

Aplikace neutronových sítí

Rekurentní sítě

Sekvenční data

- Text
- Audio
- Video
- Počasí
- Burzy
- ...

Automatické generování textu

VIOLA:

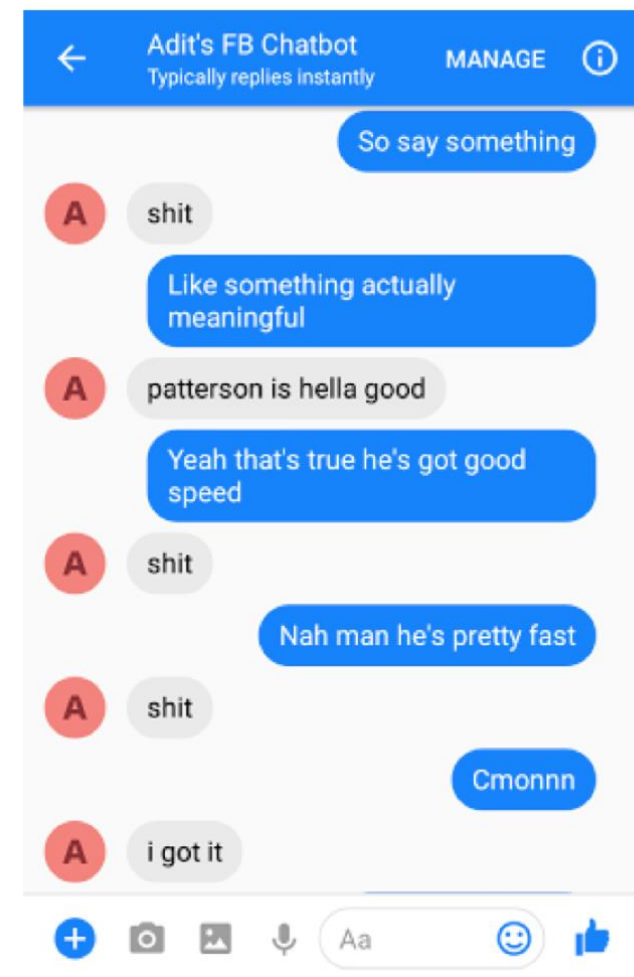
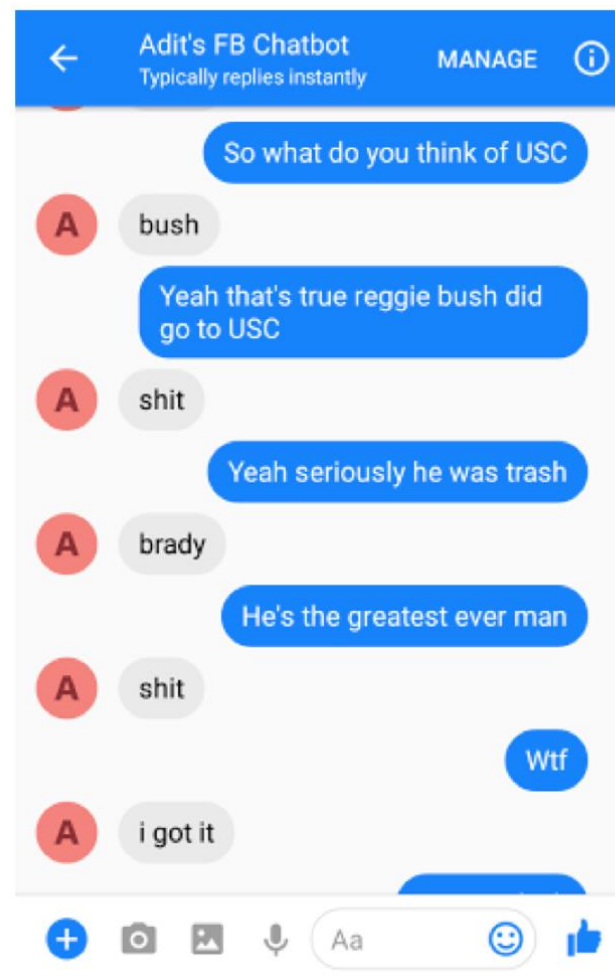
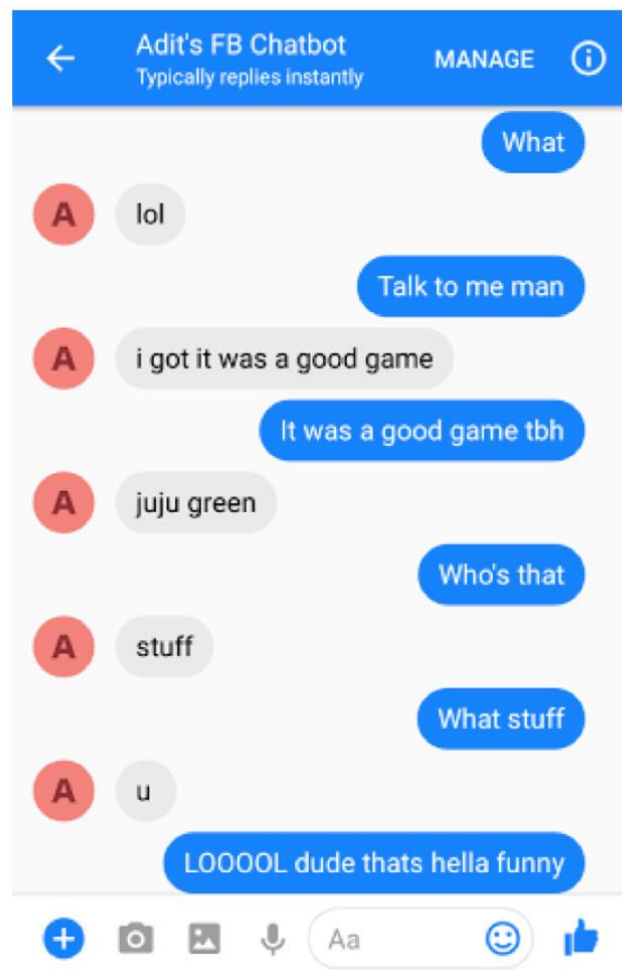
Why, Salisbury must find his flesh and thought
That which I am not apt, not a man and in fire,
To show the reining of the raven and the wars
To grace my hand reproach within, and not a fair are hand,
That Caesar and my goodly father's world;
When I was heaven of presence and our fleets,
We spare with hours, but cut thy council I am great,
Murdered and by thy master's ready there
My power to give thee but so much as hell:
Some service in the noble bondman here,
Would show him to her wine.

KING LEAR:

O, if you were a feeble sight, the courtesy of your law,
Your sight and several breath, will wear the gods
With his heads, and my hands are wonder'd at the deeds,
So drop upon your lordship's head, and your opinion
Shall be against your honour.

další příklady: <http://karpathy.github.io/2015/05/21/rnn-effectiveness/>

Chatbot



obrázek: <https://dzone.com/articles/how-i-used-deep-learning-to-train-a-chatbot-to-tal>

Klasifikace textu, analýza sentimentu

úloha: je text pozitivním nebo negativním komentářem?

DaViD'82 ★★★★★

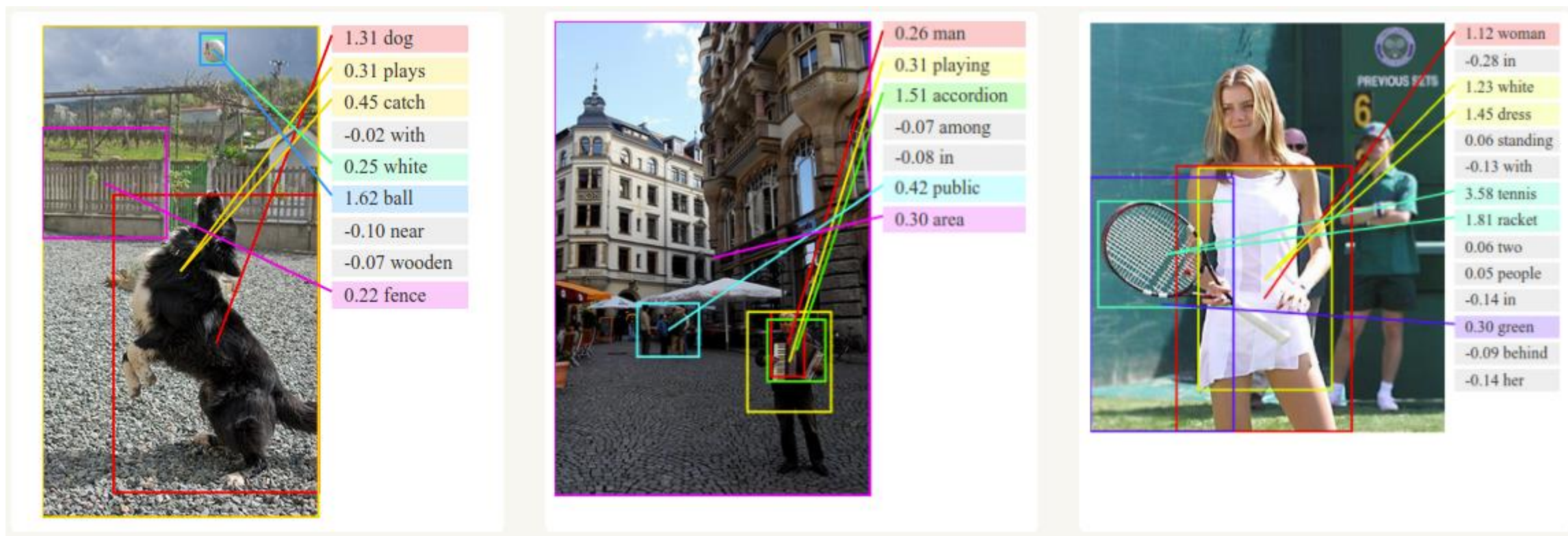
[všechny komentáře uživatele](#) / [🔗](#)

Star Wars by Charles Dickens. Škoda, že na každý dechberoucí olejmalebný výjev (a že jich tu pěkných pár je) a na každou sekvenci aspirující na to vůbec nejlepší z celého universa (emocemi, osudovostí, choreografií i nápady) připadá nějaká linie, která nikam nevede a jen kupí vatu na hromadu jiné vaty, postava která nemá žádné opodstatnění pro existenci či momenty sloužící čistě jako merchandisingová vsuvka „běžte a kupujte“. Navíc to nefunguje jako součást ságy; na žádnou (opravdu žádnou) z otázek z minulého dílu to nezodpoví, ono to většinu pro jistotu rovnou ignoruje. Ovšem čistě sama za sebe je osmička více než solidní popcornový blockbuster; zda to však v případě Star Wars stačí, je věc zcela jiná.

(15.12.2017)

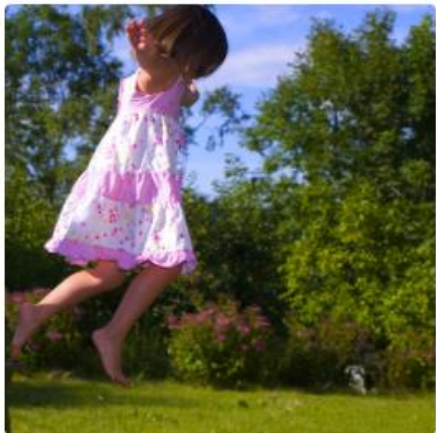
Tagování obrázků

úloha: vygenerovat text, který popisuje obrázek



obrázek: <https://cs.stanford.edu/people/karpathy/deepimagesent/>

Tagování obrázků



"girl in pink dress is jumping in air."



"black and white dog jumps over bar."



"young girl in pink shirt is swinging on swing."



"man in blue wetsuit is surfing on wave."



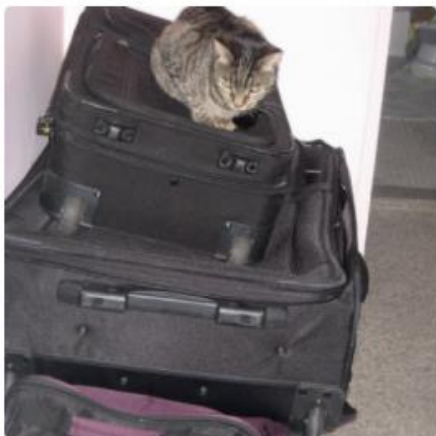
"little girl is eating piece of cake."



"baseball player is throwing ball in game."



"woman is holding bunch of bananas."



