Multi-game transfer learning with deep neural networks Independent Project

Þorgeir Auðunn Karlsson
School of Computer Science
Reykjavik University, Iceland
thorgeirk11@ru.is

***Abstract*—Humans have been playing games for centuries and have the ability to learn strategies in one game and apply them to another. This is called transfer learning and has been hard to replicate in general game playing (GGP). GGP agents are capable of independently playing many different games effectively, however they only learn strategies for a single game at a time. This paper presents a new approach of transfer learning that can be applied to GGP agents. The approach uses a deep neural network that predicts the likelihood of an action being made in a given state of a game. This model was tested on three different games; Chinese checkers, Connect four and Breakthrough. When pre-trained on different roles for the same game the model achieved higher accuracy after fewer training batches compared to randomly initialized model. This indicates that the approach is an effective way of transfer learning.**

I. Introduction

General game playing (GGP) seeks to create intelligent agents capable of independently playing many different games effectively without knowing the rules of the game beforehand. Humans have the ability of learning a strategy in one game and apply it on another, effectively transferring the knowledge gained between games. This ability has been hard to replicate in GGP agents.

GGP competitions have been held for many years and their matches have been recorded. A match consist of series of actions taken from each agent participating in the game. These datasets contain many different games, some of which have over a hundred thousand recorded matches. Each match was played by a GGP agent, where the agent tried to intelligently pick an optimal action for each state presented.

Deep neural networks (DNN) have gotten a lot of traction for their ability to learn from massive amounts of raw data and recognize advance patterns [**?**]. A DNN is composed of series layers, where the input is fed through each layer in turn before arriving at a output layer. In this paper we design a DNN that consists of interchangeable input and output layers for each game and a middle layers that stays persistent between every game. The hypothesis is that this architecture allows the model to learn a game independent representation of a state and can make the training of a unseen game be more effective then otherwise.

A game state in the game description language (GDL) [**?**] consists of propositions that are either true or false and therefore a state can be treated as a boolean array. This boolean array is the input into the ANN. The output of the ANN would be the probability for every action in the game for that given state.

II. Related work

General game playing has been studied for over a decade [**?**] and many attempts of transfer learning have been made [**?**], [**?**], [**?**] and [**?**]. These early works used methods that manipulated the state representation in order to find similar features between games. While effective to some degree they do not use neural networks.

Deep reinforcement learning (RL) has been shown to be capable of playing many different games. Lample and Chaplot showed a RL agent playing DOOM [**?**]. Stevens and Pradha showed an RL agent playing Tetris [**?**]. Atari 2600 games have also been played with deep reinforcement learning [**?**] and [**?**]. Deep reinforcement learning has even been shown to master the game GO [**?**].

However none of these methods have transfer learning between games only between the individual roles of a given game. [**?**] shows a neural evolution technique of applying transfer learning for general game playing.

III. State and action representations

The game states are represented in the game description language (GDL) [**?**]. GDL is a logic language where the state is represented with facts and terms. By grounding [**?**] the games description we can find all of the terms that can become true. Since each of these terms are either true or false for any given state they can be represented by a boolean array.

This boolean array therefore encodes the exact state of the game. E.g. for a very simple game like Tic Tac Toe the state would be represented with a boolean array of length $9*9*9+2$. That is, there are 9 cells and each of them can be an x, an o or empty and alternating the control between the two different roles is also needed.

The input into the neural network is a normalized version of this array, where true and false are represented by 1 and -1 respectively.

The actions are represented as a one hot encoding and the number of actions depend on the game and the role of a game. However the number of actions is not dependent

on the state it self, that is the neural network doesn't know which actions are legal given a state and therefore doesn't limit the number of actions. The actions are also not always the same between all the roles of a game, so the one hot encoding of the actions can be different between the roles of a given game.

IV. Selected Games

Three games were chosen based on the amount of available data and authors familiarity with them; Connect four, six player Chinese checkers and Breakthrough. The data behind each game is divided between the different roles of the game, both Connect four and Breakthrough have two roles, while Chinese checkers has six roles. All of the games are turn based and at each step of the game only a single role takes an action.

A. Chinese checkers

Chinese checkers is a strategy board game where up to six players compete on a star shaped board. This game has the largest state representation of the three games tested and when turned into a boolean array it is of size 253. The actions performed in this game depend on the role, that is the actions are not shared between the roles and one role cannot perform some of the actions of another role. Each role has at most 90 actions that it can perform.

