Mastering the game of Omok

6.S198 Deep Learning Practicum

Name: Jisoo Min¹

Instructors: Professor Hal Abelson², Natalie Lao³
   TA Mentor: Martin Schneider⁴
   Industry Mentor: Stan Bileschi⁵

¹ jisoomin@mit.edu
² hal@mit.edu
³ natalie@mit.edu
⁴ martins@mit.edu
⁵ bileschi@google.com
Table of Contents

Introduction 3
  Goal 3
  Background 3
  Game Overview 3

Implementation 4
  Overview 4
  Data Collection 4
  Data Processing 5
    LIB format dataset 5
    RIF format dataset 5
    Board Representation 5
    Move Representation 5
  Model 6
    Architecture 6
    Input and Output 7
  Experiment 8
  Computing Resources 8

Evaluation 9
  Outcomes 9
  Command Line Testing 9
  Further Extension 9

Acknowledgements 10
Introduction

Goal
The goal of this project is to train human-like neural networks for the game of Omok, predict moves made by professional players, and build an AI bot for the game.

Background
There has been an ongoing interest in developing artificial intelligence that can win professional players on classical board games. One of the biggest feats in the artificial intelligence field is AlphaGo (2016) that achieved 99.8% winning rate on the game of Go, or one of the most complex classical board games, against other programs and defeated world professional human players. AlphaGo uses a new search algorithm that combines Monte Carlo simulation with value and policy networks.

The game of Omok has simpler game strategies and rules compared to the game of Go. Regardless of the simple rules, smart moves and strategies are needed to win the game. To understand the moves and the move patterns, there has been deep learning research targeted to predict the next moves made by professional players.

To explore the game of Omok further, this project replicates the convolutional neural network approach taken by researchers at Chinese Academy of Sciences. By reducing the game of Omok to an image classification problem, we will learn the winning patterns for the game.

Game Overview
On a 15 by 15 board, two players alternate turns and place a stone of their color on each turn. The first player to place five consecutive stones horizontally, vertically, or diagonally wins. There are some additional move restrictions on the black stone player who plays first in the game. The restrictions are “three and three,” “four and four,” and “six stones.” These restrictions ban a move that simultaneously forms two open rows of three stones, two rows of fours stones, or an unbroken chain of six stones, respectively.

illegal moves of J, F, G, and Y

Implementation\(^9\)

Overview

![Diagram of the flow in processing data and fitting the model]

Data Collection

This project was supported by [GomokuWorld.com](https://play.google.com/store/apps/details?id=com.monomob.omok&hl=ko) and [Renju.net](https://github.com/jisoomin/6.S198). Approximately a hundred thousand game datasets through GomokuWorld.com and approximately a fifty thousand game datasets through Renju.net\(^{10}\) were obtained.

---

\(^7\) K is not a banned position because it does not simultaneously form two open rows of three stones


Data Processing

LIB format dataset
Files obtained from GomokuWorld.com were in the form of .lib files. Each of the .lib files stored the analyses and games. RenLib\(^{11}\) was used to convert these .lib files to .txt files that only contained one game per file with all the moves for the given game listed. Then custom Python scripts\(^{12}\) were produced to parse the files and construct board states.

RIF format dataset
One large dataset obtained from Renju.net contained all games in a single .rif file. Custom Python scripts \(^{13}\)were produced to parse the file and construct board states.

Board Representation
Each board state is represented as a \(15 \times 15 \times 3\) array. Three possible options for each intersection of the \(15 \times 15\) board were represented as \([1,0,0]\) for black, \([0,1,0]\) for white, and \([0,0,1]\) for empty.

![Sample representation of a board state with all positions empty](image)

Move Representation
Each move is represented as a one-hot vector of size 225. The position of the next move is marked as one, and rest of the positions as zeros.

![Sample representation of a move](image)

---

\(^{11}\) [http://www.renju.se/renlib/](http://www.renju.se/renlib/)

\(^{12}\) [https://github.com/jisoomin/6_S198/blob/master/proj/src/](https://github.com/jisoomin/6_S198/blob/master/proj/src/)

\(^{13}\) [https://github.com/jisoomin/6_S198/blob/master/proj/src/](https://github.com/jisoomin/6_S198/blob/master/proj/src/)
Model

Architecture

A deep convolutional neural network was trained to predict the next moves based on the board state. It is important to note that the learning rate for the optimizer had to be reduced down to 0.00001 to avoid diverging effects during the training.

```python
x = keras.layers.ZeroPadding2D(padding=3, name="padding_conv1")\(inputs\)
x = keras.layers.Conv2D(64, \(7, 7\), strides=(2, 2), use_bias=False, name="conv1")\(x\)
x = keras_resnet.layers.BatchNorm(axis=axis, epsilon=1e-5, freeze_freeze_bn, name="bn_conv1")\(x\)
x = keras.layers.Activation("relu", name="conv1_relu")\(x\)
x = keras.layers.MaxPooling2D(\(3, 3\), strides=(2, 2), padding="same", name="pool1")\(x\)
```

architecture used to train the model\(^{14}\)

For the optimizer\(^{15}\), RMSprop is used, because we would like to divide the gradient by a running average of its recent magnitude. Alternatively, rprop can be used when the purpose is to only use the sign of the gradient, but it does not work well with mini-batches.

