

Kapitola 6

Generativní umělá inteligence

V této kapitole se zaměříme na generativní umělou inteligenci, což je oblast umělé inteligence zaměřená na vytváření nového obsahu, například textu, obrázků nebo zvuku. Na rozdíl od klasických modelů, které například rozpoznávají nebo klasifikují data, generativní modely se učí data „napodobovat“ a vytvářet nové vzorky, které se podobají těm původním. Generativní umělá inteligence stojí za moderními nástroji, jako jsou jazykové modely, generátory obrázků nebo systémy pro tvorbu hudby.

Cíle kapitoly (po přečtení této kapitoly budete schopni)

- C1.1. Vysvětlit, co je generativní umělá inteligence a jak se liší od klasických přístupů v umělé inteligenci.
- C1.2. Popsat rozdíl mezi generativními a diskriminačními modely a vysvětlit, jaký typ úloh jednotlivé přístupy řeší.
- C1.3. Vysvětlit základní princip fungování generativních modelů, zejména jejich schopnost modelovat rozdělení dat a vytvářet nové vzorky.
- C1.4. Popsat, jak generativní modely postupně vytvářejí výstup (například text nebo obraz) na základě pravděpodobnosti.
- C1.5. Uvést praktické příklady využití generativní umělé inteligence v různých oblastech, jako je generování textu, obrázků nebo jiných typů dat.

Klíčové myšlenky (po přečtení této kapitoly byste si měli pamatovat)

- KM1.1. Generativní umělá inteligence je zaměřena na vytváření nových dat, která se podobají datům, na kterých byl model natrénován.
- KM1.2. Na rozdíl od klasických (diskriminačních) modelů generativní modely nejen rozpoznávají vzory v datech, ale učí se jejich strukturu a dokážou ji reprodukovat.
- KM1.3. Generativní modely pracují na základě pravděpodobnosti a při generování výstupu postupně rozhodují, jaký prvek (například slovo nebo pixel) bude následovat.
- KM1.4. Výsledná kvalita generovaného obsahu závisí především na kvalitě a množství dat, na kterých byl model natrénován.
- KM1.5. Generativní umělá inteligence nachází uplatnění v široké škále aplikací – od generování textu přes tvorbu obrázků až po syntézu zvuku nebo dat pro další modely.

Nové příkazy (po přečtení kapitoly se seznámím s těmito novými příkazy)

```
nn.Linear(), torch.relu(), torch.randn_like(), torch.sigmoid(), optimizer.step(), nn.functional.binary_cross_entropy(),  
transforms.Compose(), datasets.MNIST(), DataLoader(), optim.Adam(), optimizer.zero_grad(), loss.backward(),  
optimizer.step(), model.eval()
```

1.1 Proč vznikla generativní umělá inteligence?

V tradičním pojetí umělé inteligence jsou modely navrhovány především tak, aby analyzovaly existující data a na jejich základě prováděly určité rozhodnutí nebo odhad. Tyto modely označujeme jako diskriminační modely. Patří sem:

- klasifikační modely – typickým příkladem je model, který na základě obrázku rozhodne, zda se jedná o kočku, nebo psa, nebo model, který určí, zda je e-mail spam, a
- regresní modely – například odhad ceny domu nebo predikce teploty.

Tyto přístupy jsou velmi užitečné, ale mají jedno zásadní omezení – pracují pouze s již existujícími daty a jejich cílem je buď správně určit kategorii, nebo odhadnout konkrétní hodnotu.

Postupem času se však ukázalo, že v mnoha úlohách nestačí data pouze analyzovat, ale je potřeba je také vytvářet. Například chceme:

- generovat text – například odpovědi na otázky nebo shrnutí dokumentů apod.,
- vytvářet obrázky na základě popisu,
- doplňovat chybějící části dat,
- simulovat nové scénáře pro další modelování aj.

Právě zde vzniká potřeba nového druhu umělé inteligence – generativní umělé inteligence. Generativní modely jsou navrženy tak, aby se naučily strukturu dat tak, aby ji mohly efektivně napodobovat, nikoliv aby uměly pouze rozlišovat mezi jednotlivými třídami (v případě klasifikace) nebo pouze hledat funkci, která správně prokládá data (v případě regrese).

Jinými slovy:

model se nesnaží odpovědět na otázku „co to je“, ale spíše „jak tato data vypadají a jak je mohu znovu vytvořit“.

Díky tomu je schopen model generovat nové vzorky, které se podobají těm, na kterých byl natrénován. Rozdíl mezi klasickými a generativními přístupy lze shrnout tedy následovně:

- klasický diskriminační model: přiřazuje vstupu výstup (například třídu při klasifikaci nebo číselnou hodnotu při regresi),

- generativní model vytváří nové vzorky dat, které se podobají těm původním.



Obrázek : Rozdíl mezi diskriminačním a generativním AI modelem.

klasický AI model
kočka
generativní AI model
kočka

Dalším důležitým důvodem vzniku generativní umělé inteligence je schopnost pracovat s nedostatkem dat. Generativní modely mohou sloužit například k vytváření nových datových vzorků, které následně pomáhají zlepšit výkon jiných modelů tím, že doplňují a rozšiřují původní trénovací data. Tento přístup se označuje jako „data augmentation“.

V současnosti stojí generativní umělá inteligence za řadou moderních technologií, jako jsou jazykové modely pro generování textu, systémy pro tvorbu obrázků, nástroje pro syntézu hlasu aj. Díky těmto schopnostem se generativní AI stává jednou z nejrychleji se rozvíjejících oblastí umělé inteligence.

1.2 Jak generativní modely fungují

Abychom pochopili, jak generativní modely vytvářejí nový obsah, musíme si nejprve uvědomit jednu klíčovou myšlenku:

model se nesnaží data pouze rozpoznat, ale snaží se naučit, jak data vznikají

Jinými slovy, generativní model se učí, jak vypadá „typický“ vzorek dat, na kterých byl natrénován, a na základě této znalosti dokáže vytvářet nové vzorky, které se těm původním podobají.

Generování jako pravděpodobnostní proces

Základním principem generativních modelů je práce s pravděpodobnostmi. Model se během trénování učí, jak pravděpodobné je, že po určité části dat bude následovat další část.

Například u textu se model učí:

- jaké slovo pravděpodobně následuje po jiném slově,
- jaké věty dávají smysl,
- jaká struktura textu je typická.

Pokud například zadáme začátek věty:

„Kočka sedí na ...“,

model na základě naučených pravděpodobností odhadne, jaká slova mohou následovat, například:

1. „Kočka sedí na **stole**“,
2. „Kočka sedí na **zemi**“,
3. „Kočka sedí na **okně**“.

Model tedy nevybírá slovo náhodně, ale na základě toho, co se během trénování naučil. Pokud se například v trénovacích datech často vyskytovalo spojení „kočka sedí na stole“, bude slovo „stole“ považovat za pravděpodobnější než například „zemi“.

Podobně pokud model během trénování často viděl věty jako „hlavní město Francie je Paříž“, naučí se, že po frázi „hlavní město Francie je“ velmi často následuje slovo „Paříž“ – přiřadí mu tedy opět nejvyšší pravděpodobnost výskytu. Neznamená to, že by model „znal“ tento fakt jako člověk, ale spíše si zapamatoval, že v daném kontextu je toto slovo nejpravděpodobnější.

Postupné vytváření výstupu

Generování dat neprobíhá najednou, ale postupně – krok za krokem. Například u textu to znamená:

1. generativní model dostane vstup (například začátek věty),
2. model vygeneruje v závislosti na získaném vstupu další slovo,
3. toto nově vygenerované slovo přidá k původnímu vstupu a
4. následně dojde k opakování celého tohoto procesu tak, že model přidává neustále další a další slova k původnímu slovnímu spojení, a vytváří tak věty.

Výsledný text tak vzniká postupným skládáním jednotlivých částí.

Je důležité si uvědomit, že konkrétní způsob generování se liší v závislosti na typu dat. V této kapitole jsme si princip generování vysvětlili na příkladu textu, kde model postupně vytváří jednotlivá slova. U jiných typů dat, například u obrázků, probíhá generování technicky odlišným způsobem – model může například postupně upravovat celý obraz najednou.

Přesto lze i u těchto přístupů zachovat stejnou základní intuici: výstup nevzniká jedním krokem, ale postupně se zpřesňuje, až odpovídá vzorům, které se model naučil během trénování.

Role náhodnosti při generování obsahu

Důležitou vlastností generativních modelů je, že jejich výstup není vždy stejný. I při stejném vstupu může model vytvořit více různých variant, což je výhodné, protože se tak model neopakuje a vypadá, jako kdyby více přemýšlel nad svým vygenerovaným výstupem. To je způsobeno tím, že model nevybírá vždy tu nejpravděpodobnější možnost ze všech výstupů, ale často „zvažuje“ více možností (možností s nejvyššími pravděpodobnostmi výskytu, například právě „stole“, „zemi“ či „okně“) a vybírá mezi nimi s určitou mírou náhodnosti tak, aby:

- výstupy nebyly neustále stejné a aby
- mohl být model kreativní.

Zároveň je však potřeba si dát pozor, protože někdy může model kvůli této vlastnosti generovat méně přesné, nebo dokonce naprosto nesmyslné výsledky – například třeba, že $88 + 22$ se rovná 100.

1.3 Základní typy generativních modelů

Generativní umělá inteligence není tvořena jedním konkrétním typem modelu. V praxi existuje několik různých přístupů, které se liší tím, jakým způsobem se model učí a jak následně vytváří nová data. Abychom těmto modelům lépe porozuměli, je důležité si připomenout, že všechny generativní modely mají jeden společný cíl: naučit se, jak data „vypadají“, a následně být schopny vytvářet nové vzorky, které se těm původním podobají.

Co se model ve skutečnosti učí?

Na rozdíl od klasických modelů, které se učí přiřazovat vstupům výstupy, se generativní modely snaží naučit celkovou strukturu dat. Modely se snaží odpovědět na otázku: „Jaká data jsou v tomto světě typická a jak vypadají?“. Model si nevytváří „databázi příkladů“, vytváří si spíše jakýsi vnitřní popis toho, jak data obecně vypadají. Například u obrázků obličejů se model neučí konkrétní osoby, ale spíše obecné vlastnosti jako:

- typické uspořádání očí, nosu a úst,
- jejich vzájemnou polohu nebo
- běžné proporce obličeje.

U textu se model zase učí:

- jak na sebe navazují slova,
- jaké větné struktury jsou běžné a
- jaký styl je typický pro daný typ textu.

Tento „vnitřní popis“ tedy není uložen jako seznam pravidel, ale jako soubor naučených vzorů, které model využívá při generování nových dat.

Generování není kopírování

Je také důležité si uvědomit, že generativní model nevytváří výstupy prostým kopírováním trénovacích dat, místo toho:

- kombinuje naučené vzory,
- vytváří nové kombinace a
- generuje data, která se podobají trénovacím datům, ale nejsou jejich přesnou kopií.

Například model, který byl trénován na obrázcích obličejů, nevytvoří kopii konkrétního člověka, ale nový obličej, který odpovídá naučeným vzorům.

Latentní prostor – „mapa dat“

U mnoha generativních modelů se objevuje důležitý pojem – tzv. latentní prostor. Latentní prostor si můžeme představit jako zjednodušenou mapu, ve které jsou data uspořádána podle podobnosti:

- podobné obrázky jsou blízko sebe,
- odlišné obrázky jsou dál od sebe.

Model se během trénování naučí tuto mapu vytvořit. Při generování obsahu pak:

1. generativní model vybere bod v tomto latentním prostoru („mapě“) a
2. převede jej zpět do podoby reálných dat (například obrázku nebo textu).

Díky tomu může model vytvářet nové kombinace, které dávají smysl, i když se přesně nevyskytovaly v trénovacích datech.

Tento princip se nevztahuje pouze na obrazová data. U textu model pracuje s reprezentacemi slov nebo jejich částí (tzv. „embeddings“), které vytváří určitý „reprezentační prostor“, ve kterém jsou slova uspořádána podle podobnosti významu. Například slova jako „pes“ a „kočka“ se v těchto reprezentacích nacházejí blízko vedle sebe, zatímco slovo „auto“ bude vzdálenější.

Základní přístupy ke generování

Různé generativní modely se liší především tím, jakým způsobem nový obsah vytvářejí. V následujících částech si ukážeme a základně popíšeme tři základní přístupy.

1. Generativní adversariální sítě („Generative adversarial network“, GAN)

GAN modely představují odlišný typ generativních modelů, který byl velmi významný zejména v oblasti generování obrazových dat. Základní myšlenka GAN je založena na interakci dvou modelů, které se během trénování vzájemně „tlačí“ ke zlepšení.

GAN se tedy skládá ze dvou hlavních částí:

- generátor a
- diskriminátor.

Generátor

Generátor vytváří nová data. Na vstupu obvykle dostane náhodný vektor, který lze chápat jako náhodně zvolený bod v latentním prostoru. Na základě tohoto vstupu se snaží vytvořit například nový obrázek obličeje, zvířete nebo jiného objektu.

Diskriminátor

Diskriminátor dostává na vstupu data a jeho úkolem je rozhodnout, zda jsou:

- skutečná, tedy pocházejí z trénovacích dat,
- nebo uměle vytvořená generátorem.

Diskriminátor je tedy v jistém smyslu „kontrolor“, který se snaží odhalit, které falešné výstupy z předchozí větve, tedy posoudit, jestli jeho vstupy (výstupy z předchozí větve) jsou reálná data z trénovacího datasetu, nebo se jedná o synteticky vygenerovaná data generátorem.

Trénování generativních adversariálních sítí lze chápat jako soutěž dvou modelů:

1. generátor vytvoří nový vzorek dat,
2. diskriminátor posoudí, zda je vytvořený vzorek z kroku 1 skutečný, nebo falešný,
3. pokud diskriminátor falešný vzorek odhalí, generátor dostane signál, že musí svůj výstup zlepšit,
4. pokud je diskriminátor ve svém úsudku zmaten, učí se zase on, aby byl přesnější.

Oba modely se tak postupně vzájemně zlepšují tak, že:

- generátor se učí vytvářet stále realističtější data,
- diskriminátor se učí stále lépe rozpoznávat rozdíl mezi reálnými a vygenerovanými daty.

Tento mechanismus je velmi zajímavý, protože generátor se neučí přímo z instrukce „takto má vypadat správný obrázek“, ale z toho, že se snaží vytvářet vstupy, které projdou „kontrolou“ diskriminátoru.

Po dokončení trénování se diskriminátor obvykle již nepoužívá a zůstává jen generátor. Ten dostane náhodný vstup a na jeho základě vytvoří nový vzorek dat. Díky tomu je možné generovat například:

- nové lidské obličej,
- syntetické obrazové vzorky nebo
- jiné datové instance, které vypadají realisticky.

Výhody	Nevýhody
GAN modely byly velmi úspěšné zejména proto, že dokázaly generovat ostré, vizuálně přesvědčivé a detailní obrazy. Byl využívány například pro: <ul style="list-style-type: none">• generování obličejů,• úpravy fotografií,• zvyšování kvality obrázků,• převod jednoho stylu obrazu na jiný apod.	GAN modely však mají i významné nevýhody: <ul style="list-style-type: none">• jejich trénování bývá nestabilní,• je obtížné udržet rovnováhu mezi generátorem a diskriminátorem,• může docházet k jevu známému jako „mode collapse“, kdy generátor začne vytvářet stále velmi podobné výstupy a ztratí „rozmanitost“. To znamená, že model sice může generovat realistická data, ale ne dostatečně různorodá.

2. Difuzní modely („Diffusion models“)

Druhým velmi populárním přístupem, který v současnosti dominuje generování obrazových dat, jsou difuzní modely. Jejich základní princip je poměrně netradiční – pracují s tzv. šumem, což si můžeme zjednodušeně představit jako náhodné zrnění, které známe ze starých televizorů bez signálu. Model se učí tak, že tento šum do obrázku uměle přidává a následně se jej učí krok za krokem odstraňovat. Celý proces trénování se skládá ze dvou hlavních fází:

1. Fáze znehodnocení (přidávání šumu): model dostane na vstup kvalitní, ostrou fotografii z trénovacích dat (například fotku kočky). Poté k ní v mnoha drobných krocích přidává náhodný šum. Obrázek se postupně rozmazává a zrní, až se z něj stane naprosto nerozpoznatelná změť pixelů.
2. Fáze obnovy (odstraňování šumu): následně se model učí tento destrukční proces obrátit. Snaží se přijít na to, jak z tohoto úplného chaosu krok za krokem šum odstranit a získat zpět původní fotku kočky. Zjednodušeně si to tedy můžeme představit tak, že se tímto způsobem model učí, jak vypadají hrany, barvy a přirozené struktury objektů.

Do celého procesu však vstupuje ještě jeden klíčový prvek – naše instrukce. Abychom totiž difuznímu modelu řekli, jaký obrázek má vytvořit, zadáváme mu textový popis (často se pro něj používá anglický výraz „prompt“). Jak ale obrazový model vůbec pozná, co po něm v tomto textu vlastně chceme?

Aby difuzní model dokázal propojit obraz s textem, využívá jakéhosi „překladače“. Tento systém chytře využívá koncept takzvaného latentního prostoru (mapy dat), který jsme si už vysvětlili. Překladatel funguje tak, že texty i obrázky převádí na souřadnice v jedné a téže společné mapě. Během svého vlastního tréninku na milionech párů textu a fotografií se naučil, že slovo „kočka“ a vizuální podoba kočky musí na této matematické mapě ležet těsně vedle sebe.

Když tedy napíšete své zadání, překladatel ho přemění na tyto konkrétní souřadnice. Během fáze odstraňování šumu pak difuzní model krok za krokem upravuje vznikající obraz tak, aby se jeho vizuální vlastnosti na oné společné mapě co nejvíce přibližovaly souřadnicím našeho textu.

Při samotném generování nového obsahu pak model na základě těchto znalostí postupuje takto:

- a) na vstupu dostane plátno plné náhodného zrnění (šumu),
- b) uživatel napíše své zadání („prompt“) a „překladatel“ ho přemění na konkrétní souřadnice ve společném latentním prostoru (mapě dat),
- c) model začne aplikovat to, co se naučil ve fázi obnovy, a v sérii mnoha desítek kroků zrnění z plátna postupně vyčišťuje,
- d) během tohoto čištění krok za krokem upravuje vznikající obraz tak, aby se jeho vizuální vlastnosti na oné společné mapě co nejvíce přibližovaly souřadnicím textu uživatele,
- e) proces se opakuje, dokud z původního chaosu (šumu) nevystoupí čistý a realistický obrázek.

Výhody	Nevýhody
<p>Difuzní modely jsou v současnosti velmi oblíbené zejména proto, že dokážou velmi přesně reagovat na textové zadání. Jsou využívány například pro:</p> <ul style="list-style-type: none"> • generování detailních a realistických obrázků, • tvorbu uměleckých děl, • přesnou vizualizaci složitých scénářů aj. <p>Oproti GAN modelům je navíc proces jejich trénování stabilnější a model tolik neztrácí na rozmanitosti výstupů.</p>	<p>I přesto mají difuzní modely svá omezení, a to především kvůli způsobu, jakým obrázky vytvářejí. Znamená to, že:</p> <ul style="list-style-type: none"> • tvorba obrázku probíhá postupně a v desítkách kroků, • proces je výpočetně velmi náročný a zdoluhavý, • modely mají často problém s generováním složitých a proměnlivých detailů (typicky třeba vytváří špatný počet prstů na ruce apod.).

3. Variační autoenkodéry („Variational autoencoders“, VAE)

Třetím základním typem generativních modelů jsou variační autoenkodéry, zkráceně VAE. Abychom pochopili jejich fungování, musíme si znovu vzpomenout na pojem „latentní prostor“, který jsme si v textu vysvětlili jako zjednodušenou mapu dat. VAE pracují na principu chytré komprese a dekomprese dat. Nejde však o obyčejné bezztrátové zmenšení dat – jako když na počítači balíte soubory do formátu ZIP. Architektura modelů VAE obsahuje záměrně „úzké hrdlo“ – představte si, že musíte někomu po telefonu popsat detailní fotografii obličeje, abyste mu umožnili ji znovu nakreslit, ale máte povoleno použít jen pět slov, jste tak donuceni ignorovat drobnosti (jako je například přesné rozložení stínů nebo drobná vráska) a soustředíte se jen na ty nejzákladnější strukturální rysy – například tvar obličeje, barvu očí, přítomnost brýlí apod.

Model při trénování čelí stejnému problému – obrovské množství pixelů musí zkomprimovat do velmi malého prostoru. Model je tak svou vlastní architekturou donucen osekát původní data a ponechat si v latentním prostoru pouze ty nejvýraznější vlastnosti, ze kterých lze obrázek následně znovu poskládat.

Architektura variačního autoenkodéru se skládá ze dvou hlavních, vzájemně propojených částí:

1. kodér („encoder“): tato část dostane na vstup složitá data, například ostrou fotografii lidského obličeje. Jejím úkolem je tuto fotografii „zmáčknout“ a převést do velmi zjednodušeného popisu v latentním prostoru. Kodér záměrně „zahodí“ nepodstatné detaily a uloží si jen klíčové charakteristiky popsané pomocí pravděpodobností – například jaký je tvar obličeje, úhel natočení hlavy apod.
2. dekodér („decoder“): na druhé straně stojí dekodér, který dostane tento zjednodušený zhuštěný popis. Jeho úkolem je postupovat přesně opačně – z těchto několika základních vlastností se snaží původní obrázek opět co nejpresněji zrekonstruovat a „rozbalit“ zpět do původní plné velikosti a podoby.

Během trénování se obě části učí společně a neustále se zlepšují. Jakmile je ale model plně natrénovaný, kodér se ze systému obvykle odpojí a pro generování nových dat se používá už jen

samotný dekodér. Vzhledem k tomu, že základní VAE nepracuje s textovými „prompty“, tvorba nového obsahu probíhá následovně:

- a) model (dekodér) dostává na vstup čistě náhodně vybraný bod v latentním prostoru, čímž se vygeneruje náhodná sada základních vlastností,
- b) dekodér tyto vlastnosti začne dekódovat do konkrétní obrazové podoby,
- c) vytvoří tak zcela nový, unikátní vzorek, který dává smysl a podobá se trénovacím datům, ale nikdy se v nich přesně takto nevyskytoval.

K čemu to je v praxi dobré?

Možná si nyní říkáte, k čemu je vůbec takový model dobrý, když do něj nezasíláme text a on vytváří obrázky jen z náhodného bodu. To by byla pravda, pokud bychom model používali jen jako generátor náhodných obrázků. Skutečná síla VAE však spočívá v tom, jak dokonale uspořádaný tento latentní prostor je a že v něm můžeme cíleně „cestovat“. V praxi se dají tyto modely využít třemi hlavními způsoby:

1. Chytrá úprava existujících fotek: nezačínáme náhodným bodem, naopak vezmeme skutečnou fotografii člověka, kterou vložíme do kodéru, ten ji matematicky zanalyzuje a převede její složitý vzhled do souřadnic latentního prostoru. Protože je tento prostor logicky uspořádaný, programátoři si v něm mohou předem pomocí jiných dat spočítat, kterým matematickým směrem se mění konkrétní vlastnost (najdou například přesný směr pro „úsměv“ nebo „vyšší věk“). Tyto matematické směry pak v uživatelské aplikaci propojí s jednoduchými ovládacími prvky. Když vy jako uživatel pak potáhnete posuvníkem „úsměv“ doprava, aplikace napozadí vezme vaše zjištěné souřadnice a matematicky k nim „směr úsměvu“ přičte. Výsledek pak předá dekodéru a ten z nových souřadnic vygeneruje upravenou fotku, na které se člověk bude usmívat.
2. Plynulé přechody: vzhledem k tomu, že latentní prostor je spojitý matematický prostor, neexistují v něm žádná „hluchá místa“. Pokud kodérem zpracujeme fotku kočky (získáme souřadnice A) a fotku psa (získáme souřadnice B), můžeme mezi těmito dvěma body matematicky vypočítat libovolné množství přesných mezikroků, které leží na spojnici těchto bodů. Když tyto vypočítané mezikroky postupně jeden po druhém předáme dekodéru ke zpracování, ten z nich zrekonstruuje sérii obrázků. Výsledkem může být dokonale plynulá animace, na které se vizuální rysy kočky krok za krokem plynule transformují do podoby psa.
3. Podmíněné generování: u základního VAE dekodér vytváří výstupy na základě čistě náhodných souřadnic z celé mapy (latentního prostoru), takže předem z logiky věci vlastně nevíme, jaký objekt přesně vznikne. V pokročilejších verzích (tzv. podmíněné VAE) se proto při trénování k obrázkům přidávají rovnou i informační štítky, například kategorie „auto“ nebo „motorka“. Model se tak naučí organizovat latentní prostor do jasně vymezených shluků na základě těchto kategorií. Při následném generování nového obsahu pak můžeme omezit náhodnost – vynutíme si, aby byly výchozí souřadnice vybrány pouze v tom matematickém rozmezí, které odpovídá požadovanému štítku. Dekodér následně z těchto souřadnic zrekonstruuje obraz, čímž získáme kontrolu nad tím, jaký typ objektu bude vytvořen.

Výhody	Nevýhody
<p>Hlavní předností variačních autoenkodérů je matematicky uspořádaný latentní prostor, který umožňuje přesnou manipulaci s daty, jsou ideální pro:</p> <ul style="list-style-type: none"> • plynulé úpravy existujících dat (například změna výrazů tváře), • generování mezikroků mezi dvěma různými vzorky, • porozumění vnitřní struktuře složitých datových sad. 	<p>Způsob, jakým VAE komprimují data přes „úzké hrdlo“, s sebou nese nevyhnutelnou ztrátu informací, která se může projevat následovně:</p> <ul style="list-style-type: none"> • výstupy (zejména obrázky) mohou být ve srovnání s modely GAN nebo difuzními modely mírně rozmazané, • výstupy nemusí dosahovat vysoké úrovně ostrosti a detailů, • model má tendenci detaily průměrovat, místo aby je vykresloval přesně.

4. Architektura Transformer („Transformers“)

Čtvrtým a v současnosti technologicky nejvyspělejším přístupem, je architektura Transformer. Zatímco předchozí modely jako GAN nebo VAE se specializovaly primárně na obraz, Transformer způsobil revoluci v matematickém zpracování jazyka a jiných sekvenčních dat. Právě tato architektura stojí za modely jako ChatGPT (kde „T“ znamená právě Transformer) a umožňuje AI generovat vysoce koherentní texty, které přesně odpovídají zadanému kontextu. Fungování Transformeru stojí na třech pilířích:

a) Mechanismus pozornosti (tzv. „self-attention“)

Klíčovým prvkem je schopnost modelu vyhodnocovat vztahy mezi všemi částmi vstupních dat současně. Na rozdíl od starších systémů, které zpracovávaly data postupně, Transformer využívá tzv. „mechanismus pozornosti“, anglicky „self-attention“. Ten funguje tak, že pro každé slovo v textu model vytvoří tři různé vektory (číselné reprezentace):

1. dotaz (hledá souvislosti, anglicky „Query“),
2. klíč (nabízí kontext, anglicky „Key“) a
3. hodnota (obsahuje informace, anglicky „Value“).

Tyto vektory vznikají vynásobením vektorové reprezentace původního slova s třemi různými maticemi vah, které se model naučil během trénování.

Představte si větu:

„Jeřáb se tyčil nad staveništěm, protože bylo potřeba zvednout těžké nosníky.“

Při zpracování slova „jeřáb“ probíhá následující matematický proces:

1. Skalární součin (tzv. „matching“): model vezme vektor Query (dotaz) slova „jeřáb“ a vypočítá jeho skalární součin s vektory Key (klíče) všech ostatních slov ve větě. Výsledkem je sada čísel (skóre), která udávají míru matematické shody mezi těmito slovy.
2. Určení vah (funkce „Softmax“): skóre vzniklá v bodě 1 jsou následně normalizována pomocí funkce „Softmax“, čímž vzniknou hodnoty těchto vah v rozmezí 0 až 1. V našem případě získá vysokou váhu shoda mezi dotazem „jeřábu“ a klíči slov „staveniště“,

„zvednout“ a „nosníky“. Ostatní slova (například „nad“, „protože“) obdrží váhu blízkou nule.

3. Vážený součet (tzv. „aggregation“): v posledním kroku model vezme vektory Value (hodnot) všech slov ve větě a vynásobí je těmito vypočítanými váhami. Tím vznikne nový, výsledný vektor pro slovo „jeřáb“.

Výsledkem není statické slovo, ale kontextuální reprezentace. Můžeme si to velmi zjednodušeně představit tak, že do sebe během této operace vektor slova „jeřáb“ vložil relevantní číselné informace z okolních slov – staveníště, nosníky apod. Tím se jeho pozice v latentním prostoru matematicky posunula do oblasti, která odpovídá stavební technice – model tak v tomto konkrétním kontextu s jistotou určí, že se nejedná o ptáka ani o strom. Celý tento proces probíhá pro všechna slova ve větě paralelně v rámci jedné operace, což umožňuje modelu okamžitě zachytit komplexní strukturu souvislostí v celém textu.

b) Kontextuální vnoření

Transformer nepracuje se slovy jako s řetězci písmen, ale s jejich matematickou hodnotou prostoru. Tento proces má dvě klíčové fáze:

- Vektory významu („embeddings“): každé slovo je převedeno na seznam stovek až tisíců čísel (vektor). Tato čísla určují pozici slova v obrovském vícerozměrném prostoru. Slova s podobným významem (např. „pes“ a „vlk“) končí v tomto prostoru blízko vedle sebe. Revoluční vlastností Transformeru je, že tyto hodnoty nejsou statické. Díky předchozímu kroku pozornosti se souřadnice slova „jeřáb“ v reálném čase přepisují podle toho, zda se v jeho blízkosti nachází slovo „staveníště“ nebo „hnízd“o“. Model tak pracuje s dynamickým významem, který se neustále přizpůsobuje okolí.
- Poziční kódování („positional encoding“): protože Transformer zpracovává všechna slova ve větě najednou (paralelně), přirozeně by ztratil informaci o tom, které slovo bylo první a které poslední. Věty „Pes kousl člověka“ a „Člověk kousl psa“ by pro něj bez pozíční informace vypadaly identicky, protože obsahují stejnou sadu vektorů. Model proto ke každému vektoru slova přičte speciální matematický signál, který nese informaci o jeho přesném pořadí ve větě. Tím je zajištěno, že se význam slova neodvíjí jen od jeho sousedů, ale i od jeho pozice v celkové struktuře.

c) Vícevrstvé zpracování („stacked layers“)

Architektura Transformeru není tvořena jedním výpočtem, ale celým systémem identických bloků (vrstev) naskládaných na sebe. Data procházejí těmito patry postupně a v každém z nich se analýza prohlubuje:

- Hierarchie zpracování: při průchodu jednotlivými vrstvami dochází k postupné transformaci a propojování informací. Zatímco první vrstvy zpracovávají data v jejich surové podobě (tak, jak přišla na vstup), každá další vrstva přidává další úroveň statistických souvislostí. Výsledkem tohoto řetězce je, že se v datech postupně generují stále komplexnější matematické vzorce, které odrážejí hlubší strukturu a logiku zpracovávaného textu.

- Residuální propojení: aby se v tomto dlouhém procesu neztratila původní informace ze zadání („promptu“), využívá Transformer tzv. „zkratky“ (reziduální spoje). Ty umožňují, aby se data z dřívějších fází výpočtu promítla přímo do těch pozdějších. Tím je zajištěno, že výsledný vektor na konci výpočtu stále věrně odpovídá původnímu zadání, přestože prošel stovkami miliard matematických operací.

1.4 Multimodalita: Propojení textu, obrazu a zvuku

V předchozích částech jsme si představili modely, které se specializují na jeden konkrétní typ dat (tzv. modality). Skutečný průlom v použitelnosti generativní AI však nastal v momentě, kdy se tyto izolované systémy podařilo propojit. Tento přístup nazýváme multimodalita.

Princip společného porozumění

Základem multimodality je vytvoření společného latentního prostoru, o kterém jsme mluvili už u konkrétních modelů. U multimodálních systémů se však tento prostor trénuje tak, aby v něm slova i obrázky sdílely stejné souřadnice, pokud reprezentují stejný koncept.

- Párování dat: model je během tréninku vystaven milionům dvojic – například obrázku psa běžícího v trávě a textovému popisku „pes běží v trávě“.
- Sjednocení souřadnic: matematický algoritmus upravuje vnitřní parametry sítě tak, aby se číselný vektor (souřadnice) textu v latentním prostoru co nejvíce přiblížil vektoru odpovídajícího obrázku.
- Výsledek: model díky tomu nezpracovává pixely nebo písmena jako oddělené světy. Data jsou v latentním prostoru uložena tak, že souřadnice slova „jablko“ matematicky odpovídají oblasti, kde se nacházejí vizuální vlastnosti jablka, i když modelu není v danou chvíli žádný obrázek předložen.

Směry multimodálního generování

Díky tomuto sjednocení můžeme s AI komunikovat napříč různými formáty:

1. Text-to-Image (text na obraz): nejznámější proces, kdy uživatel zadá textový popis („prompt“) a model z odpovídajících souřadnic v latentním prostoru vygeneruje pomocí difuze nový obraz.
2. Image-to-Text (obraz na text): model dostane na vstup obrázek, zanalyzuje jeho souřadnice a Transformer následně vygeneruje textový popis toho, co se na obrázku nachází.
3. Text-to-Audio/Video: rozšíření principu, kde se k vizuálním a textovým datům přidává časová osa a zvuková stopa, což umožňuje generování mluveného slova, hudby nebo pohyblivého obrazu.

Proč je multimodalita důležitá?

Multimodalita odstraňuje bariéry mezi člověkem a strojem. Umožňuje nám komunikovat s AI přirozeně – tedy kombinací vjemů. Praktickým příkladem je situace, kdy vyfotíte rozbitou součástku a textem se zeptáte: „Jak toto opravím?“. Model propojí vizuální vjem (identifikace součástky) s textovými daty (technický manuál) a vygeneruje odpověď.

1.5 Limity a rizika generativní AI

Přestože generativní modely dosahují fascinujících výsledků, je nezbytné si uvědomit, že se stále jedná o systémy pracující na principu statistické pravděpodobnosti, nikoliv o inteligenci s vlastním vědomím nebo morálním kompasem. Právě z této podstaty pramení několik zásadních rizik.

Fenomén halucinací

Jedním z nejvýraznějších problémů jsou tzv. halucinace. K tomuto jevu dochází, když model vygeneruje informaci, která je fakticky chybná nebo vymyšlená, přestože působí velmi přesvědčivě.

- Příčina: tento jev je přímým důsledkem principu, na kterém modely fungují – tedy predikce nejpravděpodobnějšího následného prvku (slova či části slova).
- Mechanismus chyby: matematický výpočet, který model provádí, je zaměřen na udržení statistické a strukturální věrohodnosti textu. Model nemá vestavěný žádný samostatný proces, který by ověřoval pravdivost obsahu oproti realitě – model pouze pokračuje v řetězení slov tak, aby věta dávala smysl podle naučených vzorců.
- Příklad: pokud model dostane za úkol vygenerovat životopis historické osobnosti, může si vymyslet název knihy, kterou nikdy nenapsala. Udělá to proto, že v daném kontextu je výskyt názvu knihy statisticky vysoce pravděpodobný a model jej „poskládá“ z prvků, které se v souvislosti s danou osobou často vyskytují.
- Důsledek: halucinace jsou nebezpečné právě proto, že výstup může být gramaticky bezchybný a stylisticky profesionální, což v uživateli vyvolává falešný dojem odbornosti. Ověřování výstupů z nezávislých zdrojů je proto u generativní AI nezbytným krokem.

Zaujatost a zkreslení

Generativní modely jsou odrazem dat, na kterých byly trénovány. Pokud tato data obsahují například společenské předsudky, stereotypy nebo nerovnoměrné zastoupení určitých skupin, model tyto vzorce přebírá.

- Původ problému: model se neučí, co je „správné“, ale co je v datech „běžné“.
- Důsledek: při generování obrázků nebo textů může AI reprodukovat rasové, genderové nebo kulturní předsudky, protože takové spojení bylo v trénovacím datasetu statisticky dominantní.

Autorská práva a etika trénování

Proces učení generativních modelů vyžaduje obrovské množství dat, která jsou často čerpána z internetu. To přináší zásadní etické a právní otázky:

- Využití autorských děl: modely jsou často trénovány na knihách, článcích a uměleckých dílech bez přímého souhlasu autorů.
- Vlastnictví výstupu: právní systémy po celém světě teprve definují, zda může být dílo vytvořené strojem (byť na základě lidského zadání) chráněno autorským zákonem.

Bezpečnost a zneužití (Deepfake)

Schopnost modelů generovat vysoce realistický obsah (text, obraz i hlas) může být využita k neetickým účelům:

- Dezinformace: snadná tvorba falešných zpráv, které vypadají jako profesionální žurnalistika.
- Deepfakes: vytváření falešných videí nebo zvukových nahrávek reálných lidí, které mohou sloužit k podvodům nebo manipulaci s veřejným míněním.

Energetická náročnost

Provoz generativní AI není „zadarmo“. Jak jsme poposali u architektury Transformer, výpočty probíhají na obrovském množství specializovaného hardwaru, který spotřebovává značné množství elektrické energie. To představuje environmentální zátěž, který roste s každým složitějším modelem.

1.6 Praktické cvičení s řešením

V tomto příkladu si vyzkoušíme, jak v praxi funguje model VAE (Variální autoenkodér). Použijeme reálná data – tisíce ručně psaných číslic – dataset MNIST. Model se během krátkého trénovacího procesu naučí „stlačit“ každý obrázek číslice do sady pouhých 20 čísel (souřadnic) v tzv. latentním prostoru. Poté využijeme druhou část modelu k tomu, aby z úplně náhodných souřadnic „vykouzlil“ nové obrázky číslic, které nikdy neviděl. V závěru skriptu také uvidíte plynulý přechod mezi dvěma různými číslicemi, což dokazuje, že si model vytvořil logickou „mapu“ mezi jednotlivými tvary.

```
import torch
import torch.nn as nn
import torch.optim as optim
from torchvision import datasets, transforms
from torch.utils.data import DataLoader
import matplotlib.pyplot as plt

# 1. NASTAVENÍ A DATA
# Automatické stažení datasetu MNIST
transform = transforms.Compose([transforms.ToTensor()])
train_dataset = datasets.MNIST(root='./data', train=True, transform=transform, download=True)
train_loader = DataLoader(train_dataset, batch_size=64, shuffle=True)

# 2. DEFINICE ARCHITEKTURY VAE
class VAE(nn.Module):
    def __init__(self):
        super(VAE, self).__init__()
        # Kodér: 784 pixelů -> 400 skrytých prvků -> 20 souřadnic
        self.fc1 = nn.Linear(784, 400)
        self.fc_mu = nn.Linear(400, 20)
        self.fc_logvar = nn.Linear(400, 20)
        # Dekodér: 20 souřadnic -> 400 skrytých prvků -> 784 pixelů (rekonstrukce)
        self.fc3 = nn.Linear(20, 400)
        self.fc4 = nn.Linear(400, 784)

    def encode(self, x):
        h = torch.relu(self.fc1(x))
        return self.fc_mu(h), self.fc_logvar(h)

    def reparameterize(self, mu, logvar):
        std = torch.exp(0.5 * logvar)
        eps = torch.randn_like(std) # Prvek náhody pro variabilitu výstupu
        return mu + eps * std

    def decode(self, z):
        h = torch.relu(self.fc3(z))
        return torch.sigmoid(self.fc4(h))

    def forward(self, x):
        mu, logvar = self.encode(x.view(-1, 784))
        z = self.reparameterize(mu, logvar)
        return self.decode(z), mu, logvar

model = VAE()
optimizer = optim.Adam(model.parameters(), lr=1e-3)
```

```

# 3. HLAVNÍ TRÉNOVACÍ SMYČKA
print("Trénovací proces začíná ...")
model.train()
for epoch in range(1, 4):
    celkova_loss = 0
    for batch_idx, (data, _) in enumerate(train_loader):
        optimizer.zero_grad()
        rekonstrukce, mu, logvar = model(data)

        # Výpočet chyby
        BCE = nn.functional.binary_cross_entropy(rekonstrukce, data.view(-1, 784), reduction='sum')
        KLD = -0.5 * torch.sum(1 + logvar - mu.pow(2) - logvar.exp())
        loss = BCE + KLD

    loss.backward()
    celkova_loss += loss.item()
    optimizer.step()

print(f"Epocha {epoch} dokončena. Průměrná chyba: {celkova_loss / len(train_loader.dataset):.2f}")

# 4. VIZUALIZACE VÝSLEDKŮ
model.eval()
with torch.no_grad():
    # A) Generování náhodných vzorků
    sample_latent = torch.randn(10, 20)
    vygenerovano = model.decode(sample_latent).view(-1, 28, 28)

    plt.figure(figsize=(15, 4))
    for i in range(10):
        plt.subplot(2, 10, i + 1)
        plt.imshow(vygenerovano[i], cmap='gray')
        plt.axis('off')
        if i == 0: plt.ylabel("Náhodné")

    # B) Interpolace: Plynulý přechod mezi dvěma "výstupy"
    start = torch.randn(1, 20)
    konec = torch.randn(1, 20)
    for i in range(10):
        vaha = i / 9
        bod = (1 - vaha) * start + vaha * konec
        prechod = model.decode(bod).view(28, 28)

        plt.subplot(2, 10, i + 11)
        plt.imshow(prechod, cmap='gray')
        plt.axis('off')
        if i == 0: plt.ylabel("Přechod")

plt.tight_layout()
plt.show()

```