# 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. dať sieti na vstup celú šachovnicu a na výstupe očakávať opäť šachovnicu, avšak pozmenenú o jeden validný ťah,
2. dať sieti na vstup pozície jednotlivých figúrok a na výstupe očakávať pozmenené pozície figúrok,
3. dať sieti na vstup celú šachovnicu/pozície figúrok a očakávať na výstupe ťah, ktorý sa má vykonať, t.j. súradnice políčka s *nejakou* figúrkou a súradnice, kam sa má posunúť,
4. dať sieti na vstup 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 Champions[1]. Na dáta sme využili voľne dostupný parser, ktorý ich prekonvertoval do formátu FEN (Forsyth–Edwards Notation), čo je štandartná notácia pre popis priebehu šachovej partie tak, aby mohla byť opätovne zrekonštruovaná.

### Implementácia

Implementovali sme viacvrstvú sieť (multilayer perceptron) a zpätnú 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 aj s odstupom času. 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 scriptom previedli do sekvencie 64 číslic 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 |

prázdne políčko má hodnotu 0.

**Riešenie**

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

$$F_a(x) = 12 \times tanh(\frac{x}{12}) \tag{1}$$

Trénovacie dáta boli dvojice sekvencií 64 číslic. 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 jenen rozmer (každý ťah čierneho hráča sa dá previesť na ťah bieleho hráča).

## Prístup 2

Vstup tvorený FENmi sme iným scriptom previedli do sekvencie 64 číslic 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 nepriateľ stráca figúrku (t.j. figúrka sa presúva na súradnice už zahrnuté v sekvencii súradníc), súradnice vyhodenej figúrky sú (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 daný problém riešila. 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 výpočetného hľadiska. Odhad učenia bol asi tri dni a žiadny z nám prístupnych výpočetných serverov (Aisa, Aura) nepodporuje C++11 (Aisa zvláda iba c++0A.D. a Aura c++500BC). Preto sme nemali ako otestovať, či je sieť schopná sa tomuto prístupu naučiť.

Preto sme rozhodli riešiť pomocou neurónových sieti iný problém.

## Problém derivácie polynómu

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

### Príklad

Majme vstupný polynóm:

$$f(x) = 5x^3 + 2x + 8 \tag{2}$$

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

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

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

Chceme dostať deriváciu tohto polynómu

$$f'(x) = 15x^2 + 2 \tag{4}$$

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

### Zdroj dát

Ako zdroj dát sme zvolili funkciu, ktorá vygeneruje náhodný polynón 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 polynómmi. Zásadnou je funkcia, ktorá generuje dáta pre sieť, t.j. dvojice vstup a výstup. Pre rozhranie to má dopad iba na volanie učenia:

- **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:

$$F_a(x) = x\,, \tag{5}$$

ktorá má v každom bode deriváciu rovnú 1.

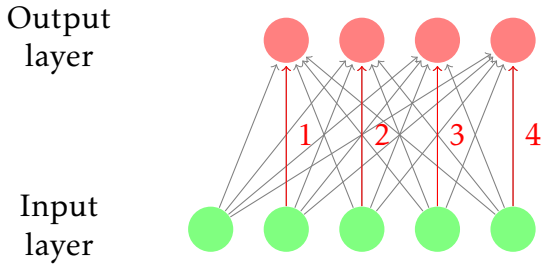
## Výsledky

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 roznych polynómov. Najprv sme zvolili topológiu siete "5 125 4", no postupne sme sa prepracovali až na minimálnu topológiu. Uvedomili sme si, že stačí sieť s topológiou "5 4":




kde ostatné hrany majú váhu 0.

Preto sme sa pokúsili tento ideál dosiahnuť. Naša sieť si po učiacom pro učiacom procese nastavila na hranách takmer identické váhy, preto aj vracala celkom presné výsledky. Vďaka tomu sa naša sieť naučila pracovať s ľubovoľnými koeficientami 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: $20.485 + 22.5747x^1 + 7.46458x^2 + -2.04699x^3$
- vstup: $20.5x^1 + 11.3x^2 + 20x^3 + -500x^4$
  derivácia: $20.7582 + 23.0364x^1 + 60.6112x^2 + -1999.19x^3$