KMI/UMIN Umělá inteligence

Deep Learning

5. Generalizace a regularizace

Jan Konečný

Katedra informatiky, PřF UP v Olomouci

19. října 2026

Co nás dnes čeká

Protokol vyhodnocení

Kapacita, bias a variance

Explicitní regularizace

Dropout a jeho příbuzní

Šum, augmentace a cíle

Early stopping a implicitní regularizace

Jak postupovat

Generalizační mezera

Síť s dostatečnou kapacitou si dokáže zapamatovat celou trénovací množinu včetně jejího šumu a chyb v anotacích. Nulová trénovací chyba proto sama o sobě nevypovídá o kvalitě modelu.

Rozdílu mezi trénovacím a testovacím výkonem říkáme generalizační mezera; regularizace je souhrnné jméno pro všechno, čím se ji snažíme udržet malou.

Skripta: kap. 5 *Generalizace a regularizace*

Dva způsoby, jak si měření zkazit

Role tří množin i pravidlo, že závislé příklady musí zůstat pohromadě, známe z přednášky 3. Zbývají dvě věci.

Opotřebení validační množiny

Každé rozhodnutí učiněné podle validačního výsledku je také učení, jen ruční. Po dvou stech vyzkoušených konfiguracích je validační číslo optimistické podobně, jako bylo předtím to trénovací; pak je třeba velká validační množina, nebo vnořená křížová validace.

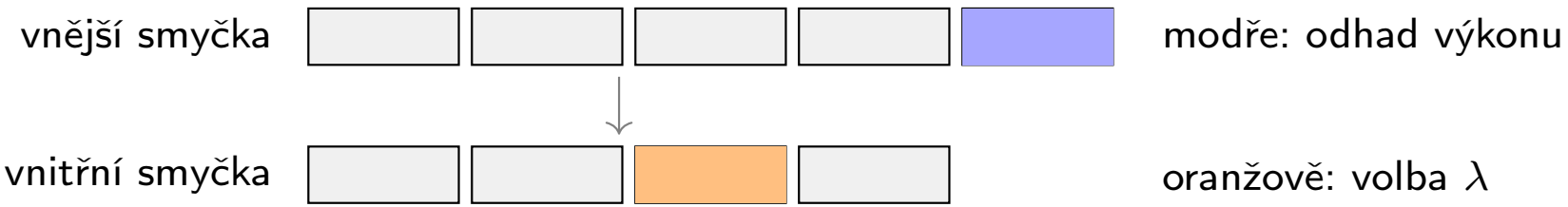
Únik informace

Do trénování se dostane něco, co by v ostrém provozu nebylo — nejčastěji přes předzpracování: průměr pro normalizaci ze všech dat naráz. Správně je rozdělit, statistiky vzít z trénovací části a přeškálovat jimi zbytek.

Testovací množina zůstává nedotčená až do konce.

Vnořená křížová validace

k-násobnou křížovou validaci známe z přednášky 3 — *k* foldů, průměr *i rozptyl*. Zbývá doplnit, jak ji použít, když se na týchž datech také *ladí*.