Fig. 1: Chinese checkers has a star shaped board with 6 players.

B. Connect four

Connect four is a two player puzzle game. This game is the simplest of the three games tested with a state representation of size 127 and only at most 8 actions. The actions are shared between the roles, that is each action is the same for each role.

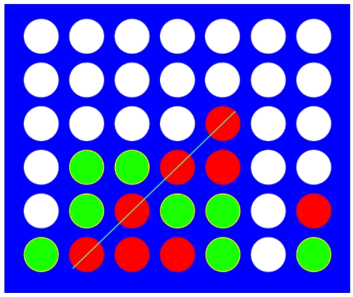

Fig. 2: Connect four. This shows a winning state for red on a diagonal line.

C. Breakthrough

Breakthrough is a two player strategy board game. This game has a state representation of size 130 and has 155 actions. The actions are not shared between the roles, same as in Chinese checkers.

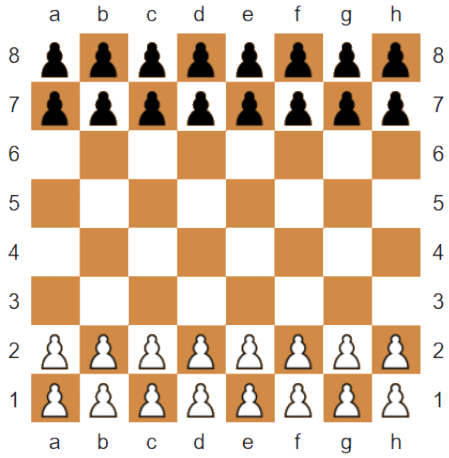

Fig. 3: Breakthrough board has the same shape as a chess board, the difference is that there are only pawns.

V. Design of the Models

Deep neural networks (DNN) are structured in a hierarchical fashion. Input into a network is fed through a series of layers before an output layer is reached. Studies have shown that it's representation of the data increases in complexity at each layer of the network. E.g. the lowest layers in the Inception model [**?**], recognize only basic lines and moving up the hierarchy, the layers recognize increasingly complex shapes and patterns.

All of the models are designed with the assumption that a DNN learns in this hierarchical fashion. That is, lower layers learn low level representations and higher level layers increase the complexity of the representation. Therefore all of our models are split into three components:

- The first few layer learn how to read the input state into some abstract representation and
- the next few layers learn how to extract important features out of this abstract representation and

- finally the last layers decide on which action is best, given those features.

The models consists of these three components; input, output and middle. Both the input and output components are changed out for each game since they handle game specific mapping and targeting. However the middle component stays persistent between every game, since it is responsible of learning the abstract reasoning and game independent representation of a state.

In this paper we created three different models with increasing complexity to achieve different learning goals. All of the models are created with Keras [**?**]. The models are single role, multi-role and multi-game. All of these models have the same foundation and will be described in more detail below.

A. Network structure

The network is build up of these three components.

1) Input component: contains a fully connected layer with 200 nodes, however the input size depends on the game since their representation is of varying sizes.

2) Output component: contains two fully connected layers and their sizes depend on the game. For both Chinese checkers and Breakthrough the first layer is of size 200, while for Connect four it is of size 50. The last layer has a size equal to the number of actions for a given role.

3) Middle component: contains two 500 node fully connected layers with a 50% dropout [**?**] between them.

All of the layers use a rectified linear activation (Relu) [**?**] except the last layer in the output component. It has a softmax activation which is used to determine the probability of each action.

B. Models

1) Single role: The simplest model only learns from a single role in one particular game. As can be seen from figure-**??**, this model connects the three components with out any branching.

2) Multi-role: This model has a single input component and multiple output components. This is due to the fact that an input conponent depends only on the game while the output conponents are role dependent. To see how the model was created see the code in figure-**??** and the model visualization in figure-**??**.

3) Multi-game: The multi game model consists of many input and output components where the middle layer is shared between all of the components. See the model visualization in figure-**??**

C. Fully connected layers

Since all games written in the GDL language, [**?**], are Markov decision problems [**?**], the history for a given state has no effect on the decision made in that state. The author decided not to invest time in recurrent neural networks (RNN) [**?**], for this problem. However it would be interesting to see if training an RNN would improve the accuracy since it could learn the play style of each agent.

