EVROPSKÁ UNIE
Evropské strukturální a investiční fondy
Operační program Výzkum, vývoj a vzdělávání

MINISTERSTVO ŠKOLSTVÍ, MLÁDEŽE A TĚLOVÝCHOVY

Název projektu	Rozvoj vzdělávání na Slezské univerzitě v Opavě
Registrační číslo projektu	CZ.02.2.69/0.0./0.0/16_015/0002400

Velké jazykové modely v kontextu Umělé Inteligence

Distanční studijní text

Jména autorů bez titulů oddělit čárkou

Karviná 2017

SLEZSKÁ UNIVERZITA
OBCHODNĚ PODNIKATELSKÁ FAKULTA V KARVINÉ

Obor:	Vypište obor(-y), do kterého tematicky spadá studijní text. Vycházet lze z klasifikace oborů vzdělání například: CZ-ISCED-F 2013 nebo ze seznamu organizace Library of Congress Classification Outline. Spadá-li opora do několika oborů, pak je vyjmenujte a oddělte čárkou, např. Ekonomie, marketing, psychologie.
Klíčová slova:	Může zde být výběr z klíčových slov kapitol. Oddělit čárkami.
Anotace:	Stručná anotace studijní opory (1 až 2 odstavce)

Autor: **Jména autorů včetně titulů**
každé na nový řádek

Obsah

ÚVODEM

Úvodní slovo autora (-ů). Několik odstavců, kde je např. vysvětleno:

- k čemu a pro koho je studijní opora určena
- jaké jsou minimální předchozí požadavky, co před tím musí student umět atd.
- jak může s textem student pracovat
- **bližší popis využití použitých distančních prvků v textu – povinné!!!**

Distanční studijní text předpokládá existenci LMS kurzu v němž je možno realizovat aktivity vyžadující komunikaci nebo interakci se studentem nebo mezi studenty navzájem. V LMS kurzu též je prostor uvést nutné kontaktní informace (sylabus, konzultační hodiny atp.), informace o způsobech komunikace se studenty a konkrétních požadavcích úspěšného absolvování kurzu.

Text by neměl přesáhnout tuto stránku!

RYCHLÝ NÁHLED STUDIJNÍ OPORY

Stručné shrnutí toho, co je náplní následujících stránek. Stačí opět několik odstavců, aby si čtenáři mohli udělat představu, co se na následujících stranách mohou dozvědět. Pro snadnější orientaci studenta je možné strukturovat text náhledu dle obsahu. **Obsah musí korespondovat s akreditací daného kurzu**.

Text by neměl přesáhnout tuto stránku!

Povinná část.

1 VELKÉ JAZYKOVÉ MODELY V KONTEXTU UMĚLÉ INTELIGENCE

RYCHLÝ NÁHLED KAPITOLY

V této kapitole se seznámíme s tzv. velkými jazykovými modely, mezi něž patří např. ChatGPT vyvinutý firmou OpenAI nebo Gemini vyvinutý AI laboratoří Google DeepMind. Seznámíme se také s tím, jak tyto jazykové modely souvisí s oborem umělá inteligence, a podíváme se na to, jak vypadá a jak pracuje Transformer, což je revoluční technologie, která umožnila dosud nevídanou kvalitu výstupů zmíněných modelů.

CÍLE KAPITOLY

Po prostudování této kapitoly budete vědět:

- Co je to jazykový model.
- Co je to umělá inteligence.
- Jakou roli hrají jazykové modely v umělé inteligenci.
- Co je to embedding slov.
- Co je to mechanismus sebe-pozornosti.
- O předchůdcích Transformerů.
- Jak Transformer generuje další slovo na základě daného kontextu.
- Jak se Transformer učí.

KLÍČOVÁ SLOVA KAPITOLY

jazykový model, umělá inteligence, Transformer, sebe-pozornost, embedding, kodér, dekodér

V posledních letech se umělá inteligence (AI) stala jedním z nejvýznamnějších a nejrychleji se rozvíjejících oborů v oblasti informatiky, přičemž velké jazykové modely hrají klíčovou roli v této revoluci. Tyto modely, které dokáží generovat, rozumět a interpretovat lidský jazyk s dosud nevídanou přesností a sofistikovaností, otevírají nové možnosti v mnoha odvětvích - od automatického překladu přes tvorbu obsahu až po programování v různých programovacích jazycích na základě pokynů v přirozeném jazyce (Górecki (2023)).

Cílem této kapitoly je seznámit studenty předmětu Inovace v ICT s tím, jak velké jazykové modely formují současný a budoucí vývoj v oblasti umělé inteligence. Budeme se zde zabývat nejen samotnými jazykovými modely a jejich vývojem, ale také širším kontextem AI, kde tyto modely nacházejí uplatnění.

Kapitola začíná definicí jazykových modelů a umělé inteligence, poskytuje přehled o tom, co představují velké jazykové modely v rámci AI, a dále se zabývá technologickým pokrokem, který vedl od prvních pokusů o modelování jazyka k sofistikovaným systémům založeným na architektuře Transformer.

KONTROLNÍ OTÁZKA

1. Co dokáží velké jazykové modely s dosud nevídanou přesností?

1.1 Jazykové modely

V této kapitole se zaměříme na jeden ze základních stavebních kamenů moderní umělé inteligence - jazykový model. Jazykové modely jsou typem umělé inteligence navržených k interpretaci, generování a porozumění lidskému jazyku. Tyto modely se staly základem pro řadu aplikací, od automatických překladačů až po virtuální asistenty, a hrají klíčovou roli v rozvoji počítačového zpracování přirozeného jazyka (NLP - z angl. Natural Language Processing).

1.1.1 Definice Jazykového Modelu

Jazykový model je algoritmus, který se snaží odhadnout pravděpodobnost následujícího slova nebo sekvence slov v daném jazyce na základě předchozích slov. Tato schopnost modelu předvídat další slovo umožňuje generování koherentního a kontextově relevantního textu. V nejjednodušším tvaru může být jazykový model použit k dokončení věty, zatímco v pokročilejších aplikacích může generovat celé odstavce textu, které jsou syntakticky správné a často i sémanticky smysluplné.

1.1.2 Funkce Jazykových Modelů

Jazykové modely se učí z obrovského množství textových dat. Prostřednictvím analýzy těchto dat se modely učí jazykové struktury, slovní zásobu, gramatiku a dokonce i styl psaní specifický pro daný jazyk nebo kontext. Tento proces učení se umožňuje modelům pochopit a generovat jazyk na základě předchozích vzorců a pravidel obsažených v trénovacích datech.

1.1.3 Typy Jazykových Modelů

Jazykové modely lze rozdělit do dvou hlavních kategorií: statistické a neuronové. Statistické jazykové modely využívají tradiční statistické metody k odhadu pravděpodobnosti slov a sekvencí na základě jejich výskytu v trénovacích datech. Na druhé straně, neurální jazykové modely, které jsou založeny na umělých neuronových sítích, poskytují větší flexibilitu a schopnost učit se složitější vzorce jazyka díky své hlubší a komplexnější struktuře.

1.1.4 Základní Principy

Základním principem, na kterém jazykové modely fungují, je učení se z kontextu. Modely jsou trénovány na rozsáhlých korpusech textu, kde se učí rozpoznávat vzorce, vztahy mezi slovy a jak tyto vztahy ovlivňují význam věty. Tímto způsobem se modely stávají schopnými predikovat, jaké slovo nejpravděpodobněji přijde dále v textu, a to na základě předchozího kontextu.

Představme si například statistický jazykový model, který byl natrénován na velkém korpusu textů v českém jazyce. Tento model se naučil pravděpodobnosti výskytu různých slov a slovních sekvencí na základě jejich frekvence v trénovacích datech. Když model dostane za úkol dokončit větu "Dnes je krásné...", analyzuje slova "Dnes je krásné" a na základě statistických pravděpodobností vypočítá, která slova jsou nejpravděpodobnější následující. Pokud v trénovacích datech často následovalo slovo "počasí" po frázi "Dnes je krásné", model s vysokou pravděpodobností generuje větu "Dnes je krásné počasí."

Uvažujme, že v našich trénovacích datech jsou dvě věty "Dnes je krásné počasí" a jedna věta "Dnes je krásné ráno". Pokud chceme spočítat pravděpodobnost následujícího slova po frázi "Dnes je krásné", využijeme jednoduché statistické metody založené na frekvenci výskytu. Pravděpodobnost výběru slova "počasí" po frázi "Dnes je krásné" se spočítá jako počet výskytů fráze "Dnes je krásné počasí" dělený celkovým počtem výskytů fráze "Dnes je krásné" následované jakýmkoli slovem. V našem případě:

- "Dnes je krásné počasí" se objevuje 2krát.
- "Dnes je krásné ráno" se objevuje 1krát.

Celkový počet výskytů "Dnes je krásné" následované jakýmkoli slovem je 3 (2 pro "počasí" + 1 pro "ráno"). Tedy pravděpodobnost, že po "Dnes je krásné" následuje slovo "počasí", je 2/3, zatímco pravděpodobnost, že následuje slovo "ráno", je 1/3.

Na tomto příkladu si také můžeme vysvětlit další dva základními koncepty ve NLP. *N-gram* je sekvence *n* po sobě jdoucích slov v textu, což modelu umožňuje zachytit kontext a závislosti mezi slovy. Například ve větě "Dnes je krásné počasí", 2-gramy (bigramy) by byly ["Dnes je", "je krásné", "krásné počasí"]. *Tokenizace* je pak proces rozdělení textu na jednotky (tokeny),

které mohou být slova, fráze nebo i jednotlivé znaky. Tokenizace věty "Dnes je krásné počasí", tak jak by ji provedly modely GPT-3.5 nebo GPT-4, je[1]

["D", "nes", " je", " kr", "ás", "**né**", " po", "č", "as", "í"],

tedy tento text o 21 znacích (včetně mezer) by se rozložil na 10 tokenů. Každý token má pak v modelu unikátní ID, které daný token reprezentuje. V našem případě by tato sekvence tokenů byla převedena na tato ID:

[35, 4978, 4864, 23975, 7206, **52235**, 3273, 13453, 300, 2483].

Jak lze vidět z hodnoty ID pro token "né" (tučně), seznam všech tokenů, se kterými umí model pracovat, je pro tyto modely větší než 50 000, což modelům umožňuje pracovat s většinou textů dostupných na Internetu a v různých jazycích.

Výše uvedený příklad demonstruje základní princip statistických jazykových modelů: odhad pravděpodobnosti následujícího slova na základě statistické analýzy výskytu slov v textových datech. Tímto způsobem může model generovat text, který je syntakticky správný a často i sémanticky smysluplný, což je základem pro mnoho aplikací v oblasti zpracování přirozeného jazyka.

KONTROLNÍ OTÁZKA

1. Co je jazykový model a jaké má aplikace?
2. Jaký je rozdíl mezi statistickými a neuronovými jazykovými modely?
3. Jak se spočítá pravděpodobnost následujícího slova v statistickém jazykovém modelu?

1.2 Umělá Inteligence

Umělá inteligence (AI) je odvětví informatiky, které se zabývá vytvářením inteligentních strojů schopných vykonávat úkoly vyžadující lidskou inteligenci. Tyto úkoly zahrnují učení, rozhodování, rozpoznávání obrazu, zpracování přirozeného jazyka a mnoho dalších. AI simuluje lidskou inteligenci kombinací informatiky, kognitivní psychologie a strojového učení, a to s cílem vytvořit systémy, které mohou učinit autonomní rozhodnutí.