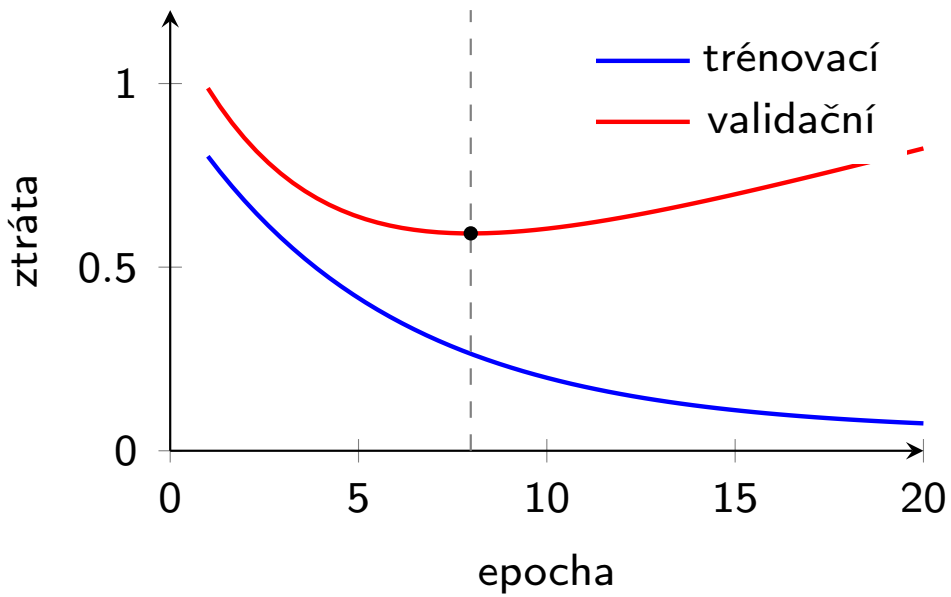


Vnější smyčka slouží jen k odhadu, vnitřní jen k ladění: hyperparametr se volí na datech, která se v tomtéž kroku nepoužijí k měření výsledku. Cenou je součin *k* vnějších foldů krát *m* vnitřních krát počet zkoušených hodnot; u velkých modelů ji proto nahradí velká validační množina.

Ladit a hodnotit nelze týmiž foldy

Vybereme-li podle křížové validace hyperparametry a tutéž hodnotu ohlásíme jako výsledek, je nadhodnocená. Platí i zde, co bylo řečeno v přednášce 3: křížová validace nahrazuje validační množinu, *ne* testovací.

Diagnostika z křivek učení

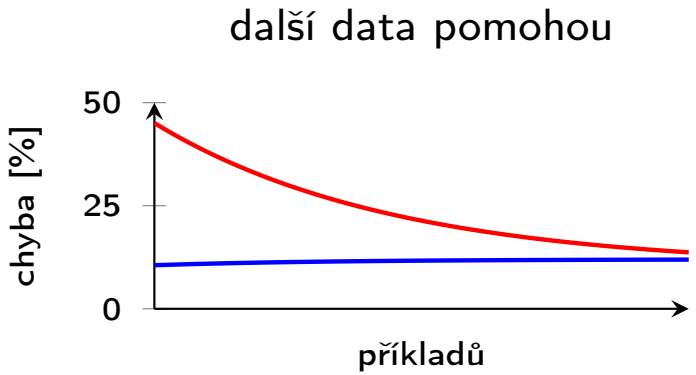


- **Underfitting:** vysoká trénovací i validační chyba.
- **Overfitting:** trénovací chyba klesá, validační přestane klesat nebo roste.
- Nejlepší validační ztráta zde nastává kolem osmé epochy (tečka).

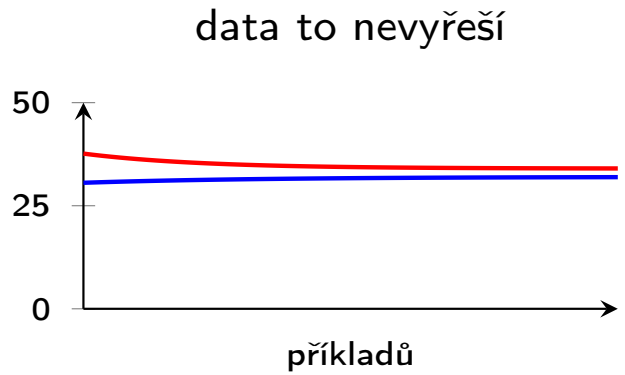
Na velikost mezery se nelze dívat samostatně: model s trénovací přesností 55 % a validační 52 % má mezeru malou, a přesto je nepoužitelný.

Jak moc pomůže víc dat

Vynesme proti *velikosti* trénovací množiny obě chyby — trénovací modře, validační červeně. Dopadnout to může dvěma způsoby (osy mají stejné měřítko):



Validační chyba pořád klesá a mezera se zmenšuje.



Obě chyby jsou vysoké a ploché, mezera malá.

Vpravo je problém v reprezentaci, kapacitě nebo optimalizaci a další příklady jej nevyřeší. Grafu se též říká learning curve; *pozor,* stejné jméno nese i graf proti počtu epoch z předchozí strany.

Klasický kompromis

Malá kapacita

Model má silné předpoklady a nemusí zachytit skutečný vztah: velká *soustavná odchylka* (*bias*).

Velká kapacita

Model reaguje na konkrétní výběr dat a jeho predikce se mezi datasety mění: velký *rozptyl* (*variance*).

Pozor na jméno: tenhle *bias* je soustavná odchylka modelu, ne posunovací parametr neuronu.

Regularizace typicky mírně zvýší trénovací chybu výměnou za větší snížení chyby validační.

U hlubokých sítí ovšem vztah kapacity a testovací chyby nemá vždy tvar jednoduchého U.

