Tomáš Oberhuber

Faculty of Nuclear Sciences and Physical Engineering
Czech Technical University in Prague

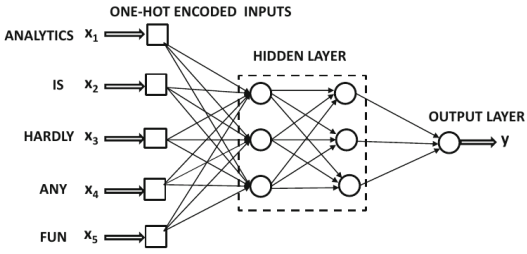
Rekurentní neuronové sítě

- plně propojené neuronové sítě jsou navržené pro zpracování mnohadimenzionálních dat bez zřejmých vzájemných závislostí
- pro některé specifické typy dat je ale výhodnější použít specializované sítě
- pro zpracování časových řad nebo textu se velice dobře hodí rekurentní neuronové sítě
- v následujícím výkladu se budeme zabývat zejména zpracováním textu

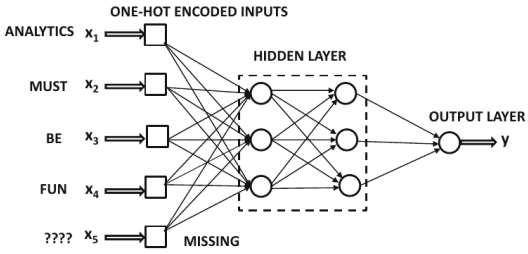
Rekurentní neuronové sítě

- jeden z hlavních problémů, se kterými se potýkáme při zpracování textu je různá celková délka
- text většinou kódujeme po slovech
- pokud bychom použili klasickou plně propojenou síť, např. různé věty by vyžadovali různý počet vstupů podle délky věty
- počet vstupů by pro delší texty musel být také extrémně velký

Rekurentní neuronové sítě



(a) 5-word sentence
"*Analytics is hardly any fun.*"



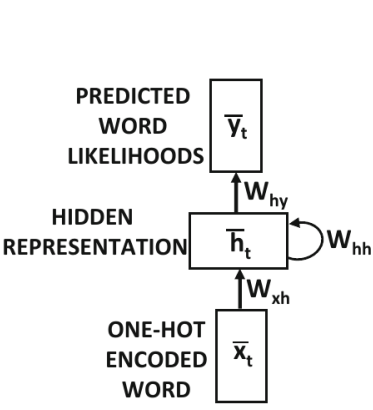
(b) 4-word sentence
"*Analytics must be fun.*"

Ch.C.Aggarwal, Neural Networks and Deep Learning.

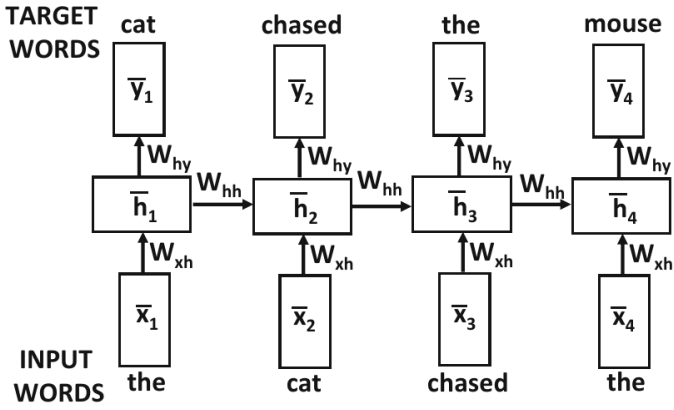
Rekurentní neuronové sítě

- naším cílem bude sestrojit NN, která bude mít fixní počet parametrů, ale bude schopna pracovat se vstupy různé délky
- RNN toto řeší tím, že jednotlivé členy posloupnosti mapuje na jednotlivé vrstvy tj. místo aby RNN měla variabilní počet vstupů, má variabilní počet vrstev a jeden vstup
- RNN navíc zavádí sdílení parametrů mezi jednotlivými vrstvami
  - to zjednodušuje trénování sítě - učíme se méně parametrů
  - jedna a ta samá vrstva navíc může snadno opakovat libovolně krát

Rekurentní neuronové sítě



(a) RNN



(b) Time-layered representation of (a)

Ch.C.Aggarwal, Neural Networks and Deep Learning.

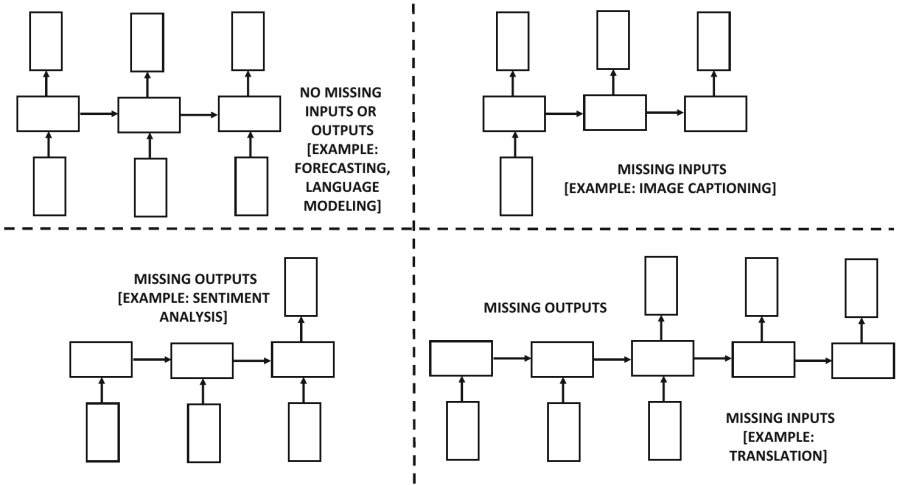
Aplikace

1. snažíme se predikovat další slovo ve větě
2. překlad z jednoho jazyka do druhého
3. generování popisků obrázků pomoci propojení s konvolučními sítěmi
4. výstupem RNN může být vektor klasifikující danou větu pro pochopení obsahu textu

