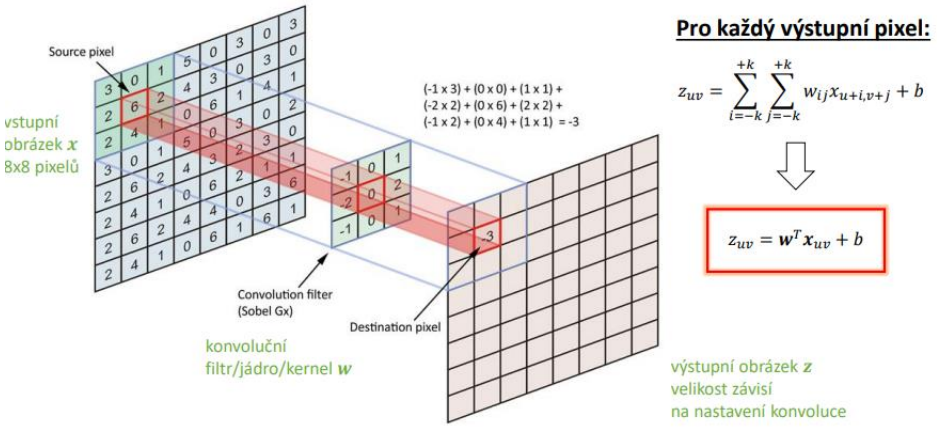
14. Konvoluční neuronové sítě a jejich aplikace. Rekurentní neuronové sítě a jejich aplikace, modely typu LSTM a GRU.

Konvoluční neuronové sítě a jejich aplikace.

Dvourozměrná konvoluce



Konvoluce jako vrstva v neurosíti

Váhy W jsou tensor tvaru

$K \times K' \times C \times F$

Bias b je vektor délky

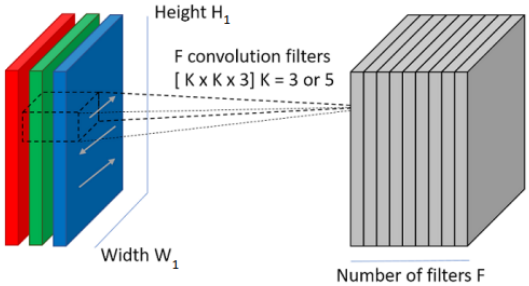
F

Výstup má rozměr

$$Q = \left\lfloor \frac{M + 2P - K}{S} \right\rfloor + 1$$

Hyperparametry:

- velikost filtru K
- počet filtrů F
- padding (okraj) P
- stride (krok) S



Input Layer (RGB pixels)
[H x W x 3]

Convolution Layer Output
[H x W x F]
assuming stride=1 and zero padding

Max pooling

- Nejčastější forma poolingu
- Robustní vůči malému posunu vstupu

Počet parametrů: 0

výstup:

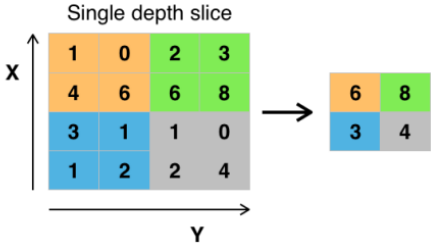
maximum přes každé okénko

vždy pouze pro jeden kanál vstupu → redukuje pouze v x a y prostoru

příklad:

vstup: 32x32x3
výstup: 16x16x3

např. 2x2 max pooling, stride=2:

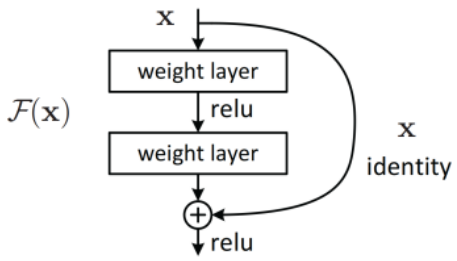


Reziduální blok

- Podobně jako inception používá složitější bloky
- Výstup sestává ze součtu konvoluce a přímo mapovaného vstupu (identity)
- Síť se tedy učí pouze rezidua

$$\mathcal{F}(\mathrm{x}) = \mathcal{H}(\mathrm{x}) - \mathrm{x}$$

- "Naučit se nuly je jednodušší než identitu"



$$\mathcal{H}(x) = \mathcal{F}(x) + \mathrm{x}$$

Figure 2. Residual learning: a building block.

