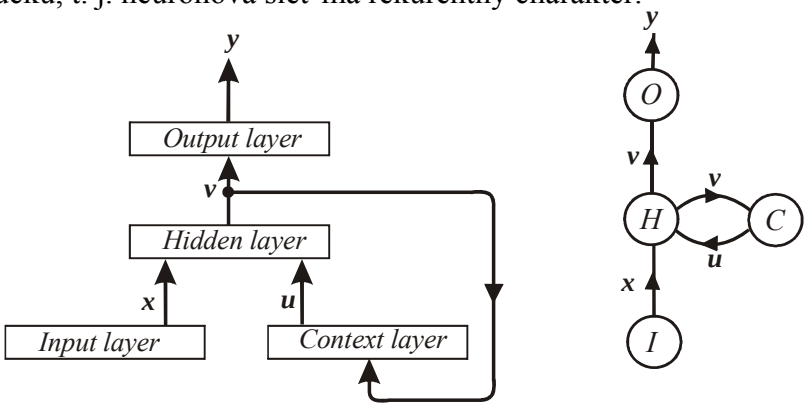
Algoritmizácia Elmanovej rekuretnej neurónovej siete

Vladimír Kvasnička
ÚAI FIIT STU

1. Diagramatická reprezentácia Elanovej rekurentnej neurónovej siete

Diagramatická rereprezentácia Elamovej neurónovej siete je znázornená na obr. 1. Táto sieť ma tri základné vrstvy neurónovej: vstupnú, skrytú a výstupnú, pričom spoje medzi neurónmi sú len medzi susednými vrstvami vstupnou-skrytou a skrytou-výstupnou. Rekurentnosť neurónovej siete je implementovaná pomou kontextovej vrstvy, ktorá spolu so skrytou vrstvou tvorí slučku, t. j. neurónová sieť má rekurentný charakter.



Obrázok 1. Diagramatická reprezentácia Elamnovej rekurentnej neurónovej siete.

Aktivity neurónovej siete sú popísané funkciami

$$\begin{aligned} \boldsymbol{x} &= input \in \{a,b,c,d\}^* \\ \boldsymbol{u} &= C(\boldsymbol{v}) \\ \boldsymbol{v} &= H(\boldsymbol{x},\boldsymbol{u}) \\ \boldsymbol{y} &= O(\boldsymbol{v}) \in R^n \end{aligned} \tag{1}$$

kde $\boldsymbol{u} = C(\boldsymbol{v})$ popisuje výstupnú aktivitu z kontextovej vrstvy $\boldsymbol{u}$ ako odozvu na aktivitu $\boldsymbol{v}$ zp skrytej vrstvy, $\boldsymbol{v} = H(\boldsymbol{x},\boldsymbol{u})$ je funkcia pre aktivitu $\boldsymbol{v}$ ako odozvu na aktivity $\boldsymbol{x}$ zo skrytej vrstvy a aktivity $\boldsymbol{u}$ z kontextovej vrstvy. Výstupná aktivita $\boldsymbol{y} = O(\boldsymbol{v})$ je odozvou na aktivitu $\boldsymbol{v}$ skrytej vrstvy. Nelineárnosť týchto rovníc pre aktivity Elamovej siete ľahko zostrojíme jesdnoduchou úpravou formúl v (1), dosadením druhej formuly do tretej

$$\begin{aligned} \boldsymbol{x} &= input \in \{a,b,c,d\}^* \\ \boldsymbol{v} &= H(\boldsymbol{x},C(\boldsymbol{v})) \quad (nelineárnosť) \\ \boldsymbol{y} &= O(\boldsymbol{v}) \in R^n \end{aligned} \tag{2}$$

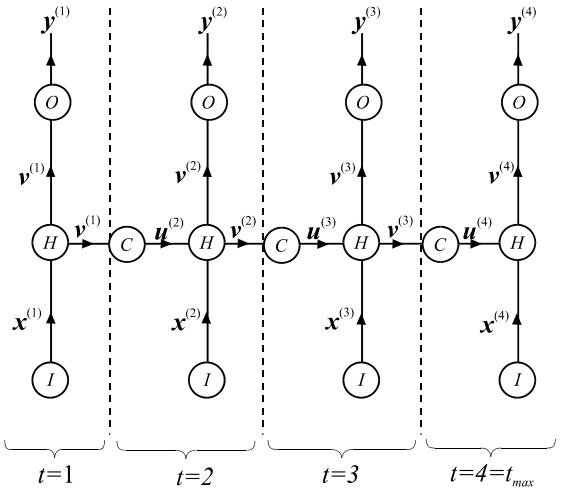
Tento systém navzájom zpriahnutých rovníc pre aktivity neurónov môže byť riešený metódou jednoduchých iterácií

$$x^{(t+1)} = f\left(x^{(t)}\right) \tag{3}$$

kde riešenie v „čase“ t+1 je spočítané pomocou funkcie $f(\cdot)$ ako odozva na argument v „čase“ t. Táto schéma je úplne špecifikovaná uvedením počiatočného riešenia $x^{(0)}$. Iteračným riešením systému (1) dostaneme

$$\left.\begin{aligned} \boldsymbol{x}^{(t)} &= \text{input} \\ \boldsymbol{u}^{(t)} &= \begin{cases} 0 & \left(\text{for } t=1\right) \\ C\left(\boldsymbol{v}^{(t-1)}\right) & \left(\text{for } t \geq 2\right) \end{cases} \\ \boldsymbol{v}^{(t)} &= H\left(\boldsymbol{x}^{(t)}, \boldsymbol{u}^{(t)}\right) \\ \boldsymbol{y}^{(t)} &= O\left(\boldsymbol{v}^{(t)}\right) \end{aligned}\right\} \quad \text{for } t = 1,2,...,t_{max} \tag{4}$$

kde $\boldsymbol{x}^{(t)}$ je vstup do neurónovej siete v “čase” t, $\boldsymbol{u}^{(t)}$ je odozva kontextovej vrstvy v “čase” t na vstup $\boldsymbol{v}^{(t-1)}$, výstup $\boldsymbol{v}^{(t)}$ zo skrytej vrstvy v čase t je odozvou na vstupy $\boldsymbol{x}^{(t)}$ a $\boldsymbol{u}^{(t)}$ v rovnakom „čase“ t, a $\boldsymbol{y}^{(t)}$ je odozva výstupnej vrstvy v čase t na aktivitu $\boldsymbol{v}^{(t)}$ skrytej vrstvy v rovnakom čase. Toto riešenie môžeme formálne diagramaticky interpretovať ako rozvinutie neurónovej siete znázornenj na obr. 1 (pozri obr. 2)

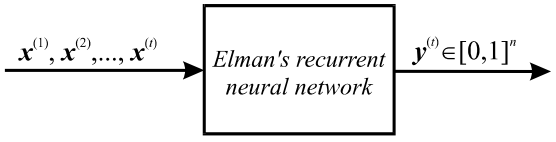


Obrázok 2. Časové rozvinmutie Elamaovej rekurentnej neurónovej siete zostrojené pomocou iteračného riešenia systému (4). Rozvinutá neurónová sieť v každom časovom okamžiku $t \in \{1,2,...,t_{\max}\}$ počíta výstup $y^{(t)}$ ako odozvu na vstup $x^{(1)}, x^{(2)}, ..., x^{(t)}$. Aktivity $v^{(1)}, u^{(2)}, v^{(2)}, ..., u^{(t_{\max})}, v^{(t_{\max})}$ sú medzivýsledky potrebné k výpočtu výstupnej postupnosti $y^{(1)}, y^{(2)}, ..., y^{(t_{\max})}$ ako odozvy na vstupnú postupnosť $x^{(1)}, x^{(2)}, ..., x^{(t_{\max})}$.

Rozvinutú neurónovú sieť môžeme chápať ako parametrickú funkciu, ktorá zobrazuje postupnosť vstupov $x^{(1)}, x^{(2)}, ..., x^{(t)}$ na výstup $y^{(t)}$ (pozri legendu k obr. 2)

$$\boldsymbol{y}^{(t)} = G\left(\boldsymbol{x}^{(1)}, \boldsymbol{x}^{(2)}, ..., \boldsymbol{x}^{(t)}; \boldsymbol{w}, \vartheta\right) \tag{5}$$

Pre t=1, 2,…, $t_{\max}$. Táto skutočnosť je diagramaticky znázornená na obr. 3.



Obrázok 3. Diagramatická interpretácia formuly (5).

