Modelowanie funkcji oceniającej w grach

Arkadiusz Paterek
paterek@mimuw.edu.pl
http://www.mimuw.edu.pl/~paterek/

18 listopada 2004

Postać funkcji

Funkcja oceniająca obliczana podczas działania programu ma postać:

$$eval_k(S) = \boldsymbol{w}_k^T \boldsymbol{f}(S)$$

gdzie $\boldsymbol{f}(S)$ to wektor n własności pozycji S, a $\boldsymbol{w}_k$ to wektor n wag przypisanych do klasyfikacji k.

Dla każdej klasyfikacji $k \in 1 \ldots K$ chcemy tak wybrać wektor wag $\boldsymbol{w}_k$ w funkcji:

$$e_k(S) = g(\boldsymbol{w}_k^T \boldsymbol{f}(S))$$

gdzie g to funkcja rosnąca i różniczkowalna, aby zminimalizować średni kwadrat błędu wartości funkcji e_k dla k-tego zbioru pozycji treningowych.

Jako funkcję g wybrano funkcję sigmoidalną: $g(x) = 1/(1 + e^{-x})$.

Równanie funkcji błędu dla k-tej klasyfikacji ma postać:

$$E_k(w) = \frac{1}{N_k} \sum_{i=1}^{N_k} (r_{ki} - g(\boldsymbol{w}^T \boldsymbol{f}(S_{ki})))^2$$

gdzie S_{ki} to i-ta pozycja spośród N_k pozycji treningowych należących do klasy k, a r_{ki} to ocena wyroczni dla tej pozycji.

Metoda najszybszego spadku

Pochodne cząstkowe funkcji błędu przyjmują postać:

$$\frac{\partial E_k}{\partial w_j}(\boldsymbol{w}) = -\frac{2}{N_k}\sum_{i=1}^{N_k} g'(\boldsymbol{w}^T \boldsymbol{f}(S_{ki})) * (r_{ki} - g(\boldsymbol{w}^T \boldsymbol{f}(S_{ki}))) * f_j(S_{ki})$$

Do dopasowywania wag wybrano metodę najszybszego spadku (ang. *gradient descent*). Jest to metoda iteracyjna. Najpierw wybieramy przybliżenie początkowe wektora wag $\boldsymbol{w}^{\mathbf{(0)}}$. Wagi kolejnych iteracji są obliczane według wzoru:

$$\delta^{(t)} = -\alpha \nabla E_k(\boldsymbol{w}^{(t)})$$

$$\boldsymbol{w}^{(t+1)} = \boldsymbol{w}^{(t)} + \delta^{(t)}$$

gdzie $\nabla E_k(\boldsymbol{w})$ to wektor pochodnych cząstkowych funkcji E_k w punkcie $\boldsymbol{w}$, a parametr $\alpha > 0$ to długość kroku.

Metoda *momentum* daje lepszą zbieżność:

$$\delta^{(t)} = -\alpha_1 \nabla E_k(\boldsymbol{w}^{(t)}) + \alpha_2 \delta^{(t-1)}$$

Wybór pozycji treningowych

- duża baza - 1523888 partii
- tylko partie graczy z rankiniem ELO, z podanym wynikiem - 112768 partii
- „ciche'' pozycje - ponad 6.2 mln pozycji
- za ocenę wyroczni posłużyły wyniki partii: wygrana - 0.999, remis - 0.5, przegrana - 0.001

Przebieg eksperymentu

Praca magisterska p.t. „Modelowanie funkcji oceniającej w szachach”: `http://www.mimuw.edu.pl/~paterek/pracamgr_ap.pdf`

Tablica 1: Wyniki ustalania wag

1	2	3	4	5	6	7
F1	W1 (778)	K1 (3)	3527869	10000	31h	0.119203
F2	W1 (778)	K2 (91)	1398573	1000	17h	0.116221
F3	W2 (5228)	K2 (91)	1398573	500	23h	0.112209
Gosu	(112)	(6)	?	?	?	?

Legenda:

1. nazwa funkcji oceniającej
2. zestaw własności pozycji (liczba własności)
3. schemat klasyfikacji (liczba klas)
4. $\max\limits_k N_k$
5. liczba iteracji w każdej z klas
6. łączny czas iteracji
7. średni kwadrat błędu dla wszystkich pozycji treningowych

Algorytm alfa-beta cięć

… w wersji fail-soft, negamax:

```
int alphabeta(depth, alpha, beta)
  if (cutoff(depth))
    return evaluate() // funkcja oceniająca
  best = -INF
  for each move do
    makeMove(move)
    result = -alphabeta(depth + 1, -beta, -alpha)
    unmakeMove(move)
    if (result >= beta)
      return result // beta cięcie
    if (result > alpha)
      alpha = result
    if (result > best)
      best = result
  return best
```

$r = \text{alphabeta}(d,\ \alpha,\ \beta)$
$m = \text{minimax}(d)$

$$
\begin{array}{lcl}
r \leq \alpha & \Rightarrow & m \leq r \\
r \geq \beta & \Rightarrow & m \geq r \\
r \in (\alpha, \beta) & \Rightarrow & m = r
\end{array}
$$