Využití konvolučních sítí:

Generování příznaků – bottlenet

Především práce s obrazem – detekce, klasifikace atd

Rekurentní neuronové sítě a jejich aplikace

• Rekurentní sítě vhodné pro sekvenční data

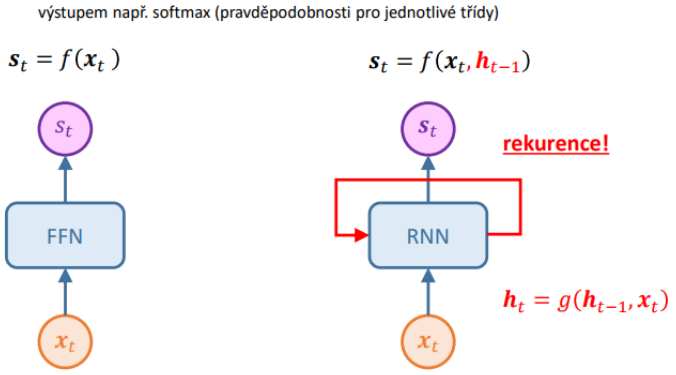
• Lze modelovat např. jazyk → jazykový model

• Použijeme znakový model na úrovni písmen → "slovníkem" je abeceda + interpunkce

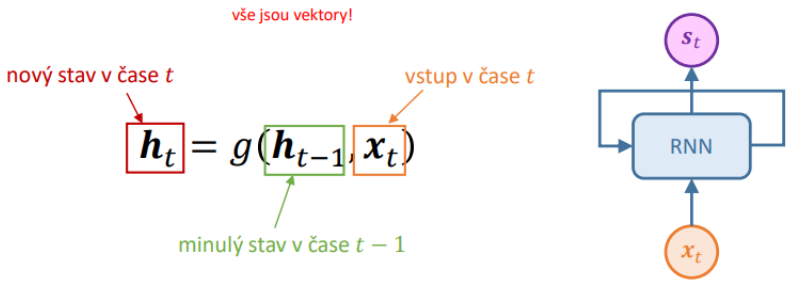
• Vstup: vstupní znak

• 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

Standardní vs rekurentní síť



Vnitřní stav rekurentní sítě



Dopředný průchod "vanilla" RNN

Rekurentní síť má v každém kroku **dva vstupy** $(\boldsymbol{x}_t, \boldsymbol{h}_{t-1})$ a **dva výstupy** $(\boldsymbol{h}_t, \boldsymbol{s}_t)$

1. výstup $\boldsymbol{h}_t = \tanh\left(W^{xh}\boldsymbol{x}_t + W^{hh}\boldsymbol{h}_{t-1} + b^h\right)$

$$\boldsymbol{h}_t = \tanh\left(W^{xh}\cdot \boldsymbol{x}_t + W^{hh}\cdot \boldsymbol{h}_{t-1} + b^h\right)$$

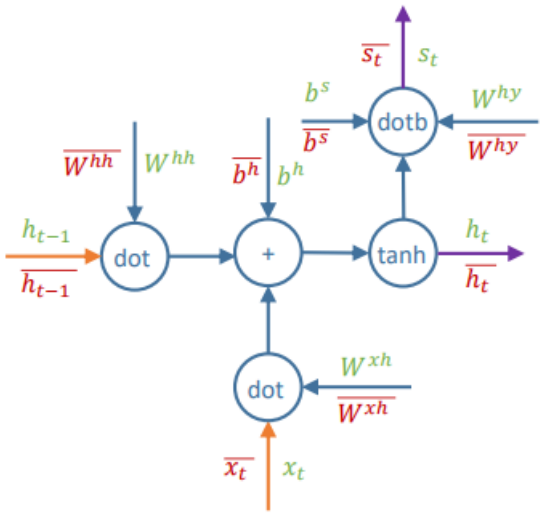
jako nelinearita se u RNN používá $g(\cdot) = \tanh(\cdot)$