D. Measure of accuracy

The output of each model is a vector representing the likelihood of an action being performed for the current input state. All of the models use a soft-max classifier to achieve this probability vector. Mean squared error (MSE) was used to measure the effectiveness as it has been shown to be an effective loss function [**?**]. This loss function is used to measure how far the prediction is from the actual label.

There are two ways to measure accuracy for this problem. The first is to use the loss function, as it gives an estimate of how far off the prediction is. The second is to use the frequency of the network predicted the correct label. The only problem with the second approach is that if for example two actions are equally likely on a given state then the network will only achieve 50% accuracy on that state.

For this paper the second approach was used to measure accuracy, that is, how often the model predicted the correct label for a given state.

VI. Experimental setup

There were three experiment types conducted; single role, multi-role and multi-game experiments. All of the experiments were run over a 100 times and a 95% confidence interval is shown on the results.

All of the experiments were done on all three games where the batch consits of 128 samples. The experiments were run for either 3.000 or 15.000 batches. After each training batch the model was estimated on unseen test data and the validation accuracy plotted.

A. Single Role Experiments

The single role experiment is trained on a randomly initialized model for 50.000 batches. This was done to show how the learning curves evolved for each of the games. It also serves as a baseline for the other experiments and are used to compare the performance of the other approaches.

B. Multiple Role Experiments

The multi-role experiments first train on the same game using all other roles except the first one and then train on the first role. The learning curve for this pre-trained model is compared to a randomly initialized model. This experiments should give an indication of the effectiveness of the middle layer to understand a single game.

C. Multiple Game Experiments

The multi-game experiments first training on the other two games before training on the test game. The learning curve of this pre-trained model is also compared to a randomly initialized model. Note that the number of

games that are used to pre-train the model is only two. The experiments should likewise give a indication of the effectiveness of the middle layer to handle multiple games.

VII. Results

This section details the results from all three types of experiments.

A. Single-role experiments

The single-role experiment, see figure-**??**, shows the learning curves for the individual games. Connect four, Breakthrough and chinese checkers are represented by the red squares, blue circles and green diamonds, respectively. the accuracy for each of the games are different. Here is the baseline runs for all of the games when run for 50.000 batches.

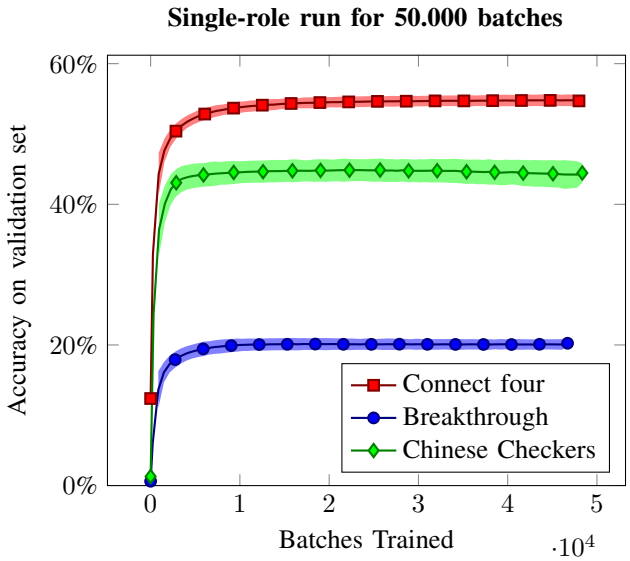

Fig. 4: Here is the baseline runs for all of the games when run for 50.000 batches. All of these runs were done on a randomly initialized models. The area around each line represents the 95% confidence interval.

B. Multi-role experiments

Here are the results from the multi-role experiments. All of the following graphs have the same structure, the y axis is the accuracy achieved and the x axis is the number of training batches trained. The lines indicate the mean accuracy where the red squares and the blue circles indicates the pre-trained and randomly initialized models, respectively. The area around each line indicates the 95% confidence intervals.