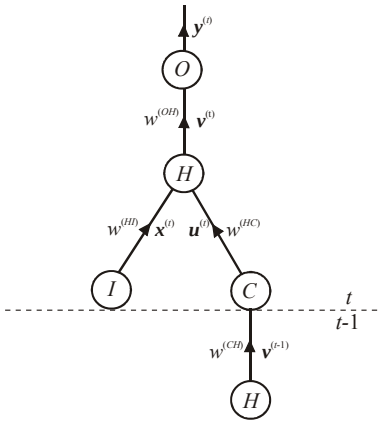
Aktivity jednotlivých vrstiev neurónov v rozvinutej neurónovej sieti sú určené formulami (pozri obr. 4)

$$x_i^{(t)} = \text{input} \tag{6a}$$

$$u_i^{(t)} = \begin{cases} 0 & (t=1) \\ t\left(\vartheta_i^{(C)} + \sum_{j=1}^{N_H} w_{ij}^{(CH)} v_j^{(t-1)} \right) & (t \geq 2) \end{cases} \tag{6b}$$

$$v_i^{(t)} = t\left(\vartheta_i^{(H)} + \sum_{j=1}^{N_I} w_{ij}^{(HI)} x_j^{(t)} + \sum_{j=1}^{N_C} w_{ij}^{(HC)} u_j^{(t)} \right) \tag{6c}$$

$$y_i^{(t)} = t\left(\vartheta_i^{(O)} + \sum_{j=1}^{N_H} w_{ij}^{(OH)} v_j^{(t)} \right) \tag{6d}$$



Obrázok 4. Element rozvinutej neurónovej siete znázornenej na obr. 2. Pomocou tohto diagramu ľahko špecifikujeme jednotlivé aktivity neurónov pre daný časový okamžik *t*.

Adaptačná fáza neurónovej siete je určená pomocou tréningovej množiny

$$A_{train} = \left\{ \left(\boldsymbol{x}^{(1)}, \boldsymbol{x}^{(2)}, ..., \boldsymbol{x}^{(t_{max})} \right) \Big/ \left(\boldsymbol{y}_{req}^{(1)}, \boldsymbol{y}_{req}^{(2)}, ..., \boldsymbol{y}_{req}^{(t_{max})} \right) \right\} \tag{7}$$

a účelovej funkcie

$$E = \frac{1}{2} \sum_{t=1}^{t_{max}} \omega_t \left(\boldsymbol{y}^{(t)} - \boldsymbol{y}_{req}^{(t)} \right)^2 \tag{8}$$

kde pomocou koeficientov ω_t môžeme modelovať zahrnutie jednotlivých časových stavov do adaptačného procesu neurónovej siete. Adaptácia neurónovej siete je realizovaná pomocou minimalizácie tejto účelovej funkcie

$$\left(w_{opt}, \vartheta_{opt} \right) = \arg \min_{(w,\vartheta)} E\left(w, \vartheta \right) \tag{9}$$

Tento optimalizačný problém je obvykle uskutočnený pomocou jednoduchej gradientovej metódy najprudšieho spádu

$$w_{k+1} = w_k - \lambda \, grad \, E\left(w_k \right) \tag{10}$$

kde parameter $\lambda>0$ je nazývaný *rýchlosť učenia*. Gradient účelovej funkcie *E* sa počíta aplikáciou štandardnej metódy spätného šírenia chýb známej z teórie dopredných neurónových sietí

$$\left(\frac{\partial E}{\partial \vartheta_i}\right) = t'(\xi_i)\left(g_i + \sum_k \frac{\partial E}{\partial \vartheta_k} w_{ki}\right) \tag{11a}$$

$$t'(\xi_i) = t(\xi_i)\left[1 - t(\xi_i)\right], \quad g_i = \begin{cases} x_i - x_{i,req} & (i \in O) \\ 0 & (i \notin O) \end{cases} \tag{11b}$$

$$\frac{\partial E}{\partial w_{ij}} = \frac{\partial E}{\partial \vartheta_i} x_j \tag{11c}$$

2. Výpočet parciálnych derivácií účelovej funkcie

Použitím všeobecných formúl (11) môžeme pristúpiť k výpočtu parciálnych derivácií účelovej funkcie vzhľadom k prahovým koeficientom

(1) Inicializácia, $t=t_{\max}$

$$\left(\frac{\partial E}{\partial \vartheta_i^{(O)}}\right)^{(t_{max})} = y_i^{(t_{max})}\left(1 - y_i^{(t_{max})}\right)\omega_{t_{max}}\left(y_i^{(t_{max})} - y_{i,req}^{(t_{max})}\right) \tag{12a}$$

$$\left(\frac{\partial E}{\partial \vartheta_i^{(H)}}\right)^{(t_{max})} = v_i^{(t_{max})}\left(1 - v_i^{(t_{max})}\right)\sum_{j=1}^{N_O}\left(\frac{\partial E}{\partial \vartheta_l^{(O)}}\right)^{(t_{max})} w_{ji}^{(OH)} \tag{12b}$$

$$\left(\frac{\partial E}{\partial \vartheta_i^{(C)}}\right)^{(t_{max})} = u_i^{(t_{max})}\left(1 - u_i^{(t_{max})}\right)\sum_{j=1}^{N_H}\left(\frac{\partial E}{\partial \vartheta_j^{(H)}}\right)^{(t_{max})} w_{ji}^{(HC)} \tag{12c}$$

(2) Iterácia, $1 \le t < t_{\max}$

$$\left(\frac{\partial E}{\partial \vartheta_i^{(O)}}\right)^{(t)} = y_i^{(t)}\left(1 - y_i^{(t)}\right)\omega_t\left(y_i^{(t)} - y_{i,req}^{(t)}\right) \tag{13a}$$

$$\left(\frac{\partial E}{\partial \vartheta_i^{(H)}}\right)^{(t)} = v_i^{(t)}\left(1 - v_i^{(t)}\right)\left(\begin{array}{l} \sum_{j=1}^{N_O}\left(\frac{\partial E}{\partial \vartheta_j^{(O)}}\right)^{(t)} w_{ji}^{(OH)} + \\ \sum_{j=1}^{N_C}\left(\frac{\partial E}{\partial \vartheta_l^{(C)}}\right)^{(t+1)} w_{ji}^{(CH)} \end{array}\right) \tag{13b}$$

$$\left(\frac{\partial E}{\partial \vartheta_i^{(C)}}\right)^{(t)} = u_i^{(t)}\left(1 - u_i^{(t)}\right)\sum_{j=1}^{N_H}\left(\frac{\partial E}{\partial \vartheta_l^{(H)}}\right)^{(t)} w_{li}^{(HC)} \tag{13c}$$

Použitim formuly (12c) z týchto parciálnych derivácií vzhľadom k prahovým koeficientom môžeme zostrojiť parciálne derivácie vzhľadom k váhovým koeficientom.

(1) Inicializácia, $t=t_{\max}$

$$\left(\frac{\partial E}{\partial w_{ij}^{(OH)}}\right)^{(t_{max})} = \left(\frac{\partial E}{\partial \vartheta_i^{(O)}}\right)^{(t_{max})} v_j^{(t_{max})} \tag{14a}$$

$$\left(\frac{\partial E}{\partial w_{ij}^{(HC)}}\right)^{(t_{max})} = \left(\frac{\partial E}{\partial \vartheta_i^{(H)}}\right)^{(t_{max})} u_j^{(t_{max})} \tag{14b}$$

$$\left(\frac{\partial E}{\partial w_{ij}^{(HI)}}\right)^{(t_{max})} = \left(\frac{\partial E}{\partial \vartheta_i^{(H)}}\right)^{(t_{max})} x_j^{(t_{max})} \tag{14c}$$

(2) Iterácia, $1 \le t < t_{\max}$

$$\left(\frac{\partial E}{\partial w_{ij}^{(OH)}}\right)^{(t)} = \left(\frac{\partial E}{\partial \vartheta_i^{(O)}}\right)^{(t)} v_j^{(t)} \tag{15a}$$

$$\left(\frac{\partial E}{\partial w_{ij}^{(CH)}}\right)^{(t)} = \left(\frac{\partial E}{\partial \vartheta_i^{(C)}}\right)^{(t)} v_j^{(t-1)} \tag{15b}$$

$$\left(\frac{\partial E}{\partial w_{ij}^{(HC)}}\right)^{(t)} = \left(\frac{\partial E}{\partial \vartheta_i^{(H)}}\right)^{(t)} u_j^{(t)} \tag{15c}$$

$$\left(\frac{\partial E}{\partial w_{ij}^{(HI)}}\right)^{(t)} = \left(\frac{\partial E}{\partial \vartheta_i^{(H)}}\right)^{(t)} x_j^{(t)} \tag{15d}$$