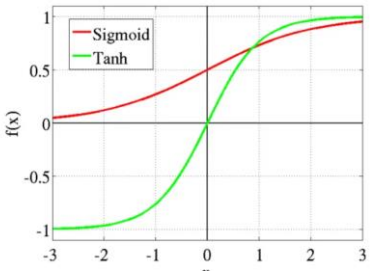
2. výstup $\boldsymbol{s}_t = W^{hs}\boldsymbol{h}_t + b^s$

$$\boldsymbol{s}_t = W^{hs}\cdot \boldsymbol{h}_t + b^s$$

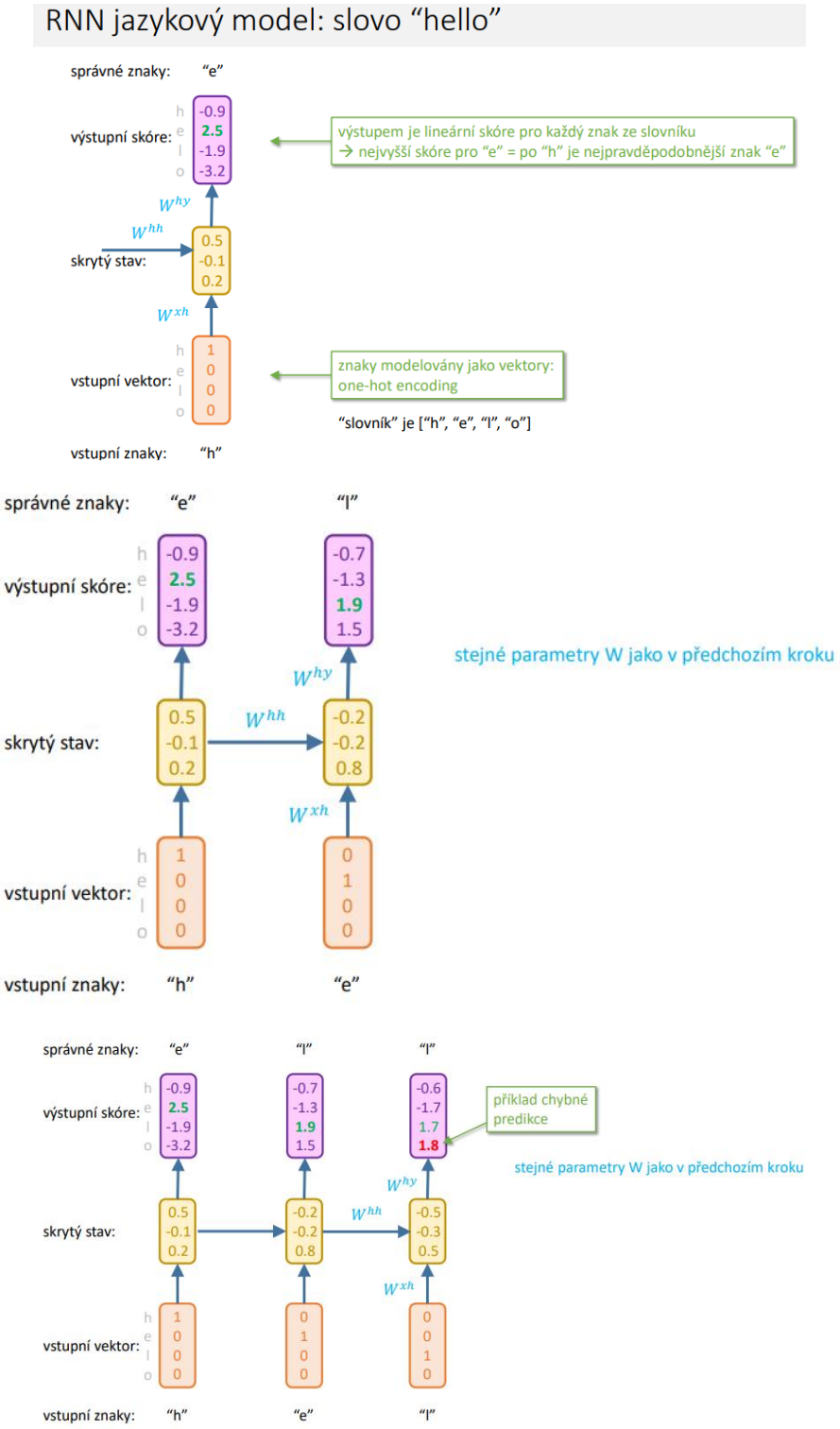
tři různé váhové matice W^{xh}, W^{hh}, W^{hs}

