Learning to play Go

Vsevolod Karpov & Nicolas B. Carbone

The game of Go

- Go is a board game. Very Popular in Asia
- Played on 19x19 (most popular) board.
- The objective of the game:
    - Capture most territory
- After 100 moves:
    - ca. 1*10^(249) possible positions
    - Chess: ca. 1.6*10^(160)
- A heuristic approach will not work. Too many possible moves!

<u>Main Goal:</u>

- We want to train a network to master a 9x9 board.

Previous work - AlphaGo Zero

- Improved version of AlphaGo
- NN that beats best human players
- Pure reinforcement learning is better than learning from human games (AlphaGo), to achieve superhuman levels
- Uses Monte-Carlo-Tree-Search to explore the best predicted moves
- Also has a raw network which does not use any form of tree search - performs fairly well

AlphaGo vs Lee Sedol in Seoul march 2016

Reinforcement Learning -Our approach

1) Start N networks with random weights.
2) Let each network play 2 matches against every other network (one as black and one as white).
3) Choose best N/2 networks based on win ratio (reward function).
4) Delete worst N/2. Clone best N/2.
5) Adjust weights of all networks by a random number in range (-1,1)*learning_rate
6) Repeat from 2)

- Current board and previous 6 boards are used as input
- No tree search for future moves.

Network architecture : Stage 1

- Conv2d (features=32,stride=1,kernel=3,padding=1) , ReLU , BatchNorm
- Conv2d (features=64,stride=1,kernel=3,padding=1) , ReLU , BatchNorm
- Conv2d (features=128,stride=1,kernel=3,padding=1) ,ReLU
- MaxPool(kernel=2,stride=2,padding=0) , BatchNorm
- Conv2d (features=256,stride=1,kernel=3,padding=1) ,RuLU
- MaxPool(kernel=2,stride=2,padding=0) , BatchNorm
- Linear(82)
- Linear(82)
- Linear(82)
- Linear(82)

Network : Stage 1

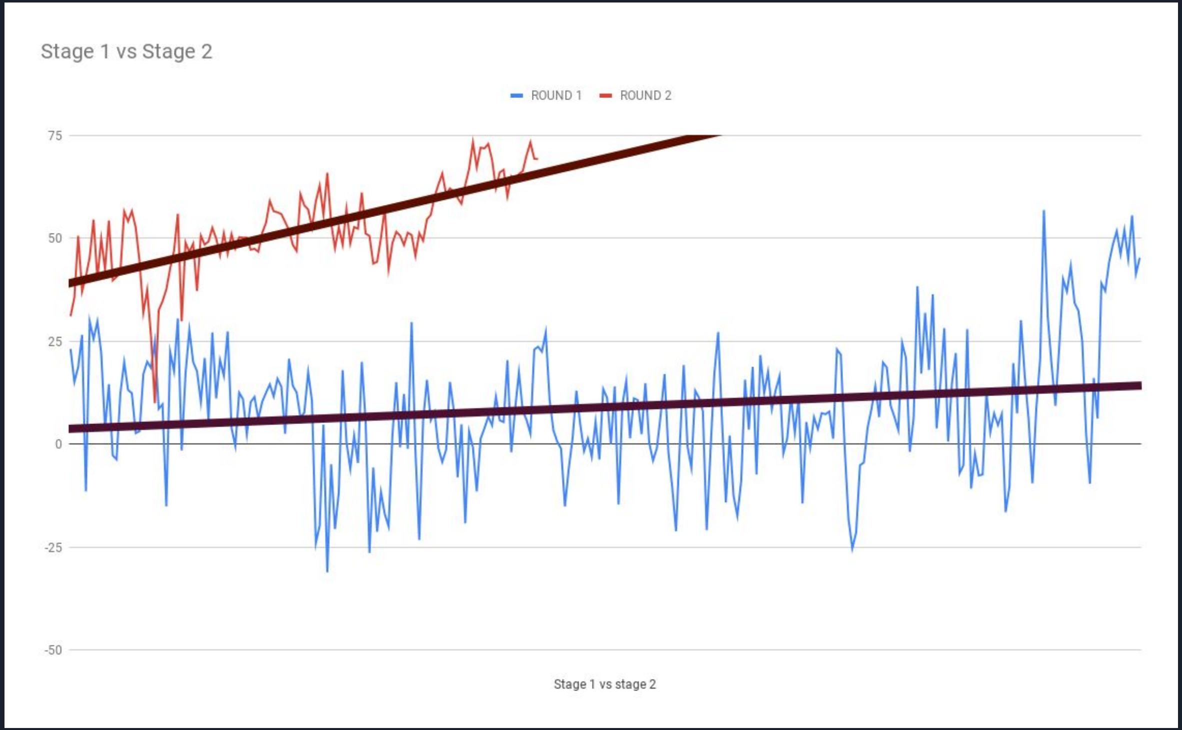
- Stage 1 was meant to get a feeling of training time and progress
- Learning rate : 0.005, N = 10
- Trained for 500 epochs.
    - Achieved 100% win ratio against random play
- Used for validation for stage 2.