"black cat is sitting on top of suitcase."

obrázek: <https://cs.stanford.edu/people/karpathy/deepimagesent/>

Automatický překlad textu



obrázek: <http://blog.webcertain.com/machine-translation-technology-the-search-engine-takeover/18/02/2015/>

Automatický překlad textu



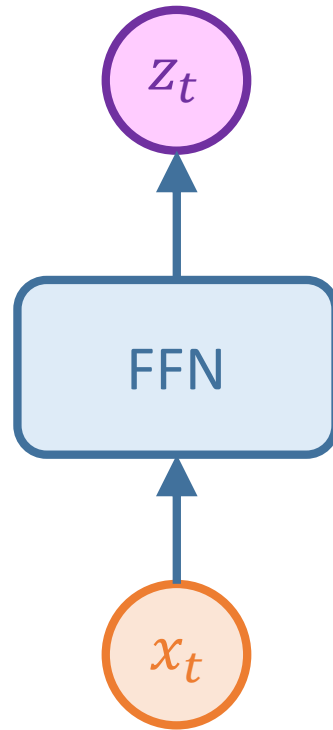
“Ho-ho, teď mám strojní pušku!”

obrázek: <http://g.cz/8-dukazu-ze-prekladatele-ceskych-dabingu-maji-modrou-knizku/#>

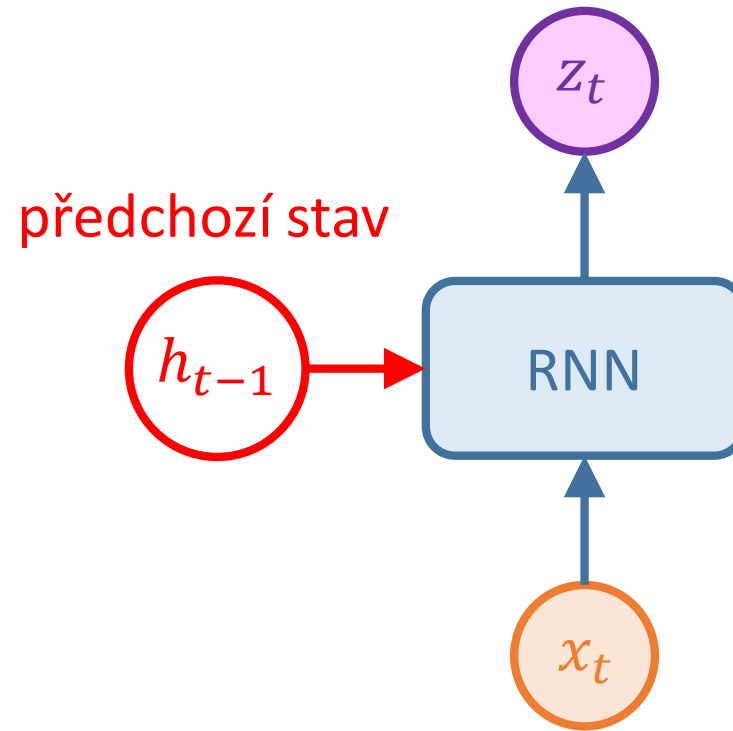
Standardní vs rekurentní síť

výstupem např. softmax (pravděpodobnosti pro jednotlivé třídy)

$$z_t = f(x_t)$$



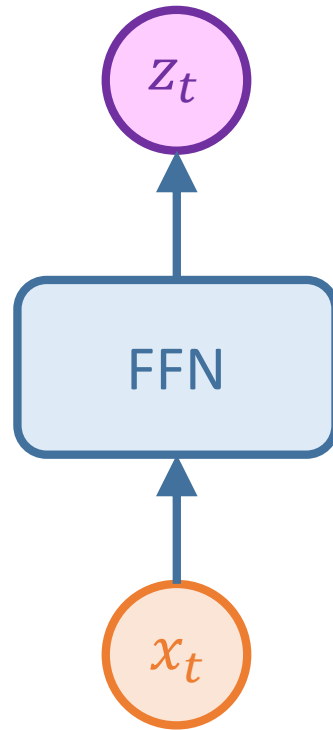
$$z_t = f(x_t, h_{t-1})$$



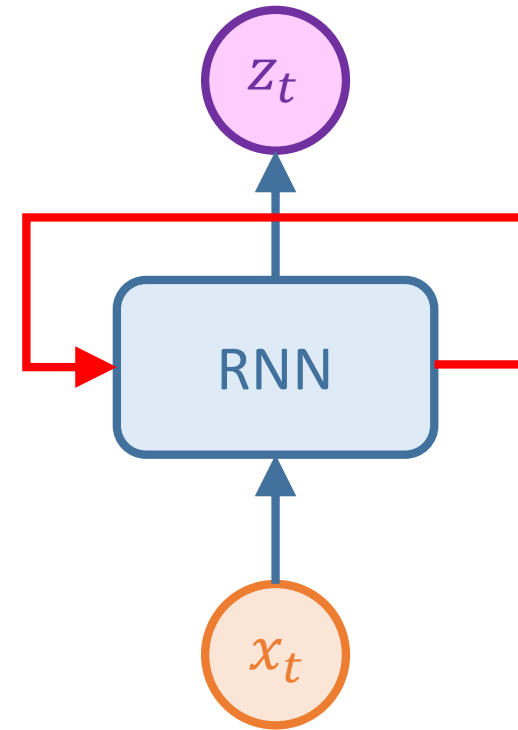
Standardní vs rekurentní síť

výstupem např. softmax (pravděpodobnosti pro jednotlivé třídy)

$$z_t = f(x_t)$$



$$z_t = f(x_t, h_{t-1})$$



rekurence!

$$h_t = g(h_{t-1}, x_t)$$

Vnitřní stav rekurentní sítě

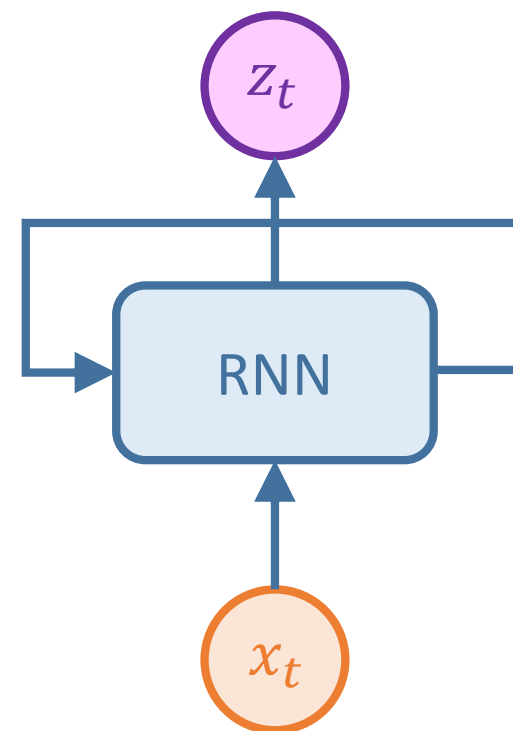
vše jsou vektory!

nový stav v čase t

$$h_t = g(h_{t-1}, x_t)$$

vstup v čase t

minulý stav v čase $t - 1$



Výstupy rekurentní síť

- Rekurentní síť má v každém kroku dva vstupy a **produkuje dva výstupy**

- Nový vnitřní stav

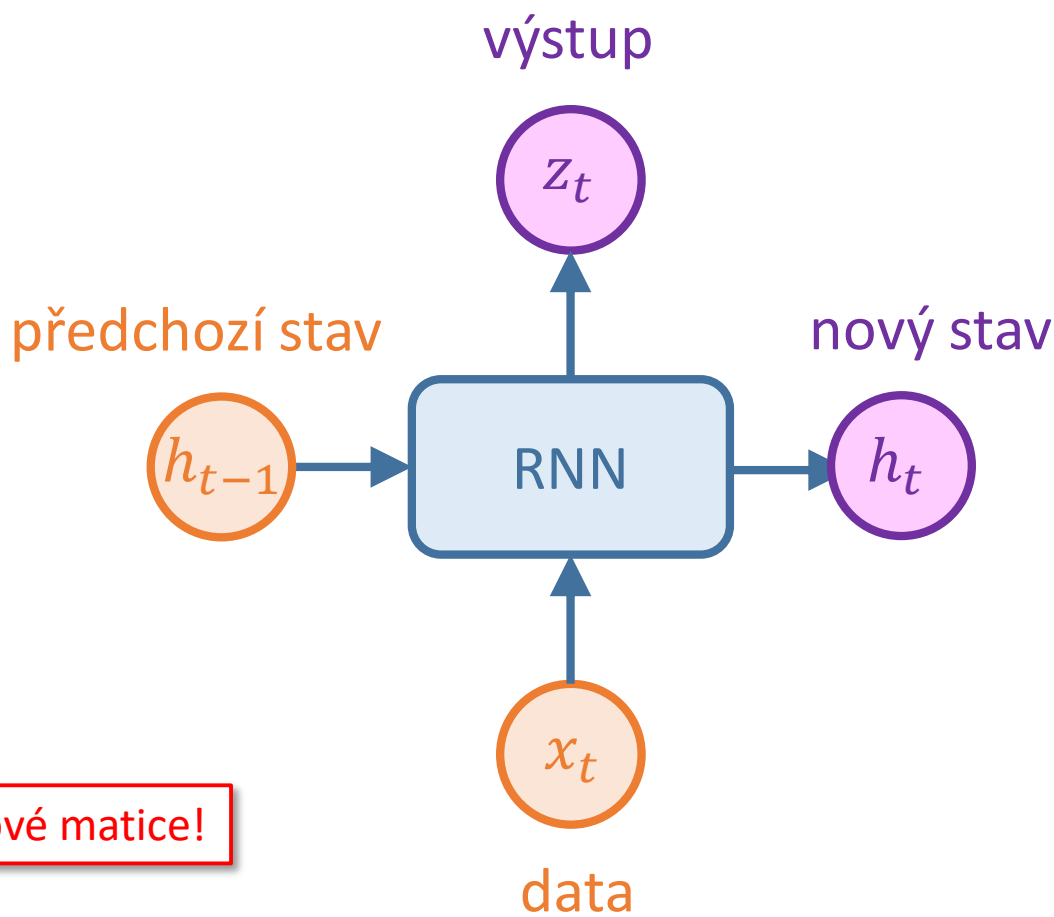
$$h_t = g(W^{xh}x_t + W^{hh}h_{t-1})$$

- Výstup

$$z_t = f(W^{hy}h_t)$$

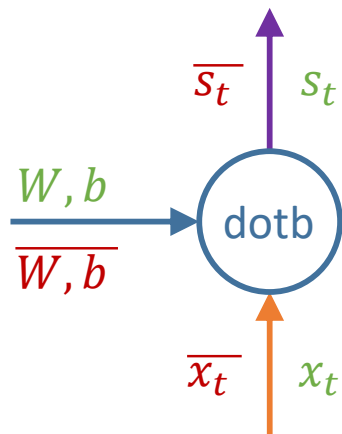
tři různé váhové matice!

zjednodušený výpočetní graf

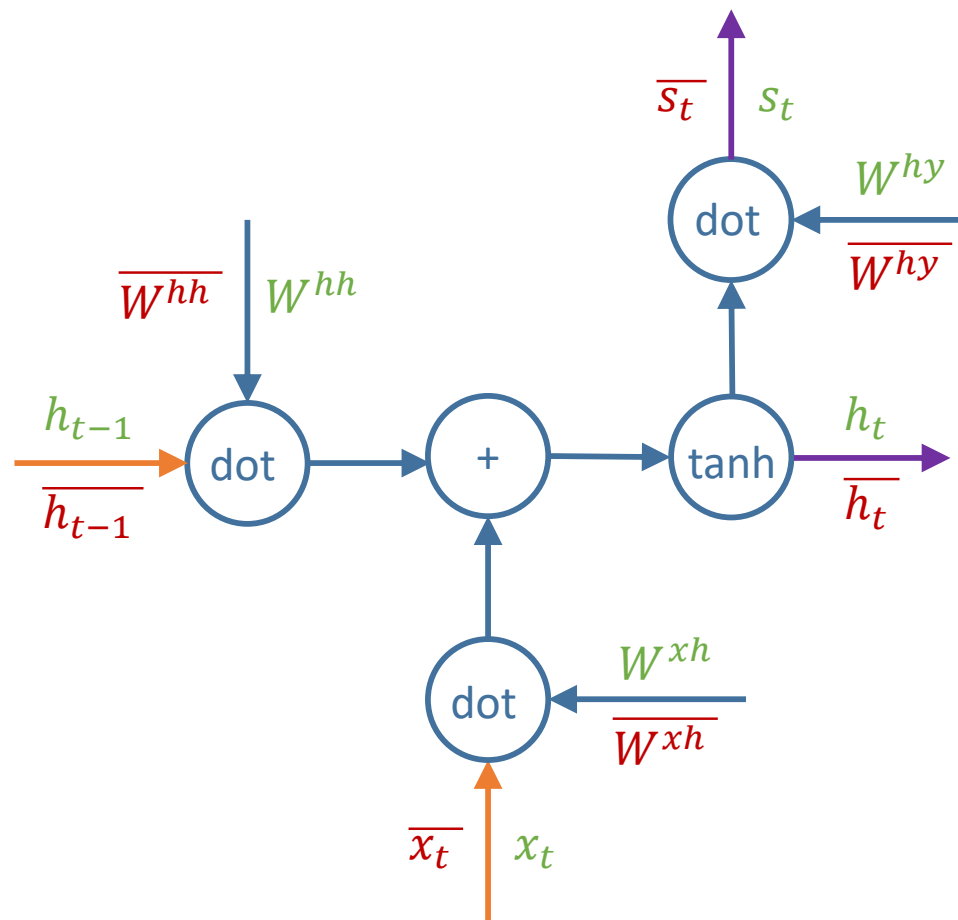


Podrobný výpočetní graf jednoho kroku rekurentní sítě

standardní dopředná síť:



rekurentní síť:



Znakový jazykový model založený na RNN

- Rekurentní sítě vhodné pro sekvenční data
- Lze modelovat např. jazyk → jazykový model
 - zachycuje statistiku jazyka, např. sekvence “sekat trávu” je pravděpodobnější než “sekat krávu”
 - různé úrovně: slova, slabiky, znaky
- Použijeme znakový model na úrovni písmen → “slovníkem” je abeceda + interpunkce
- Vstup: vstupní znak
- Výstup: skóre/pravděpodobnost pro každý znak, že následovat má právě on
- Není to stejné jako kdybychom vzali sekvenci n znaků a snažili se predikovat $(n + 1)$ -tý; zde využíváme rekurenci: RNN má vnitřní stav, který si “pamatuje”

RNN jazykový model: slovo "hello"

správné znaky: "e"

výstupní skóre:
h -0.9
e 2.5
l -1.9
o -3.2

W^{hy}

W^{hh}

skrytý stav:

0.5
-0.1
0.2

W^{xh}

vstupní vektor:

h 1
e 0
l 0
o 0

vstupní znaky: "h"