rekurentní síť:





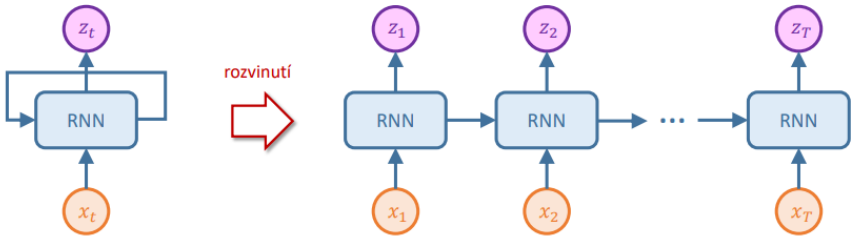
Základní RNN:



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

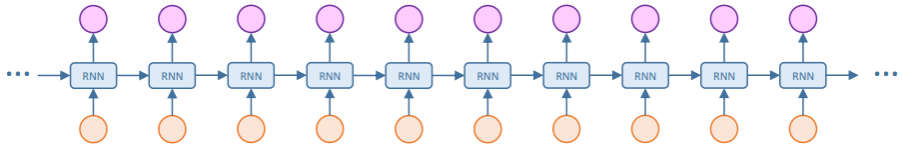
→ celá trénovací sekvence jako jeden velký výpočetní graf!



Problematika:

Dlouhé sekvence

velký výpočetní graf →
podobný problém jako u velmi hlubokých sítí:
mizející/explodující gradient (vanishing/exploding gradient)

