26. Neuronové sítě pro strukturovaná data (konvoluční a rekurentní sítě, (žádná haha) motivace, základní vlastnosti, použití).

Základy

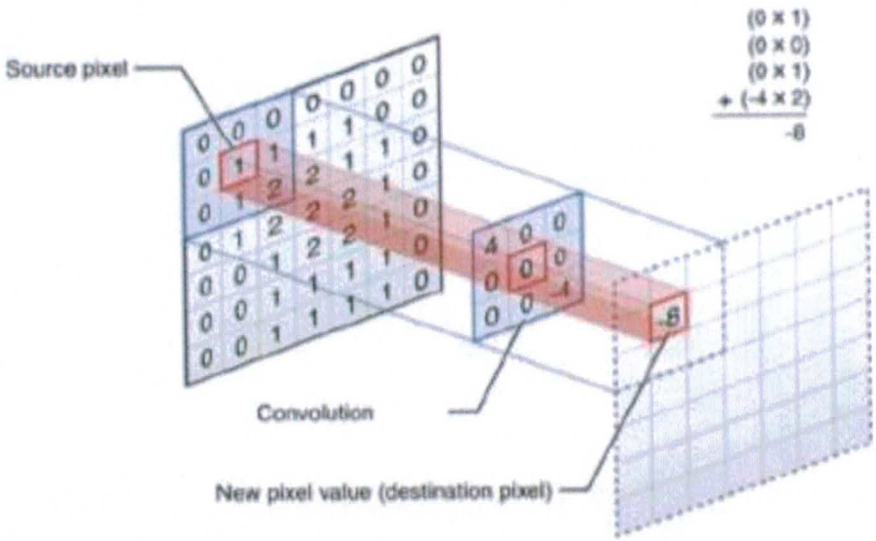
Vstupy a výstupy neuronových sítí jsou nutně float hodnoty kvůli kompatibilitě s optimalizačním algoritmem **gradientního sestupu** (otázka 25.). Přesto existuje mnoho implementací využívajících mnoho druhů různých strukturovaných dat. Musí se tedy pro použití sítí najít způsob jak vstupní/výstupní informace reprezentovat jako tyto hodnoty. Příklady některých vstupních dat a jejich kódování:

- Obrazová data: Stačí pouze převést RGB hodnoty z Int do Float. Často se provádí navíc normalizace z rozsahu 0-255 do -1.0 – 1.0. Nejčastěji v podobě WxHxC, kde W a H je šířka a výška obrazu a C je počet kanálů (RGB = 3)
- Textová data: Primitiva textu (tokeny - něco na úrovni slabik) se zakódují pomocí předtrénované kódující funkce. Vznikne matice o rozměrech LxE kde L je délka sekvence (textu) a E je rozměr kódování jednoho primitiva.
- Kategorická data - musí být předem znám počet kategorií; reprezentuje se většinou jako float pravděpodobnosti dané třídy a počet tříd je "hardcoded"

Konvoluční sítě

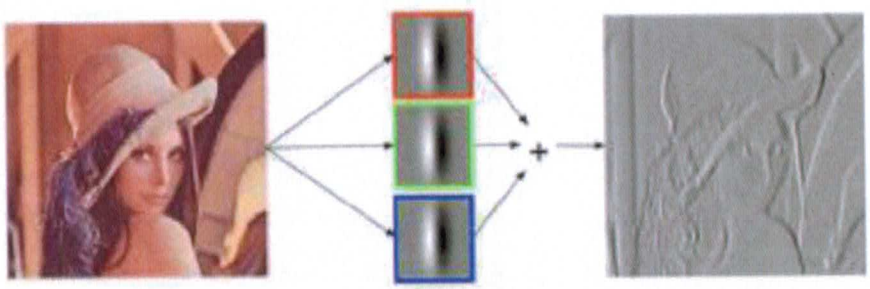
Konvoluční sítě jsou primárně používané převážně pro účely zpracování obrazu (mají ale širší využití - dají se použít i na zpracování přirozeného jazyka). Jedná se o relativně rychlou a efektivní implementaci, která analyzuje vstupní data pomocí lokálních operací. Jejich základem je konvoluční filtr. Princip konvolučního filtru spočívá v "ježdení" filtrem po obraze a vzájemným násobením jeho hodnot s hodnotami obrazu. Pro milovníky rovnic to znamená, že výstup filtru je vypočten následovně:

