4. prednáška

Učenie neurónových sietí pomocou učenia s odmenou a trestom (reinforcement learning)

Historické poznámky o učení s odmenou a trestom

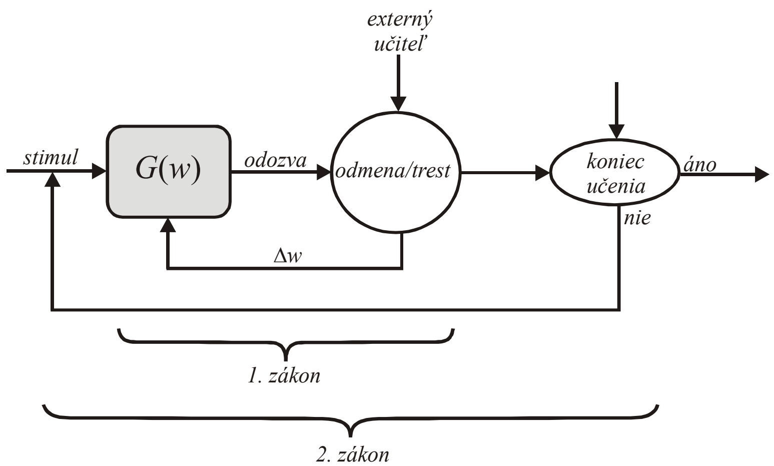
- Metafora neurónových sietí umožňuje použiť učenie s odmenou a trestom počítačovej inteligencii.
- Agent nie je hodnotený po každom elementárnom kroku, ale až na záver riešenia. Ak sa mu podarilo nájsť východ z bludiska, potom je *odmenený* (výška odmeny je nepriamo úmerná počtu krokov, ktoré boli na to potrebné), v opačnom prípade, ak v bludisku zablúdil je *potrestaný*.
- Základy tohto učenia naformuloval počiatkom minulého storočia americký psychológ Edward Thorndike prostredníctvom dvoch zákonov:

Zákony učenia s odmenou a trestom podľa E. Thorndikea

Edward Thorndike (1874-1949)

1. Zákon účinku: *Ak odozva na opakujúci sa stimul je kladná (odmena), potom väzba medzi stimulom a odozvou sa postupne zosilňuje. V opačnom prípade, ak odozva je záporná (trest), potom väzba medzi stimulom a odozvou postupne zaniká.*
2. Zákon opakovaného používania: *Požadované správanie je výsledkom častého používania dvojica stimul a odozva*

Schématické znázornnenie učenia s odmenou a trestom



Význam učenia s odmenou a trestom pre neurónové siete

- Učenie s odmenou a trestom poskytuje neurónovým sieťam nové možnosti aplikácie, menovite ich použitie ako kognitívneho orgánu agentov, ktorý sú hodnotený až na záver svojich aktivít, podľa toho, či dosiahli alebo nedosiahli stanovený cieľ.
- Pri tomto učení nie je potrebné používať externého učiteľa, ktorý klasifikuje každý elementárny pohyb agenta (t. j. vytvára tréningovú množinu), externý učiteľ hodnotí len záver činnosti agenta, či dosiahol alebo nedosiahol svoj cieľ, a ak ho dosiahol, potom hodnotí aj kvalitu získaného cieľového riešenia (napr. môžu byť preferované také ciele, ktoré vyžadujú menší počet krokov).
- Aktuálny kognitívny orgán sa v priebehu predpísaného počtu krokov nechá konštantný a na záver sa hodnotia pohybové schopnosti agenta. V prípade, že sa dobre pohyboval, potom je odmenený a jeho nastavenie parametrov w jeho kognitívneho orgánu sa zmení tak, aby sa zosilnilo správanie agenta, ktoré viedlo k jeho odmene. V opačnom prípade, ak sa agent nepohybuje, vykonáva nekoordinované pohyby, potom je potrestaný a parametre w sa zmenia tak, aby sa zoslabilo správanie agenta, ktoré viedlo ku nekoordinovaným pohybom.

