Dauphine | PSL
UNIVERSITÉ PARIS

Projet d'Apprentissage Profond :

Réaliser un réseau de neurone complexe spécifique au jeu de GO

Olivier RANDAVEL
Maxime TALARMAIN

March 28, 2020

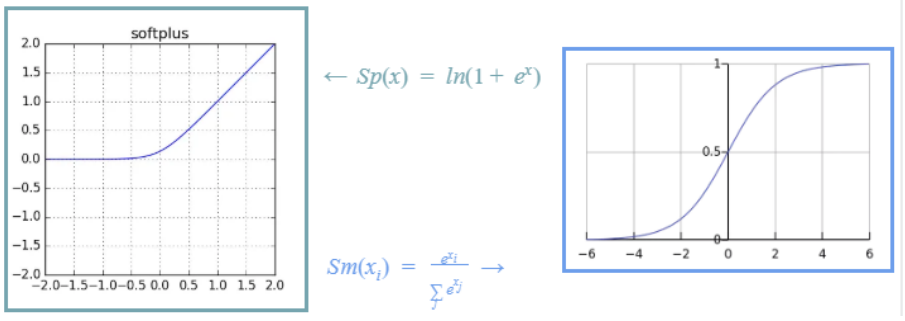
Ce rapport se découpe en trois parties. La première est consacrée à la description de l'architecture de notre modèle. La deuxième partie montre nos différentes expérimentations pour améliorer le modèle. La dernière se concentre sur l'implémentation du `fit_generator`. Cet ordre décrit le cheminement de notre projet de façon chronologique. Dans ce projet, pour évaluer nos modèles nous avons tenu compte des métriques suivantes `loss`, `valid_loss`, `valid_policy_acc` et `valid_value_acc`.

Tout d'abord notre première expérimentation s'est faite à l'aide du réseau décrit dans ce document : [3]. Puis, nous avons construit un réseau à l'image du resnet [1]. L'architecture finale de notre modèle est présentée en annexe de ce document. Elle est composée de 10 couches convolutives successives permettant d'obtenir deux outputs correspondant à la value et la policy. Ce modèle a été réalisé à l'aide de 700 000 parties et a été obtenu après 9 epochs. Le choix de 700 000 parties s'est imposé par la limite de 25 GO de RAM disponible sur Colab. L'évaluation s'est faite grâce à un échantillon d'apprentissage de 89%, un échantillon de validation de 10% et un échantillon test de 1%. Ce premier modèle nous a permis d'obtenir les valeurs suivantes.

`MODEL_MT_OR_V3`	623700/623700 [==============================] - 89s 143us/sample - loss: 3.0098 - `policy_loss`: 2.7870 - `value_loss`: 0.2228 - `policy_acc`: 0.3400 - `value_acc`: 0.6256 - `val_loss`: 3.2498 - `val_policy_loss`: 3.0200 - `val_value_loss`: 0.2300 - `val_policy_acc`: 0.3063 - `val_value_acc`: 0.5863

Secondairement, nous avons testé plusieurs architectures différentes et essayé d'améliorer au mieux notre modèle. Nous avons d'abord exploré d'autres types de couches, puis nous avons testé différentes fonctions d'activation. D'après la littérature [6] sur les réseaux profonds, il est recommandé d'utiliser un LeakyRelu suivi d'un BatchNormalization. Cela permet de renormaliser les données suites à plusieurs couches convolutives. Néanmoins, l'ajout de ces couches au réseau de neurone n'a pas été concluant puisque cela n'a pas contribué à améliorer les performances.Nous avons également fait l'hypothèse que des couches de Pooling tel que MaxPooling ou AveragePooling permettraient une meilleure performance du modèle. Cela a plutôt eu l'effet inverse puisque l'accuracy est moins bonne lorsque l'on ajoute ces couches au modèle. Différents types de ResNet ont été testés, la localisation des couches additives a été challengée empiriquement. Nous avons observé que deux couches additives avaient de meilleures performances que dans le cas d'une ou de trois. Enfin, nous avons optimisé la précision à l'aide du kernel en choisissant une taille de 4.

