

PV021 Neuronové siete

Záverečná správa

Adam Krupička, Matej Troják

Problém šachového ťahu

V našom projekte sme chceli riešiť problém pohybu šachových figúrok po šachovnici. Napadlo nás niekoľko prístupov, ako tento problém riešiť:

1. vstupom je celá šachovnica a výstupom opäť šachovnica, avšak pozmenená o jeden validný ťah,
2. vstupom sú pozície jednotlivých figúrok a výstupom pozmenené pozície figúrok,
3. vstupom je celá šachovnica/pozície figúrok a výstupom ťah, ktorý sa má vykonať, t.j. súradnice políčka s *nejakou* figúrkou a súradnice, kam sa má posunúť,
4. vstupom sú dve šachovnice a sieť odpovie hodnotou medzi 0 a 1 ako *dobrý* je ťah (inšpirované iným zdrojom, neriešené)

Zdroj dát

Použili sme záznamy šachových partií z Games of World Champions¹. Na dáta sme využili voľne dostupný parser, ktorý ich prekonvertoval do formátu FEN (Forsyth–Edwards Notation), čo je štandardná notácia pre popis priebehu šachovej partie tak, aby mohla byť opätovne zrekonštruovaná.

Implementácia

Implementovali sme viacvrstvú sieť (multilayer perceptron) a spätnú propagáciu (backpropagation algorithm) v jazyku C++. Implementovali sme triedy pre neurón, sieť a šachovnicu. Urobili sme serializáciu siete tak, aby sme po ukončení učenia s ňou mohli znova pracovať. Celý program má 3 rôzne funkcie:

¹<http://www.chess.com/download/view/games-of-world-champions>

- **new** - vytvorenie novej siete, parameter je "počet neurónov vo vrstvách" a názov siete
`./main new '3 81 2' my.net`
- **learn** - spustí učenie pomocou spätnej propagácie nad daným vstupným súborom dát a sieťou
`./main learn input.txt my.net`
- **eval** - vyhodnotí sieť nad daným vstupom pre danú sieť
`./main eval '8 5 1 5 8 4 1 4 8 3 8 6 1 3 ... 2 2 3 2 4 2 5 2 6 2 7 2 8' my.net`

Prístup 1

Vstup tvorený FENmi sme jednoduchým skriptom previedli do sekvencie 64 číslíc v rozsahu hodnôt $[-6, 6]$. Figúrky majú nasledujúcu číselnú reprezentáciu (jedná sa o biele figúrky, čierne majú opačné znamienko):

pešiak	1	strelec	4
veža	2	kráľ	5
kôň	3	kráľovná	6

a prázdne políčko má hodnotu 0.

Riešenie

Nakoľko sme potrebovali rozsah hodnôt $[-6, 6]$, ako aktivačnú funkciu sme zvolili hyperbolický tangens s upravenými koeficientmi tak, aby bol na danom intervale približne lineárny.

$$F_a(x) = 12 \times \tanh\left(\frac{x}{12}\right) \quad (1)$$

Trénovacie dáta boli dvojice sekvencií 64 číslíc. Ťah je reprezentovaný tak, že na políčko, z ktorého sa figúrka hýbe, sa priradí číslo nula a políčko, na ktoré sa figúrka hýbe, dostane pôvodnú hodnotu figúrky. Nakoľko v trénovacích dátach boli ťahy ako pre bieleho, tak i čierneho hráča, rozhodli sme sa používať iba biele ťahy. Tým sme dáta zmenšili o jeden rozmer (každý ťah čierneho hráča sa dá previesť na ťah bieleho hráča).

Prístup 2

Vstup tvorený FENmi sme iným skriptom previedli do sekvencie 64 čísl v rozsahu hodnôt $[0, 8]$. Čísla sú vnímané ako dvojice, kde každá dvojica predstavuje súradnice konkrétnej figúrky na šachovnici. Poradie figúrok je teda fixne dané. Ak sa figúrka na šachovnici nenachádza, dostane súradnice $(0, 0)$.

Riešenie

Pre potrebný rozsah hodnôt sme zvolili funkciu:

$$F_a(x) = \begin{cases} 0 & x \leq 0 \\ x & x > 0 \end{cases}$$

Ťah je reprezentovaný zmenou súradníc jednej figúrky. V prípade, ak sa jedná o ťah, kedy súper stráca figúrku (t.j. figúrka sa presúva na súradnice už zahrnuté v sekvencii súradníc), súradnice vyhodenej figúrky sú nastavené na hodnoty $(0, 0)$.

Prístup 3

Tento prístup je veľmi podobný prístupu číslo 2, rozdiel je iba vo výstupe siete. Namiesto toho, aby bol ťah zahrnutý v rozložení novej šachovnice, je výstupom explicitne daný ťah, t.j. súradnice políčka, z ktorého sa má figúrka pohnúť a súradnice **kam** sa má figúrka pohnúť.

Výsledky

Problémom u všetkých typov prístupov bolo nájsť parametre siete tak, aby sa daný problém naučila vyhodnocovať. Skúšali sme mnoho kombinácii topológie siete a rýchlosti učenia, no vždy neúspešne. Pri prístupe 3, kedy funkčnosť siete vyzerala nádejne (nastavenie: 64, 128, 64, 16, 4), bol zase problém z výpočtetného hľadiska. Odhad učenia na našich dátach bol asi tri dni a nemali sme prostriedky ako otestovať, či je sieť schopná sa tomuto prístupu naučiť.

