KMI/UMIN Umělá inteligence

Deep Learning

8. Sekvenční modely a vektory slov

Jan Konečný

Katedra informatiky, PřF UP v Olomouci

16. listopadu 2026

Co nás dnes čeká

Rekurentní síť

Encoder–decoder a attention

Vektorové reprezentace slov

Vstup s proměnnou délkou

Dosud jsme předpokládali, že vstup má pevnou velikost. Věta, záznam z EKG ani nahrávka řeči ji však nemají.

Rekurentní síť čte posloupnost postupně a mezi kroky si předává vnitřní stav. Váhy jsou v každém kroku tytéž, takže model funguje pro sekvence libovolné délky.

Platí se za to dvojím: kroky na sobě závisejí, a nelze je proto počítat současně, a informace z dávné minulosti se cestou ztrácí.

Druhá polovina přednášky patří tomu, jak vůbec reprezentovat slovo — k tomu se vrátíme u Word2Vecu.

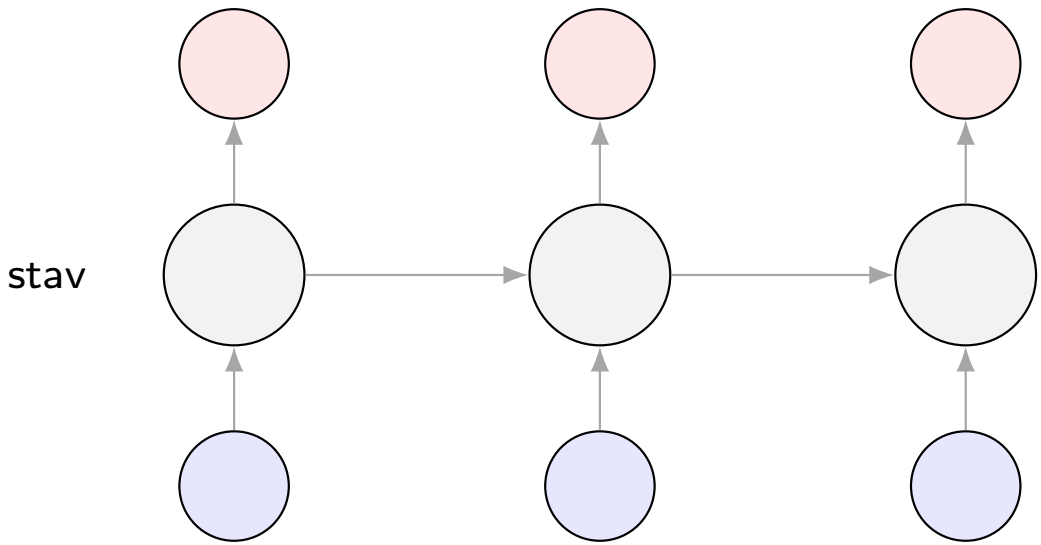
Skripta: kap. 11–12 *Sekvenční a rekurentní sítě; Word2Vec*

Rekurentní přechod

Pro vstup $\boldsymbol{x}_t$ a skrytý stav $\boldsymbol{h}_t$ platí

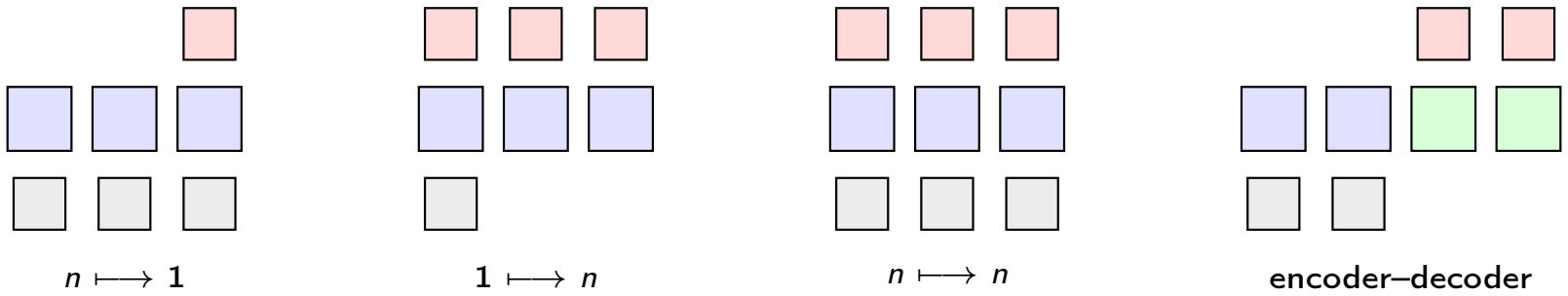
$$\boldsymbol{h}_t = \tanh\big(W_{xh}\boldsymbol{x}_t + W_{hh}\boldsymbol{h}_{t-1} + \boldsymbol{b}_h\big), \qquad \boldsymbol{y}_t = g\big(W_{hy}\boldsymbol{h}_t + \boldsymbol{b}_y\big).$$

Matice jsou sdílené přes všechny časové kroky; počáteční stav bývá nulový nebo naučený.