RNN jsou turingovsky komplétní, tj. dokáží simulovat libovolný algoritmus. Není to ale příliš efektivní.

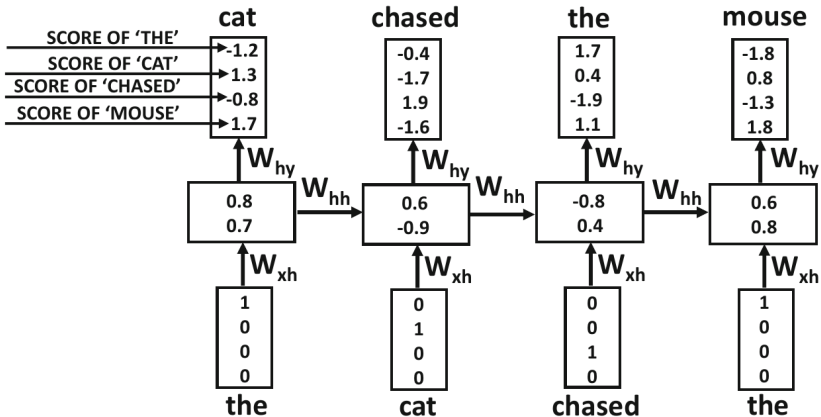
H.T.Siegelmann, E. D. Sontag, On the computational power of neural nets, COLT '92: Proceedings of the fifth annual workshop on Computational learning theory, 1992 Pages 440–449.

Aplikace



Ch.C.Aggarwal, Neural Networks and Deep Learning.

Aplikace



Ch.C.Aggarwal, Neural Networks and Deep Learning.

Architektura

Budeme používat následující značení:

- $\vec{x}^{(t)}$ je vstupní vektor v diskrétním čase t
- $\vec{y}^{(t)}$ je výstupní vektor v diskrétním čase t
- $\vec{h}^{(t)}$ je stav skryté vrstvy v diskrétním čase t
- $\mathbb{W}_{xh}, \mathbb{W}_{hh}, \mathbb{W}_{hy}$ jsou matice vnitřních vah

Obecně pak platí:

$$\begin{aligned} \vec{h}^{(t)} &= \phi(\mathbb{W}_{hh}\vec{h}^{(t-1)} + \mathbb{W}_{xh}\vec{x}^{(t)}), \\ \vec{y}^{(t)} &= \xi(\mathbb{W}_{hy}\vec{h}^{(t)}), \end{aligned}$$

pro $t = 1, 2, 3 \ldots$.
Za funkce ϕ a ξ se často volí `tanh`, sigmoid σ nebo *softmax* aplikované po složkách.

Architektura

- pro $t = 1$ definujeme $\vec{h}^{(0)} = \vec{h}_{ini} = \vec{0}$ např.
- pracujeme-li se slovníkem o velikosti d slov, pak $\vec{x}^{(t)}, \vec{y}^{(t)} \in \mathbb{R}^n$
- $\vec{x}^{(t)}$ je *one-hot* kódování daného slova
- $\vec{y}^{(t)}$ je vektor vyjadřující pravděpodobnost daných slov na výstupu
- platí

$$
\begin{aligned}
\vec{h}^{(1)} &= \phi(\vec{h}^{(0)}, \vec{x}^{(1)}), \\
\vec{h}^{(2)} &= \phi(\vec{h}^{(1)}, \vec{x}^{(2)}) = \phi(\phi(\vec{h}^{(0)}, \vec{x}^{(1)}), \vec{x}^{(2)}), \\
\vec{h}^{(3)} &= \phi(\vec{h}^{(2)}, \vec{x}^{(3)}) = \phi(\phi(\phi(\vec{h}^{(0)}, \vec{x}^{(1)}), \vec{x}^{(2)}), \vec{x}^{(3)}), \\
&\vdots \\
\vec{h}^{(t)} &= \Phi_t(\vec{x}^{(t)}, \ldots, \vec{x}^{(1)})
\end{aligned}
$$