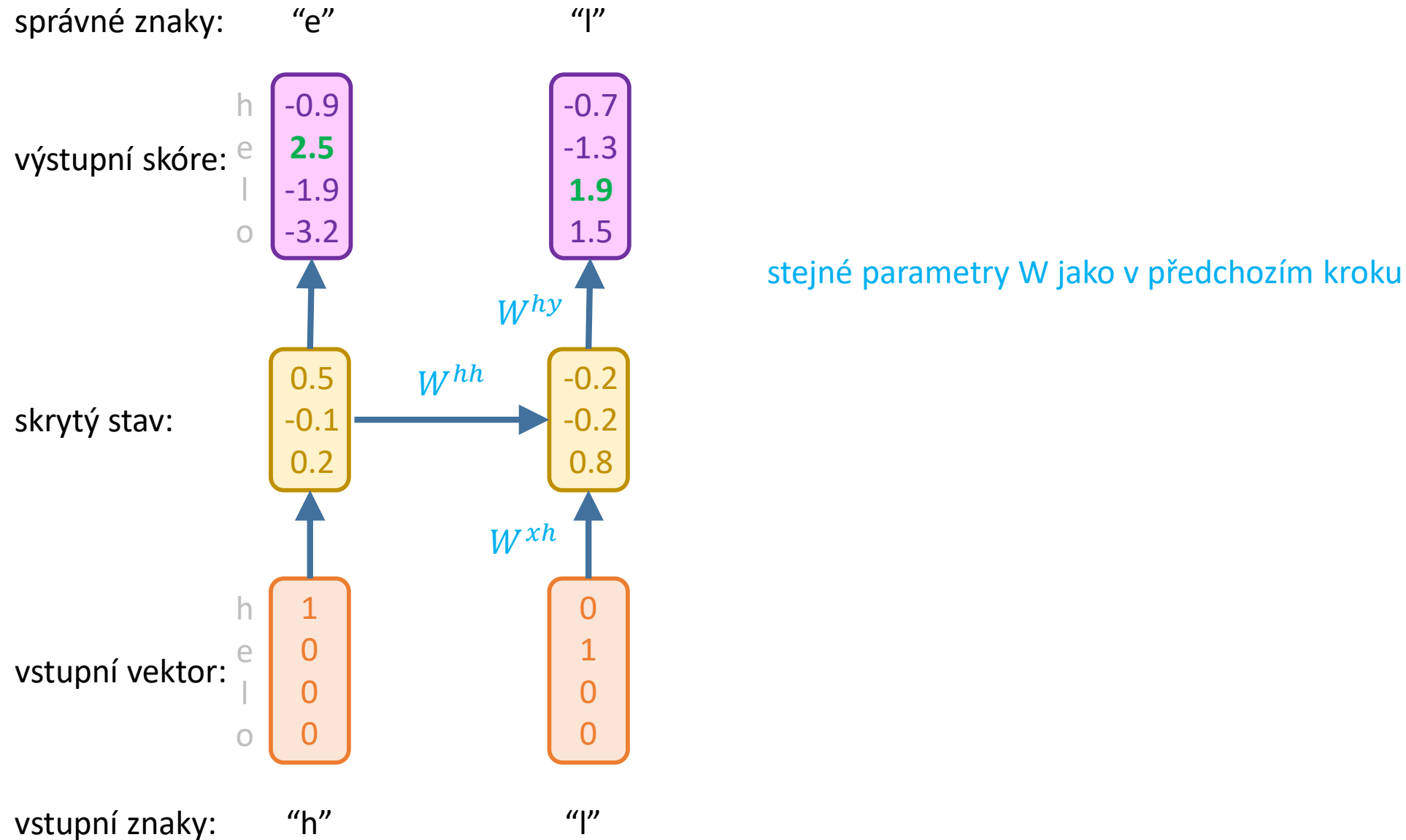
výstupem je lineární skóre pro každý znak ze slovníku

→ nejvyšší skóre pro "e" = po "h" je nejpravděpodobnější znak "e"

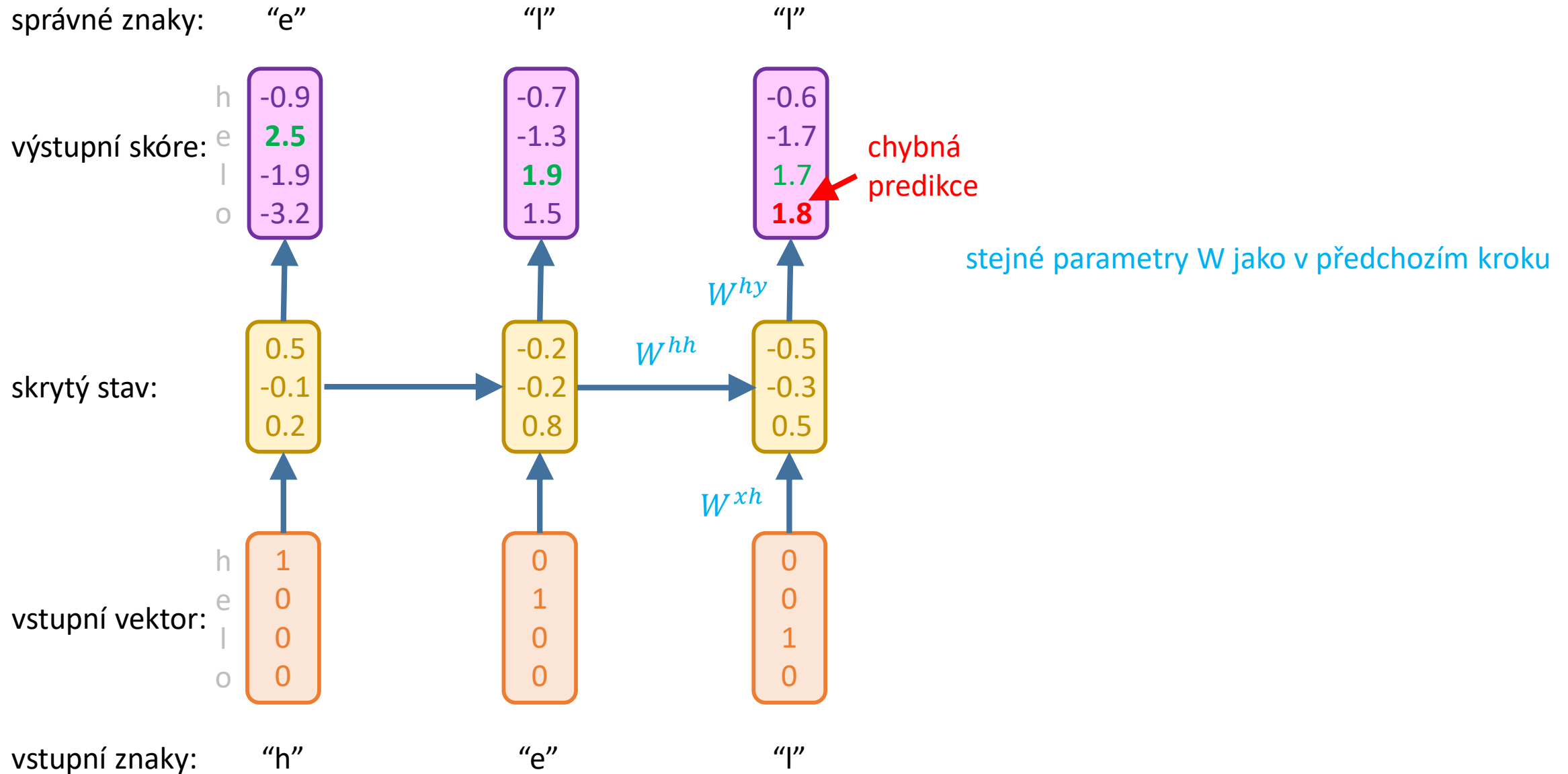
znaky modelovány jako vektory:
one-hot encoding

"slovník" je ["h", "e", "l", "o"]

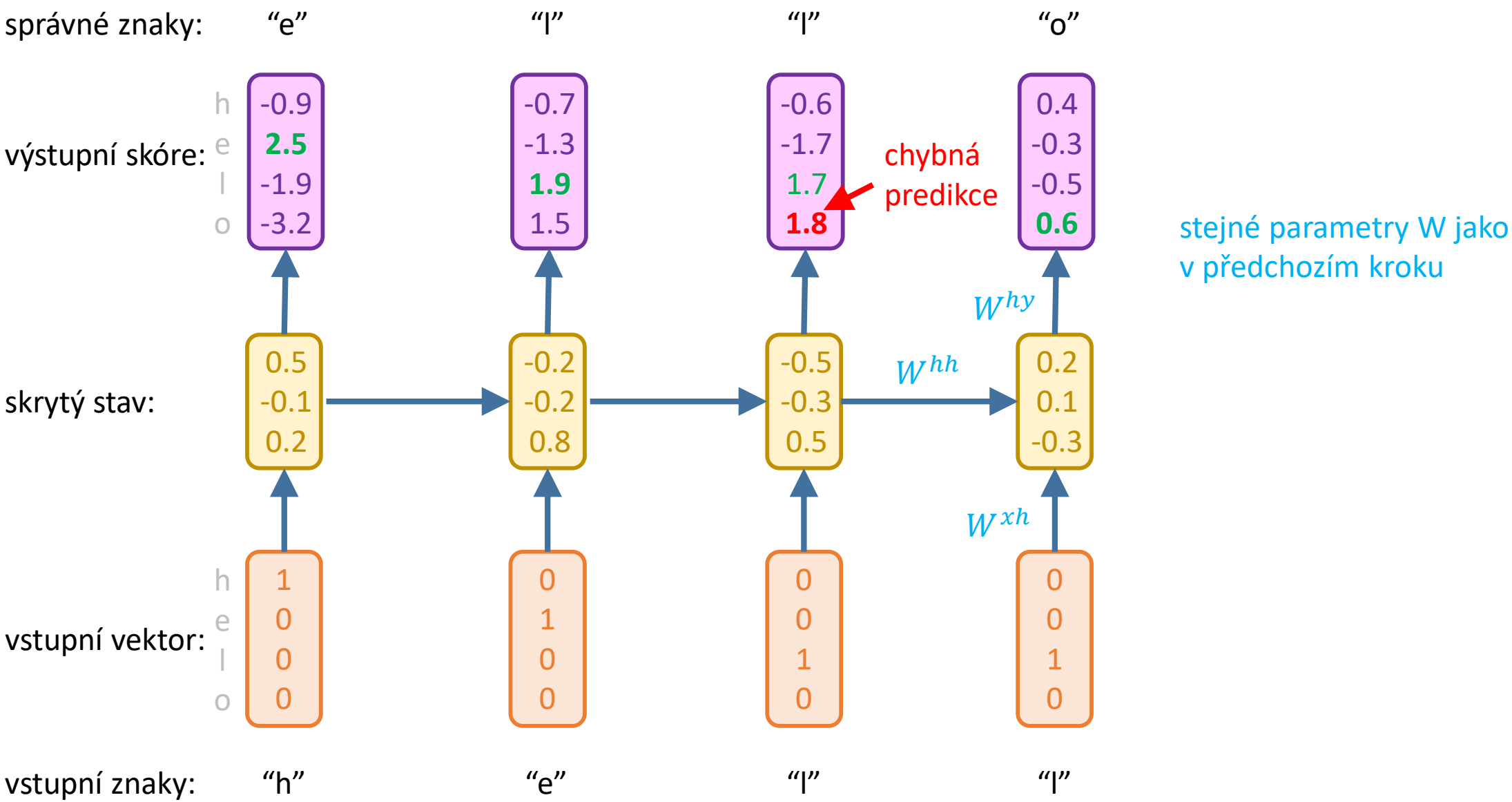
RNN jazykový model: slovo "hello"



RNN jazykový model: slovo "hello"



RNN jazykový model: slovo "hello"



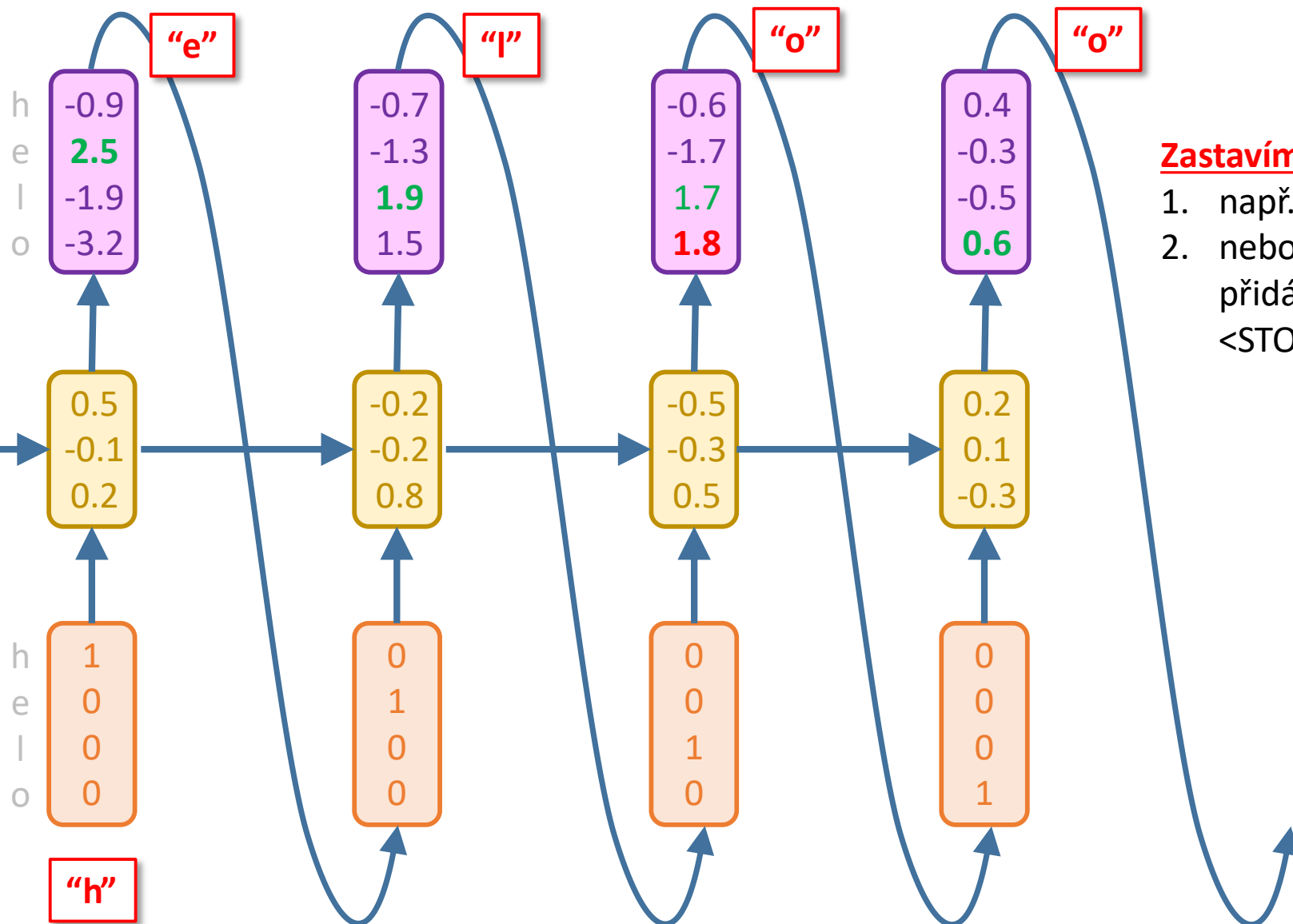
Generování textu pomocí RNN (vzorkování, sampling)