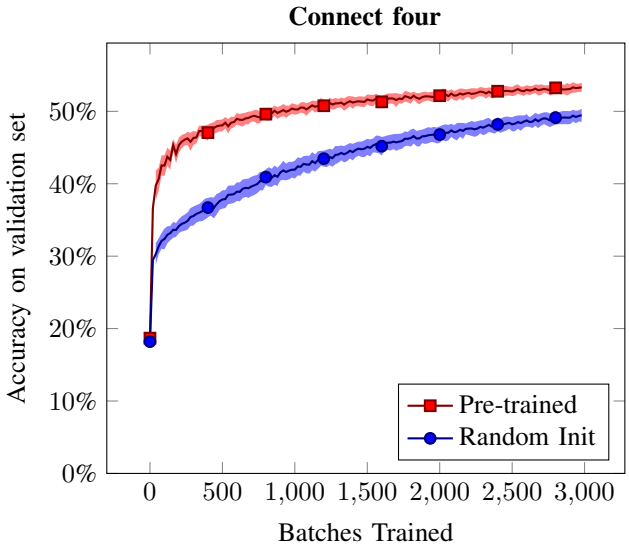

Fig. 5: Connect four pre-trained vs randomly initialized. The lines indicate mean accuracy where the red squares and blue circles represent the pre-trained and randomly initialized models, respectively. The areas around the lines are the 95% confidence intervals.

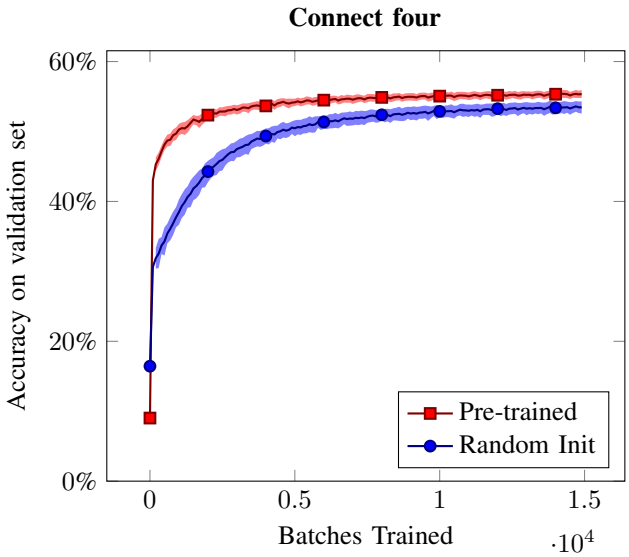

Fig. 6: Connect four pre-trained vs randomly initialized. The lines indicate mean accuracy where the red squares and blue circles represent the pre-trained and randomly initialized models, respectively. The areas around the lines are the 95% confidence intervals.

1) Connect four: As can be seen in figure-**??**, the pre-trained model has an advantage early on and achieves much higher accuracy for the first thousand batches. However the graph also indicates that the gap between the two models shrinks substantially as the learned batch number increases.

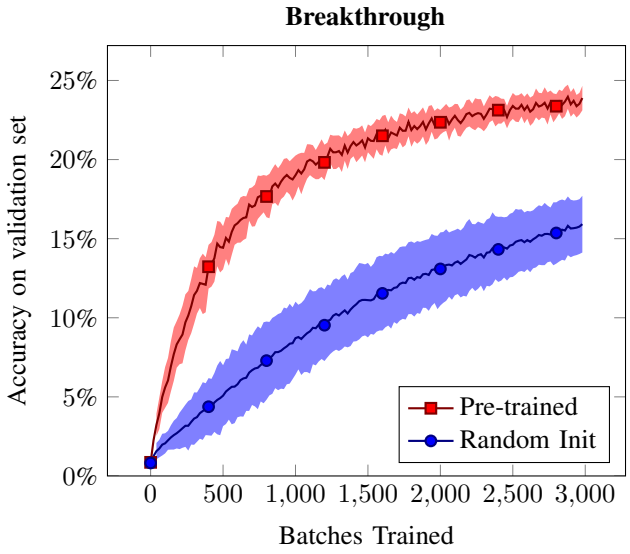

Fig. 7: Breakthrough. The lines indicate mean accuracy where the red squares and blue circles represent the pre-trained and randomly initialized models, respectively. The areas around the lines are the 95% confidence intervals.

