Elektrotehniški vestnik 75(5): 311–316, 2008
Electrotechnical Review, Ljubljana, Slovenija

Hitro učenje rekurentnih nevronskih mrež s korekcijsko shemo LRP

Branko Šter

Univerza v Ljubljani, Fakulteta za računalništvo in informatiko, Tržaška 25, 1000 Ljubljana, Slovenija
E-pošta: branko.ster@fri.uni-lj.si

Povzetek. Primerjali smo dva algoritma za učenje rekurentnih nevronskih mrež – standardni gradientni RTRL (ang. Real Time Recurrent Learning) in hevristični LRP (ang. Linear Reward-Penalty), ki se sicer uporablja kot korekcijska shema pri učečih avtomatih. Kot problemsko domeno smo uporabili vhodno/izhodna zaporedja končnih avtomatov, ki ustrezajo problemu zakasnjenega ekskluzivnega ALI pri različnih zakasnitvah. Rezultati so pokazali, da pri paketnem učenju LRP deluje nekajkrat hitreje kot RTRL. Pri sprotnem učenju pa sta algoritma po hitrosti primerljiva v mrežah z do 40 nevroni, pri večjih pa je LRP znatno hitrejši.

Ključne besede: rekurentne nevronske mreže, gradientno učenje, hevristični algoritmi, učeči avtomati, končni avtomati

Fast training of recurrent neural networks with the LRP reinforcement scheme

Extended abstract. Recurrent neural networks (RNNs) [3] are neural networks containing feedback connections which create loops and enable the functionality of memorizing. RNNs are able to solve time-dependent tasks, such as modeling dynamical systems, predicting time-series [4], learning sequences, induction of finite state automata [1, 2], etc.

Two algorithms for training recurrent neural networks were compared, i.e. the standard gradient RTRL (Real Time Recurrent Learning) algorithm and heuristic LRP (Linear Reward-Penalty) algorithm, which is otherwise used as a reinforcement scheme in learning automata [5]. The RTRL algorithm [8] may be used in two variants: the batch RTRL, where the same sequence of input/output samples is applied continually for calculation of gradients and consequently for updating the weights, and the on-line RTRL, where the weight updating is done after each presented sample.

The results on different variants of the delayed XOR(τ) task at τ=2, 3, 4 (see Figure 2 for $\tau = 2$ and Figure 3 for $\tau = 3$) and different RNN sizes show that LRP operates much faster than the batch RTRL (Table 1).

The course of the mean squared error (MSE) of the batch RTRL and of the LRP algorithm is also quite different. Figure 4 shows the course of the MSE for different sizes of RNN. The batch RTRL requires a long time before the error begins decreasing; as it finally happens, it decreases quickly and smoothly. The MSE of LRP decreases more monotonically, but on the other hand much less smoothly.

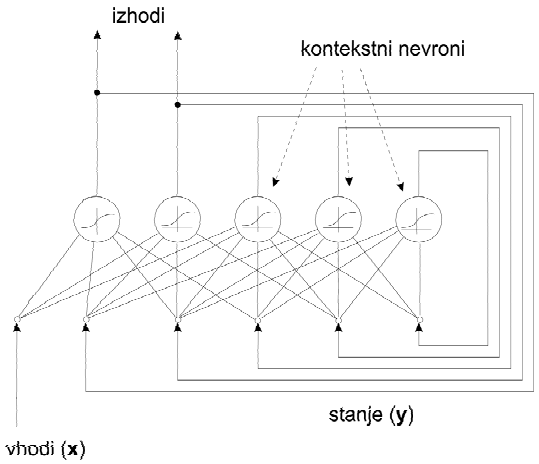
By using a moving window, LRP was also compared to the on-line RTRL. LRP was of a comparable speed in RNNs of up to 40 neurons, but much faster in larger RNNs (Table 2).

Key words: recurrent neural networks, gradient learning, heuristic algorithms, learning automata, finite state automata

Prejet 10. junij, 2008
Odobren 6. oktober, 2008

1 Uvod

Rekurentne nevronske mreže (RNM) so nevronske mreže [3], ki vsebujejo povratne povezave oz. 'feedback' (slika 1). Zanke, ki nastanejo z uvedbo povratnih povezav, omogočajo funkcionalnost pomnjenja. Posebna vrsta



Slika 1. Polno povezana rekurentna nevronska mreža
Figure 1. Fully connected recurrent neural network

RNM so sicer tudi asociativni pomnilniki, npr. znana Hopfieldova nevronska mreža, ki imajo določeno število stabilnih stanj, niso pa namenjeni reševanju časovnih problemov [1, 4], ki nas v našem primeru predvsem zanimajo.