nemusíme vždy vybrat
nejpravděpodobnější
znak → náhodný výběr
dle predikovaných
pravděpodobností

skrytý stav obvykle
inicializujeme na nuly

apriorní rozložení

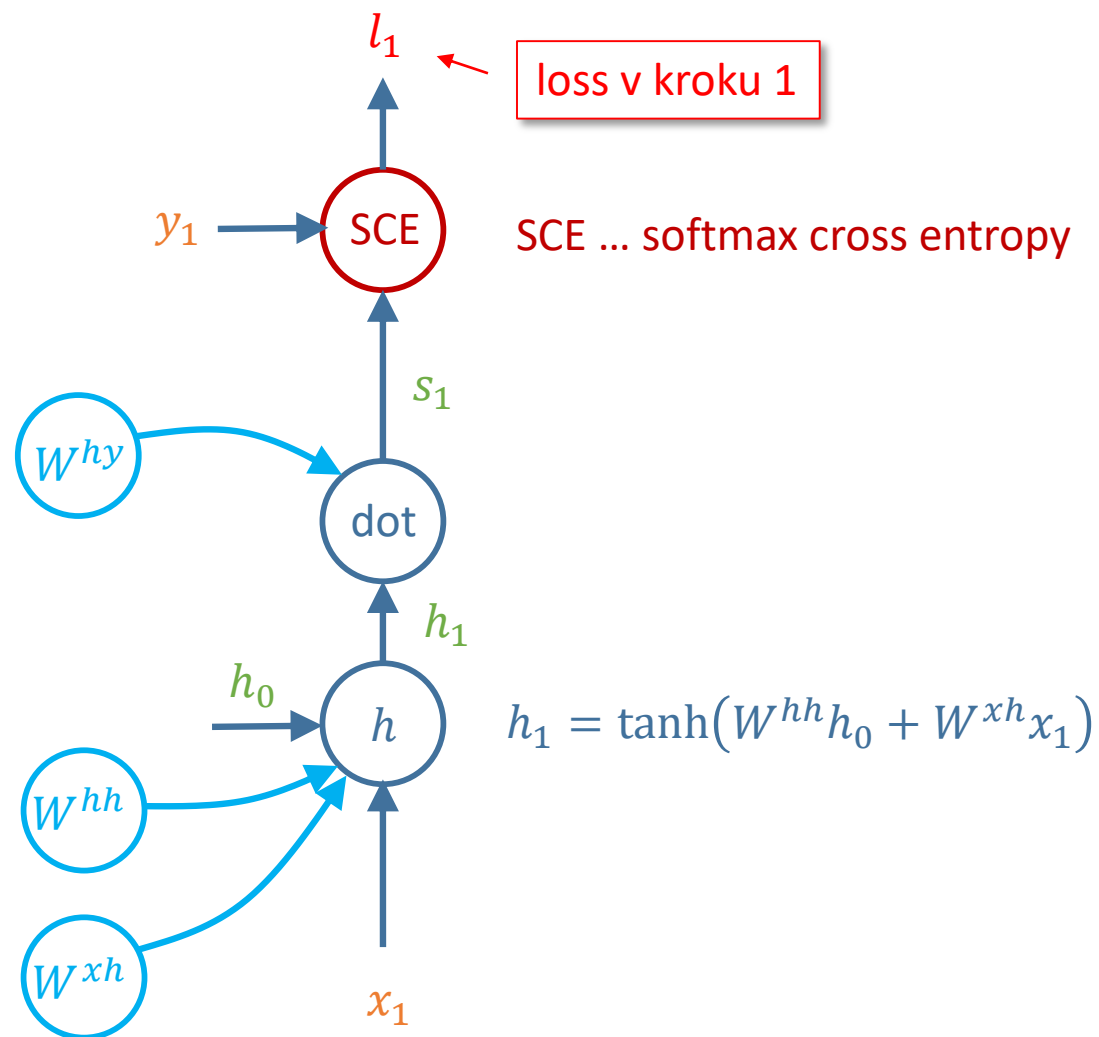
náhodný začátek:



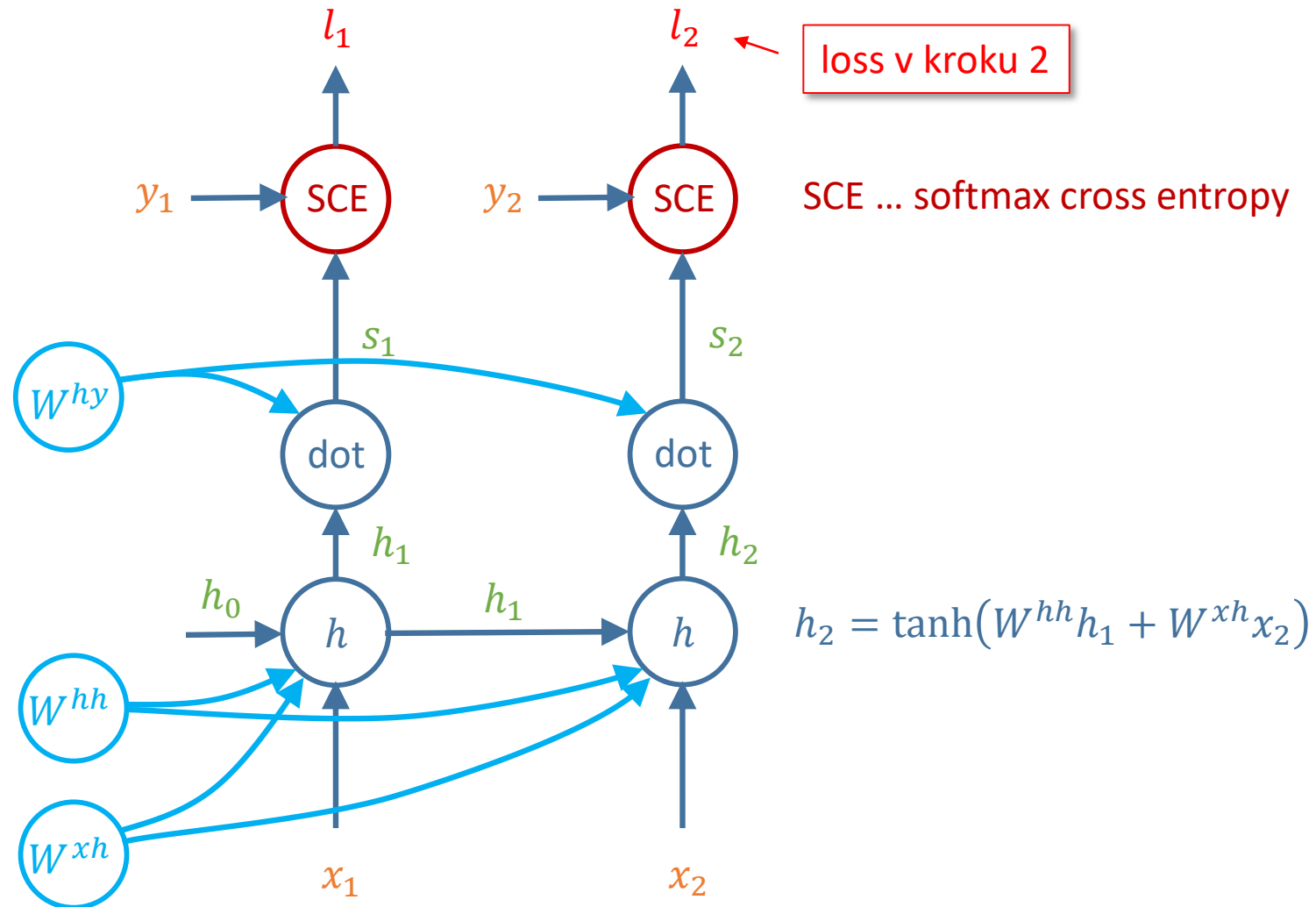
Zastavíme:

1. např. po 4 znacích
2. nebo do slovníku přidáme speciální <STOP> token

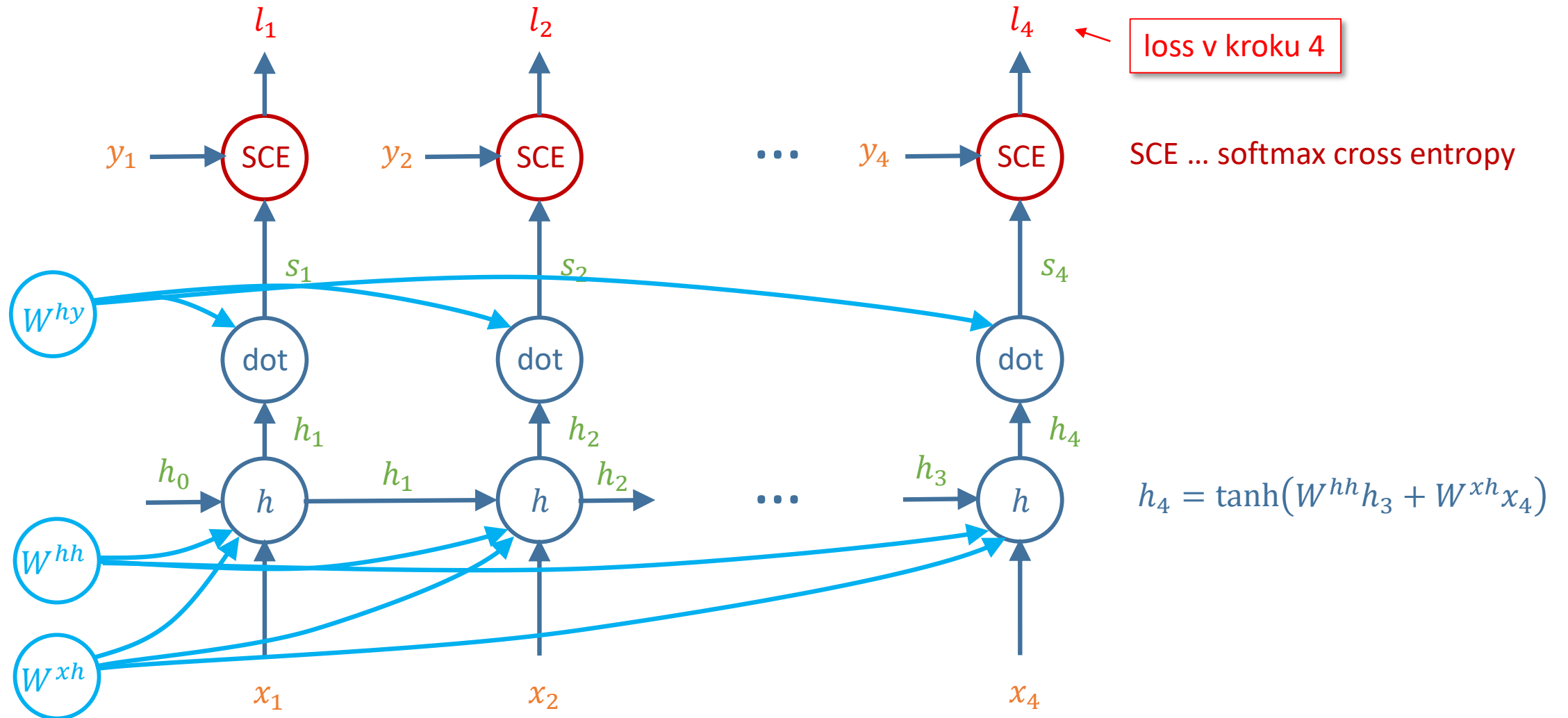
RNN jazykový model: výsledný výpočetní graf



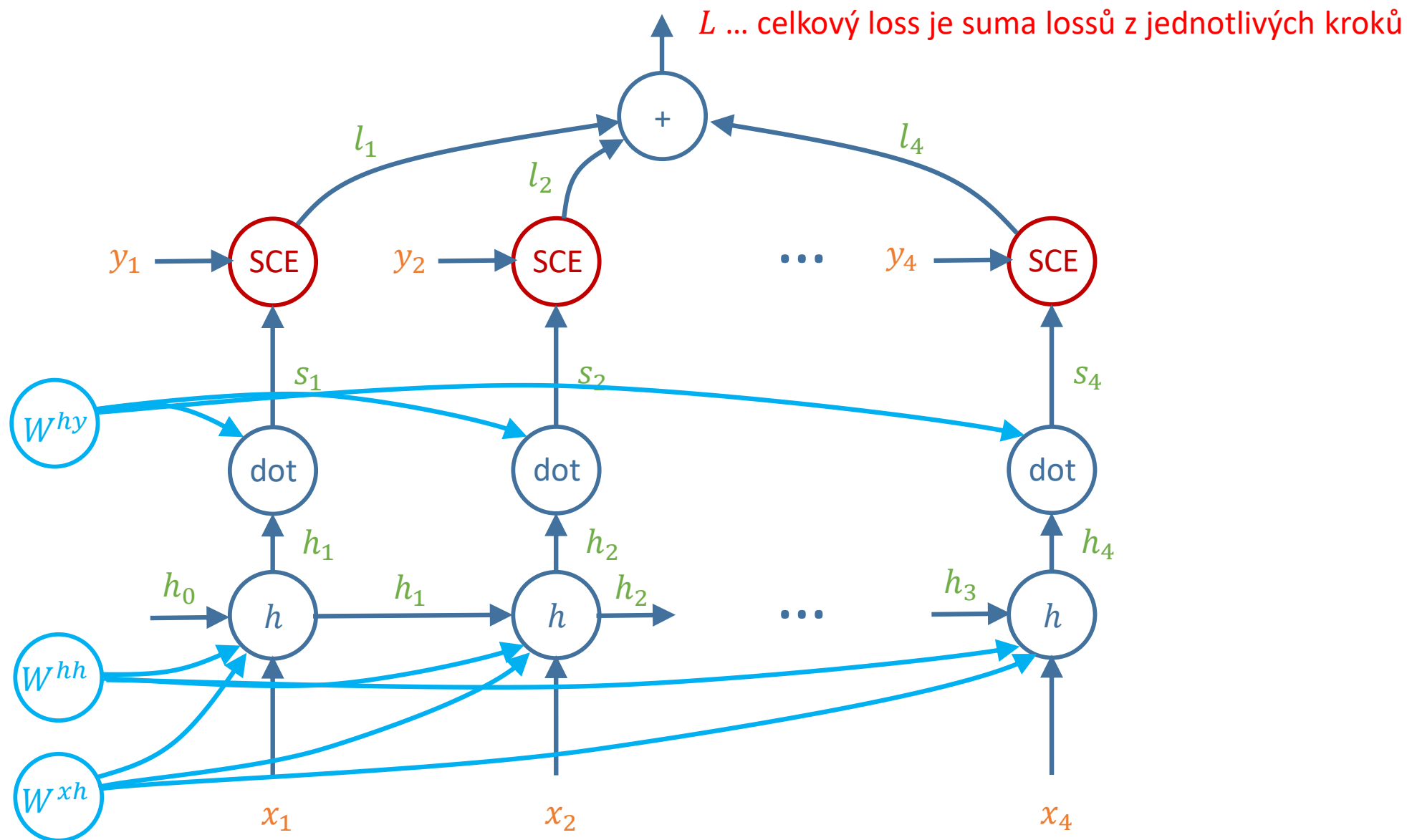
RNN jazykový model: výsledný výpočetní graf