Network architecture : Stage 2
(Similar to AlphaGo)

- Sequential layer x7
  - Conv2d (features=64,stride=1,kernel=3,padding=1) , ReLU , BatchNorm, x2
  - Passthrough
- Conv2d (features=128,stride=1,kernel=3,padding=1) , ReLU , BatchNorm
- Conv2d (features=256,stride=1,kernel=3,padding=1) ,ReLU
- MaxPool(kernel=2,stride=2,padding=0) , BatchNorm
- Conv2d (features=512,stride=1,kernel=3,padding=1) ,RuLU
- MaxPool(kernel=2,stride=2,padding=0) , BatchNorm
- Linear(656)
- Linear(328)
- Linear(164)
- Linear(82)
- Softmax

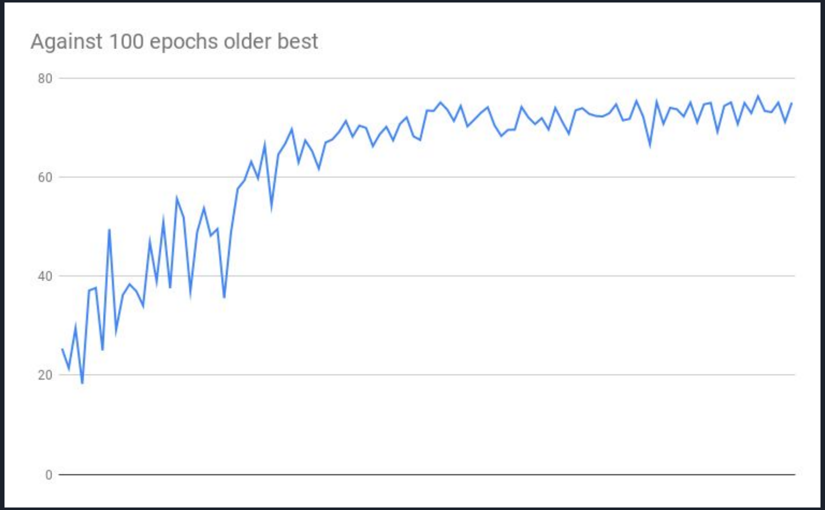
Stage 1 vs Stage 2 validation

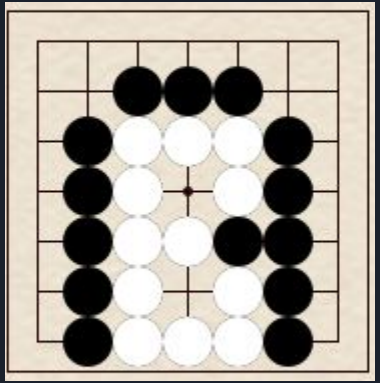
- Blue :
  - learning_rate = 0.0005
  - N = 10
- Red :
  - Learing_rate = 0.0001
  - N = 20
- Run for the same amount of physical time (less epochs for red)



Results

- Network play style after 2000 epochs:
  - Connects stones (good)
  - Tries to capture (good)
  - Tries to fill the whole board (questionable)
  - Does not form Two Eyes (very very bad)
- Steady improvement each 100 epochs