1.2.1 Historie a Vývoj

Umělá inteligence má bohatou historii, která sahá až do poloviny 20. století, kdy výzkumníci začali experimentovat s prvními programy schopnými simulovat aspekty lidského myšlení a

[1] https://platform.openai.com/tokenizer

učení. Od těch dob se AI vyvíjela exponenciálním tempem, přičemž nedávné průlomy v oblasti strojového učení a neuronových sítí, jako např. učení posilováním (reinforcement learning), kde velký úspěch zaznamenal např. systém AlphaGo z AI laboratoře DeepMind (Silver et al. (2017)), otevřely nové možnosti pro aplikace AI v praxi.

1.2.2 KATEGORIE UMĚLÉ INTELIGENCE

Umělá inteligence se obvykle dělí do dvou hlavních kategorií: úzká AI a obecná AI. Úzká AI (Narrow AI), také známá jako slabá AI, se odkazuje na systémy navržené a vytrénované k vykonávání specifických úkolů bez vědomí mimo jejich definovaný účel. Tuto kategorii zastupuje např. výše zmíněný AlphaGo, který sice je schopen hrát Go již na nadlidské úrovní, nicméně žádnou další úlohu nebo hru řešit nebo hrát neumí. Naproti tomu obecná AI (AGI – Artificial General Intelligence), často označovaná jako silná AI, se vztahuje na teoretické systémy schopné pochopit, učit se a aplikovat inteligenci přes široké spektrum úkolů s lidskou úrovní adaptability. Ačkoli v době psaní tohoto textu ještě žádný systém, který lze AGI označit, neexistoval, vytvoření AGI je hlavním cílem tvůrců systémů jako ChatGPT nebo Gemini. Takový autonomní systém by poté překonával člověka ve vykonávání většiny ekonomicky hodnotných úloh, a tedy by mohl mít hluboký dopad na ekonomiku, pracovní trh a celkovou strukturu společnosti. Vytvoření AGI by znamenalo, že by stroje nejenže převzaly rutinní a manuální úkoly, ale také by měly schopnost vykonávat složité intelektuální úkoly, které dnes vyžadují lidskou expertizu a kreativní myšlení. To by mohlo vést k významnému nárůstu produktivity a efektivity ve všech odvětvích, od medicíny a vědy přes inženýrství až po umění.

Na jedné straně by to mohlo přinést obrovské výhody, jako je zlepšení zdravotní péče díky personalizovaným léčebným plánům generovaným AI, rychlejší výzkum a vývoj v oblasti nových technologií a léků, a možnost řešit složité globální problémy, jako je změna klimatu, s dosud nevídanou účinností.

Na druhé straně by však rozšíření AGI mohlo představovat výzvy, jako je ztráta pracovních míst v důsledku automatizace, etické dilema ohledně rozhodovacího procesu AI a potenciální bezpečnostní rizika, pokud by tyto systémy byly zneužity nebo pokud by se jejich chování stalo nepředvídatelným. Zvládnutí těchto výzev by vyžadovalo pečlivé zvážení a implementaci politik a regulací, které by zajistily, že vývoj a nasazení AGI by prospívalo lidstvu jako celku a minimalizovalo potenciální negativní dopady.

KONTROLNÍ OTÁZKA

1. Co je umělá inteligence a jaké úkoly dokáže vykonávat?
2. Jaký je rozdíl mezi úzkou AI a obecnou AI?
3. Jaké přínosy a problémy by představovalo vytvoření AGI?

1.3 Předchůdci Transformerů

V této kapitole se zaměříme na technologie a koncepty, které předcházely vývoji v současnosti nejvýkonnějších modelů, jako je GPT (Generative Pre-trained Transformer). Tito předchůdci položili základy pro pokrok v oblasti velkých jazykových modelů a umožnili vývoj vysoce výkonných současných modelů.

1.3.1 Statistické Modely

Jak již bylo zmíněno výše, první jazykové modely byly založeny na statistických metodách. Tyto modely využívaly jednoduché statistické techniky k odhadu pravděpodobnosti slov a frází v textu na základě jejich výskytu v trénovacích datech. Přestože byly tyto modely omezené ve své schopnosti zachytit složitější jazykové struktury a závislosti, položily důležité základy pro pochopení a modelování jazyka.

1.3.2 Ranné Neuronové Sítě

Dalším krokem ve vývoji bylo využití raných neuronových sítí pro modelování jazyka. Tyto modely začaly využívat schopností neuronových sítí učit se a adaptovat se na složitější vzorce v datech. Jednou z klíčových inovací bylo zavedení rekurentních neuronových sítí (RNN) a jejich variant, jako jsou LSTM (Long Short-Term Memory) a GRU (Gated Recurrent Units), které byly schopné lépe zachytit časové závislosti v textu.

1.3.3 Problémy s Rannými Modely

Problémy s ranými modely, zejména s RNN a jejich pokročilými variantami jako LSTM a GRU, jsou zásadní pro pochopení vývoje v oblasti zpracování přirozeného jazyka a motivace pro vznik architektury Transformer. Tyto výzvy jsou spojeny se zásadními omezeními v architektuře a fungování těchto modelů:

1.3.3.1 Gradientní Mizení a Explodující Gradienty

- **Gradientní Mizení (Vanishing gradients)**: Jde o problém, při kterém gradient chybové funkce (podstatný pro aktualizaci vah během učení modelu na daných datech) exponenciálně klesá s každým dalším krokem zpětné propagace (backpropagation), což způsobuje, že se váhy neuronů ve spodních vrstvách sítě aktualizují velmi málo nebo vůbec. To znamená, že model přestává být schopným učit se dlouhodobým závislostem ve větách, protože informace potřebné pro učení se ztrácí, jakmile se pohybujeme v dlouhé větě zpět.
- **Explodující gradienty (Exploding gradients)**: Na druhé straně, když gradient chybové funkce roste exponenciálně, může to vést k velmi velkým změnám ve váhách, což způsobuje nestabilitu modelu. To může vést k tomu, že model přestane rozumně konvergovat k minimální chybě, protože váhy se aktualizují příliš agresivně.

Tyto problémy jsou obzvláště výrazné v úlohách, které zahrnují dlouhé sekvence dat, jako je generování textu, kde je důležité zachytit závislosti mezi slovy, která jsou od sebe vzdálená.

1.3.3.2 Náročnost na Výpočetní Zdroje

RNN a jejich varianty vyžadují sekvenční zpracování dat, což znamená, že každý prvek sekvence musí být zpracován jeden po druhém. Toto omezení značně zvyšuje výpočetní náročnost a čas potřebný pro trénování modelů, zejména při práci s dlouhými textovými sekvencemi. Navíc tento sekvenční přístup znemožňuje efektivní paralelizaci, což je klíčové pro rychlé trénování modelů na moderních hardwarových platformách.

1.3.3.3 Obtížnost Modelování Složitějších Závislostí

Ačkoli LSTM a GRU byly navrženy tak, aby lépe zachytávaly dlouhodobé závislosti než základní RNN, stále čelily omezením ve své schopnosti modelovat velmi složité jazykové struktury a nuance. To je dáno tím, že jejich schopnost zachytit kontext a závislosti je stále omezena kapacitou jejich architektury, což může vést k nedostatečnému výkonu v některých úlohách NLP.

1.3.3.4 Motivace pro Transformer

Tyto výzvy vedly k hledání nových přístupů v NLP, což nakonec vedlo k vývoji architektury Transformer. Transformer překonává mnohé z těchto omezení díky svému jedinečnému využití mechanismů sebe-pozornosti, které umožňují modelu efektivně zpracovávat a interpretovat dlouhé sekvence dat paralelně, čímž se zvyšuje rychlost trénování a zlepšuje schopnost modelu zachytit složité závislosti v datech.

KONTROLNÍ OTÁZKA

1. Co jsou to gradientní mizení a explodující gradienty a jak ovlivňují trénování RNN a jejich variant?
2. Proč je sekvenční zpracování dat v RNN a jejich variantách náročné na výpočetní zdroje?
3. Jak architektura Transformer překonává omezení raných sekvenčních modelů?

1.4 Transformer

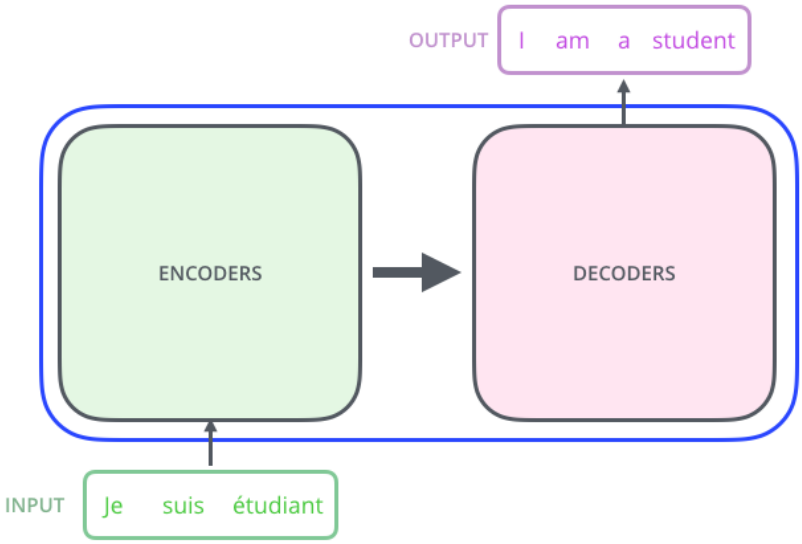
Transformer je model hlubokých neuronových sítí založený na mechanismu sebe-pozornosti (self-attention), navržený v roce 2017 v článku "Attention Is All You Need" (Vaswani (2017)). Neobsahuje zpětnou vazbu jako modely RNN a tím pádem vyžaduje méně času na trénování. To umožnilo vznik tzv. velkých jazykových modelů, které jsou trénovány na obrovských textových korpusech jako Wikipedie nebo Common Crawl. Vstupní text je rozdělen na n-gramy zakódované jako tokeny a každý token pak je převeden na jeho vektorovou reprezentaci pomocí

embeddingu. V každé vrstvě Transformeru je poté každý token tzv. kontextualizován v rámci kontextového okna s dalšími tokeny prostřednictvím paralelního mechanismu vícečetné sebe-pozornosti, což umožňuje zesílení signálu pro klíčové tokeny a oslabení méně důležitých tokenů. Článek o Transformeru, publikovaný v roce 2017, je založen na mechanismu sebe-pozornosti založeném na funkci Softmax, navrženém v Bahdanau a kol. (2014) pro strojový překlad, a Fast Weight Controller, podobný Transformeru, navrženém v Schmidhuber (1992). Vzhledem k zásadní roli, kterou tento model hraje v NLP a AI, si ho nyní více rozebereme[2].