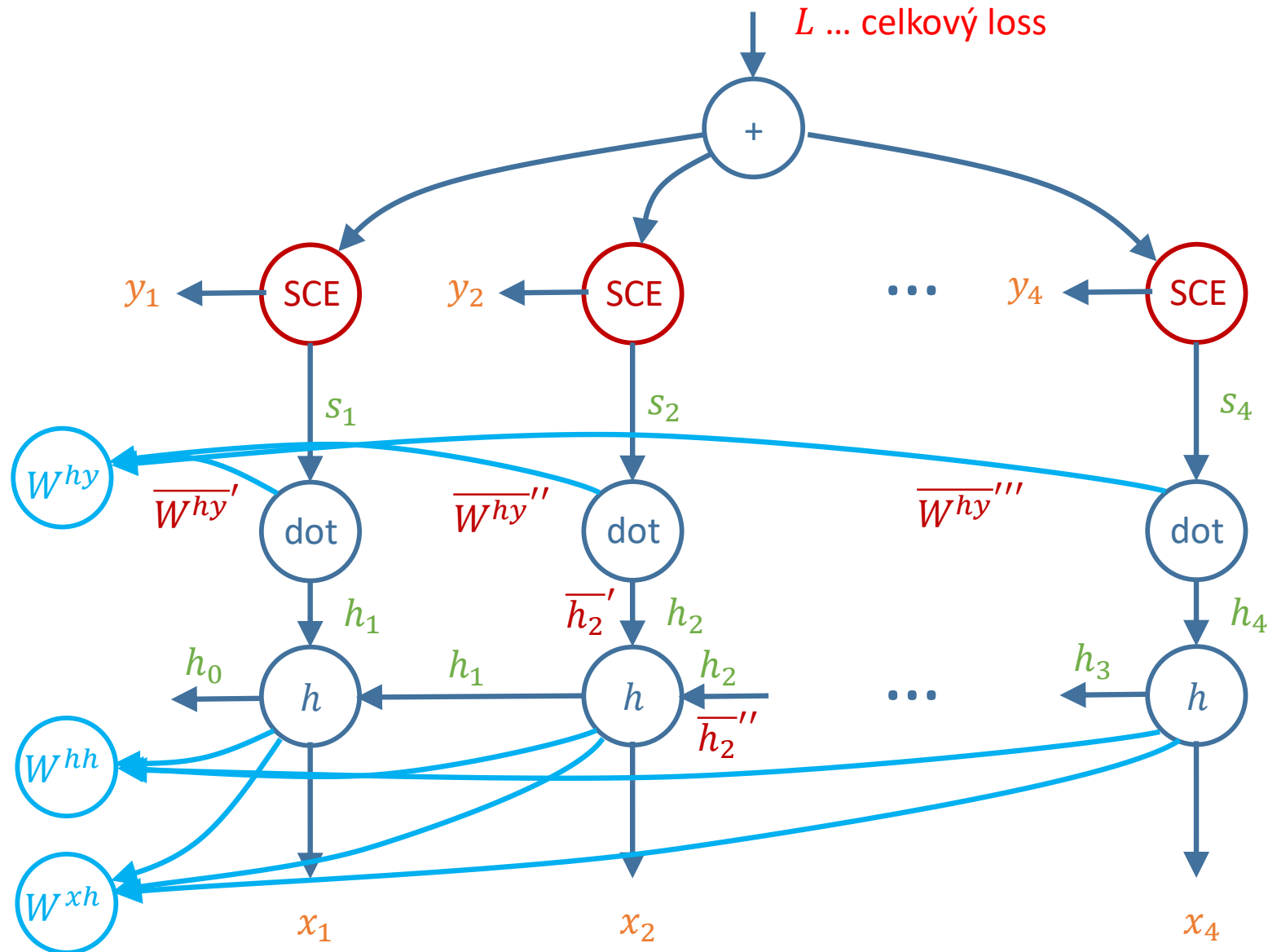
RNN jazykový model: výsledný výpočetní graf



RNN jazykový model: výsledný výpočetní graf



RNN jazykový model: zpětný průchod



parametry W jsou pro všechny kroky neměnné \rightarrow vícenásobné používání jednoho uzlu \rightarrow **celkový gradient se přes jednotlivé kroky sčítá**

např.:

$$\overline{W}^{hy} = \overline{W}^{hy'} + \overline{W}^{hy''} + \overline{W}^{hy'''}$$

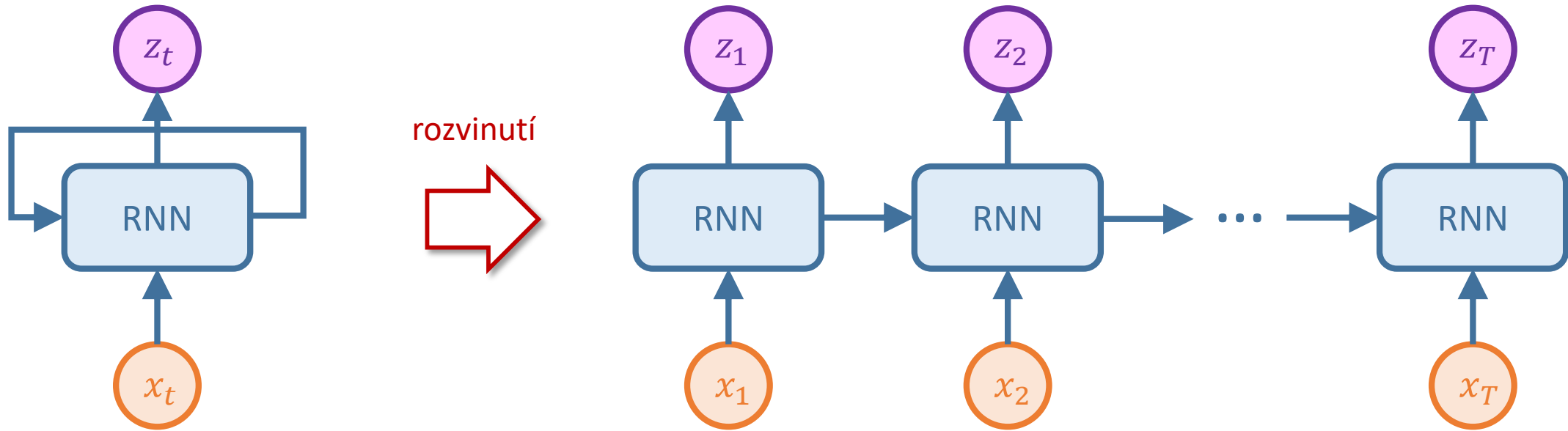
nebo:

$$\overline{h}_2 = \overline{h}_2' + \overline{h}_2''$$

Zpětná propagace v čase

Uvedený postup učení se v anglické literatuře označuje jako backpropagation through time (BPTT), tedy jako kdybychom rozvinuli rekurentní síť v čase

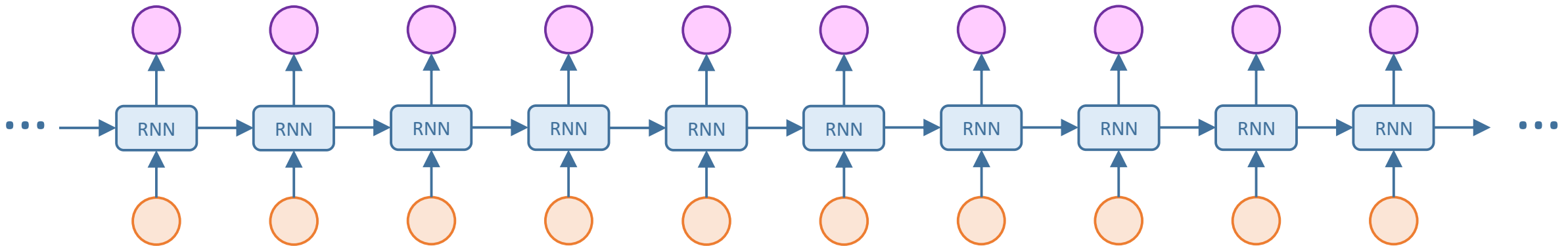
→ celá trénovací sekvence jako jeden velký výpočetní graf!



Dlouhé sekvence

velký výpočetní graf →

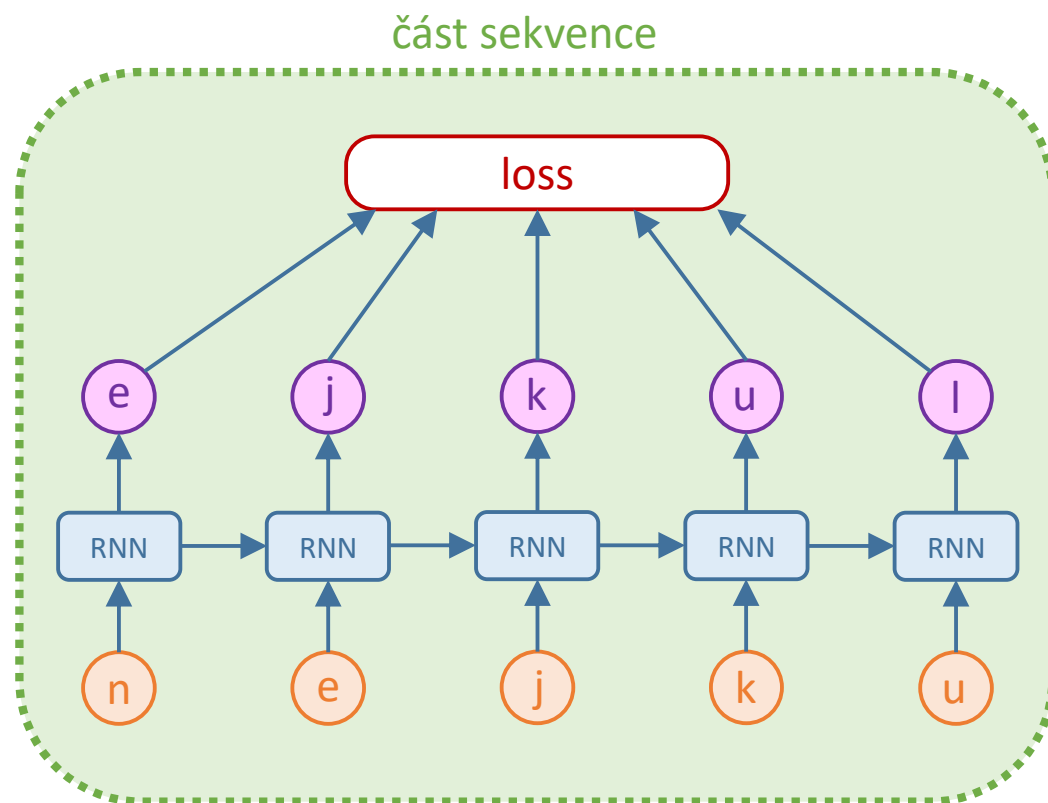
podobný problém jako u velmi hlubokých sítí: umírající gradient (vanishing gradient)



řešení:

1. lepší architektura (LSTM, více dále)
2. rozdělení sekvence na menší části → truncated backpropagation through time

