Domača naloga - 2.del

Verjetnostne gramatike in posodabljanje verjetnosti

Vito Rozman

7. junij 2023

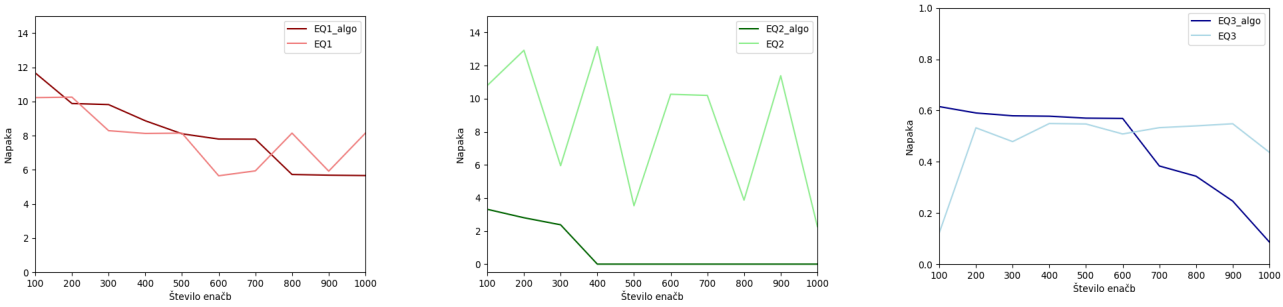
1 Psevdo koda algoritma

Spodaj je psevdo koda algoritma za verjetnostne gramatike. Najprej je potrebno ustavriti objekt **eqAlgo**, ki spejme levo in desno stran spemenljivk v enačbi, gramatiko, podatke in število generiranja enačb ob vsaki iteraciji. Potem za zagon algoritma definitamo objektno funkcijo **model**, kjer podamo število iteracij (*Iter*), željeno napako (*loss*) in parameter posodabljanja verjetnosti (Δ).

Data: $Iter$, $loss$, Δ
Result: Napaka enačbe, Dobljena enačba
for $i = 0, \ldots, Iter$ **do**
  (NAPAKA$_i$, ENAČBA$_i$) = **GenerirajEnačbo()**
  if $NAPAKA_i < loss$ **then**
    Vrni: NAPAKA$_i$, ENAČBA$_i$
  end
  else if $NAPAKA_i < NAPAKA_{i-1}$ **then**
    PosodobiVerjetnosti(ENAČBA$_i$, Δ)
  end
  else if $NAPAKA_i \geq NAPAKA_{i-1}$ **then**
    PosodobiVerjetnosti(ENAČBA$_{i-1}$, Δ)
  end
  else if $(NAPAKA_i - NAPAKA_0 \geq NAPAKA_0 - BestNAPAKA)$ *or* $(i \in [20, 40, 60, ...])$ **then**
    PosodobiVerjetnosti(BestENAČBA, Δ)
  end
  Zapomni si vse porebne spremenljivke za naslednje korake
end
Vrni: BestNAPAKA, BestENAČBA

2 Rezultati

Na sliki 1 so rezutati primerjave razvitega verjetnostnega algoritma in algoritma, ki vzorči enačbe nalključno. Pri enačbi 1b se je algoritem najbolje izkazal, ker je enačbo našel že v 400 iteracijah. Pri enačbi 1a se model ni tako dobro iskazal, kar bi morda lahko pomenilo da gramatika ni tako dobra za dano enačbo. V primeru 1c pa vidimo da bi algoritem morda našel enačbo z višanjem iteracij.



(a) $y = x_1 - 3x_2 - x_3 - x_5$ (b) $y = x_1^5 x_2^3$ (c) $y = \sin(x_1) + \sin(\frac{x_2}{x_1^2})$

Slika 1: Primerjave delovanja verjetnostnega algoritma

3 Komentarji in možne izboljšave

Algoritem sem napisal tako da lahko sprejme poljubno gramatiko. V ta namen sem uvedel nov objekt v obliki slovarja {spremenljivka : {pravilo : verjetnost pravila, ...},...}, ki mi je omogočil dokaj hitro posodabljanje verjetnosti. Za posodabljanje verjetnosti sem testiral dva pristopa softmax in fullfill s predpisoma:

$$\text{Softmax: } p_i = \frac{\exp^{p_i \pm \Delta}}{\sum_j \exp^{p_j \pm \Delta}}, \quad \text{Fulfill: } p_i = \frac{(p_i \pm \Delta)_+}{\sum_j (p_j \pm \Delta)_+}.$$

Softmax se ni iskazal za dobro ker so vhodne verjetnosti med 0 in 1, in bi moral paramater Δ izbirati bolj pametno, zato sem raje uporabljal fulfill. Paziti sem moral tudi na robne primere, ker če ima gramatika rekurzivna pravila, morajo vseeno obstajati pravila ki niso rekurzivna s pozitivno verjetnostjo, ker drugače se model zacikla.

Ob testeranju sem ugotovil da model ne deluje presenetljivo hitreje od naključnega vzorčenja, kar je morda posledica večanja števila enačb ob robnih izidih (vidno v kodi). Druga izpopolnitev bi bila lahko, da glede na dano napako posodablja verjetnosti in ne s fiksnim korakom Δ. Kot zadnje, če se napaka modela oddaljuje od začetne napake (in tudi vakih 10 iteracij), postavim model v njegovo najboljše stanje do tega trenutka, v tem primeru bi lahko drugače posodobil verjetnosti (morda za manjšo vrednost).

Dodatni napotki ob zagonu algoritma

Algoritem poženemo kot je opisenao zgoraj. V skripti **algo.py** se nahaja rared z implementacijo algoritma. Na dnu kode je zakomentiran primer enega zagona algoritma. Uporabljena knjižnjica je **ProGED** (verzija 0.8.5), ki jo v času razvijanja algoritma nisem posodobil!