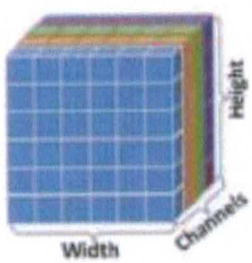
$y_{i,j}=\Sigma^{2}_{m=0}\ \Sigma^{2}_{n=0}\ (x_{i+(m-1)\ ,j+(n-1)}*k_{m,n})$ pro jádro filtru o velikosti 3x3 (dle obrázku); k je **konvoluční jádro** (kernel).



Konvoluční vrstva

Hodnoty jádra filtru, kterými se násobí pixely, jsou učitelné váhy, stejně jako tomu je u běžných vrstev neuronových sítí. Stejně jako u ostatních vrstev neuronové sítě, i tento koncept vyžaduje **aktivační funkci**. V případě RGB obrazu, kde vstupní data mají 3 kanály, má konvoluční jádro podobu NxNx3, kde výstupem operace konvoluce je beze změny pouze **jedna float hodnota**.

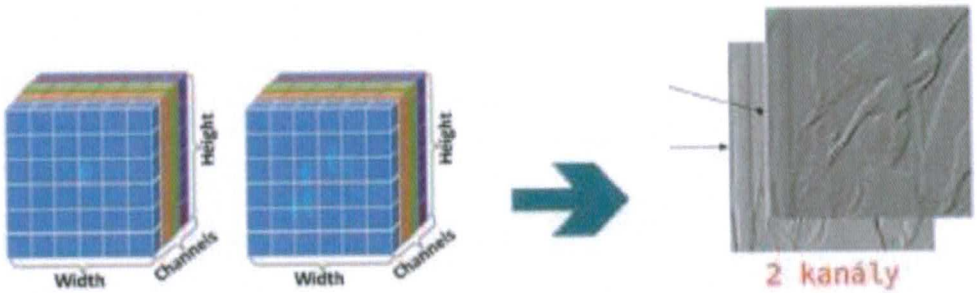




Obecně v konvolučních vrstvách bývá více jader, kde každé z nich generuje při aplikaci opět jediný float. Výstup této vrstvy má tedy počet kanálů shodný s počtem konvolučních jader. **Pozor, je potřeba si neplést počet vstupních vrstev s počtem jader, a tím pádem počtem výstupních kanálů - počet vstupních kanálů na podobu výstupu nemá vliv (viz obrázek).**

Jediné, co počet vstupních kanálů ovlivňuje, je velikost kernelu, a tím pádem počet parametrů sítě.

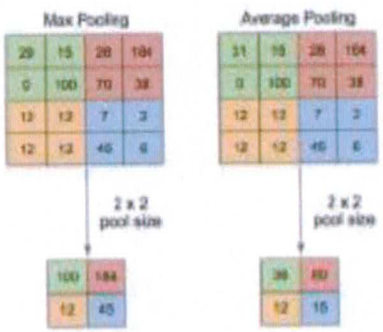
Jedna konvoluční vrstva o 2 jádrech přijímající vstup o 4 kanálech (např. obraz v RGBA – RGBA se normálně nepoužívá, jde jen o demonstrační příklad) bude vypadat takto:



Pooling vrstva

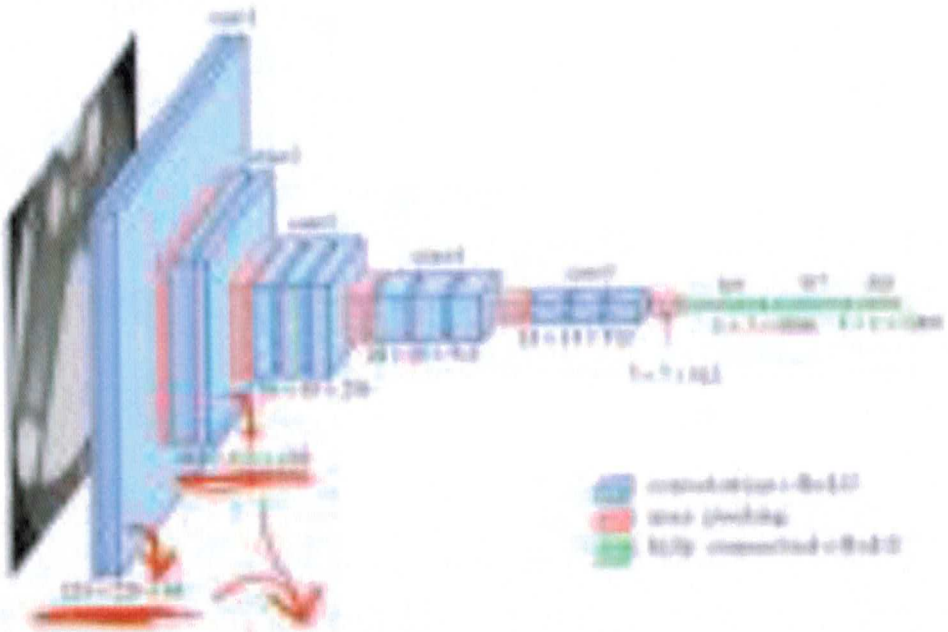
Mimo konvoluční vrstvy obsahují konvoluční sítě další prvek, a to jsou pooling vrstvy. Ty obecně v neuronových sítích používáme k redukci velikosti informace. **Sumarizují tedy nějakou širší oblast dat do kompaktnější informace.**

- Downsampling
  - Local pixels are aggregated into a single pixel
  - Spatial resolution is reduced, channels stay the same
  - Reduction by AVERAGE or MAX functions.
  - MAX-pooling directly encourages translation invariance



Příklad: podoba tensoru při dopředném průchodu

Příkladem neuronové sítě seskládané pomocí konvolučních vrstev a poolingu může být architektura VGG. Při průchodu takovou sítí dochází k redukci rozměrů vstupního "obrázku" *(po průchodu konvolučními vrstvami už to moc obrázky nejsou...)* a zároveň dochází k navýšení počtu kanálů. Na obrázku je možné vidět síť VGG, která se skládá z konvolučních vrstev, pooling vrstev a ReLU aktivací:



Je důležité umět určit, jak se budou skrze průchod neuronovou sítí měnit rozměry aktivací (informací). Uveďme si tedy příklad s neuronovou sítí z obrázku a vstupní fotografií o rozměru 224x224x3 (RGB). Tzn. formát zápisu bude **ŠÍŘKA x VÝŠKA x POČET_KANÁLŮ**.
Ukažme si tedy odůvodnění změny rozměrů skrze průchod:

1) Vstupní fotografie má rozměr **224x224x3** (fun fact: ta 3 nic neovlivní, může tam být cokoliv)
2) Jelikož je na obrázku uvedeno, že výstupní rozměr po průchodu prvními dvěma vrstvami má rozměr **224x224x64**, musí to tedy znamenat, že alespoň poslední z těchto dvou vrstev musí nutně mít **64 konvolučních jader**.
3) Z rozměrů výstupního **tensoru** (vektor/matice) následujících tří vrstev je zjevné, že **pooling má velikost okna 2x2** (224/2 = 112). Alespoň poslední z konvolučních vrstev (z této trojice) v této úrovni musí mít **128 jader**.
4) ...

Rekurentní sítě

Rekurentní sítě jsou velmi vhodné pro zpracování sekvenčních dat. Jejich výhodou je, že celá sekvence nemusí být dopředu známa díky iterativnímu průchodu skrze vstupní data. Sekvenční data můžeme chápat jako text, zvukové nahrávky, jiná měření v čase, apod. Stejně jako každé jiné neuronové sítě, i tyto přijímají pouze data v podobě floatových hodnot, a proto je potřeba provést jejich předzpracování.