Podle úlohy se používá výstup jen v posledním kroku, výstup v každém kroku, nebo autoregresní generování, kdy se minulý výstup stává částí dalšího vstupu.

Čtyři tvary sekvenční úlohy



- **Sekvence $\longmapsto$ jeden výstup:** klasifikace sentimentu, diagnóza z celého záznamu.
- **Jeden vstup $\longmapsto$ sekvence:** popis obrázku větou.
- **Značkování po pozicích:** slovní druhy, detekce anomálie v čase.
- **Encoder–decoder:** překlad, kde se délka vstupu a výstupu liší a pozice si neodpovídají.

Demonstrace: dva kroky rekurence

Vezměme skalární vstup, dvourozměrný stav a nulový bias:

$$W_{xh} = \begin{pmatrix} 0{,}5 \\ -0{,}3 \end{pmatrix}, \qquad W_{hh} = \begin{pmatrix} 0{,}1 & 0{,}4 \\ -0{,}2 & 0{,}3 \end{pmatrix}, \qquad \boldsymbol{h}_0 = 0.$$

t	x_t	argument tanh	$\boldsymbol{h}_t$
1	1	$(0{,}500;\ -0{,}300)$	$(0{,}462;\ -0{,}291)$
2	-1	$(-0{,}570;\ 0{,}120)$	$(-0{,}516;\ 0{,}120)$

Ve druhém kroku se sečetly dva příspěvky: $-W_{xh}$ od nového vstupu a $W_{hh}\boldsymbol{h}_1 = (-0{,}070;\ -0{,}180)$ od stavu. Stav tedy nese informaci o prvním kroku dál, ale tanh ji zároveň stlačuje do intervalu $(-1, 1)$.

Táž dvojice matic se použije i v kroku třetím, stém i tisícím — odtud plyne jak úspora parametrů, tak potíže se stabilitou.

Backpropagation through time

BPTT rozvine síť do výpočetního grafu s jednou kopií buňky pro každý časový krok. Kopie sdílejí parametry, takže se jejich gradientní příspěvky sčítají.

Gradient vzhledem ke stavu v čase k obsahuje součin derivací $\partial \boldsymbol{h}_j / \partial \boldsymbol{h}_{j-1}$, jednu za každý krok. Jde o týž součin mnoha činitelů jako u hlubokých sítí; nenásobí se však derivace různých vrstev, ale opakovaně tentýž rekurentní přechod. Sto kroků v čase se tedy chová jako stovrstvá síť se sdílenými vahami.

Příklad 1 (Padesát kroků)

Má-li lokální derivace ve všech krocích velikost 0,9, přenese se přes 50 kroků faktor $0{,}9^{50} \doteq 0{,}0052$. Pro derivaci 1,1 vznikne $1{,}1^{50} \doteq 117$.

R. Pascanu, T. Mikolov, Y. Bengio: On the Difficulty of Training Recurrent Neural Networks. ICML (2013).

Ořezání řeší jen polovinu problému

Explodující gradient omezíme ořezáním podle normy:

$$\boldsymbol{g} \leftarrow \boldsymbol{g} \min\left(1, \frac{\tau}{\|\boldsymbol{g}\|_2}\right).$$

S mizejícím gradientem to nedělá nic — a právě ten je důvodem, proč se jednoduchá RNN neučí dlouhé závislosti.

Podobně praktické a podobně omezené je zkrácené BPTT: graf se po pevném počtu kroků odpojí, čímž se ušetří paměť, ale zároveň se zastropuje, jak daleko může gradient dosáhnout.

Buňky s branami

Nápad, na kterém stojí LSTM i GRU, je stejný jako u reziduálních spojení: vedle násobící cesty vést ještě jednu, po které se stav jen *přičítá*.

LSTM

Odděluje stav buňky $\boldsymbol{c}_t$ od výstupního stavu $\boldsymbol{h}_t$:

$$\boldsymbol{c}_t = \boldsymbol{f}_t \odot \boldsymbol{c}_{t-1} + \boldsymbol{i}_t \odot \widetilde{\boldsymbol{c}}_t, \qquad \boldsymbol{h}_t = \boldsymbol{o}_t \odot \tanh(\boldsymbol{c}_t).$$