Standardni algoritem za učenje polno povezanih

rekurentnih nevronskih mrež je gradientni algoritem RTRL (ang. Real Time Recurrent Learning) [8]. Le-ta pa zaradi svoje velike računske zahtevnosti lahko pri velikih rekurentnih nevronskih mrežah konvergira zelo počasi. Mogoč je tudi alternativen način učenja s hevrističnimi postopki, ki ponavadi nekako aproksimirajo gradient, vendar z manjšo računsko zahtevnostjo. Eden takih postopkov je tudi algoritem LRP (ang. Linear Reward-Penalty), ki se sicer uporablja kot korekcijska shema pri učečih avtomatih [5]. Že v [7] so uporabili korekcijsko shemo učečih avtomatov za učenje rekurentne nevronske mreže, niso pa izvedli primerjav z algoritmom RTRL glede razlik v hitrosti konvergence, niti ni rezultatov, ki bi pričali o dejanski hitrosti korekcijske sheme učečih avtomatov, če jo uporabimo pri učenju RNM. Prav tako v [7] ni bilo poskusov z učenjem na zaporedja končnih avtomatov, kar je sicer pogost način uporabe RNM.

V nadaljevanju sledi opis obeh algoritmov, RTRL in LRP ter rezultati obeh pri različnih velikostih RNM.

1.1 Algoritem RTRL

Splošen okvir za reševanje časovnih problemov z rekurentnimi nevronskimi mrežami so postavili v [6], kjer so RNM razvili (ang. unfold) v večnivojski perceptron, ki se v vsakem koraku poveča za en nivo. Ta pristop se imenuje vzvratno učenje skozi čas (ang. Backpropagation Through Time - BPTT). Glavni problem tega pristopa je rastoča poraba pomnilnika v primeru rastočega učnega zaporedja. Algoritem RTRL [8] se temu problemu izogne, je pa sicer prav tako splošen kakor BPTT. Obstaja tudi nekaj drugih pristopov, ki uporabljajo posebne arhitekture (RNM niso polno povezane) ali pa temeljijo na računsko omejenih aproksimacijah BPTT.

Opišimo algoritem RTRL. Naj ima RNM n nevronov in m vhodov. Z $\mathbf{y}(t)$ označimo n-terico izhodov nevronov v času t, $\mathbf{x}(t)$ pa naj bo m-terica zunanjih vhodnih signalov v RNM. $\mathbf{y}(t)$ in $\mathbf{x}(t)$ lahko združimo v $(n+m+1)$-terico $\mathbf{z}(t)$, ki pomeni celoten vhod v RNM, tj. *zunanji* vhod $\mathbf{x}(t)$ in stanje $\mathbf{y}(t)$. Stanje je *povratno vezano*, kot kaže slika 1. Dodatna 1 nastopa zaradi prostega člena (ang. bias) za vsak nevron, tj. uteži, ki ni vezana na vhodni signal oz. si lahko predstavljamo, kakor da je vezana na konstanto 1. Množica indeksov k, za katere je z_k izhod nevrona, naj bo U, množica indeksov k, za katere je z_k zunanji vhod, naj bo I, indeks b prostega člena pa naj bo edini element množice B, $B = \{b\}$. Velja

$$z_k(t) = \begin{cases} x_k(t), & \text{če } k \in I \\ y_k(t), & \text{če } k \in U \\ 1, & \text{če } k \in B\,. \end{cases} \tag{1}$$

Naj bo $\mathbf{W}$ matrika uteži RNM, dimenzije $n \times (n+m+1)$, kjer 1 nastopa zaradi prostega člena. Naj bo

$$net_k(t) = \sum_{l \in U \cup I \cup B} w_{kl} z_l(t) \tag{2}$$

utežena vsota vseh vhodov k-tega nevrona, tj. zunanjih vhodov v RNM, povratno vezanih izhodov in dodatne linije za prosti člen, v času t. Izhod nevrona v naslednjem časovnem koraku je

$$y_k(t+1) = f_k(net_k(t))\,, \tag{3}$$

kjer je

$$f_k(net_k) = \frac{1}{1+e^{-net_k}} \tag{4}$$

sigmoidna funkcija. Enačbi (3) in (4) določata celotno *dinamiko delovanja* RNM.

