「にゅ～ろす」ニューラルネットワーク仕様書

１．リバーシゲームの概要

リバーシゲームとは、２名で対戦するボードゲームです。ボードには８×８マスの格子が切られ、各々のマス目の中に石が置かれます。対戦する２名は黒石を置く側と白石を置く側の双方に分かれます。黒石側が先手で、黒石側と白石側が交互に１つづつ自分の石をボード上に置いていきます。最後に多くの石がボード上にある側の勝利となります。

ゲーム開始時には、ボードの中心に黒白２個づつの石が置かれます。ゲーム中、石を打てる場所は、既にボード上にある自分色の石と、これから打つ石との間に、隙間なく相手色の石が並んでいる所だけです。石を打つ際には、このようにして挟まれた場所を自分の石に置き換えます。新しく打った石から見て８方全てについて、相手色の石が自分色の石に挟まれている場合は、置き換えの対象となります。ゲームの途中で１方だけが打ち場のない場合は、パスとなり順番が飛ばされます。また、双方ともに打ち場のない場合はゲーム終了となります。

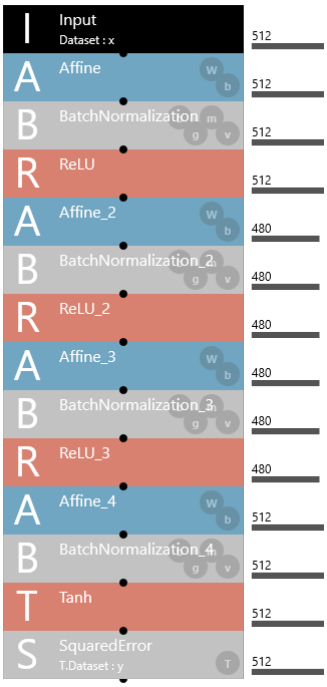
２．「にゅ～ろす」の概要

にゅ～ろすは思考ルーチンにニューラルネットワークを使用した、人間 v.s.コンピュータの対戦型リバーシゲームです。思考ルーチンは対戦を重ねるごとに強く賢くなっていきます。ニューラルネットワークデータは、にゅ～ろす終了時ににゅ～ろすの作業フォルダ中に、データファイル"neuroth.dat"として格納されます。

にゅ～ろすは単純にリバーシゲームを楽しむことを目的として開発しました。そのため、カスタマイズ機能やオプション機能はほとんど付いていません。その代わり、リバーシゲームを楽しむための手続きを極力簡単にして、実行後すぐにゲームが楽しめて、中断したい時にはすぐに終了できるように設計しています。

3．ネットワークの仕様

「にゅ～ろす」version 4.32 では、各面の評価関数として、5 層のニューラルネットワークを使用しています。ネットワーク構造はほぼ下図の通りです。
※実際には入力層出口にも閾値と活性化関数を使用しています。



3.1. 入力層の仕様

入力層は 512 個(64 x 8)の float32 で構成しています。このうち 64 個を 1 グループとして、8 グループの面情報を格納しています。各グループの情報は次の通りです。現在の面に対して、自手が置く位置を決めた上で、次の面情報を入力層に格納します。

(1) 現手を置いたとき、変化するマスの位置
(2) 現手を置いたとき、次手が取れる位置
(3) 現在の面の、自石の位置
(4) 現在の面の、他石の位置
(5) 現在の面の、空マスの位置
(6) 現在の面になるとき、変化した位置
(7) 現在の面で、自手を置ける位置
(8) 現在の面の、自石か他石かの区別

3.2. 隠れ層の構造

隠れ層は 2 層が 512 個の float32、3･4 層が 480 個の同じく float32 で構成しています。入出層を含め全層は、全層結合層(Affine)で接続しています。また、全層で正規化(Normalization)を行っています。

3.3. 活性化関数

入力層と最終層を除き、ReLU を採用しました。次のような簡単なコードで実現できるので、CPU/GPU いずれの実行時も高速に動作します。

- ReLU 処理　　: (in > 0) * in
- 微分処理　　: (in > 0) * grad

また、出力層は±1 を取り出すために、tanh を採用しています。コンパイル時に tanh の計算結果を配列に格納し、実行中は配列を参照することで高速動作させています。

