Report of an experiment using Keras

Yoshihiro Mizoguchi

2018/07/1618:04:47

We made several trials using Keras to observe machine learnig effects using Keras. We consider neural networks which consider a board score of the domineering game. The domineering games with small size are well considered and we can compute all game records within practical time. But to investigage a convergence phenomena of machine learnig statuses, we choose this game using 4×4 board as an example.

At first, we compute the number of status of boards in the game. The number of possible statuses are 2442. Since we can identify symmetric boards and rotated boards are same. So the essential number of possible statuses are 488. Further, this game is profitable for the 1st player. There are 4 possible positions for the 1st player's 1st positions identifing syumetric statuses. If the 1st player choose the 2 of them, then he is determined to win if he choose appropriate positions for each his turn. If he missed to choose the first 2 profittable positions, the 2nd player is determined to win.

We note that the 1st player should choose a one of the following two reduced statuses.

.	.	.	.
.	.	.	O
.	.	.	O
.	.	.	.

6,24576

.	.	.	.
.	.	O	.
.	.	O	.
.	.	.	.

96,1536

If the 1st player chooses a one of the following two reduced statuses, the 2nd player is determined to win.

.	.	.	.
.	.	.	.
.	.	.	O
.	.	.	O

3,12,12288,49152

.	.	.	.
.	.	.	.
.	.	O	.
.	.	O	.

48,192,768,3072

Although this game is determined at the first stage, it is a good example to investigation of machine learning options and statuses at the begining.

1 Dense Neural Network

At first, we started to choose a simple dense network(cf. Fig.1). We consider a score function of a board status. Each board with dominos have an advantage for a player, we decide a score of the board to 1 if the next player fails. That is a player can check his action's evaluation by doing an act and evaluating the score of the board.

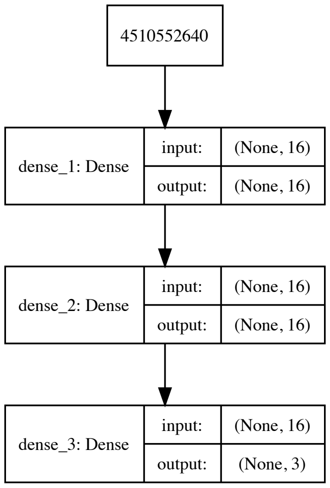
This is a simple network, we put all possible 2442 cases board scores as a training data.

We showed the learning records of 1000 epochs traing and 9000 epochs traing.

2 Convolutional Neural Network

Next, we introduce a simple convolutional network(cf. Fig.3).

We showed the learning records of 1000 epochs traing and 2000 epochs traing. As you can see, it is easy to understand advantages of convolution layers.



```
#
# Keras implementation
#
model = Sequential()
model.add(Dense(16, activation='relu', input_dim=16))
model.add(Dense(16, activation='tanh'))
model.add(Dense(3, activation='softmax'))
model.compile(loss='categorical_crossentropy',
              optimizer='sgd', metrics=['accuracy'])
```

Figure 1: Simple Dense Nueral Network

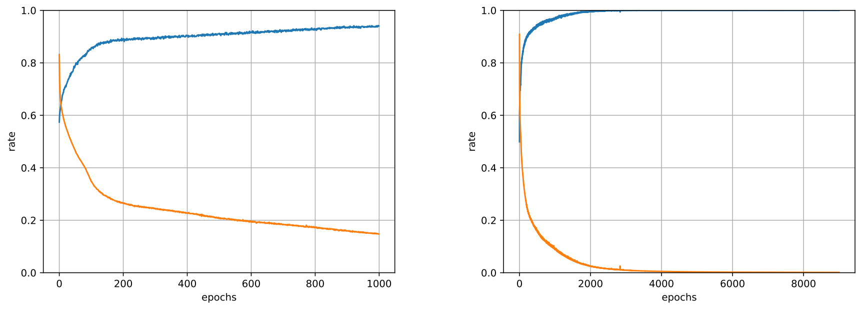
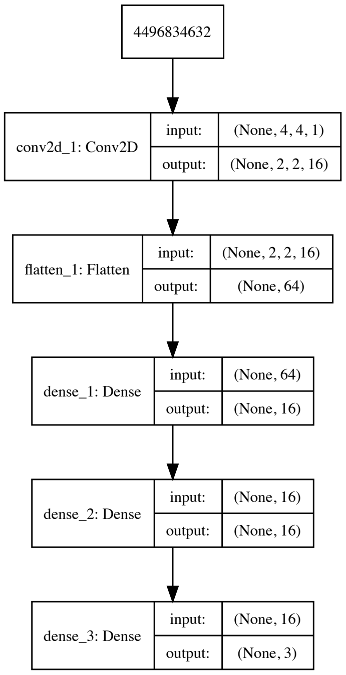


Figure 2: Simple Dense Nueral Network Training



```
#
# Keras implementation
#
model = Sequential()
model.add(Conv2D(16, kernel_size=(3, 3),
                 activation='relu',input_shape=(4,4,1)))
model.add(Flatten())
model.add(Dense(16, activation='relu'))
model.add(Dense(16, activation='tanh'))
model.add(Dense(3, activation='softmax'))
model.compile(loss='categorical_crossentropy',
              optimizer='sgd', metrics=['accuracy'])
```

Figure 3: Simple Convolutional Nueral Network

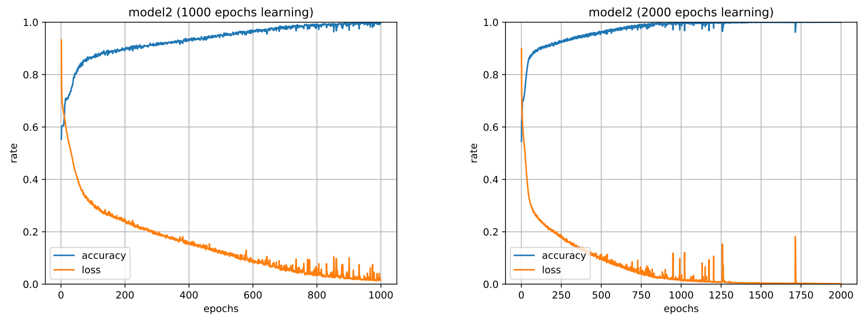


Figure 4: Simple Convolutional Nueral Network Training

3 AI Players for domineering game

After learning, we can make an AI player using our learned neural network.

```
class AIPlayer:
   def __init__(self, modelname='model1', epochs=10):
       self.modelname = modelname
       self.epochs = epochs
       fn = './'+modelname+"_{0:05d}".format(epochs)
       self.model = load_model(fn+'.h5')
       return None
   def tolist(self):
       for x in self.perfect:
           y=[x[0]]+(x[1].reshape((16)).tolist())
       return y
   def eval(self, board):
       b = board.reshape((16)).tolist()
       r = self.model.predict(np.array([b]))
       return([0,1,-1][np.argmax(r)])
   def play(self, board):
       acts = calc_actions(board)
       a = []
       for action in acts:
           next_state = calc_next_state(board,action)
           v = self.eval(next_state)
           if (v == 1):
               a.append(action)
       if (a == []):
           return choice(acts)
       else:
           return choice(a)
```