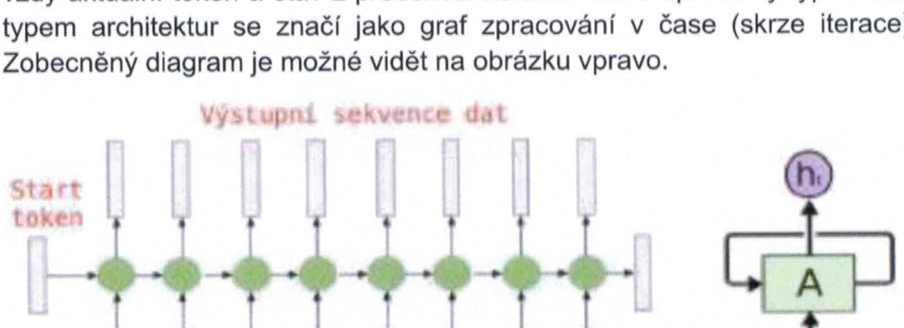
Tokeny

Předzpracování sekvenčních dat se provádí zakódováním textu do sekvence tokenů (vektorová reprezentace primitiv textu - nemusí jít nutně o slova). Mezi nejznámější způsoby kódování slov patří:

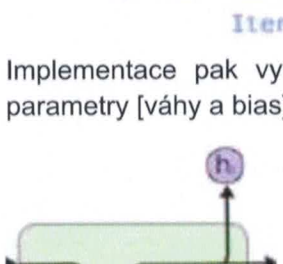
- **One-hot-encoding & LUT**
  - One-hot-encoding = N-dimenzionální vektor, kde každé slovo je reprezentováno jako 1 na daném místě vektoru; N je počet slov ve slovníku
  - LUT = tabulka vektorů, kde pronásobením s One-hot-embedding reprezentací získáme zakódovanou reprezentaci slova; LUT tabulka musí být předem vytvořena nějakou metodou
- **Word2Vec** = algoritmus využívající komplexnější princip; kouká se zároveň na významy slov a takto zakódované slova je možné porovnávat podle jejich významu (dokonce s nimi lze provádět operace typu: "král" + "žena" = "královna")
- *(Dnes se používají jiné složitější přístupy)*

Základní varianta rekurentní sítě

Jedná se o rekurentní ("iterativní") průchod jednou částí sítě token po tokenu, kde její vstup je vždy aktuální token a stav z předchozí iterace. Jde o specifický typ architektury. Průchod tímto typem architektur se značí jako graf zpracování v čase (skrze iterace) - vyznačeno vlevo. Zobecněný diagram je možné vidět na obrázku vpravo.



Implementace pak vypadá následovně (Tanh je aktivační funkce/nelinearita; W a b jsou parametry [váhy a bias]):



$$h_t = tanh(W[h_{t-1}; x_t] + b)$$

Výstup v každé iteraci jde použít i pro jiné účely než jen jako vstup pro následující iteraci, ale to záleží na konkrétní implementaci - někdy stačí jen výstup poslední iterace. Motivace za tímhle řešením je promítnutí informací z předchozích částí sekvence do zpracování ostatních částí. **Na rozdíl od konvolučních sítí, které pracují lokálně, se tato implementace snaží do jisté míry modelovat globálnější návaznosti dat**, které se obzvlášť v textu často objevují. Tato schopnost se ale s větší délkou sekvence ztrácí a přestává být efektivní - stejně jako u hlubokých

neuronových sítí bez využití residuálních spojení (vysvětleno v otázce 23. v sekci Residuální spojení).