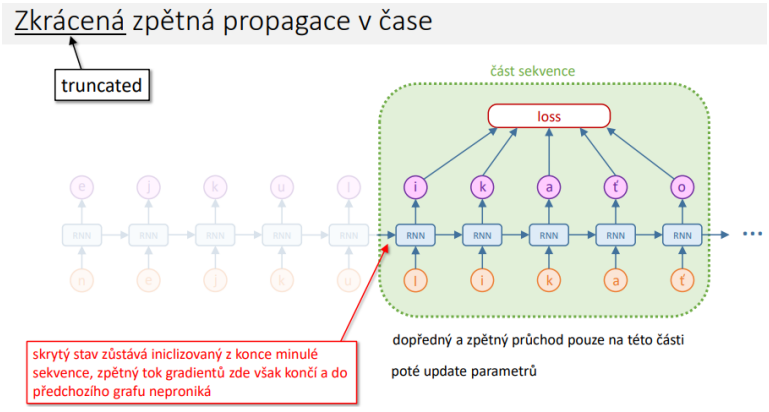


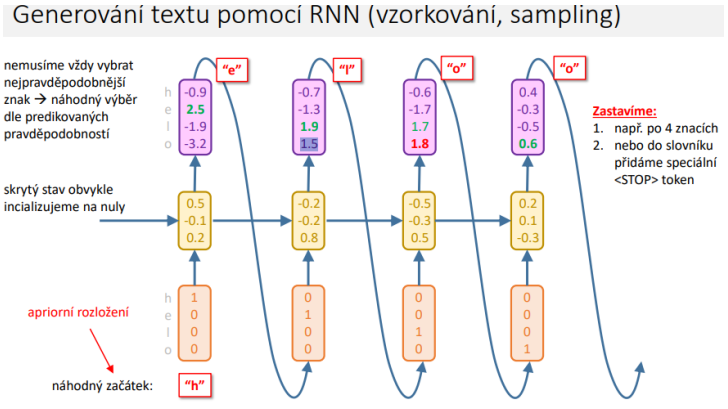
řešení:

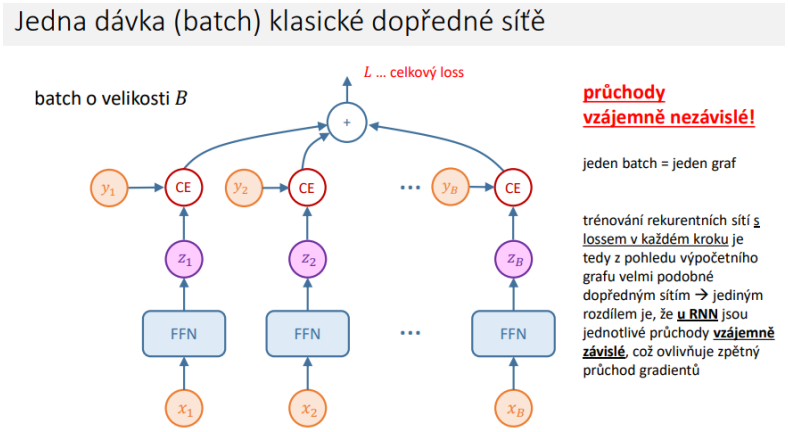
1. rozdělení sekvence na menší části → truncated backpropagation through time
2. lepší architektura (LSTM, více dále)

Vylepšení rozdělení na sekvence:

Zkrácená zpětná propagace v čase







Příklady:

Automatické generování textu

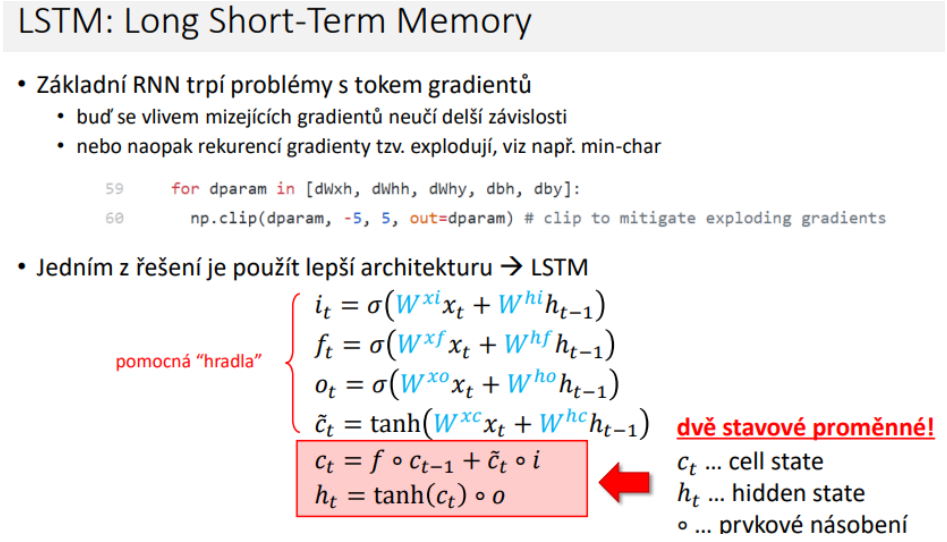
Chatbot

Tagování obrázků

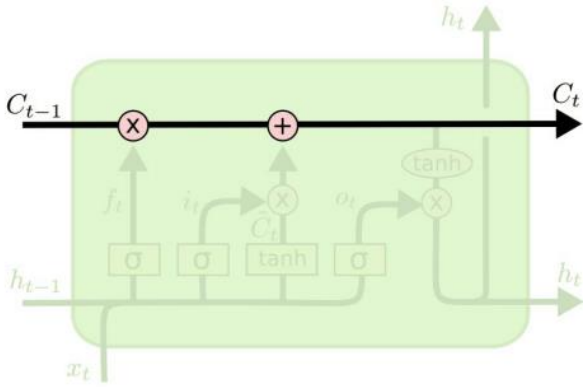
Překlad textu

Modely typu LSTM a GRU

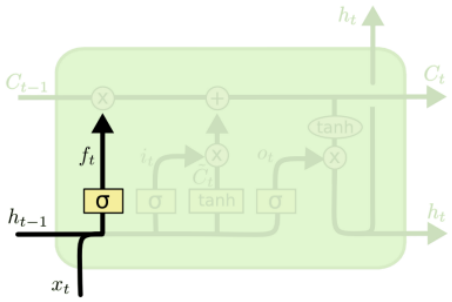
LSTM



podobné jako reziduální spoje (identity mapping connection)



LSTM: forget gate

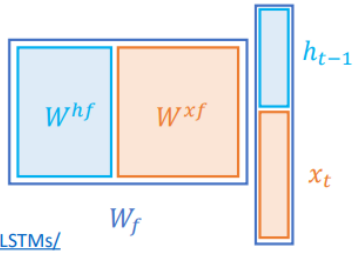


upravuje, co se zapomene, reguluje *c*

ekvivalentní zápis

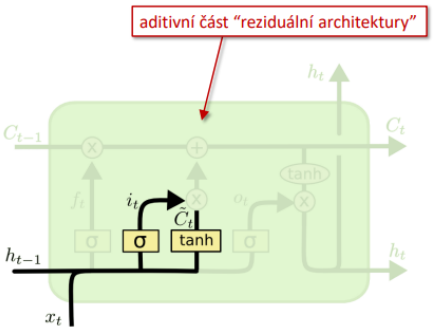
$$W^{hf} h_{t-1} + W^{xf} x_t$$

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



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

LSTM: input gate a update cell stavu



$$c_t = f_t \circ c_{t-1} + \tilde{c}_t \circ i_t$$

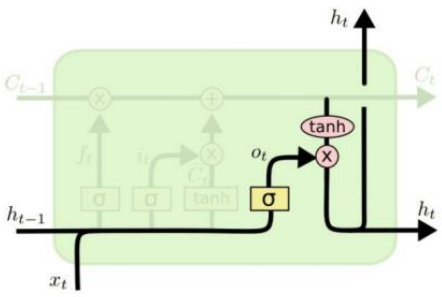
update cell stavu

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

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

rozhoduje, co se zapíše nového do cell stavu

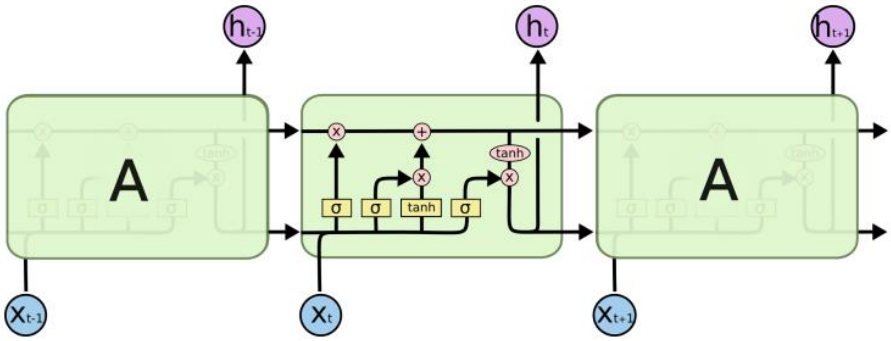
LSTM: output gate



$$o_t = \sigma\left(W_o\ [h_{t-1}, x_t]\ +\ b_o\right)$$
$$h_t = o_t * \tanh\left(C_t\right)$$