Introductory notes

Let us consider an agent determined by the following two sets:

(1) A discrete **set of agent states**

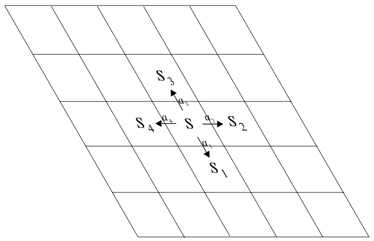
$$S = \{s_1, s_2, s_3, \ldots\}$$

(2) a discrete **set of agent actions**

$$A = \{a_1, a_2, a_3, \ldots\}$$

Agent actions are interpreted as mappings of agents states onto itself

$$s' = a(s)$$



The agent is endowed with a ***cognitive device*** (or **predictor**) that evaluates each agent states by a real number called the ***prediction***

$$P(w): S \to R$$

or explicitly

$$z_i = P(s_i; w)$$

The mapping – predictor is parametric, i.e. it manifests a **plasticity** (with respect to the parameters w)

Assumption: A selection of a respective action $a \in A$ applied to a state $s \in S$ is ***controlled*** by the cognitive device.

Each abstract state $s \in S$ is numerically represented by an n-dimensional real (often binary) vector $\boldsymbol{x}$

$$s \in S \quad \Leftrightarrow \quad \boldsymbol{x} = (x_1, x_2, \ldots, x_n) \in R^n$$

Subject of the learning

Let us have a sequence of agent states and its evaluation

$$s_1, s_2, \ldots, s_m, z$$

where z is an evaluation corresponding to a fact whether the sequence has a required property

$$z = \begin{cases} 1 & (\textit{sequence has the required property}) \\ 0 & (\textit{otherwise}) \end{cases}$$

Assumption. The sequence of states $s_1, s_2, \ldots, s_m$ is constructed **quasirandomly**, i.e. if we have a subsequence $s_1, s_2, \ldots, s_i$(for $1 \leq i < m$), then its enlargement about a next state s_{i+1} is performed according to a prediction $z_i = P(s_i; w)$.

Goal: To modify the agent cognitive device $P(w)$ such that a sequence of predictions

$$P_1 = P(s_1; w), P_2 = P(s_2; w), \ldots, P_m = P(s_m; w)$$

is an estimation of $z = P_{m+1}$

An outline of construction of updating formula

Let us consider a sequence of states that are evaluated by the same required property z

$$(s_1, z), (s_2, z), \ldots, (s_m, z)$$

A quality of prediction is determined by the objective function

$$E(w) = \frac{1}{2} \sum_{t=1}^{m} \left(z - P(s_t; w)\right)^2$$

A steepest-descent recurrent formula for minimization of this objective function looks as follows

$$w := w - \alpha \, grad_w E(w) = w + \Delta w \tag{1a}$$

$$\Delta w = \sum_{t=1}^{m} \alpha (z - P_t) grad_w P_t \tag{1b}$$

Above updating formula (1a-b) can be rewritten in another alternative form

$$\begin{aligned} z - P_t &= P_{m+1} - P_t \\ &= P_{m+1} - P_m + P_m - P_t \\ &= P_{m+1} - P_m + P_m - P_{m-1} + P_{m-1} - P_t \\ &= \ldots\ldots\ldots\ldots\ldots\ldots\ldots\ldots\ldots\ldots \\ &= \sum_{k=t}^{m} (P_{k+1} - P_k) \end{aligned}$$

If we use algebraic identity

$$\sum_{t=1}^{m} \sum_{k=t}^{m} A_{kt} = \sum_{t=1}^{m} \sum_{k=1}^{t} A_{tk}$$

then the updating formula is

$$\Delta w = \sum_{t=1}^{m} \alpha(z - P_t) grad_w P_t$$
$$= \sum_{t=1}^{m} \alpha\left(\sum_{k=t}^{m} (P_{k+1} - P_k)\right) grad_w P_t \qquad (2)$$
$$= \sum_{t=1}^{m} \alpha(P_{t+1} - P_t) \sum_{k=1}^{t} grad_w P_k$$

Summarizing,

$$\Delta w = \sum_{t=1}^{m} \Delta w_t \qquad (3a)$$

$$\Delta w_t = \alpha(P_{t+1} - P_t) \sum_{k=1}^{t} grad_w P_k \qquad (3b)$$

This formula is "generalized" into a form called the TD(λ) family of learning procedures

$$\Delta w_t = \alpha(P_{t+1} - P_t) \sum_{k=1}^{t} \lambda^{t-k} grad_w P_k \qquad (4)$$

where the weighting parameter $0 \le \lambda \le 1$. For $\lambda=1$ formula (4) gives original result (3), while $\lambda=0$ it gives

$$\Delta w_t = \alpha(P_{t+1} - P_t) grad_w P_t$$

i.e. the weight increment is determined only by the most recent observation.