1.4.1 TRANSFORMER Z NADHLEDU

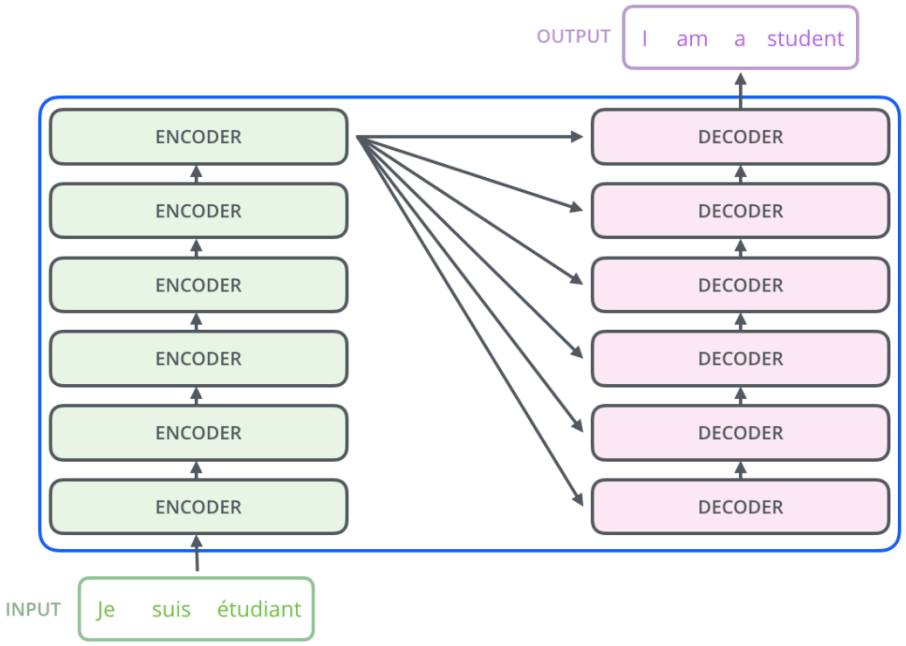
Začněme tím, že se na model podíváme jako na jednu černou skříňku. V aplikaci strojového překladu by model vzal větu v jednom jazyce a vygeneroval její překlad v jiném jazyce, například z francouzštiny do angličtiny. Vstupní sekvence by tedy byla "Je suis étudiant" a výstupní pak "I am a student."

Když se podíváme dovnitř této skříňky, uvidíme kódovací část (encoders), dekódovací část (decoders) a spojení mezi nimi.

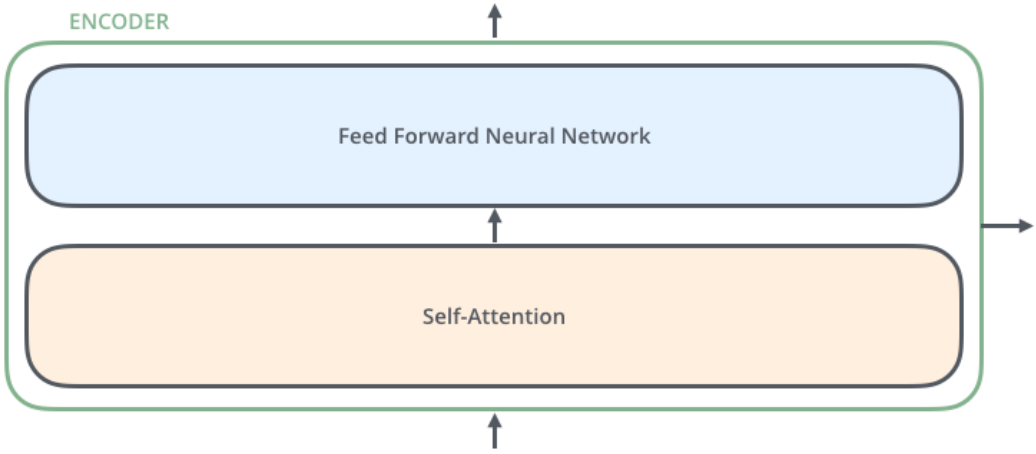


[2] Následující text vychází z Alammar (2018), ze kterého jsou převzaty i některé ilustrace.

Kódovací část je sada kodérů (encoder) (na obrázku níže jich je např. šest na sobě). Dekódovací část je pak sada dekodérů (decoder) o stejném počtu.



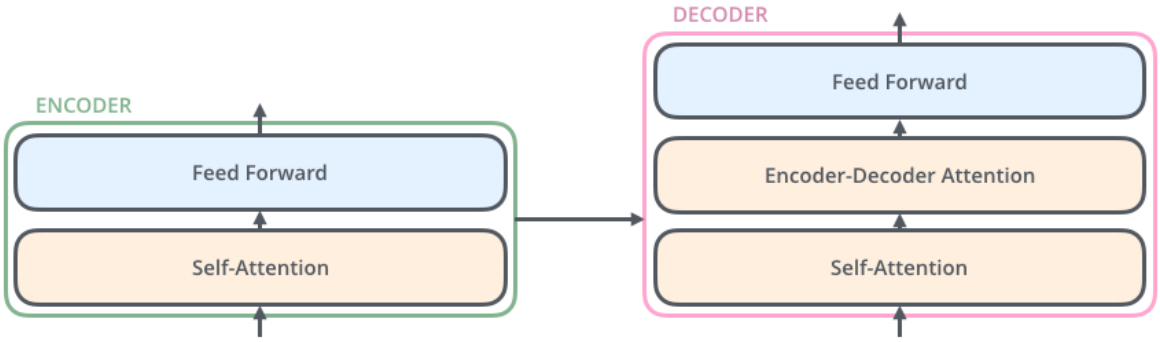
Všechny kodéry mají stejnou strukturu, avšak každý z nich může mít různé parametry, které určují jeho vlastnosti. Každý z kodérů je rozdělen na dvě dílčí vrstvy:



Vstupy kodéru nejprve procházejí vrstvou sebe-pozornosti (Self-Attention), tedy vrstvou, která kodéru pomáhá při kódování konkrétního tokenu sledovat ostatní tokeny ve vstupní sekvenci. Na sebe-pozornost, což je zásadní prvek Transformeru, se blíže podíváme později.

Výstupy z vrstvy sebe-pozornosti jsou potom přivedeny do dopředné neuronové sítě (Feed Forward Neural Network), někdy také označované jako vícevrstvý perceptron (Multi Layered Perceptron). Tato síť pak transformuje každý výstup z vrstvy sebe-pozornosti do upravené podoby.

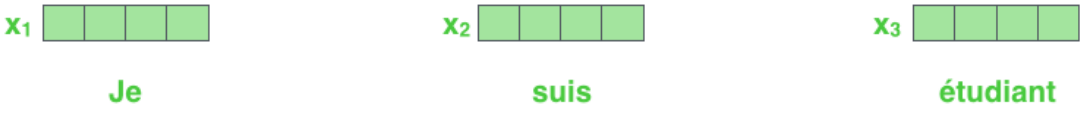
Dekodér má také obě tyto vrstvy, ale navíc je mezi nimi další vrstva pozornosti (Encoder-Decoder Attention), která pomáhá dekodéru zaměřit se na relevantní části vstupní věty.



1.4.2 EMBEDDING

Nyní, když jsme se seznámili s hlavními složkami Transformeru, se začneme zabývat různými vektorovými reprezentacemi slov, a tím, jak tyto vektory procházejí mezi komponentami Transformeru, aby se vstup natrénovaného modelu (sekvence tokenů) proměnil ve výstup (opět nějaká sekvence tokenů).

Stejně jako v jiných NLP aplikacích obecně začneme tím, že každý vstupní token převedeme na jeho vektorovou reprezentaci pomocí *embeddingu*. Tento algoritmus převede každý token na vektor o 512ti složkách. Pro ilustraci však budeme používat jenom zjednodušenou verzi se čtyřmi složkami, jako na tomto obrázku.



Ale co to vlastně embedding je? Hlavní myšlenkou embeddingu je přeměna textu na čísla. Tato přeměna je potřeba proto, protože mnoho algoritmů strojového učení (včetně hlubokých neuronových sítí) vyžaduje, aby jejich vstupem byly vektory reálných čísel; na řetězcích prostého textu prostě nefungují. Proto se k převedení slov nebo frází ze slovníku na odpovídající vektor reálných čísel používá technika modelování přirozeného jazyka, jako je embedding. Kromě toho, že je tato vektorová reprezentace vhodná pro zpracování učícími se algoritmy, má dvě důležité vlastnosti:

- Snížení dimenzionality - jedná se o úspornější reprezentaci
- Kontextová podobnost - jedná se o expresivnější reprezentaci.

Cílem embeddingu je vytvořit vektorovou reprezentaci původního textu s mnohem menší dimenzí. Tyto reprezentace se nazývají vektory tokenů.

Vektory tokenů se používají pro sémantický rozbor, k extrakci významu z textu, aby bylo možné porozumět přirozenému jazyku. Aby byl jazykový model schopen porozumět významu analyzovaného textu, musí si být vědom kontextové podobnosti slov v něm. Například, že často

nacházíme slova o ovoci (jako jablko nebo pomeranč) ve větách, kde se pěstuje, trhá, jí a odšťavňuje, ale neočekávali bychom, že najdeme stejné pojmy v těsné blízkosti, řekněme, slova letoun.

Vektory vytvořené embeddingem zachovávají tyto podobnosti, takže pro slova, která se v textu pravidelně vyskytují ve vzájemné blízkosti, budou také jejich vektorové reprezentace ve vzájemné blízkosti, měřené například Eukleidovskou vzdáleností.

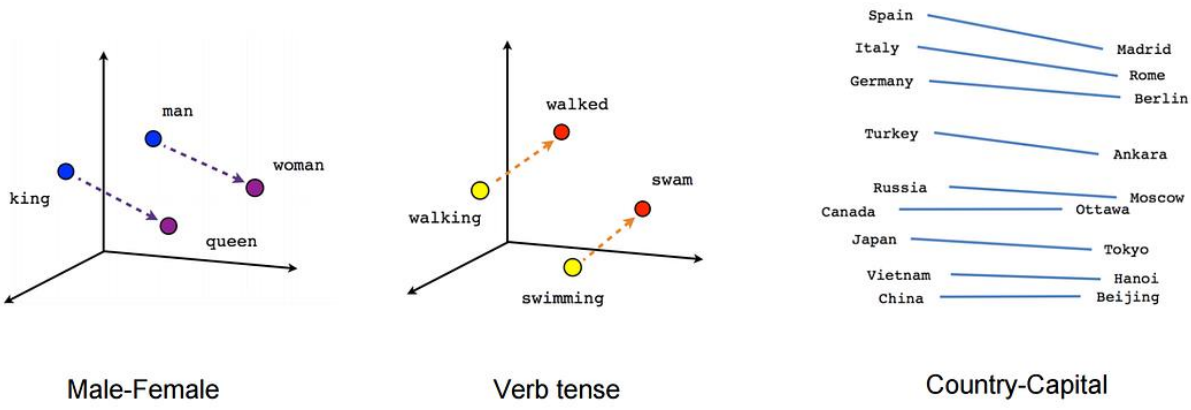
Takže odpověď na otázku „Co je to embedding?“ je: Je to prostředek ke tvorbě vektorové reprezentace korpusu textu o nízké dimenzi, který však zachovává významovou podobnost slov.