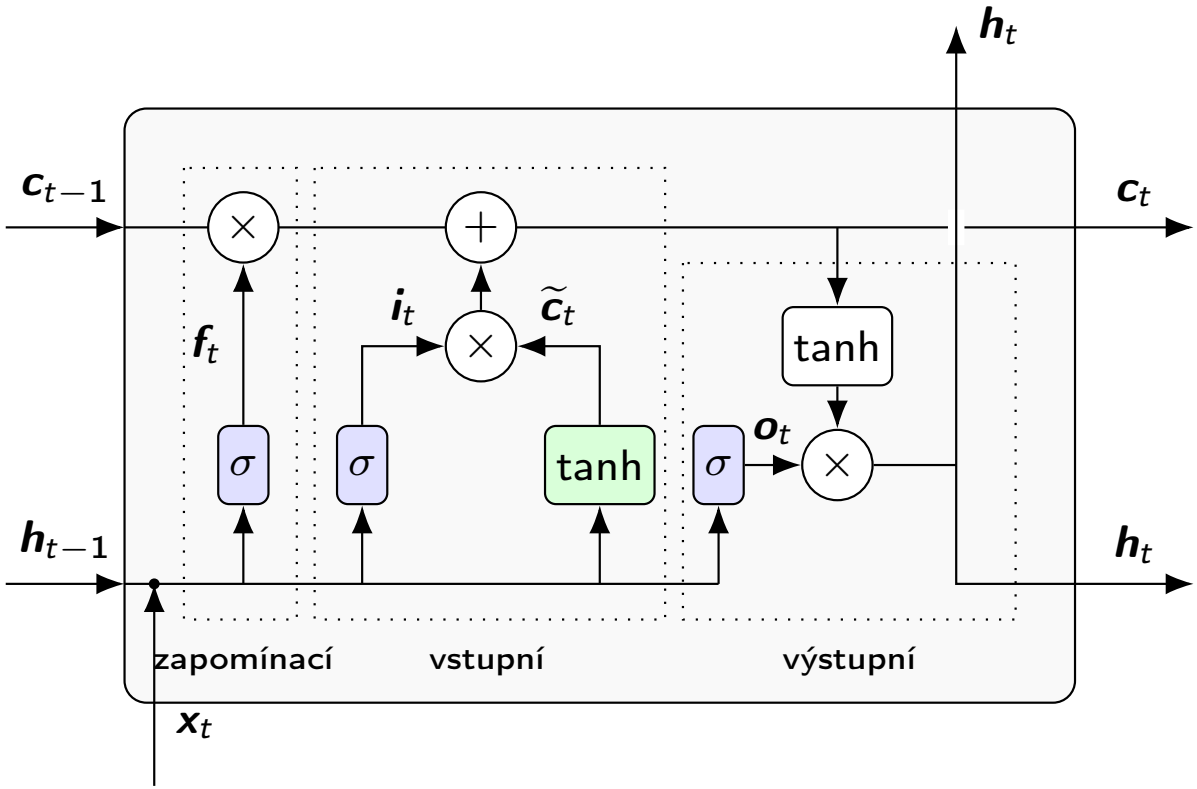
Zapomínací brána $\boldsymbol{f}_t$ řídí přenos starého stavu, vstupní brána $\boldsymbol{i}_t$ zápis kandidáta a výstupní brána $\boldsymbol{o}_t$ určuje, co se zpřístupní dál.

Příklad 2 (Stav jako řízená paměť)

Pro $c_{t-1} = 1$, $f_t = 0{,}8$, $i_t = 0{,}3$ a $\widetilde{c}_t = 0{,}5$ vyjde $c_t = 0{,}8 + 0{,}15 = 0{,}95$: buňka uchovala většinu starého stavu a přidala část nové informace.

S. Hochreiter, J. Schmidhuber: Long Short-Term Memory. *Neural Computation* 9(8) (1997).

LSTM: schéma buňky



Horní linka nese stav buňky: násobí se zapomínací branou a přičítá se k němu zápis. Ze spodní linky $[\boldsymbol{h}_{t-1}, \boldsymbol{x}_t]$ se počítají brány i kandidát. Barevné obdélníky mají vlastní váhy, bílé značky jsou operace po složkách.

LSTM: všechny brány najednou

$$
\begin{aligned}
\boldsymbol{f}_t &= \sigma\big(W_f[\boldsymbol{h}_{t-1}, \boldsymbol{x}_t] + \boldsymbol{b}_f\big) && \text{zapomínací} \\
\boldsymbol{i}_t &= \sigma\big(W_i[\boldsymbol{h}_{t-1}, \boldsymbol{x}_t] + \boldsymbol{b}_i\big) && \text{vstupní} \\
\widetilde{\boldsymbol{c}}_t &= \tanh\big(W_c[\boldsymbol{h}_{t-1}, \boldsymbol{x}_t] + \boldsymbol{b}_c\big) && \text{kandidát} \\
\boldsymbol{o}_t &= \sigma\big(W_o[\boldsymbol{h}_{t-1}, \boldsymbol{x}_t] + \boldsymbol{b}_o\big) && \text{výstupní}
\end{aligned}
$$

Klíčová je aditivní cesta $\boldsymbol{c}_t = \boldsymbol{f}_t \odot \boldsymbol{c}_{t-1} + \boldsymbol{i}_t \odot \widetilde{\boldsymbol{c}}_t$. Derivace $\partial \boldsymbol{c}_t / \partial \boldsymbol{c}_{t-1} = \boldsymbol{f}_t$ je součin bran, ne součin matic, takže při $\boldsymbol{f}_t \approx 1$ projde gradient mnoha kroky beze změny měřítka.

