Université Paris-Dauphine

Mathématiques et Informatique de la Décision et des Organisations

Deep Learning Project : ComputerGo

Authors :
MAYARD Hippolyte

Advisor :
CAZENAVE Tristan

M2 Intelligence Artificielle Systèmes Données

Mars 2020

Table des matières

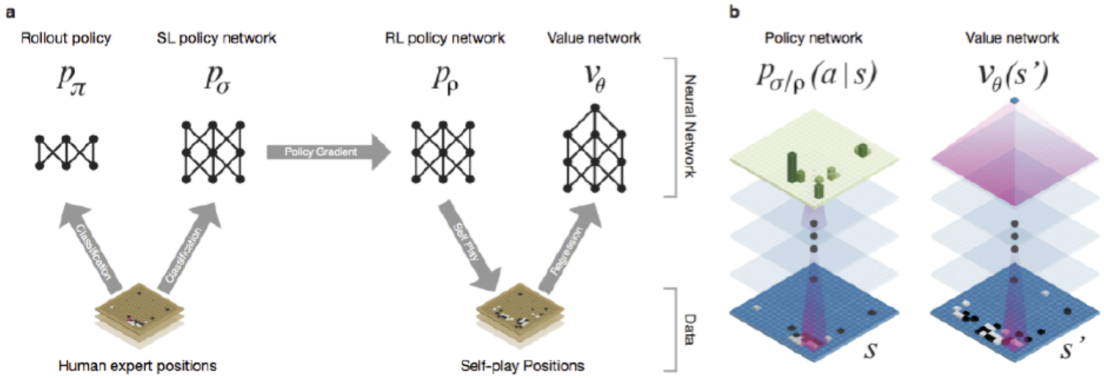
1 Introduction

In this report, we propose a Deep Learning architecture for Computer Go with a constraint of less than 1 million parameters. Based on several papers (see the References section), we have optimized the training of our network.
In order to make our training the more efficient, we used the Google Colaboratory environment which provides the access to a GPU.

Regarding the data, the games used to generate the training data are self play games of Facebook ELF opengo Go program. We have been using the 500 000 such provided games to train our model. The input data is hence composed of 8 19x19 planes containing the following informations; color to play, ladders, current state on two planes, and the two previous states on four planes. The output targets are the policy -a vector of size 361 with 1.0 for the move played, 0.0 for the other moves-, and the value -1.0 if White won, 0.0 if Black won- and the state at the end of the game, that takes two planes.

As explained in [1], solving the Computer Go problem implies dealing with Reinforcement Learning and Monte Carlo Tree Search. In this matter, this paper was very useful in understanding Alpha Go architecture as well as the founding principles of such networks. The following figure describes architecture of the Alpha Go as explained in [1].

FIGURE 1: Neural Network training pipeline and architecture



2 Goliath features

2.1 A residual network and Spatial Batch Normalization

As we have seen in the previous section, we used the Alpha Go architecture [1] to build our network in a two headed network which trains both policy and value. In this section, we will explain the choices that led in constructing Goliath. Moreover, Darkforest [2] shows good results in predicting the next 3 moves. Both of the architectures of Alpha Go [1] and Darkforest [2] are based on a convolutional layer followed by a dense layer with a ReLU activation function. In order to improve the training, residual layers have been experimented with convolutional and ReLU layers, based on [3].

Golois [4] is based on residual layers as well as Spatial Batch Normalization. Using Spatial Batch Normalization produced good results and made the training more accurate for Goliath. We based Goliath's architecture on [3] and [4], it is given below in figure 2 and figure 3. First, the input layer is residual. We parallelized a 5x5 convolutional layer followed by a ReLU layer to a 1x1 convolutional layer followed too by a ReLU layer.
The residual layers are described in the figure 3.
The difference in Goliath architecture lies in the fact that we build a full residual network. As shown in figure 4, we connected the output of the residual layer to the value and the policy layers. This addition gave Goliath better performances.
The entire architecture of Goliath is given in the Appendix.