Ztrátová funkce

- výstupní vektor transformujeme na pravděpodobnosti (pokud to již nesplňuje)

$$\vec{p}^{(t)} = softmax(y_1^{(t)}, \ldots, y_d^{(t)})$$

- ztrátovou funkci definujeme např. pomocí cross-entropy

$$L = -\sum_{t=1}^{T}\sum_{i=1}^{d} t_i^{(t)} \log p_i^{(t)},$$

  kde $\vec{t}^{(t)}$ (target,ground-truth) je požadovaný výstup RNN, tj. one-hot kódování správného slova.
- gradient této ztrátové funkce již známe

$$\frac{\partial L}{\partial y_k^{(t)}} = p_k^{(t)} - t_k^{(t)}$$

Zpětná propagace

- pro odvození zpětné propagace v kroku t si RNN představíme rozvinutou v čase a provedeme backpropagation jako u běžné NN
- budeme tedy uvažovat matice vah $\mathbb{W}_{xh}^{(t)}$, $\mathbb{W}_{hh,t}^{(t)}$ a $\mathbb{W}_{hy,t}^{(t)}$
- napočítáme gradienty

$$\frac{\partial L}{\partial W_{xh}^{(t)}}, \quad \frac{\partial L}{\partial W_{hh}^{(t)}}, \quad \frac{\partial L}{\partial W_{hy}^{(t)}}$$

- dále si uvědomíme, že v naší NN rozvinuté v čase ale musí být váhy $\mathbb{W}_{xh}^{(t)}$, $\mathbb{W}_{hh,t}^{(t)}$ a $\mathbb{W}_{hy,t}^{(t)}$ sdílené pro všechna t, tj.

$$\begin{aligned}
\mathbb{W}_{xh}^{(t)} &= \mathbb{W}_{xh} \text{ pro } t = 1, 2, \ldots, \\
\mathbb{W}_{hh}^{(t)} &= \mathbb{W}_{hh} \text{ pro } t = 1, 2, \ldots, \\
\mathbb{W}_{hy}^{(t)} &= \mathbb{W}_{hy} \text{ pro } t = 1, 2, \ldots
\end{aligned}$$

Zpětná propagace v čase

- pak tedy dostáváme

$$\frac{\partial L}{\partial \mathbb{W}_{xh}} = \sum_{t=1}^{T} \frac{\partial L}{\partial \mathbb{W}_{xh}^{(t)}},$$
$$\frac{\partial L}{\partial \mathbb{W}_{hh}} = \sum_{t=1}^{T} \frac{\partial L}{\partial \mathbb{W}_{hh}^{(t)}},$$
$$\frac{\partial L}{\partial \mathbb{W}_{hy}} = \sum_{t=1}^{T} \frac{\partial L}{\partial \mathbb{W}_{hy}^{(t)}}.$$

- často se používá pojem *backpropagation through time* - BPTT

- v aplikacích, kde T může být hodně velké se provádí tzv. *truncated BPTT*
- podobně jako u SGD nepočítám gradient přes všechny časové vrstvy, ale např. je pro posledních 100 kroků

Obousměrné sítě

- klasické RNN mohou reagovat jen na historii vstupních dat
- někdy je ale výhodnější znát celý kontext, tj. reagovat na data, která teprve následují
- za tím účelem se používají obousměrné RNN
- místo vnitřních stavů $\vec{h}^{(t)}$ tak budeme mít stavy $\vec{h}_f^{(t)}$ a $\vec{h}_b^{(t)}$
- místo matic vah $\mathbb{W}_{xh}$, $\mathbb{W}_{hh}$ a $\mathbb{W}_{hy}$ tak budeme mít matice

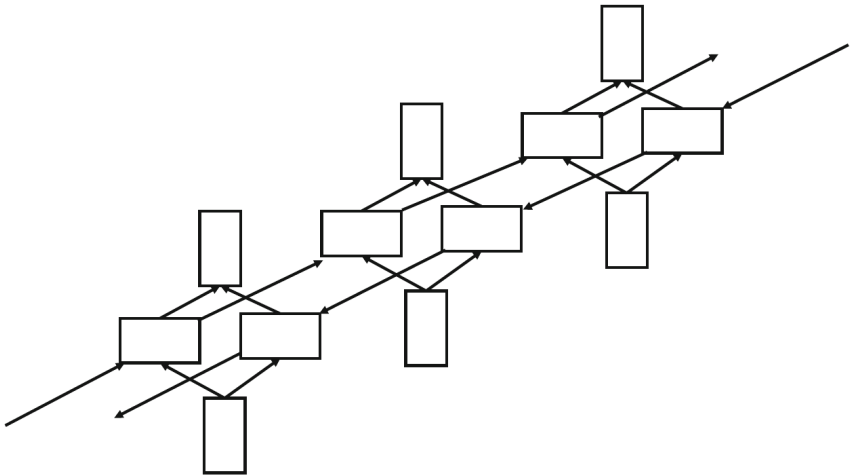
$$\mathbb{W}_{fxh}, \mathbb{W}_{fhh}, \mathbb{W}_{fhy} \quad \text{a} \quad \mathbb{W}_{bxh}, \mathbb{W}_{bhh}, \mathbb{W}_{bhy}$$

- síť je definována vztahy

$$\begin{aligned}
\vec{h}_f^{(t)} &= \phi(\mathbb{W}_{fxh}\vec{x}^{(t)} + \mathbb{W}_{fhh}\vec{h}_f^{(t-1)}),\\
\vec{h}_b^{(t)} &= \phi(\mathbb{W}_{bxh}\vec{x}^{(t)} + \mathbb{W}_{bhh}\vec{h}_b^{(t+1)}),\\
\vec{y}^{(t)} &= \xi(\mathbb{W}_{fhy}\vec{h}_f^{(t)} + \mathbb{W}_{bhy}\vec{h}_b^{(t)})
\end{aligned}$$

- všimneme si, že se zde nevyskytuje žádná interakce mezi dopřednými a zpětnými vnitřními stavy, tj. oba směry jsou na sobě nezávislé
- lze tedy nejprve provést dopředný chod a napočítat stavy $\vec{h}_f^{(t)}$ a následně provést zpětný chod a napočítat $\vec{h}_b^{(t)}$
  - případně mohou provést i oba chody současně, tj. paralelně
- následně se znalosti $\vec{h}_f^{(t)}$ a $\vec{h}_b^{(t)}$ lze dopočítat $\vec{y}^{(t)}$

Obousměrné sítě



Ch.C.Aggarwal, Neural Networks and Deep Learning.