Pomocou týchto elementárnych parciálnych derivácii môžeme zostrojiť celkové parciálne derivácie tak, že ich vysumujeme pre všetky časové okamžiky $t = 1, 2, ..., t_{\max}$.

$$\frac{\partial E}{\partial \vartheta_i^{(O)}} = \sum_{t=1}^{t_{max}} \left(\frac{\partial E}{\partial \vartheta_i^{(O)}}\right)^{(t)} \tag{16a}$$

$$\frac{\partial E}{\partial \vartheta_i^{(H)}} = \sum_{t=1}^{t_{max}} \left(\frac{\partial E}{\partial \vartheta_i^{(H)}}\right)^{(t)} \tag{16b}$$

$$\frac{\partial E}{\partial \vartheta_i^{(C)}} = \sum_{t=2}^{t_{max}} \left(\frac{\partial E}{\partial \vartheta_i^{(C)}}\right)^{(t)} \tag{16c}$$

$$\frac{\partial E}{\partial w_{ij}^{(OH)}} = \sum_{t=1}^{t_{max}} \left(\frac{\partial E}{\partial w_{ij}^{(OH)}}\right)^{(t)} = \sum_{t=1}^{t_{max}} \left(\frac{\partial E}{\partial \vartheta_i^{(O)}}\right)^{(t)} v_j^{(t)} \tag{16d}$$

$$\frac{\partial E}{\partial w_{ij}^{(HC)}} = \sum_{t=2}^{t_{max}} \left(\frac{\partial E}{\partial w_{ij}^{(HC)}}\right)^{(t)} = \sum_{t=2}^{t_{max}} \left(\frac{\partial E}{\partial \vartheta_i^{(H)}}\right)^{(t)} u_j^{(t)} \tag{16e}$$

$$\frac{\partial E}{\partial w_{ij}^{(CH)}} = \sum_{t=1}^{t_{max}-1} \left(\frac{\partial E}{\partial w_{ij}^{(CH)}}\right)^{(t)} = \sum_{t=1}^{t_{max}-1} \left(\frac{\partial E}{\partial \vartheta_i^{(C)}}\right)^{(t+1)} v_j^{(t)} \tag{16f}$$

$$\frac{\partial E}{\partial w_{ij}^{(HI)}} = \sum_{t=1}^{t_{max}} \left(\frac{\partial E}{\partial w_{ij}^{(HI)}}\right)^{(t)} = \sum_{t=1}^{t_{max}} \left(\frac{\partial E}{\partial \vartheta_i^{(HI)}}\right)^{(t)} x_j^{(t)} \tag{16g}$$

3. Algoritmizácia Elmanovej rekurentnej neurónovej sieti

```
Procedure Inputdata(output: N_input,N_hidden,N_context,N_output,
                           N_pattern,length_pool,x_pool,y_req_pool);
begin readln(data_in,N_pattern);
      readln(data_in,N_input,N_hidden,N_context,N_output);
      for i:=1 to N_pattern do
      begin readln(data_in,length_pool[i]);
            for j:=1 to length_pool[i] do
            begin for k:=1 to N_input do
                  read(data_in,x_pool[i,j,k]);
                  readln(data_in);
            end;
            for k:=1 to N_output do
            read(data_in,y_req_pool[i,k]);
            readln(data_in);
      end;
end;
```

Obrázok 5. Pseudopasclovský kód pre procedúru `Inputdata`, ktorá zabezpečuje vstup základných konštánt metódy a tréningovej množiny, vzhľadom ku ktorej sa uskutoční adaptačný proces. Vstupné údaje sa čítajú zo súboru nazvanom `data_in`, pričom jednotlivé celočíselné konštanty majú tento význam: `N_input` – počet neurónov vstupnej vrstvy, `N_hidden` – počet neurónov skrytej vrstvy, `N_context` – počet neurónovej kontextuálnej vrstvy, `N_pattern` – počet objektov tréningovej množiny. 1-rozmenrý súbor `length_pool` obsahuje dĺžky vstupných reťazcov z tréningovej množiny. 3-rozmerný súbor `x_pool` obsahuje vstupné reťazce z tréningovej množiny a 2-roznerný súbor `y_req_pool` obsahuje požadované výstupné aktivity pre objekty z tréningovej množiny.

```
procedure Coefficients_Ini(input: N_input,N_hidden,N_context,N_output;
                           output: Θ_H,Θ_C,Θ_O, w_HI,w_HC,w_CH,w_OH);
begin readln(RandSeed);
      for i:=1 to N_output do
      begin Θ_O[i]:=1-2*random;
            for j:=1 to N_hidden do
            w_OH[i,j]:=1-2*random;
      end;
      for i:=1 to N_hidden do
      begin Θ_H[i]:=1-2*random;
            for j:=1 to N_input do
            w_HI[i,j]:=1-2*random;
            for j:=1 to N_context do
            w_HC[i,j]:=1-2*random;
      end;
      for i:=1 to N_context do
      begin Θ_C[i]:=1-2*random;
            for j:=1 to N_hidden do
            w_CH[i,j]:=1-2*random;
      end;
end;
```

Obrázok 6. Procedúra `Coefficients_Ini` uskutočňuje náhodnú inicializáciu váhových koeficientov neurónovej siete, pričom generátor náhodných čísel je inicializovaný načítaním celočíselneho parametra `RandSeed.` 1-rozmenrné súbory `Θ_O`, `Θ_H`, `Θ_C` obsahujú prahové koeficienty neurónov z výstupnej, skrytej a kontextuálnej vrstvy, 2-roznerné súbory `w_OH`, `w_HI`, `w_HC`, `w_CH` obsahujú váhové koeficienty pre spoje medzi neurónmi z vrstiev skrytej-výstupnej, výstupnej-skrytej, kontextuálnej-skrytej a skrytej-kontextuálnej.

```
procedure Activities(input: Θ_O,Θ_H,Θ_C,w_OH,w_HI,w_HC,w_CH,x,t_max;
                     output:u,v,y);
begin for i:=1 to N_context do u[1,i]:=0.0;
      for i:=1 to N_hidden do
      begin sum:= Θ_H[i];
            for j:=1 to N_input do
            sum:=sum+w_HI[i,j]*x[1,j];
            v[1,i]:=1/(1+exp(-sum));
      end;
      for i:=1 to N_output do
      begin sum:= Θ_O[i];
            for j:=1 to N_hidden do
            sum:=sum+w_OH[i,j]*v[1,j];
            y[1,i]:=1/(1+exp(-sum));
      end;
      for t:=2 to t_max do
      begin for i:=1 to N_context do
            begin sum:= Θ_C[i];
                  for j:=1 to N_hidden do
                  sum:=sum+w_CH[i,j]*v[t-1,j];
                  u[t,i]:=1/(1+exp(-sum));
            end;
            for i:=1 to N_hidden do
            begin sum:= Θ_H[i];
                  for j:=1 to N_input do
                  sum:=sum+w_HI[i,j]*x[t,j];
                  for j:=1 to N_context do
                  sum:=sum+w_HC[i,j]*u[t,j];
                  v[t,i]:=1/(1+exp(-sum));
            end;
            for i:=1 to N_output do
            begin sum:= Θ_O[i];
                  for j:=1 to N_hidden do
                  sum:=sum+w_OH[i,j]*v[t,j];
                  y[t,i]:=1/(1+exp(-sum));
            end;
      end;
end;
```