3.4. 出力層の仕様

出力層は 64 個を 1 グループとして、現在の面に置くべきマスの位置に 1 が、置けるけど置くべきでない位置に-1 が、それぞれ格納されることを期待値とします。全 8 グループの格納場所には、奇数グループは 1 グループと同じ値を、偶数グループは 1 グループの正負反転値を期待値とします。差動化して正負のバランスを取ることにより、学習の進み方を安定化しています。

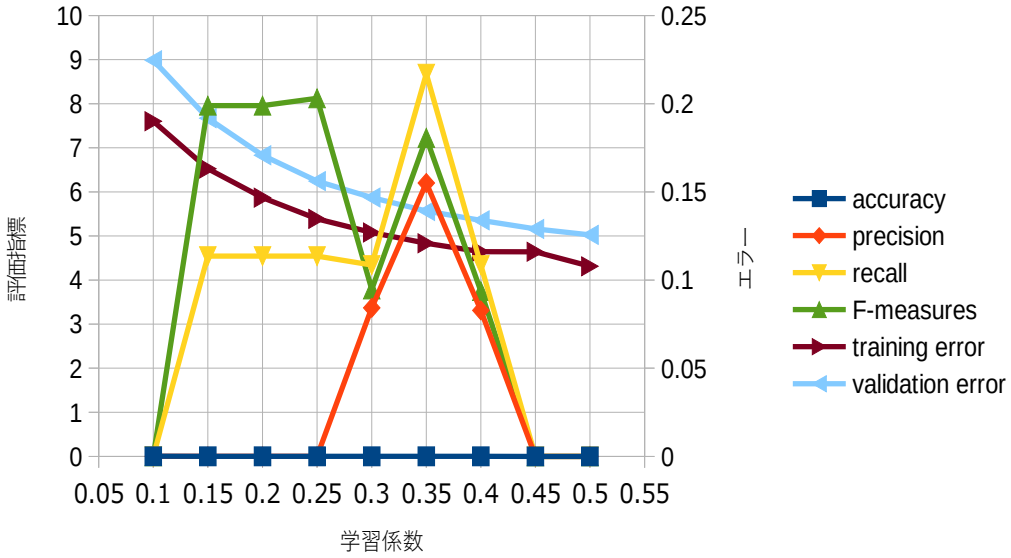
3.5. 最適化手法

にゅ～ろすでは、SGD を用いた最適化を行っています。学習データが面単位なので各回の変化が大きく、学習結果をネットワークに長期記憶させることが課題です。そのため、比較的大きな学習係数と小さな学習係数をネットワーク中で混在させて、各回の特徴の記憶と全体の流れの記憶を両立させています。

４．細かな調整

4.1. 学習係数の依存性

Neural Network Console を用いて、SGD の学習係数依存性を見てみました。学習係数が 0.15~0.4 の範囲でネットワークが反応しているように見えます。今版の学習係数は 0.2~0.4 の範囲で使用しています。