Summary of RL-DT(λ) method

$$w := w + \Delta w \qquad (s1)$$

$$\Delta w = \sum_{t=1}^{m} \Delta w_t \qquad (s2)$$

$$\Delta w_t = \alpha(P_{t+1} - P_t) e_t(\lambda) \qquad (s3)$$

$$e_t(\lambda) = \lambda e_{t-1}(\lambda) + grad_w P_t \qquad (s4a)$$

$$e_1(\lambda) = grad_w P_1 \qquad (s4b)$$

These formulae make possible simple recursive implementation of the gradient updating

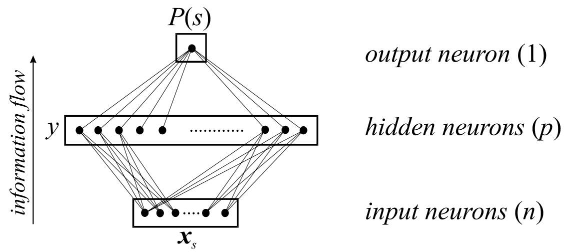
```
Δw:=0;
for t:=1 to m do
begin if t=1 then e(λ):=gradP1
      else e(λ):=λe(λ)+gradPt;
      Δw:=Δw+α(Pt+1-Pt)e(λ);
end;
w:=w+Δw;
```

Neural-network architecture of cognitive device

The predictor evaluation of states is performed by a parametric mapping

$$P(s)=P(\boldsymbol{x}_s;w)$$

This mapping is realized by a feed-forward neural network composed of one layer of hidden neurons



Activities of neurons

(1) Hidden neurons

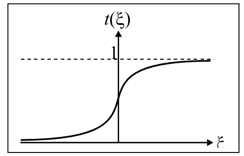
$$y_i=t\left(\sum_{j=1}^{n} w_{ij}x_j^{(s)}+\vartheta_i\right)\;(i=1,2,\ldots,p)$$

(2) Output neuron

$$z_s=P(\boldsymbol{x}_s;w)=t\left(\sum_{i=1}^{p}\tilde{w}_j y_j+\tilde{\vartheta}\right)$$

$t(\xi)$ is an activation „squashing“ function specified by the logistic sigmoid function

$$t(\xi)=\frac{1}{1+e^{-\xi}},\qquad t:R\rightarrow(0,1)\Rightarrow 0<t(\xi)<1$$



Gradient of $P(X)$ with respect to weight and threshold coefficients

(1) Gradient with respect to output weight and threshold coefficients

$$\frac{\partial P(\boldsymbol{x})}{\partial\tilde{\vartheta}}=P(\boldsymbol{x})\left[1-P(\boldsymbol{x})\right],\;\frac{\partial P(\boldsymbol{x})}{\partial\tilde{w}_j}=\frac{\partial P(\boldsymbol{x})}{\partial\tilde{\vartheta}}y_j$$

(2) Gradient with respect to hidden weight and threshold coefficients

$$\frac{\partial P(\boldsymbol{x})}{\partial\vartheta_i}=y_i(1-y_i)\frac{\partial P(\boldsymbol{x})}{\partial\tilde{\vartheta}}\tilde{w}_i,\;\frac{\partial P(\boldsymbol{x})}{\partial w_{ij}}=\frac{\partial P(\boldsymbol{x})}{\partial\vartheta_i}x_j^{(s)}$$

These partial derivatives are calculated recurrently by the **backpropagation** method.

Goal of learning

To adapt the weight and threshold coefficients of the predictor - neural network $P(s)=P(\boldsymbol{x}_s;w)$ such that the produced walks will have the required property.

The learning of predictor – neural network will be performed by the RL-TD(λ) method.