Z učnim algoritmom uteži nastavimo tako, da izhodi RNM ustrezajo želenim izhodom v vsakem trenutku t. Želeni izhod k-tega nevrona v trenutku t označimo z $d_k(t)$. Želeni izhodi navadno niso podani za vse nevrone, temveč le za nekatere (slika 1). Izhodi teh nevronov so obenem tudi zunanji izhodi RNM, drugi nevroni (t.i. kontekstni nevroni) pa so vezani le povratno. Njihova vloga je (neformalno) analogna notranjim stanjem končnega avtomata; bistvena razlika pa je seveda v tem, da je RNM zvezen sistem. Naj bo T podmnožica indeksov U, za katere obstajajo želene vrednosti d_k. Razlika med želenim in dejanskim izhodom k-tega nevrona je

$$e_k(t) = \begin{cases} d_k(t) - y_k(t), & \text{če } k \in T \\ 0, & \text{sicer}\,. \end{cases} \tag{5}$$

Celotna napaka mreže v času t je definirana kot

$$E(t) = 1/2 \sum_{k \in U} [e_k(t)]^2\,. \tag{6}$$

Celotna napaka mreže za neko zaporedje vhodno-izhodnih parov $(\mathbf{x}(t), \mathbf{d}(t))$ v časovnem intervalu od t_0 do t_1 je

$$E_{\text{total}}(t_0, t_1) = \sum_{t=t_0+1}^{t_1} E(t)\,. \tag{7}$$

Odziv na vhode v času t_0 nastane šele v času $t_0 + 1$, skladno z enačbo (3). E_{total} je napaka, ki jo želimo minimizirati z gradientnim postopkom. To storimo tako, da v vsakem ciklu izračunamo gradient napake E_{total} in uteži spremenimo za majhen korak v negativni smeri gradienta E_{total}. Postopek ponavljamo, dokler napaka ne pade pod določeno vrednost, lahko pa tudi vnaprej določeno število ciklov. Ker je napaka E_{total} vsota napak $E(t)$ v posameznih trenutkih, je tudi gradient E_{total} vsota gradientov $E(t)$. Sprememba j-te uteži i-tega nevrona je

$$\Delta w_{ij} = -\alpha \frac{\partial E_{\text{total}}}{\partial w_{ij}} \tag{8}$$

$$= -\alpha \sum_{t=t_0+1}^{t_1} \frac{\partial E(t)}{\partial w_{ij}} \tag{9}$$

$$= \sum_{t=t_0+1}^{t_1} \Delta w_{ij}(t)\,. \tag{10}$$

Torej je sprememba uteži v času t kar gradient trenutne napake $E(t)$ po utežeh:

$$\Delta \mathbf{W}(t) = -\alpha \frac{\partial E(t)}{\partial \mathbf{W}} \tag{11}$$

oz.

$$\Delta w_{ij}(t) = -\alpha \frac{\partial E(t)}{\partial w_{ij}}, \quad i \in U, j \in U \cup I \cup B. \tag{12}$$

Velja

$$\frac{\partial E(t)}{\partial w_{ij}} = -\sum_{k \in U} e_k(t) \frac{\partial y_k(t)}{\partial w_{ij}}, \tag{13}$$

odvod izhoda k-tega nevrona po uteži w_{ij} pa je

$$\frac{\partial y_k(t+1)}{\partial w_{ij}} = f'_k(net_k(t)) \left[\sum_{l \in U} w_{kl} \frac{\partial y_l(t)}{\partial w_{ij}} + \delta_{ik} z_j(t) \right], \tag{14}$$

kjer je δ_{ik} Kroneckerjeva delta funkcija. Tako dobimo dinamični sistem s spremenljivkami

$$p^k_{ij}(t) = \frac{\partial y_k(t)}{\partial w_{ij}} \tag{15}$$

za vse $k \in U, i \in U, j \in U \cup I \cup B$ in naslednjo dinamiko:

$$p^k_{ij}(t+1) = f'_k(net_k(t)) \left[\sum_{l \in U} w_{kl} p^l_{ij}(t) + \delta_{ik} z_j(t) \right]. \tag{16}$$

Začetni pogoj je

$$p^k_{ij}(t_0) = 0, \tag{17}$$

ker ni informacij o dogajanju v času $t_0 - 1$.