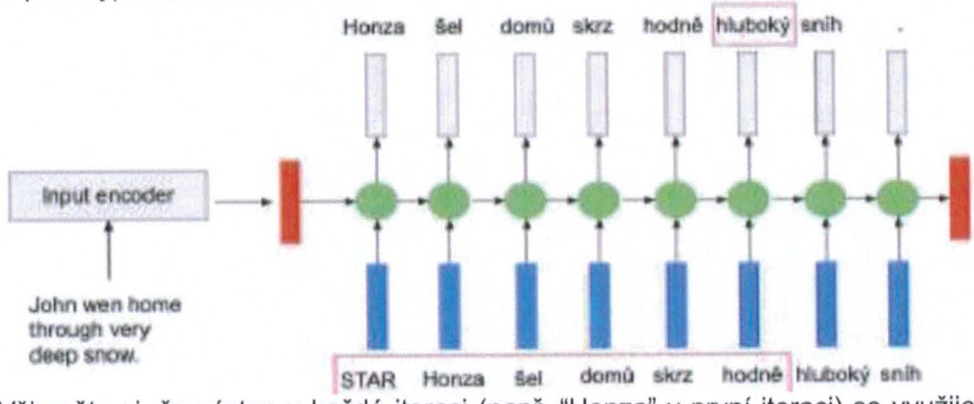
Autoregresivní faktorizace

(Strašidelný název, že?)
V případě kdy bychom chtěli trénovat klasickou rekurentní síť pro účely (například) překladu do jiného jazyka, narazili bychom na problém. Při překladu je potřeba generovat celou konzistentní větu - což nemusí odpovídat pouze 1:1 zobrazení (věta v cizím jazyce lze napsat vícero způsoby). To by bylo možné například tak, že vypíšeme všechny možné věty v daném jazyce a budeme trénovat klasifikační síť, která má na vstupu větu v původním jazyce a snaží se přiřadit vysokou pravděpodobnost větám v cílovém jazyce, které jsou správným překladem. Tohle je ale zcela neproveditelné vzhledem k počtu takových možností.

Existuje ale řešení: Rozložit pravděpodobnost celých vět na součin pravděpodobností všech slov, vždy podmíněných předchozími slovy ve větě.

$$P(w_1, w_2, w_3, \ldots, w_n) = \prod P(w_i | w_{i-1}, w_{i-2}, \ldots, w_1)$$

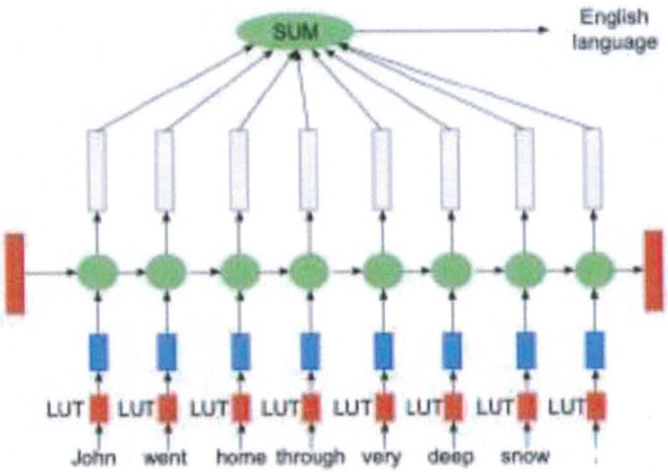
To se dá vyřešit přiřazením předchozích výstupů jako vstup v následující iteraci. Na diagramu by to poté vypadalo následovně:



Všimněte si, že výstup v každé iteraci (např. "Honza" v první iteraci) se využije jako vstup pro neuronovu síť v následující iteraci (druhý výskyt "Honza" v dolní části grafu).