Then for the loss function, *categorical_crossentropy* is chosen because we are interested in getting the probabilities for each option.

\[
H(p, q) = - \sum_{x \forall} p(x) \log(q(x))
\]

\(x\): discrete variable, \(q(x)\): estimate for the true distribution \(p(x)\)

\(^{14}\) [https://github.com/broadinstitute/keras-resnet/blob/master/keras_resnet/models/_2d.py](https://github.com/broadinstitute/keras-resnet/blob/master/keras_resnet/models/_2d.py)

def categorical_crossentropy(output, target, from_logits=False):
    """Categorical crossentropy between an output tensor and a target tensor.
    Arguments:
    output: A tensor resulting from a softmax
    (unless `from_logits` is True, in which case `output` is expected to be the logits).
    target: A tensor of the same shape as `output`.
    from_logits: Boolean, whether `output` is the result of a softmax, or is a tensor of logits.
    Returns:
    Output tensor.
    """
    # Note: nn.softmax_cross_entropy_with_logits
    # expects logits, Keras expects probabilities.
    if not from_logits:
        # scale preds so that the class probs of each sample sum to 1
        output /= math_ops.reduce_sum(
            output, reduction_indices=len(output.get_shape()) - 1, keepdims=True)
        # manual computation of crossentropy
        epsilon = _to_tensor(EPSILON, output.dtype.base_dtype)
        output = clip_ops.clip_by_value(output, epsilon, 1. - epsilon)
        return -math_ops.reduce_sum(
            target * math_ops.log(output),
            reduction_indices=len(output.get_shape()) - 1)
    else:
        return nn.softmax_cross_entropy_with_logits(labels=target, logits=output)

After the model was compiled with the above parameters, it was trained on the preprocessed datasets.

```
model.fit(board_images, next_moves, epochs=epochs,
            batch_size=batch_size,
            validation_split=validation_split,
            shuffle=True,
            callbacks=[checkpointer])
```

model trained\(^{17}\) using the `model.fit()` API on Keras

Input and Output

The model was trained with a pair of `(board_state, next_move)`. Once the model was trained, it was tested with an input of a board state and an output of a size 225 vector with probabilities for each move positions. It is important to note that the board stones had to be flipped to take into account the player’s turn on a given board state.

\(^{16}\) https://github.com/tensorflow/tensorflow/blob/r1.1/tensorflow/contrib/keras/python/keras/backend.py

\(^{17}\) epochs=30, batch_size=256, validation_split=0.15 were used as default values
Experiment
Below is a line-by-line instructions on how to train a given dataset.

~$ pip3 install --user virtualenv
~$ mkdir ~/virtualenv
~$ cd ~/virtualenv
~/virtualenv$ python3 -m venv omok
~/virtualenv$ cd ~
~$ cd ~/6S198/proj/src
~/6S198/proj/src$ source ~/virtualenv/omok/bin/activate
~/6S198/proj/src (omok)$ pip3 install -r requirements.txt
~/6S198/proj/src (omok)$ python test_model.py path/to/dataset [--epochs EPOCHS] [--batch_size BATCH_SIZE] [--split SPLIT] [--lr LR]

Note: path/to/dataset is either a directory of TXT files or a single RIF file. Checkpointer hdf5 weight files will be saved under output/[name_of_dataset]/.

Computing Resources
Initial test were done on the local personal machine that had 2.7 GHz Intel Core i7, 16 GB 2133 MHz LPDDR3. Then larger experiments were tested on Google Cloud (6 vCPUs, 32 GB memory). Lastly, for faster training, MIT Engaging Cluster (234 64GB, 2 x 8-core 2.0GHz CPUs, 90 K20m GPU, 16 Xeon phi, base OS - RHEL/Centos 6.4) was used.
Evaluation

Outcomes

value of cost functions decreased over epochs

Command Line Testing

To simply view an interactive game board testing, please see slide 29 of the final presentation. Below is a line-by-line instructions on how to test a trained model.

~$ pip3 install --user virtualenv
~$ mkdir ~/virtualenv
~$ cd ~/virtualenv
~/virtualenv$ python3 -m venv omok
~/virtualenv$ cd ~
~$ cd ~/6S198/proj/src
~/6S198/proj/src$ source ~/virtualenv/omok/bin/activate
~/6S198/proj/src (omok)$ pip3 install -r requirements.txt
~/6S198/proj/src (omok)$ python test_model.py path/to/hdf5/file

Note: Checkpointer hdf5 weight files will be located under output/[name_of_dataset]/ after successfully training the model as described in the experiment section. You will be prompted to play the game on the console by entering game positions such as “h8” for every iteration.

Future Improvements

Below are some areas for future improvements.

1. Finer Implementation
   ○ give different weights to players
   ○ consider board rotations
   - train model on all 8 million datasets
   - train model on different parameters
   - test more architectures and evaluate performance

Further Extension
Online gaming will be available in a few weeks on http://web.mit.edu/jisoomin/www/6S198/proj/.

Acknowledgements
I would like to thank my TA mentor Martin Schneider and my industry mentor Stan Bileschi for providing me both technical and high-level guidance in the project. I would also like to thank the project manager Jessy Lin for organizing all logistical details throughout the project period, and the course instructors Professor Hal Abelson and Natalie Lao for giving me the opportunity to learn about deep learning. Finally, I would like to thank all the course staff, MIT Engaging Cluster staff\(^{18}\), GomokuWorld.com, and all other researchers across the world\(^{19}\) who provided me with resources throughout the process.

\(^{18}\) engaging-admin@techsquare.com
\(^{19}\) shaokun2014@ia.ac.cn