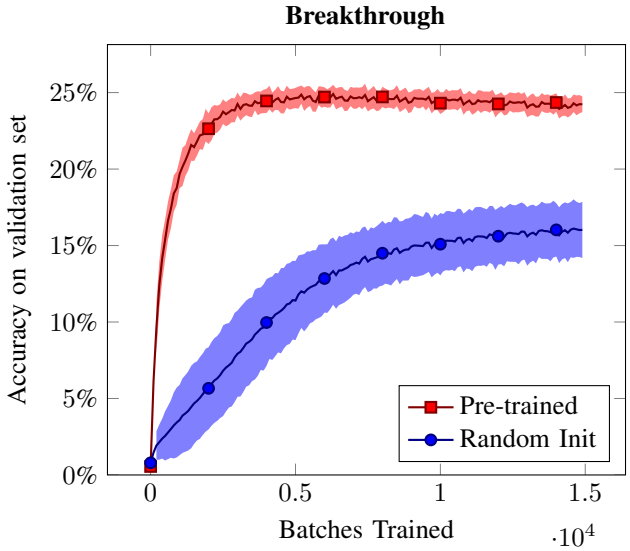

Fig. 8: Breakthrough. The lines indicate mean accuracy where the red squares and blue circles represent the pre-trained and randomly initialized models, respectively. The areas around the lines are the 95% confidence intervals.

2) Breakthrough: For the Breakthrough test, see figure-**??**, we can also clearly see that the pre-trained model has an advantage early on and achieves much higher accuracy for the first thousand batches. Note that the standard deviation is much larger then for Connect four, which should be expected since this game is more complex. The standard deviation is also lower on the pre-trained model which also is a good indicator that the training has learn some weights that are in general better suited for training.

The gap between the pre-trained and random models dose not shrink all too much for the first 3000 batches. However these results should be taken with a grain of salt since the accuracy of both models is very poor, less than 30%.

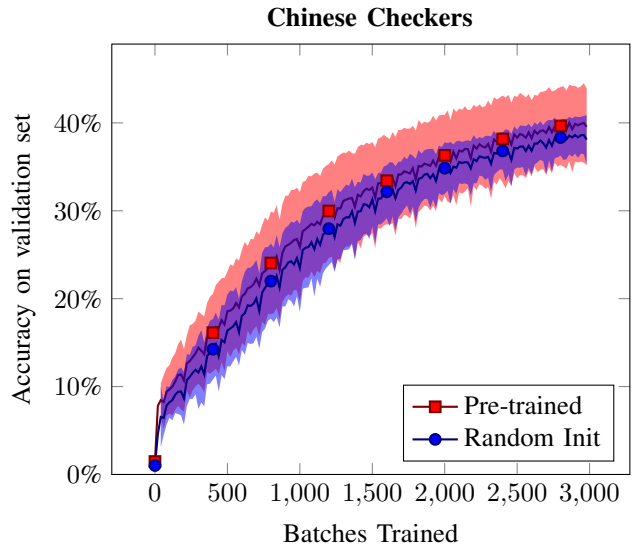

Fig. 9: Chinese checkers. The lines indicate mean accuracy where the red squares and blue circles represent the pre-trained and randomly initialized models, respectively. The areas around the lines are the 95% confidence interval.

3) Chinese checkers: For the Chinese checkers experiment, see figure-**??**.

Note that in contrast to Breakthrough the pre-trained model has a higher standard deviation than the random initialized model.

C. Multi-game experiments

Here are the results from the mulit-game experiments.

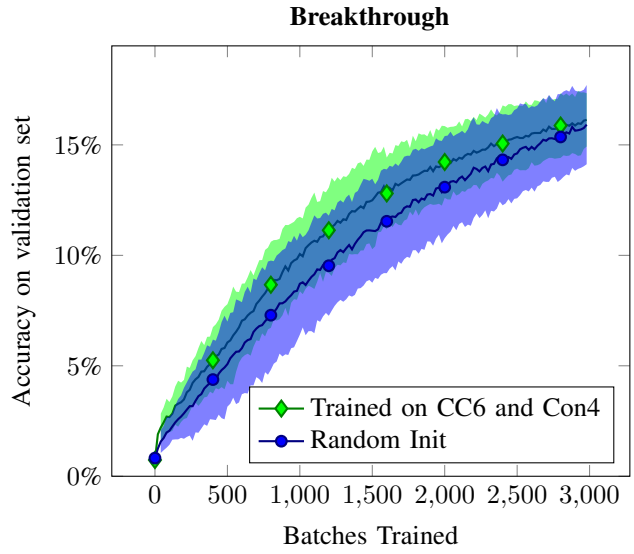

Fig. 10: Multi-game trained model (green diamonds) vs randomly initialized model (blue circles). The pre-trained model is trained on Chinese checkers 6 (CC6) and Connect four (Con4). The area around each line represents the 95% confidence interval.