Rekurentní sítě pro klasifikaci

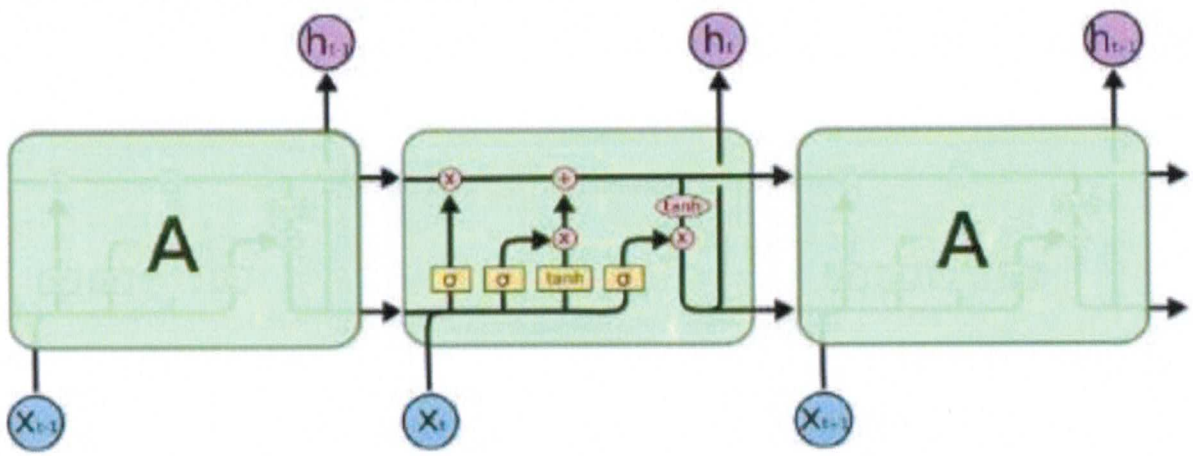
Pro úlohu klasifikace je intuitivní využít posledního (výstupního) stavu sítě pro učení klasifikace (v grafech průchodu v sítí v čase značeno jako poslední výstup vpravo). **Tento přístup ale není ideální, jelikož při velkém počtu iterací se informace průchodem sítí může téměř ztratit** (vysvětleno v otázce 23. V sekci Residuální spojení). Řešením může být klasifikace za pomocí agregace informací výstupů neuronové sítě skrze veškeré iterace, jak je naznačeno na obrázku:



Pozn. tuto klasifikační schopnost je potřeba stále trénovat spočtením loss funkce a provedením zpětné propagace nad tímto agregovaným výstupem. Zpětná propagace se provádí stejně jako u klasické sítě - graf na obrázku neobsahuje žádné cykly (podmínka výpočetního grafu), tudíž implementace je naprosto možná (viz. otázka 25).

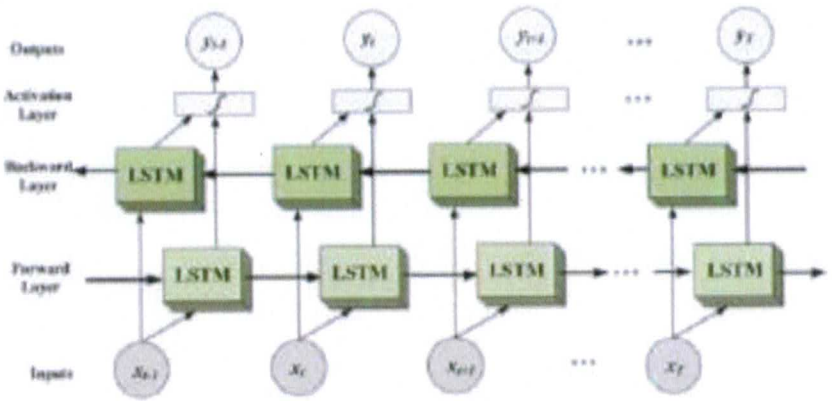
LSTM a GRU

Long Short Term Memory (a její zjednodušená varianta GRU) tento problém řeší obdobným způsobem jako residuální spoje v konvolučních sítích. Původní informaci z předchozí iterace pouze modifikují a nenahrazují ji novou informací, jako tomu je u klasických rekurentních sítí. Implementace umožňuje zapomínání informací z předchozí iterace (forget gate). Schéma oproti klasickým rekurentním sítím vypadá následovně:



Bidirectional recurrent layer

Jde o prakticky o dvě klasické rekurentní vrstvy, kde jedna zpracovává sekvenci v pořadí a druhá v opačném *(protože proč ne...)*. Výstupy obou těchto vrstev se konkatenují (výstupní vektor každé iterace je 2x delší). Dá se využít pouze v případech, kdy je možné se dívat "do budoucna" - konec sekvence musí být znám předem jinak by průchod opačným směrem nefungoval.