Praktická drobnost s velkým dopadem

Bias zapomínací brány se inicializuje kladně (typicky 1), aby buňka na začátku učení pamatovala. S nulovým biasem je $\boldsymbol{f}_t \approx 0{,}5$ a stav se každý krok půlí.

GRU a obousměrné zpracování

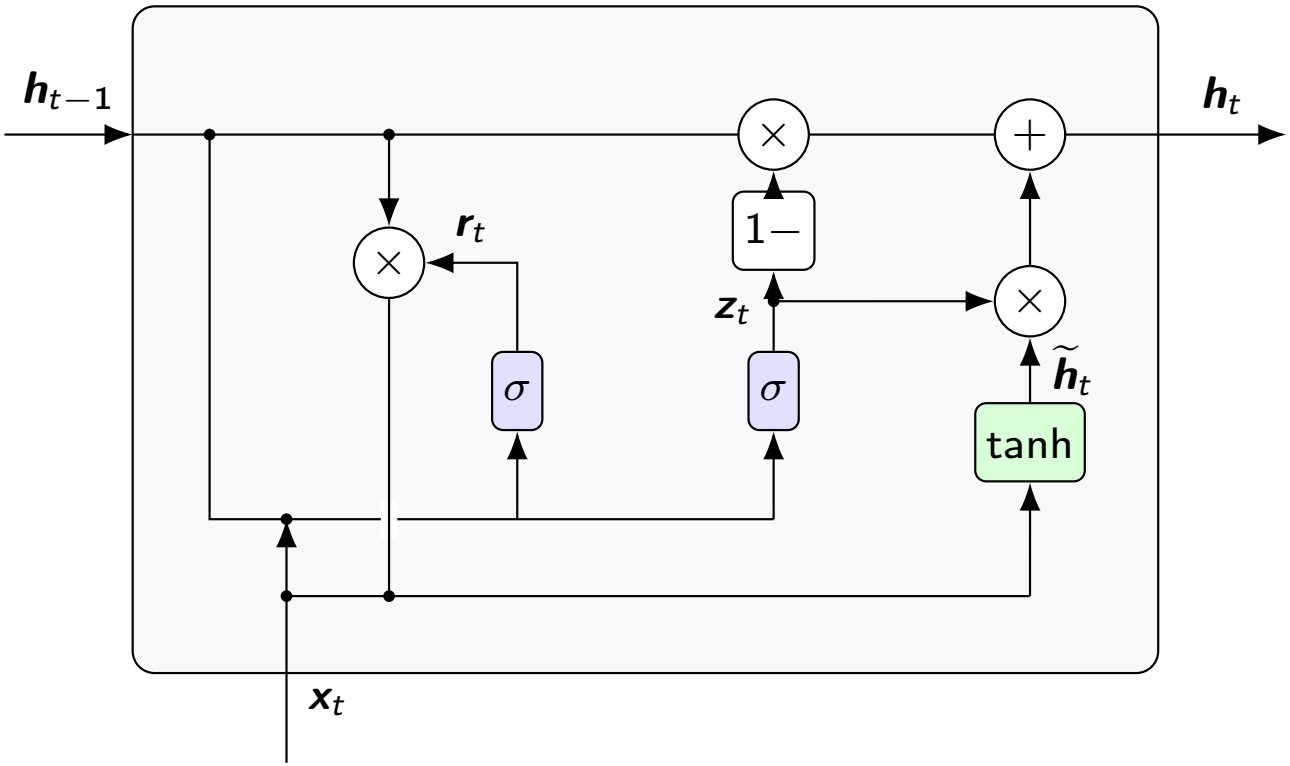
GRU spojuje paměťový a skrytý stav do jediného vektoru; aktualizační brána řídí přechod od starého stavu ke kandidátovi, resetovací brána určuje, nakolik kandidát využije minulý stav.

GRU má o jednu bránu méně a zhruba o čtvrtinu méně parametrů než stejně široká LSTM. Rozsáhlá srovnání ale nenašla jednoznačně lepší variantu; rozdíly jsou menší než vliv pečlivého naladění.

Obousměrná RNN používá dvě nezávislé rekurence a každá pozice tak využívá levý i pravý kontext. Použít ji lze ovšem jen tam, kde je celá posloupnost k dispozici předem: při značkování hotového textu ano, při předpovědi zítřejší spotřeby ne.

K. Cho a kol.: Learning Phrase Representations using RNN Encoder–Decoder for Statistical Machine Translation. EMNLP (2014).

GRU: schéma buňky



Linka nahoře je jediný stav. Aktualizační brána rozdělí váhu mezi starý stav $(1 - \boldsymbol{z}_t)$ a kandidáta $(\boldsymbol{z}_t)$. Resetovací brána vlevo ztlumí minulý stav dřív, než vstoupí do kandidáta; brány samy vidí $\boldsymbol{h}_{t-1}$ beze změny.