V tomto textu se nebudeme zabývat tím, jak přesně je tato reprezentace vytvořena, avšak detaily je možno najít např. Collis (2017), ze kterého jsou použity i následující příklady. Výsledkem tvorby embedding reprezentace je tzv. *matice embeddingu* (embedding matrix). Matice embeddingu má velikost (počet tokenů ve slovníku) x (velikost reprezentace každého tokenu v embedding modelu), kterou lze libovolně nastavit – u prvního modelu Transformeru byl použit embedding model s velikostí reprezentace 512 složek. Pokud tedy mámě např. 50 000 tokenů v našem slovníku (všechny jedinečné tokeny, které se objevují v textech, na kterých budeme Transformer trénovat), matice bude mít velikost 50 000 × 512. Každý řádek matice pak obsahuje vektorovou reprezentaci jednoho tokenu v našem slovníku. V následujícím příkladu má čtvrtý token v našem slovníku (tedy token s ID = 4) reprezentaci (10,12,19).

$$[0 \quad 0 \quad 0 \quad 1 \quad 0] \times \begin{bmatrix} 17 & 24 & 1 \\ 23 & 5 & 7 \\ 4 & 6 & 13 \\ 10 & 12 & 19 \\ 11 & 18 & 25 \end{bmatrix} = [10 \quad 12 \quad 19]$$

Velikost reprezentace jednoho tokenu bývá typicky mnohem menší než velikost našeho slovníku (spíše jsou to stovky než desítky tisíc). Nastavení velikosti této reprezentace je pak kompromisem: více složek v reprezentaci znamená další výpočetní složitost, čímž se prodlužuje doba nutná pro zpracování reprezentace, nicméně také umožňuje jemnější reprezentaci jazyka a tím potenciálně lepší modely.

Zajímavým rysem embeddingu je to, že protože se jedná o numerickou reprezentaci kontextových podobností mezi slovy, lze s nimi manipulovat aritmeticky. Populárním příkladem tohoto rysu je odečtení reprezentace slova „král“ od „muže“ a přičtení reprezentace slova „žena“. Odpověď samozřejmě závisí na našich trénovacích datech, ale pravděpodobně uvidíme, že jedním z nejlepších výsledků bude slovo „královna“, jak je zobrazeno na následujícím obrázku (více detailů je možno najít např. Migdał (2017)).



Male-Female Verb tense Country-Capital

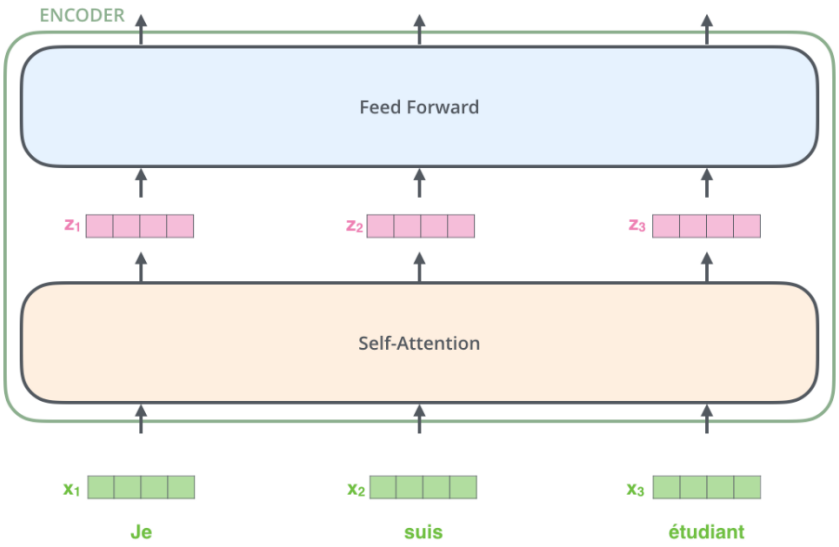
Embedding tedy reprezentuje slova pomocí podobností založených na jejich kontextu, který může být pohlaví, čas, geografie, atd. Zobrazené čáry jsou pouze matematické vektory, takže můžeme vidět, jak bychom se mohli pohybovat v prostoru reprezentací od „king“ po „queen“ odečtením „man“ a přičtením „woman“.

Stručně řečeno, cílem embeddingu je přeměnit slova na čísla, která pak mohou modely jako Transfomer zpracovávat lépe než kdyby bylo každé slovo pouze reprezentováno jako jedno celé číslo (např. ID) bez jakékoli informace o tom, jak souvisí s ostatními slovy.

1.4.3 EMBEDDING V TRANSFORMERU

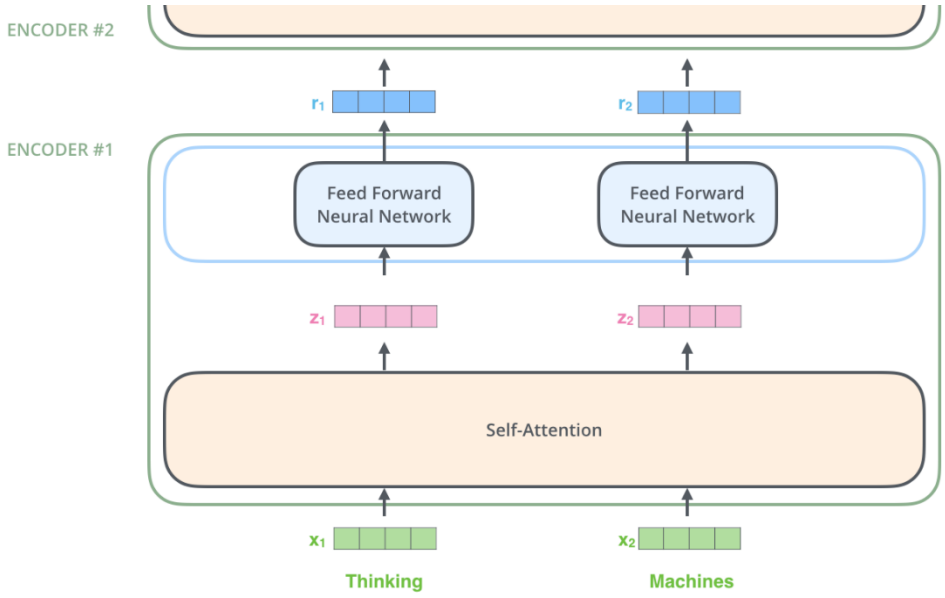
Embedding se týká pouze spodního kodéru, kde se jednotlivá ID tokenů vstupní sekvence převedou na vektorové reprezentace, např. od 512 složkách. To, co potom mají všechny kodéry společné je, že kodér v každé vrstvě Transformeru obdrží sadu vektorů, z nichž každý má vždy stejný počet složek, např. zmíněných 512. Ve spodním kodéru jsou tyto vektory dané embeddingem a v ostatních kodérech jsou to výstupy kodéru, který je přímo pod ním.

Po převedení tokenů naší vstupní sekvence na vektory pomocí embeddingu pak každá tento vektor prochází skrze obě vrstvy kodéru.



Zde se začíná projevovat jedna z klíčových vlastností Transformeru, a to, že token v každé pozici vstupní sekvence (v našem příkladu jsou to pozice 1 (Je), 2 (suis) a 3 (étudiant)) prochází svou vlastní cestou v kodéru. Mezi těmito cestami ve vrstvě sebe-pozornosti existují závislosti. Ve vrstvě dopředné neuronové sítě však tyto závislosti nejsou, a proto lze výpočet výstupu těchto neuronových sítí provádět paralelně, což umožňuje trénovat i modely s velmi vysokým počtem parametrů.

Nyní změníme náš příklad na kratší větu (Thinking Machines) a podíváme se, co se děje v jednotlivých vrstvách kodéru. Jak jsme se již zmínili výše, kodér dostává na vstup sadu vektorů (x – na obrázku níže). Tuto sadu pak zpracovává tak, že vektory předává do vrstvy sebe-pozornosti, ze které získá vektory z. Tyto vektory se pak předají do neuronové sítě, čímž se získají vektory r, což je výstup kodéru, a tento výstup se pak posílá nahoru dalšímu kodéru, jak vidíme na obrázku níže.



Právě přechod od reprezentace typu x na reprezentaci typu z, tady aplikace sebe-pozornosti, přináší dramatické zlepšení kvality výstupů takovýchto jazykových modelů. Proto se nyní na sebe-pozornost podíváme důkladněji.

1.4.4 Sebe-pozornost z Nadhledu

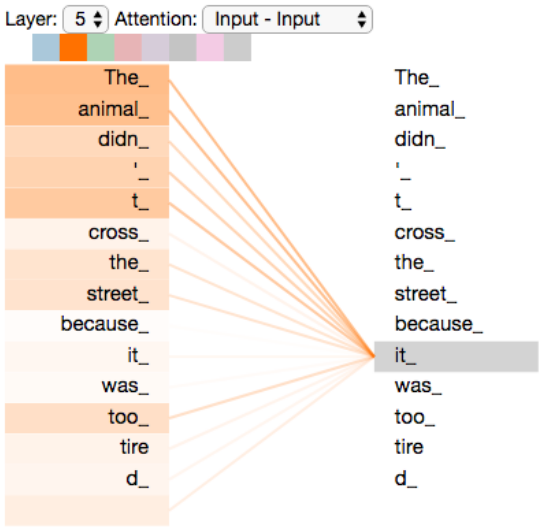
Řekněme, že následující věta je vstupní sekvencí, kterou chceme přeložit z angličtiny do jiného jazyka:

"`The animal didn't cross the street because` **`it`** `was too tired`".

K čemu se v této větě vztahuje slovo "it"? K ulici (street), nebo ke zvířeti (animal)? Pro člověka je to jednoduchá otázka, ale pro algoritmus už to tak jednoduchá otázka není. Když model zpracovává slovo "it", sebe-pozornost mu umožňuje spojit "it" se slovem "animal".

Během toho, jak model zpracovává každý token ve vstupní sekvenci, sebe-pozornost mu umožňuje podívat se na jiné tokeny v sekvenci a hledat souvislosti, které mohou pomoci k lepšímu zakódování daného tokenu - tedy opět nějaké další vektorové reprezentaci tohoto tokenu. Sebe-pozornost je tedy metoda, kterou Transformer používá pro to, aby mohl upravit reprezentaci právě zpracovávaného tokenu na základě další relevantních tokenů, které právě zpracovává.

Například, pokud token "it" kódujeme v kodéru č. 5 (Layer 5), část mechanismu sebe-pozornosti se zaměřila na token "animal" a zapojila část jeho reprezentace do kódování slova "it", jak je ilustrováno na obrázku níže. Díky vektorové reprezentaci tokenů tedy může ke slovům přistupovat aritmeticky a například sečíst slovo "it" se slovem "animal", čímž dostaneme reprezentaci, která představuje částečně "it" a částečně "animal", díky čemuž Transformer lépe chápe souvislosti v textu. Podívejme se nyní na tento mechanismus detailněji.



1.4.5 SEBE-POZORNOST V DETAILECH

Jak už víme, embedding umožňuje efektivní vektorovou reprezentaci slov. Má však jednu obrovskou Achillovu patu: slova, která mají více než jednu definici. Pokud embedding přiřadí vektor řekněme slovu „bank“, přiřadí stejný vektor všem definicím slova „bank“. Co když chcete toto slovo použít v různých kontextech? Zde pak právě vstupuje do hry sebe-pozornost, což je způsob, jak rozlišit stejná slova, která však mají různý význam podle použitého kontextu.

Abychom porozuměli sebe-pozornosti, podívejme se na tyto dvě věty[3]:

Věta 1: The **bank** of the river.

Věta 2: Money in the **bank**.

