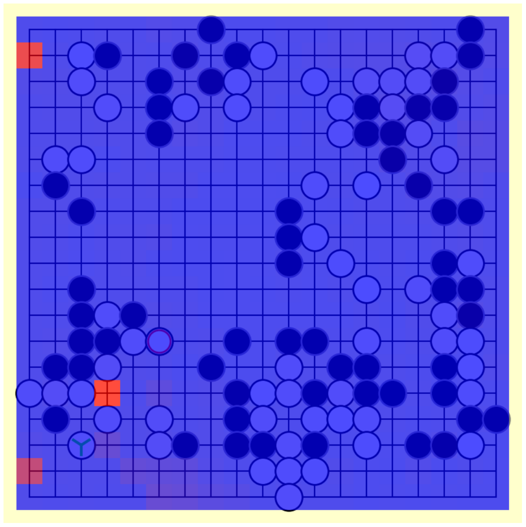


(b) BART transformer architecture

Figure 4: Convolution and transformer architectures

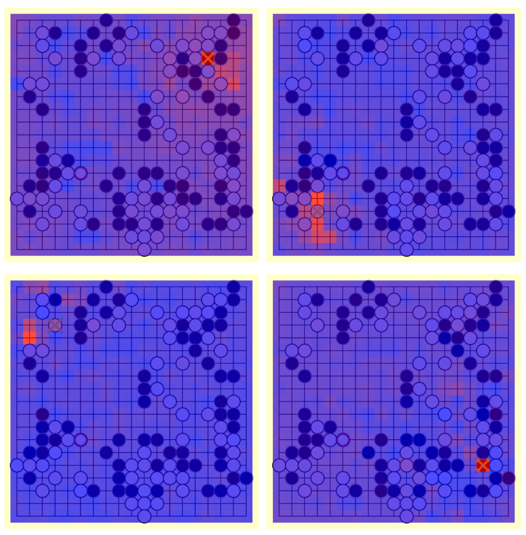


(a) Convolution logits change

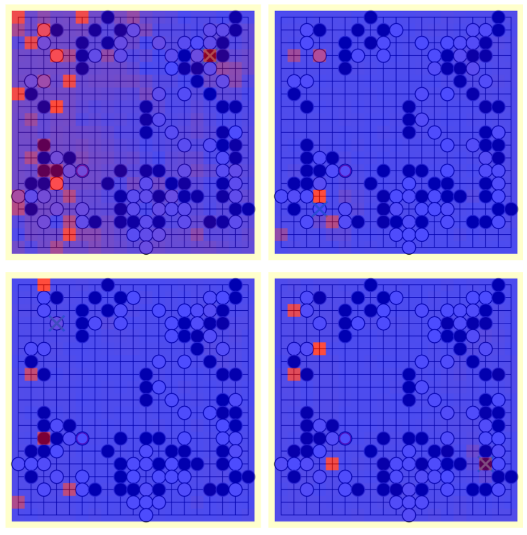


(b) Transformer logits change

Figure 5: Change in the logits of the last layer after removing the market stone. Convolution shows a smooth gradual change, whereas transformer attends to single points



(a) Convolution logits change



(b) Transformer logits change

Figure 6: More examples of last layer logit change after removing marked stone