GRU: rovnice

$$\boldsymbol{z}_t = \sigma\big(W_z[\boldsymbol{h}_{t-1}, \boldsymbol{x}_t]\big), \qquad \boldsymbol{r}_t = \sigma\big(W_r[\boldsymbol{h}_{t-1}, \boldsymbol{x}_t]\big),$$
$$\widetilde{\boldsymbol{h}}_t = \tanh\big(W[\boldsymbol{r}_t \odot \boldsymbol{h}_{t-1}, \boldsymbol{x}_t]\big), \quad \boldsymbol{h}_t = (1 - \boldsymbol{z}_t) \odot \boldsymbol{h}_{t-1} + \boldsymbol{z}_t \odot \widetilde{\boldsymbol{h}}_t.$$

Aktualizační brána $\boldsymbol{z}_t$ přímo interpoluje mezi starým stavem a kandidátem: $\boldsymbol{z}_t \approx 0$ znamená „nech vše, jak bylo“, $\boldsymbol{z}_t \approx 1$ „přepiš“. Resetovací brána $\boldsymbol{r}_t$ určuje, kolik minulosti smí kandidát použít.

Proti LSTM chybí oddělený stav buňky a výstupní brána. Zapomínání a zápis jsou svázané: co se přidá, to se musí ubrat.

Kterou zvolit

Empirická srovnání nedávají jednoznačnou odpověď. GRU je menší a rychlejší, LSTM má o něco větší kapacitu na dlouhé závislosti. Rozdíl bývá menší než vliv velikosti stavu a naladění kroku.

Kolik parametrů má rekurentní vrstva

Pro vstup dimenze d_x a stav d_h má jedna brána $d_h d_x + d_h^2 + d_h$ parametrů. Počet bran určuje zbytek:

buňka	bran	parametrů pro $d_x = 100$, $d_h = 256$
prostá RNN	1	91 392
GRU	3	274 176
LSTM	4	365 568

Rozhodující je člen d_h^2: zdvojnásobení stavu počet parametrů zhruba zčtyřnásobí, zatímco delší sekvence počet parametrů nemění vůbec — roste jen čas a paměť na uložené aktivace.

Paměť při trénování je úměrná $B \cdot T \cdot d_h$, protože BPTT potřebuje stavy ze všech kroků. Právě proto se používá zkrácené BPTT.

Dávkování sekvencí různé délky

Sekvence v dávce mají různou délku, tenzor však musí být obdélníkový. Kratší se proto doplní o výplňový token.

Výplň se nesmí započítat

Bez masky se výplňové kroky dostanou do ztráty i do posledního stavu. Model pak „umí“ předvídat výplň a metriky jsou nadhodnocené o podíl výplně v dávce.

Obvyklá opatření:

- maska délek ve ztrátě i při agregaci stavů,
- řazení podle délky do přibližně stejně dlouhých dávek, aby se výplň minimalizovala,
- u obousměrné sítě spouštět zpětný směr od skutečného konce, ne od konce výplně.

Je to častý zdroj tichých chyb: nic nespadne, jen výsledek nesedí.

Model encoder–decoder

Encoder vytvoří stavy $\boldsymbol{h}_1, \ldots, \boldsymbol{h}_n$; v základní variantě předá dekodéru pouze poslední stav. Dekodér pak autoregresně počítá

$$p(y_t \mid y_{<t}, \boldsymbol{x}_{1:n}) = \text{softmax}(W_o \boldsymbol{s}_t).$$

Při trénování se používá teacher forcing: dekodér dostává jako vstup skutečný předchozí token. Trénink je díky tomu rychlejší a stabilnější, jenže model se nikdy neučí pokračovat po vlastní chybě.

Příklad 3 (Pevný kontext jako úzké hrdlo)

Encoder zpracuje větu o 100 tokenech, ale dekodéru předá jediný vektor. Informace o prvním jménu i posledním čísle sdílejí tutéž kapacitu.

I. Sutskever, O. Vinyals, Q. V. Le: Sequence to Sequence Learning with Neural Networks. NIPS (2014).

Attention odstraní úzké hrdlo

Každý krok dekodéru dostane vlastní kontextový vektor:

$$\alpha_{t,i} = \mathrm{softmax}_i\big(a(\boldsymbol{s}_{t-1}, \boldsymbol{h}_i)\big), \qquad \boldsymbol{c}_t = \sum_i \alpha_{t,i}\, \boldsymbol{h}_i.$$

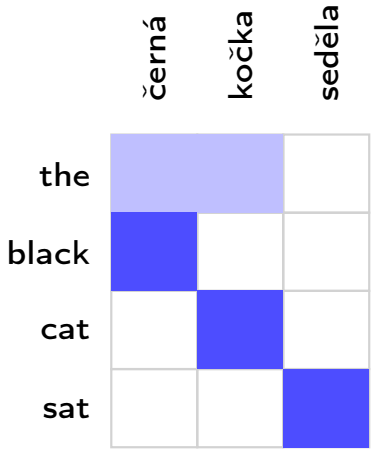
Váhy tvoří rozdělení přes vstupní pozice; kontext je vážený průměr, nikoli výběr jediné pozice.