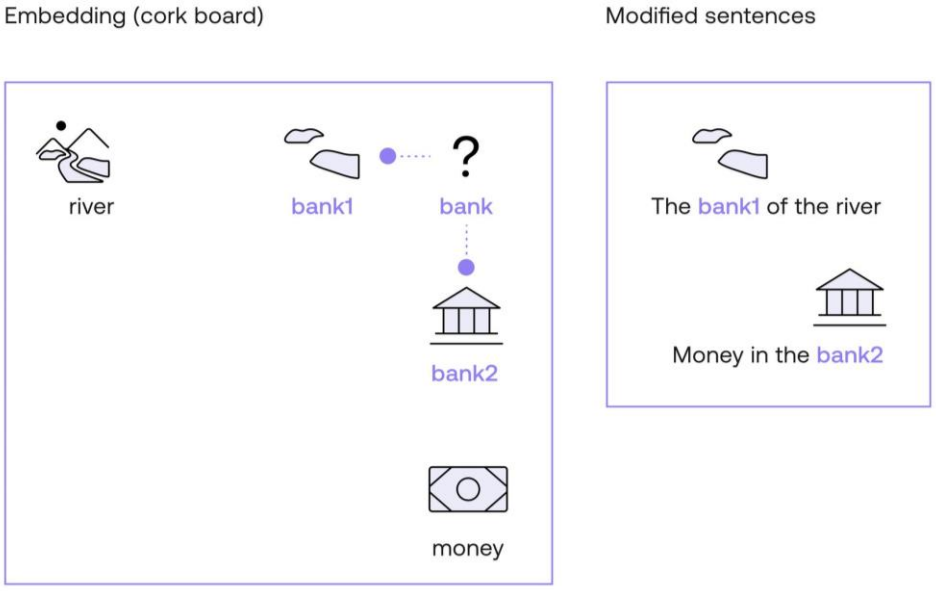
[3] https://txt.cohere.com/what-is-attention-in-language-models/

Jak počítač pozná, že slovo „bank“ v první větě odkazuje na přírodní úkaz (břeh) a ve druhé větě na finanční instituci (banku)? Položme si jednodušší otázku: Jak to může vědět člověk? Jak jsme přišli na tyto dva různé významy? Způsob, jakým jsme to udělali, bylo pravděpodobně to, že jsme podívali na sousední slova. V první větě bylo slovo „river“ tím, co naznačovalo přírodní úkaz, a ve druhé větě to bylo slovo „money“, které nás vedlo k souvislosti s finanční institucí. Stručně řečeno, potřebujeme způsob, jak využít další slova ve větě, abychom pochopili, který význam slova „bank“ chceme použít.

Podívejme se na náš příklad geometricky. Představme si, že slova „bank“, „river“ a „money“ jsou připevněna na nástěnce (cork board), viz obrázek níže. Dále si představme, že tato nástěnka obsahuje všechna další existující slova, a to tak, že dvě podobná slova (například „jablko“ a „hruška“) jsou blízko sebe. Jak už víme, taková reprezentace slov je výsledkem embeddingu. Nyní na této nástěnce nejsou „bank“, „river“ ani „money“ zrovna blízko. Nicméně, co můžeme udělat je, že vezmeme slovo „bank“ a přesuneme jej mírně směrem ke slovu „river“. Nazvěme toto slovo „bank1“. Nyní vezmeme další kopii slova „bank“ a mírně ji posuneme směrem ke slovu „money“. Říkejte tomu „bank2“. Nyní zvažme následující dvě upravené věty.

Upravená věta 1: The **bank1** of the river.

Upravená věta 2: Money in the **bank2**.

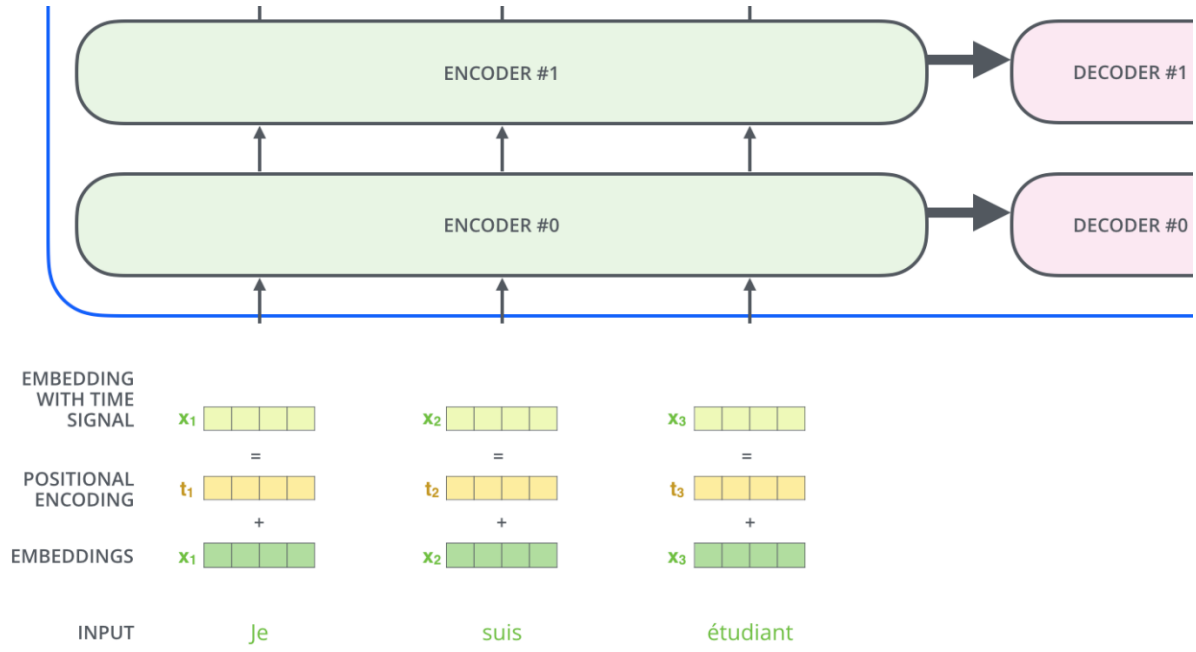


V těchto dvou větách už počítač ví trochu více o významu slova „bank“, protože slovo bylo rozděleno do dvou odlišných slov. Jedno, jehož význam je blíže k „river“, a další, jehož význam je blíže k „money“. Toto je hlavní princip mechanismu sebe-pozornosti, který řeší zmíněnou Achilovu patu embeddingu.

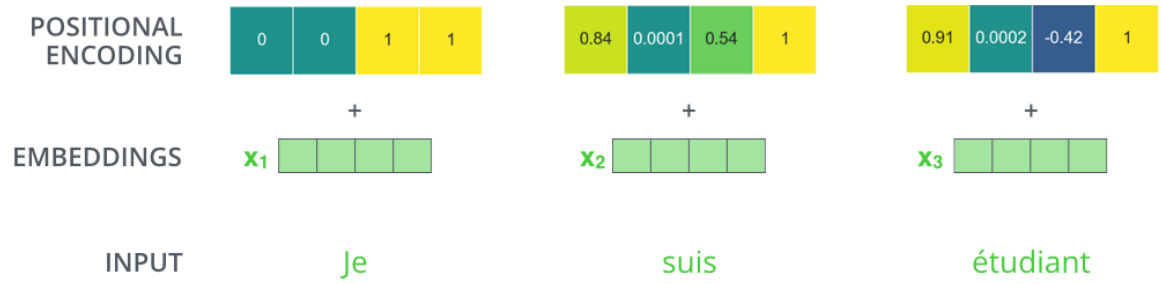
1.4.6 Reprezentace Pořadí Slov ve Vstupní Větě

Jedna věc, která v modelu, jak jsme ho dosud popisovali, chybí, je způsob, jak zohlednit pořadí slov ve vstupní sekvenci. Představme si větu "The dog chases the cat" a porovnejme ji se větou "The cat chases the dog". I když obě věty obsahují stejná slova, význam je diametrálně odlišný právě kvůli pořadí slov. V prvním případě je pes ten, kdo pronásleduje kočku, zatímco ve druhém případě kočka pronásleduje psa. Modely zpracovávající přirozený jazyk musí toto pořadí zohlednit, aby správně pochopily význam a kontext vět. Transformer toho dosahuje právě pomocí tzv. *pozičních vektorů*, které umožňují modelu zachytit tuto důležitou složku informace o pořadí slov. Tyto vektory mají specifický vzor, který mu pomáhá určit pozici každého slova nebo vzdálenost mezi různými slovy ve zpracovávané větě nebo sekvenci slov. Intuice je taková, že přidání těchto hodnot k embedding vektorům umožňuje vytvoření smysluplných vzdáleností mezi vektorovými reprezentacemi.

Abychom tedy dali modelu informaci o pořadí slov, přičteme poziční vektory k původním reprezentacím z embeddingu, jak je zobrazeno níže.



Pokud předpokládáme, že reprezentace z embeddingu mají čtyři složky, poziční kódování (positional encoding) by mohlo vypadat takto:



Poziční kódování je navrženo tak, aby modelu poskytlo informace o pozici slov ve vstupní větě. Při vektorové reprezentaci o čtyřech složkách tedy toto kódování může být reprezentováno jako sada vektorů, kde každý vektor představuje unikátní pozici v sekvenci. Poziční kódování se obvykle generuje pomocí funkcí sinus a kosinus s různými frekvencemi podle předem definovaných rovnic. Např. takto:

- Pozice 1: $([\sin(\omega_1), \cos(\omega_1), \sin(\omega_2), \cos(\omega_2)])$

- Pozice 2: $([\sin(\omega_1 * 2), \cos(\omega_1 * 2), \sin(\omega_2 * 2), \cos(\omega_2 * 2)])$

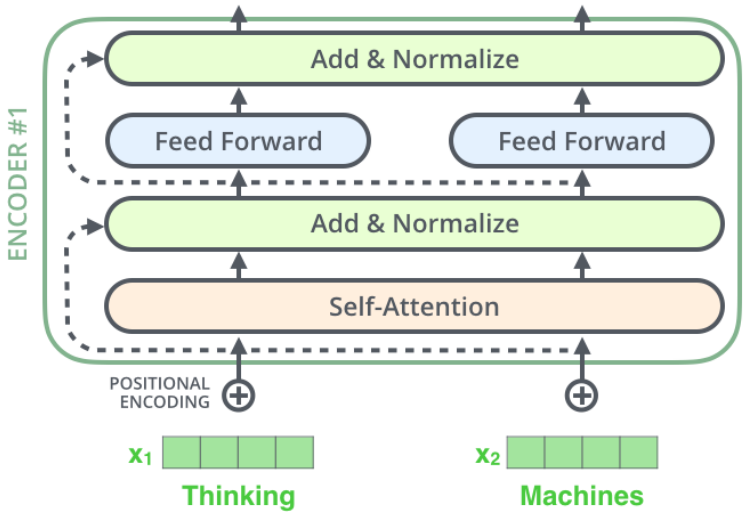
- Pozice 3: $([\sin(\omega_1 * 3), \cos(\omega_1 * 3), \sin(\omega_2 * 3), \cos(\omega_2 * 3)])$

- Pozice 4: $([\sin(\omega_1 * 4), \cos(\omega_1 * 4), \sin(\omega_2 * 4), \cos(\omega_2 * 4)])$