Potrebna sprememba uteži v času t je torej

$$\Delta w_{ij}(t) = \alpha \sum_{k \in U} e_k(t) p^k_{ij}(t). \tag{18}$$

Odvod sigmoidne funkcije [6] je

$$f'_k(net_k(t)) = y_k(t+1)[1 - y_k(t+1)]. \tag{19}$$

1.2 Algoritem LRP

Učeči avtomati [5] so preprosti matematični modeli, ki delujejo v interakciji z okoljem, le-to pa je predstavljeno v obliki stohastičnega modela. Učeči avtomati izvajajo akcije, okolje pa se nanje odziva z vrednostno oceno. Sposobni so spreminjati svoje lastnosti oz. parametre tako, da maksimirajo ugodne odzive okolja. Ena od njihovih bistvenih lastnosti je, da delujejo v interakciji z okoljem takoj (lahko bi rekli 'on-line'), ne pa šele po dolgotrajnem zbiranju statistike odzivanja okolja. Tako omogočajo sprotno adaptacijo na okolje.

Korekcijske sheme učečih avtomatov popravljajo verjetnosti izvajanja akcij glede na odzive okolja. V najpreprostejšem primeru se okolje odziva binarno: nagrada oz. kazen. Če je določena akcija nagrajena (ugoden odziv okolja), naj se ji posledično verjetnost izvajanja poveča, če pa je kaznovana (neugoden odziv okolja), naj se ji verjetnost izvajanja zmanjša. Zaradi ohranitve totalne verjetnosti vseh akcij se v prvem primeru verjetnost vsem drugim nekoliko zmanjša, v drugem primeru pa nekoliko poveča.

Korekcijsko shemo učečih avtomatov lahko uporabimo tudi za učenje nevronske mreže [7]. Pri tem akcija pomeni majhno spremembo oz. perturbacijo naključne uteži: ena akcija pomeni povečanje uteži za fiksno vrednost DW, druga akcija pa zmanjšanje uteži za isto vrednost. Zato je število akcij N_A učečega avtomata enako dvakratnemu številu uteži N_W nevronske mreže. Zmanjšanje napake E_{total} prek nekega zaporedja zaradi spremembe posamezne uteži je ugoden odziv oz. nagrada, povečanje napake pa pomeni neugoden odziv oz. kazen.

Naj bo N_W število prostih parametrov RNM (uteži in pragovi oz. prosti členi). Korekcijska shema LRP (linear reward-penalty, originalna oznaka v [5] je L_{R-P}) popravlja $N_A = 2N_W$ perturbacijskih verjetnosti po i-ti perturbaciji takole. Če je i-ta akcija nagrajena v m-tem koraku, se ji verjetnost poveča kot

$$p_i(m+1) = p_i(m) + \lambda[1 - p_i(m)], \tag{20}$$

verjetnosti drugih $N_A - 1$ akcij pa se zmanjšajo kot

$$p_{j \neq i}(m+1) = (1 - \lambda) p_j(m). \tag{21}$$

Če pa je i-ta akcija kaznovana, se ji verjetnost zmanjša kot

$$p_i(m+1) = (1 - \lambda) p_i(m), \tag{22}$$

verjetnosti drugih pa se povečajo kot

$$p_{j \neq i}(m+1) = \frac{\lambda}{N_A - 1} + (1 - \lambda) p_j(m). \tag{23}$$

λ je korekcijski parameter, $0 < \lambda < 1$.

Na začetku je verjetnost vseh akcij enaka, $p_j(0) = 1/N_A$, $j = 1, .., N_A$, pozneje pa se verjetnosti popravljajo. Na ta način nam vektor verjetnosti sprememb uteži stohastično aproksimira gradient. Med premikanjem po utežnem prostoru LRP adaptivno spreminja smer aproksimacije gradienta.

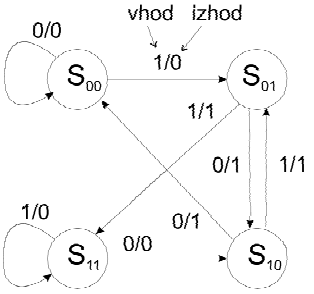
2 Poskusi

Primerjali smo konvergenco algoritmov RTRL in LRP. Velikost nevronskih mrež je bila določena s številom nevronov n.

