23. Problém generalizace strojového učení a přístup k jeho řešení (trénovací, validační a testovací sada, regularizace, předtrénování, multi-task learning, augmentace dat, dropout, ...).

Základy práce s neuronovými sítěmi

V případě přípravy dat pro práci s neuronovými sítěmi jsou potřeba konkrétně tři části datové sady. **Musí se nutně jednat o disjunktní množiny.** Jde konkrétně o:

- **trénovací** sadu (podmnožinu) – určená (nečekaně) k **natrénování neuronové sítě**; jde pokud možno o **největší** z těchto množin; v případě komplexnějších úloh se může jednat o miliony datových vzorků
- **validační** sadu – může se používat pro **průběžnou validaci v průběhu tréninku** pro účely ověření správné **generalizace**; používá se pro **odhad specifických parametrů** pro provoz (příklad může být stanovení mezní hodnoty pro binární klasifikaci – výstup sítě je float hodnota "jistoty" ANO/NE, podle které je potřeba vytvořit toto binární označení prahováním)
- **testovací** sada – ta už se používá k **otestování výkonnosti sítě v "real world" provozu na již natrénované síti**

Data musí být "ušita" na míru a v případě učení s učitelem musí obsahovat **anotace** (požadovaná podoba výstupu pro každý vzorek dat). Ačkoli výsledky loss se v průběhu tréninku můžou snižovat, výsledky validací se můžou zhoršovat (viz. další kapitola).

Během trénování většinou dochází k několika průchodům celou trénovací datovou sadou. Jednomu průchodu se říká **epocha**. Epocha ve většině případů není procházena po jednotlivých vzorcích dat, ale po dávkách (viz **mini-batch gradient** v otázce 25).

Problém generalizace

Pojem **generalizace** je schopnost metod strojového učení performovat na dříve neviděných datech. Je to optimální stav, kterého chceme při trénování neuronových sítí dosáhnout. Oproti tomuto optimálnímu stavu známe pojmy **overfitting** (přetrénování) a **underfitting** (podtrénování), které oba značí neschopnost generalizace.

Underfitting je stav, kdy neuronová síť není dostatečně naučená a není schopna dostatečně aproximovat požadovanou funkci. Underfitting může být způsobený:

- nedostatečně dlouhým tréninkem
- nedostatečnou kvalitou trénovacích dat
- příliš komplexní úlohou na zvolený model neuronové sítě

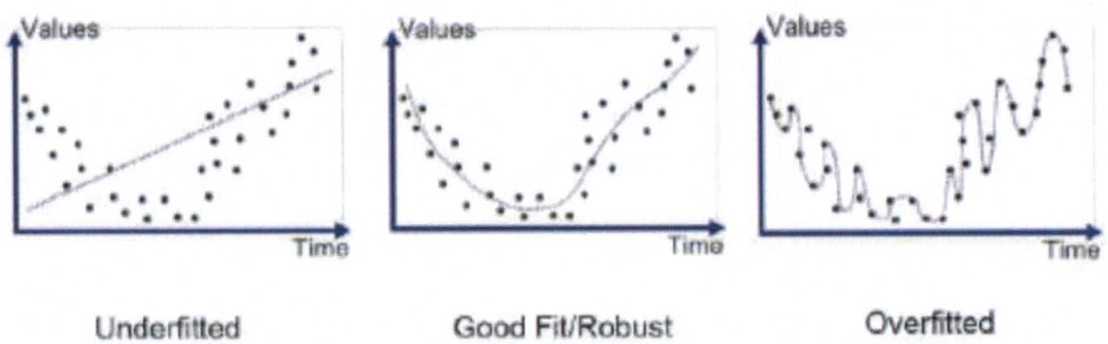
Underfitting lze poznat podle toho, že trénovací loss dostatečně neklesá (to dost záleží na vybrané loss funkci) a hlavně to jde poznat na výsledků validací.

Overfitting je přesně opačný jev, kdy dojde k přílišnému přizpůsobení trénovacím datům. V extrémním případě si to můžeme představit tak, že se neuronová síť naučí přesnou podobu trénovacích dat a umí predikovat správné výstupy pouze v těchto případech. Overfitting je způsobený:

- příliš dlouhým tréninkem
- malou diverzitou (nedostatkem) trénovacích dat
- komplexita modelu je overkill vzhledem k jednoduchosti úlohy

Overfitting jde poznat podle toho, že hodnoty loss během tréninku klesají, ale výsledky validací jsou na tom stále hůř (nebo obecně špatně).

Na obrázku je demonstrován každý z případů:



*Konkrétnější problémy co mohou nastat během trénování jsou fenomény **vanishing gradients** (mizející gradienty) a **exploding gradients** (explodující gradienty). Jde o vlastnost spočtených gradientů při zpětném průchodu sítí (detailněji v otázce 25). Jak názvy vypovídají, jde o příliš malé nebo příliš velké hodnoty gradientů, kde může docházet k problémům s float výpočty, nebo dokonce generování NaN hodnot a pádu či úplného zastavení učení. V případě explodujících gradientů pomáhá **regularizace**. V obou případech potom pomáhá tzv. **gradient clipping** (stanoví se min a max hodnoty, které gradienty mohou dosahovat), lepší inicializace vah před tréninkem, změna architektury, loss funkce, optimalizátoru nebo změna rychlosti učení (learning rate).*

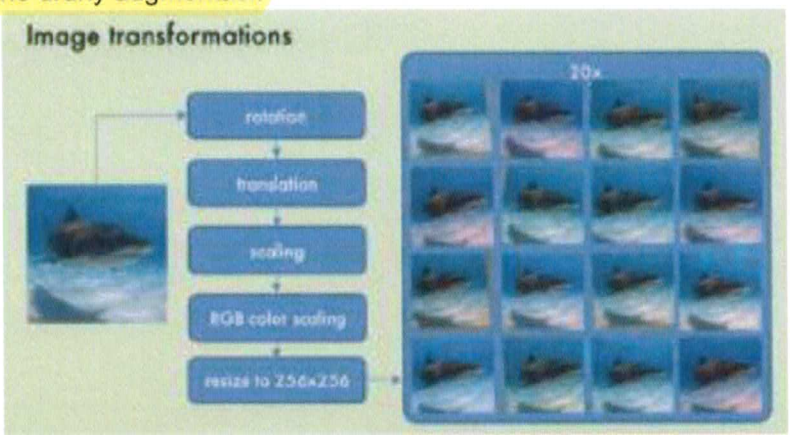
Regularizace - technika, která omezuje složitost modelu přidáním postihu za příliš velké váhy za účelem lepší generalizace modelu

Řešení problémů s generalizací

Augmentace

Jedním z řešení obou problémů generalizace (underfitting a overfitting) se dá řešit augmentací dat. Při augmentaci jde o "drobnou" modifikaci trénovacích dat pro zajištění větší diverzity. Jedná se o hodně "mocný" nástroj a je velmi efektivní. Modifikovaná informace ale **musí zůstat čitelná**

(pořád musí zůstat podoba dat obdobná datům z "reálného světa"). V případě obrazových dat může jít např. o náhodnou rotaci, posunutí, drobná úprava barev, rozmazání, apod. Každý typ dat podporuje jiné druhy augmentací.



L1 a L2 regularizace

Princip L1 a L2 regularizace spočívá v penalizaci neuronové sítě při učení v případě vysokých hodnot vah neuronů. Jednotlivé regularizační prvky se přičítají k hodnotě loss následným způsobem:

L1 Regularization

$$\text{Modified loss function} = \text{Loss function} + \lambda \sum_{i=1}^{n} |W_i|$$

L2 Regularization

$$\text{Modified loss function} = \text{Loss function} + \lambda \sum_{i=1}^{n} W_i^2$$

kde n značí počet vah a λ je parametr určující vliv regularizačního členu. L1 regularizace je sama o sobě neefektivní kvůli podobě její funkce. Ostrý přechod tvořený absolutní hodnotou vynucuje snižování vah bez ohledu na jejich hodnoty. To způsobuje **řídkost vah**, kde mnoho neuronů má nulové váhy (informace se ignorují a jsou pro výpočty zbytečné), což může vést k nestabilnímu učení a neefiktivitě sítě. L2 regularizace je obecně efektivnější, ale používá se často kombinace obou.