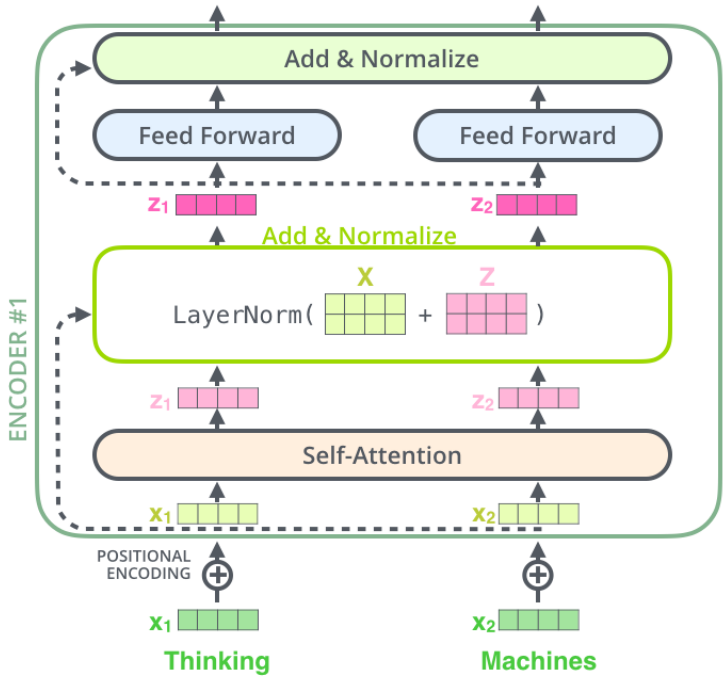
Zde jsou (ω_1) a (ω_2) jsou předem zvolené frekvence použité pro generování pozičních vektorů. Tyto frekvence se liší pro každou dimenzi vektoru, tím pádem je model schopen po jejich přičtení k embeddingu odlišit pozice jednotlivých tokenů ve vstupní sekvenci.

1.4.7 Residua

Než budeme pokračovat dál, je potřeba zmínit jeden detail v architektuře kodéru. Každá podvrstva (sebe-pozornost a dopředná neuronová síť) v každém kodéru má kolem sebe tzv. *reziduální spojení* a je následována krokem *normalizace vrstvy* (layer-normalization), viz Ba a kol. (2016). Reziduální spojení, která jsou v obrázku níže označena přerušovanou čárou, umožňují modelu přenášet hodnoty vektorů reprezentací slov přímo přes jednu nebo více vrstev bez jejich změny. Tato kombinace reziduálních spojení a normalizace vrstvy je klíčová pro efektivní učení a vysokou výkonnost transformátorových modelů, protože pomáhá řešit problémy jako gradientním mizením a explodujícími gradienty, které jsme rozebírali v souvislosti s Rannými modely.

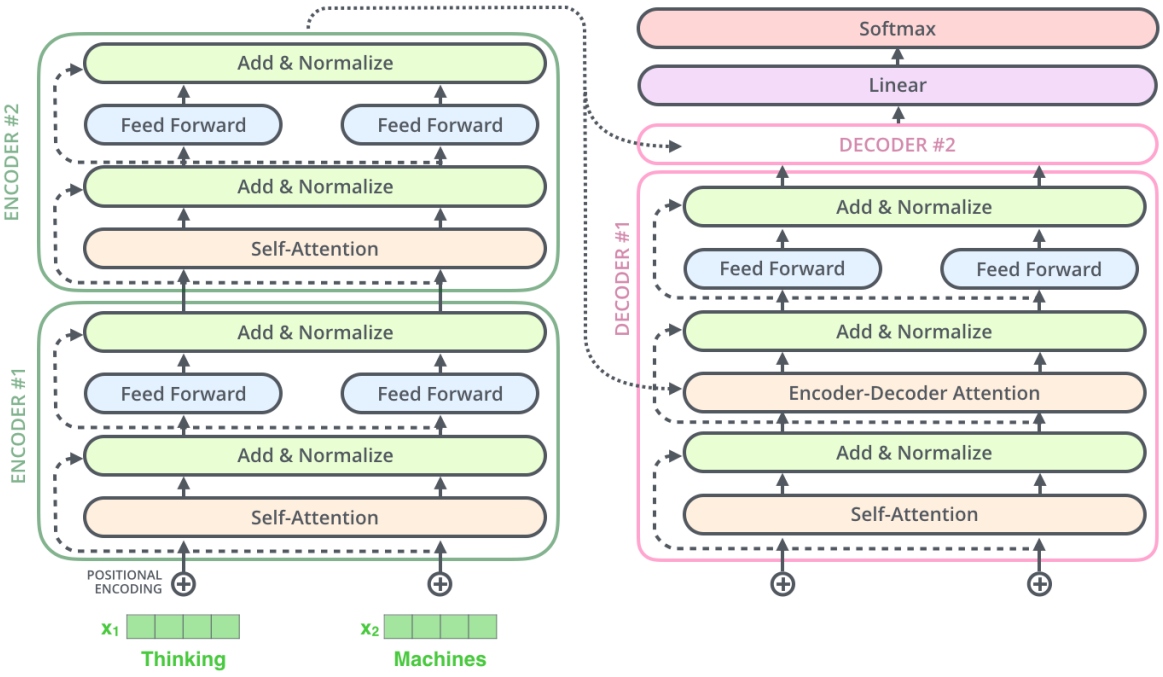


Pokud bychom si měli vizualizovat vektory a normalizaci vrstvy sebe-pozornosti, vypadalo by to takto:



Tento obrázek nám také dává dobrou představu o možnostech paralelizace procesu. Zatímco vrstva sebe-pozornosti potřebuje pro zpracování reprezentací tokenů všechny tokeny ve vstupní sekvenci, tedy zde nelze paralelizovat, průchod reprezentace tokenu dopřednou neuronovou sítí je *nezávislý* na ostatních tokenech, a tudíž můžeme paralelizovat. Uvědomíme-li si, že např. model GPT-4 zpracovává vstupní sekvence o velikosti až 128000 tokenů, je zde tím pádem možné až 128000 násobné zrychlení.

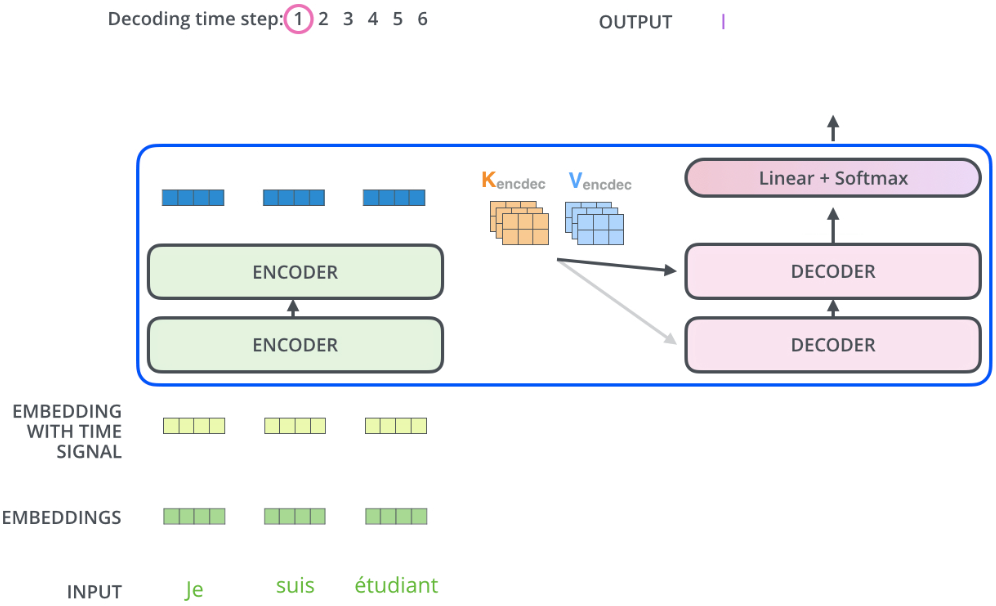
Reziduální spojení a normalizace jsou použity také v podvrstvách dekodéru. Pokud bychom uvažovali Transformer se dvěma kodéry a dekodéry, vypadalo by to asi takto:



1.4.8 DEKODÉR

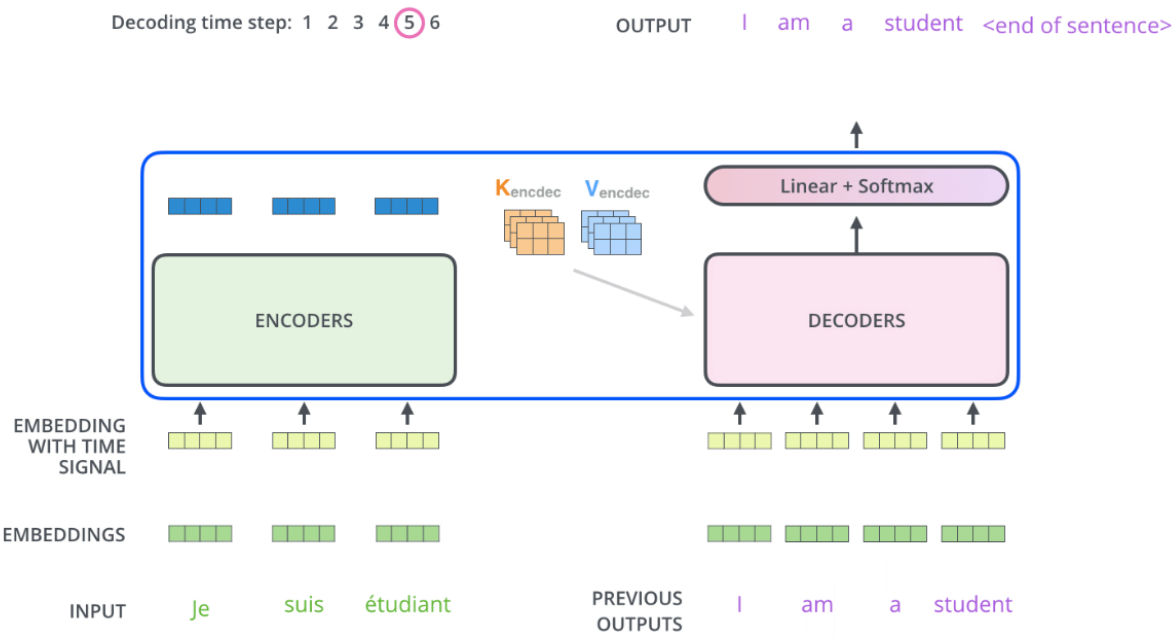
Nyní, když jsme probrali většinu konceptů na straně kodéru, v podstatě také víme, jak fungují dekodéry. Podívejme se teď na to, jak spolu kodéry a dekodéry vzájemně spolupracují.

Kodér začne zpracováním vstupní věty. Výstup z horního kodéru je pak přeměněn na sadu pozornostních vektorů označených jako K a V, které se pak používají každým dekodérem ve vrstvě pozornosti kodér-dekodér (Encoder-Decoder Attention), která pomáhá dekodéru soustředit se na vhodná místa ve vstupní sekvenci:



Po dokončení fáze kódování začínáme s fází dekódování. Každý krok ve fázi dekódování vydá jeden prvek z výstupní sekvence (v tomto případě větu anglického překladu, cože je první slovo "I").