Sítě si zapamatují i náhodné labely

Zhang a kol. natrénovali standardní konvoluční sítě na obrázcích, jejichž labely byly *náhodně přeházené*. Sítě dosáhly nulové trénovací chyby — celý dataset se prostě naučily nazpaměť.

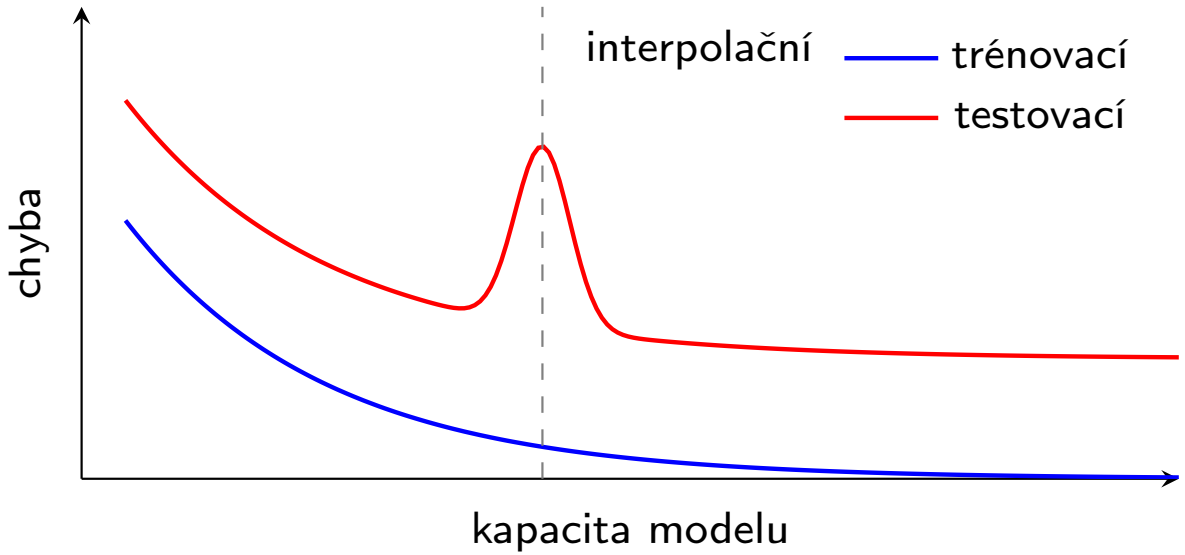
Z toho plyne několik věcí:

- Kapacita běžné sítě stačí na zapamatování celého trénovacího souboru, takže ji nelze použít jako vysvětlení dobré generalizace.
- Míry složitosti, které se počítají jen z velikosti modelu, tady nedávají užitečnou mez.
- Explicitní regularizace generalizaci zlepšuje, ale zapamatování nezabrání — není to tedy ona, co generalizaci způsobuje.

Rozdíl mezi učením a memorováním je vidět na průběhu: skutečné labely se model naučí rychle, náhodné pomalu a až po mnoha epochách. To je zároveň praktické vodítko, jak poznat chybně anotovaná data.

C. Zhang, S. Bengio, M. Hardt, B. Recht, O. Vinyals: Understanding Deep Learning Requires Rethinking Generalization. ICLR (2017).

Double descent



Při růstu kapacity se testovací chyba může poblíž bodu, kdy model poprvé dokonale interpoluje trénovací data, zvýšit a u ještě větších modelů znovu klesnout.

M. Belkin, D. Hsu, S. Ma, S. Mandal: Reconciling Modern Machine-Learning Practice and the Classical Bias–Variance Trade-Off. *PNAS* 116(32) (2019).

Deep double descent

Nemonotónní průběh nevzniká jen při změně počtu parametrů. Rozlišují se tři pohledy:

- model-wise — při zvětšování modelu,
- epoch-wise — při prodlužování tréninku,
- sample-wise — při změně počtu trénovacích příkladů.

Autoři je spojují pojmem *efektivní komplexity*, která závisí nejen na architektuře, ale i na trénovacím postupu a počtu epoch.

Praktický důsledek

Kapacitu měřme ve více bodech a sledujme u toho i trénovací chybu. Objeví-li se vrchol právě tam, kde trénovací chyba poprvé padne k nule, jde nejspíš o interpolační práh, za nímž se křivka může znovu zlepšovat.