Zpětná propagace pak probíhá takto:

- napočítají se vnitřní stavy $\vec{h}_f^{(t)}$ a $\vec{h}_b^{(t)}$
- napočítají se výstupní stavy $\vec{y}^{(t)}$
- napočítají se parciální derivace ztrátové funkce vůči výstupům sítě $\frac{\partial L}{\partial \vec{y}^{(t)}}$
- napočítají se parciální derivace ztrátové funkce vůči vnitřním stavům sítě

$$\frac{\partial L}{\partial \vec{h}_f^{(t)}} \text{ a } \frac{\partial L}{\partial \vec{h}_b^{(t)}}$$

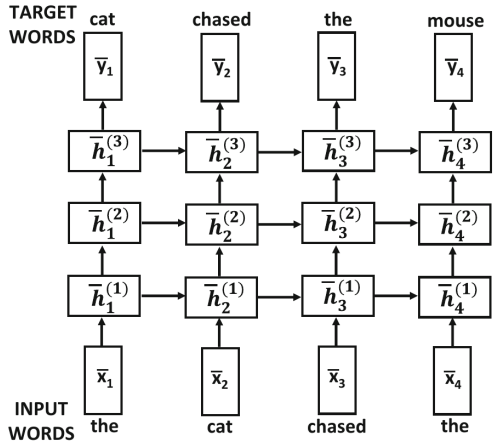
- napočítají se parciální derivace ztrátové funkce vůči parametrům sítě

$$\frac{\partial L}{\partial \mathbb{W}_{fxh}^{(t)}}, \frac{\partial L}{\partial \mathbb{W}_{fhh}^{(t)}}, \frac{\partial L}{\partial \mathbb{W}_{fhy}^{(t)}} \quad \text{a} \quad \frac{\partial L}{\partial \mathbb{W}_{bxh}^{(t)}}, \frac{\partial L}{\partial \mathbb{W}_{bhh}^{(t)}}, \frac{\partial L}{\partial \mathbb{W}_{bhy}^{(t)}}$$

- následně se sečtou pro výpočet parciálních derivací vůči sdíleným parametrům

Vícevrstvé RNN

- v reálných aplikacích se používají vícevrstvé RNN
- mají více než jeden vnitřní stav
- váhy na jednotlivých vrstvách jsou opět sdílené pro všechny kroky $t = 1, 2, 3, \ldots$



Ch.C.Aggarwal, Neural Networks and Deep Learning.

- nejprve upravíme vztah pro RNN s jednou vrstvou

$$\vec{h}^{(t)} = \phi\left(\mathbb{W}_{xh}\vec{x}^{(t)} + \mathbb{W}_{hh}\vec{h}^{(t-1)}\right) = \phi\left(\mathbb{W}\begin{pmatrix}\vec{x}^{(t)}\\ \vec{h}^{(t-1)}\end{pmatrix}\right),$$

kde

$$\mathbb{W} = (\mathbb{W}_{xh}, \mathbb{W}_{hh}).$$

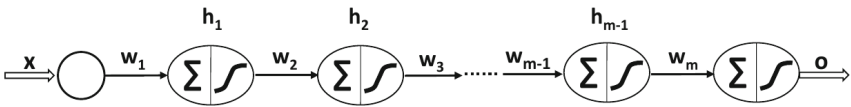
- s tímto značením pak můžeme psát

$$\underbrace{\vec{h}^{(t,k)}}_{h_t^{(k)} \text{ na obrázku}} = \phi\left(\mathbb{W}^{(k)}\begin{pmatrix}\vec{h}^{(t,k-1)}\\ \vec{h}^{(t-1,k)}\end{pmatrix}\right)$$

kde $\vec{h}^{(t,0)} = \vec{x}^{(t)}$.

Problematika trénování RNN

- hloubka RNN je dána délkou vstupu, RNN proto mohou mít velký počet vrstev
- připomeneme si zjednodušený příklad DNN



- ukázeli jsme, že platí

$$\frac{\partial L}{\partial h_1} = \prod_{i=1}^{l} \phi'(h_i) w_{i+1}$$

Problematika trénování RNN

- a protože v RNN jsou váhy sdílené, tj. $w_i = w$, dostávám

$$\frac{\partial L}{\partial h_1} = \prod_{i=1}^{l} \phi'(h_i) w$$

- je-li tedy nějaká váha hodně malá nebo velká, problém mizejícího nebo explodujícího gradientu se projeví o to více
- učení RNN je o to více komplikované a je potřeba věnovat větší pozornost postupům pro ošetření mizejících/explodujících gradientů
- u RNN se dost osvědčuje *gradient clipping*