- **Bahdanau:** aditivní skóre $\boldsymbol{v}_a^\top \tanh(W_s \boldsymbol{s} + W_h \boldsymbol{h} + \boldsymbol{b}_a)$, kontext se počítá před aktualizací dekodéru.
- **Luong:** skalární součin nebo bilineární skóre $\boldsymbol{s}^\top W \boldsymbol{h}$, kontext se kombinuje až se stavem aktualizovaným.

D. Bahdanau, K. Cho, Y. Bengio: Neural Machine Translation by Jointly Learning to Align and Translate. ICLR (2015). — M.-T. Luong, H. Pham, C. D. Manning: Effective Approaches to Attention-based NMT. EMNLP (2015).

Váhy attention jako zarovnání

Matice $\alpha_{t,i}$ říká, kolik váhy dal t-tý krok dekodéru i-té pozici vstupu.
U překladu z ní vystoupí zarovnání slov (řádky vstup, sloupce výstup):



Zarovnání se nikde nezadává — vzniká jako vedlejší produkt učení překladu.
Anglický člen nemá v češtině protějšek, a jeho váha se proto rozprostře.

Interpretace má meze

Vysoká váha znamená „odtud se braly hodnoty“, ne „tohle rozhodlo“.
Stejně opatrně jsme v sedmé přednášce četli heatmapy konvolučních sítí.

Dotazy, klíče a hodnoty

Moderní formulace odděluje tři role: stav dekodéru vytváří dotaz $\boldsymbol{q}$, každý stav encoderu klíč $\boldsymbol{k}_i$ a hodnotu $\boldsymbol{v}_i$.

$$\text{Attention}(Q, K, V) = \text{softmax}\left(\frac{QK^\top}{\sqrt{d_k}}\right) V.$$

Klíč slouží k výpočtu váhy, hodnota nese informaci do výstupu; dělení $\sqrt{d_k}$ stabilizuje měřítko skóre.

Tím se dostáváme na dohled Transformeru: stačí zahodit rekurenci a nechat sekvenci, aby věnovala pozornost sama sobě. To je téma příští přednášky.

Jak z modelu dostat výstup

Model dává rozdělení nad dalším tokenem; celou posloupnost je z něj potřeba teprve sestavit.

- **Hladové dekódování** bere v každém kroku argmax. Levné, ale jediná raná chyba se už nedá vzít zpět.
- **Beam search** udržuje B nejlepších částečných hypotéz a v každém kroku je rozšiřuje. Prohledává víc, stále však jen zlomek prostoru.

Delší výstup je vždy méně pravděpodobný

Logaritmy pravděpodobností se sčítají, takže součet s délkou klesá a beam search sám od sebe preferuje krátké věty. Skóre se proto dělí délkou, nebo $(5 + \ell)^\beta / 6^\beta$.

Větší B nezlepšuje výsledek neomezeně: od jisté hodnoty začne model vybírat obecné, bezpečné a málo informativní odpovědi.

Od one-hot k hustým vektorům

One-hot vektor umí slovo jednoznačně identifikovat, ale neříká nic o jeho významu: každá dvě různá slova jsou stejně vzdálená a vektor má rozměr celého slovníku.

Word2Vec se místo toho učí husté vektory z kontextů v korpusu. Slova používaná podobným způsobem získají podobné reprezentace.

Distribuční hypotéza

Slovo poznáme podle společnosti, v níž se vyskytuje. „Kočka“ a „pes“ sdílejí kontexty „krmí“, „spí“, „veterinář“. Stejně blízko ale skončí i „dobrý“ a „špatný“, protože se objevují v týchž větných pozicích; to je trvalá slabina distribučního přístupu.

T. Mikolov, K. Chen, G. Corrado, J. Dean: Efficient Estimation of Word Representations in Vector Space. arXiv:1301.3781 (2013).

CBOW a skip-gram

CBOW

Zprůměruje embeddingy okolních slov a predikuje prostřední token. Pořadí kontextu ignoruje, ale jeden příklad využije více sousedů; učení bývá rychlé.

Skip-gram

Z cílového tokenu predikuje jednotlivá slova v okolí. Jedna pozice vytvoří několik dvojic, což je dražší, ale vzácné slovo se dostane ke slovu vícekrát.

Odtud běžné doporučení: CBOW na velké korpusy, skip-gram na malé a na vzácnou slovní zásobu.

Příklad 4 (Dvojice z krátké věty)

Pro větu „malý pes rychle běží“ a okno $c = 1$ vzniknou mimo jiné dvojice (pes, malý), (pes, rychle), (rychle, pes) a (rychle, běží). Směr dvojice je důležitý: první člen vybírá vstupní embedding, druhý výstupní.

Proč se nepoužívá plný softmax

Každé slovo má dva vektory: u_w, řádek vstupní tabulky $U \in \mathbb{R}^{V \times D}$, když je cílovým slovem, a v_c, řádek výstupní tabulky $V_o \in \mathbb{R}^{V \times D}$, když stojí v kontextu. Logit dvojice je $s(w, c) = v_c^\top u_w$ a plný softmax vyžaduje

$$p(c \mid w) = \frac{\exp(v_c^\top u_w)}{\sum_{j=1}^{V} \exp(v_j^\top u_w)}.$$

Jmenovatel je zásadní překážkou: pro každou dvojici by bylo nutné spočítat skalární součin se všemi sto tisíci řádky výstupní matice.