P. Nakkiran a kol.: Deep Double Descent: Where Bigger Models and More Data Hurt. arXiv:1912.02292 (2019).

Regularizovaný optimalizační problém

$$\min_{\boldsymbol{w}} \frac{1}{n} \sum_{i=1}^{n} L\big(f_{\boldsymbol{w}}(\boldsymbol{x}_i), y_i\big) + \lambda\, \Omega(\boldsymbol{w})$$

První člen měří shodu s daty, druhý preferuje určitou strukturu parametrů; $\lambda \geq 0$ řídí kompromis.

L2

$\Omega = \|\boldsymbol{w}\|_2^2$, gradient $2\lambda\boldsymbol{w}$. Velké váhy jsou penalizovány silněji. Tvrdíme tím: malé váhy jsou předem pravděpodobnější.

L1

$\Omega = \|\boldsymbol{w}\|_1$. Derivace má mimo nulu stálou velikost, nezávislou na tom, jak je váha malá. Malé váhy proto dotlačí až na nulu: matice vah zřídne.

Příklad 1 (Jeden krok L2)

Pro $\boldsymbol{w} = (3, -4)$, gradient $(1, -2)$, $\eta = 0{,}1$ a $\lambda = 0{,}05$ je celkový gradient $(1{,}3, -2{,}4)$, tedy nové váhy $(2{,}87, -3{,}76)$.

Demonstrace: co penalizace udělá s vahou

Vezměme jednu souřadnici s kvadratickou ztrátou a L2 penalizací:

$$E(w) = \tfrac{1}{2} a\,(w - c)^2 + \alpha w^2 \quad \longmapsto \quad w^\star = \frac{a\,c}{a + 2\alpha}.$$

Váha se tedy nesmaže, jen se *smršťí* k nule — a o kolik, to určuje poměr křivosti a a síly penalizace α. *Křivost* je druhá derivace ztráty: jak prudce ztráta roste, když se od optima odchýlíme.

křivost a	bez penalizace	s $\alpha = 0{,}5$
10 (data si na směru trvají)	1,000	0,909
1	1,000	0,500
0,1 (data jsou lhostejná)	1,000	0,091

Penalizace tedy nepůsobí všude stejně: směry, které ztrátu skutečně ovlivňují, téměř nezmění, zatímco směry, v nichž je ztráta plochá, stlačí k nule. Právě ty jsou určeny šumem v datech.
Zde je také vidět, proč penalizovat bias nemá smysl: posunutí není směr, ve kterém by šlo přeučit.

Další varianty a co se nepenalizuje

- Elastic net sečte obě penalizace a poměrem ρ mezi nimi přepíná: $\rho\|\boldsymbol{w}\|_1 + (1-\rho)\|\boldsymbol{w}\|_2^2$.
- Penalizovat lze i celé *skupiny* vah naráz (group lasso). Pak nemizí rozsypaná jednotlivá čísla, ale rovnou celý neuron — a to se na zmenšení modelu hodí mnohem líp.
- Samotná řídkost nic nezrychlí: běžná GPU počítá všechny součiny bez ohledu na to, kolik je mezi nimi nul.

Co se z weight decay vyjímá

Biasy a parametry normalizačních vrstev (ty potkáme v přednášce 6).
V biasu se přeučit nedá, jen posouvá. A tlačit k nule koeficient, kterým normalizace škáluje svůj výstup, znamená rušit přesně to, kvůli čemu tam je.

Weight decay není totéž co L2

Weight decay zmenší váhy přímo:

$$\boldsymbol{w} \leftarrow (1 - \eta\lambda)\,\boldsymbol{w} - \eta\,\nabla E(\boldsymbol{w}).$$

U prostého SGD je to ekvivalentní L2 penalizaci (až na konvenci faktoru 2). U adaptivních metod se však gradient penalizace škáluje stejným mechanismem jako gradient datové ztráty, takže souřadnice nejsou zmenšovány rovnoměrně.

AdamW proto odděluje adaptivní krok od zmenšení vah.

Praktický důsledek

Hodnota λ opsaná z cizího článku neznamená v jiném optimalizátoru totéž. Než ji převezmete, zjistěte, zda knihovna počítá L2 penalizaci nebo oddělený weight decay a na které skupiny parametrů jej aplikuje.

I. Loshchilov, F. Hutter: Decoupled Weight Decay Regularization. ICLR (2019).

Jak silnou penalizaci zvolit

Hodnota λ se obvykle hledá na škále od 10^{-6} do 10^{-2} po řádech, a teprve kolem nalezeného optima jemněji.

Trénovací chyba s λ jen roste; validační má minimum. Je-li toto minimum na kraji zkoumaného rozsahu, rozsah je zvolen špatně.

Hyperparametry se ovlivňují

Optimální λ závisí na učicím kroku, velikosti dávky i počtu epoch; u AdamW navíc znamená totéž číslo jinou sílu než u SGD.

Dropout

Aktivace se během trénování násobí náhodnou maskou. Pro pravděpodobnost vynechání p se používá *inverted* varianta

$$\widetilde{\boldsymbol{h}} = \frac{\boldsymbol{m}}{1-p} \odot \boldsymbol{h}, \qquad m_j \sim \text{Bernoulli}(1-p).$$

Každé m_j je tedy jednička s pravděpodobností $1-p$ a nula s pravděpodobností p — nezávisle pro každou jednotku a každý průchod. Škálování faktorem $1/(1-p)$ srovná očekávanou hodnotu aktivace už během trénování, takže při inferenci stačí masku vypnout.

Příklad 2 (Inverted dropout)

Pro $\boldsymbol{h} = (2, 0, 3, 1)$, $p = 0{,}5$ a masku $\boldsymbol{m} = (1, 0, 1, 0)$ dostaneme $\widetilde{\boldsymbol{h}} = (4, 0, 6, 0)$. Jiná maska vznikne v dalším průchodu; průměr přes mnoho masek se blíží původní aktivaci.

N. Srivastava, G. Hinton, A. Krizhevsky, I. Sutskever, R. Salakhutdinov: Dropout: A Simple Way to Prevent Neural Networks from Overfitting. *JMLR* 15 (2014).

Dropout jako průměr mnoha sítí

Každá maska definuje jinou podsíť. Pro vrstvu s n jednotkami jich existuje 2^n a trénink je všechny sdílí — učí se tedy nepředstavitelné množství sítí naráz, a všechny mají jednu společnou sadu vah.
Při inferenci se těch podsítí neprochází ani zlomek; použije se celá síť s přeškálovanými aktivacemi tak, aby odpovídaly *průměru* přes masky. Prakticky se škáluje už při trénování (*inverted dropout*).

Příklad 3 (Kde se bere faktor)

Při $p = 0{,}5$ je jednotka aktivní v polovině masek, takže střední příspěvek je poloviční. Inverted dropout proto při trénování dělí zachované aktivace $1 - p$; při vyhodnocení se pak nedělá nic.

Trénovací a vyhodnocovací režim

Zapomenuté přepnutí režimu je častá tichá chyba: model s aktivním dropoutem vrací při validaci pokaždé jiný výsledek a vypadá horší, než je.

Kam dropout patří a kam ne

- V plně propojené vrstvě se umisťuje za nelinearitu.
- V konvoluční síti nesou sousední pixely skoro totéž, takže vynulovat každý zvlášť skoro nic neskryje — vypínají se proto celé mapy naráz (spatial dropout). Ke konvolucím se dostaneme v přednášce 7.
- V sítích pro sekvence a v Transformerech má dropout vlastní místa a pravidla; to si necháme na přednášky 9 a 10.

Při velkém předtrénování bývá nastaven na nulu, protože korpus je natolik rozsáhlý, že k přeučení nedochází; při doladění na několika tisících příkladech se naopak uplatní.

Strukturní vynechávání

- **DropConnect** nuluje jednotlivé váhy místo aktivací.
- **DropBlock** nuluje souvislý čtverec obrázku, ne rozsypané pixely.
- **Stochastic depth** přeskočí během trénování celou vrstvu; při inferenci se používá síť celá.

Tyhle varianty vypínají celé stavební prvky, a proto působí podstatně silněji než nulování jednotlivých čísel. Pravděpodobnost vynechání se obvykle zvyšuje s hloubkou.

Přeskočit přitom nelze libovolnou vrstvu, jen takovou, bez které zbytku sítě pořád sedí rozměry. Takové vrstvy mají moderní architektury od přednášky 7; pod jménem DropPath se totéž dělá i jednotlivým větvím sítě.

G. Huang, Y. Sun, Z. Liu, D. Sedra, K. Q. Weinberger: Deep Networks with Stochastic Depth. ECCV (2016).

Regularizace šumem

K vstupu, aktivacím nebo vahám lze během trénování přidávat náhodný šum; model se pak učí stabilní predikci v okolí trénovacích příkladů. Pro malý šum na vstupu se dá dokázat, že je to totéž, jako kdybychom ke ztrátě přidali člen trestající prudké změny výstupu při malé změně vstupu.