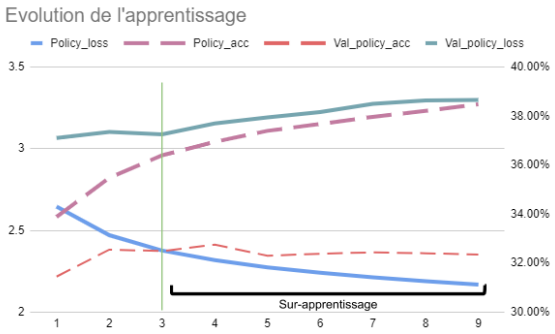
Nous avons également essayé d'améliorer le modèle en utilisant d'autres fonctions d'activation. Pour se faire nous nous sommes inspirés de ces sites : [2] et [5]. Notre attention se portant sur la précision fournie par la politique, nous avons testé plusieurs fonctions d'activation semblables à la fonction softmax. Les fonctions softplus et softmax :



La fonction softplus ne retourne pas de probabilité, c'est à dire de valeur entre 0 et 1, contrairement à la fonction softmax. Dans le cas de l'évaluation d'une politique, dont l'objectif est de classifier la meilleure politique parmi 361, il est plus judicieux de faire appel à une fonction non bornée supérieurement. En effet, dans le cadre d'un réseau de neurones profond, le vanishing gradient détériore l'apprentissage. Pour diminuer cet effet, une fonction d'activation tel que softplus a de bonnes propriétés puisqu'elle est non bornée supérieurement.

`MODEL_MT_OR_V4`	623700/623700 [==============================] - 89s 143us/sample - loss: 2.9036 - `policy_loss`: 2.6816 - `value_loss`: 0.2220 - `policy_acc`: 0.3352 - `value_acc`: 0.6272 - `val_loss`: 3.2945 - `val_policy_loss`: 3.0608 - `val_value_loss`: 0.2336 - `val_policy_acc`: 0.3181 - `val_value_acc`: 0.6056

De plus, nous avons expérimenté différents optimizers que nous retrouvons dans la documentation de keras. Cependant, quelque soit l'optimizer choisi, on converge au même point à une différence de temps de calcul près. Un petit learning rate converge beaucoup trop lentement (inférieur à 0.01). En conclusion, nous avons garder l'optimizer le plus rapide, dans notre cas : $keras.optimizers.SGD(lr = 0.1)$. Enfin, nous avons limité le sur-apprentissage que nous avions observé, à l'aide de l'utilisation du dropout ou de regularizer. Nos résulats se sont avérés peu concluant donc nous avons exclu cette optimization de notre modèle final.



Dans cette dernière partie, nous avons implémenté le `fit_generator` à l'aide de la documentation fournie par keras : [4]. Cette fonction a été exécutée pour améliorer le modèle `MODEL_MT_OR_V4`. Cette fonction nécessite de créer une boucle qui va générer un certain nombre de batchs du jeu de données *games.data* grâce à l'appel de la fonction *getBatch* qui compile ces batchs au fur et à mesure. Pour cela on a dû initialement, générer des matrices aléatoires pour `input_data`, policy, value et end. L'utilisation de "yield" permet de générer au fur et à mesure ces batchs. On a choisi $N = 10000$ avec un `step_per_epochs` = 10 puisque le notebook ne supportait pas une taille plus importante. Après 600 epochs et l'utilisation d'un optimizer dont le learning rate s'incrémente de un tous les 100 epochs, nous avons obtenu ces résulats sur notre échantillon de validation constitué de 99 000 parties :

`MODEL_MT_OR_V7`	99000/99000 [==============================] - 19s 196us/sample - loss: 3.0411 - `val_policy_loss`: 2.8179 - `val_value_loss`: 0.2232 - `val_policy_acc`: 0.3443 - `val_value_acc`: 0.6302

References

[1] Tristan Cazenave. *Residual Networks for Computer Go*. URL: https://www.lamsade.dauphine.fr/~cazenave/papers/resnet.pdf. (accessed: 16.03.2020).

[2] Andrei Ciuparu. *Soft++, a multi-parametric non-saturating non-linearity that improves convergence in deep neural architectures*. URL: https://www.sciencedirect.com/science/article/pii/S0925231219317163. (accessed: 16.03.2020).