Echo-State Networks

- jde o velice zajímavý nápad, jak zjednodušit a urychlit učení RNN
- reálné RNN mají vždy více skrytých vrstev
- echo-state NN dělají to, že váhy na vnitřních vrstvách se nastaví náhodně a učí se jen poslední výstupní vrstva
  - je potřeba ale zaručit, že spektrální poloměr matic vah bude ≤ 1 a některá další doporučení
- podstata je v tom, že skryté vrstvy musí být např. řádově větší než vstupní data
- náhodně nastavené vnitřní vrstvy tak vlastně vytváří mnoho náhodných transformací vstupních dat, ze kterých si pak výstupní vrstva během trénování vybere jen ty transformace, které dávají smysl
- toto funguje ale jen pro RNN s menšími vstupními daty, tj. menší dimenzí vstupních vektorů

Long Short-Term Memory - LSTM

- jeden pohled na problém mizejících a explodujících gradientů je ten, že RNN jsou dobré spíše na učení kratších sekvencí
- RNN tedy jakoby mají spíše kratkou paměť
- LSTM je způsob, jak RNN pomoci k dlouhodobější paměti
- kromě skrytých vrstev $\vec{h}^{(k)}$ zavedem ještě další pomocné stavy $\vec{c}^{(k)}$ (*cell state*), které budou právě reprezentovat kumulování informace a tím pádem dlouhodobější paměť

Long Short-Term Memory - LSTM

- připomeneme si vztah pro skryté vrstvy u vícevrtsvé RNN

$$\vec{h}^{(t,k)} = \phi\left(\mathbb{W}^{(k)}\begin{pmatrix}\vec{h}^{(t,k-1)}\\ \vec{h}^{(t-1,k)}\end{pmatrix}\right)$$

kde $\vec{h}^{(t,0)} = \vec{x}^{(t)}$.

- dále zavedem následující pomocná hradla (*gates*)
  - $\vec{g}_i$ - vstupní hradlo - *input gate*
  - $\vec{g}_f$ - hradlo zapomínání - *forget gate*
  - $\vec{g}_o$ - výstupní hradlo - *output gate*
  - $\vec{c}$ - nový $\vec{c}$-stav

- $\vec{g}_i, \vec{g}_f, \vec{g}_o, \vec{c} \in \mathbb{R}^p$, kde p je dimenze $\vec{h}^{(t,k)}$
- síť je řízena následujícím předpisem

$$\begin{pmatrix}\vec{g}_i\\ \vec{g}_f\\ \vec{g}_o\\ \vec{c}\end{pmatrix} = \begin{pmatrix}\text{sigm}\\ \text{sigm}\\ \text{sigm}\\ \tanh\end{pmatrix}\begin{pmatrix}\mathbb{W}_i^{(k)}\\ \mathbb{W}_f^{(k)}\\ \mathbb{W}_o^{(k)}\\ \mathbb{W}_c^{(k)}\end{pmatrix}\begin{pmatrix}\vec{h}^{(t,k-1)}\\ \vec{h}^{(t-1,k)}\end{pmatrix}$$

$$\vec{c}^{(t,k)} = \vec{g}_f \odot \vec{c}^{(t-1,k)} + \vec{g}_i \odot \vec{c}$$

$$\vec{h}^{(t,k)} = \vec{g}_o \odot \tanh(\vec{c}^{(t,k)})$$

- kde $\odot$ značí násobení po složkách

Long Short-Term Memory - LSTM

- pomocné vektory $\vec{g}_i, \vec{g}_f, \vec{g}_o$ mají opravdu význam logických hradel a tedy jejich hodnoty by měly být binární
- pro účely gradientního učení ale potřebujeme, aby to byly spojité funkce, proto používáme sigmoid jako nelinearitu
- násobení takových funkcí pak lze chápat jako zobecnění logické operace AND
- *cell state* přepočítáváme jako

$$\vec{c}^{(t,k)} = \underbrace{\vec{g}_f \odot \vec{c}^{(t-1,k)}}_{\text{zahodit předchozí historii?}} + \underbrace{\vec{g}_i \odot \vec{c}}_{\text{přidat nové } \vec{c}?}$$

- vnitřní stavy přepočítáváme jako

$$\vec{h}^{(t,k)} = \underbrace{\vec{g}_o \odot \tanh(\vec{c}^{(t,k)})}_{\text{cell state selektivně přepouštíme do } \vec{h}^{(t,k)}}$$

Long Short-Term Memory - LSTM

- ukážeme se jednoduchý příklad s jednou skrytou vrstvou a dimenzí jedna

$$\begin{pmatrix} g_i \\ g_f \\ g_o \\ c \end{pmatrix} = \begin{pmatrix} \text{sigm} \\ \text{sigm} \\ \text{sigm} \\ \tanh \end{pmatrix} \begin{pmatrix} w_{i1} & w_{i2} \\ w_{f1} & w_{f2} \\ w_{o1} & w_{o2} \\ w_{c1} & w_{c2} \end{pmatrix} \begin{pmatrix} h^{(t)} \\ h^{(t-1)} \end{pmatrix}$$

$$c^{(t)} = g_f c^{(t-1)} + g_i c$$

$$h^{(t)} = g_o \tanh(c^{(t)})$$

- pro gradient pak dostáváme

$$\nabla_{\vec{w}} h^{(t)} = \nabla_{\vec{w}} g_o \tanh(c^{(t)}) + g_o \tanh'(c^{(t)}) \nabla_{\vec{w}} c^{(t)}$$

$$\nabla_{\vec{w}} c^{(t)} = \nabla_{\vec{w}} g_f c^{(t-1)} + g_f \nabla_{\vec{w}} c^{(t-1)} + \nabla_{\vec{w}} g_i c + g_i \nabla_{\vec{w}} c$$

