# 1 Ray について

OSS 版の Ray に以下のような改良を施してあります.

1. 木探索の着手評価に Factorization Machines + Bradley-Terry モデルを利用
2. 木探索部とシミュレーション部に特徴を追加
3. 木探索部, シミュレーション部ともに細かなパラメータ調整
4. Deep Learning(後述)

# 2 Deep Learning 対応

## 2.1 利用する深層学習フレームワーク

ネットワークの学習には Pytorch を利用しています.

## 2.2 ネットワークアーキテクチャ

Preactive Residual Network に対して以下の手を加えています.

1. ReLU を使うところに TanhExp
2. 各 Residual Block に SE モジュールを付与

出力は以下の 5 つありますが, 対局時には Player's Policy, Value, Ownership, Score の 4 つを利用します.

- Player's Policy：次の自分の着手の予測
- Opponent' Policy：自分が着手した後の相手の着手の予測
- Value：最終的な勝敗の予測
- Ownership：各交点の占有率
- Score：最終的な目数差の予測

入力には石の配置や呼吸点数, 着手の履歴以外に

- 対局ルール：日本ルールか中国ルール化
- コミの値：コミの値の 0.1 倍
- 持碁の有無：持碁ありの時は 1, 持碁なしの時は 1

を入れることで, いろいろな棋譜を混ぜて学習できるようにしています.

## 2.3 自作 CUDA レイヤ

以下のレイヤに関しては, CUDA で自作しました.

- TanhExp Layer
- Concatenate Layer

## 3　強化学習の方式

完全にゼロから学習すると非常にコストがかかるため, 人間の棋譜で事前学習したネットワークを起点に強化学習を実施しています. 日本ルールで自己対局するとパスのタイミングが非常に難しくなるため, Ownership の予測値を使い, 未確定の地があるときはパスの評価を低く補正することで早期のパスを回避しつつ, 地が確定し次第パスするようにしています. 終局時の地の判定のロジックを自力で実装するのがかなり大変なので, 終局時の死活判定には GNU Go を利用しています.