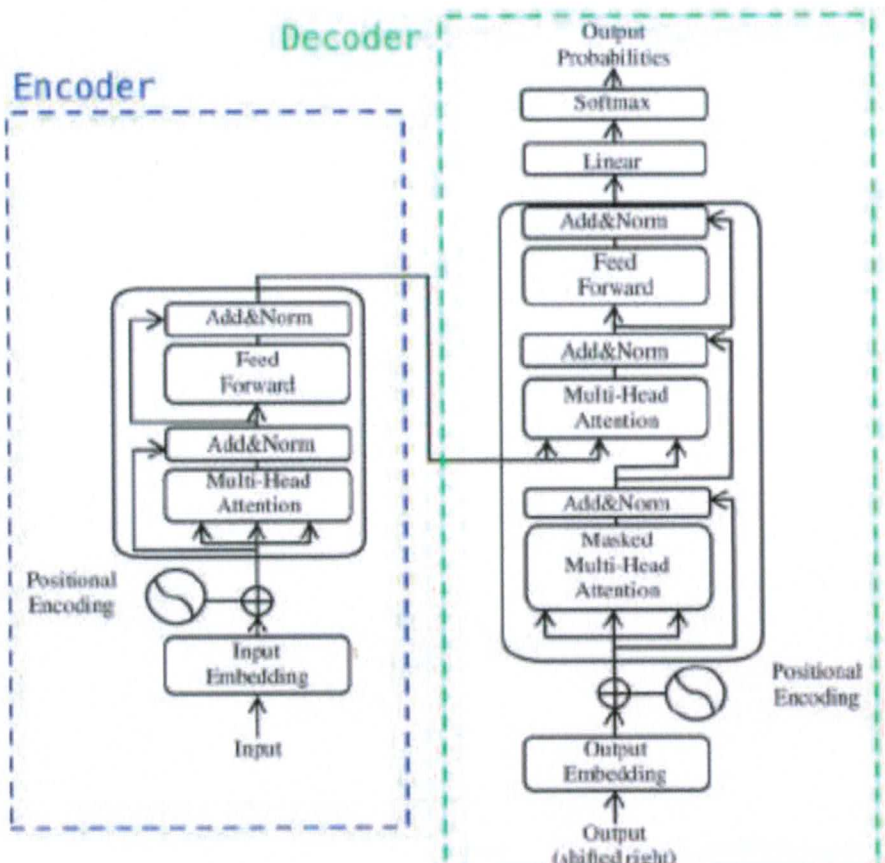


Transformery (otázka jako sebrová attention a transformery rozvinuje)

Transformery řeší problémy s globálními závislostmi next level způsobem oproti rekurentním sítím. Zde se vyloženě počítají souvislosti každého tokenu sekvence s každým dalším tokenem. Mají ale oproti rekurentním sítím jednu nevýhodu, **vstupní data musí být dopředu známá**. To ale znamená, že oproti rekurentním sítím, **vstupní data musí mít přesně daný rozměr**. Základní princip spočívá v attention vrstvách. Existují adaptace transformerů i pro obrazová data - Vision Transformery.

Transformery mají obecně dvě části, a to encoder a decoder. Encoder slouží k zakódování informace a decoder se používá k transformaci této informace dle požadované úlohy. V některých implementacích stačí jen encoder (např. když chceme biometrický údaj transformovat na unikátní klíč, ...). Celý transformer se učí tak, že anotace **(ground truth)** se "strkají" přímo do dekodéru v místě s označením "Output (shifted right)".