Obrázok 7. Procedúra `Activities` počíta aktivity skrytých (`v`), kontextuálnych (`u`) a výstupných (`y`) neurónov pre daný objekt tréningovej množiny špecifikovaný 2-rozmerným súborom `x`. Výpočet je uskutočnený pomocou formúl (6b-d). Prechodová funkcia je špecifikovaná ako štandardná sigmoiálna funkcia $t(\xi) = 1/(1+e^{-\xi})$.

```
procedure Gradient(input:Θ_O,Θ_H,Θ_C,w_OH,w_HI,w_HC,w_CH,
                        x_pool,y_req_pool,N_pattern;
                   output:grad_Θ_O,grad_Θ_H,grad_Θ_C,grad_w_OH,
                          grad_w_HI,grad_w_HC,grad_w_CH);
begin for i:=1 to N_output do
      begin grad_Θ_O[i]:=0.0;
            for j:=1 to N_hidden do
            grad_w_OH[i,j]:=0.0;
      end;
      for i:=1 to N_hidden do
      begin grad_Θ_H[i]:=0.0;
            for j:=1 to N_input do
            grad_w_HI[i,j]:=0.0;
            for j:=1 to N_context do
```

```
            grad_w_HC[i,j]:=0.0;
      end;
      for i:=1 to N_context do
      begin grad_Θ_C[i]:=0.0;
            for j:=1 to N_hidden do
            grad_w_CH[i,j]:=0.0;
      end;
      for index:=1 to N_pattern do
      begin t_max:=length[index];
            for t:=1 to t_max do
            for j:=1 to N_input do
            x[t,j]:=x_pool[index,t,j];
            for j:=1 to N_output do
            y_req[j]:=y_req_pool[index,j];
            Activities(Θ_O,Θ_H,Θ_C,w_OH,w_HI,w_HC,w_CH,x,t_max,u,v,y);
            for i:=1 to N_output do
            grad_Θ_O_aux[t_max,i]:=y[t_max,i]*(1-act_y[t_max,i])*
                                   (y[t_max,i]-y_req[i]);
            for i:=1 to N_hidden do
            begin sum:=0.0;
                  for l:=1 to N_output do
                  sum:=sum+grad_Θ_O_aux[t_max,l]*w_OH[l,i];
                  grad_Θ_H_aux[t_max,i]:=sum*v[t_max,i]*(1-v[t_max,i]);
            end;
            for i:=1 to N_context do
            begin sum:=0.0;
                  for l:=1 to N_hidden do
                  sum:=sum+grad_Θ_H_aux[t_max,l]*w_HC[l,i];
                  grad_Θ_C_aux[t_max,i]:=sum*u[t_max,i]*(1-u[t_max,i]);
            end;
            for t:=t_max-1 downto 1 do
            begin for i:=1 to N_hidden do
                  begin sum:=0.0;
                        for l:=1 to N_context do
                        sum:=sum+grad_Θ_C_aux[t+1,l]*w_CH[l,i];
                        grad_Θ_H_aux[t,i]:=sum*v[t,i]*(1-v[t,i]);
                  end;
                  for i:=1 to N_context do
                  begin sum:=0.0;
                        for l:=1 to N_hidden do
                        sum:=sum+grad_theta_H_aux[t,l]*w_HC[l,i];
                        grad_Θ_C_aux[t,i]:=sum*act_u[t,i]*(1-act_u[t,i]);
                  end;
            end;
            for i:=1 to N_output do
            begin grad_Θ_O_tot[i]:=grad_Θ_O_tot[i]+
                                   grad_Θ_O_aux[t_max,i];
                  for j:=1 to N_hidden do
                  grad_w_OH_tot[i,j]:=grad_w_OH_tot[i,j]+
                  grad_theta_O_aux[t_max,i]*v[t_max,j];
            end;
            for i:=1 to N_hidden do
            begin for t:=1 to t_max do
                  grad_Θ_H[i]:=grad_Θ_H[i]+grad_Θ_H_aux[t,i];
                  for j:=1 to N_context do
                  for t:=2 to t_max do
                  grad_w_HC[i,j]:=grad_w_HC[i,j]+grad_Θ_H_aux[t,i]*
                                  act_u[t,j];
                  for j:=1 to N_input do
```

```
                  for t:=1 to t_max do
                  grad_w_HI[i,j]:=grad_w_HI[i,j]+ grad_theta_H_aux[t,i]*
                                    x[t,j];
            end;
            for i:=1 to N_context do
            begin for t:=2 to length do
                  grad_theta_C[i]:=grad_theta_C[i]+grad_theta_C_aux[t,i];
                  for j:=1 to N_hidden do
                  for t:=2 to t_max do
                  grad_w_CH[i,j]:=grad_w_CH[i,j]+grad_theta_C_aux[t,i]*
                                    v[t-1,j];
            end;
      end;
end;
```

Obrázok 8. Procedúra `Gradient` počíta celkové parciálne derivácie vzhľadom k prahovým a váhovým koeficientom, ktoré sú špecifikované ako sumácie daných výsledkov získaných pre všetky objekty tréningovej množiny. K výpočtu boli použité formuly (12-16).

```
procedure Coefficient_update(input:grad_Θ_O,grad_Θ_H,grad_Θ_C,grad_w_OH,
                                   grad_w_HI,grad_w_HC,grad_w_CH;
                             input:Θ_O,Θ_H,Θ_C,w_OH,w_HI,w_CH);
begin for i:=1 to N_output do
      begin Θ_O[i]:=theta_O[i]-lambda*grad_Θ_O[i];
            for j:=1 to N_hidden do
            w_OH[i,j]:=w_OH[i,j]-λ*grad_w_OH[i,j];
      end;
      for i:=1 to N_hidden do
      begin Θ_H[i]:= Θ_H[i]- λ*grad_Θ_H[i];
            for j:=1 to N_input do
            w_HI[i,j]:=w_HI[i,j]- λ*grad_w_HI [i,j];
            for j:=1 to N_context do
            w_HC[i,j]:=w_HC[i,j]- λ*grad_w_HC [i,j];
      end;
      for i:=1 to N_context do
      begin Θ_C[i]:= Θ_C[i]- λ*grad_Θ_C [i];
            for j:=1 to N_hidden do
            w_CH[i,j]:=w_CH[i,j]- λ*grad_w_CH [i,j];
      end;
end;
```

Obrázok 9. Procedúra `Coefficient_update` uskutočňuje zmenu prahových a váhových koeficientov pri adaptačnom procese prostredníctvom formuly (10), ktorá tvorí základ jednoduchej gradientovej metódy najprudšieho spádu. Proces je veľmi citlivý na vhodné nastavenie rýchlosti učenia λ, obvykle sa volí v intervale od 0.01 do 0.5. V mnohých prípadoch k jeho nastaveniu sa používa jednoduchá „dynamická“ heuristika, že ak v priebehu adaptácie účelová funkcia monotónne klesá, potom rýchlosť učenia sa zväčší $\lambda \leftarrow \lambda * 2$, v opačnom prípade, ak adaptácia diverguje (účelová funkcia sa neustál zväčšuje), rýchlosť učenia sa zmenší $\lambda \leftarrow \lambda / 10$.

```
Procedure Adaptation;
begin  readln(time_max);
       Inputdata(N_input,N_hidden,N_context,N_output,
                 N_pattern,length_pool,x_pool,y_req_pool);
       Coefficients_Ini(N_input,N_hidden,N_context,N_output,
                        Θ_H,Θ_C,Θ_O, w_HI,w_HC,w_CH,w_OH);
       time:=0;
       while time<time_max do
```

```
        begin time:=time+1;
              Gradient(input:Θ_O,Θ_H,Θ_C,w_OH,w_HI,w_HC,w_CH,
                       x_pool,y_req_pool,N_pattern;
                       output:grad_Θ_O,grad_Θ_H,grad_Θ_C,grad_w_OH,
                       grad_w_HC,grad_w_CH);
              Coefficient_update(grad_Θ_O,grad_Θ_H,grad_Θ_C,grad_w_OH,
                                 grad_w_HI,grad_w_HC,grad_w_CH,
                                 Θ_O,Θ_H,Θ_C,w_OH,w_HI,w_CH);
        end;
end;
```

Obrázok 10. Procedura `Adaptation` uskutočňuje adaptačný proces Elamnovej rekurentnej neurónovej siete. Celočíselná konštanta `time_max` určuje maximálny počet krokov adaptácie, obvykle sa špecifikuje v intervale 10.000 až 100.000. Poznamenajme, že rýchlosť konvergencie adaptačného procesu je silne ovplyvnená veľkosťou rýchlosti učenia λ.