Zkrácená zpětná propagace v čase

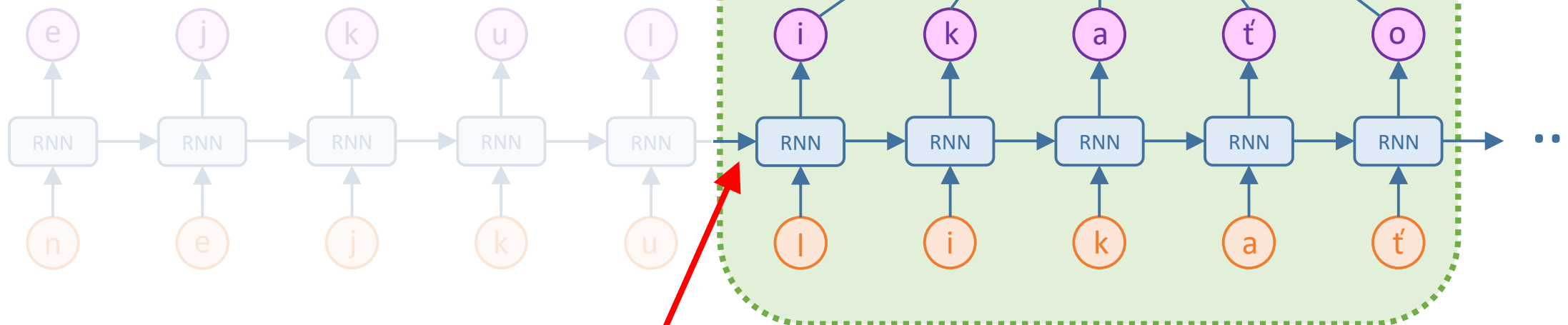


dopředný a zpětný průchod pouze na této části

poté update parametrů

Zkrácená zpětná propagace v čase

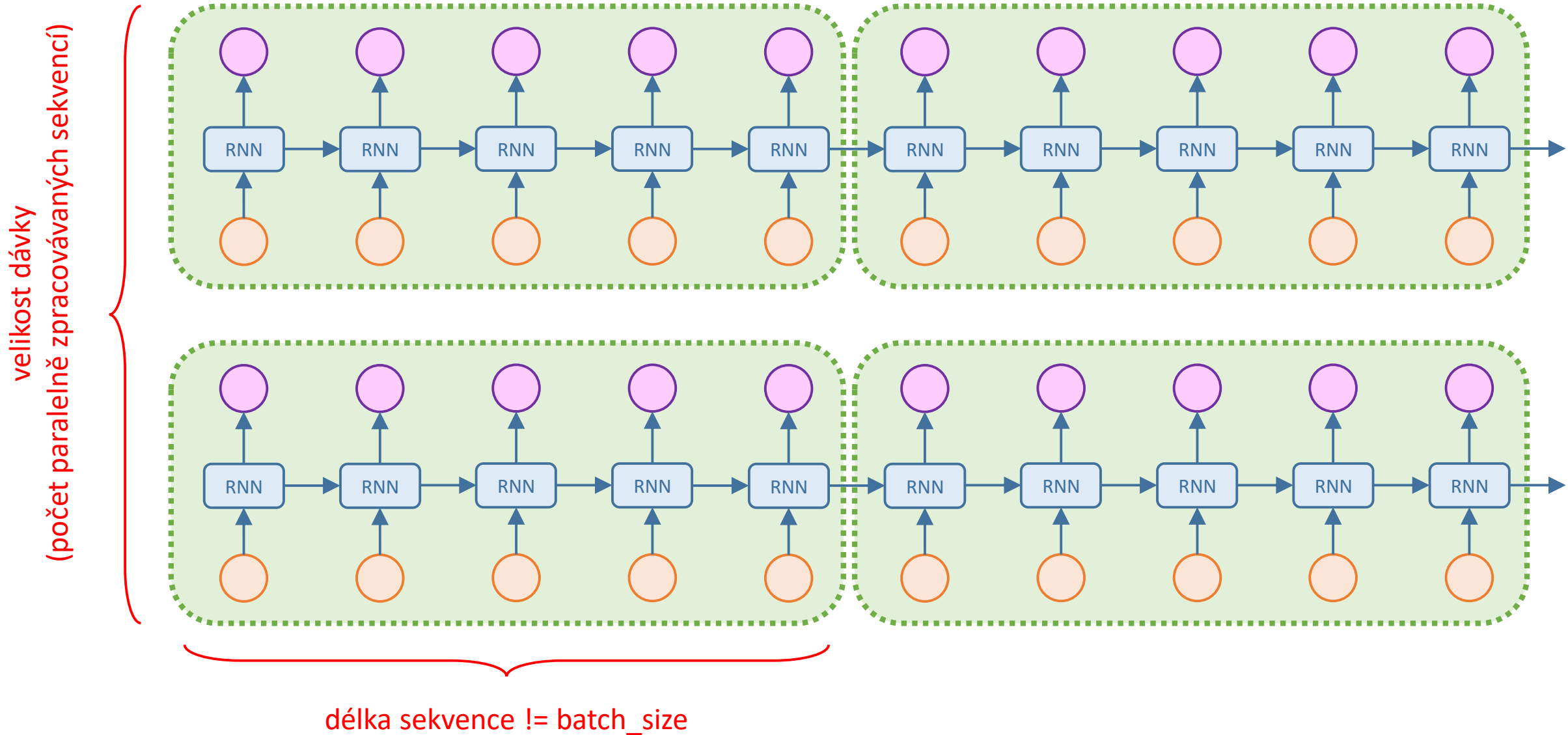
truncated



skrytý stav zůstává inicializovaný z konce minulé sekvence, zpětný tok gradientů zde však končí a do předchozího grafu neproniká

dopředný a zpětný průchod pouze na této části
poté update parametrů

Délka sekvence vs velikost dávky (batch size)



Min-char by Andrej Karpathy

```
27 def lossFun(inputs, targets, hprev):
28     """
29     inputs,targets are both list of integers.
30     hprev is Hx1 array of initial hidden state
31     returns the loss, gradients on model parameters, and last hidden state
32     """
33     xs, hs, ys, ps = {}, {}, {}, {}
34     hs[-1] = np.copy(hprev)
35     loss = 0
36     # forward pass
37     for t in xrange(len(inputs)):
38         xs[t] = np.zeros((vocab_size,1)) # encode in 1-of-k representation
39         xs[t][inputs[t]] = 1
40         hs[t] = np.tanh(np.dot(Wxh, xs[t]) + np.dot(Whh, hs[t-1]) + bh) # hidden state
41         ys[t] = np.dot(Why, hs[t]) + by # unnormalized log probabilities for next chars
42         ps[t] = np.exp(ys[t]) / np.sum(np.exp(ys[t])) # probabilities for next chars
43         loss += -np.log(ps[t][targets[t],0]) # softmax (cross-entropy loss)
```

dostupné na: <https://gist.github.com/karpathy/d4dee566867f8291f086>

Min-char by Andrej Karpathy

```
27  def lossFun(inputs, targets, hprev):  
  
    ...  
  
44  # backward pass: compute gradients going backwards  
45  dWxh, dWhh, dWhy = np.zeros_like(Wxh), np.zeros_like(Whh), np.zeros_like(Why)  
46  dbh, dby = np.zeros_like(bh), np.zeros_like(by)  
47  dhnext = np.zeros_like(hs[0])  
48  for t in reversed(xrange(len(inputs))):  
49      dy = np.copy(ps[t])  
50      dy[targets[t]] -= 1 # backprop into y. see http://cs231n.github.io/neural-networks-case-study/#grad if confused here  
51      dWhy += np.dot(dy, hs[t].T)  
52      dby += dy  
53      dh = np.dot(Why.T, dy) + dhnext # backprop into h  
54      dhraw = (1 - hs[t] * hs[t]) * dh # backprop through tanh nonlinearity  
55      dbh += dhraw  
56      dWxh += np.dot(dhraw, xs[t].T)  
57      dWhh += np.dot(dhraw, hs[t-1].T)  
58      dhnext = np.dot(Whh.T, dhraw)
```

dostupné na: <https://gist.github.com/karpathy/d4dee566867f8291f086>

Min-char by Andrej Karpathy

```
63  def sample(h, seed_ix, n):
64      """
65      sample a sequence of integers from the model
66      h is memory state, seed_ix is seed letter for first time step
67      """
68      x = np.zeros((vocab_size, 1))
69      x[seed_ix] = 1
70      ixes = []
71      for t in xrange(n):
72          h = np.tanh(np.dot(Wxh, x) + np.dot(Whh, h) + bh)
73          y = np.dot(Why, h) + by
74          p = np.exp(y) / np.sum(np.exp(y))
75          ix = np.random.choice(range(vocab_size), p=p.ravel())
76          x = np.zeros((vocab_size, 1))
77          x[ix] = 1
78          ixes.append(ix)
79      return ixes
```

dostupné na: <https://gist.github.com/karpathy/d4dee566867f8291f086>

Min-char by Andrej Karpathy

```
7  # data I/O
8  data = open('input.txt', 'r').read() # should be simple plain text file
9  chars = list(set(data))
10 data_size, vocab_size = len(data), len(chars)
11 print 'data has %d characters, %d unique.' % (data_size, vocab_size)
12 char_to_ix = { ch:i for i,ch in enumerate(chars) }
13 ix_to_char = { i:ch for i,ch in enumerate(chars) }
14
15 # hyperparameters
16 hidden_size = 100 # size of hidden layer of neurons
17 seq_length = 25 # number of steps to unroll the RNN for
18 learning_rate = 1e-1
19
20 # model parameters
21 Wxh = np.random.randn(hidden_size, vocab_size)*0.01 # input to hidden
22 Whh = np.random.randn(hidden_size, hidden_size)*0.01 # hidden to hidden
23 Why = np.random.randn(vocab_size, hidden_size)*0.01 # hidden to output
24 bh = np.zeros((hidden_size, 1)) # hidden bias
25 by = np.zeros((vocab_size, 1)) # output bias
```

dostupné na: <https://gist.github.com/karpathy/d4dee566867f8291f086>

Min-char by Andrej Karpathy

```

99     # forward seq_length characters through the net and fetch gradient
100    loss, dWxh, dWhh, dWhy, dbh, dby, hprev = lossFun(inputs, targets, hprev)
101    smooth_loss = smooth_loss * 0.999 + loss * 0.001
102    if n % 100 == 0: print 'iter %d, loss: %f' % (n, smooth_loss) # print progress
103
104    # perform parameter update with Adagrad
105    for param, dparam, mem in zip([Wxh, Whh, Why, bh, by],
106                                  [dWxh, dWhh, dWhy, dbh, dby],
107                                  [mWxh, mWhh, mWhy, mbh, mby]):
108        mem += dparam * dparam
109        param += -learning_rate * dparam / np.sqrt(mem + 1e-8) # adagrad update

```

dostupné na: <https://gist.github.com/karpathy/d4dee566867f8291f086>

char-rnn by Andrej Karpathy

PANDARUS:

Alas, I think he shall be come approached and the day
When little strain would be attain'd into being never fed,
And who is but a chain and subjects of his death,
I should not sleep.

Second Senator:

They are away this miseries, produced upon my soul,
Breaking and strongly should be buried, when I perish
The earth and thoughts of many states.

DUKE VINCENTIO:

Well, your wit is in the care of side and that.

Second Lord:

They would be ruled after this chamber, and
my fair nudes begun out of the fact, to be conveyed,
Whose noble souls I'll have the heart of the wars.

Clown:

Come, sir, I will make did behold your worship.

VIOLA:

I'll drink it.

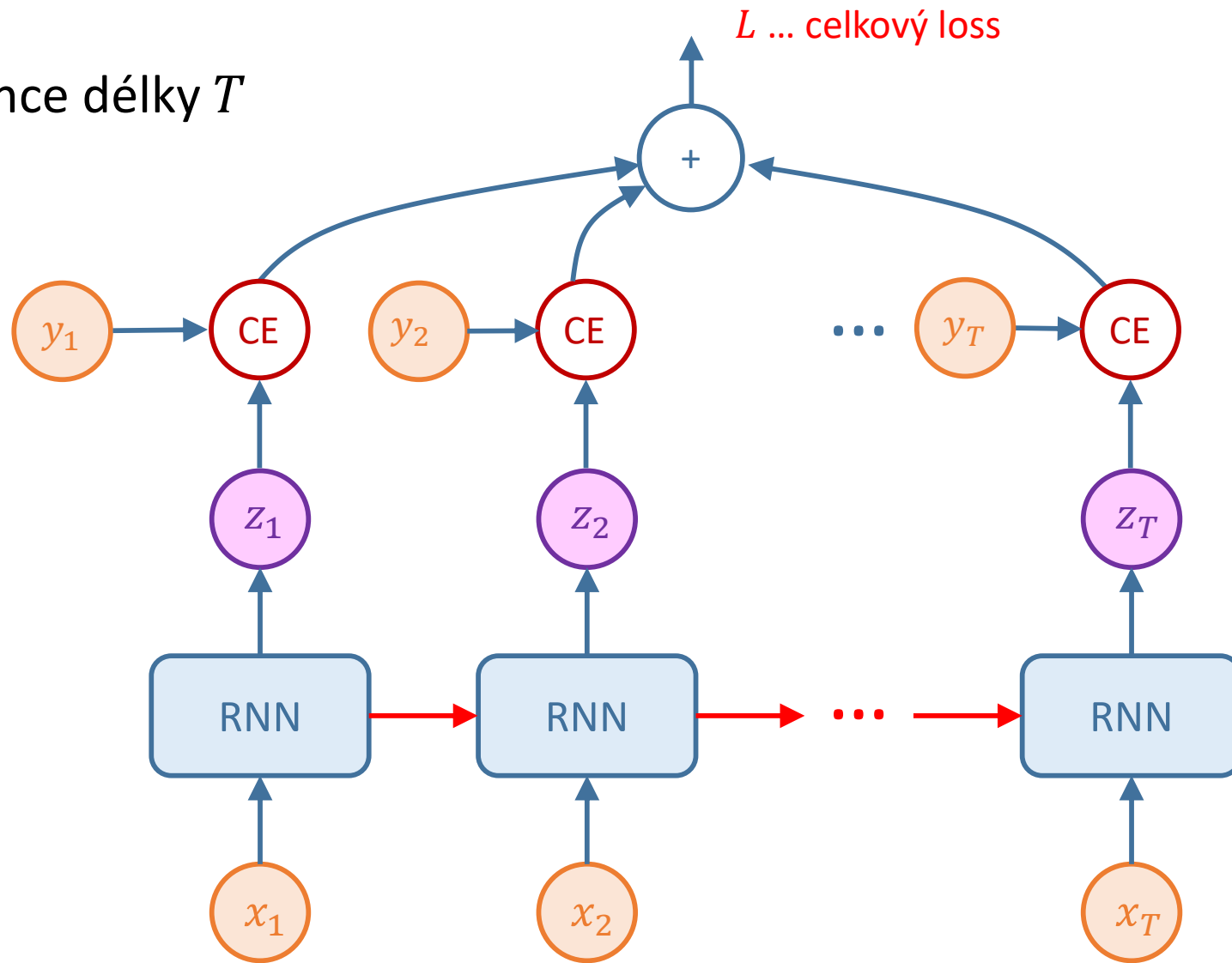
char-rnn by Andrej Karpathy

```
/*
 * Increment the size file of the new incorrect UI_FILTER group information
 * of the size generatively.
 */
static int indicate_policy(void)
{
    int error;
    if (fd == MARN_EPT) {
        /*
         * The kernel blank will coeld it to userspace.
         */
        if (ss->segment < mem_total)
            unblock_graph_and_set_blocked();
        else
            ret = 1;
        goto bail;
    }
    segaddr = in_SB(in.addr);
    selector = seg / 16;
    setup_works = true;
    for (i = 0; i < blocks; i++) {
        seq = buf[i++];
        bpf = bd->bd.next + i * search;
        if (fd) {
            current = blocked;
        }
    }
    rw->name = "Getjbbregs";
    bprm_self_clearl(&iv->version);
    regs->new = blocks[(BPF_STATS << info->historidac)] | PFMR_CLOBATHINC_SECONDS << 12;
    return segtable;
}
```

