Jozef Waldhauser

Úloha 6

Evolučné učenie neurónovej siete

Obsah

[Úloha 6 – zadanie 2](#_Toc197277058)

[Úloha 6 – zdroje 2](#_Toc197277059)

[Teória 2](#_Toc197277060)

[Design algoritmu 3](#_Toc197277061)

[Implementácia 4](#_Toc197277062)

[Vstupy 4](#_Toc197277063)

[Implementácia senzorickej funkcie 5](#_Toc197277064)

[Implementácia nn\_function 6](#_Toc197277065)

[Implementácia nn\_navigate\_me 6](#_Toc197277066)

[Implementácia handle\_mes\_fitnesses 7](#_Toc197277067)

[Analýza 7](#_Toc197277068)

# Úloha 6 – zadanie

1. naimplementujte vlastní senzorické funkce, kterými budou agenti vnímat prosředí
2. navrhněte funkcinn\_function(inp, wei), která na základě vstupů inp a vektoru vah wei neuronové sítě (genomu neuronové sítě / zlinearizované sítě) provede výpočet výstupů této sítě
3. navrhněte funkci nn\_navigate\_me(me, inp), která pro agenta me a jeho senzorické vstupy inp provede výpočet jeho pohybu na základě jeho vnitřního genomu reprezentujícího jeho neuronovou síť (zda pojede up, down, left, right)
4. doimplementujte mechanismus výpočtu fitness jedinců ve funkci handle\_mes\_fitnesses(mes), která dostane seznam jedinců mes a vypočítá jim fitness na základě jednoho herního kola
5. nastavte vhodně parametry evolučního algoritmu (mutace, crossover, selekce)

**Úlohou bolo vytvoriť navigačný systém s neurónovou sieťou pomocou evolučných algoritmov. Má obsahovať senzorické funkcie, neurónovú sieť (nn\_function) navigačný mechanizmus (nn\_navigate\_me) a systém výpočtu fitness hodnoty pre jedinca.**

# Úloha 6 – zdroje

<https://www.youtube.com/watch?v=aircAruvnKk> - But what is a neural network?

<https://www.youtube.com/watch?v=jmmW0F0biz0> – Neural networks explained in 1min

<https://www.youtube.com/watch?v=rEDzUT3ymw4> – Neural networks explained in 5min

<https://www.youtube.com/watch?v=UZDiGooFs54> – The moment we stopped understanding AI

<https://en.wikipedia.org/wiki/Taxicab_geometry> - manhattan distance (posun 2D matrix)

# Teória

V biológií neurón prijíma signály z viacerých vstupov (dendrity/synapsy) a ak je nad určitou hranicou „thresholdom“ tak potom odošle signál cez axon.

V programovaní sa prakticky snažíme o to isté ako vstupy ovplyvnia „thresholdy“ dokážeme definovať váhami.

kde , ... sú vstupy a sú váhy, ***b*** je bias tzv. posun.

Do genómu si môžeme uložiť prakticky všetky tieto hodnoty a rozvíjať ich algoritmom.

Veľkosť genómu dokážeme vypočítať ako:

počet vstupov do neurónov \* počet všetkých neurónov **(váhy vstupov win)** +  
počet všetkých neurónov **(biasy vstupov bin)** +  
počet všetkých neurónov \* počet výstupov **(váhy výstupov wout)** +  
počet výstupov **(biasy výstupov bout)**

*(zátvorka vyjadruje čo tým vypočítame)*

Genóm bude mať tvar podobný tomuto:  
[ input\_weight1, input\_bias1, input\_weightx ... output\_weight1, output\_bias1, weightx...]

|  |
| --- |
| GENOME\_SIZE = (NUM\_INPUTS \* NUM\_NEURONS + NUM\_NEURONS + NUM\_NEURONS \* NUM\_OUTPUTS + NUM\_OUTPUTS) |

Následne môžem okolo tohto rozvíjať model neurónovej siete resp. celý algoritmus:

# Design algoritmu

* **Vstupy**:
  + Sieť dostane 5 čísel (senzorické údaje, napr. vzdialenosť k cieľu).
  + Predstavujú informácie o svete, kde sa agent pohybuje.
* **Skrytá vrstva**:
  + Má 10 neurónov (ktoré počítajú)
  + Každý neurón vezme vstupy, vynásobí ich váhami (číslami určujúcimi vplyv), pridá bias (posun) a **použije ReLU (rectified linear unit je naša aktivačná funkcia).**
  + Výsledkom je 10 nových čísel.
* **Výstupná vrstva**:
  + Má 4 neuróny (reprezentujú pohyby: hore, dole, vľavo, vpravo).
  + Berie 10 čísel zo skrytej vrstvy, vynásobí ich váhami, pridá biasy a použije **softmax** **(premení hodnoty na pravdepodobnosti, ktoré sa sčítajú na 1).**
  + Výsledkom sú 4 pravdepodobnosti (napr. 60 % hore, 20 % dole, atď.).
* **Rozhodnutie**:
  + **Sieť vyberie pohyb s najvyššou pravdepodobnosťou** (napr. "hore", ak má 60 %).
* **Fungovanie**:
  + Vstupy prejdú skrytou vrstvou, potom výstupnou vrstvou, a sieť urobí rozhodnutie (pohyb).
  + Evolučný algoritmus evolvuje vyučené neural networks (ďalej nn).

# Implementácia

Zadaním sme mali implementovať 3 funkcie:

1. Senzorickú funkciu (v mojom prípade sense\_environment(position, target))
2. nn\_function(inp, wei) – vypočíta **výstupy** nn
3. nn\_navigate\_me(me, inp) – na základe genomu **vypočíta kde sa má pohnúť**
4. handle\_mes\_fitnesses(mes) - vypočíta **fintess jedincov mes**

## Vstupy

|  |
| --- |
| WORLD\_SIZE = 10         # velkost stvorcovej mriezky sveta (10x10)  TARGET\_POSITION = [9, 9]   # cielova pozicia je v pravom dolnom rohu  MAX\_STEPS = 50            # maximalny pocet krokov povolenych na jedno kolo  NUM\_INPUTS = 5          # 5 senzorovych vstupov (4 smery k cielu + 1 rozsirena funkcia)  NUM\_NEURONS = 10        # 10 neuronov v skrytej vrstve  NUM\_OUTPUTS = 4         # 4 vystupy (hore, dole, vlavo, vpravo)  GENOME\_SIZE = (NUM\_INPUTS \* NUM\_NEURONS + NUM\_NEURONS + NUM\_NEURONS \* NUM\_OUTPUTS + NUM\_OUTPUTS)  POPULATION\_SIZE = 500     # pocet jedincov v populacii  MUTATION\_PROB = 0.2       # pravdepodobnost mutacie jedinca  GENE\_MUTATION\_PROB = 0.1  # pravdepodobnost mutacie kazdeho genu  CROSSOVER\_PROB = 0.7      # pravdepodobnost krizenia dvoch jedincov  TOURNAMENT\_SIZE = 5       # velkost turnajoveho vyberu  NUM\_ELITE\_INDIVIDUALS = 20  # pocet najlepsich jedincov, ktori sa zachovaju nezmeneni  MAX\_GENERATIONS = 200     # maximalny pocet generacii evolucie  NUM\_GAME\_ROUNDS = 3       # pocet hernych kol na vyhodnotenie jedinca |

Pre pochopenie nasledujúcich implementácií opíšem prostredie, v ktorom sa náš agent pohybuje: svet je štvorcová mriežka (10x10) a cieľom je dostať sa na políčko [9,9] v pravom hornom rohu. Piaty vstup je tzv. „manhattan distance“ čo opíšem pri implementácií senzorickej funkcie.

## Implementácia senzorickej funkcie

|  |
| --- |
| def sense\_environment(position, target):      x, y = position      target\_x, target\_y = target      distance\_up = max(0, target\_y - y) / WORLD\_SIZE      distance\_down = max(0, y - target\_y) / WORLD\_SIZE      distance\_left = max(0, x - target\_x) / WORLD\_SIZE      distance\_right = max(0, target\_x - x) / WORLD\_SIZE      # normalizovana manhattan vzdialenost k cielu (blizsie = vyssia hodnota)      manhattan\_distance = (abs(x - target\_x) + abs(y - target\_y)) / (2 \* WORLD\_SIZE)      target\_proximity = 1 - manhattan\_distance  # inverzia: blizsie = vyssia hodnota      return [          1 - distance\_up,    # blizsie = vyssia hodnota          1 - distance\_down,          1 - distance\_left,          1 - distance\_right,          target\_proximity      ] |

Táto funkcia reprezentuje „oči“ agenta, hovorí mu kde sa nachádza a kde je cieľ.

target\_proximity reprezentuje **manhattan distance:**

Euklidovská vzdialenosť = vzdušná čiara, **manhattan distance = ideme po uličkách.**

Je to normalizované číslo aby bol výsledok medzi 0 a 1. Prakticky čím vyššie je číslo, tým bližšie je k cieľu.

V praxi to môže byť napríklad robot, ktorý hľadá svoju nabíjaciu stanicu vybavenú GPS.