Algorithm of learning

Step 1. *Generate randomly weight and threshold coefficients of the cognitive device predictor - neural network.*

Step 2. *Generate quasirandomly a walk – sequence of states. If the of sequence will has the required property, then set $z=z_{max}$, otherwise set $z=z_{min}$.*

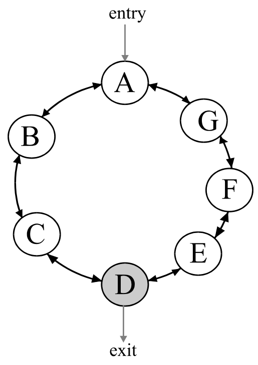
Step 3. *Update weight and threshold coefficients by the RL-TD(λ) formula, where parameter $0\le\lambda\le 1$ is kept fixed through whole learning process.*

Step 4. *Check whether convergence criteria are satisfied, if so, then continue in step 5, otherwise continue in step 2.*

Step 5. *Stop.*

First illustrative example

Let us consider a simple example that corresponds to a generator of bounded random walks composed of six states $A,B,\ldots,F,G$



Examples of walks:

(1) $\mathcal{W}_1$=ABCBAGFED, | $\mathcal{W}_1$|=9

$\mathcal{W}_2$=ABCD, | $\mathcal{W}_2$|=4 (the only shortest walk)

Our goal is to construct walks $\mathcal{W}$ that

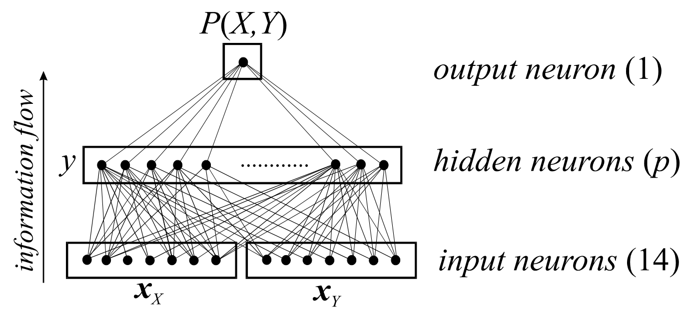
(1) start in the initial state A and end in the terminal state D, and

(2) are of shortest length, i.e. $|\mathcal{W}|=4$.

States are represented by five 6-dimensional binary "unit" vectors

#	state	binary vector $\boldsymbol{x}$
1	A	(1000000)
2	B	(0100000)
3	C	(0010000)
4	D	(0001000)
5	E	(0000100)
6	F	(0000010)
7	G	(0000001)

Each oriented edge (X,Y) is evaluated by a prediction $P(X,Y)$ numerically realized by the feed-forward neural network with input-neuron activities specified by vectors $\boldsymbol{x}_X$ and $\boldsymbol{x}_Y$ assigned to X and Y, respectively



The prediction $P(X,Y)$ is a probability that an actual walk $\mathcal{W}=A...UVX$ will be extended to a walk $\mathcal{W}'=A...UVXY$

$$A...UVX \begin{cases} \nearrow A...UVXV & (Pr_1=P(X,V)) \\ \searrow A...UVXY & (Pr_2=P(X,Y)) \end{cases}$$

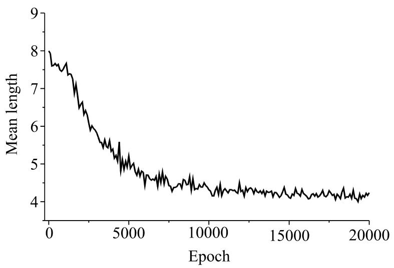
$$Pr_1' = \frac{Pr_1}{Pr_1+Pr_2}, \quad Pr_2' = \frac{Pr_2}{Pr_1+Pr_2}$$

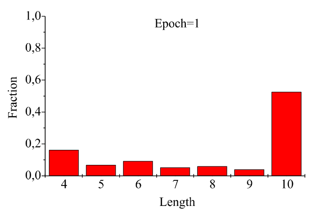
Set of parameters of learning process

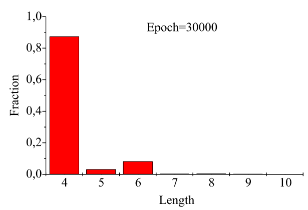
1. α=0.1 (learning rate parameter).
2. λ=0.9 (temporal-difference parameter).
3. p=5 (number of hidden neurons).
4. Initial values of weight and threshold coefficients are randomly generated from the open interval (-2,2).
5. The learning process is stopped after 30000 epochs.
6. Quasirandomly generated walks are evaluated as follows