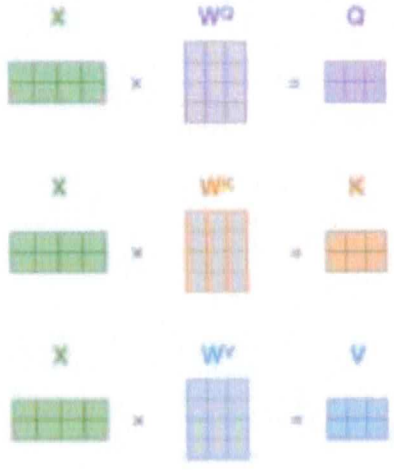
8/11



Attention

Základní stavební blok transformerů attention funguje poměrně jednoduchým způsobem. Máme 3 "učící se" matice (pojmenované W^Q, W^K a W^V), kterými násobíme vstupní data klasickým maticovým násobením.

9/11



Po vynásobení získáme matice Q, K a V, které dosadíme do vzorce:

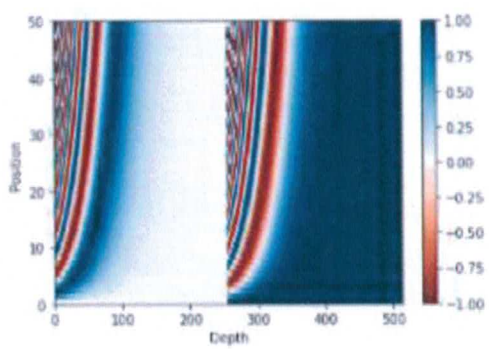
$$\text{softmax}\left(\frac{Q \times K^T}{\sqrt{d_k}}\right) V$$

d_k je normalizace dle velikosti matice K a často se z vzorců Attention vrstvy vypouští. *Význam těchto matic je mimo rozsah probíraného tématu.*

Výsledkem této rovnice je výstup attention vrstvy. V transformerech se můžeme setkat v variantou multihead self-attention, což je akorát obdoba využití vícero konvolučních jader v konvolučních vrstvách.

Poziční kódování

Jelikož princip implementace transformerů pomíjí informaci o pozici tokenu v sekvenci, je potřeba tyto informace dodat explicitně pozičním kódováním. Jde prakticky o přičtení nějaké hodnoty k float reprezentaci každého tokenu. Používají se k tomu kombinace sinových a kosinových funkcí - je to pro síť "příjemnější" než přičítání indexů. Jak vypadá taková kombinace sinových a kosinových funkcí je vidět na obrázku:



Výběr známých úloh řešených neuronovými sítěmi

(Rozdělení do kategorií jsem provedl já, nemá smysl se je učit.)

Generativní úlohy

- GPT-like chatboti (transformery)
- generování obrázků dle textového popisu (generativní sítě)
- text2speech (generativní sítě)

Klasifikační úlohy

- klasifikace zvířat/rostlin/hub (konvoluční sítě)
- obecná klasifikace známých objektů v obraze (konvoluční sítě a vision transformery)
- klasifikace vlastností objektů v obraze (konvoluční sítě)
- klasifikace "emocí" z textu (transformery)
- klasifikace druhu díla z textu (transformery)

Predikce

- optical flow - odhad pohybu objektů v obrázku (konvoluční sítě)
- predikce akcí agenta dle pozorovaného prostoru (ne vždy se řeší neuronkami a když už, tak to budou nejspíš konvoluční sítě)

Rozpoznávání

- speech2text (transformery a pokud jde o real-time, tak jsou možné i rekurentní sítě)
- rozpoznávání větných členů v textu (transformery nebo rekurentní sítě)
- rozpoznávání známých objektů v obraze (konvoluční sítě)
- detekce vad výrobků v obraze (bezkonkurenčně konvoluční sítě)
- rozpoznávání obličejů v obraze (konvoluční sítě a možná i vision transformery)
- pose estimation (konvoluční sítě)

Pokud v textu najdete chybu, nebudete něčemu rozumět nebo budete mít dojem, že by bylo vhodné něco doplnit, kontaktujte na discordu uživatele no.body.the.sad.slider.boy.