Příklad 5 (Velikost tabulky)

Pro $V = 100\,000$ a $D = 300$ má jedna tabulka 30 milionů parametrů, tedy přibližně 120 MB ve float32, a skip-gram během trénování používá obě. Jediná tabulka by nestačila: skóre slova se sebou samým by bylo $\|u_w\|^2$, tedy vždy velké, ačkoli slovo vedle sebe sama skoro nestojí.

Negative sampling

Skutečnou dvojici odlišíme od K náhodných kontextů:

$$\log \sigma(v_c^\top u_w) + \sum_{k=1}^{K} \log \sigma(-v_{n_k}^\top u_w).$$

Původní úloha „které z V slov přijde“ se tím promění v $K+1$ otázek typu ano/ne.

Gradient podle vstupního embeddingu je

$$\nabla_{u_w} \ell = \big(\sigma(s^+) - 1\big) v_c + \sum_k \sigma(s_k^-) v_{n_k} :$$

pozitivní kontext u_w přitahuje, negativní odpuzuje silou $\sigma(s_k^-)$. Snadný negativ s velmi záporným skóre má téměř nulový gradient.

Negativní tokeny se nevzorkují rovnoměrně: používá se distribuce úměrná $f(w)^{3/4}$, která tlumí dominanci nejčastějších slov.

T. Mikolov, I. Sutskever, K. Chen, G. Corrado, J. Dean: Distributed Representations of Words and Phrases and their Compositionality. NIPS (2013).

Demonstrace: jeden krok SGNS

Dvourozměrné embeddingy, jeden pozitivní a jeden negativní kontext:

$$\boldsymbol{u}_w = (0{,}5;\ -0{,}2), \quad \boldsymbol{v}_c = (0{,}4;\ 0{,}1), \quad \boldsymbol{v}_n = (-0{,}3;\ 0{,}6), \quad \eta = 0{,}1.$$

	skóre	σ(skóre)
pozitivní $\boldsymbol{v}_c^\top \boldsymbol{u}_w$	0,180	0,545
negativní $\boldsymbol{v}_n^\top \boldsymbol{u}_w$	−0,270	0,433

Gradient podle $\boldsymbol{u}_w$ je

$$(0{,}545 - 1)\,\boldsymbol{v}_c + 0{,}433\,\boldsymbol{v}_n = (-0{,}312;\ 0{,}214),$$

takže po kroku $\boldsymbol{u}_w = (0{,}531;\ -0{,}221)$ a pozitivní skóre vzroste z 0,180 na 0,190.

Model tedy nedělá nic tajemného: přitáhne vektor ke skutečnému kontextu a odtlačí jej od náhodného.

Co Word2Vec ve skutečnosti počítá

Pointwise mutual information dvojice je

$$\mathrm{PMI}(w, c) = \log \frac{P(w, c)}{P(w)P(c)}.$$

Za idealizovaných předpokladů vede optimum skip-gramu s negative samplingem přibližně k

$$u_w^\top v_c \approx \mathrm{PMI}(w, c) - \log K.$$

Word2Vec tedy faktorizuje matici souvýskytů, kterou lingvisté počítali už dříve. To vysvětluje, proč z úlohy „hádej sousední slovo“ vznikne rozumná globální geometrie.

Rovnost platí jen v idealizovaném případě: mocnina 3/4, subsampling i konečná dimenze efektivní matici mění a PMI navíc není konečná pro dvojici, která se v korpusu nevyskytla — takových je většina.

Praktické detaily, na kterých záleží

- **Subsampling** zahazuje velmi častá slova s pravděpodobností rostoucí s četností. Učení zrychlí a zvýrazní obsahová slova; s předložkami ale mizí informace, kterou potřebují syntaktické úlohy.
- **Slovník** určuje minimální četnost tokenu. Velkorysý slovník zaplní paměť řádky pro překlepy, přehnaná normalizace smaže rozdíly, na kterých záleží.
- **Čeština** navíc trpí skloňováním: „stromu“, „stromem“ a „stromy“ jsou pro model tři různá slova, každé s třetinou dat.

Nejrychlejší kontrolou je prohlédnout si dvacet nejčastějších tokenů ve slovníku. Objeví-li se tam interpunkce nebo kusy URL, je chyba v tokenizaci a embeddingy nemá smysl trénovat.

Případová studie: malý český korpus

Zadání: 20 milionů tokenů zpravodajských textů, cílem jsou embeddingy pro klasifikaci krátkých článků.

volba	důvod
skip-gram	korpus je malý, vzácná slova potřebují víc dvojic
$D = 200$	větší dimenze se na 20 M tokenech nenaplní
$K = 10$ negativů	obvyklé rozmezí pro malé korpusy je 5–20
FastText	skloňování: tvary sdílejí znakové n-gramy
min. četnost 5	pod ní jsou převážně překlepy a vlastní jména

Kontrola po natrénování: dvacet nejčastějších tokenů, seznam sousedů pro deset ručně vybraných slov a histogram kosinových podobností náhodných dvojic.