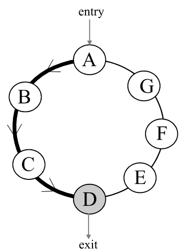
$$z = \begin{cases} 1 & (\text{if } |\mathcal{W}|=4) \\ 0 & (\text{otherwise}) \end{cases}$$

Learning process



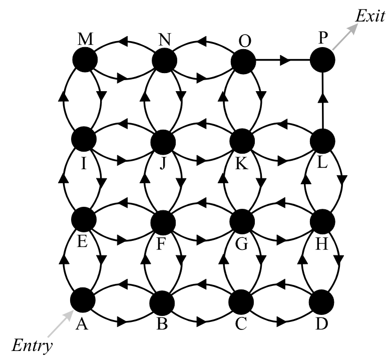






Second illustrative example

Let us consider a slightly complex example than the previous one, it corresponds to a generator of bounded random walks composed of sixteen states *A, B, ...,O, P*.



Our goal is to construct walks *w* that

(1) start in the initial state *A* and end in the terminal state *P*,

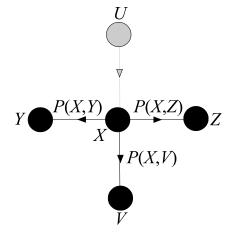
(2) are of shortest length, i.e. |*w*|=6.

States are represented by fifteen 15-dimensional binary "unit" vectors

#	state	binary vector
1	A	(1000000000000000)
2	B	(0100000000000000)
3	C	(0010000000000000)
4	D	(0001000000000000)
………		
13	M	(0000000000001000)
14	N	(0000000000000100)
15	O	(0000000000000010)
16	P	(0000000000000001)

Each oriented edge (*X,Y*) is evaluated by a **predictor** *P*(*X,Y*) with the following meaning:

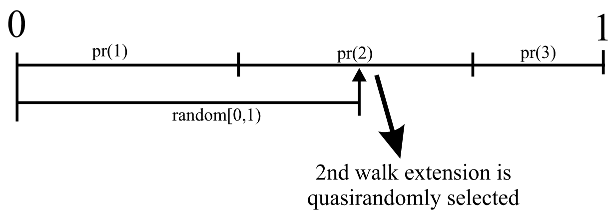
Let us have a walk *w*=(*A*…*UX*) terminated in the state *X*, and let the last state *X* has one to three forthcoming neighbor states denoted *Y*, *Z*, and *V*, respectively. The walk is extended by one of them with probability proportional predictors *P*(*X*,*Y*), *P*(*X*,*Z*), and *P*(*X*,*V*).



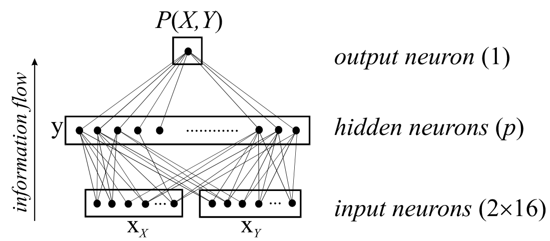
$$\mathcal{W} = A...UX \rightarrow \begin{cases} \mathcal{W}' = A...UXY\left(p_Y \approx P\left(X,Y\right)\right) \\ \mathcal{W}' = A...UXZ\left(p_Z \approx P\left(X,Z\right)\right) \\ \mathcal{W}' = A...UXV\left(p_V \approx P\left(X,V\right)\right) \end{cases}$$

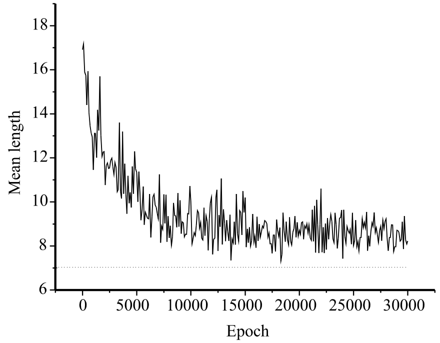
This type of quasirandom selection is numerically realized by the **"roulette wheel"** (see Goldberg's implementation of GA)