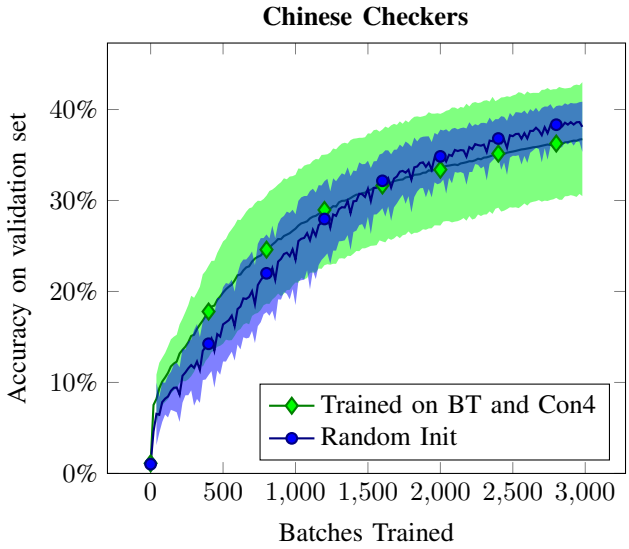

Fig. 11: Multi-game trained model (green diamonds) vs randomly initialized model (blue circles). The pre-trained model is trained on Breakthrough (BT) and Connect four (Con4). The area around each line represents the 95% confidence interval.

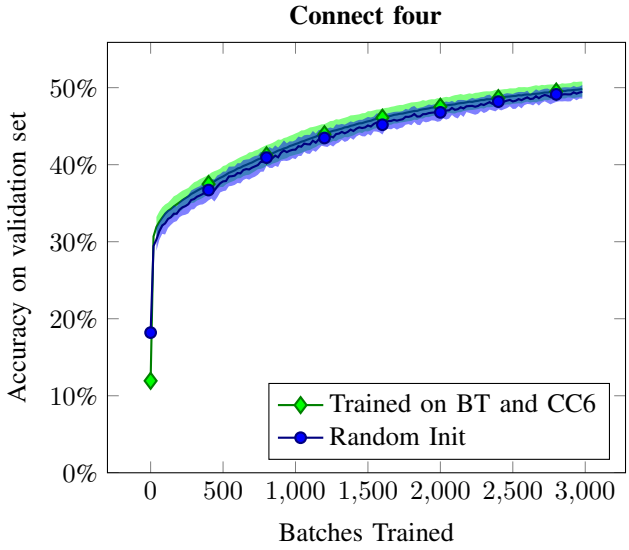

Fig. 12: Multi-game trained model (green diamonds) vs randomly initialized model (blue circles). The pre-trained model is trained on Breakthrough (BT) and Chinese checkers (CC6). The area around each line represents the 95% confidence interval.

VIII. Future Development

Missing

A. *Ideas for further development*

- Train with RNN.
  Split the data by the step counter for each match and train it on a RNN. This could learn a play strategy for each role.
- Shuffle the state and action representations.
  To reuse the same data we could shuffle the bit array and train the model on many different state and action representations for a single game. This could force the network to learn the lower and higher representation of the game in the appropriate layers and move the logic reasoning to the middle layer.
- Training with frozen middle layer.
  Training on multiple roles and then freezing the middle layer results in better performance when training on a unseen role.
- Neural evolution.
  Introduce smarter middle layer with neural evolutionary approch.

IX. Conclusions

Missing

APPENDIX

Fig. 13: Single role model. Network created with keras and is the simplest model with only a single input and output and is intended to only train on a single role for a given game. Image generated by Tensorboard.