Long Short-Term Memory - LSTM

- gradienty hradel vypadají takto

$$
\begin{aligned}
\nabla_{(w_{i1},w_{i2})} g_i &= \sigma'\left(w_{i1}h^{(t)} + w_{i2}h^{(t-1)}\right)\begin{pmatrix} h^{(t)} \\ h^{(t-1)} \end{pmatrix} \\
\nabla_{(w_{f1},w_{f2})} g_f &= \sigma'\left(w_{f1}h^{(t)} + w_{f2}h^{(t-1)}\right)\begin{pmatrix} h^{(t)} \\ h^{(t-1)} \end{pmatrix} \\
\nabla_{(w_{o1},w_{o2})} g_o &= \sigma'\left(w_{o1}h^{(t)} + w_{o2}h^{(t-1)}\right)\begin{pmatrix} h^{(t)} \\ h^{(t-1)} \end{pmatrix} \\
\nabla_{(w_{c1},w_{c2})} c &= \tanh'\left(w_{c1}h^{(t)} + w_{c2}h^{(t-1)}\right)\begin{pmatrix} h^{(t)} \\ h^{(t-1)} \end{pmatrix}
\end{aligned}
$$

Long Short-Term Memory - LSTM

- dále pak je

$$\nabla_{(w_{i1},w_{i2},w_{f1},w_{f2},w_{c1},w_{c2})} c^{(t)} = \begin{pmatrix} c\sigma'\left(w_{i1}h^{(t)} + w_{i2}h^{(t-1)}\right)\begin{pmatrix} h^{(t)} \\ h^{(t-1)} \end{pmatrix} \\ c^{(t-1)}\sigma'\left(w_{i1}h^{(t)} + w_{i2}h^{(t-1)}\right)\begin{pmatrix} h^{(t)} \\ h^{(t-1)} \end{pmatrix} \\ g_i\sigma'\left(w_{c1}h^{(t)} + w_{c2}h^{(t-1)}\right)\begin{pmatrix} h^{(t)} \\ h^{(t-1)} \end{pmatrix} \end{pmatrix} + g_f \nabla_{\vec{w}} c^{(t-1)}$$

- a nakonec

$$\nabla_{\vec{w}} h^{(t)} = \begin{pmatrix} g_o \tanh'(c^{(t)}) c\sigma'\left(w_{i1}h^{(t)} + w_{i2}h^{(t-1)}\right)\begin{pmatrix} h^{(t)} \\ h^{(t-1)} \end{pmatrix} \\ g_o \tanh'(c^{(t)}) c^{(t-1)}\sigma'\left(w_{i1}h^{(t)} + w_{i2}h^{(t-1)}\right)\begin{pmatrix} h^{(t)} \\ h^{(t-1)} \end{pmatrix} \\ \tanh c^{(t)}\sigma'\left(w_{o1}h^{(t)} + w_{o2}h^{(t-1)}\right)\begin{pmatrix} h^{(t)} \\ h^{(t-1)} \end{pmatrix} \\ g_o \tanh'(c^{(t)}) g_i\sigma'\left(w_{c1}h^{(t)} + w_{c2}h^{(t-1)}\right)\begin{pmatrix} h^{(t)} \\ h^{(t-1)} \end{pmatrix} \end{pmatrix} + g_o \tanh'(c^{(t)}) g_f \nabla_{\vec{w}} c^{(t-1)}$$

Long Short-Term Memory - LSTM

- gradienty jsou tedy "korigovány" pomocí nelinearit σ a $\tanh$ a tedy nemohou být extrémně velké
- derivace těchto nelinearit sice mohou být nulové, ale děje se to systematicky, tj. je to dáno návrhem sítě a ne tím, že se někde nečekaně pronásobí hodně malých vah
- *cell state* $\vec{c}^{(t)}$ může přímo ovlivňovat $\vec{h}^{(t)}$ a nese v sobě vlastně historii předchozích vstupů - jeho předpis závisí na $\vec{c}^{(t-1)}$
  - hradlo $\vec{g}_f$ řídí, zda se tato informace přenáší dále nebo se zapomíná
- $\vec{c}^{(t)}$ se také označuje jako *leaky unit*
- obecně se tyto jednotky používají u RNN právě pro přenos informace ze vzdálenější historie, resp. udržují dlouhodobější kontext¨

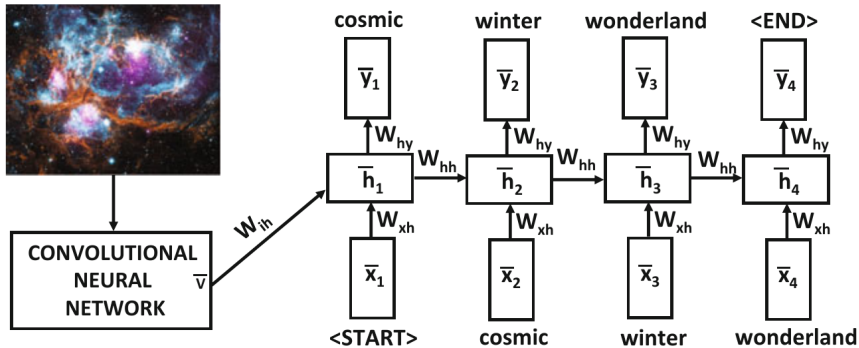
Gated Recurrent Units - GRU

- jde o sítě podobne LSTM
- nemají ale explictní pomocný stav - *cell state*
- jejich předpis je následující