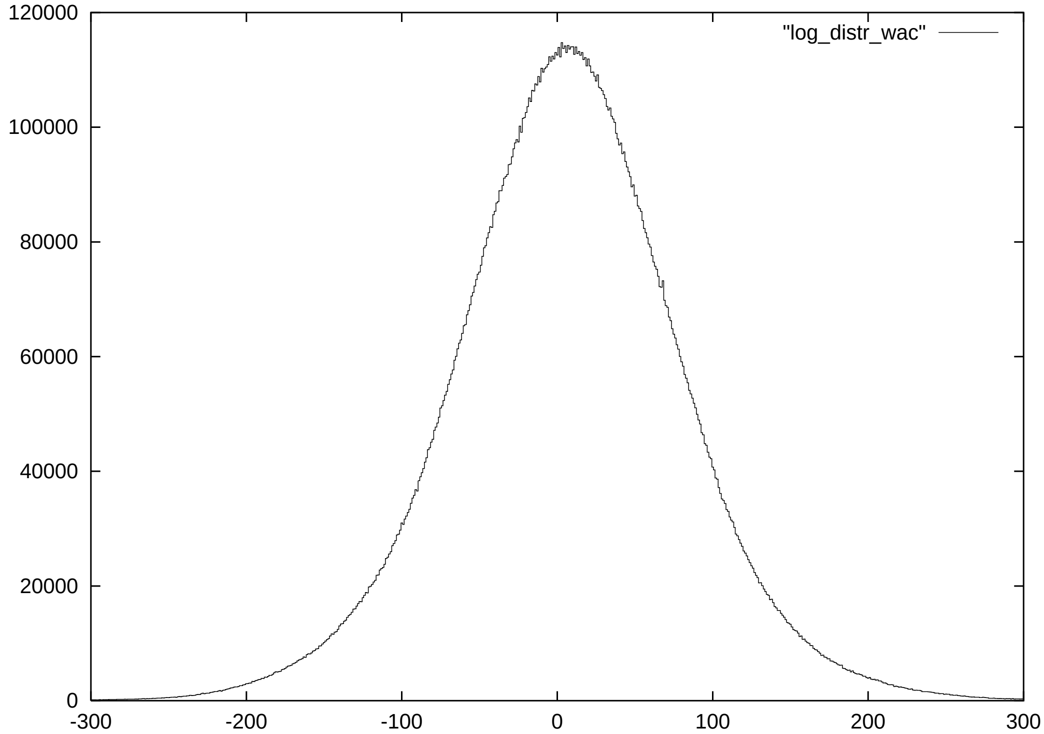
Algorytm MTD(f)

Sformułowanie algorytmu przy użyciu funkcji alphabeta:

```
int mtdf(depth, guess)
  lowerBound = -INF
  upperBound = INF
  result = alphabeta(depth, guess - 1, guess)
  if (result >= guess)
    lowerBound = result
    while (true)
      result = alphabeta(depth, lowerBound, lowerBound + 1)
      if (result <= lowerBound)
        return result
      else
        lowerBound = result
  else
    upperBound = result
    while (true)
      result = alphabeta(depth, upperBound - 1, upperBound)
      if (result >= upperBound)
        return result
      else
        upperBound = result
```

Leniwa ocena pozycji

Rozkład różnicy między oceną materialną, a oceną pełną (test WAC, losowe pozycje z przeszukiwania):



$\mu = 6.83$, $\sigma = 74.01$

```
int evaluate(alpha, beta)
  score = evaluate_material()
  if (score + MARGIN <= alpha)
    return score + MARGIN
  else if (score - MARGIN >= beta)
    return score - MARGIN
  ...
```

Eval hash

Stan projektu

Program Gosu:
`http://rainbow.mimuw.edu.pl/~ap/gosu/gosu.html`

Siła taktyczna programu:

Tablica 2: Wyniki testu WAC

program	1s	2s	5s	10s	NPS
M	254	267	278	284	143708
R	251	269	281	287	117682
F1	245	258	279	288	113860
F2	243	258	272	283	109502
F3	237	255	271	282	61835
Gosu v.0.5	272	281	290	293	262484
Crafty v.18.15	270	281	294	293	*589201

Jest dostępnych ponad 300 silników szachowych, komercyjnych i niekomercyjnych, zgodnych z interfejsami: WinBoard, UCI lub ChessBase.

WBEC - miejsce ok. 80/195
http://wbec-ridderkerk.nl

Surak's Computer Chess Trophy - miejsce 103/304.
http://www.grailmaster.com/misc/chess/comp/elo1.htm

MPPS - 5. miejsce http://mpps.64pola.pl

Podsumowanie

- Metoda Michaela Buro ustalania parametrów działa dla gry w szachy.

- Leniwa ocena pozycji działa dobrze w połączeniu z algorytmem MTD(f).

- Zastosowanie technik: leniwej oceny pozycji oraz *eval hash* pozwala na stosowanie skomplikowanej oceny pozycji (kosztownej obliczeniowo).

Co dalej?

- Jak pokazać, że metoda Buro ustalania parametrów działa dobrze w programach grających w szachy?
  - Zastosować do ustalenia parametrów funkcji oceniającej silnego programu.
  - Porównać *value adaptation* z *move adaptation*.

- Automatyczne ustalanie innych parametrów programu, np. rozszerzenia przeszukiwania, redukcja w technice pustego ruchu, itd.