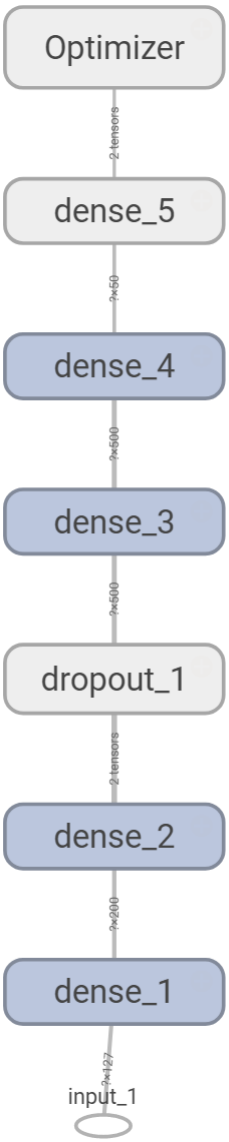

Fig. 14: Multi-role model. This network is created with Kears and has a single input but multiple outputs, one for each role. Image generated by Tensorboard.

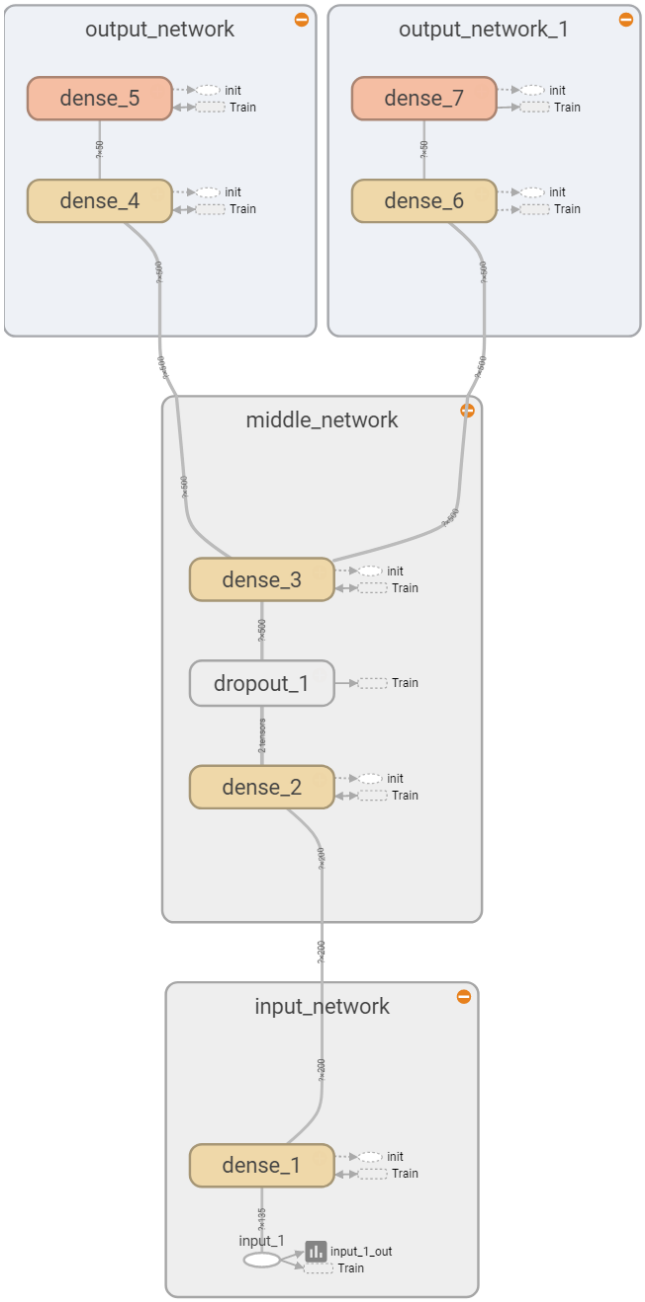

Fig. 15: Muti-role model for Connect four

```
with tf.name_scope("input_network"):
in_con4 = Input(shape=(135,))
con4 = Dense(200, activation='relu')(in_con4)

with tf.name_scope("middle_network"):
mid = Dense(500, activation='relu')(con4)
mid = Dropout(0.5)(mid)
mid = Dense(500, activation='relu')(mid)

models = []
for i in range(2):
    with tf.name_scope("output_network"):
        out = Dense(50, activation='relu')(mid)
        out = Dense(8, activation='softmax')(out)
        models.append(Model(inputs=in_con4, outputs=out))

# models = [role0, role1]
```

Fig. 16: Multi-gane model. This network is created with Kears. It has multiple inputs and multiple outputs, where each input is a game state and the outputs are the roles for those games. Image generated by Tensorboard.