## Implementácia nn\_function

|  |
| --- |
| def nn\_function(inp, wei):      input\_weights = np.array(wei[:NUM\_INPUTS \* NUM\_NEURONS])      input\_weights = input\_weights.reshape(NUM\_INPUTS, NUM\_NEURONS)        biases = np.array(wei[NUM\_INPUTS \* NUM\_NEURONS:                                   NUM\_INPUTS \* NUM\_NEURONS + NUM\_NEURONS])        output\_weights = np.array(wei[NUM\_INPUTS \* NUM\_NEURONS + NUM\_NEURONS:-NUM\_OUTPUTS])      output\_weights = output\_weights.reshape(NUM\_NEURONS, NUM\_OUTPUTS)        output\_biases = np.array(wei[-NUM\_OUTPUTS:])      layer = np.dot(inp, input\_weights) + biases      layer = np.maximum(0, layer)  # relu aktivacia      output = np.dot(layer, output\_weights) + output\_biases      exp\_output = np.exp(output - np.max(output)) # softmax      probabilities = exp\_output / exp\_output.sum()      return probabilities |

Táto funkcia berie **inp** vstupy - a **wei** váhy – genóm. Dá sa povedať že toto je najdôležitejšia funkcia, keďže robí výpočty, kde sa má agent pohnúť.

Vstupy: a výstupy: . Celý reťazec:

1. **Vstupná vrstva:**

Môže byť aj viac vstupov, ale pre jednoduchosť nechám 1: reprezentuje vzdialenosť od cieľa (v našom prípade musia byť minimálne 4). Toto určuje ako vstupy ovplyvňujú aktivitu každého neurónu

1. **ReLU aktivácia:**

Výstupy sú aktivované cez ReLU: – **ak je výstup kladný nezmení sa a ak je záporný alebo 0 tak sa nahradí nulou.**

1. **Výstupná vrstva:**

Sem sa aplikujú váhy a biasy výstupnej vrstvy na ReLU aktivované vstupy.

1. **Softmax: =**

Vypočíta pravdepodobnosti pohybu do určitého smeru.

## Implementácia nn\_navigate\_me

|  |
| --- |
| def nn\_navigate\_me(me, inp):      # funkcia, ktora pre agenta me a jeho senzoricke vstupy inp vykona vypocet      # jeho pohybu na zaklade jeho vnutorneho genomu      action\_probabilities = nn\_function(inp, me)      return np.argmax(action\_probabilities) |

Rozhoduje kde sa agent pohne na základe maximálnej šance (np.argmax).

## Implementácia handle\_mes\_fitnesses

|  |
| --- |
| def handle\_mes\_fitnesses(mes):      results = []        for i, individual in enumerate(mes):          start\_position = [random.randint(0, WORLD\_SIZE-1), random.randint(0, WORLD\_SIZE-1)]          while start\_position == TARGET\_POSITION:              start\_position = [random.randint(0, WORLD\_SIZE-1), random.randint(0, WORLD\_SIZE-1)]          score, steps, target\_reached = simulate\_game\_round(individual, start\_position)            print(f"Jedinec {i+1}: Skóre={score:.2f}, Kroky={steps}, Cieľ dosiahnutý={target\_reached}")            results.append((individual, score))        print("=" \* 50)      print("Vystup handle\_mes\_fitnesses (fitnessy najlepsich jedincov)")      print("=" \* 50)      print(f"\nPriemer fitness: {np.mean([r[1] for r in results]):.2f}")      print(f"Najlepšia fitness: {np.max([r[1] for r in results]):.2f}")      print(f"Najhoršia fitness: {np.min([r[1] for r in results]):.2f}")      print("=" \* 50)        return results |

# A graph with a line graph AI-generated content may be incorrect.Experimenty/Analýza

Obrázok 1 max fitness graf pri 200 generáciách

## Prvé vstupné hodnoty

POPULATION\_SIZE = 500

MUTATION\_PROB = 0.2

GENE\_MUTATION\_PROB = 0.1

CROSSOVER\_PROB = 0.7

TOURNAMENT\_SIZE = 5

NUM\_ELITE\_INDIVIDUALS = 20

MAX\_GENERATIONS = 200

NUM\_GAME\_ROUNDS = 3

Výsledkom bolo, že fitness dosiahol hodnotu ~208 a agent sa vedel správne navigovať. **Okolo 30. generácie sa výsledky začali stabilizovať**, preto som sa rozhodol otestovať aj tento počet generácií. Výsledný fitness bol stále dobrý, avšak **agent sa nevedel dostať správne do cieľa.** Hodnoty zatiaľ nechám nezmenené a zistím, pri akej generácii agent dosiahne úspech.