FIGURE 2: The residual input layer for Goliath

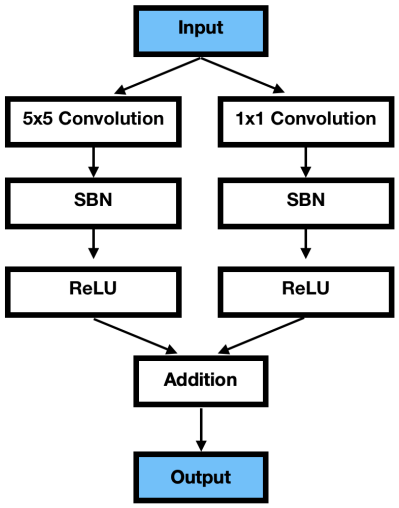
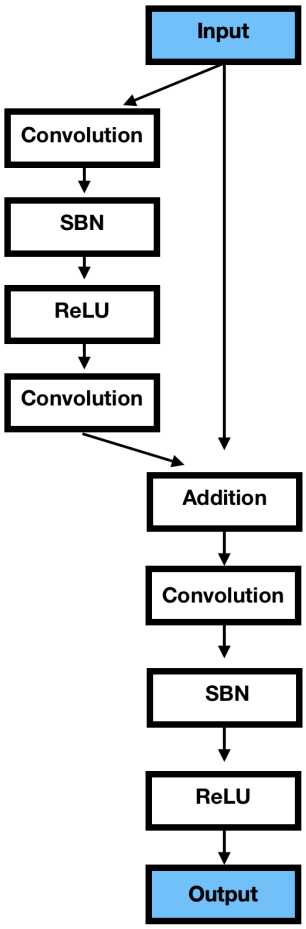


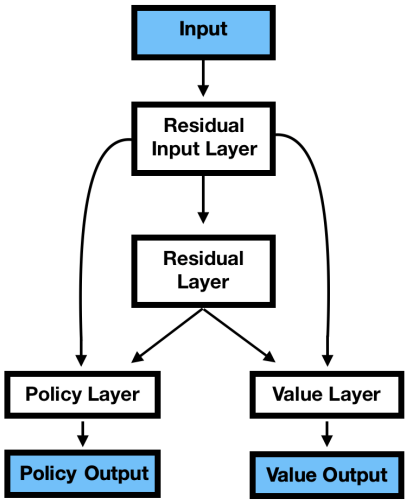
FIGURE 3: The residual layer for Goliath



2.2 Average pooling

In an attempt to decrease the number of parameters of our model, we used [5] which compares a spatial average pooling residual network with a standard one. The one using spatial average pooling appears to train faster than a standard residual network with a comparable final loss.
In this way, we used a spatial average pooling for the value and the policy network. As expected, the spatial average pooling decreased the number of parameters of our model as well as the training time.

FIGURE 4: Goliath architecture

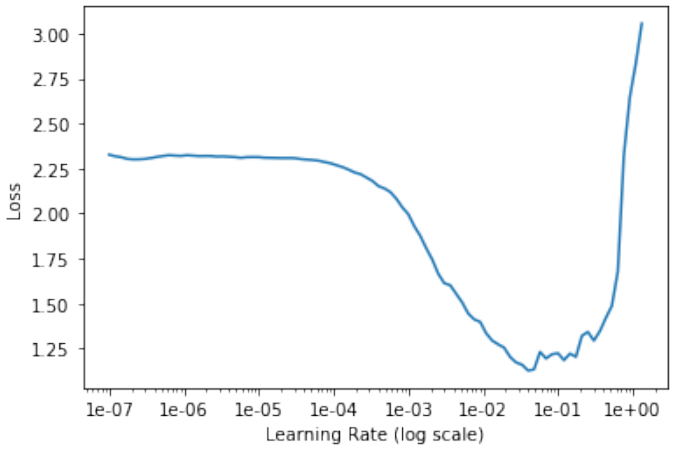


2.3 An adaptative learning rate

During the training of our model, we figured out that after some time, the network was not learning anymore. In this case, we have been seeking for solutions regarding this issue. The first solution was to make the network deeper by increasing the number of residual layers and decreasing the number of channels. The other solution was to decrease the learning rate when the accuracy starts not increasing anymore. Hence, we decided to use an adaptative learning rate. Through the Callback function in Tensorflow, it is possible to adjust the learning rate. Therefore, we have used an open source code on Github that computes the optimal learning rate for an epoch [6].

Figure 5 below represents the evolution of the loss according to the learning rate for a given epoch. The optimized learning rate is the minimizer of the loss for this epoch. In this way, before each epoch we optimized the learning rate according to the loss for the given batch.

FIGURE 5: Adaptative learning rate



3 Network training

In order to train the network we build minibatches of size 128 composed of 128 states chosen randomly in the training set. The games used to generate the training data are self play games of Facebook ELF opengo Go program. We have been using the 500 000 such provided games to train our model.

Figure 6 gives the evolution of the accuracy for the training and the test sets of Goliath.
Figure 7 gives the evolution of the loss for the training and the test sets. Goliath scores 42.65% accuracy. We can see that the optimization of the learning rate plays an important role in the training, as the network seems to stop learning from the epoch 230 and starts learning again after finding the optimal learning rate.
Figure 8 below give the evolution of the learning rate according to the number of epochs. We can see that from epoch 230, when the network stopped learning from the data, we activated the learning rate optimization, which increased the accuracy of the model and reduced the loss.