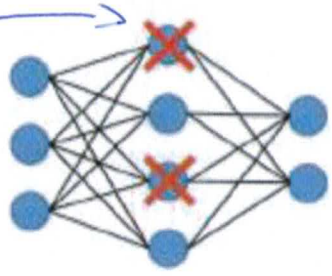
Dropout

– další způsob omezení síly modelu (způsob regularizace)

Dropout je účinná metoda při řešení problémů s generalizací. Jde o **náhodné "vypínání" neuronů neuronové sítě v průběhu tréninku**. Nejčastěji se jedná o dočasné nastavení nulových vah pro náhodně zvolené neurony v průchodu jedné dávky **mini-batch**. Neuronová síť se tak naučí vytvořit více robustní spojení, které pak hraje důležitou roli ve stabilitě. Při následné **inferenci** (dopředný průchod na již natrénovaném modelu) se tyto neurony znovu "zapnou", jejich

nulování se provádí pouze v průběhu tréninku. Neuronové sítě se často učí redundantní spojení a ty často dokážou zlepšit jejich stabilitu. Nesmí jich být ovšem moc.

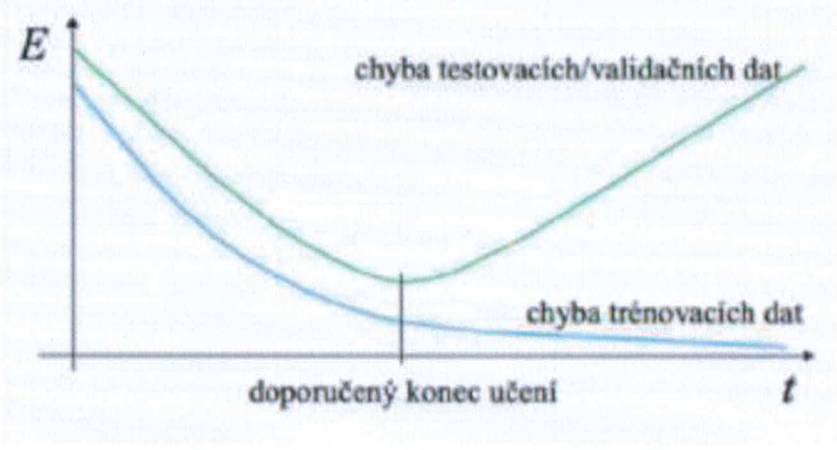
Dropout



Včasné zastavení

Early stopping

Jedním ze způsobů řešení problémů s generalizací (konkrétně overfitting) je včasné zastavení. Dlouhý trénink obzvláště při využití příliš komplexního modelu neuronové sítě pro jednoduchou úlohu často dokáže způsobit právě overfitting. Brzké zastavení tréninku tento problém umí vyřešit - pro to se často provádí průběžná validace během tréninku, která může sloužit jako indikátor pro zastavení tréninku v té nejoptimálnější fázi. Je nutné ale pamatovat, že tato validace nemusí přesně reprezentovat data reálného světa.



Předtrénování a finetuning

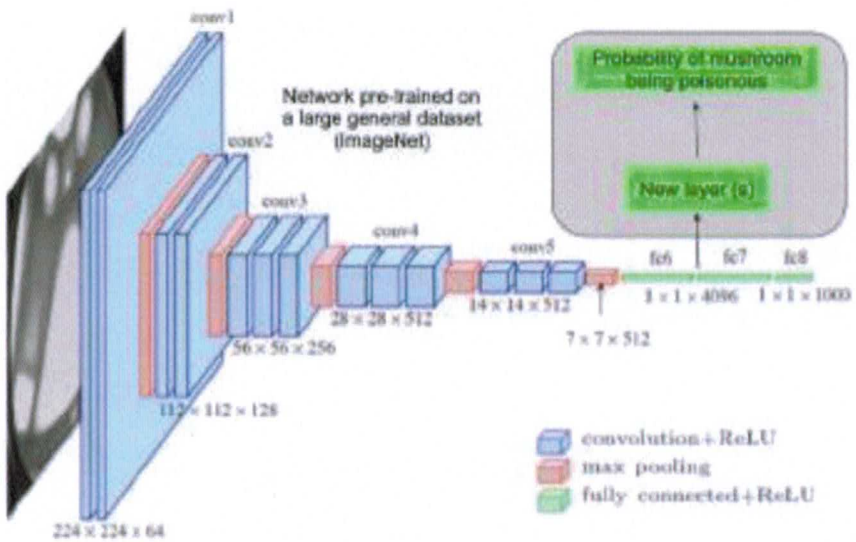
Jedním z nejpoužívanějších způsobů pro dosažení kvalitních výsledků je využití předtrénovaných modelů neuronové sítě. Výsledky téměř všech optimalizačních algoritmů jsou výrazně lepší v případě, kdy se nevychází z nulového/náhodného bodu, ale je provedena inicializace již v nějakém lepším bodu prostoru parametrů. To obzvlášť platí pro neuronové sítě. (Příkladem mohou být sítě pro zpracování obrazu, kde se využívají předtrénované modely, které už umí rozpoznávat základní primitiva v obraze.)

Základním principem je využití předtrénované **páteřní sítě** (backbone) a přidání vrstvy (může jich být více) specifické pro řešenou úlohu na konec sítě (**hlava**). Může se jednat například o přidání hlavy pro binární nebo vícetřídní klasifikaci. Tato vrstva má nejčastěji podobu lineární vrstvy a

často se v ní provádí také finální redukce dimenzionality (tentokrát ne způsobem poolingu, ale přímo konstrukcí neuronů způsobem, který redukuje dimenzionalitu). Lineární vrstva neuronové sítě je běžná ("vanilla") vrstva bez aktivační funkce (nelinearity), která modeluje pouze lineární transformaci.

Doladění (**finetuning**) probíhá buďto učením pouze hlavy sítě zatímco páteřní síť zůstává beze změny, a nebo se provádí trénink celé sítě naráz s různými volbami rychlosti učení (learning rate) pro hlavu a páteřní síť. Při trénování celé sítě je možné dosáhnout lepších výsledků.

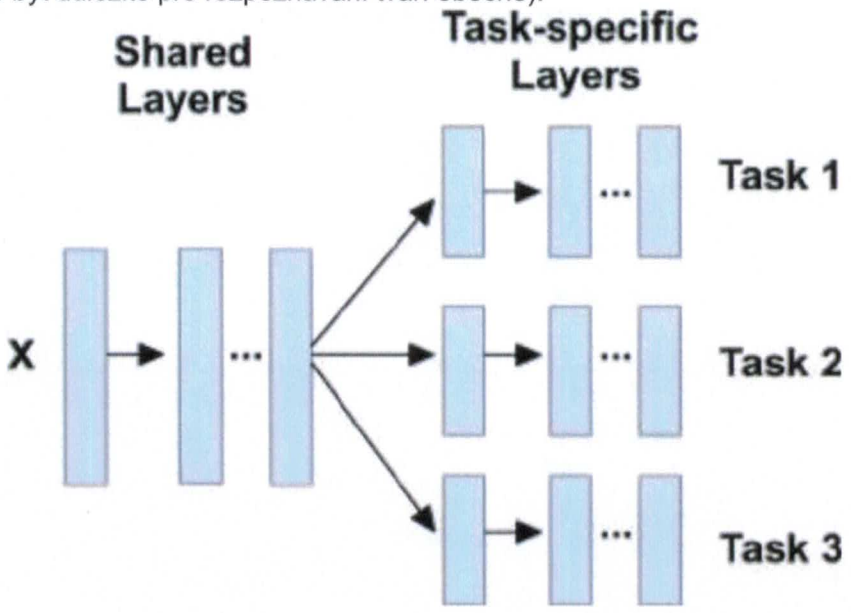
Pokud nevyužíváme předtrénované modely, je potřeba alespoň správná inicializace vah. Pokud je necháme všechny na 0, tak bude mít síť problém se cokoli naučit. Obvykle inicializujeme váhy nějakým malým náhodným číslem. Jednou z nejpoužívanějších metod je Xavier initialization.



Multitask learning

Multitask učení je princip, kdy učíme neuronovou síť řešit více úloh naráz. **Nemůže se jednat o náhodné úlohy, musí mít spolu něco společného.** Obvyklá implementace zahrnuje využití alespoň separátní lineární vrstvy (hlavy sítě) pro účely každé úlohy, zatímco zbytek sítě zůstává společný. Loss funkce bývá pro každou úlohu rozdílná a často jsou potřeba dodatečné anotace ke každé z úloh. Většinou nás ve výsledku zajímá pouze řešení jedné z těchto úloh a tento multitask využíváme jen pro zlepšení generalizace (nemusí to tak ale být). Tento způsob dokáže zlepšit generalizaci a může pomoci při učení neuronové sítě. (Příkladem může být učení neuronové sítě rozpoznávat obličeje s pomocnými úlohami, jako je odhad pohlaví osoby, klasifikace barvy vlasů, detekce roušky, ...).

Cílem multitask učení bývá "přinucení" páteřní sítě vracet aktivace (výstup), ze kterých je možné tyto informace následně "vytáhnout" nějakou lineární transformací. To dokáže zlepšit výsledky, jelikož daný faktor (cíl úlohy) může hrát důležitou roli v podobě výstupů sítě. (např. identifikace pohlaví může být důležité pro rozpoznávání tváří obecně).



Batch normalizace

Batch normalizace se používá k normalizaci vstupních dat do standardní podoby. Může pomoci k rychlejšímu učení sítě a lepší stabilitě tréninku. Batch normalizace je většinou implementována, jako kdyby šlo o vrstvu neuronové sítě, která se následně vkládá mezi ostatní vrstvy. Jde o normalizace vzhledem k jednomu **mini-batch**. Tato normalizace bere jako vstup výstup předchozí funkce nebo dokonce přímo vstupní data. Výstupem jsou normalizovaná data tak, že mají střední hodnotu 0 a rozptyl. Výpočet probíhá takto:

Input: Values of x over a mini-batch: $\mathcal{B} = \{x_{1...m}\}$;
Parameters to be learned: γ, β
Output: $\{y_i = \mathrm{BN}_{\gamma,\beta}(x_i)\}$

$$\mu_{\mathcal{B}} \leftarrow \frac{1}{m}\sum_{i=1}^{m} x_i \qquad \text{// mini-batch mean}$$

$$\sigma^2_{\mathcal{B}} \leftarrow \frac{1}{m}\sum_{i=1}^{m}(x_i - \mu_{\mathcal{B}})^2 \qquad \text{// mini-batch variance}$$

$$\widehat{x}_i \leftarrow \frac{x_i - \mu_{\mathcal{B}}}{\sqrt{\sigma^2_{\mathcal{B}} + \epsilon}} \qquad \text{// normalize}$$

$$y_i \leftarrow \gamma \widehat{x}_i + \beta \equiv \mathrm{BN}_{\gamma,\beta}(x_i) \qquad \text{// scale and shift}$$

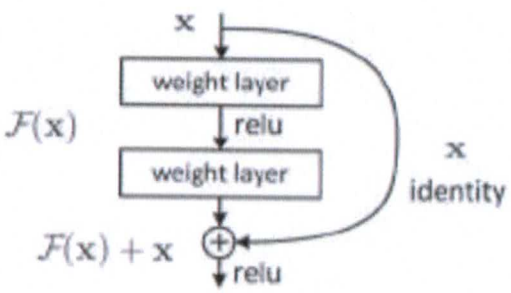
Algorithm 1: Batch Normalizing Transform, applied to activation x over a mini-batch.

Parametry β a γ jsou učitelné parametry, které provádějí dodatečnou lineární transformaci nad normalizovanými daty.

Batch normalizace se nedá použít stejným způsobem při inferenci - tam statistiky mini-batchů nemáme k dispozici. Součástí implementace vrstvy batch normalizace často bývá vytvoření globálních statistik trénovací sady (**running average** a **standard deviation**), které se po ukončení tréninku zafixují. U neuronových sítí se obecně počítá s tím, že trénovací data by měly mít pokud možno co nejpodobnější distribuci vzhledem k datům reálného světa.

Residuální spojení

Residuální spojení pomáhají řešit problém s trénováním hlubokých neuronových vrstev. Řešení spočívá přímo ve změně architektury přidáním residuálního spoje, zobrazeného na následujícím obrázku:



Jde zde prakticky o přičtení původní informace k výsledku aplikace vrstev neuronové sítě. Váhy v síti, kterými násobíme vstupní data, často mívají hodnoty < 1 a hluboké sítě poté trpí na ztrátu informace pronásobením velkým množstvím neuronů s takovými hodnotami. Řešení je "jednou za čas" přičíst původní hodnotu, aby nedocházelo k úplné ztrátě informace.

- řeší hlavně gradient vanishing

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