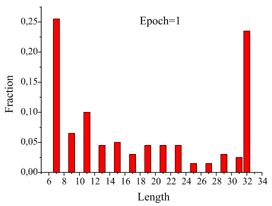
$$pr_1 = \frac{p_Y}{p_Y + p_Z + p_V},\ pr_2 = \frac{p_Z}{p_Y + p_Z + p_V},\ pr_3 = \frac{p_V}{p_Y + p_Z + p_V}$$

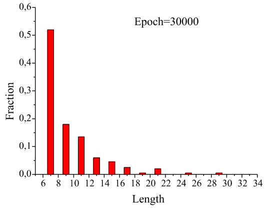


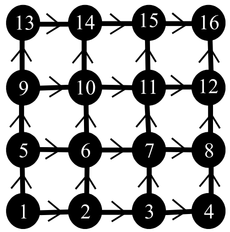
The predictor *P*(*X,Y*) is numerically realized by the feed-forward neural network with input-neuron activities specified by the vector representation of states *X*.







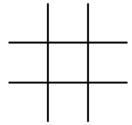




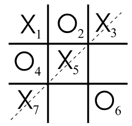
Tretí ilustračný príklad

Hra piškvorky (tic-tac-toe)

The American Heritage Disctionary: *A game played by two people, each trying to make a line of three X's or three O's in a boxlike figure with nine spaces*.



X-hráč (prvý)
O-hráč (druhý)



X-hráč zvíťazil

Hra je zahájená prvým hráčom (X), na jeho ťah odpovedá druhý hráč (O), toto striedanie hráčov sa opakuje až do konca hry. **Koniec hry** nastáva **víťaztvom** hráča, ktorý dosiahol „riadkovu“ pozíciu troch svojich znakov, alebo **remízou**, ak po deviatich ťahoch ani jeden z hráčov nedosiahool víťaznú pozíciu.

Algoritmus hry

1. krok. Hra je zahájená prvým hráčom, $G \leftarrow G_1$, a počiatočnou pozíciou, $P \leftarrow P_{\mathrm{ini}}$.

2. krok. Hráč G vytvorí z pozície P uložením svojho znaku na prázdne miesta množinu všetkých možných nasledujúcich pozícií

$$O_{gen}(P) = \{P_1, P_2, \ldots, P_n\}$$

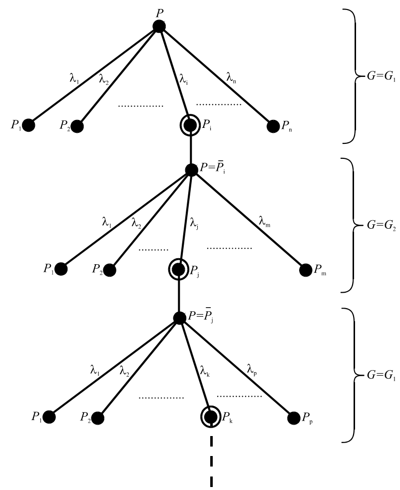
3. krok. Ak množina je prázdna, potom oba hráči G_1 a G_2 remizujú a hra pokračuje krokom 4.

4. krok. Každá pozícia P_{i} je ohodnotená koeficientom $0<\lambda_{\mathrm{i}}<1$. Hráč vyberie za nasledujúcu pozíciu takú $P' \in O_{\mathrm{gen}}(P)$, ktorá je ohodnotená maximálnym koeficientom λ, $P \leftarrow P'$. Ak pozícia P je víťazná, potom hráč G víťazí a hra pokračuje krokom 4.

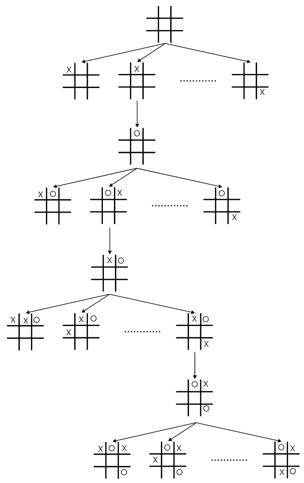
5. krok. **Krok 3**. Hra prechádza na iného hráča, $G \leftarrow \overline{G} = G_2$, pozíciu P si vytvorí inverziou aktuálnej pozície, $P \leftarrow \overline{P}$, hra pokračuje krokom 2.

