Hierarchia domniemań

Im reguła jest bardziej szczegółowa tym powinna być ważniejsza. * to tak jak z Bayesowską informacją

Multiplekser

- Szyny adresowe, u nas dwie [0,1]
- Szyny danych, u nas cztery [0,1,2,3]

W szynie adresowej: która z szyn danych ma być użyta do przetworzenia informacji * [0, 0] -> 0 * [1, 1] -> 3

```
D3D2D1D0A1A0: output
* * * 0 0 0 : 0
* * * 1 0 0 : 1
* * 0 * 0 1 : 0
* * 1 * 0 1 : 1
* 0 * * 1 0 : 0
* 1 * * 1 0 : 1
0 * * * 1 1 : 0
1 * * * 1 1 : 1
```

Równoważne z ośmioma regułami klasyfikatorów. Dość dużo jak na prosty układ. Można to obejść - wprowadzić poprawkę do reguł samych klasyfikatorów...

$$B_i = C_{bid}[f(S_p)]S_i$$

* S_p - szczegółowość wzorca, liczba ustalonych bitów we wzorcu (zamiast gwiazdek!). * $f(S_p) = c + S_p$, $c > 0$

Nowa wersja reguł: * 4 reguły gdzie cośtam daje zero * dodatkowa ******: 1

Więc zeszliśmy z 8 do 5. Jedynka będzie chciała zadziałać ZAWSZE, ale priorytet mają te które dają zero. Więc to jest:

```
zera = {"0***11",
        "**1*01",
        "*0**10",
        "0***11"}
if input in zera:
  return 0
else:
  return 1
```

Wewnętrzna pamięć systemu może pomieścić tylko jeden komunikat - tylko jedna reguła może być użyta. Jeśli jest sygnał pasujący do reguł {1, 3, 5, 7} to z uwagi na ich siłę tylko jedna z nich zwróci wartość na output.

Jeśli jest sygnał pasujący do reguł o siłach 2, 4, 9, 8 to zadziała reguła 9 i nie ma ona żadnej konkurencji.

Przykład symulacji

Procent odpowiedzi poprawnych uśrednionych po 50 krokach * Bez hierarchi domniemań: oscyluje między 90% a 96% poprawności odpowiedzi * Z hierarchią domniemań: zbiega od razu do 100% i lekko oscyluje od 98% do 100%

Możliwa liczba reguł tutaj: $3^6 * 2 $, 3 opcje [0, 1, *], 6 pól, 2 opcje zwrotu, razem 1458

Możliwa strategia w rzeczywistym przypadku gdy nie znamy tych reguł

Z 1458 losowych reguł losujemy 100 reguł które konkurują ze sobą, żeby właściwie określić działanie multipleksera

Przepuszczamy przez "rynek konkurencyjny" z hierarchią domniemań i co 5k iteracji przepuszczamy reguły przez algorytm genetyczny, żeby powymieniał je i dobrał dobrał efektywność.

Strategie ewolucyjne 1965

1. Jak w monte carlo - brak genotypu, dokonujemy ewoluucji bezpośrednio na fenotypie
2. Kluczowa rola mutacji

Szukamy pewnego optymalnego rozwiązania $\vec{x} \in R^K$

- $\vec{x}_{t+1} = \vec{x}_t + \vec{\eta}_t$
- Z $(\vec{x}_t, \vec{x}_{t+1})$ wybieramy lepszego (tzw. strategia jeden plus jeden)
- $G(\eta_t^{(1)})$ = normal(mean=0, std = $\sigma_t^{(1)}$)

Reguła 1/5 sukcesu

Jeśli w L kolejnych generacjach liczba mutacji zakończonych sukcesem jest mniejsza niż L/5, należy zwiększyć siłę mutacji

patrz simulated_anneling.py

selekcja $\mu + \lambda$

Mamy μ rodziców, λ dzieci, wybieramy μ spośród $\mu + \lambda$ osobników. Turniejowe losowanie. Konkurują ze sobą rodzice i dzieci. ### selekcja μ, λ Konkurują ze sobą tylko dzieci, więc mamy mieć $\lambda > \mu$. Metoda dużo mniej zachowawcza - może gubić istniejące dobre rozwiązania.

Problem wielowymiarowy

- Różne sigma w każdą stronę
- Jedna sigma we wszystkie strony + reguła anizotropii.
- reguła jednej piątej zmienia moduł wektora sigma
- dodatkowa reguła obracająca wektor sigma o kąt: $\theta_{t+1} = \theta_t$ + gaussowska zmienna losowa
- czyli generalnie po prostu zrobić w biegunowych

Operator krzyżowania dla strategii ewolucyjnej: trochę nasza doróbka

procentowe wkłady poszczególnych rodziców #noniemów I to samo dla sigm