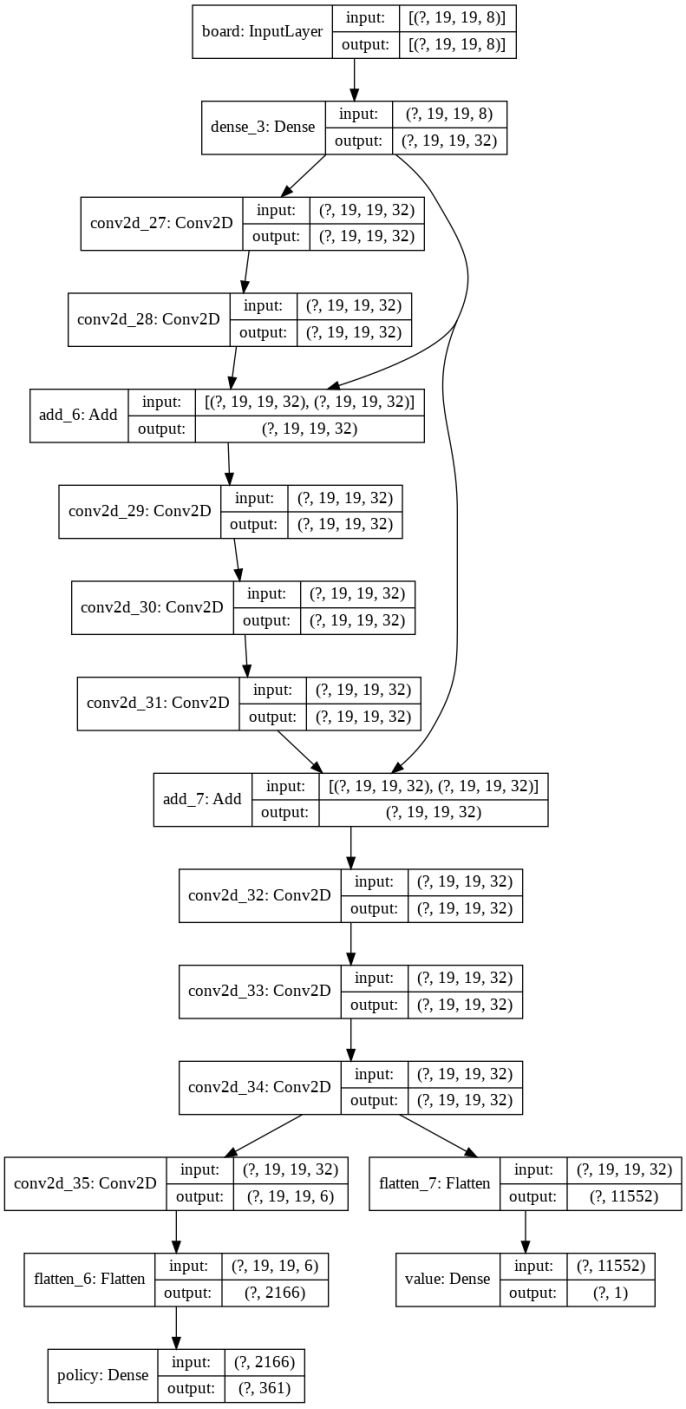
[3] David Foster. *How to build your own AlphaZero AI using Python and Keras*. URL: https://medium.com/applied-data-science/how-to-build-your-own-alphazero-ai-using-%20python-and-keras-7f664945c188. (accessed: 16.03.2020).

[4] Naruu et Gabriel Demarmiesse. *The Sequential model API*. URL: https://keras.io/models/sequential/. (accessed: 16.03.2020).

[5] Naruu et Gabriel Demarmiesse. *Usage of activations*. URL: https://keras.io/activations/. (accessed: 16.03.2020).

[6] Dmytro Mishkin. *Batch normalization*. URL: https://github.com/ducha-aiki/caffenet-benchmark/blob/master/batchnorm.md. (accessed: 16.03.2020).