Dodatok A

Program v Pascale pre Elmanovu rekurentnú neurónovú sieť

```
program Elman recurrent neural network;
{******************************************************
 *                                                    *
 *    Program for a performance of recurrent neural   *
 *               network of Elman type                *
 *                                                    *
 *                  Vlado Kvasnicka                   *
 *             Department of Mathematics              *
 *            Slovak Technicak University             *
 *            81237 Bratislava,  Slovakia             *
 *            email: kvasnic@cvt.stuba.sk             *
 *                                                    *
 *                   June 21, 1998                    *
 *                                                    *
 *    Note: Formulae for calculation of gradients     *
 *          were verified numerically!                *
 ******************************************************}
const N_input       = 4;
      N_hidden      = 10;
      N_context     = 10;
      N_output      = 1;
      length_max    = 10; {max. length of token string}
      N_pattern_max = 40;
      lambda        = 0.1;
      time_max      = 30000;
      print_f       = 50;

type vector_input   = array [1..N_input] of single;
     vector_hidden  = array [1..N_hidden] of single;
     vector_context = array [1..N_context] of single;
     vector_output  = array [1..N_output] of single;
     matrix_OH      = array [1..N_output,1..N_hidden] of single;
     matrix_HC      = array [1..N_hidden,1..N_context] of single;
     matrix_HI      = array [1..N_hidden,1..N_input] of single;
     matrix_CH      = array [1..N_context,1..N_hidden] of single;
     matrix_act_O   = array [1..length_max,1..N_output] of single;
     matrix_act_H   = array [1..length_max,1..N_hidden] of single;
     matrix_act_C   = array [1..length_max,1..N_context] of single;
     matrix_act_I   = array [1..length_max,1..N_input] of single;
     matrix_pool_x  = array [1..N_pattern_max,1..length_max,1..N_input]
                            of single;
     matrix_pool_y  = array [1..N_pattern_max,1..N_output] of single;
     vector_length  = array [1..N_pattern_max] of integer;


var result,data_in   : text;
    N_pattern        : integer;
    N_test           : integer;
    length           : integer;
    time             : integer;
    theta_O          : vector_output;
    theta_H          : vector_hidden;
    theta_C          : vector_context;
    w_OH             : matrix_OH;
    w_HI             : matrix_HI;
    w_HC             : matrix_HC;
    w_CH             : matrix_CH;
    Pool_act_x       : matrix_pool_x;
    Pool_act_y       : matrix_pool_y;
    Pool_length      : vector_length;
```

```
    Pool_act_x_test  : matrix_pool_x;
    Pool_act_y_test  : matrix_pool_y;
    Pool_length_test : vector_length;
    grad_theta_O_tot : vector_output;
    grad_theta_H_tot : vector_hidden;
    grad_theta_C_tot : vector_context;
    grad_w_OH_tot    : matrix_OH;
    grad_w_HI_tot    : matrix_HI;
    grad_w_HC_tot    : matrix_HC;
    grad_w_CH_tot    : matrix_CH;


procedure Data_input(var N_pattern        : integer;
                     var N_test           : integer;
                     var Pool_length      : vector_length;
                     var Pool_act_x       : Matrix_pool_x;
                     var Pool_act_y       : matrix_pool_y;
                     var Pool_length_test : vector_length;
                     var Pool_act_x_test  : Matrix_pool_x;
                     var Pool_act_y_test  : matrix_pool_y);
var i,j,k : integer;
begin readln(data_in,N_pattern);
      readln(data_in,N_test);
      for i:=1 to N_pattern do
      begin readln(data_in,Pool_length[i]);
            for j:=1 to Pool_length[i] do
            begin for k:=1 to N_input do
                  read(data_in,Pool_act_x[i,j,k]);
                  readln(data_in);
            end;
            for k:=1 to N_output do read(data_in,Pool_act_y[i,k]);
            readln(data_in);
      end;
      for i:=1 to N_test do
      begin readln(data_in,Pool_length_test[i]);
            for j:=1 to Pool_length_test[i] do
            begin for k:=1 to N_input do
                  read(data_in,Pool_act_x_test[i,j,k]);
                  readln(data_in);
            end;
            for k:=1 to N_output do read(data_in,Pool_act_y_test[i,k]);
            readln(data_in);
      end;
end;


procedure Coefficients_Ini(var theta_O : vector_output;
                           var theta_H : vector_hidden;
                           var theta_C : vector_context;
                           var w_OH    : matrix_OH;
                           var w_HI    : matrix_HI;
                           var w_HC    : matrix_HC;
                           var w_CH    : matrix_CH);
var i,j : integer;
begin for i:=1 to N_output do
      begin theta_O[i]:=1-2*random;
            for j:=1 to N_hidden do
            w_OH[i,j]:=1-2*random;
      end;
      for i:=1 to N_hidden do
```

```
      begin theta_H[i]:=1-2*random;
            for j:=1 to N_input do
            w_HI[i,j]:=1-2*random;
            for j:=1 to N_context do
            w_HC[i,j]:=1-2*random;
      end;
      for i:=1 to N_context do
      begin theta_C[i]:=1-2*random;
            for j:=1 to N_hidden do
            w_CH[i,j]:=1-2*random;
      end;
end;


procedure Activities(    theta_O : vector_output;
                         theta_H : vector_hidden;
                         theta_C : vector_context;
                         w_OH    : matrix_OH;
                         w_HI    : matrix_HI;
                         w_HC    : matrix_HC;
                         w_CH    : matrix_CH;
                         length  : integer;
                         act_x   : matrix_act_I;
                     var act_u   : matrix_act_C;
                     var act_v   : matrix_act_H;
                     var act_y   : matrix_act_O);
var i,j,t : integer;
    sum   : real;
begin {t=1, initialization}
      for i:=1 to N_context do act_u[1,i]:=0.0;
      for i:=1 to N_hidden do
      begin sum:=theta_H[i];
            for j:=1 to N_input do
            sum:=sum+w_HI[i,j]*act_x[1,j];
            act_v[1,i]:=1/(1+exp(-sum));
      end;
      for i:=1 to N_output do
      begin sum:=theta_O[i];
            for j:=1 to N_hidden do
            sum:=sum+w_OH[i,j]*act_v[1,j];
            act_y[1,i]:=1/(1+exp(-sum));
      end;
      {t=2,3,...length, iteration}
      for t:=2 to length do
      begin for i:=1 to N_context do
            begin sum:=theta_C[i];
                  for j:=1 to N_hidden do
                  sum:=sum+w_CH[i,j]*act_v[t-1,j];
                  act_u[t,i]:=1/(1+exp(-sum));
            end;
            for i:=1 to N_hidden do
            begin sum:=theta_H[i];
                  for j:=1 to N_input do
                  sum:=sum+w_HI[i,j]*act_x[t,j];
                  for j:=1 to N_context do
                  sum:=sum+w_HC[i,j]*act_u[t,j];
                  act_v[t,i]:=1/(1+exp(-sum));
            end;
            for i:=1 to N_output do
            begin sum:=theta_O[i];
                  for j:=1 to N_hidden do
```

```
                  sum:=sum+w_OH[i,j]*act_v[t,j];
                  act_y[t,i]:=1/(1+exp(-sum));
            end;
      end;
      {all activities are recurrently calculated}
end;

procedure Gradient_tot(      theta_O          : vector_output;
                             theta_H          : vector_hidden;
                             theta_C          : vector_context;
                             w_OH             : matrix_OH;
                             w_HI             : matrix_HI;
                             w_HC             : matrix_HC;
                             w_CH             : matrix_CH;
                         var grad_theta_O_tot : vector_output;
                         var grad_theta_H_tot : vector_hidden;
                         var grad_theta_C_tot : vector_context;
                         var grad_w_OH_tot    : matrix_OH;
                         var grad_w_HI_tot    : matrix_HI;
                         var grad_w_HC_tot    : matrix_HC;
                         var grad_w_CH_tot    : matrix_CH);

var i,j,l,index,t    : integer;
    length           : integer;
    sum              : real;
    grad_theta_O_aux : matrix_act_O;
    grad_theta_H_aux : matrix_act_H;
    grad_theta_C_aux : matrix_act_C;
    act_x   : matrix_act_I;
    act_u   : matrix_act_C;
    act_v   : matrix_act_H;
    act_y   : matrix_act_O;
    act_y_req : vector_output;
begin {wiping of total gradients}
      for i:=1 to N_output do