Tento proces se pak opakuje, dokud není dosaženo speciálního symbolu, který indikuje, že dekodér Transformeru dokončil svůj výstup. Výstup každého kroku je veden do spodního dekodéru v dalším časovém kroku a dekodéry předávají své výsledky dekódování nahoru, stejně jako to dělaly kodéry. A stejně jako jsme to udělali s vstupy kodéru, vložíme a přidáme poziční kódování k těmto vstupům dekodéru, abychom indikovali pozici každého slova.



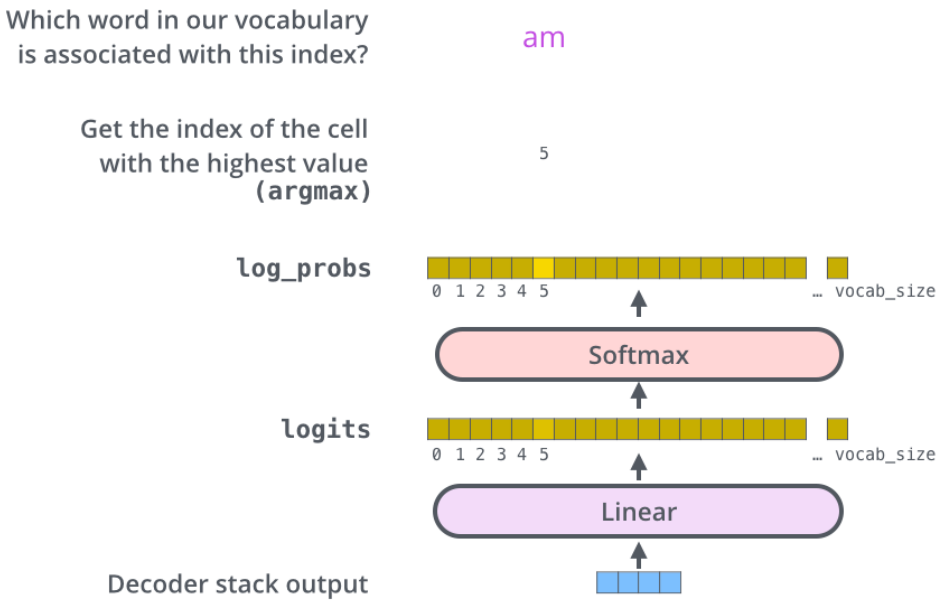
Vrstvy sebe-pozornosti v dekodéru fungují mírně odlišně než v kodéru: V dekodéru je vrstvě sebe-pozornosti povoleno věnovat se pouze *dřívějším* pozicím ve výstupní sekvenci, aby se zabránilo úniku informací z "budoucnosti". Toto omezení je zásadní, protože v době generování každého slova v sekvenci nemá dekodér ještě přístup k informacím o slovech, která mají být vygenerována v budoucnu. Tímto způsobem se zajišťuje, že každé slovo je generováno pouze na základě již vygenerovaných slov a relevantních informací z kodéru, což odpovídá postupnému charakteru přirozeného jazyka a sekvenční povaze úlohy generování textu. Maskování budoucích pozic tedy předchází nechtěnému "podvádění" tím, že se zamezí pohledu dekodéru na slova, která má teprve generovat, což umožňuje modelu učit se efektivnější a koherentní generování textu.

1.4.9 Finální Lineární Vrstva a Vrstva Softmax

Výstup dekodéru je vektor s reálnými čísly. Jak jej však přeměníme na slovo? To je úkol finální lineární vrstvy (Linear), po které následuje vrstva Softmax. Lineární vrstva je jednoduchá plně propojená neuronová síť, která promítá vektor generovaný sadou dekodérů do mnohem většího vektoru zvaného logity.

Předpokládejme, že náš model zná 10 000 unikátních anglických slov („výstupní slovník“ našeho modelu), které se naučil z trénovacích dat. To by znamenalo, že vektor logitů bude mít 10 000 složek – každá složka odpovídá skóre unikátního slova, kde skóre odpovídá tomu, jak moc si model "myslí", že by toto slovo mělo následovat po předchozích slovech. Takto interpretujeme výstup modelu po projekci lineární vrstvou.

Vrstva Softmax poté tyto skóre přemění na pravděpodobnosti (všechny kladné, všechny dohromady dají součet 1). Buňka s nejvyšší pravděpodobností je vybrána, a slovo s ní spojené je generováno jako výstup Transformeru pro tento časový krok. Na obrázku níže bylo skóre nejvyšší pro token ID = 5, který v našem slovníku odpovídá slovu "am".



KONTROLNÍ OTÁZKA

1. Co je hlavním přínosem použití mechanismu sebe-pozornosti v architektuře Transformer?
2. Jaký je rozdíl mezi kódovací a dekódovací částí v Transformeru?
3. Jaké jsou dva hlavní typy vrstev nacházející se v každém kodéru Transformeru?
4. Jak Transformer řeší problém s pořadím slov ve vstupní sekvenci?
5. Jaký je účel finální lineární vrstvy a vrstvy Softmax v architektuře Transformeru?

1.5 Učení Transformeru

V předchozí části jsme se seznámili s tím, jak Tranformer převede vstupní sekvenci tokenů na výstupní. Nyní se stručně podívejme na to, jak se Transformer učí z trénovacích dat. Během učení se sekvence z trénovacích dat postupně předávají Transformeru, který na jejich základě generuje výstupní sekvence. Protože ale v trénovacích datech máme i pokračování vstupních sekvencí, můžeme výstup Transformeru s tímto pokračováním porovnat.

Představme si, že náš výstupní slovník obsahuje pouze šest slov („a“, „am“, „i“, „thanks“, „student“ a „<eos>“ (zkratka pro 'end of sentence')), a jejich ID (Index) jsou určeny jako na obrázku níže.

Output Vocabulary

WORD	a	am	I	thanks	student	<eos>
INDEX	0	1	2	3	4	5

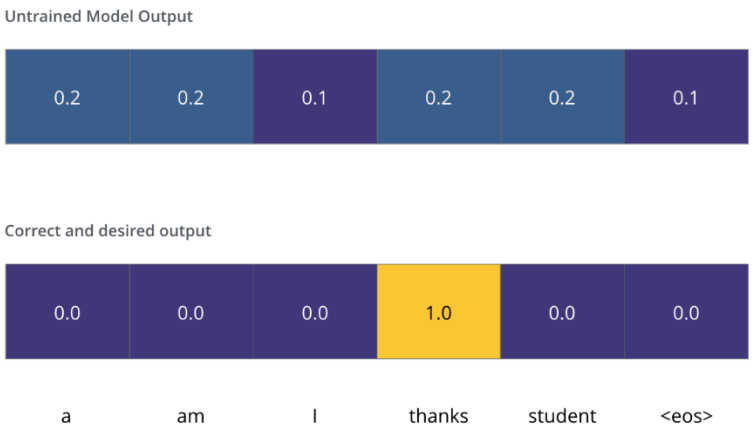
Jakmile definujeme náš výstupní slovník, můžeme použít vektor stejné šířky k označení každého slova v našem slovníku. Toto se nazývá *one-hot* kódování, kde vektor má počet složek roven velikosti slovníku, a složka, která odpovídá ID, je rovna 1, jinak 0. Například slovo „am“ můžeme zakódovat následujícím vektorem:

One-hot encoding of the word “am”

0.0	1.0	0.0	0.0	0.0	0.0

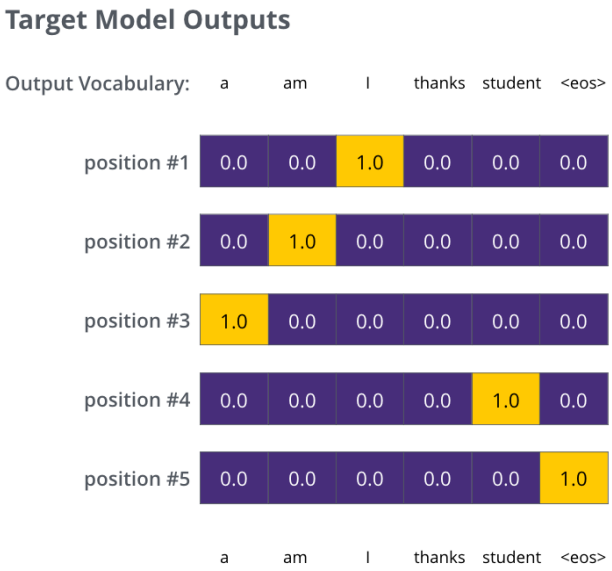
Nyní můžeme přejít k chybové funkci modelu – metrice, kterou minimalizujeme během fáze učení, abychom získali pokud možno co nejkvalitnější model.

Předpokládejme, že provádíme první krok ve fázi učení na jednoduchém příkladu – překládáme „merci“ na „thanks“. To znamená, že chceme, aby výstupem bylo pravděpodobnostní rozdělení ukazující na slovo „thanks“. Ale protože parametry Transformeru jsou na začátku učení nastaveny náhodně, je nepravděpodobné, že bychom hned dostali tento výsledek. Spíše dostaneme něco, jako na obrázku níže, kde jsou pravděpodobnosti jednotlivých slov přiřazeny nenatrénovaným (Untrained) modelem v podstatě náhodně.

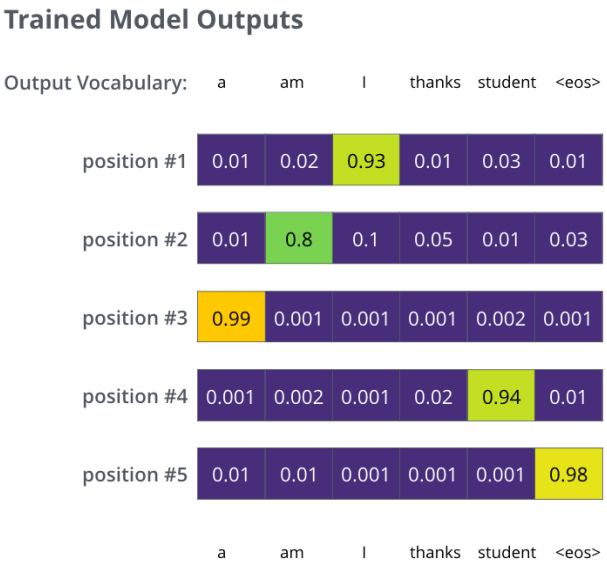


Jak ale porovnáme dvě taková pravděpodobnostní rozdělení? Jednoduše to lze udělat např. tak, že odečteme jedno rozdělení od druhého. Ale pozor, toto je zjednodušený příklad. Realističtěji budeme používat sekvenci delší než jedno slovo. Například – vstup: „je suis étudiant“ a očekávaný výstup: „i am a student“. To ve skutečnosti znamená, že chceme, aby náš model postupně generoval pravděpodobnostní rozdělení, kde:

- Každé pravděpodobnostní rozdělení je reprezentováno vektorem o šířce velikosti slovníku (6 v našem zjednodušeném příkladu, ale realističtěji číslo jako 30 000 nebo 50 000)
- První pravděpodobnostní rozdělení má nejvyšší pravděpodobnost v buňce spojené se slovem „i“
- Druhé pravděpodobnostní rozdělení má nejvyšší pravděpodobnost v buňce spojené se slovem „am“
- A tak dále, až páté pravděpodobnostní rozdělení ukazuje symbol ‚<eos>‘, který také odpovídá jednomu prvku ze slovníku.