$$\begin{pmatrix} \vec{z} \\ \vec{r} \end{pmatrix} = \begin{pmatrix} \sigma \\ \sigma \end{pmatrix} \begin{pmatrix} \mathbb{W}_z^{(k)} \\ \mathbb{W}_r^{(k)} \end{pmatrix} \begin{pmatrix} \vec{h}^{(t,k-1)} \\ \vec{h}^{(t-1,k)} \end{pmatrix}$$

$$\vec{h}^{(t,k)} = \vec{z} \odot \vec{h}^{(t-1,k)} + (1 - \vec{z}) \odot \tanh \mathbb{W}_h^{(k)} \begin{pmatrix} \vec{h}^{(t,k-1)} \\ \vec{r} \odot \vec{h}^{(k,t-1)} \end{pmatrix}$$

Generování popisků k obrázkům



Ch.C.Aggarwal, Neural Networks and Deep Learning.

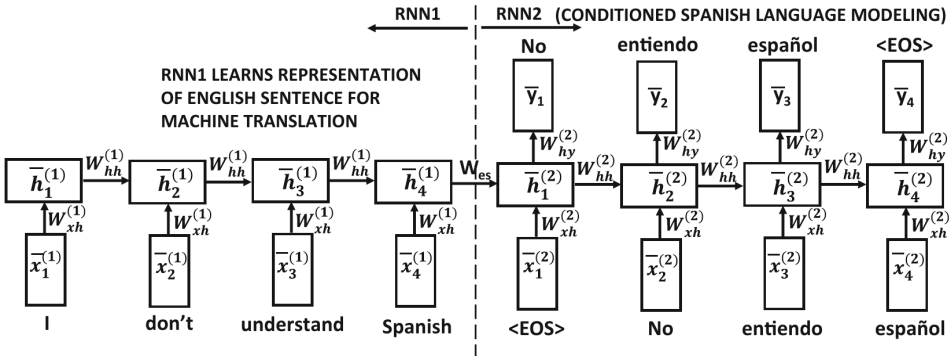
Generování popisků k obrázkům

- CNN extrahuje obsah obrázku do formy vektoru $\vec{v}$, který je pak předán RNN, která z něj vygeneruje popisek
- předpis pro RNN je následující

$$\begin{aligned}
\vec{h}^{(1)} &= \tanh\left(\mathbb{W}_{xh}\vec{x}^{(1)} + \mathbb{W}_{ih}\vec{v}\right) \\
\vec{h}^{(t)} &= \tanh\left(\mathbb{W}_{xh}\vec{x}^{(t)} + \mathbb{W}_{hh}\right) \text{ pro } t \geq 2 \\
\vec{y}^{(t)} &= \mathbb{W}_{hy}\vec{h}^{(t)}
\end{aligned}$$

- obě sítě jsou trénovány současně, tj. dostávají dvojice obrázek-popisek a podle nich se pomocí SGD přepočítávají váhy CNN i RNN

Překlad textu



Ch.C.Aggarwal, Neural Networks and Deep Learning.

Překlad textu

- zde propojujeme dvě RNN
- první načítá překládanou větu a obsah transformuje do vnitřních stavů
- výstup z RNN1 je předán RNN2, která generuje větu v jiném jazyce
- problém může nastávat u dlouhých vět
- pak se někdy používá trik, kdy se vstupní věta načítá v opačném pořadí
- to způsobí, že vnitřní stavy v posledních RNN1 krocích načítání jsou více ovlivněny začátkem věty
- díky tomu mohou přesněji generovat začátek přeložené věty
- to je důležité pro úspěšné následující pokračování generované věty

Odpovídání na otázky

- na odpovědi na otázky se můžeme dívat jako na *sentence-to-sentence modelling* podobně jako u překladu z jednoho jazyka do druhého
- příkladem mohou být tyto dvě věty

  Who is the director of the Godfather movie. $\rightarrow$ *The director is Francis Ford Coppola.*
- rozdíl oproti překladu je v tom, že první věta neobsahuje veškerou potřebnou informaci k vygenerování druhé věty
- jedno možné řešení je využít tzv. *memory networks*

J. Weston, S. Chopra, A. Bordes, Memory networks, arXiv:1410.3916, 2015.

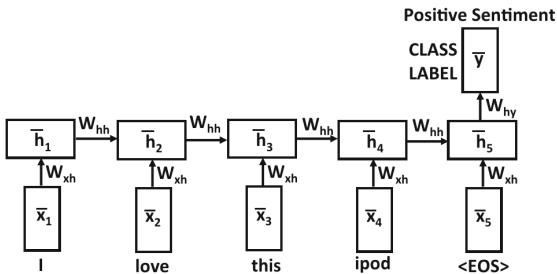
J. Weston, A. Bordes, S. Chopra, A. M. Rush, B. van Merriënboer, A. Joulin, T. Mikolov, Towards AI-Complete Question Answering: A Set of Prerequisite Toy Tasks, arXiv:1502.05698, 2015.