      begin grad_theta_O_tot[i]:=0.0;
            for j:=1 to N_hidden do
            grad_w_OH_tot[i,j]:=0.0;
      end;
      for i:=1 to N_hidden do
      begin grad_theta_H_tot[i]:=0.0;
            for j:=1 to N_input do
            grad_w_HI_tot[i,j]:=0.0;
            for j:=1 to N_context do
            grad_w_HC_tot[i,j]:=0.0;
      end;
      for i:=1 to N_context do
      begin grad_theta_C_tot[i]:=0.0;
            for j:=1 to N_hidden do
            grad_w_CH_tot[i,j]:=0.0;
      end;

      {calculation of partial derivatives for all training patterns}
      for index:=1 to N_pattern do
      begin length:=Pool_length[index];
            for t:=1 to length do
            for j:=1 to N_input do
            act_x[t,j]:=Pool_act_x[index,t,j];
            for j:=1 to N_output do
            act_y_req[j]:=Pool_act_y[index,j];
```

```
Activities(theta_O,theta_H,theta_C,w_OH,w_HI,w_HC,w_CH,length,
           act_x,act_u,act_v,act_y);

{t=length, initialization of backpropagation}
for i:=1 to N_output do
grad_theta_O_aux[length,i]:=act_y[length,i]*
                            (1-act_y[length,i])*
                            (act_y[length,i]-act_y_req[i]);
for i:=1 to N_hidden do
begin sum:=0.0;
      for l:=1 to N_output do
      sum:=sum+grad_theta_O_aux[length,l]*w_OH[l,i];
      grad_theta_H_aux[length,i]:=sum*act_v[length,i]*
                                  (1-act_v[length,i]);
end;
for i:=1 to N_context do
begin sum:=0.0;
      for l:=1 to N_hidden do
      sum:=sum+grad_theta_H_aux[length,l]*w_HC[l,i];
      grad_theta_C_aux[length,i]:=sum*act_u[length,i]*
                                  (1-act_u[length,i]);
end;

{t=length-1,...,1, iteration of backpropagation}
for t:=length-1 downto 1 do
begin for i:=1 to N_hidden do
      begin sum:=0.0;
            for l:=1 to N_context do
            sum:=sum+grad_theta_C_aux[t+1,l]*w_CH[l,i];
            grad_theta_H_aux[t,i]:=sum*act_v[t,i]*
                                   (1-act_v[t,i]);
      end;
      for i:=1 to N_context do
      begin sum:=0.0;
            for l:=1 to N_hidden do
            sum:=sum+grad_theta_H_aux[t,l]*w_HC[l,i];
            grad_theta_C_aux[t,i]:=sum*act_u[t,i]*
                                   (1-act_u[t,i]);
      end;
end;

for i:=1 to N_output do
begin grad_theta_O_tot[i]:=grad_theta_O_tot[i]+
                           grad_theta_O_aux[length,i];
      for j:=1 to N_hidden do
      grad_w_OH_tot[i,j]:=grad_w_OH_tot[i,j]+
      grad_theta_O_aux[length,i]*act_v[length,j];
end;
for i:=1 to N_hidden do
begin for t:=1 to length do
      grad_theta_H_tot[i]:=grad_theta_H_tot[i]+
                           grad_theta_H_aux[t,i];
      for j:=1 to N_context do
      for t:=2 to length do
      grad_w_HC_tot[i,j]:=grad_w_HC_tot[i,j]+
      grad_theta_H_aux[t,i]*act_u[t,j];
      for j:=1 to N_input do
      for t:=1 to length do
      grad_w_HI_tot[i,j]:=grad_w_HI_tot[i,j]+
      grad_theta_H_aux[t,i]*act_x[t,j];
end;
```

```
            for i:=1 to N_context do
            begin for t:=2 to length do
                  grad_theta_C_tot[i]:=grad_theta_C_tot[i]+
                  grad_theta_C_aux[t,i];
                  for j:=1 to N_hidden do
                  for t:=2 to length do
                  grad_w_CH_tot[i,j]:=grad_w_CH_tot[i,j]+
                  grad_theta_C_aux[t,i]*act_v[t-1,j];
            end;
      end;
end;

procedure Coefficient_update(var theta_O          : vector_output;
                             var theta_H          : vector_hidden;
                             var theta_C          : vector_context;
                             var w_OH             : matrix_OH;
                             var w_HI             : matrix_HI;
                             var w_HC             : matrix_HC;
                             var w_CH             : matrix_CH;
                                 grad_theta_O_tot : vector_output;
                                 grad_theta_H_tot : vector_hidden;
                                 grad_theta_C_tot : vector_context;
                                 grad_w_OH_tot    : matrix_OH;
                                 grad_w_HI_tot    : matrix_HI;
                                 grad_w_HC_tot    : matrix_HC;
                                 grad_w_CH_tot    : matrix_CH);
var i,j : integer;
begin for i:=1 to N_output do
      begin theta_O[i]:=theta_O[i]-lambda*grad_theta_O_tot[i];
            for j:=1 to N_hidden do
            w_OH[i,j]:=w_OH[i,j]-lambda*grad_w_OH_tot[i,j];
      end;
      for i:=1 to N_hidden do
      begin theta_H[i]:=theta_H[i]-lambda*grad_theta_H_tot[i];
            for j:=1 to N_input do
            w_HI[i,j]:=w_HI[i,j]-lambda*grad_w_HI_tot[i,j];
            for j:=1 to N_context do
            w_HC[i,j]:=w_HC[i,j]-lambda*grad_w_HC_tot[i,j];
      end;
      for i:=1 to N_context do
      begin theta_C[i]:=theta_C[i]-lambda*grad_theta_C_tot[i];
            for j:=1 to N_hidden do
            w_CH[i,j]:=w_CH[i,j]-lambda*grad_w_CH_tot[i,j];
      end;
end;


procedure Print_check(time : integer);
var index,i,j,t : integer;
    length      : integer;
    sum,E_train : real;
    E_test      : real;
    act_x       : matrix_act_I;
    act_u       : matrix_act_C;
    act_v       : matrix_act_H;
    act_y       : matrix_act_O;
    act_y_req   : vector_output;
begin writeln('iteration # ',time);
      sum:=0.0;
      for index:=1 to N_pattern do
      begin length:=Pool_length[index];
```

```
            for t:=1 to length do
            for j:=1 to N_input do
            act_x[t,j]:=Pool_act_x[index,t,j];
            for j:=1 to N_output do
            act_y_req[j]:=Pool_act_y[index,j];

            Activities(theta_O,theta_H,theta_C,w_OH,w_HI,w_HC,w_CH,length,
                       act_x,act_u,act_v,act_y);

            for i:=1 to N_output do
            sum:=sum+sqr(act_y[length,i]-act_y_req[i]);

            {write('pattern #',index:2,' y_cal=(');
            for i:=1 to N_output do
            begin write(act_y[length,i]:4:2);
                  if i<N_output then write(',');
            end;
            write('   y_req=(');
            for i:=1 to N_output do
            begin write(act_y_req[i]:4:2);
                  if i<N_output then write(',');
            end;
            writeln(')');}
      end;
      E_train:=0.5*sum;

      sum:=0.0;
      for index:=1 to N_test do
      begin length:=Pool_length_test[index];
            for t:=1 to length do
            for j:=1 to N_input do
            act_x[t,j]:=Pool_act_x_test[index,t,j];
            for j:=1 to N_output do
            act_y_req[j]:=Pool_act_y_test[index,j];

            Activities(theta_O,theta_H,theta_C,w_OH,w_HI,w_HC,w_CH,length,
                       act_x,act_u,act_v,act_y);

            for i:=1 to N_output do
            sum:=sum+sqr(act_y[length,i]-act_y_req[i]);

            write('pattern #',index:2,' y_cal=(');
            for i:=1 to N_output do
            begin write(act_y[length,i]:4:2);
                  if i<N_output then write(',');
            end;
            write(')');
            write('  y_req=(');
            for i:=1 to N_output do
            begin write(act_y_req[i]:4:2);
                  if i<N_output then write(',');
            end;
            writeln(')');
      end;
      E_test:=0.5*sum;

      writeln('E_train=',E_train:7:4);
      writeln('E_test =',E_test :7:4);
      writeln;
      writeln(result,time,'  ',E_train:7:4,'  ',E_test:7:4)
end;
```

```
begin  write('RandSeed='); readln(RandSeed);
       assign(result,'result'); rewrite(result);
       assign(data_in,'data_in'); reset(data_in);

       Data_input(N_pattern,N_test,Pool_length,Pool_act_x,Pool_act_y,
                  Pool_length_test,Pool_act_x_test,Pool_act_y_test);
       Coefficients_Ini(theta_O,theta_H,theta_C,
                        w_OH,w_HI,w_HC,w_CH);
       time:=0;
       while time<time_max do
       begin time:=time+1;
             Gradient_tot(theta_O,theta_H,theta_C,w_OH,w_HI,w_HC,w_CH,
                          grad_theta_O_tot,grad_theta_H_tot,
                          grad_theta_C_tot,
                          grad_w_OH_tot,grad_w_HI_tot,grad_w_HC_tot,
                          grad_w_CH_tot);

             Coefficient_update(theta_O,theta_H,theta_C,
                                w_OH,w_HI,w_HC,w_CH,
                                grad_theta_O_tot,grad_theta_H_tot,
                                grad_theta_C_tot,
                                grad_w_OH_tot,grad_w_HI_tot,grad_w_HC_tot,
                                grad_w_CH_tot);

             if (time=1) or (time mod print_f= 0) then Print_check(time);
       end;
       close(result);
end.
```