Preto sme rozhodli riešiť pomocou neurónových sietí iný problém.

Problém derivácie polynómu

Ideou je, že sa sieti na vstup dá polynóm a ona vráti jeho deriváciu.

Príklad

Majme vstupný polynóm:

$$f(x) = 5x^3 + 2x + 8 \quad (2)$$

Takýto polynóm bude interpretovaný pre sieť nasledovne (uvažujme, že sieť je schopná načítavať polynómy stupňa maximálne 5):

$$f(x) = 8x^0 + 2x^1 + 0x^2 + 5x^3 + 0x^4 + 0x^5 \quad (3)$$

takže finálnym vstupom pre sieť je sekvencia čísel "8 2 0 5 0 0".

Chceme dostať deriváciu tohto polynómu

$$f'(x) = 2 + 15x^2 \quad (4)$$

čiže sekvenciu čísel "2 0 15 0 0", ktorá je o jednu číslicu kratšia.

Zdroj dát

Ako zdroj dát sme zvolili funkciu, ktorá vygeneruje náhodný polynóm z daného rozsahu hodnôt (maximálny stupeň polynómu a rozsah koeficientov) a zároveň vypočíta jeho deriváciu.

Implementácia

Implementácia čo sa siete týka sa nezmenila. Pribudla trieda Polynoms, ktorá pracuje s polynómami. Zásadnou je funkcia, ktorá generuje dáta pre sieť, t.j. dvojice vstup a výstup. Pre rozhranie to má dopad na nasledujúcu funkcionálnosť:

- **learn** - spustí učenie pomocou spätnej propagácie nad daným počtom vstupných polynómov a sieťou
./main learn 10000 my.net
- **eval** - vyhodnotí sieť nad daným vstupom pre danú sieť
./main eval '2 1 3' my.net

Riešenie

Nakoľko rozsah koeficientov, ktoré môžu byť v polynóme použité je z všeobecného hľadiska neobmedzený, rozhodli sme sa použiť lineárnu funkciu:

$$F_a(x) = x, \quad (5)$$

ktorá má v každom bode deriváciu rovnú 1.

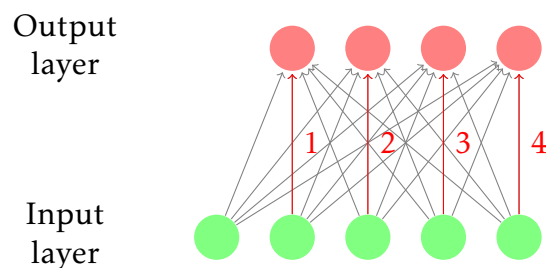
Výsledky

Prvým pokusom bolo realizovať derivovanie pomocou neurónovej siete na veľmi obmedzených polynómoch. Zvolili sme maximálny stupeň polynómu 3 a rozsah koeficientov ako $[0, 3]$. Topológiu siete sme zvolili "3 81 2". Problém bol v rýchlosti učenia, ktorú sme museli nastaviť na veľmi malú hodnotu, aby sa sieť správne učila – 0.0001.

Následne po dostatočnom učení (asi 20000 náhodne vygenerovaných polynómov) sme dosiahli pozitívne výsledky vo výstupoch siete. Tu je niekoľko príkladov:

- vstup: $2 + 1x^1 + 3x^2$
derivácia: $1.05352 + 5.89288x^1$
- vstup: $2 + 1x^1$
derivácia: $1.00556 + 0.129582x^1$
- vstup: $3 + 3x^1 + 3x^2$
derivácia: $2.93755 + 5.82714x^1$

Druhým výsledkom je iný prístup k učeniu. Namiesto toho, aby sme sieť učili konkrétne polynómy, dávali sme jej na vstup vždy iba jeden člen s nenulovým koeficientom. Tým sme chceli dosiahnuť *nezávislosť* jednotlivých členov medzi sebou. Zvolili sme maximálny stupeň polynómu 4 a rozsah koeficientov ako $[0, 10]$. To znamená, že dokopy sme sieť učili iba sadou dát, kde sa vyskytovalo dokopy 55 rôznych polynómov. Zvolili sme minimálnu topológiu siete, čiže "5 4". Ideálna sieť by vyzerala takto:



kde ostatné hrany majú váhu 0.

Preto sme sa pokúsili tento ideál dosiahnuť. Naša sieť si po učiacom procese nastavila na hranách takmer identické váhy, preto aj vracala celkom presné výsledky. Vďaka tomu sa naša sieť naučila pracovať s ľubovoľnými koeficientmi bez toho, aby sme ju to museli explicitne učiť. Nemá problém ani s desatinnými a zápornými číslami. Príklady:

- vstup: $20.5x^1 + 11.3x^2 + 2.5x^3 + -0.5x^4$
derivácia: $20.48 + 22.5747x^1 + 7.46458x^2 + -2.04699x^3$
- vstup: $20.5x^1 + 11.3x^2 + 20x^3 + -500x^4$
derivácia: $20.7582 + 23.0364x^1 + 60.6112x^2 + -1999.19x^3$

Rozdelenie práce

- Implementácia neurónovej siete → Adam Krupička
- Spracovanie vstupu, záverečná správa → Matej Troják
- Kreatívna činnosť a iné → spoločne