zdroj: <http://karpathy.github.io/2015/05/21/rnn-effectiveness/>

Jedna sekvence rekurentní sítě

sekvence délky T

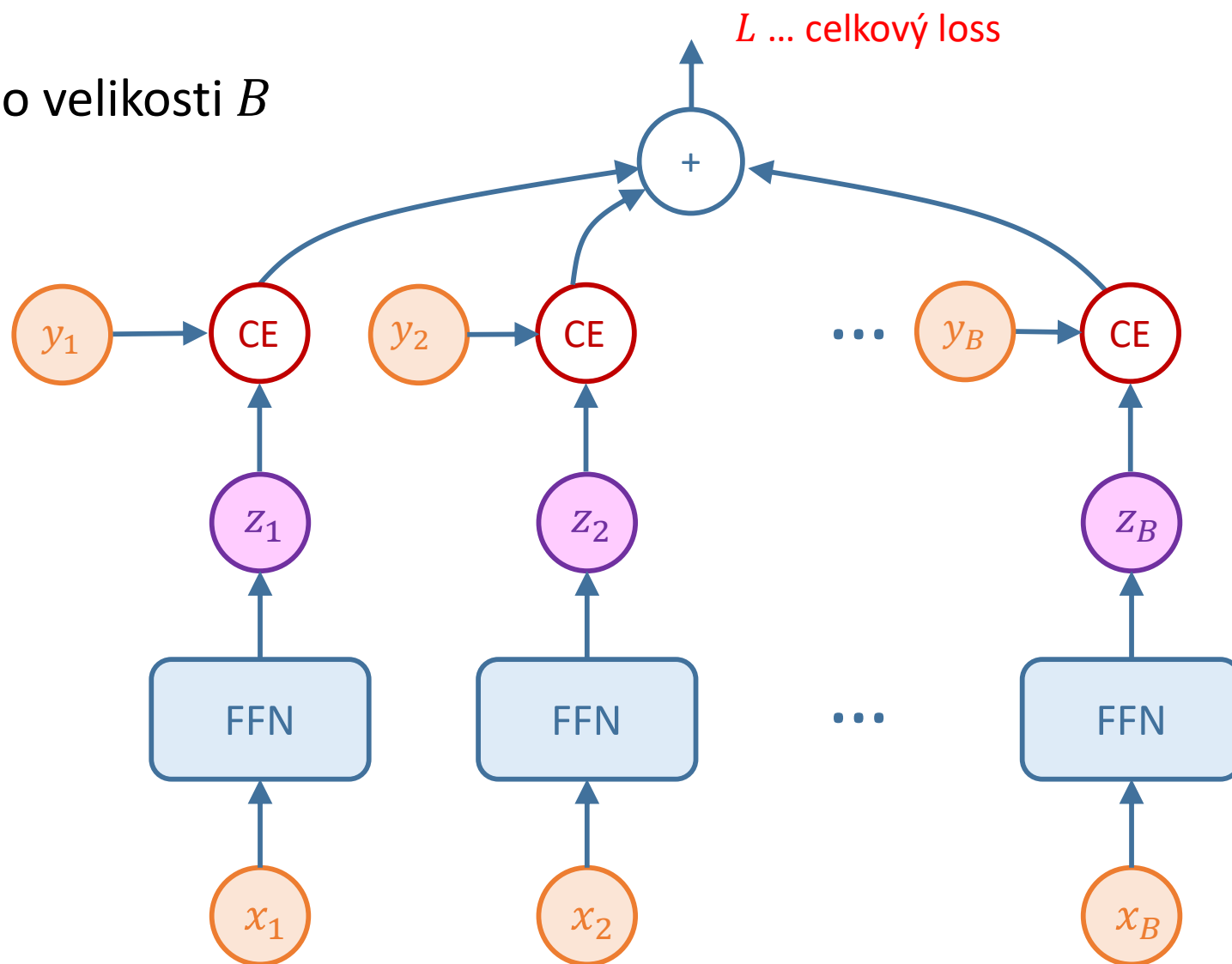


závislé průchody

jedna sekvence = jeden graf

Jedna dávka (batch) klasické dopředné sítě

batch o velikosti B



průchody
vzájemně nezávislé!

jeden batch = jeden graf

trénování rekurentních sítí
je tedy z pohledu výpočetního
grafu velmi podobné
dopředným sítím → jediným
rozdílem je, že **u RNN** jsou
jednotlivé průchody **vzájemně**
závislé, což ovlivňuje zpětný
průchod gradientů

Dlouhodobé závislosti

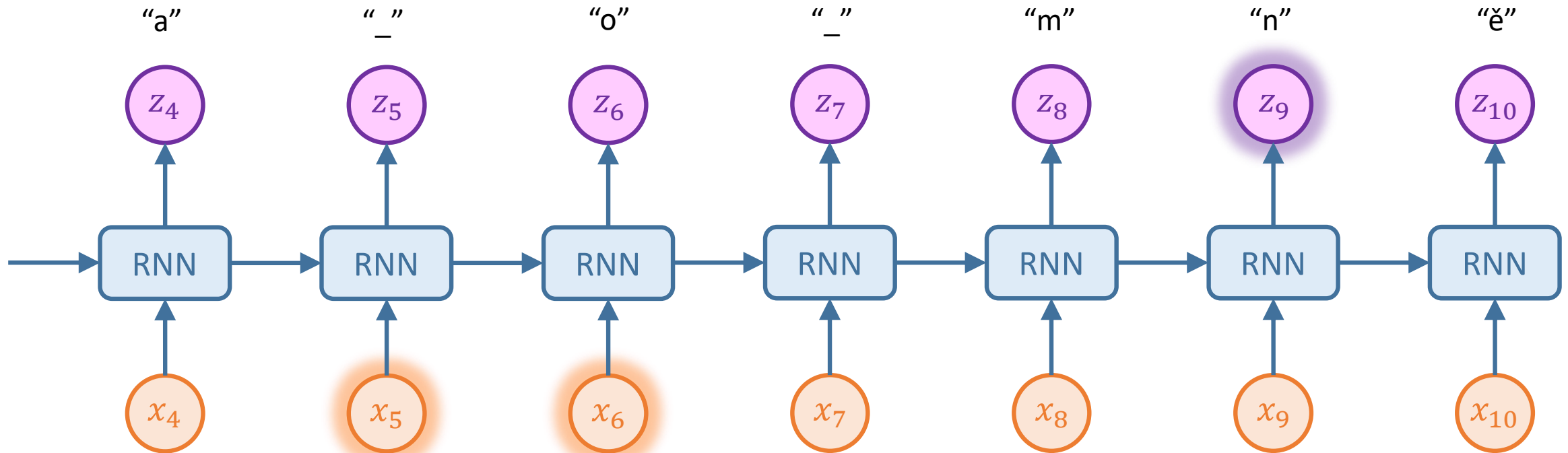
závisí na předložce!

kniha o mně

vs

kniha pro mě

zda-li skóre pro 9. vzorek má být nejvyšší pro m "n" nebo "ě" závisí na vzorcích č. 5 a č. 6



Dlouhodobé závislosti

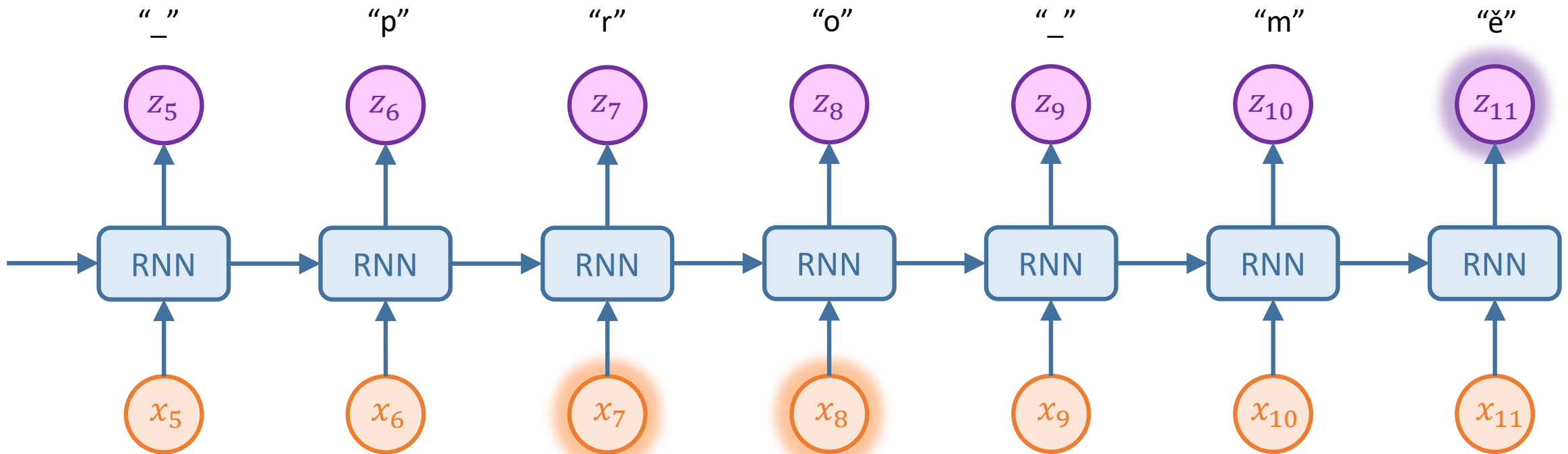
závisí na předložce!

kniha o mně

vs

kniha pro mě

zda-li skóre pro 11. vzorek má být nejvyšší pro m "n" nebo "ě" závisí na vzorcích č. 7 a č. 8



LSTM: Long Short-Term Memory

- Základní RNN trpí problémy s tokem gradientů
 - buď se vlivem mizejících gradientů neučí delší závislosti
 - nebo naopak rekurencí gradienty tzv. explodují, viz např. min-char

```
59     for dparam in [dWxh, dWhh, dWhy, dbh, dby]:  
60         np.clip(dparam, -5, 5, out=dparam) # clip to mitigate exploding gradients
```

- Jedním z řešení je použít lepší architekturu

pomocná "hradla" $\left\{ \begin{array}{l} i = \sigma(W^{xi}x_t + W^{hi}h_{t-1}) \\ f = \sigma(W^{xf}x_t + W^{hf}h_{t-1}) \\ o = \sigma(W^{xo}x_t + W^{ho}h_{t-1}) \\ g = \tanh(W^{xg}x_t + W^{hg}h_{t-1}) \end{array} \right.$

$$\begin{array}{l} c_t = f \circ c_{t-1} + g \circ i \\ h_t = \tanh(c_t) \circ o \end{array}$$

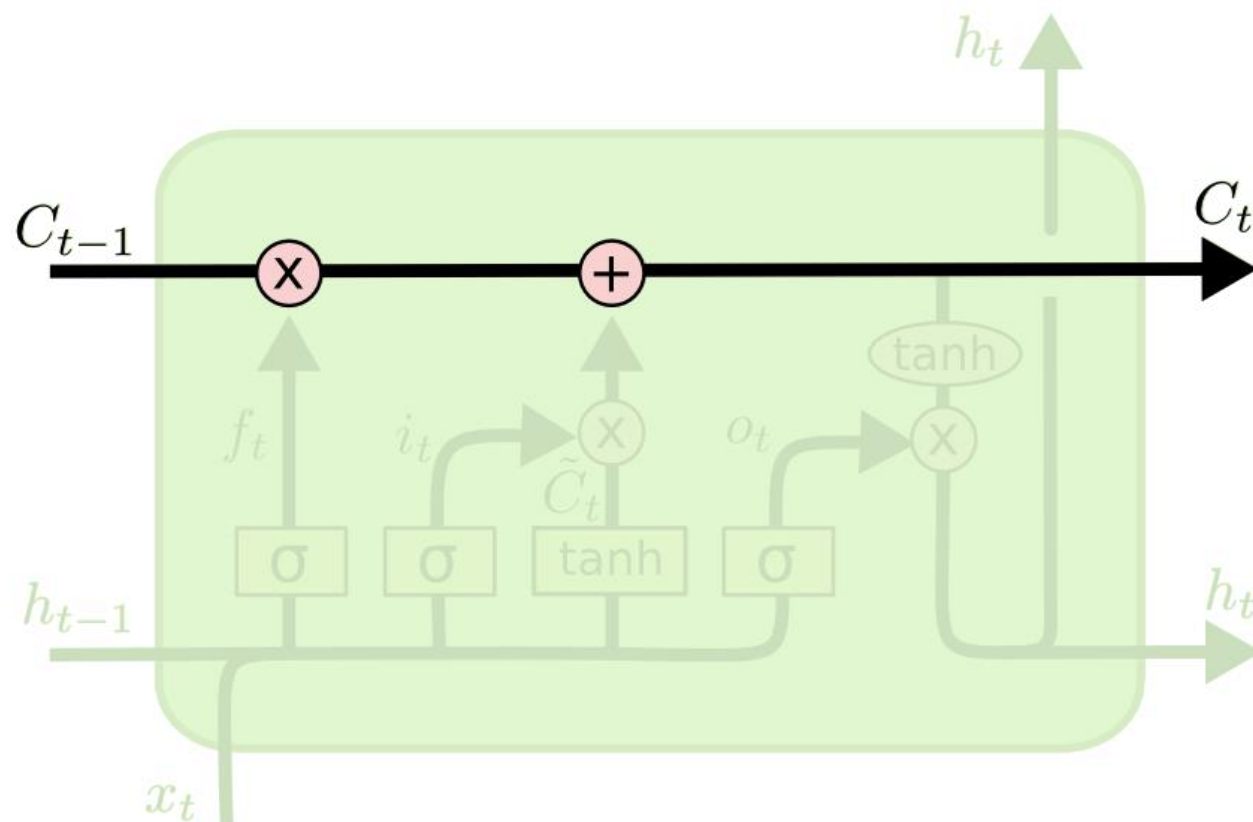
◦ ... prvkové násobení

dvě stavové proměnné!