6. krok. Koniec hry.

Reprezentácia algoritmu pomocou stromu riešení



Vrchná časť stromu riešení



Dimenzia stavového priestoru je určená pomocou jednoduchých kombinatorických úvach takto

$$N = \sum_{p=1}^{5} \binom{9}{p} \left[\binom{9-p}{p-1} + \binom{9-p}{p} \right] - 2 \times 6 \times 13 = 5889$$

Pomocou metódy spätného prehľadávanie je možné zostrojiť celý strom riešení, kde počty koncových pozícií sú uvedené v tabuľke

No.	Počet	Typ
1	131184	víťazstvo hráča X
2	77904	víťazstvo hráča O
3	46080	remíza hráčov X a O
	255168	celkový počet

Z tejto tabuľky vyplýva, že prvý hráč X má väčšiu šancu hru vyhrať. Podrobnou analýzou sa dá ukázať, že aj hráč O môže hru forsírovať tak, že remizuje. Celkový počet koncových vetví v strome riešení možno jednoducho odhadnúť ako 9!=362880.

Model hry

Model obsahuje 6 pravidiel s klesajúcou prioritou:

1. pravidlo. *Hráč vykoná ťah, ktorý vedie k jeho víťazstvu.*

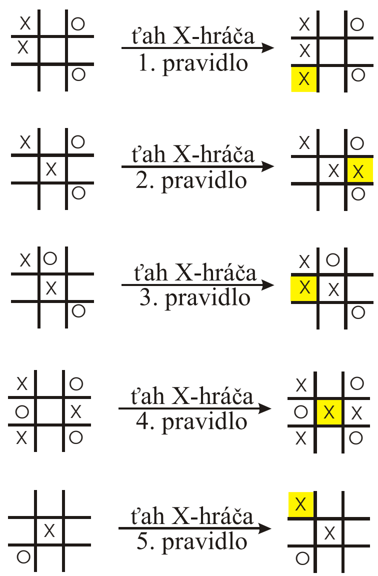
2. pravidlo. *Hráč vykoná ťah, ktorý zabráni víťazstvu oponenta v nasledujúcom ťahu.*

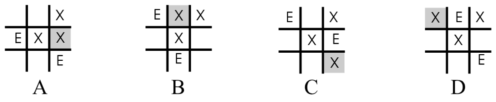
3. pravidlo. *Hráč vykoná ťah, ktorým si pripraví možnosť použitia 1. pravidla v nasledujúcom ťahu (tzv. vidlička).*

4. pravidlo. *Hráč obsadí stredové pole.*

5. pravidlo. *Hráč obsadí rohové pole.*

6. pravidlo. *Hráč obsadí voľné pole.*





Diagramy A-D znázorňujú základné typy vidličkových pozícií, ktoré sú aplikovateľné použitím pravidla 3.

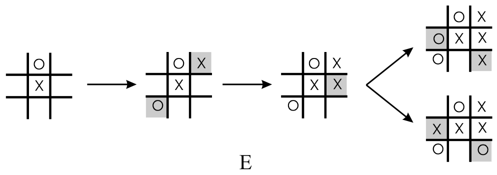


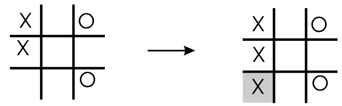
Diagram E ukazuje pozíciu, ktorá je prehraná pre hráča O už po prvom ťahu. Pravidlá nášho modelu hry nepostihujú túto možnosť, jedná sa o predpoveď o tri ťahy dopredu.

Pozícia je reprezentovaná 9-rozmerným vektorom

$$x(P) = (x_1, x_2, \ldots, x_9) \in \{0,1,-1\}^9$$

kde jednotlivé zložky určujú jednotlivé políčka v pozícii P

$$x_i = \begin{cases} 0 \ (i - \text{té pole je neobsadené}) \\ 1 \ (i - \text{té pole je obsadené X}) \\ \text{-1} \ (i - \text{té pole je obsaden O}) \end{cases}$$



P=(1,0,-1,1,0,0,0,0,-1) ⟶ P'=(1,0,-1,1,0,0,1,0,-1)

Agent hrá proti modelu hry piškvorky

Algoritmus modelu

1. krok. *Váhové koeficienty neurónovej siete sú náhodne vygenerované z intervalu [-1,1].*

2. krok. *Polož* t:=1.