4.2. ネットワークの発散

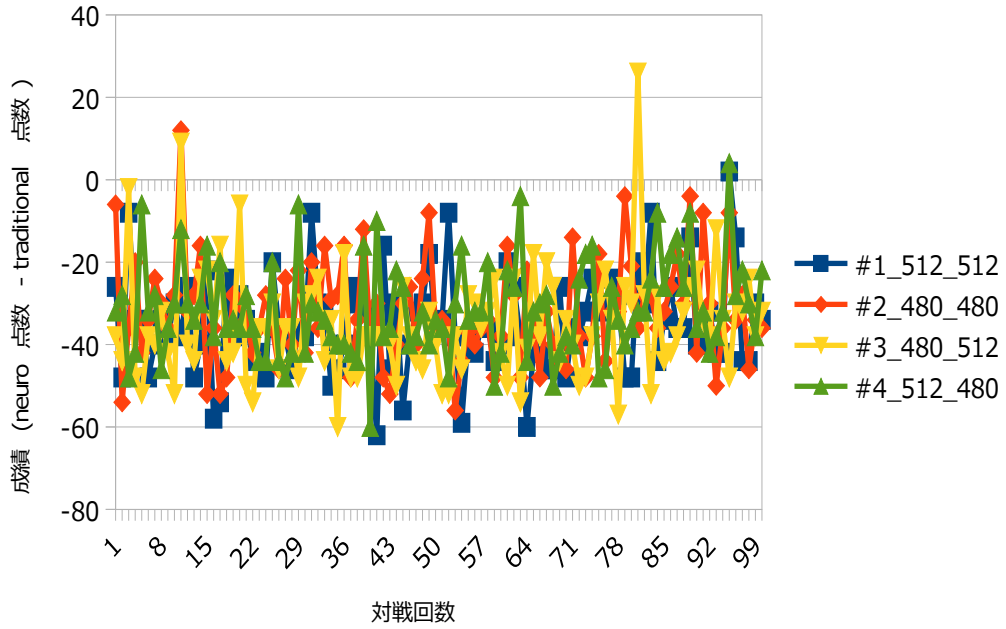
活性化関数に tanh を使った場合、バグの無い限り発散はしませんでしたが、ReLU を使うと簡単に発散してしまいます。にゅ～ろすでも tanh を生のままで、デモモードにて 100 戦くらいやると、中間層の結合係数や閾値が発散して nan になってしまいます。

これを抑えるために全層に σ を用いた正規化を使用していますが、まだ十分でないケースがあます。今版では追加の暫定対策として次の特殊な ReLU を使用しています。今後、何らかの見直しを行う予定です。

- ReLU 処理　　: (in > 0 && in < 1000) * in + 1000 * (in >= 1000)
- 微分処理　　: (in > 0 && in < 1000) * grad

4.3. ネットワークのくびれ

多層ニューラルネットワークの隠れ層のセル数を絞る方法は、多くの文献で語られるようになりました。今版では、入出力層の 512 個に対し、3・4 層を 480 個に絞っています。にゅ～ろすのデモモードを使って、この効果を見てみました。結果はあまり変わらなかったのですが、両層とも 480 個のときの平均成績が若干良く、発散の程度も低かったので採用しました。



5.「にゅ～ろす」のコマンドラインオプション

-help オプションで usage を表示します。

```
> neuroth -help

Neuroth version 4.32 Copyright (c) 1994-2020 by Kappa v.v.v
usage: neuroth [option]
option)    -mode     <MODE>   : MODE = win*|text|demo
           -think    <TYPE>   : TYPE = neuro*|traditional
           -cuda     <#GPU>   : with CUDA on #GPU
           -dump              : debug info for neural network
           -nnc               : csv for neural network console
```

6. ネットワーク情報の出力

6.1. ネットワーク統計データの出力 (-dump)

テキストモードかデモモード(-mode text | demo)実行時に、1 戦終了毎にネットワークの統計データを表示します。出力対象のネットワークは、s<n>:閾値、n<n>:結合係数、d<n>:最終手の伝搬データ、b<n>:最終手の逆伝搬のデータで、n は入力層側からの順番です。統計データは該当ネットワーク中の平均(ave)、標準偏差(sd)、最小値(min)、最大値(max)を表示します。

```
             ave         sd          min         max
s<n>         0.06        0.62        -1.40       2.93
       :                       :
             ave         sd          min         max
n<n>         -0.05       0.55        -3.70       3.53
       :                       :
             ave         sd          min         max
d<n>         -0.509      0.763       -1.000      1.000
       :                       :
             ave         sd          min         max
b<n>         0.004       0.301       -1.247      1.266
       :                       :
```

6.2. Neural Network Console 用データの出力(-nnc)

ソニー製 Neural Network Console 用の CSV ファイルを表示します。全てのモードで有効です。リダイレクトでファイルに出力して使用します。

以上