Po dostatečně dlouhém tréninku modelu na dostatečně velkých datech bychom doufali, že Transformer začne generovat pravděpodobnostní rozdělení jako na obrázku níže, které by ve výsledku dalo správný anglický překlad:



Protože Transformer generuje jako výstup postupně, tedy slovo za slovem, standardní přístup je takový, že Transformer vygeneruje první pravděpodobnostní rozdělení, a výstupní slovo se vybere jako slovo odpovídající nejvyšší pravděpodobnosti z daného rozdělení a zbylá slova se neuvažují. Tento způsob se nazývá hladové (greedy) dekódování. Po vygenerování celé výstupní sekvence se všechna generovaná pravděpodobnostní rozdělení (Trained Model Outputs) porovnají s očekávanými pravděpodobnostními rozděleními dané one-hot kódováním (Target Model Outputs) a spočítá se chyba modelu, např. suma kvadrátů rozdílů mezi složkami vektorů. Poté se, stejně jako u standardních neuronových sítí, parametry upraví např. pomocí algoritmu zpětného šíření chyby (back-propagation), a učící proces se opakuje, dokud není dosaženo nějakého kritéria pro zastavení učení, např. maximální čas učení, maximální počet iteraci, apod.

KONTROLNÍ OTÁZKA

1. Jak se porovnávají pravděpodobnostní rozdělení generovaná Transformerem s očekávanými výstupy během učení?
2. Co znamená pojem 'greedy dekódování' v kontextu generování výstupní sekvence Transformerem?

1.6 Záblesky AGI

Po tom, co jsme se seznámili s fungováním a učením Transformerů, si můžeme položit otázku, jak tyto systémy, které ve své podstatě pouze předpovídají následující slovo v textové sekvenci, dokážou generovat výstupy, které vykazují známky pokročilé inteligence. Kvalita a soudržnost textů generovaných například modelem GPT-3 a jeho následníky ukázaly, že i "jednoduchá" predikce slov může vést k výsledkům, které někteří odborníci považují za překvapivě smysluplné a inteligentní. To naznačuje, že pro generování koherentního a relevantního textu musí modely vykazovat určitou úroveň porozumění kontextu a závislostem v jazyce, což jsou atributy tradičně spojované s přirozenou inteligencí.

Zajímavostí je, že vývoj a pokrok v technologii Transformer nám poskytují záblesky toho, co by mohlo být považováno za začátky obecné umělé inteligence (AGI) - schopnosti strojů zpracovávat informace a vykonávat úkoly na úrovni srovnatelné s lidskou inteligencí, přesahující zaměření pouze na specifické úlohy. Modely Transformer ukazují, že je možno směřovat k systémům, které nejsou omezené na jednu úlohu, ale jsou schopné flexibilního a inteligentního zpracování rozlišných typů informací.

I když skutečné AGI, systém s plně autonomní inteligencí srovnatelnou s lidskou, ještě neexistuje, vývoj Transformerů naznačuje, že bychom se k jeho realizaci mohli blížit. Vývojáři a výzkumníci v oblasti umělé inteligence vkládají do těchto modelů velké naděje, protože jejich unikátní schopnosti zpracovávat a generovat jazyk naznačují možnou cestu k dosažení pokročilejších forem AI, možná i směrem k AGI. Tento pokrok nejenže posouvá hranice toho, co je možné v oblasti NLP, ale také otevírá nové možnosti pro pochopení a vytváření inteligentních systémů.

SHRNUTÍ KAPITOLY

Tato kapitola se věnuje zásadnímu pokroku v oblasti umělé inteligence (AI), který umožnil vývoj velkých jazykových modelů, jako jsou ChatGPT od OpenAI a Gemini od Google DeepMind. Milníkem je technologie Transformer, která umožnila dosažení dosud nevídané kvality v generování, pochopení a interpretaci lidského jazyka. Kapitola popisuje, co jsou to jazykové modely, jejich zásadní roli v AI, a představuje koncepty jako embedding slov a mechanismus sebe-pozornosti. Dále se zabývá historií a vývojem od prvních statistických modelů po sofistikované systémy založené na Transformeru, které jsou schopné efektivně zpracovávat dlouhé textové sekvence díky paralelnímu zpracování a pokročilému modelování jazykových struktur. V kontextu učení Transformeru je vysvětleno, jak modely generují výstupy a jak se učí z trénovacích dat. Kapitola také naznačuje, že vývoj těchto technologií může představovat první kroky směrem k obecné umělé inteligenci, což otevírá diskuzi o potenciálu a výzvách spojených s vývojem systémů s lidskou úrovní adaptability a inteligence.

OTÁZKY

1. Co je jazykový model a jaké má aplikace?
2. Jaký je rozdíl mezi statistickými a neuronovými jazykovými modely?
3. Jak se spočítá pravděpodobnost následujícího slova v statistickém jazykovém modelu?
4. Co je umělá inteligence a jaké úkoly dokáže vykonávat?
5. Jaký je rozdíl mezi úzkou AI a obecnou AI?
6. Jaké přínosy a problémy by představovalo vytvoření AGI?
7. Co jsou to gradientní mizení a explodující gradienty a jak ovlivňují trénování RNN a jejich variant?
8. Proč je sekvenční zpracování dat v RNN a jejich variantách náročné na výpočetní zdroje?
9. Jak architektura Transformer překonává omezení raných sekvenčních modelů?
10. Co je hlavním přínosem použití mechanismu sebe-pozornosti v architektuře Transformer?
11. Jaký je rozdíl mezi kódovací a dekódovací částí v Transformeru?
12. Jaké jsou dva hlavní typy vrstev nacházející se v každém kodéru Transformeru?
13. Jak Transformer řeší problém s pořadím slov ve vstupní sekvenci?
14. Jaký je účel finální lineární vrstvy a vrstvy Softmax v architektuře Transformeru?
15. Jak se porovnávají pravděpodobnostní rozdělení generovaná Transformerem s očekávanými výstupy během učení?
16. Co znamená pojem 'greedy dekódování' v kontextu generování výstupní sekvence Transformerem?

ODPOVĚDI

1. Jazykový model je algoritmus pro interpretaci, generování a porozumění lidskému jazyku, používaný v aplikacích jako automatické překladače a virtuální asistenti.
2. Statistické modely využívají tradiční statistické metody pro odhad pravděpodobnosti, zatímco neuronové modely používají umělé neuronové sítě pro flexibilnější učení složitějších jazykových vzorců.
3. Pravděpodobnost následujícího slova se v statistickém modelu spočítá na základě frekvence jeho výskytu po dané sekvenci slov v trénovacích datech.
4. Umělá inteligence je technologie simulující lidskou inteligenci pro úkoly jako učení, rozhodování a zpracování přirozeného jazyka.
5. Úzká AI je navržena pro specifické úkoly, zatímco obecná AI by měla pochopit, učit se a aplikovat inteligenci přes široké spektrum úkolů.
6. AGI by přineslo výhody jako zlepšení zdravotní péče a efektivity, ale i výzvy jako ztráta pracovních míst a etické dilema.
7. Gradientní mizení způsobuje, že se váhy ve spodních vrstvách sítě aktualizují málo, zatímco explodující gradienty vedou k nestabilitě modelu.
8. Sekvenční zpracování je náročné na výpočetní zdroje, protože každý prvek sekvence musí být zpracován postupně, což brání efektivní paralelizaci.

9. Transformer překonává omezení díky mechanismu sebe-pozornosti, který umožňuje paralelní zpracování dat a efektivnější modelování závislostí.
10. Hlavním přínosem je efektivnější zpracování dlouhých sekvencí dat a lepší zachycení složitých jazykových struktur.
11. Kódovací část zpracovává vstupní text, zatímco dekódovací generuje výstupní text na základě informací z kódovací části.
12. V každém kodéru jsou vrstva sebe-pozornosti a dopředná neuronová síť.
13. Používá poziční vektory pro přidání informace o pořadí slov k vektorovým reprezentacím.
14. Přeměňují vektorový výstup dekodéru na slova tím, že promítají vektor do většího prostoru logitů a převádějí je na pravděpodobnosti.
15. Porovnávají se pomocí chybové funkce, která měří rozdíl mezi generovanými a očekávanými pravděpodobnostními rozděleními.
16. 'Greedy dekódování' znamená výběr slova s nejvyšší pravděpodobností z generovaného rozdělení pro každý krok bez zvažování celkové optimální sekvence.

LITERATURA

Alammar, J., 2018. *The Illustrated Transformer* [online]. Blog post [vid. 13. února 2024]. Dostupné z: https://jalammar.github.io/illustrated-transformer

Ba, J.L., Kiros, J.R. and Hinton, G.E., 2016. Layer normalization. *arXiv preprint arXiv:1607.06450*.

Collis, J., 2017. *Glossary of Deep Learning: Word Embedding* [online]. Blog post [vid. 13. února 2024]. Dostupné z: https://medium.com/deeper-learning/glossary-of-deep-learning-word-embedding-f90c3cec34ca

Górecki, J., 2023. Pair programming with ChatGPT for sampling and estimation of copulas. *Computational Statistics*, pp. 1-31.

Migdał, P., 2017. *king - man + woman is queen; but why?* [online]. Blog post [vid. 13. února 2024]. Dostupné z: https://p.migdal.pl/2017/01/06/king-man-woman-queen-why.html

Schmidhuber, Jürgen (1992). Learning to control fast-weight memories: an alternative to recurrent nets. *Neural Computation*. 4 (1): 131–139

Silver, D., Schrittwieser, J., Simonyan, K., Antonoglou, I., Huang, A., Guez, A., Hubert, T., Baker, L., Lai, M., Bolton, A. and Chen, Y., 2017. Mastering the game of go without human knowledge. *Nature*, *550*(7676), pp. 354-359.

Vaswani, A., Shazeer, N., Parmar, N., Uszkoreit, J., Jones, L., Gomez, A.N., Kaiser, Ł. and Polosukhin, I., 2017. Attention is all you need. *Advances in neural information processing systems*, *30*.

SHRNUTÍ STUDIJNÍ OPORY

Stručné shrnutí či rekapitulace celého studijního textu, včetně doporučení studentům ke studiu, literatuře … Závěrečné slovo autora.

PŘEHLED DOSTUPNÝCH IKON


Čas potřebný ke studiu


Cíle kapitoly


Klíčová slova


Nezapomeňte na odpočinek


Průvodce studiem


Průvodce textem


Rychlý náhled


Shrnutí


Tutoriály

Definice


K zapamatování


Případová studie


Řešená úloha

Věta


Kontrolní otázka


Korespondenční úkol


Odpovědi


Otázky


Samostatný úkol


Další zdroje


Pro zájemce


Úkol k zamyšlení

Pozn. Tuto část dokumentu nedoporučujeme upravovat, aby byla zachována správná funkčnost vložených maker. Tento poslední oddíl může být zamknut v MS Word 2010 prostřednictvím menu Revize/Omezit úpravy.

Takto je rovněž omezena možnost měnit například styly v dokumentu. Pro jejich úpravu nebo přidávání či odebírání je opět nutné omezení úprav zrušit. Zámek není chráněn heslem.

Název: **Velké jazykové modely** v kontextu Umělé Inteligence

Autor: **Jména autorů každé na nový řádek včetně titulů**

Vydavatel: Slezská univerzita v Opavě

Obchodně podnikatelská fakulta v Karviné

Určeno: studentům SU OPF Karviná

Počet stran: 39