3. krok. *S 50% pravdepodobnosťou deklaruj agenta ako prvého* X*-hráča a model ako druhého* O*-hráča (v opačnom prípade je agent deklarovaný ako druhý* O*-hráč a model ako prvý* X*-hráč). Na záver hry pomocou metódy TD(λ) opraví váhové koeficienty kognitívneho orgánu agenta.*

4. krok. *Polož* t:=t+1.

5. krok. *Ak* $t<t_{max}$, *potom pokračuj krokom 3, v opačnom prípade prejdi na krok 6.*

6. krok. *Koniec algoritmu.*

Parametre adaptačného procesu

1.α=0.1 (rýchlosť učenia).

7.λ=0.3 (temporal-difference parameter).

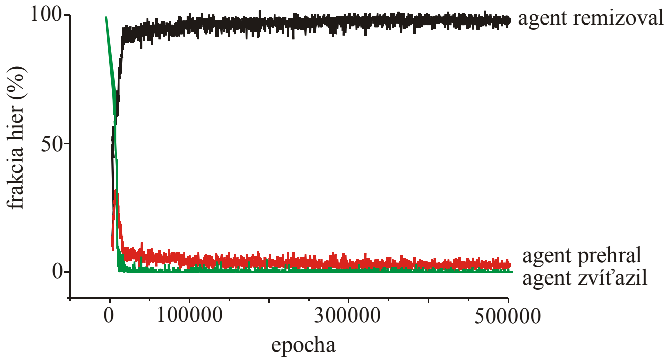
8. p=30 (počet skrytých neurónov).

9. Počiatočné hodnoty váhových a prahových koeficientov sú náhodne vyberané z otvoreného intervalu (-2,2).

10. Učenia je zastavené po 500000 epochách.

11. Pozície sú ohodnocované podľa formule

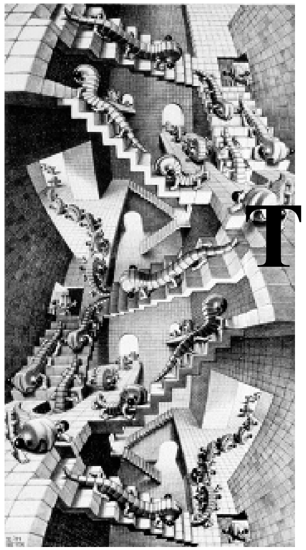
$$z = \begin{cases} 1 & (\text{prvý hráč vyhral}) \\ 0.5 & (\text{hráči remizovali}) \\ 0 & (\text{prvý hráč prehral}) \end{cases}$$



Priebeh frakcií hier (zo 100), ktoré hral agent proti modelu hry, pričom polovicu hier (50) hral ako prvý a druhú polovicu hier hral ako druhý. Z priebehu jednotlivých prípadov jasne vyplýva, že neurónová sieť je schopná tak kvalitnej spontánnej adaptácie, že dokáže neprehrávať s modelom.

Závery

- Použitie učenia s odmenou a trestom v multiagentových systémoch, kde agenti majú kognitívny orgán implementovaný pomocou neurónovej siete, umožňuje štúdium ***emergencie stratégie*** v MAS. V použitom prístupe, agenti hrali proti „presným“ pravidlám hry, takže vyemergovali agenti, ktorý proti modelu neprehrali.
- Existenciu „presných“ pravidiel môžeme potlačiť pomocou ***koevolučného modelu***, kde máme populáciu agentov (neurónových sietí), ktorá je rozdelená na dve podpopulácie, bielych a čiernych agentov. Striedavo hrahú proti sebe tak, že jedna podpopulácia má zafixovanú neurónovú sieť, zatiaľ čo druhá podpopulácia si ju adaptuje pomocou učenia s odmenou a trestom. Aj v tomto prípade spontánne emergujú agenti, ktorý dokonale hrajú TTT.



The End