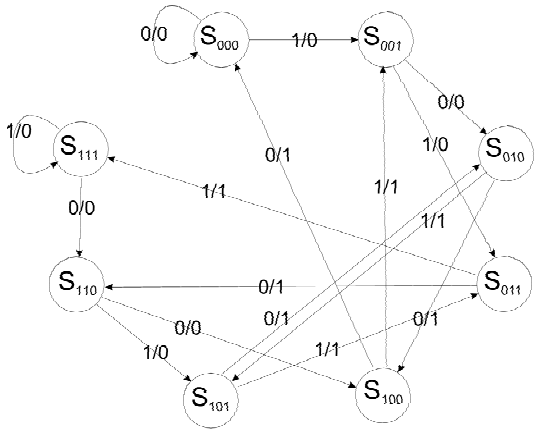
Kot problemsko domeno smo izbrali *zakasnjen ekskluzivni ALI* (krajše dXOR(τ)), pri katerem želena izhodna vrednost ustreza funkciji XOR vhoda x pred τ trenutki in vhoda x pred $\tau + 1$ trenutki:

$$d(t) = x(t - \tau) \oplus x(t - \tau - 1). \tag{24}$$

Vhod x je naključna binarna vrednost. Temu problemu ustreza končni avtomat z 2^τ stanji. Slika 2 prikazuje končni avtomat za $\tau = 2$, slika 3 pa končni avtomat za $\tau = 3$.



Slika 2. Končni avtomat za problem dXOR(2). Vsak prehod vsebuje par vhod/izhod
Figure 2. Finite state machine for dXOR(2) task. Each transition has its input/ouput pair



Slika 3. Končni avtomat za problem dXOR(3). Vsak prehod vsebuje par vhod/izhod
Figure 3. Finite state machine for dXOR(3) task. Each transition has its input/ouput pair

RNM dejansko izvaja indukcijo končnega avtomata na podlagi opazovanja odzivov (izhodov) na naključno vhodno zaporedje. Upoštevati je treba tudi dejstvo, da je RNM zvezen dinamični sistem, končni avtomat pa diskreten. Zato je treba izhod RNM na koncu diskretizirati. Za zanesljivo indukcijo končnega avtomata je potrebno dovolj dolgo zaporedje vhodno-izhodnih parov.

Učenje nevronskih mrež je lahko paketno ('batch'), pri katerem se popravljajo uteži glede na fiksno učno množico (oz. zaporedje pri časovnih problemih), ki jo mreža 'vidi' mnogokrat, ali pa sprotno ('on-line'), kjer se uteži popravljajo po vsakem vzorcu, učna množica (zaporedje) pa se ne ponavlja. Za sprotno učenje je seveda potrebna možnost generiranja poljubno dolgega zaporedja, kar pa pri poznanem avtomatu ni problem. Paketni algoritem RTRL bomo imenovali RTRLp, sprotni algoritem RTRL pa RTRLs.

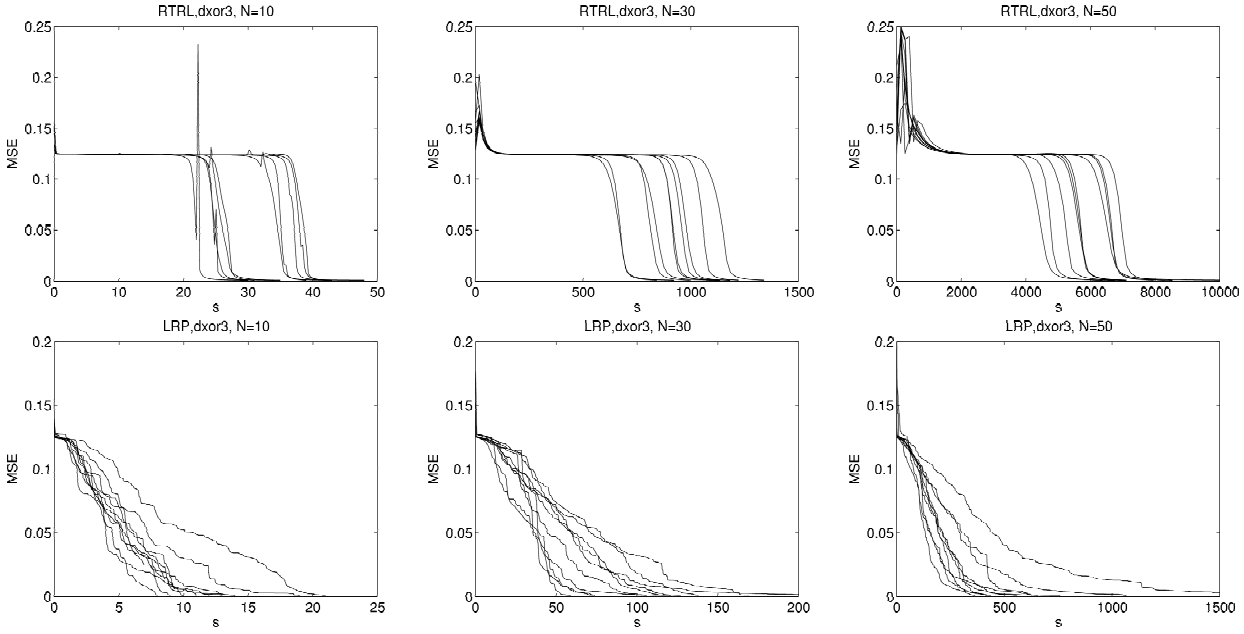
Tabela 1. Povprečen čas izvajanja v sekundah (v oklepajih je standardna deviacija) paketnega RTRL in LRP na problemih dXOR(τ), $\tau = 2, 3, 4$, na 10 poskusih za vsak τ. Učno zaporedje vsebuje 1000 vzorcev; n je število nevronov RNM
Table 1. Average execution time in seconds (standard deviation in parentheses) of the batch RTRL and LRP on dXOR(τ) task, $\tau = 2, 3, 4$, on 10 experiments for each τ. The training sequence consists of 1000 samples; n is the number of neurons of the RNN

τ	n	RTRLp	LRP
2	10	22.1 (5.2)	14.2 (3.5)
	20	215.1 (40.8)	57.9 (24.4)
	30	893.0 (80.1)	185.4 (76.8)
	40	2451.2 (227.4)	472.6 (142.4)
	50	5822.1 (513.7)	912.2 (391.0)
3	10	37.5 (6.4)	13.0 (4.2)
	20	310.3 (76.6)	45.7 (10.6)
	30	1100.1 (145.6)	120.9 (49.5)
	40	3137.6 (252.0)	277.9 (67.4)
	50	7688.2 (1266.7)	725.0 (471.0)
4	10	53.0 (22.5)	18.4 (30.5)
	20	376.5 (30.0)	35.0 (11.3)
	30	1427.1 (180.7)	131.9 (68.4)
	40	4178.3 (467.6)	218.4 (98.2)
	50	9734.2 (1001.0)	405.2 (140.9)

Algoritem LRP za sprotno popravljanje uteži ni najprimernejši, kajti vpliv perturbacije posamezne uteži je težko ali celo nemogoče vrednotiti na podlagi razlike v kvaliteti napovedi izhoda zaporedja v dveh sosednih trenutkih. Zato je za zanesljivo oceno delovanja mreže pri podanih utežeh potrebno vrednotenje na zaporedju zadostne dolžine. Neposredno je torej algoritem LRP mogoče primerjati le z RTRLp.

Algoritem LRP potrebuje za rešitev nekega problema bistveno večje število korakov kakor RTRLp, vendar pa je čas izvajanja korakov LRP tudi znatno krajši. Zato primerjava števila korakov tukaj ni smiselna. Treba je upoštevati dejanski čas izvajanja. Le-ta pa je odvisen tudi od tipa računalnika. Poskuse smo izvajali na PC računalniku s procesorjem Intel Core2 Duo, 2.33 GHz. Razumljivo je, da so absolutni časi izvajanja vezani na konkreten tip računalnika. Pomembnejše je razmerje med rezultati (tj. časi) obeh algoritmov, ker nas zanima predvsem primerjava. Oba programa sta bila kodirana v C++ in prevedena z enako mero optimizacije.

Najprej smo uporabili paketno učenje, tj. učenje na zaporedju omejene dolžine, v našem primeru 1000 vzorcev. Merili smo čas delovanja algoritma v sekundah, števila korakov pa torej nismo opazovali. Kot merilo za konvergenco algoritma smo izbrali padec srednje kvadratne napake (MSE, tj. povprečena E_{total}) pod 10^{-3}. Izkazalo se je, da to v večini primerov zadostuje za 100-



Slika 4. Potek povprečne kvadratne napake (MSE) algoritmov RTRLp in LRP na problemu dXOR(3) za različne velikosti RNM (10, 30 in 50 nevronov) - 10 poskusov za vsako velikost
Figure 4. Development of the mean-squared error (MSE) of the batch RTRL (RTRLp) and the LRP algorithm on the dXOR(3) task for differently sized RNNs (10, 30 and 50 neurons) - ten experiments for each size

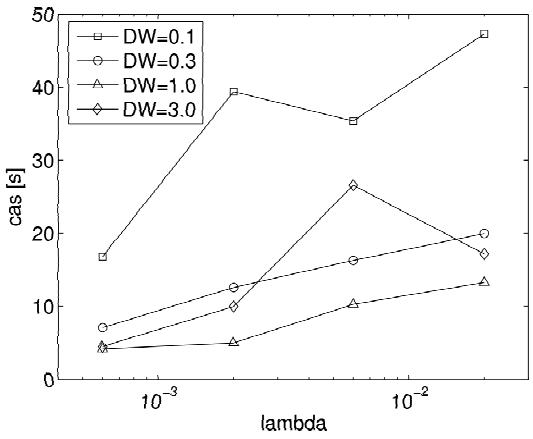
odstotno pravilno napoved RNM na testnem zaporedju, ki prav tako vsebuje 1000 vzorcev. Glavni parameter RTRL je učni korak α. Le-ta je bil izbran kot največji ($\alpha = 10$), pri katerem še ne pride do prevelikih oscilacij napake MSE. Privzeti vrednosti parametrov algoritma LRP sta bili: $DW = 0.1$ in $\lambda = 0.02$. Pri teh je delovanje dovolj robustno, nismo pa ju v tej fazi optimizirali.

Tabela 1 prikazuje rezultate na problemih dXOR(2), dXOR(3) in dXOR(4) za mreže različnih velikosti: 10, 20, 30, 40 in 50 nevronov, za algoritma RTRLp in LRP. Razvidno je, da je LRP bistveno hitrejši - v povprečju približno petkrat hitrejši.

Slika 4 prikazuje poteke povprečne kvadratne napake (MSE) algoritmov RTRLp in LRP za nekaj različnih velikosti RNM. Jasno lahko vidimo povsem različen tip poteka MSE obeh algoritmov. Algoritem RTRLp potrebuje veliko časa, preden začne napaka upadati; ko pa do tega pride, se to zgodi hitro. Poleg tega pa je potek značilno gladek. Napaka pri LRP pada bolj enakomerno, obenem pa veliko bolj živahno, potek ni gladek.

Preizkusili smo tudi vpliv glavnih parametrov algoritma LRP na rezultate, tj. čas konvergence. Na problemu dXOR(3) pri $n = 20$ nevronih smo parameter DW spreminjali kot 0.1, 0.3, 1.0, 3.0, parameter λ pa smo spreminjali kot 0.0006, 0.002, 0.006, 0.02. Dobimo torej 16 kombinacij obeh parametrov. Slika 5 podaja rezultate. V večini primerov se dobro izkaže relativno velik DW in majhen λ. Treba pa se je tudi zavedati, da to velja za konkreten problem z RNM konkretne velikosti.

Ker se zaporedje dolžine 1000 lahko pri preprostih



Slika 5. Povprečen čas izvajanja (v sekundah) 10 tekov algoritma LRP na problemu dXOR(3) za različne vrednosti parametrov DW in λ. Učno zaporedje vsebuje 1000 vzorcev. Število nevronov $n = 20$
Figure 5. Average execution time (in seconds) of ten runs of the LRP algorithm on the dXOR(3) task. The training sequence consists of 1000 samples. The number of neurons is $n = 20$

problemih izkaže za nepotrebno veliko, je boljši način učenja na zaporedje poznanega avtomata sprotni ('online') način. V ta namen uporabimo sprotno varianto algoritma RTRL, torej RTRLs. Ker algoritem LRP ne more delovati na sproten način, se lahko temu približamo tako, da vzamemo učno zaporedje določene dolžine (recimo temu okno), ki ga sčasoma pomikamo naprej. Da se napaka ne bi spreminjala prehitro, kar slabo vpliva na de-