Dodatok B

Data_in (vstupné dáta)

```
20 N_train
10 N_test
3   train#1   abb 0
1 1 0 0 a
0 0 1 1 b
0 0 1 1 b
0   y_req
3   train#2   bab 1
0 0 1 1 b
1 1 0 0 a
0 0 1 1 b
1   y_req
4   train#3   aabb 0
1 1 0 0 a
1 1 0 0 a
0 0 1 1 b
0 0 1 1 b
0   y_req
5   train#4   babaa 1
0 0 1 1 b
1 1 0 0 a
0 0 1 1 b
1 1 0 0 a
1 1 0 0 a
1   y_req
3   train#5   aba 0
1 1 0 0 a
0 0 1 1 b
1 1 0 0 a
0   y_req
6   train#6  bbbaba 1
0 0 1 1 b
0 0 1 1 b
0 0 1 1 b
1 1 0 0 a
0 0 1 1 b
1 1 0 0 a
1   y_req
6   train#7   aaabab 1
1 1 0 0 a
1 1 0 0 a
1 1 0 0 a
0 0 1 1 b
1 1 0 0 a
0 0 1 1 b
1   y_req
6   train#8   bbbaaa 0
0 0 1 1 b
0 0 1 1 b
0 0 1 1 b
1 1 0 0 a
```

```
1 1 0 0 a
1 1 0 0 a
0   y_req
6   train#9  aaabba 0
1 1 0 0 a
1 1 0 0 a
1 1 0 0 a
0 0 1 1 b
0 0 1 1 b
1 1 0 0 a
0   y_req
5   train#10  bbabb 1
0 0 1 1 b
0 0 1 1 b
1 1 0 0 a
0 0 1 1 b
0 0 1 1 b
1   y_req
6   train#11  ababaa 1
1 1 0 0 a
0 0 1 1 b
1 1 0 0 a
0 0 1 1 b
1 1 0 0 a
1 1 0 0 a
1   y_req
6   train#12  aabbaa 0
1 1 0 0 a
1 1 0 0 a
0 0 1 1 b
0 0 1 1 b
1 1 0 0 a
1 1 0 0 a
0   y_req
6   train#13  abaaaa 0
1 1 0 0 a
0 0 1 1 b
1 1 0 0 a
1 1 0 0 a
1 1 0 0 a
1 1 0 0 a
0   y_req
6   train#14  bbaabb 0
0 0 1 1 b
0 0 1 1 b
1 1 0 0 a
1 1 0 0 a
0 0 1 1 b
0 0 1 1 b
0   y_req
5   train#15  aabab  1
```

```
1 1 0 0 a
1 1 0 0 a
0 0 1 1 b
1 1 0 0 a
0 0 1 1 b
1   y_req
3   train#16 bba 0
0 0 1 1 b
0 0 1 1 b
1 1 0 0 a
0   y_req
4   train#17 bbaa  0
0 0 1 1 b
0 0 1 1 b
1 1 0 0 a
1 1 0 0 a
0   y_req
5   train#18 aabab  1
1 1 0 0 a
1 1 0 0 a
0 0 1 1 b
1 1 0 0 a
0 0 1 1 b
1   y_req
6   train#19 ababaa  1
1 1 0 0 a
0 0 1 1 b
1 1 0 0 a
0 0 1 1 b
1 1 0 0 a
1 1 0 0 a
1   y_req
5   train#20 bbaab 0
0 0 1 1 b
0 0 1 1 b
1 1 0 0 a
1 1 0 0 a
0 0 1 1 b
0   y_req
7   test#1  bababab 1
0 0 1 1 b
1 1 0 0 a
0 0 1 1 b
1 1 0 0 a
0 0 1 1 b
1 1 0 0 a
0 0 1 1 b
1   y_req
6   test#2  bbbbab  1
0 0 1 1 b
0 0 1 1 b
```

```
0 0 1 1 b
0 0 1 1 b
1 1 0 0 a
0 0 1 1 b
1   y_req
6   test#3  aaabbb  0
1 1 0 0 a
1 1 0 0 a
1 1 0 0 a
0 0 1 1 b
0 0 1 1 b
0 0 1 1 b
0   y_req
7   test#4  abaabba  0
1 1 0 0 a
0 0 1 1 b
1 1 0 0 a
1 1 0 0 a
0 0 1 1 b
0 0 1 1 b
1 1 0 0 a
0   y_req
9   test#5  aaababaaa 1
1 1 0 0 a
1 1 0 0 a
1 1 0 0 a
0 0 1 1 b
1 1 0 0 a
0 0 1 1 b
1 1 0 0 a
1 1 0 0 a
1 1 0 0 a
1   y_req
7   test#6  ababaab  1
1 1 0 0 a
0 0 1 1 b
1 1 0 0 a
0 0 1 1 b
1 1 0 0 a
1 1 0 0 a
0 0 1 1 b
1   y_req
6   test#7  abbbaa  0
1 1 0 0 a
0 0 1 1 b
0 0 1 1 b
0 0 1 1 b
1 1 0 0 a
1 1 0 0 a
0   y_req
5   test#8  abbaa 0
```

```
1 1 0 0 a
0 0 1 1 b
0 0 1 1 b
1 1 0 0 a
1 1 0 0 a
0   y_req
8   test#9  bbbaabba  0
0 0 1 1 b
0 0 1 1 b
0 0 1 1 b
1 1 0 0 a
1 1 0 0 a
0 0 1 1 b
0 0 1 1 b
1 1 0 0 a
0   y_req
8   test#10  aababbbb  1
1 1 0 0 a
1 1 0 0 a
0 0 1 1 b
1 1 0 0 a
0 0 1 1 b
0 0 1 1 b
0 0 1 1 b
0 0 1 1 b
1   y_req
```

Dodatok C

Numerické výsledky

Model calculations performed by the Elman's recurrent neural network

Training set

No.	Token string	y_{req}
1	abb	0
2	bab	1
3	aabb	0
4	babaa	1
5	aba	0
6	bbbaba	1
7	aaabab	1
8	bbbaaa	0
9	aaabba	0
10	bbabb	1
11	ababaa	1
12	aabbaa	0
13	abaaaa	0
14	bbaabb	0
15	aabab	1
16	bba	0
17	bbaa	0
18	aabab	1
19	ababaa	1
20	bbaab	0

Testing set

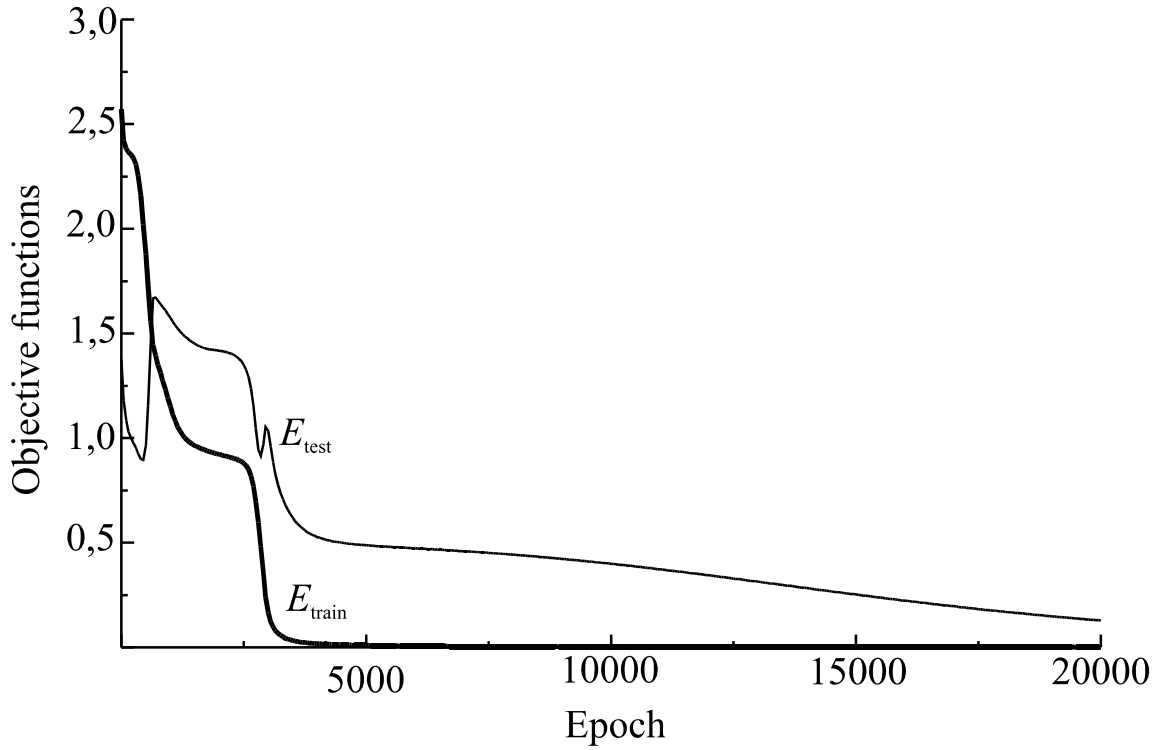
No.	Token string	y_{req}
1	bababab	1
2	bbbbab	1
3	aaabbb	0
4	abaabba	0
5	aaababaaa	1
6	ababaab	1
7	abbbaa	0
8	abbaa	0
9	bbbaabba	0
10	aababbbb	1