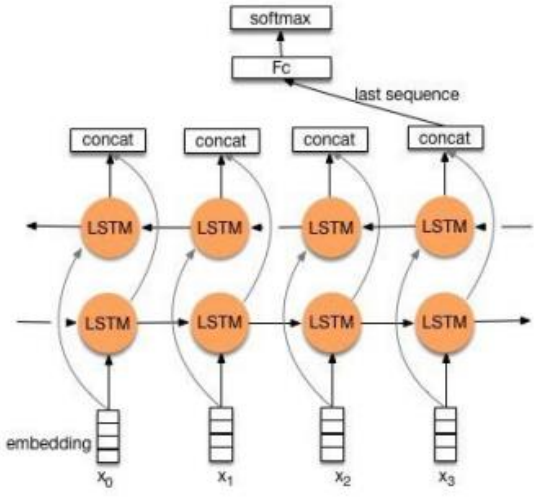
jak se zkombinuje cell stav a hidden stav pro vygenerování výstupu

LSTM rozvinutá v čase



Vylepšení:

Vstupní sekvence je procházena z obou směrů

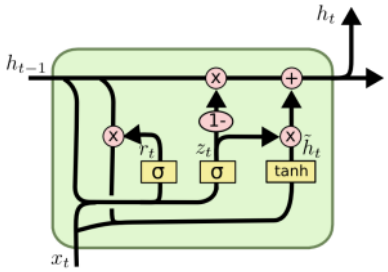


http://doc.paddlepaddle.org/develop/doc/_images/bi_lstm.jpg

GRU

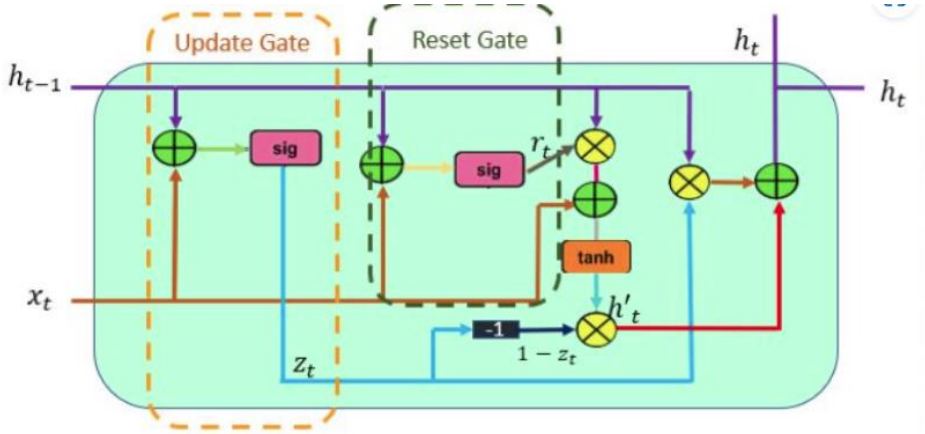
Gated Recurrent Unit (GRU)

variace na LSTM → zjednodušení, pouze jedna stavová proměnná h_t



$$z_t = \sigma\left(W_z \cdot [h_{t-1}, x_t]\right)$$
$$r_t = \sigma\left(W_r \cdot [h_{t-1}, x_t]\right)$$
$$\tilde{h}_t = \tanh\left(W \cdot [r_t * h_{t-1}, x_t]\right)$$
$$h_t = (1 - z_t) * h_{t-1} + z_t * \tilde{h}_t$$

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



Prostor pro dopsání rozdílů: