FEYNMANOVA METODA · V JEDNÉ HODINĚ

Jak se učí *stroje*

Od „co znamená, že se stroj učí" až k transformerům a generativním modelům — cesta od prvního principu pro zvědavého člověka, který nemá za sebou žádnou matematickou přípravu.

ÚROVEŇ	Začátečník bez předchozí znalosti — jen zájem.
ROZSAH	Čtrnáct kapitol, přibližně jedna hodina pozorného čtení.
STYL	Postupné budování pojmů; matematika ve vyznačených blocích lze přeskočit.
CO SI ODNESEŠ	Vhled do toho, co se za posledních deset let stalo a kam to spěje.

OBSAH

Co tě čeká

Každá kapitola staví na předchozí. Pokud někde uvízneš, vrať se o jednu zpět — pojem, který chybí, byl pravděpodobně postaven předtím.

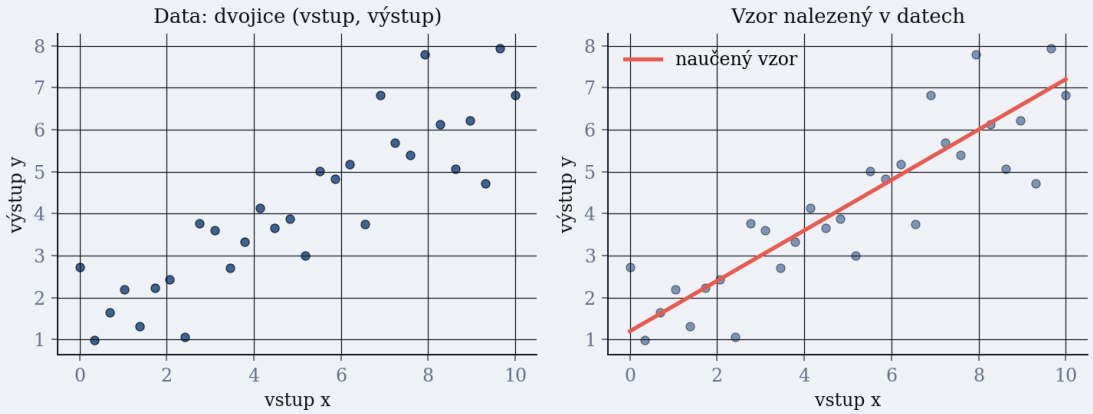
KAPITOLA 01

Co znamená, že se stroj učí

Posun od „řekni počítači, co dělat" k „ukaž mu příklady a nech ho najít vzor"

Než se ponoříme do algoritmů, položme si nejprve obyčejnou otázku: **co vlastně znamená, že se stroj učí?** Když napíšeš program, který sčítá čísla, zadáš mu pravidlo a on ho vykonává. Pravidlo jsi formuloval ty. Stroj nedělá nic jiného než otrocky následuje instrukce.

Strojové učení obrací tento vztah. Místo abys formuloval pravidlo, ukážeš stroji **příklady** — dvojice „takhle to vypadá vstup, takovou má mít odpověď" — a stroj sám hledá vzor, který tyto dvojice spojuje. Vzor není napsaný v jeho kódu; vzniká během učení a žije v podobě **parametrů**, čísel, která se postupně přizpůsobují, dokud model na příklady neodpovídá rozumně.



Obr. 1 — Vlevo: data, dvojice (vstup, výstup). Vpravo: vzor, který v datech model našel. Pravidlo nikdo nezapsal — vzniklo z dat.

Tomuto přístupu se říká **učení z dat**. Klasický program a strojové učení nejsou protiklady — oba mají své místo. Pokud znáš pravidlo (sčítání, výpočet úroku), naprogramuj ho přímo. Pokud pravidlo neznáš, ale máš dost příkladů (rozpoznat kočku na fotce, předpovědět cenu nemovitosti), nech ho stroj objevit.

> **PROČ TO ZAČALO FUNGOVAT TEPRVE NEDÁVNO**
>
> *Většina algoritmů, o kterých si budeme povídat, byla vymyšlena už v sedmdesátých až devadesátých letech. Co se změnilo? Tři věci najednou: dostatek **dat** (internet, smartphone, kamery), levný **výpočetní výkon** (GPU původně pro hry) a několik **algoritmických triků**, které dovolily trénovat hlubší modely. Bez všech tří najednou by se nic nestalo.*

Stojí za to si hned na začátku ujasnit slovník. **Model** je matematický objekt — funkce s parametry, která bere vstup a dává výstup. **Učení** nebo **trénink** je proces, kterým se parametry modelu nastavují podle dat. **Predikce** nebo **inference** je použití naučeného modelu na nový vstup. Tyto tři fáze se v každé následující kapitole vrátí.

A poslední věc — strojové učení se obvykle rozděluje podle toho, **co o datech víme**. Pokud máme ke každému vstupu i správnou odpověď, mluvíme o **učení s učitelem** (angl. *supervised*). Pokud máme jen vstupy a hledáme strukturu, je to **učení bez učitele** (*unsupervised*). Třetí varianta je **zpětnovazební učení** (*reinforcement*), kde agent zkouší akce a dostává odměny; zmíníme jen okrajově.

NEJOBECNĚJŠÍ FORMULACE

```
najdi parametry  θ*  =  argmin  L(f(x; θ), y)
```

minimalizuj rozdíl mezi tím, co model říká, a tím, co je správně

V tomto vzorci je všechno. **x** je vstup, **y** je správná odpověď, **f(x; θ)** je model — funkce, která pro vstup x a parametry θ vyrobí předpověď. **L** je *loss function*, která říká, jak moc je předpověď špatně. **θ*** jsou nejlepší parametry. Celé strojové učení se točí kolem dvou otázek: jakou volit funkci f a jak najít to argmin.

> **MATEMATIKU MŮŽEŠ PŘESKAKOVAT**
>
> *Vzorce v tmavých blocích jsou pro toho, kdo chce vidět přesnou podobu věcí. Pokud z nich máš nepříjemný pocit, klidně je přeskakuj — **slovní popis pod nimi obsahuje totéž**. Pochopení textu na nich nezávisí.*

KAPITOLA 02

Supervised: učení s odpověďmi

Lineární regrese jako nejjednodušší případ — proložit data čárou

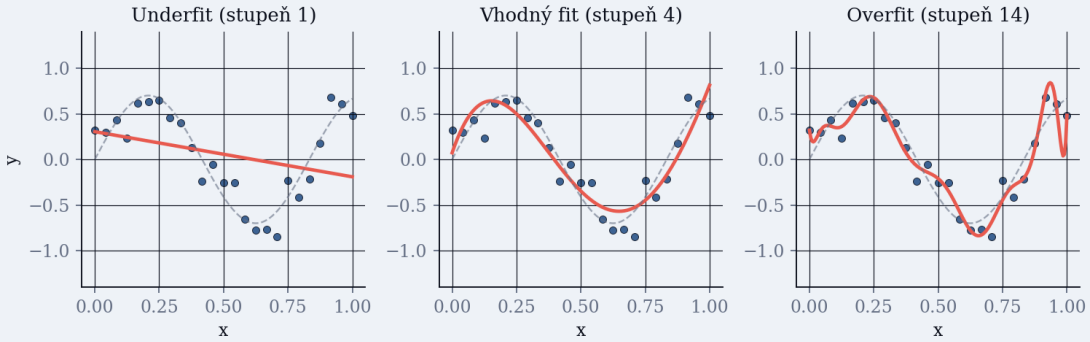
Začneme nejjednodušší možnou úlohou. Máš seznam dvojic — třeba (rozloha bytu, cena bytu) — a chceš odhadnout cenu pro byt, který v seznamu není. Když si data nakreslíš, vidíš, že body víceméně leží podél přímky. Stačí najít tu „nejlepší" přímku a pro novou rozlohu odečíst odpovídající cenu. Tomu se říká **lineární regrese** a jde o nejstarší a nejstravitelnější úlohu strojového učení.

LINEÁRNÍ MODEL

$$\hat{y} = w \cdot x + b$$

předpověď je vážená vstup plus posun; w a b jsou parametry

Co znamená „nejlepší přímka"? Definujeme si **chybu** — pro každý bod změříme, jak moc se liší skutečná hodnota y od toho, co předpovídá náš model ŷ. Tyto rozdíly umocníme (aby se kladné a záporné nezrušily) a sečteme. Výsledku se říká **střední kvadratická chyba** a my chceme parametry, které ji minimalizují.



Obr. 2 — Tři pokusy proložit stejná data. Vlevo příliš jednoduchá funkce „nestihne" zachytit tvar (underfitting), uprostřed sedne pěkně, vpravo je naopak příliš složitá a kopíruje šum — obtahuje i šum, ne jen vzor (overfitting).

Tyto tři obrázky jsou důležitější, než se na první pohled zdá. Ukazují **fundamentální kompromis** celého strojového učení. Když je model příliš jednoduchý, není schopen zachytit, co v datech je. Když je příliš složitý, místo vzoru se naučí náhodu. Mezi tím leží sladké místo — model dostatečně mocný, aby vystihl signál, ale ne natolik, aby memoroval šum.

Tady se objevuje **nejdůležitější trik** celého oboru: oddělit data, na kterých se učím, od dat, na kterých zkouším, jak dobře jsem se naučil. Té první sadě se říká **trénovací**, té druhé **testovací**. Pokud model na trénovací sadě excelluje, ale na testovací selhává, je přeučený — naučil se data nazpaměť, ne vzor.

KAPITOLA 03

Klasifikace a logistická regrese

Když odpovědí není číslo, ale kategorie

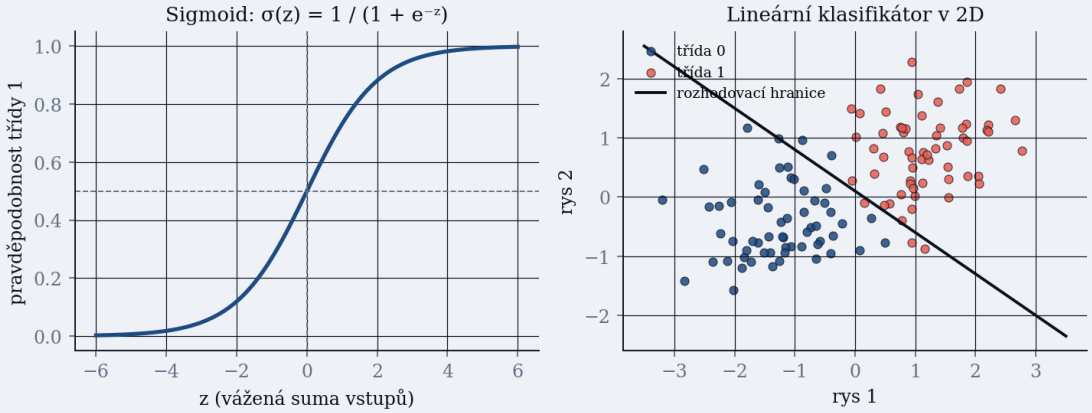
Regrese odpovídá na otázku „kolik". Klasifikace na otázku „co". E-mail je spam, nebo není? Tato fotka je pes, kočka, nebo kůň? Pacient má rakovinu, nebo nemá? Výstup není spojité číslo — je to jedna z konečně mnoha tříd.

Mohli bychom použít stejnou přímku jako v regresi a říct: „pokud je výstup větší než nula, říkáme třída 1, jinak třída 0." Funguje to, ale chtěli bychom víc — chtěli bychom **pravděpodobnost**, ne jen rozhodnutí. Místo „je to spam" radši „je to spam s pravděpodobností 0.87". K tomu potřebujeme funkci, která libovolné číslo zmáčkne do intervalu (0, 1). Tomu slouží **sigmoid**.

LOGISTICKÁ REGRESE

```
P(y=1 | x)  =  σ(w · x + b)          σ(z) = 1 / (1 + e^(-z))
```

sigmoid mačká vážený součet do intervalu (0, 1) — pravděpodobnost



Obr. 3 — Vlevo: sigmoidní křivka — z velkého kladného z dělá hodnoty blízké 1, z velkého záporného hodnoty blízké 0. Vpravo: dvě třídy bodů a přímka, která je co nejlépe odděluje. Bod nad přímkou patří modré třídě, pod ní červené.

Ačkoli „regrese" v názvu zní jako spojitá úloha, **logistická regrese** je nejpoužívanější klasifikátor. Trénuje se podobně jako lineární regrese — minimalizuje se chybová funkce, jen je trochu jiná (tzv. *cross-entropy*, která trestá sebevědomé špatné odpovědi více než nejisté). Pro několik tříd se sigmoid zobecňuje na **softmax**, která dává pravděpodobnostní rozdělení přes všechny kategorie.

PŘÍMKA NESTAČÍ VŠEMU

Lineární klasifikátor předpokládá, že třídy lze oddělit přímkou (v 3D rovinou, ve více dimenzích nadrovinou). V realitě to často nestačí — třídy jsou navzájem prokroucené. K tomu se hodí buď ručně navrhnout chytřejší rysy, anebo nechat to objevit neuronovou síť. K tomu se dostaneme.

KAPITOLA 04

Loss a gradientní sestup

Jak vlastně stroj „hledá" ty správné parametry

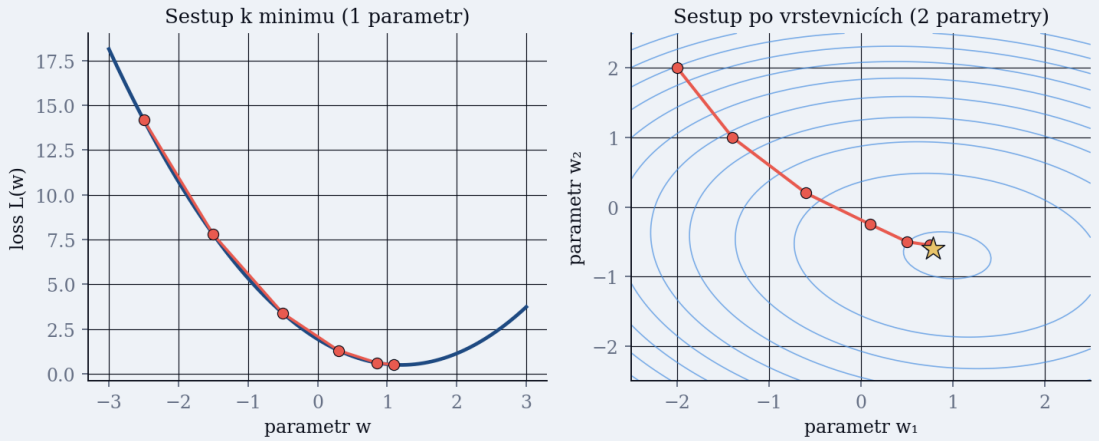
V kapitolách výše jsme říkali „minimalizujeme chybu", ale neřekli jsme, jak. U lineární regrese existuje vzorec, který minimum najde přímo. U složitějších modelů už ne — a přesto musíme nějak najít parametry, které dávají malou loss. Klíčový nápad se jmenuje **gradientní sestup** a je překvapivě jednoduchý.

Představ si krajinu, kde každé místo je jedna kombinace parametrů a **nadmořská výška** odpovídá hodnotě loss pro tyto parametry. Trénink je pak procházka touto krajinou, kde hledáš nejnižší údolí. Začneš na náhodném místě a v každém kroku se rozhlédneš, odkud terén nejvíc klesá, a uděláš krok tím směrem. Tomu „nejvíc klesá" odpovídá záporný **gradient** — vektor parciálních derivací, který říká, jak loss reaguje na malé změny každého parametru.

GRADIENTNÍ SESTUP

$$\theta_{nový} = \theta_{starý} - \eta \cdot \nabla L(\theta)$$

odeber kousek ve směru, kterým loss nejrychleji klesá



Obr. 4 — Vlevo: pohled v 1D, parabola loss s několika kroky. Vpravo: vrstevnice loss surfacu ve 2 parametrech; červená cesta je trajektorie sestupu, hvězda je minimum.

Symbol **η** (eta) je **learning rate** — délka kroku. Příliš velký krok tě přenese přes minimum, příliš malý znamená, že se k cíli plížíš věčně. Volba learning rate je jedna z hlavních věcí, které „ladíš", když trénuješ model.

Velký dataset má milion příkladů, a spočítat gradient ze všech najednou by trvalo věčnost. Proto se používá **stochastický gradientní sestup** (SGD): v každém kroku vezmeš malý *batch* (např. 64 příkladů) a spočítáš gradient jen z něj. Šum spíš pomáhá utéct z mělkých údolí.

> **PROČ TO VŮBEC FUNGUJE**
>
> *V loss krajinách reálných modelů je obrovské množství lokálních minim a sedlových bodů. Empiricky platí, že u dostatečně velkých sítí je **většina lokálních minim kvalitativně podobná tomu globálnímu**. Proč, plně rozumově neodvodíme — je to pozorování.*

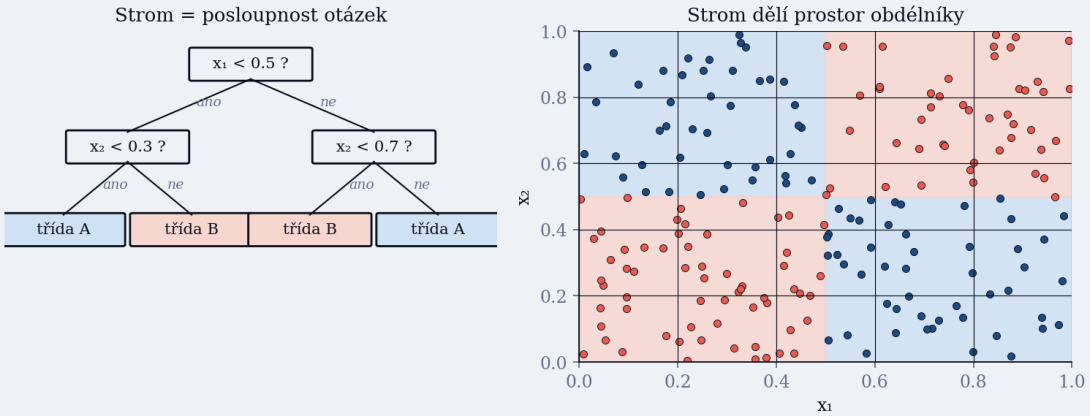
KAPITOLA 05

Stromy a náhodné lesy

Učení postavené na otázkách typu „je x větší než 0.5?"

Lineární modely jsou krásné, ale světu nestačí. Co když chceš model, který sám pochopí, že „když je věk pod 30 a výplata nad 60 tisíc, půjčku schválit" — kombinace pravidel, která spolu interagují? K tomu se hodí jiný přístup: **rozhodovací strom**.

Rozhodovací strom je posloupnost otázek typu „je *tento rys* větší než *nějaká hodnota*?". Začneš v kořenu, podle odpovědi se posuneš na levou nebo pravou větev, tam dostaneš další otázku, a tak dál — dokud nedojdeš k **listu**, kde je odpověď. Trénink stromu znamená najít, na které otázky se ptát a v jakém pořadí, aby se data co nejlépe oddělila.



Obr. 5 — Vlevo: schéma malého stromu. Cesta z kořene k listu je řetězec otázek. Vpravo: jak strom dělí 2D prostor — vždy na osově zarovnané obdélníky, protože každá otázka se týká jednoho rysu.

Stromy jsou populární ze dvou důvodů. Jsou **interpretovatelné** — můžeš se podívat, na základě jakých otázek model rozhodl, a porovnat to s lidským uvažováním. A umějí pracovat s **kategoriálními rysy** i smíšenými typy bez nutnosti složitého předzpracování. Mají ale problém — jsou křehké. Drobná změna v datech vede k úplně jinému stromu.

Tomu se dá zabránit chytře: místo jednoho stromu nauč jich **tisíc**, každý na náhodně vybrané podmnožině dat a rysů, a předpověď udělej **hlasováním**. Tomu se říká **náhodný les** (random forest). Jeden strom může chybit, ale tisíc různých stromů typicky chybí dohromady méně. Náhodné lesy a jejich příbuzní (gradient-boosted trees, XGBoost) jsou stále nejlepší volbou pro **tabulková data** — to znamená data v podobě excelové tabulky, kde řádky jsou pozorování a sloupce rysy.

> **NEURONOVÉ SÍTĚ NEJSOU VŠEMOCNÉ**
>
> *Pro tabulková data — typu zákaznická databáze, lékařské záznamy, finanční metriky — náhodné lesy a boostingové stromy **typicky předhánějí neuronové sítě**. Neuronové sítě dominují tam, kde mají hodně dat a kde rysy nejsou plochá tabulka, ale strukturovaná věc — obrázek, zvuk, text. Jako u většiny věcí: nástroj se vybírá podle úlohy.*

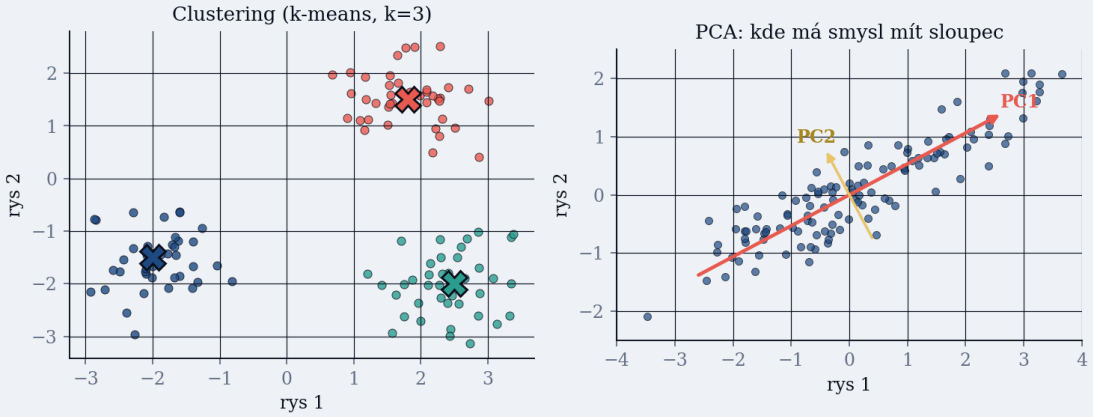
KAPITOLA 06

Unsupervised: učení bez odpovědí

Co dělat, když máme jen vstupy a žádné správné odpovědi

Až dosud jsme vždycky měli k dispozici „správnou odpověď" pro každý vstup. Často ale taková odpověď neexistuje — máš jen hromadu dat a rád bys v nich našel **strukturu**. Tomu slouží **unsupervised learning**. Dva nejtypičtější úkoly jsou shlukování a redukce dimenze.

Při **shlukování** chceš data rozdělit do skupin, které jsou si vnitřně podobné. Nejznámější algoritmus se jmenuje **k-means**: zvolíš počet clusterů *k*, umístíš jejich centra náhodně, přiřadíš každý bod nejbližšímu centru a posuneš centra do průměru bodů, které k nim patří. Opakuješ, dokud se centra přestanou hýbat. Z chaosu vznikne struktura.



Obr. 6 — Vlevo: tři skupiny bodů a jejich centroidy nalezené k-means. Vpravo: data, která vypadají 2D, ale skoro celá variance leží podél jedné osy. PCA tuto osu najde a říká, že stačí jeden sloupec místo dvou.

Druhý hlavní úkol je **redukce dimenze**. Když máš data se stovkou rysů, je v nich většinou hodně redundance — některé rysy spolu úzce souvisejí. **PCA** (principal component analysis) hledá nové osy, podle kterých data nejvíc variují. Z původních sta sloupců často stačí pár prvních nových os, abys měl skoro veškerou informaci. To je užitečné pro **vizualizaci** (data ze 100 dimenzí promítnout do 2 a nakreslit) a pro **kompresi**.

K-MEANS — JEDEN KROK

```
krok 1:  každému bodu přiřaď nejbližší centroid
```

krok 2: posuň centroid do průměru bodů, které k němu patří

Unsupervised přístupy jsou dnes víc než „samostatná disciplína". Jsou základem moderních **self-supervised** metod, kde model nedostává od člověka označená data, ale úkol si vytvoří sám — třeba „doplň chybějící slovo ve větě". Tak se trénují velké jazykové modely. Hranice mezi supervised a unsupervised je tedy v praxi mnohem méně ostrá, než zní z definice.

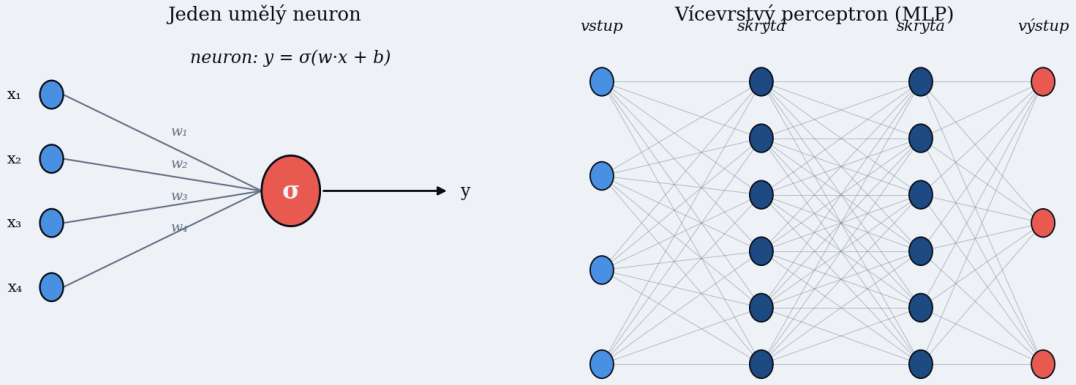
KAPITOLA 07

Neuron a vícevrstvý perceptron

Stavíme z lineárních modelů něco, co umí mnohem víc

Lineární modely (regrese, logistická regrese) mají jasnou mez — umí jen to, co se dá oddělit přímkou nebo nadrovinou. Co kdybychom jich pospojovali víc do série, a mezi ně dali nějakou **nelinearitu**? Vznikne **neuronová síť**. Nápad je starý přes půlstoletí, ale jeho síla se ukázala teprve v posledních dvou dekádách.

Základní stavební kámen je **neuron** — jednotka, která vezme několik vstupů, každý vynásobí vahou, sečte je dohromady, přičte posun (*bias*) a výsledek pošle přes nelineární funkci, tzv. **aktivační funkci**. Sigmoid známe; populárnější volbou v moderních sítích je ale **ReLU** — „rectified linear unit", což je prostě *max(0, z)*: záporné hodnoty se zařízne na nulu, kladné prochází.



Obr. 7 — Vlevo: jeden umělý neuron s váženými vstupy a aktivační funkcí. Vpravo: vícevrstvý perceptron — neurony seřazené do vrstev, každá vrstva propojená s další. Vstup vlevo, výstup vpravo, mezi tím skryté vrstvy.

JEDEN NEURON

$$y = \sigma(w_1 \cdot x_1 + w_2 \cdot x_2 + \dots + w_n \cdot x_n + b)$$

vážená suma vstupů, plus bias, projde aktivací

Jednoduchý fakt, který je ale překvapivě silný: **vícevrstvá neuronová síť s nelineární aktivací umí aproximovat libovolnou rozumnou funkci**, jen má-li dost neuronů. Tomu se říká *univerzální aproximační teorém* a říká, že MLP je principiálně velmi mocný nástroj. V praxi ale „dost neuronů" může znamenat astronomické množství, a navíc zbývá dořešit, jak parametry naučit. K tomu se dostaneme v další kapitole.

Architektura sítě — kolik vrstev, kolik neuronů na vrstvu, jaká aktivační funkce — je věc designu. Pro klasifikaci na 10 tříd typicky výstupní vrstva má 10 neuronů následovaných softmaxem (zobecnění sigmoidu). Pro regresi má jeden neuron bez aktivace. Skryté vrstvy mívají typicky stovky až tisíce neuronů.

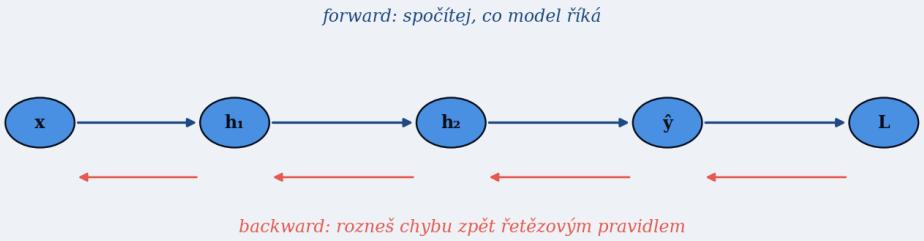
KAPITOLA 08

Backpropagation: jak se síť učí

Řetězové pravidlo z první derivace — důvod, proč deep learning vůbec funguje

Síť má desítky tisíc až miliardy parametrů. Loss je číslo na konci — vzdálenost mezi tím, co síť řekla, a správnou odpovědí. Otázka: jak změnit každý jeden parametr, aby loss klesla? Odpověď je **backpropagation** — algoritmus, který spočítá gradient celé sítě efektivně tím, že chyby šíří zpět vrstvu po vrstvě.

Obrázek si představ takhle. Když síť spočítá výstup, jde to **dopředu**: vstup se transformuje přes vrstvu 1, výstup vrstvy 1 se posílá do vrstvy 2, a tak dál až k loss. Jakmile známe loss, jdeme **zpět**: spočítáme, jak loss reaguje na změny ve výstupu poslední vrstvy. Pak, pomocí **řetězového pravidla** z derivací, spočítáme, jak loss reaguje na výstup předposlední vrstvy. A tak dál, vrstvu po vrstvě, až k samotným váhám.



Obr. 8 — Forward pass (modře): vstup se postupně transformuje až k loss. Backward pass (červeně): gradient loss se šíří proti směru — každá vrstva ho dostane a pomocí řetězového pravidla z něj vyrobí gradient pro předchozí vrstvu.

ŘETĚZOVÉ PRAVIDLO

$$\partial L/\partial w = \partial L/\partial y \cdot \partial y/\partial h \cdot \partial h/\partial w$$

gradient se rozkládá na součin lokálních gradientů — jako ozubená kola

Tak triviální to je. Geniální není matematika — řetězové pravidlo je obyčejná derivace. Geniální je **uvědomění**, že tahle struktura umožňuje velmi efektivní výpočet. Bez backpropagation by trénování velkých sítí trvalo nemožně dlouho. S ním zvládá moderní GPU spočítat gradient miliardového modelu za zlomek vteřiny.

> **BACKPROP NENÍ MAGIE**
>
> *Kdyby tě jednou pohltila myšlenka, že strojové učení je „jiný druh počítání", připomeň si, že backpropagation je obyčejná aplikace pravidla, které ses učil ve druhém ročníku gymnázia: derivace složené funkce je součinem derivací. Nic víc, nic míň. Jen aplikované obrovskou silou na obrovský počet parametrů.*

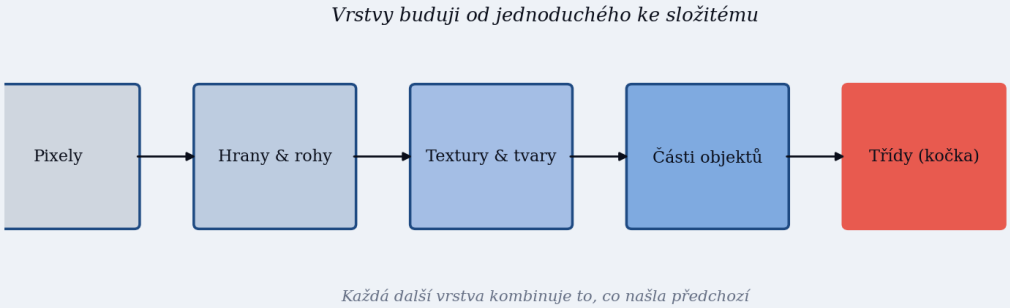
KAPITOLA 09

Hloubka a hierarchie reprezentací

Proč „víc vrstev" znamená „chytřejší model" — a proč to dlouho nefungovalo

Univerzální aproximační teorém říká, že stačí jedna skrytá vrstva. Proč tedy mluvíme o **hlubokém** učení a sítích s desítkami až stovkami vrstev? Protože jedna vrstva, aby uměla totéž, co deset, by potřebovala **exponenciálně více neuronů**. Hluboká síť je v praxi *parametricky úspornější* a lépe se učí strukturovaná data.

Důvod je intuitivní: realita je **hierarchická**. Rozpoznat kočku na fotce neznamená jeden krok od pixelů. Nejdřív se z pixelů sestaví **hrany** (ostré přechody jasu), z hran **tvary** (kruhy, špičky), z tvarů **části objektů** (oko, ucho, ocas) a teprve z částí **celý objekt**. Hluboká síť tuto hierarchii kopíruje — každá další vrstva pracuje s reprezentacemi, které jí předala předchozí.



Obr. 9 — Hierarchie reprezentací v hluboké síti pro vidění. Spodní vrstvy „vidí" jen pixely, střední se naučí abstraktnější rysy, horní rozpoznávají objekty. Tato struktura nevzniká návrhem — vzniká spontánně z dat.

Hluboké sítě měly dlouho jeden problém: **mizení gradientu**. Když gradient putuje řetězovým pravidlem skrz mnoho vrstev, násobí se mnoho čísel, a pokud jsou pod jedničkou, výsledek se v dolních vrstvách smršťí skoro k nule. Ty pak nedostanou žádný signál a neučí se. Trvalo do roku 2010, než se vyřešilo: lepší aktivační funkce (ReLU místo sigmoidu), lepší inicializace vah, batch normalizace, a — zásadně — **residuální spojení**, která posílají signál „zkratkou" mezi vzdálenými vrstvami.

> **CO JE PŘELOMOVÝ ROK**
>
> *Za zlom se obvykle bere rok 2012, kdy AlexNet — hluboká konvoluční síť — vyhrála soutěž ImageNet s tak velkým náskokem, že si akademie najednou musela přiznat, že hluboké učení není slepá ulička. Od té doby vývoj nezpomalil ani na den.*

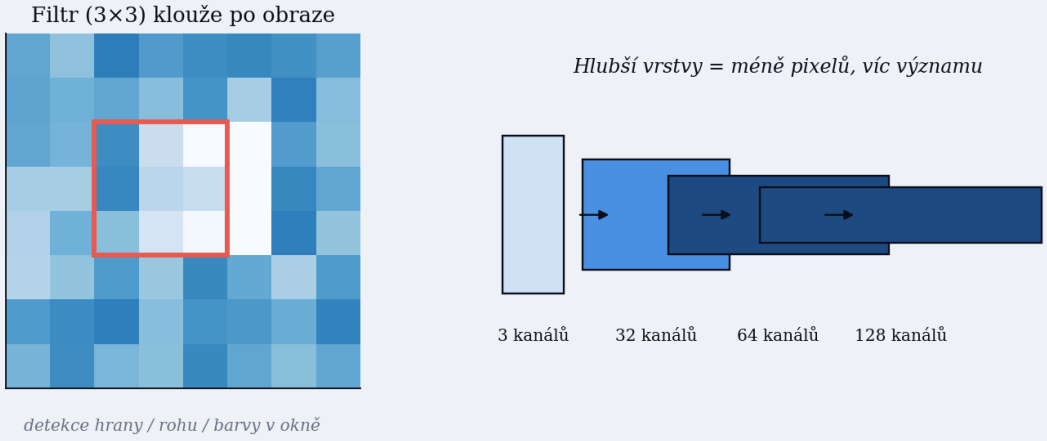
KAPITOLA 10

Konvoluční sítě: vidění

Architektura, která dala počítačům oči

Plně propojená síť (*fully connected*), o které jsme mluvili, má jeden velký problém pro obrazy. Obraz s rozlišením 200×200 v barvách má 120 000 čísel. Pokud první skrytá vrstva má 1000 neuronů, máš tam 120 milionů parametrů jen pro vstup. Navíc síť musí **znovu** objevit, že kočka v levém horním rohu je stejná věc jako kočka ve spodní pravé části. Marná práce.

Konvoluční síť (CNN) řeší obojí jednou myšlenkou: *filtr klouže po obraze*. Místo toho, aby každý neuron měl vlastní váhy pro každý pixel, definujeme malý filtr (třeba 3×3 nebo 5×5), který se aplikuje na celou plochu obrazu posuvným způsobem. Filtr má jen pár desítek parametrů, ale díky posouvání pokryje celou plochu.



Obr. 10 — Vlevo: 3×3 filtr klouže po obraze. Vpravo: hlubší vrstvy mají méně prostorového rozlišení, ale více kanálů — každý kanál je jeden naučený detektor (hrana, textura, …).

Co filtr **dělá**? Vrací silnou odezvu, když se v jeho okně vyskytne určitý vzor — třeba „svislá hrana" nebo „diagonála". Síť obvykle používá desítky filtrů paralelně, každý hledá jiný vzor. Výstup vrstvy je sada **feature map** — obrazů, kde každý pixel říká „tady jsem viděl vzor, který mě zajímá".

Mezi konvolučními vrstvami obvykle bývá **pooling** — operace, která zmenší rozlišení (typicky vezme z každého 2×2 čtverce maximum). Tím se síť stává *posunově invariantní* — kočka v rohu i uprostřed dává podobnou odezvu. CNN dominovaly počítačovému vidění od roku 2012 do roku 2020. Pak je v některých úlohách začaly dotahovat **vision transformery** — ale o transformerech až v Kapitole 12.

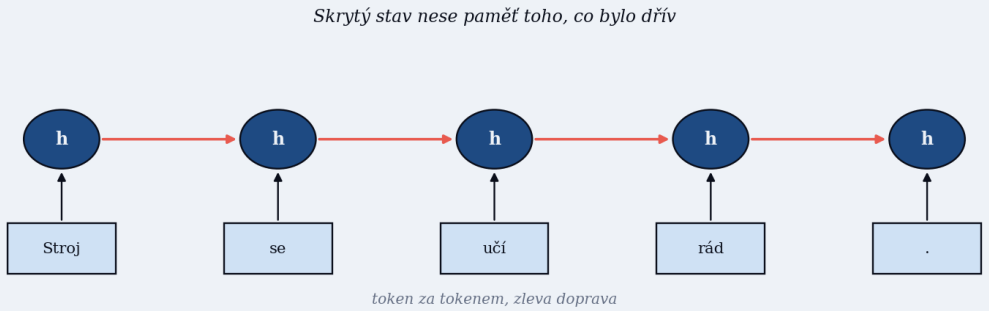
KAPITOLA 11

Rekurence a paměť: sekvenční data

Když má smysl pořadí — text, řeč, časové řady

Obrázek má danou velikost a jeho rozměr je předem známý. Věta nikoli — má jednou pět slov, jindy padesát. A navíc má **pořadí** — slovo na začátku ovlivňuje význam slova na konci. Standardní MLP ani CNN se s tím dobře neperou. Pro sekvence se historicky používaly **rekurentní neuronové sítě** (RNN).

Idea RNN: síť má **skrytý stav**, který v čase nese paměť toho, co viděla. V každém kroku dostane další token (slovo, znak, vzorek řeči), zkombinuje ho se starým skrytým stavem a vyrobí nový stav. Stav tedy postupně akumuluje informaci o celé sekvenci. Na konci buď vyrobí předpověď (klasifikace celé věty), nebo postupně generuje výstupní sekvenci.



Obr. 11 — Rekurentní síť „rozbalená" v čase. Pro každý token vznikne kopie sítě, a skrytý stav (modré uzly) cestuje zleva doprava — nese kontext.

Klasická RNN má ale problém s **dlouhým kontextem**. Když potřebuješ informaci z prvního slova věty u posledního, musí projít celým řetězcem aktivací — a gradient se cestou ztrácí (problém mizejícího gradientu z Kapitoly 9). Proto se vymyslely vylepšené architektury jako **LSTM** (Long Short-Term Memory) a **GRU** (Gated Recurrent Unit), které mají *brány* rozhodující, co si pamatovat a co zapomínat. Po roce 2015 se staly standardem pro strojový překlad, rozpoznávání řeči, autocomplete.

> **PROČ DNES MLUVÍME O RNN V MINULÉM ČASE**
>
> *V roce 2017 přišel Transformer — architektura, která zpracovává celou sekvenci najednou, paralelně, bez rekurence. V mnoha úlohách převálcoval LSTM a stal se základem GPT, BERT, a všech moderních jazykových modelů. RNN ale stále žijí — především v aplikacích, kde data **teče** v reálném čase a paměť potřebuješ sekvenčně.*

KAPITOLA 12

Attention a Transformer

Mechanismus, který v roce 2017 změnil svět umělé inteligence

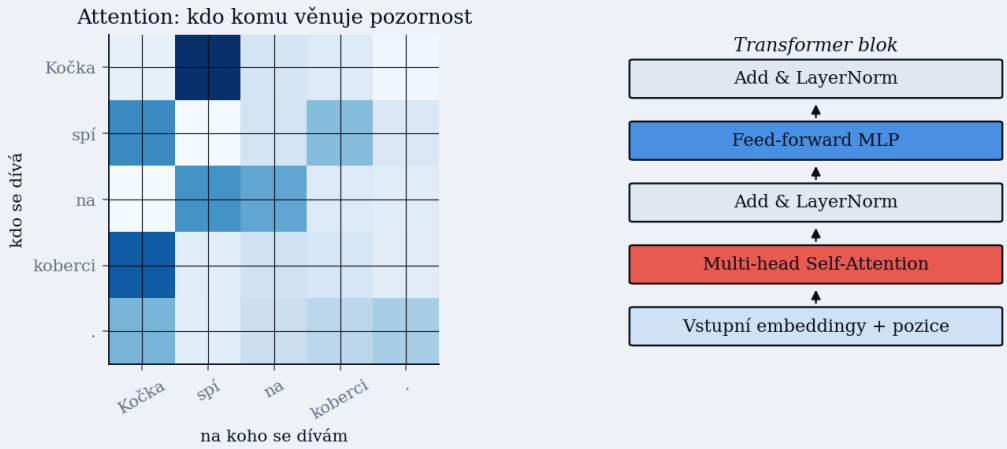
Pojďme k jádru moderní revoluce. Klíčový nápad je jednoduchý: **každý token v sekvenci se může „podívat" na všechny ostatní** — i na ty vzdálené — a vzít si od nich kontext. Tomu se říká **attention**. Místo aby informace plynula sériově od slova ke slovu (jako v RNN), každý token v jediném kroku vidí celý kontext.

Mechanicky to funguje tak, že každý token vyrobí tři vektory: **query** (na co se ptám), **key** (co nabízím) a **value** (jakou informaci nesu). Pro každý token se pak spočítá podobnost jeho query s *klíči* všech tokenů (včetně sebe). Z těchto podobností se udělá pravděpodobnostní rozdělení (softmax) — to je **attention matrix**. A finální výstup je vážený součet hodnot všech tokenů, kde váhy jsou ty pravděpodobnosti.

SCALED DOT-PRODUCT ATTENTION

$$\text{Attention}(Q, K, V) = \text{softmax}(Q \cdot K^T / \sqrt{d}) \cdot V$$

každý token spočítá vážený průměr informací od ostatních



Obr. 12 — Vlevo: attention matrix pro krátkou větu — tmavší pole znamená silnější vazbu. „Spí" se silně dívá na „kočka" (kdo to dělá). Vpravo: jeden Transformer blok — attention, pak feed-forward, normalizace, residuální spojení.

Síla attention je v paralelismu. RNN musí zpracovat token po tokenu, sériově. Transformer počítá vztahy *všech proti všem* najednou — což GPU miluje. Trénink je proto rychlejší a může pracovat s mnohem delšími sekvencemi. Jediný blok attention nestačí na všechno; transformer skládá **desítky bloků** za sebe a každý má více „hlav" attention — každá hlava se může soustředit na jiný druh vztahu (gramatický, sémantický, dlouhý dosah, krátký dosah).

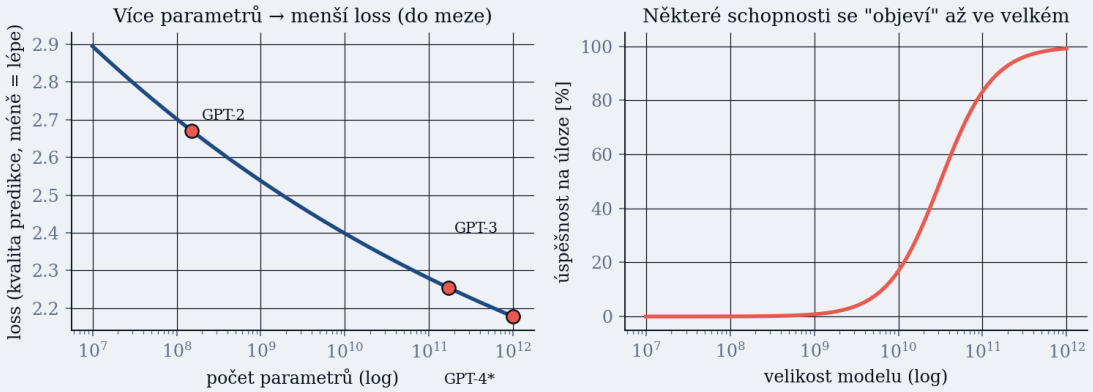
KAPITOLA 13

Velké jazykové modely

Když škáluješ Transformer dost, něco zajímavého se začne dít

Transformer je architektura. Aby z něj byl **jazykový model**, dáme mu cíl: „předpověz další slovo". Trénuješ na obrovském množství textu z internetu — model vidí kus věty a má říct, co bude dál. Jednoduchý úkol, ale když ho děláš na petabajtech textu s modelem o miliardách parametrů, naučíš se překvapivě hodně.

Aby model uměl pokračovat větu, musí mít *nějaké* chápání gramatiky, faktů o světě, logiky, kontextu, sarkasmu, idiomů. Nepředkládá se mu žádné z toho explicitně — vše musí **vyvodit** jen z toho, kdy které slovo následuje které. Říká se tomu **self-supervised learning**: úkol si model vyrábí sám z dat, žádné lidské anotace nejsou potřeba.



Obr. 13 — Vlevo: empiricky zjištěná zákonitost „škálovacích zákonů" — loss klesá s velikostí modelu jako mocninná funkce, do určité meze. Vpravo: některé schopnosti se objeví skokově až při určité velikosti — emergent abilities.

Tomu, co vidíš na grafech vlevo, se říká **scaling laws**. Když zvětšíš model, data, nebo výpočet o jeden řád, loss klesne o předvídatelný kousek. Tato zákonitost platí přes několik řádů velikosti a je důvodem, proč se velké laboratoře vrhly do tréninku gigantických modelů — **věděly dopředu, že to bude lepší**, jen nevěděly, jak moc.

Jakmile máš trénovaný „bázový" model, stojíš před jedním krokem: **alignment**. Surový model umí pokračovat libovolný text, ale neví, že má být užitečný a slušný. Proto se **doladí** (fine-tuning) na lidských instrukcích a upravuje pomocí **RLHF** (Reinforcement Learning from Human Feedback) — lidé hodnotí odpovědi a model se učí preferovat ty, které lidé preferují. Tak vzniká chatbot.

> **LIMITY, KTERÉ STOJÍ ZA POVAHU**
>
> *Jazykový model neví, co je pravda. Generuje to, co je v jeho tréninkových datech pravděpodobné. Někdy správně, jindy halucinuje — vyrábí věrohodně znějící, ale nesprávné odpovědi. Nemá perzistentní paměť a jeho znalost končí cutoffem.*

Posledních pár let se přidávají **další schopnosti**: práce s obrázky a zvukem (multimodalita), spouštění nástrojů (vyhledávač, kalkulačka, kód), uvažování krok za krokem (*chain-of-thought*), agentické chování. Hranice mezi „jazykový model" a „kognitivní pomocník" se rozmazává a každý model po roce vypadá zastarale.

Stojí za to si všimnout, že všechno, co dnes vidíš pod výrazem „AI" — chatbot, generátor obrázků, překladač, asistent v kódu — sdílí stejný matematický základ z Kapitol 1 až 12. Přidává se k tomu jen velikost, data a vyladění úkolu. Není to magie. Je to optimalizace funkce na **obrovských** datech.

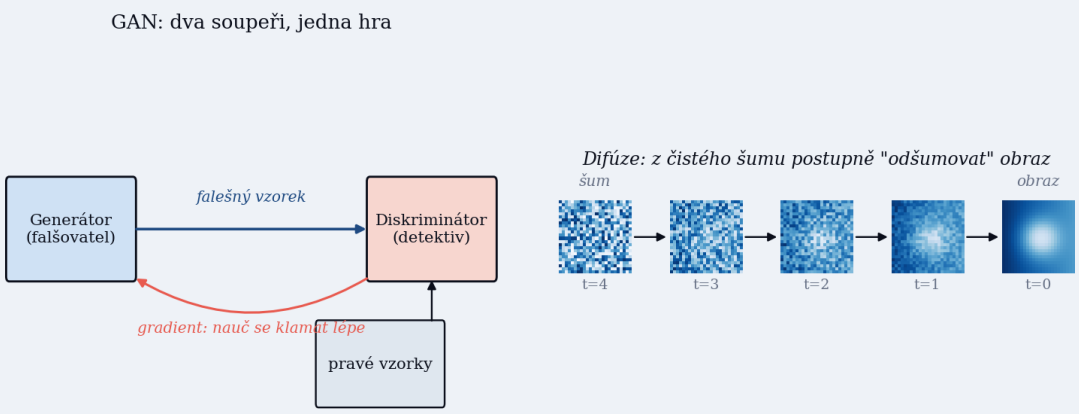
KAPITOLA 14

Generativní modely: GAN, VAE, difúze

Modely, které data nejen klasifikují, ale samy je tvoří

Až dosud se model snažil **poznat** — z obrázku říct, co je to za zvíře. Generativní model dělá opak — z popisu „kočka na střeše v noci" má vytvořit obraz, který předtím neexistoval. Tři hlavní rodiny generativních modelů řeší tento úkol různými chytrými způsoby.

První historicky úspěšná rodina jsou **GAN** (Generative Adversarial Networks). Koncept je dramatický: postavíš proti sobě dvě sítě a necháš je hrát. **Generátor** vyrábí falešné vzorky a snaží se je vydávat za pravé. **Diskriminátor** je detektiv, který se snaží odlišit pravé od falešných. Obě sítě se trénují současně — diskriminátor se učí lépe rozeznávat, generátor se učí lépe falšovat. V rovnováze generátor vyrábí vzorky tak dobré, že je ani diskriminátor nerozezná.



Obr. 14 — Vlevo: GAN jako duel dvou sítí. Vpravo: difúzní model — z čistého šumu se postupně „odšumovává" obraz. Každý krok odebere kus náhody.

Druhá rodina jsou **VAE** (Variational Autoencoders). Encoder zkomprimuje vzorek do nízkodimenzionálního *latentního* vektoru, Decoder z něj vzorek rekonstruuje. Latentní prostor je trénován tak, aby měl pravidelný tvar (gaussovský) — sampuješ z něj nové vektory a Decoder generuje nové podobné vzorky.

Třetí rodina, která dnes dominuje, jsou **difúzní modely** (Stable Diffusion, DALL-E, Midjourney). Princip: vezmi obraz, postupně přidávej šum, dokud nezbude čistý šum. To umíš ručně. Nyní nauč síť **zpětně** šum odebírat — krok za krokem. Pak generuješ tak, že začneš se šumem a postupně ho „odšumováváš".

DIFÚZNÍ MODEL — DVA SMĚRY

```
forward:   x_0 → x_1 → … → x_T  (přidávám šum, vím jak)
```

reverse: x_T → x_{T-1} → … → x_0 (síť odebírá šum, učí se to)

Difúzní modely se snadno spojily s textovým podmíněním — přidat do odšumování informaci „chceme, aby výsledek odpovídal této větě" — a tak vznikly modely **text-to-image**. Stejný princip dnes funguje pro zvuk, video, 3D modely. Klíčový vhled: **generování není jeden krok, ale postupné dolaďování**.

> **JEDNA RODINA, RŮZNÉ TVÁŘE**
>
> *Jazykový model je také generativní model — generuje text token po tokenu, autoregressivně. Difúzní model generuje obraz postupným odšumováním. Princip stejný: rozsekat těžkou úlohu na sled snadných kroků. Jeden z nejhlubších společných rysů celého moderního AI.*

Obecný motiv, který se přes všechny generativní rodiny vrací: pravá náročnost není „naučit se data nazpaměť", ale „**naučit se distribuci**, ze které data pocházejí". Když ji modeluješ, umíš z ní vzorkovat — a vzorky vypadají jako pravá data, i když přesnou kopii nikdy neviděl.

SYNTÉZA

Co si odnášíme

Stručný přehled cesty a několik orientačních bodů, kam dál.

Prošli jsme cestu, která vypadá z kraje strašidelně, ale ze střechy je krajina prostá. Strojové učení je **optimalizace**: definuj funkci s parametry, definuj loss, minimalizuj loss gradientním sestupem. Všechno ostatní jsou jen různá rozhodnutí o **tvaru funkce** a **tvaru loss**.

Lineární regrese a logistická regrese definují funkci jako vážený součet vstupů. Stromy jako kaskádu otázek. Neuronová síť jako kompozici nelineárních transformací. CNN jsou neuronové sítě s prostorovou strukturou pro obrazy. RNN s časovou strukturou pro sekvence. Transformer s attention pro globální kontext. A generativní modely (GAN, VAE, difúze) všechny tyto stavební kameny aplikují na úlohu „nauč se distribuci a vzorkuj z ní".

Kde jsme dnes? Modely se škálují dál — jak velikostí, tak rozsahem dat. Multimodalita (text + obraz + zvuk + video v jednom modelu) je standardem. Agentické modely, které **jednají** ve světě (vyhledávají, spouští kód, manipulují s rozhraními) jsou aktivní fronta. **Reasoning modely**, které neodpovídají okamžitě, ale „přemýšlejí" v sérii kroků, ukázaly v posledním roce dramatické zlepšení na úlohách, které dříve modelům nepatřily — matematika, programování, vědecké úvahy.

Pokud tě tohle území zajímá hlouběji, mám tři návrhy. Pro **matematickou stranu**: klasická kniha „Pattern Recognition and Machine Learning" (Bishop) nebo „Deep Learning" (Goodfellow, Bengio, Courville). Pro **praktickou stranu**: kurzy na fast.ai od Jeremyho Howarda — postavené přesně tak, aby se začalo „velkým modelem na pravých datech" a teorie se přidává postupně. A pro **průběžný přehled** dění: blog Distill (i když se aktualizuje pomalu), a aktuální papery na arXivu, sekce cs.LG a cs.CL.

„Co nedokážu znovu postavit, tomu nerozumím."
— Richard Feynman