Annexe


board: InputLayer
input: [(?, 19, 19, 8)]
output: [(?, 19, 19, 8)]
dense_3: Dense
input: (?, 19, 19, 8)
output: (?, 19, 19, 32)
conv2d_27: Conv2D
input: (?, 19, 19, 32)
output: (?, 19, 19, 32)
conv2d_28: Conv2D
input: (?, 19, 19, 32)
output: (?, 19, 19, 32)
add_6: Add
input: [(?, 19, 19, 32), (?, 19, 19, 32)]
output: (?, 19, 19, 32)
conv2d_29: Conv2D
input: (?, 19, 19, 32)
output: (?, 19, 19, 32)
conv2d_30: Conv2D
input: (?, 19, 19, 32)
output: (?, 19, 19, 32)
conv2d_31: Conv2D
input: (?, 19, 19, 32)
output: (?, 19, 19, 32)
add_7: Add
input: [(?, 19, 19, 32), (?, 19, 19, 32)]
output: (?, 19, 19, 32)
conv2d_32: Conv2D
input: (?, 19, 19, 32)
output: (?, 19, 19, 32)
conv2d_33: Conv2D
input: (?, 19, 19, 32)
output: (?, 19, 19, 32)
conv2d_34: Conv2D
input: (?, 19, 19, 32)
output: (?, 19, 19, 32)
conv2d_35: Conv2D
input: (?, 19, 19, 32)
output: (?, 19, 19, 6)
flatten_7: Flatten
input: (?, 19, 19, 32)
output: (?, 11552)
flatten_6: Flatten
input: (?, 19, 19, 6)
output: (?, 2166)
value: Dense
input: (?, 11552)
output: (?, 1)
policy: Dense
input: (?, 2166)
output: (?, 361)