Tabela 2. Povprečni časi izvajanja (v oklepajih je standardna deviacija) algoritmov RTRLs in LRP s pomikanjem okna pri parametrih $DW = 0.3$, $\lambda = 0.001$, $\omega = 200$, $\gamma = 200$, na problemu dXOR(3)
Table 2. Average execution times (standard deviation in parentheses) of algorithms RTRLs and LRP with moving window at parameters $DW = 0.3$, $\lambda = 0.001$, $\omega = 200$, $\gamma = 200$, on the dXOR(3) task

N	RTRLs	LRP
10	0.40 (0.52)	0.4 (0.70)
20	4.0 (1.05)	2.3 (0.67)
30	14.0 (2.6)	13.4 (19.6)
40	33.8 (5.1)	32.6 (29.0)
50	68.6 (5.8)	46.1 (32.6)
60	208.9 (31.9)	129.2 (41.0)
70	420.1 (207.8)	180.2 (32.7)
80	988.6 (208.0)	243.8 (62.8)

lovanje LRP, okno pomaknemo za en vzorec na vsakih γ korakov. Velikost okna označimo z ω.

Tabela 2 prikazuje rezultate sprotnega učenja algoritmov RTRLs ter LRP s pomikanjem okna. Velikost okna ω je bila 200, γ prav tako 200. Skladno z rezultati poskusa na sliki 5 je bil izbran manjši parameter λ. Tudi za RTRLs je bil izbran ugoden parameter: $\alpha = 1$ je bila največja vrednost učnega koraka, pri kateri ni prihajalo do prevelikih nihanj MSE. Vidimo, da sta algoritma pri manjših RNM (do 40 nevronov) primerljiva, pri večjih pa je LRP znatno hitrejši, npr. pri 80 nevronih že za faktor 4.

3 Sklep

Čeprav algoritem LRP potrebuje veliko več iteracij od RTRL, je časovno gledano bistveno hitrejši. Še posebno to velja za paketno učenje, pri katerem imamo na voljo le eno zaporedje omejene dolžine. Kadar RNN učimo na vhodno/izhodno obnašanje znanega avtomata, ponavadi raje uporabimo sprotni algoritem RTRL. Rezultati kažejo, da sta algoritma LRP in sprotni RTRL po hitrosti enakovredna pri RNM z do 40 nevroni, pri večjih RNM pa je algoritem LRP znatno hitrejši.

4 Literatura

[1] A. Cleeremans, D. Servan-Schreiber, J. L. McClelland, Finite State Automata and Simple Recurrent Networks, *Neural Computation* 1(3): 372–381, 1989.

[2] I. Gabrijel, A. Dobnikar, On-line identification and reconstruction of finite automata with generalized recurrent neural networks, *Neural Networks* 16: 101–120, 2003.

[3] S. Haykin, *Neural Networks: A Comprehensive Foundation*, Macmillan College Publishing Company, 1994.

[4] U. Lotric, A. Dobnikar, Predicting time series using neural networks with wavelet-based denoising layers, *Neural computing and applications* 14(1): 11–17, 2005.

[5] K. Narendra, M. Thathachar, *Learning Automata: An Introduction*, Prentice-Hall, Englewood Cliffs, New Jersey, 1989.

[6] D. E. Rumelhart, G. E. Hinton, R. J. Williams, Learning internal representations by error propagation, v D. E. Rumelhart, J. L. McClelland (ur.): *Parallel Distributed Processing* 1: 318–362, The MIT Press, Cambridge, Massachusetts, 1986.

[7] M. Sundareshan, T. Condarcure, Recurrent neural network training by a learning automaton approach for trajectory learning and control system design, *IEEE Transactions on Neural Networks* 9(3): 354–368, 1998.

[8] R. J. Williams, D. Zipser, A Learning Algorithm for Continually Running Fully Recurrent Neural Networks. *Neural Computation* 1(2): 270–280, 1989.

Branko Šter je docent na Fakulteti za računalništvo in informatiko Univerze v Ljubljani. Diplomiral je leta 1993 in magistriral leta 1996 na Fakulteti za elektrotehniko ter doktoriral leta 1999 na Fakulteti za računalništvo in informatiko. Raziskovalno se ukvarja z nevronskimi mrežami, mehkim računanjem, mobilno robotiko in dinamičnimi sistemi.