c_t ... cell state
 h_t ... hidden state

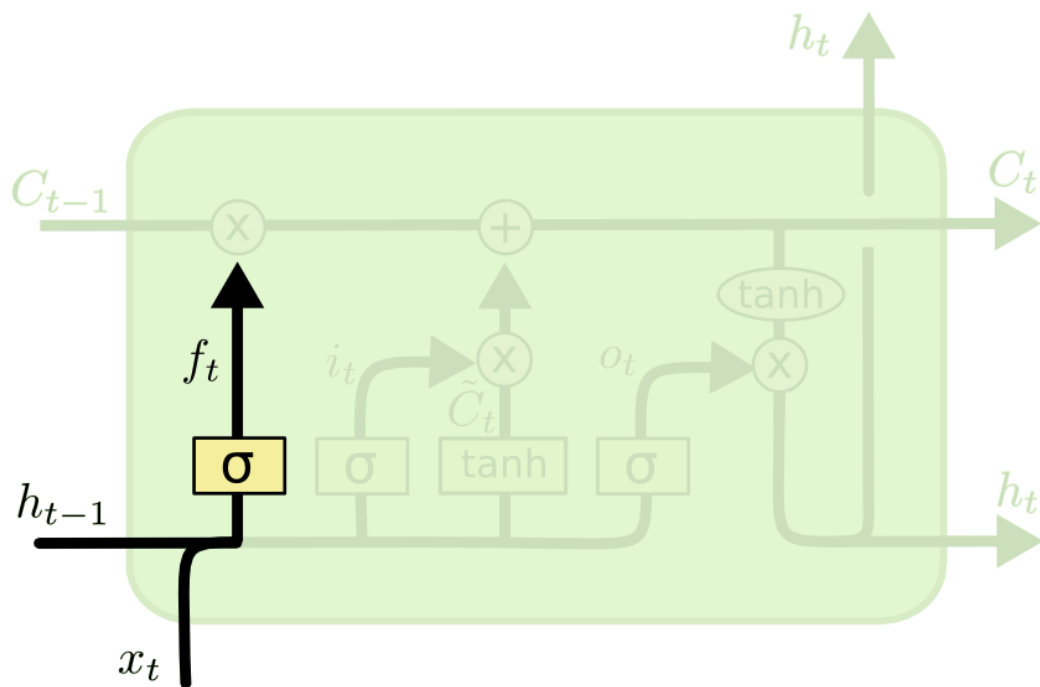
LSTM: cell stav

podobné jako reziduální spoje (identity mapping spoj)



obrázek: <http://colah.github.io/posts/2015-08-Understanding-LSTMs/>

LSTM: forget gate

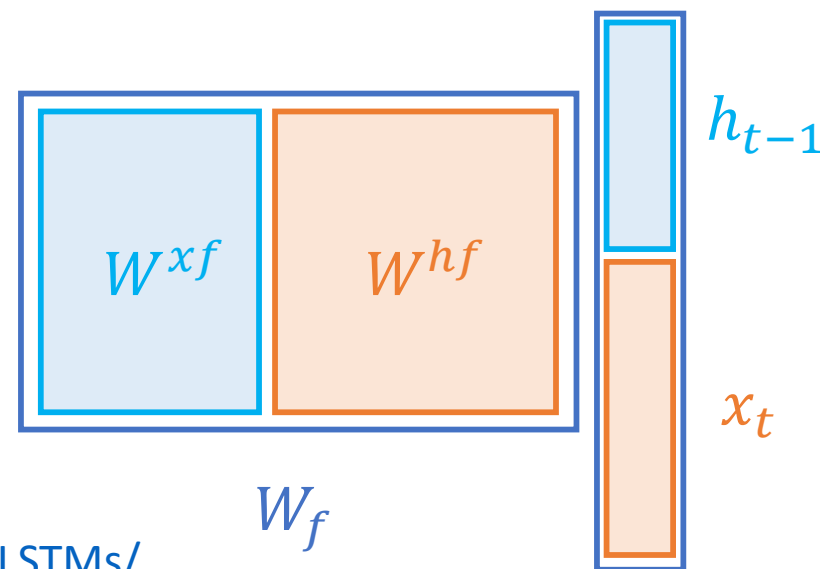


upravuje, co se zapomene, reguluje c

ekvivalentní zápis

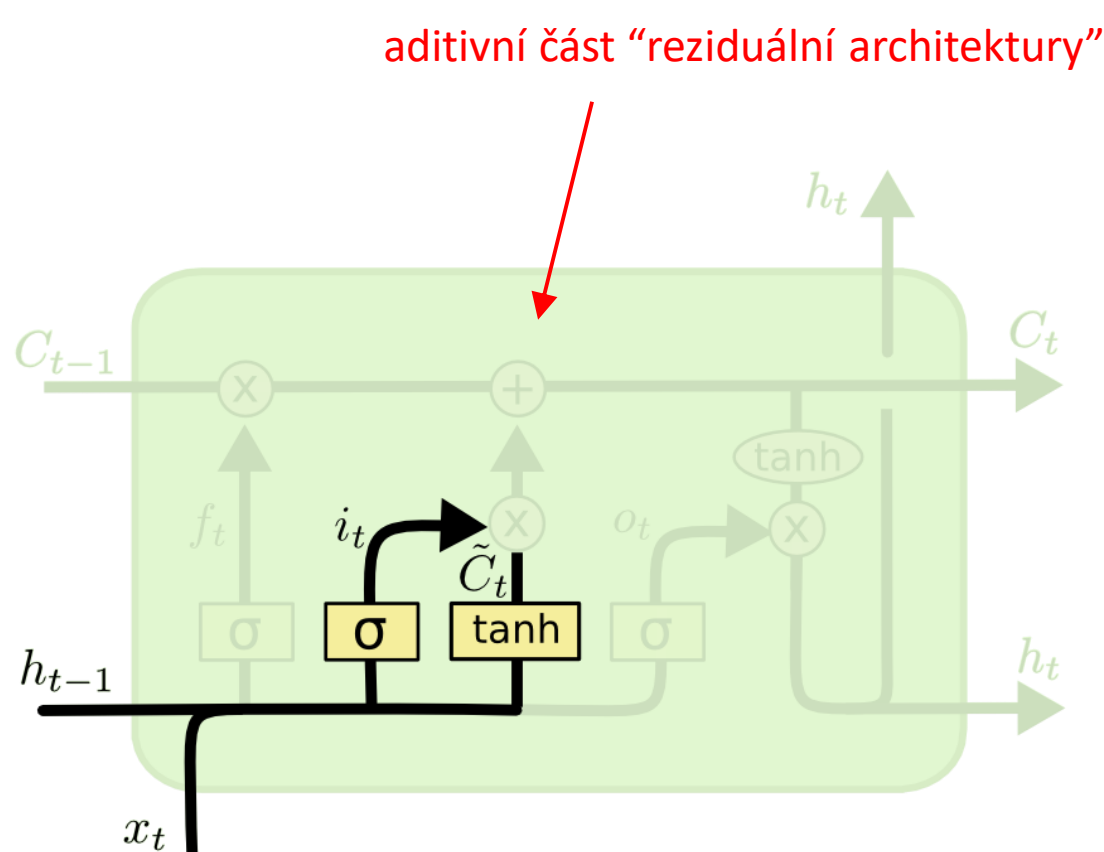
$$W^{xf} x_t + W^{hf} h_{t-1}$$

$$f_t = \sigma (W_f \cdot [h_{t-1}, x_t] + b_f)$$



obrázek: <http://colah.github.io/posts/2015-08-Understanding-LSTMs/>

LSTM: input gate a update cell stavu



$$c_t = f \circ c_{t-1} + \underbrace{g \circ i}_{\tilde{c}_t}$$

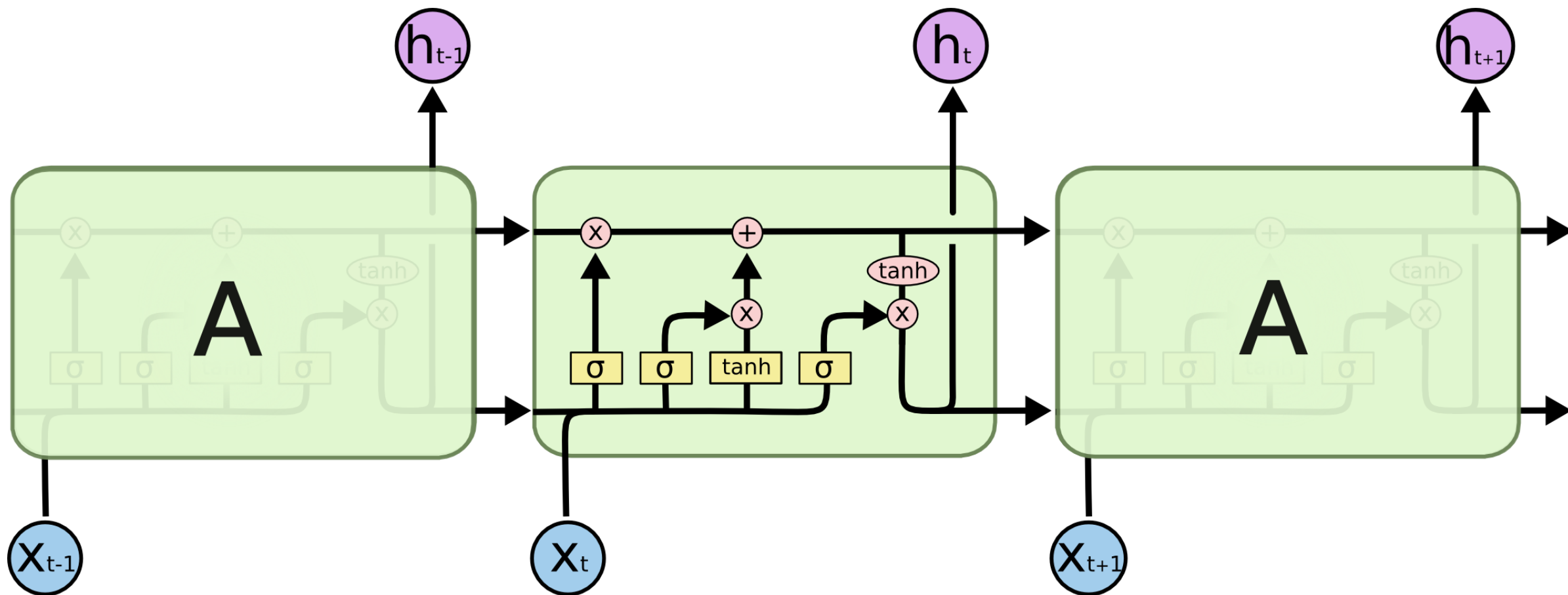
$$i_t = \sigma(W_i \cdot [h_{t-1}, x_t] + b_i)$$

$$\tilde{C}_t = \tanh(W_C \cdot [h_{t-1}, x_t] + b_C)$$

rozhoduje, co se zapíše nového do cell stavu

obrázek: <http://colah.github.io/posts/2015-08-Understanding-LSTMs/>

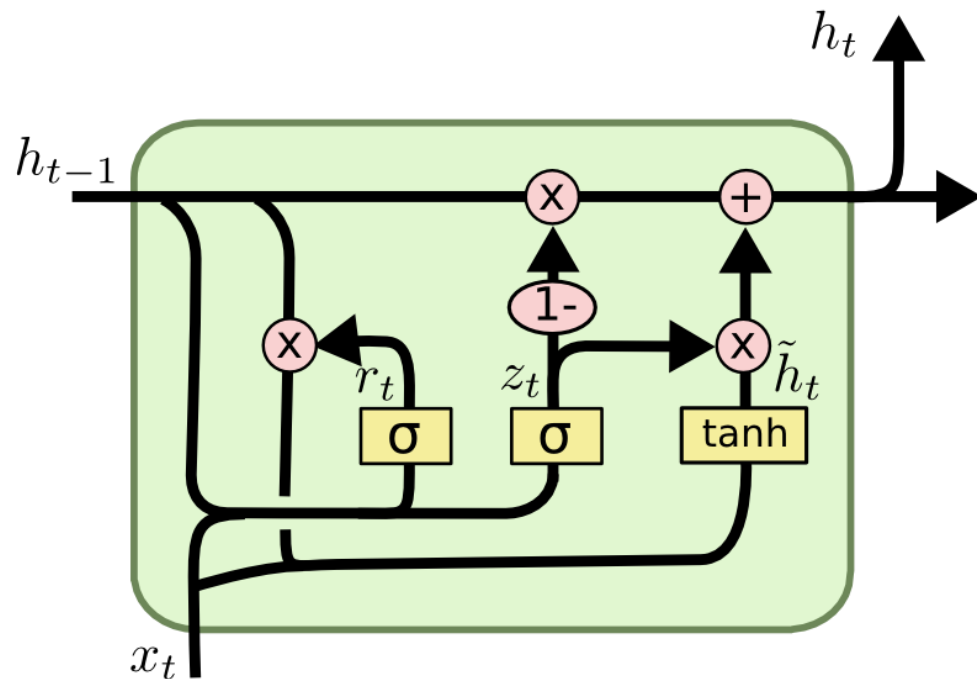
LSTM rozvinutá v čase



obrázek: <http://colah.github.io/posts/2015-08-Understanding-LSTMs/>

Gated Recurrent Unit (GRU)

variance na LSTM → zjednodušení



$$z_t = \sigma (W_z \cdot [h_{t-1}, x_t])$$

$$r_t = \sigma (W_r \cdot [h_{t-1}, x_t])$$

$$\tilde{h}_t = \tanh (W \cdot [r_t * h_{t-1}, x_t])$$

$$h_t = (1 - z_t) * h_{t-1} + z_t * \tilde{h}_t$$

v praxi funguje velmi podobně jako LSTM (ne vždy), ovšem rychlejší a efektivnější

obrázek: <http://colah.github.io/posts/2015-08-Understanding-LSTMs/>

Shrnutí

- Rekurentní sítě vhodné pro sekvenční data
- Trénování probíhá rozvinutím výpočetního grafu → zpětná propagace v čase
- Vstupní sekvence se dělí na kratší kousky → jeden kousek = jeden update parametrů
- Batch size odpovídá počtu paralelně zpracovávaných kousků
- Základní RNN se pro problémy s tokem gradientů téměř nepoužívá
- Zdaleka nejrozšířenější LSTM, která mnoho problémů řeší
- Pro zvýšení efektivity možné vyzkoušet GRU