Pour réaliser ce projet, nous avons utilisé le notebook Colab. Colab met à disposition un GPU, ainsi que 25 GO RAM et 60 GO de disque dure.

```
# import librairies
!pip install pybind11
#!pip3 install --upgrade tensorflow
import pybind11
from numpy.random import seed
seed(1)
print(pybind11.get_include())
!pip install torch
import tensorflow as tf
import torch
from tensorflow.keras.preprocessing.image import img_to_array
from tensorflow.keras.models import load_model
import datetime
import tensorflow.keras as keras
from tensorflow.keras import layers
from keras.models import Sequential
from keras.layers.core import Dense, Activation
from keras.optimizers import SGD,Adam,Adamax,nadam,RMSprop
import numpy as np
import matplotlib.pyplot as plt
from keras.layers import Dense, Dropout, Flatten, Conv2D, MaxPool2D,
    GlobalAveragePooling2D
from keras import regularizers
from keras.callbacks import ModelCheckpoint
!pip show tensorflow
from google.colab import drive
drive.mount('/content/drive', force_remount=True)
%cd "/content/drive/My Drive/Dauphine/Semestre 1/Deep_learning_CT/DeepLearningProject/"
!ls
!bash compile.sh
```

```
# create test sample
def create_test_train_db(db,size) :
  shape_ = db.shape[0]
  train = db[:int(shape_*size)]
  test = db[int(shape_*size):]
  return train,test

def load_data(input_data = np.load ('/content/drive/My Drive/Dauphine/Semestre 1/
    Deep_learning_CT/DeepLearningProject/input_data.npy'),
              policy = np.load('/content/drive/My Drive/Dauphine/Semestre 1/
    Deep_learning_CT/DeepLearningProject/policy.npy'),
              value = np.load('/content/drive/My Drive/Dauphine/Semestre 1/
    Deep_learning_CT/DeepLearningProject/value.npy'),
              end = np.load('/content/drive/My Drive/Dauphine/Semestre 1/Deep_learning_CT
    /DeepLearningProject/end.npy')) :
  input_data_train, input_data_test = create_test_train_db(input_data, 0.99)
  policy_train, policy_test = create_test_train_db(policy, 0.99)
  value_train, value_test = create_test_train_db(value, 0.99)
  end_train, end_test = create_test_train_db(end, 0.99)
  return input_data_train, policy_train, value_train, input_data_test, policy_test,
    value_test
```

```
17
18  def dataset_(N =700000) :
19    import golois
20    planes = 8
21    moves = 361
22    dynamicBatch = True
23    if dynamicBatch:
24      import sys
25      sys.path.append('/content/drive/My Drive/Dauphine/Semestre 1/Deep_learning_CT/
        DeepLearningProject/golois.cpp')
26      #import golois.cpp
27      input_data = np.random.randint(2, size=(N, 19, 19, planes))
28      input_data = input_data.astype ('float32')
29
30      policy = np.random.randint(moves, size=(N,))
31      policy = keras.utils.to_categorical (policy)
32
33      value = np.random.randint(2, size=(N,))
34      value = value.astype ('float32')
35
36      end = np.random.randint(2, size=(N, 19, 19, 2))
37      end = end.astype ('float32')
38      golois.getBatch(input_data, policy, value,end)
39      input_data_train, policy_train, value_train, input_data_test, policy_test, value_test
          = load_data(input_data, policy, value,end)
40      return input_data_train, policy_train, value_train, input_data_test, policy_test,
        value_test
41  input_data_train, policy_train, value_train, input_data_test, policy_test, value_test =
      dataset_(N =700000)
```

```
1   #create generator fonction
2   def generatorbis(N=100000):
3     import golois
4     import sys
5     planes = 8
6     moves = 361
7     epochs = 30
8     sys.path.append('/content/drive/My Drive/Dauphine/Semestre 1/Deep_learning_CT/
        DeepLearningProject/golois.cpp')
9     input_data = np.random.randint(2, size=(N, 19, 19, planes))
10    input_data = input_data.astype('float32')
11
12    policy = np.random.randint(moves, size=(N,))
13    policy = keras.utils.to_categorical (policy)
14
15    value = np.random.randint(2, size=(N,))
16    value = value.astype('float32')
17
18    end = np.random.randint(2, size=(N, 19, 19, 2))
19    end = end.astype('float32')
20    count=0
21    while count<(98000000//N):
22      golois.getBatch (input_data, policy, value, end)
23      yield input_data,{'policy': policy, 'value': value}
24      count+=1
```

```
#implementation of the model
def model_2(input, optimizer = keras.optimizers.SGD(lr=0.1)) :
  x = layers.Conv2D(32, 4, activation='relu', padding='same')(input)
  ident = x
  x = layers.Conv2D(32, 4, activation='relu', padding='same')(x)
  x = layers.Conv2D(32, 4, activation='relu', padding='same')(x)
  x = layers.add([ident,x])
  ident = x
  x = layers.Conv2D(32, 4, activation='relu', padding='same')(x)
  x = layers.Conv2D(32, 4, activation='relu', padding='same')(x)
  x = layers.Conv2D(32, 4, activation='relu', padding='same')(x)
  x = layers.add([ident,x])

  x = layers.Conv2D(32, 4, activation='relu', padding='same')(x)
  x = layers.Conv2D(32, 4, activation='relu', padding='same')(x)
  x = layers.Conv2D(32, 4, activation='relu', padding='same')(x)

  policy_head = layers.Conv2D(6, 4, activation='relu', padding='same')(x)
  policy_head = layers.Flatten()(policy_head)
  policy_head = layers.Dense(361, activation='softplus', name='policy')(policy_head)

  value_head = layers.Flatten()(x)
  value_head = layers.Dense(1, activation='sigmoid', name='value')(value_head)
  return policy_head,value_head, optimizer
```

```
#implement model_MT_OR_v4
input = keras.Input(shape=(19, 19, 8), name='board')
policy_head, value_head,optimizer = model_2(input, optimizer = keras.optimizers.SGD(lr
    =0.1))
model = keras.Model(inputs=input, outputs=[policy_head, value_head])
model.compile(optimizer=optimizer,
              loss={'value': 'mse', 'policy': 'categorical_crossentropy'},
              metrics=['accuracy'])
history = model.fit(input_data_train, {'policy': policy_train, 'value': value_train},
          epochs=14, batch_size=123, validation_split=0.1)
```

```
#load and evaluate a saved model
model = load_model('model_MT_OR_v4.h5')
for i in range(1,2) :
  print(datetime.datetime.now())
  input = keras.Input(shape=(19, 19, 8), name='board')
  if i > 1 :
    save_name = 'weights-improvement_fffitgenerator_' + str(i) +'.h5'
    model.save(save_name)
    model = load_model(save_name)
  policy_head, value_head,optimizer = model_2(input, optimizer = keras.optimizers.SGD(lr
    =0.1/i))
  model.compile(optimizer=optimizer,
              loss={'value': 'mse', 'policy': 'categorical_crossentropy'},
              metrics=['accuracy'])
```

```
    model.summary()
    input_data,policy,value = load_data()[:3]
    history = model.fit_generator(generator=generatorbis(N=10000),
                    validation_data=(input_data,(policy,value)),
                    use_multiprocessing=None,
                    steps_per_epoch = 2,
                    epochs= 100, verbose=0)
```