|  |  |  |  |
| --- | --- | --- | --- |
| Počet generácií | Max Fitness | Avg. Fitness | Dostal sa do cieľa? |
| 30 | 149 | 129.75 | Nie (0/3) |
| 50 | 145 | 133.55 | Áno (3/3) |
| 100 | 147 | 128 | Áno (3/3) |
| 200 | 148 | 139.7 | Áno (3/3) |
| 500 | 148 | 141.5 | Áno (3/3) |

A graph with a line

AI-generated content may be incorrect.Našiel som chybu pri zobrazovaní svojho grafu – maximálna a nie priemerná fitness.

A graph with a line

AI-generated content may be incorrect.Obrázok 3 pokus pri 30 generáciách

Obrázok 2 zmena z maximálnej na priemernú fitness

A graph with a line

AI-generated content may be incorrect.

Obrázok 4 - avg. fitness počas 100 generácií

A graph with lines and dots

AI-generated content may be incorrect.

Obrázok 5 - 200 generácií

A graph with blue lines

AI-generated content may be incorrect.

Obrázok 6 - Vývoj priemernej fitness počas 500 generácií

## Experimentovanie s ostatnými parametrami

### Vstupné parametre:

POPULATION\_SIZE = 200     # pocet jedincov v populacii

MUTATION\_PROB = 0.2       # pravdepodobnost mutacie jedinca

GENE\_MUTATION\_PROB = 0.1  # pravdepodobnost mutacie kazdeho genu

CROSSOVER\_PROB = 0.7      # pravdepodobnost krizenia dvoch jedincov

TOURNAMENT\_SIZE = 5       # velkost turnajoveho vyberu

NUM\_ELITE\_INDIVIDUALS = 20  # pocet najlepsich jedincov, ktori sa zachovaju nezmeneni

MAX\_GENERATIONS = 50     # maximalny pocet generacii evolucie

NUM\_GAME\_ROUNDS = 3       # pocet hernych kol na vyhodnotenie jedinca

|  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- |
| POP SIZE | MUTAT | GENE MUT | CROSSOVR | TRN SIZE | ELITES | Avg. Fitness | Dosiahnutý cieľ (3 kolá) | Pozorovaná zmena |
| 200 | 0.2 | 0.1 | 0.7 | 5 | 20 | 135.4 | 3/3 | očakávaná funkcia |
| 20 | 0.2 | 0.1 | 0.7 | 5 | 20 | 19.20 | 0/3 | Nefunkčné |
| 20 | 0.2 | 0.1 | 0.7 | 5 | 5 | 38.00 | 0/3 | 2x fitness predošl. behu, nefunkčne |
| 20 | 0.2 | 0.1 | 0.7 | 5 | 1 | 138.00 | 3/3 | funguje každý 2. beh |
| 20 | 0.2 | 0.1 | 0.8 | 5 | 1 | 140.00 | 3/3 | fung. 4/5 behov |
| 20 | 0.2 | 0.1 | 0.9 | 5 | 1 | 134.00 | 3/3 | fung. 3/5 behov |
| 200 | 0.7 | 0.1 | 0.7 | 5 | 20 | 128.30 | 3/3 | očakávaná funkcia |
| 200 | 0.1 | 0.1 | 0.7 | 50 | 20 | 141.85 | 3/3 | sporadicky nefunguje |

Malá populácia (20 jedincov) je vo väčšine prípadov nefunkčná, no ak sa ponechá iba 1 elitný jedinec a zvýši sa crossover, môže mať prekvapivo dobré výsledky – až v 4 z 5 behov sa agenti dostali do cieľa.

Veľká populácia (200) s klasickými nastaveniami fungovala konzistentne dobre.

V prípade nastavení s vysokou mutáciou (0.7) boli výsledky stále stabilné, aj keď mierne menej presné.

Zaujímavé správanie sa prejavilo, keď agent začínal na rovnakej vertikálnej úrovni ako cieľ – niektoré simulácie boli úspešné len kvôli tomu, že agent išiel priamo k cieľu bez potreby navigácie, čo mohlo skresliť úspešnosť.

# Záver

Pomocou jednoduchej neurónovej siete a evolučného **algoritmu je možné efektívne trénovať agenta na navigáciu v prostredí.** Výsledky potvrdili, že správne nastavené parametre populácie, mutácie a selekcie výrazne ovplyvňujú úspešnosť učenia. Neurónová sieť sa naučila rozumne rozhodovať na základe senzorických vstupov a dosahovala stabilné výsledky už po 50 generáciách.