$$y_{calc}(\alpha) = \begin{cases} 0\,(\text{if } \boldsymbol{bab} \notin \alpha) \\ 1\,(\text{if } \boldsymbol{bab} \in \alpha) \end{cases}$$

Elman's recurrent neural network is adapted with respect to the training set, then its adapted form is **verified** by patterns (strings) from the testing set.

$$E_{train} = \frac{1}{2} \sum_{\alpha \in A_{train}} \left(y_{calc} - y_{req} \right)^2$$

$$E_{test} = \frac{1}{2} \sum_{\alpha \in A_{test}} \left(y_{calc} - y_{req} \right)^2$$



Classification of testing patterns by adapted Elman's recurrent neural network

pattern	y_{calc}	y_{req}
bababab	**0.60**	1
bbbbab	0.94	1
aaabbb	0.01	0
abaabba	0.01	0
aaababaaa	0.99	1
ababaab	0.99	1
abbbaa	0.01	0
abbaa	0.01	0
bbbaabba	0.01	0
aababbbb	0.99	1

Conclusion: Elman's recurrent neural network is able to classify (almost correctly) token strings of variable lengths whether these strings contain a substring ***bab***

Dodatok D

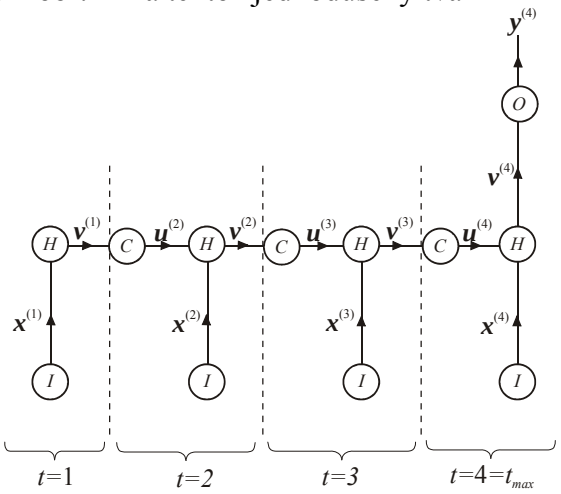
Návrh projektov

Projekt 1.

Upraviť vzorce pre výpočet parciálnych derivácií účelovej funkcie *E*, ktorá má tvar (porovnaj s (8))

$$E = \frac{1}{2}\left(\boldsymbol{y}^{(t_{max})} - \boldsymbol{y}_{req} \right)^2$$

Rozvinutá neurónová sieť z obr. 2 má tento zjednodušený tvar

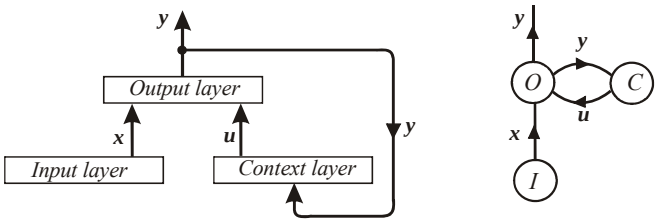


Projekt 2.

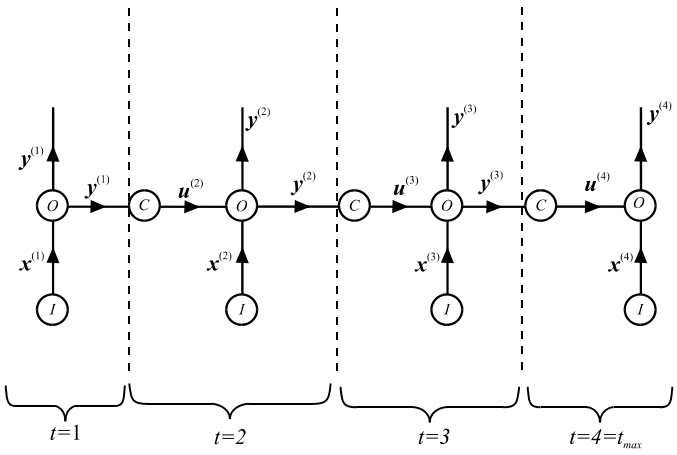
Zostrojte vzorce pre výpočet parciálnych derivácií účelovej funkcie *E* definovanej

$$E = \frac{1}{2}\left(\boldsymbol{y}^{(t_{max})} - \boldsymbol{y}_{req} \right)^2$$

pre rekurentný perceptrón



kde neurónová sieť v rozvinutej forme má tvar

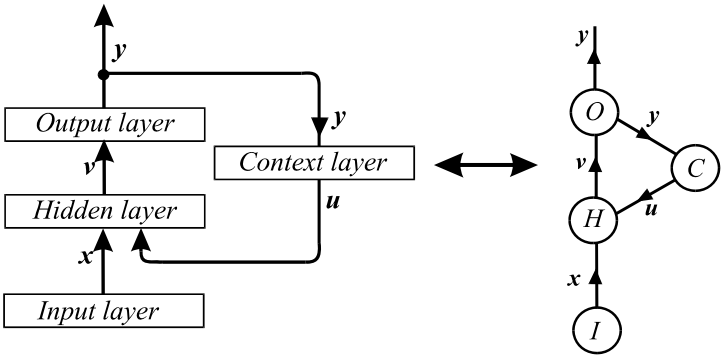


Projekt 3.

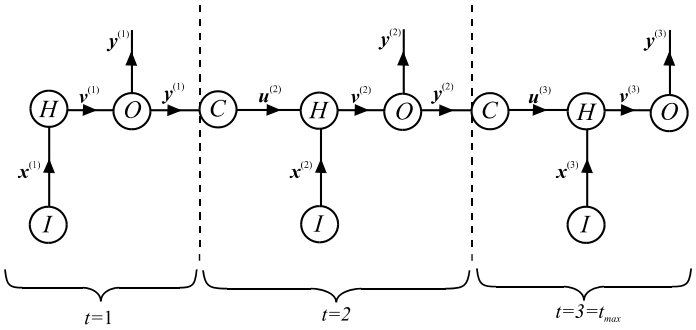
Zostrojte vzorce pre výpočet parciálnych derivácií účelovej funkcie

$$E = \frac{1}{2}\left(\boldsymbol{y}^{(t_{max})} - \boldsymbol{y}_{req} \right)^2$$

pre Jordanovu rekuretnú neurónovú sieť



kde rozvinutá neurónová sieť má tvar



Projekt 4.

Navrhnite jednoduchú gramatiku pre generovanie reťazcov nad abecedou $A = \{a,b,c,d\}$. Tréningovú množinu vytvorte tak, že 2/3 objektov patrí do jazyka generovaného danou gramatikou a 1/3 objektov nepatrí do tohto jazyka. Podobným spôsobom vytvorte aj testovaciu množinu.Použite Elmanovu alebo Jordanovu rekurentnú neurónovú sieť ku klasifikácii reťazcov, či patria alebo nepatria do jazyka.