WANNにおけるシナプス荷重の調整と活性化関数の慎重な選択

(指導教員 山口 智 教授)

山口研究室 2031133 増田 瑞樹

1. WANN

Weight Agnostic Neural Networks(WANN) は，ネットワーク構造を更新することで，どんなシナプス荷重に対してもそれなりの精度でタスクを解くことができるようになる．変更するノードは隠れ層からランダムに選択され，現在採用されている活性化関数をのぞいた活性化関数の中から，同等の確率で選択される．また，ネットワークの評価には-2.0, -1.0, -0.5, +0.5, +1.0, +2.0 の共有重みを用いた 6 つの評価の平均を採用している．

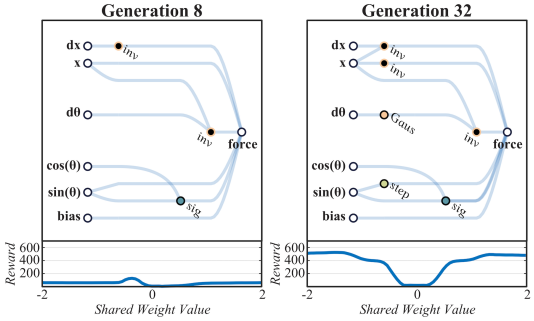

図 1: 共有重みと評価値とネットワーク構造

2. 提案手法

提案手法では，2 つの活性化関数の慎重な選択方法を提案している．ひとつは前回説明したネットワークがよく使用するであろうノードへの入力が $-r$ から $+r$ までの関数同士の出力の差の積分を考慮する手法 (1)，今回実装したもうひとつの手法はその個体が実際に適応度を測定したときに入力されていた入力ノードの値を参考に，該当ノードの入力がいくつかを概算し，それに対応する出力の差を考慮する手法 (2) になる．
これら 2 つの手法によって活性化関数同士の距離を計算し，式 (3) に適用して変更確率を決定する．関数同士の距離の逆数に ϵ を加えたものをルーレット選択している．
in_f は，テスト時の入力ノード情報と共有重みの値がわかれば導出することができる．該当ノードの入力 in_f は，それに接続されているノードの出力とシナプス荷重がわかれば計算することができ，シナプス荷重は共有重みから，ノードの出力はノードの入力と活性化関数からわかる．最終的に入力ノードの出力とその時使用していた共有重みを保存しておけばネットワークのすべてのノードの入力を計算することができる．
また，入力ノードの出力はテスト時の入力からハイパーパラメータで指定された数だけランダムに選択し採用する．これはミニバッチ法のような効果を生み，すべての入力を考慮したり一回だけの入力考慮したりするよりも安定することを期待する．

$$d(f_a, f_b) = \int_{-r}^{r} (f_a(x) - f_b(x))^2 \tag{1}$$

$$d(f_a, f_b) = (\sum_m (f_a(in_m)) - \sum_m (f_b(in_m)))^2 \tag{2}$$

$$P_i = \begin{cases} \dfrac{1}{d(f_s, f_i) + \epsilon} & (i \neq s) \\ 0 & (i = s) \end{cases} \tag{3}$$

表 1: 説明

変数	意味
P_i	s から i へ活性化関数 ID が変更される見込み
$d(f_s, f_i)$	活性化関数が s と i の距離
f_i	ID が i の活性化関数
i	活性化関数 ID
s	現在の活性化関数 ID
m	ミニバッチサイズ x 共有重み x 試行回数
in	ノードに入力される値
ϵ	逆数の大きさを抑えるための小さい値
r	関数の考慮範囲

$input_m$ はあらかじめ保存してある入力ノード情報から計算することができる．入力ノード情報はどの個体か，どの共有重みかにそれぞれミニバッチサイズ回の入力情報を保存してある．異なる活性化関数を用いてノードの出力の差を求める．

3. 進捗

入力ノード情報の取得，任意の個体，ノード，共有重み，入力状況での出力の取得は実装済み．現在プログラムを走らせている．

4. 今後の目標

1. 活性化関数変更方法による適応度の比較
2. シナプス荷重の調整についての実装

ソースコード 1: 入力ノード情報の取得方法

```
for i in 個体 {
    for j in 共有重み {
        for k in 試行 {
            for x in ステップ {
                inputlist[i][j] に 入力ノードリスト を追加
            }
        }
        for y ミニバッチサイズ {
            batchlist[y] = list[i][j][乱数]
        }
        list[i][j] = blist
    }
}
```

ソースコード 2: in(f) の取得方法

```
for n in 活性化関数 {
    for m in 共有重み {
        for o in ミニバッチサイズ {
            table[n] += nodeOut(活性化関数を変更したいノードID, n, m, list[活性化関数を変更したい個
                体][m][o])
        }
    }
}

function nodeOut(該当ノード, 該当関数, 共有重み, 入力ノードリスト){
    for i in 目的地が該当ノードの接続 {
        if i の出発地が入力層 {
            前出力 += 入力ノードリスト [i の出発地]
        }
        if i の出発地が隠れ層 {
            前出力 += nodeOut(i の出発地のノード ID, 活性化関数ID, 共有重み, 入力ノードリスト)
        }
    }
    入力 = 前出力 * 共有重み
    出力 = 該当関数 (入力)
    return 出力
}
```

表 2: 変数の説明

変数	意味
list	4 次元配列 (個体数 x 共有重みの数 x ミニバッチサイズ x タスクの入力ノード数)
共有重み	-2.0, -1.0, -0.5, +0.5, +1.0, +2.0 の 6 つ
試行	初期条件によりたまたま精度が悪くなるのを抑制する，今回は 4
ステップ	個体がテストに存在し始めてから，テストが終わるまでのフレーム，1 秒に 50 ステップ
ミニバッチサイズ	データを切り取る大きさ，今回は 8