Teprve pak má smysl měřit cílovou úlohu — a porovnat výsledek s veřejnými předtrénovanými vektory pro češtinu.

GloVe a FastText

GloVe

Vychází z matice souvýskytů a učí vektory tak, aby jejich skalární součin s biasy aproximoval $\log X_{ij}$. Využívá tedy agregovanou globální statistiku, zatímco skip-gram ji vzorkuje přes lokální dvojice.

FastText

Reprezentuje slovo součtem jeho vektoru a vektorů znakových n-gramů, takže umí sestavit i vektor slova, které v korpusu nebylo. Pro češtinu velká výhoda: tvary jednoho slova sdílejí podslova.

Podobný pravopis ovšem neznamená podobný význam: „kráva“ a „kravata“ sdílejí pět písmen, ale nic víc.

J. Pennington, R. Socher, C. D. Manning: GloVe. EMNLP (2014). — P. Bojanowski, E. Grave, A. Joulin, T. Mikolov: Enriching Word Vectors with Subword Information. *TACL* 5 (2017).

Jak embeddingy vyhodnotit

- **Kosinová podobnost** a seznam nejbližších sousedů: nejnázornější pohled dovnitř. Normalizace odstraní vliv délky vektoru, která bývá spíše odrazem četnosti než významu.
- **Analogie** (3CosAdd): vyjde-li, znamená to pouze, že daný vztah je v lokální geometrii přibližně lineární.
- **Extrinsická evaluace** vloží embeddingy do cílové úlohy. Intrinsické testy se počítají rychle, ale s výkonem v aplikaci korelují jen slabě.

Anizotropie a hubness

Mají-li vektory velkou společnou střední složku, je většina kosinových podobností kladná a několik tokenů se stává sousedem téměř všeho. Poznáme to z histogramu podobností náhodných dvojic.

Analogie a co se z nich dá číst

Úloha „*a* je k *b* jako *c* je k ?“ se řeší hledáním

$$\arg \max_{d \notin \{a,b,c\}} \cos\big(\boldsymbol{d},\ \boldsymbol{b} - \boldsymbol{a} + \boldsymbol{c}\big).$$

Tři důvody, proč výsledky vypadají lépe, než jsou

- Vstupní slova jsou z hledání *vyloučena*; bez toho by odpovědí často bylo samo *c*.
- Řada dvojic je řešitelná už tím, že se vezme nejbližší soused *c* — vektorová aritmetika k tomu není potřeba.
- Test měří jen lokální linearitu geometrie, ne „porozumění vztahu“.

Pro češtinu jsou navíc morfologické analogie („strom“ : „stromy“) snadné, zatímco sémantické trpí velikostí korpusu.

O. Levy, Y. Goldberg: Linguistic Regularities in Sparse and Explicit Word Representations. CoNLL (2014).

Od statických ke kontextovým reprezentacím

Word2Vec přiřadí tokenu „zámek“ jediný vektor bez ohledu na to, zda jde o budovu nebo mechanismus ve dveřích.

ELMo vytváří reprezentaci z obousměrného rekurentního jazykového modelu a mění ji podle věty; Transformer pokračuje stejným směrem pomocí self-attention.

Statické embeddingy tím ale nezastaraly: tabulka o stovkách megabajtů se načte okamžitě, vyhledání vektoru je jedno čtení z paměti a pro klasifikaci krátkých textů to postačuje.

Data mají datum

Embedding zachycuje, jak se v korpusu píše, včetně stereotypů a dobově podmíněných formulací. Vektory z roku 2013 odrážejí tehdejší úzus a novější pojmy vůbec neobsahují.

M. E. Peters a kol.: Deep Contextualized Word Representations. NAACL-HLT (2018).

Co všechno se do embeddingů dostane

Embedding je statistika korpusu, včetně toho, co bychom v něm raději neměli.

- **Stereotypy.** Profese se v textech pojí s rodem; vektorová geometrie tuto asociaci převezme a model postavený nad embeddingy ji přenese do svých rozhodnutí.
- **Soukromí.** Vzácná jména, adresy nebo identifikátory dostanou vlastní řádek. Model je sice nereprodukuje doslova, ale sousedé takového vektoru mohou prozradit kontext, ve kterém se vyskytoval.
- **Doména a datum.** Vektory z jednoho zdroje nemusí platit v jiném; nové pojmy v nich chybí úplně.

Zásahy do geometrie (odečtení „genderové osy“) asociaci zmenší v testu, ale ne nutně v úloze. Spolehlivější je hlídat výsledek cílového modelu než samotné vektory.

T. Bolukbasi a kol.: Man is to Computer Programmer as Woman is to Homemaker? NIPS (2016). — H. Gonen, Y. Goldberg: Lipstick on a Pig. NAACL (2019).