GOat : A Neural Network for playing the game of Go

Mohamed Rahmouni, Emilie Greff

January 2022

1 Introduction

in this report, we present the approach taken to model GOat, our neural network that play the game of Go. We will also previous approaches we have dealt with

2 A First Approach : ResNet

This approach has been motivated by both the script provided by the professor but also the paper Residual Networks for Computer Go published in 2018.

For this first approach, we used the script provided at the beginning of the project as baseline. We first trained directly this model to assess the political category accuracy and the value mse we start from. We noticed that by training this model on 50 epochs, it achieves an accuracy of 0,27 and an mse of 0,13.

From this, we started making some small improvements based in particular on the papers present on the website. The first script was composed of six loops of residual layers where each loop was made of two convolutions before ReLu activation and BatchNormalization. We first tried to add more loops, with more convolution layers of different combination. Indeed, in the paper *Residual Networks for Computer Go* for example, it is shown that The residual input layer for computer Go composed from parallelized convolutions of different shapes give better results.

We also tried to incorporate some little aspects of Deep Learning seen in class such as Dropout and regularization. We also were inspired by the paper *Spatial Average Pooling for Computer Go* to improve the architecture of the value network using Spatial Average Pooling.

However, despite all this work, the political category accuracy wouldn't go beyond 0,33, which is low. But most importantly, after reading *Mobile Networks for Computer Go*, we gave up the residual architecture to start working on the Mobile Net one

3 Mobile Net

This paper from 2021 proposes to evaluate the interest of Mobile Networks for the game of Go using supervised learning as well as the use of a policy head and a value head different from the AlphaZero heads. The paper shows that the mobile net architecture achieves better results than both residual network and Alphazero one.

For this part, we've been mainly inspired by the code in the annexe of the paper. We used the model proposed on the paper and trained it for 100 epochs and noticed we could already achieve a political accuracy of 0,38, without any improvements and using only the dataset with 100k data. We've concluded that this would be the perfect architecture to start from.

We then started by trying different learning rates and optimizers on small range of epochs to notice a potential trend of the optimal parameters. However, the results showed us that using a stochastic gradient with a learning of 0.005 already achieves great results, with better calculation times, in comparison to other combinations.

3.1 Model architecture

As explicited before, our architecture was mainly based on the principle of MobileNet. It consists in having blocks as in residual networks where the input of a block is added to its output. We set the number of filters to 200, the number of blocks to 33 and the trunk to 64.

3.2 Training the model

To train the model, it was a little bit tedious since we used Colab and so we had to stay connected while training. We made sure to save the model at every epoch to avoid losing it because of an unintentional stop from Colab. Also, when importing the dataset with 1M data, the code blocked after twenty or so epochs of training because of memory problems. We used the package gc that have the attribute collect that free the memory from useless elements at each predefined number of epochs (we set it to 3 epochs). We trained our model with batch sizes of 64 and N = 10000. To boost performances, we also divided the learning by two for some epochs but also the batch size at the end of 500 epochs. In the end, we've trained the model for about 700 epochs. We couldn't print a curve of policy accuracy and value mse development over epochs because of the colab stops. We provided in the report the code for trianing our model, which explicits well our strategy.

```
1
2  loss = []
3  policy_loss = []
4  value_loss = []
5  policy_categorical_accuracy = []
6  value_mse = []
7  ep = []
8
9  for i in range(1, epochs + 1):
10     ep.append(i)
11
12 print(model.optimizer.lr)
13
14 for i in range(1, epochs + 1):
15
16     if i%3 == 0 :
17       gc.collect()
18
19     if i in [500, 550, 600, 650]:
20         model.optimizer.lr = model.optimizer.lr / 2
21     if i > 500 and i % 50 == 0:
22         model.optimizer.lr = model.optimizer.lr / 1.2
23     if i >= 500:
24         batch = 32
25
26     print('epoch␣' + str(i))
27     golois.getBatch(input_data, policy,
28                     value, end,
29                     groups, i*N)
30     history = model.fit(input_data,
31                         {'policy': policy,
32                         'value': value},
33                         epochs=1,
34                         batch_size=batch)
35     model.save('RAHMOUNI_GREFF_GO.h5')
36     if (i % 10 == 0):
37         golois.getValidation(input_data, policy,
38                              value, end)
39         val = model.evaluate(input_data,
40                              [policy, value],
41                              verbose=0,
42                              batch_size=batch)
43         print("val␣=", val)
```

Extract from the code for the training phase

4 Conclusion

By implementing the Mobile Net Architecture with some small personalized touches, notably in the way of training the model, we were able to achieve an accuracy of 0,49 on the policy accuracy and a value mse of 0,051 for the validation set. This led us to have a decent ranking and winrate in the last tournament we've participated to.

With a little more time, we would have tried to implement new improvements on optimization process, the activation function and the convolution layers, based on the recent *Cosine Annealing, Mixnet and Swish Activation for Computer Go* paper.