Odpovídání na otázky

- druhé možné řešení je využít RNN pro překlad otázky do podoby vhodné např. pro databázový dotaz
- například
  Who is the director of the Godfather movie. $\rightarrow$ **directorOf**("*Godfather*")

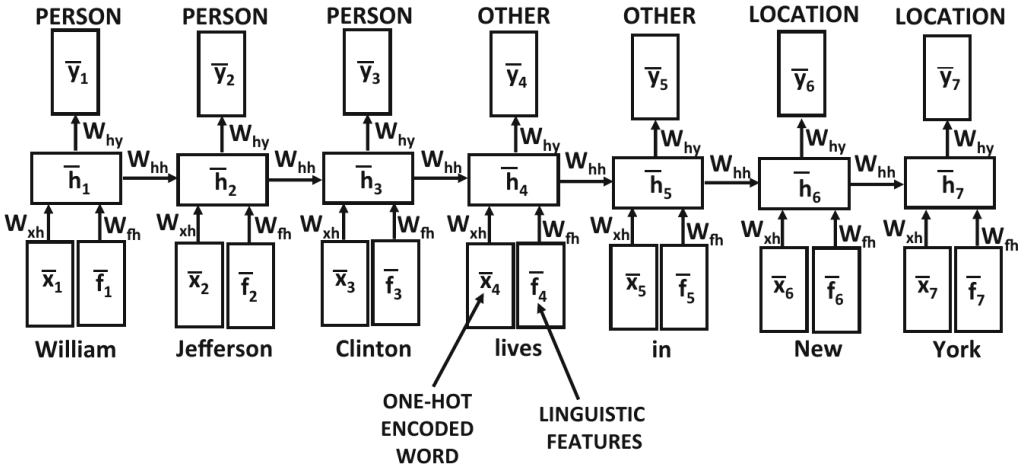
Analýza vět

- cílem je např. určit, zda věta vyjadřuje pozitivní nebo negativní pocit (*sentiment*) - *sentiment analysis*



Ch.C.Aggarwal, Neural Networks and Deep Learning.

Klasifikace slov ve větě



Ch.C.Aggarwal, Neural Networks and Deep Learning.

Predikce časových řad

- v této úloze nejde o práci s textem, ale předpovídání vývoje časových řad, např.
  - teplota
  - ceny akcií nebo některých výrobků
- problém jev tom, že RNN mají problémy se zpracováním dlouhých řad
- jako jedno z řešení se nabízí použití echo-state sítí - pokud počet predikovaných řad není příliš veliký
- často je také potřeba provést vhodný preprocesing dat, tj. různé normalizace a transformace

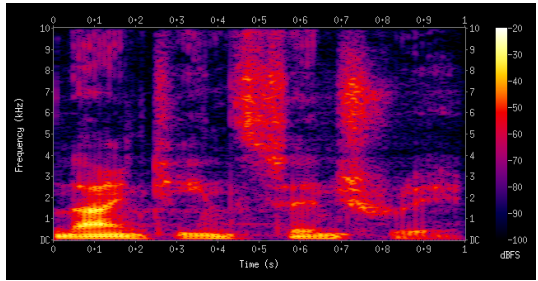
Predikce struktury proteinů

- proteiny jsou řetězce základních aminokyseliny, kterých je 23
- protein lze proto kódovat jako slovo v abecedě o 23 znacích
- cílem je odhadnout strukturu proteinu na základě znalosti posloupnosti aminokyselin, které ho tvoří
- lze použít obousměrnou RNN
- vstupem jsou jednotlivé aminokyseliny popsané pomocí *one-hot-encoding*
- výstupem je předpokládaný tvar - např. dvoušroubovice, beta skládaný list (*beta sheet*), *coiled coil*

P. Baldi, S. Brunak, P. Frasconi, G. Soda, G. Pollastri, Exploiting the past and the future in protein secondary structure prediction, Bioinformatics, vol. 15, issue 11, pp. 937–946, 1999.

Rozpoznání mluveného slova

- zde je cílem převést zvukový záznam do podoby psaného textu
- za vstup se volí spectrogram
- ten se rozdělí na pásy o šířce 254 samplů transformovaných Fourrierovou tr.
- mezi následujícími pásy je překryv 127 transformovaných sámplů
- používá se obousměrná RNN



A. Graves, N. Jaitly, Towards End-To-End Speech Recognition with Recurrent Neural Networks, Proceedings of the 31st International Conference on Machine Learning, PMLR 32(2):1764-1772, 2014.

Rozpoznání ručně psaného textu

- cílem je převést ručně psaný text do posloupnosti znaků
- ručně psaný text lze popsat jako záznam pozic pohybujícího se hrotu pera
- nejprve se extrahují příznaky jako rychlost psaní, sklon tahů, apod.
- tyto příznaky se pak předávájí jako vstup do RNN
- projevuje se zde tzv. *Sayerův paradox*
  - abychom dobře rozpoznali jednotlivé znaky, potřebovali bychom vědět, kde je začátek a konec zápisu daného znaku
  - to ale neznáme a získáme to až rozpoznáním jednotlivých znaků
  - podobný problém se vyskytuje i u rozpoznání mluveného slova

A. Graves, M. Liwicki, S. Fernández, R. Bertolami, H. Bunke, J. Schmidhuber,A Novel Connectionist System for Unconstrained Handwriting Recognition, IEEE Transactions on Pattern Analysis and Machine Intelligence, vol. 31, no. 5, pp. 855-868, 2009.