- Example of white having two eyes. Black cannot capture the white group

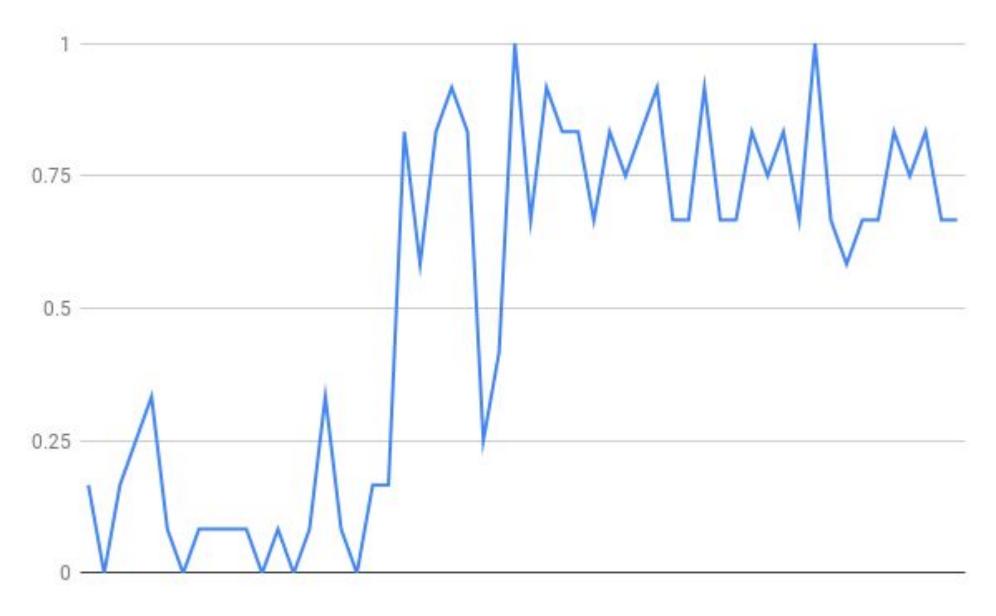
Discussion

- Clearly, the bigger N, the better. Significant increase in performance with bigger N?
    - Better preservation of good mutations
- Lower learning rate prevents divergence
- Play style could become proper over time.
- *“If you learn from idiots, you will remain an idiot”*
- Maybe better play style could be “forced” by changing reward function?

Discussion

- We tried to modify reward function.
- Reward faster play
  - Should result in less “board filling”
  - Resulted in networks giving up (passing) too early
- Reward bigger score
  - Resulted in more “infinite games”

- In an act of desperation, we tried to train from start.
- N = 6, learning_rate = 0.0001
- Reward function rewards fast play.
- <u>Top player is selected and cloned 5 times.</u>
- Result shows win rate against previously trained Stage 2 net over 60 epochs.



Playing example
Circle - Old, Cross - New

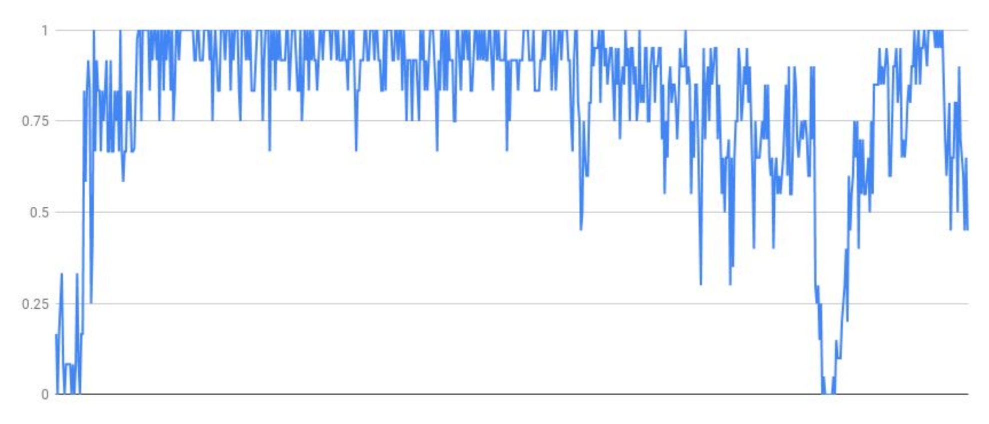
```
- o o o x x - x -
o o o x x x x x o
o o o o x x x x o
o o o o x x x - x
o o o o o x o o x
- o o o o o x x x
o o o o x x x x x
o o x x x x x x x
o x x - x x x x x
o is thinking [...]
o selects:  [0, 5]
```

```
- o o o x x - x -
o o o x x x x x o
o o o o x x x x o
o o o o x x x - x
o o o o o x o o x
o o o o o o x x x
o o o o x x x x x
o o x x x x x x x
o x x - x x x x x
x is thinking [...]
x selects:  [0, 0]
```

```
x - - - x x - x -
- - - x x x x x o
- - - - x x x x o
- - - - x x x - x
- - - - - x o o x
- - - - - - x x x
- - - - x x x x x
- - x x x x x x x
- x x - x x x x x
```

Discussion

- Correction, better play style CAN be forced, but reward function must be such from start.
- It’s doable to train with small N but you have to “get lucky”.
- Bigger N =>
  - More minima are explored and evaluated simultaneously.
  - Bigger probability of getting to global minima.
  - Better chance of preserving good mutations



Conclusion/Summary

More training needed

Bigger N (100-1000) should be used ?

-> Needs stronger hardware - a single GPU is not enough

Need deeper network?

Q&A