FIGURE 6: Evolution of the training and test accuracy

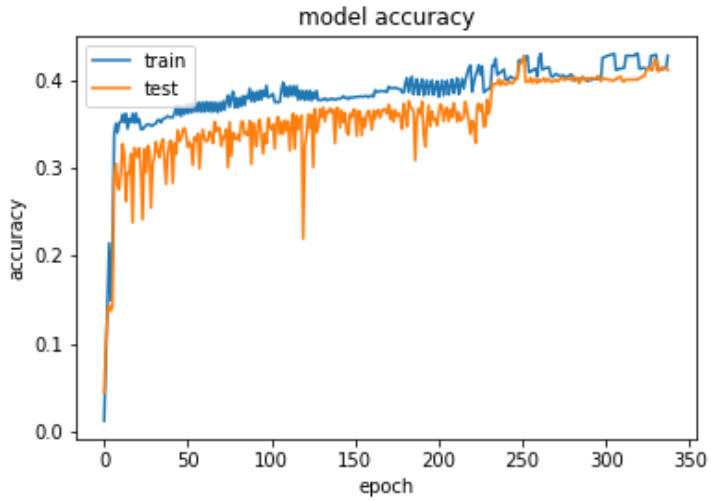


FIGURE 7: Evolution of the training and test loss

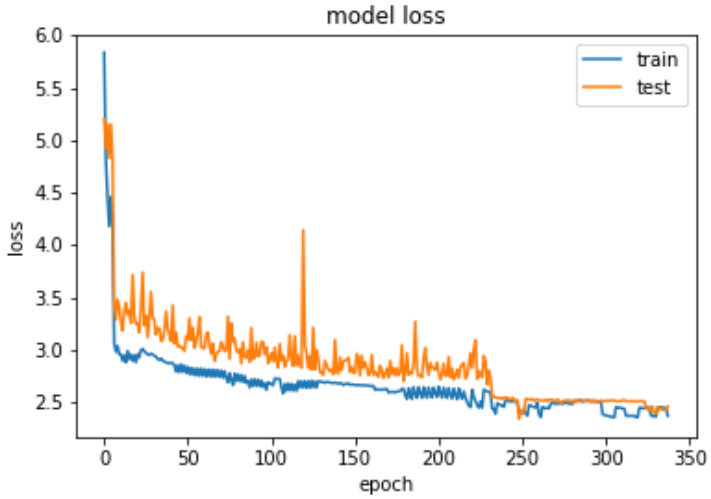
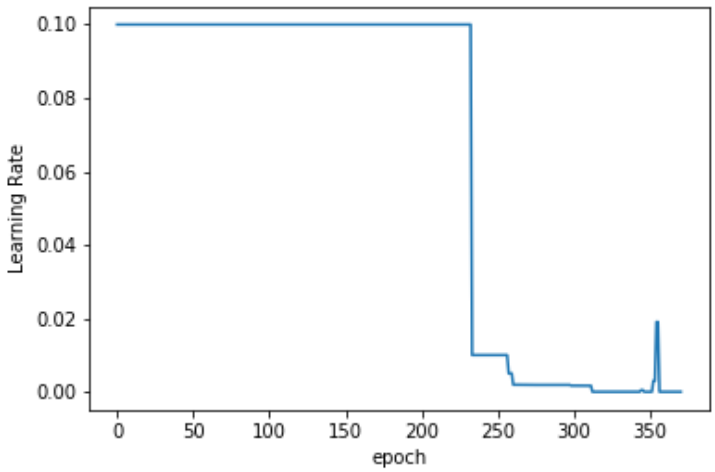


FIGURE 8: Evolution of the learning rate against the number of epochs



4 Conclusion

Through several articles on how to improve the performances of a neural network in Computer Go, we built a model based on residual layers which is called Goliath. By training this model and optimizing the learning rate according to the epochs, our model succeeded in scoring 42.56% on Facebook ELF opengo self play data.

5 Références

[1] D. Silver, A. Huang, C. J. Maddison, A. Guez, L. Sifre, G. van den Driessche, J. Schrittwieser, I. Antonoglou, V. Panneershelvam, M. Lanctot, S. Dieleman, D. Grewe, J. Nham, N. Kalchbrenner, I. Sutskever, T. Lillicrap, M. Leach, K. Kavukcuoglu, T. Graepel, and D. Hassabis, “Mastering the game of go with deep neural networks and tree search”

[2] Yuandong Tian and Yan Zhu. Better computer go player with neural network and long-term prediction. arXiv preprint arXiv :1511.06410, 2015.

[3] T. Cazeanave, "Residual Network for Computer Go"

[4] T. Cazeanave, "Improved Policy Network for Computer Go"

[5] T. Cazeanave, "Spatial average pooling for computer go", CGW at IJCAI 2018

[6] Github https ://github.com/avanwyk/tensorflow-projects/tree/master/lr-finder

6 Appendix

FIGURE 9: Neural Network training pipeline and architecture