Šum musí napodobovat skutečnou variabilitu

Přičteme-li gaussovský šum k příznaku, který nemůže být záporný, nebo náhodně zpřeházíme kódy kategorií, vyrobíme příklady, jaké v datech nenastávají. Přehnaně silný šum navíc zvýší bias a skončíme u underfittingu.

Zvlášť ošidná je změna velikosti dávky: mění současně počet aktualizací za epochu, statistiky normalizace i efektivní velikost kroku. Přisoudit výsledné zlepšení „regularizaci šumem“ je pak dohad.

C. M. Bishop: Training with Noise is Equivalent to Tikhonov Regularization. *Neural Computation* 7(1) (1995).

Augmentace: co nemá měnit odpověď

Augmentace vytváří transformovaný příklad $T(\boldsymbol{x})$ se stejným nebo odpovídajícím cílem. Seznam použitých transformací je soupisem toho, co v dané doméně *nemá* měnit odpověď.

- U obrazu: výřez, posun, změny jasu a kontrastu, měřítko, někdy zrcadlení.
- Při detekci a segmentaci se současně transformují boxy i masky.
- U zvuku: posun času, realistický šum, maskování pásem.
- U textu jsou transformace nebezpečnější: parafráze může změnit význam.

Příklad 4 (Transformace, která mění label)

Zrcadlení fotografie kočky třídu zachová. Zrcadlením dopravní značky se šipkou vznikne značka s opačným významem a u rentgenu se prohodí strany těla.

Mixup a CutMix

Mixup

Vážený průměr dvou příkladů *i* jejich cílů. Model se tím tlačí k plynulému chování mezi trénovacími body. Smícháme-li kočku a psa v poměru 0,7 ku 0,3, má stejné hodnoty i cíl.

CutMix

Obdélníková oblast jednoho obrazu se vloží do druhého a cíle se smíchají podle podílu ploch. Zachovává lokální ostré struktury a nutí model využívat více částí objektu.

Prolnutá fotografie neodpovídá žádnému skutečnému příkladu, přesto poskytuje použitelný učicí signál.

Test-time augmentation je něco jiného: průměruje predikce několika transformovaných verzí téhož vstupu až při nasazení. S trénováním nemá nic společného a platí se za ni několikanásobným výpočtem.

H. Zhang a kol.: mixup: Beyond Empirical Risk Minimization. ICLR (2018). — S. Yun a kol.: CutMix. ICCV (2019).

Regularizace cílových hodnot

Label smoothing nahrazuje ostrý one-hot cíl rozdělením

$$\widetilde{y}_k = (1 - \varepsilon)\, y_k + \frac{\varepsilon}{K}.$$

Pro $K = 4$ a $\varepsilon = 0{,}1$ má správná třída hodnotu 0,925 a ostatní 0,025.

Cíl je nyní *dosažitelný*: k jedničce se dá dojít jen nekonečně velkým logitem, k hodnotě 0,925 konečným. Síť proto přestane logity neomezeně zvětšovat.

Za to se platí

Label smoothing stahuje reprezentace jedné třídy do těsných shluků, a maže tím jemnou informaci o tom, jak jsou si třídy navzájem podobné. Vadí to všude, kde se právě tahle informace dál používá — třeba když velký model slouží jako učitel menšího.

R. Müller, S. Kornblith, G. E. Hinton: When Does Label Smoothing Help? NeurIPS (2019).

Co s label smoothing zaměňovat nelze

- **Confidence penalty** přidá ke ztrátě člen odměňující vyšší entropii predikce — míří stejným směrem.
- **Fokální ztráta** vypadá příbuzně, ale řeší jiný problém: převahu snadných příkladů u nevyvážených tříd.

Jsou-li labely chybné, nespraví to žádný z těchto koeficientů. Pomůže jedině data pročistit nebo šum v anotaci explicitně modelovat.

Early stopping

Sledujeme předem zvolenou validační metriku, ukládáme nejlepší checkpoint a ukončíme trénink, pokud po daný počet kontrol nenastane dostatečné zlepšení.

- **patience** brání tomu, aby trénink skončil kvůli jedné nešťastné epoše,
- δ zabraňuje počítat mezi zlepšení pohyb na třetím desetinném místě.

Pozor na kombinaci s plánem učicího kroku

Běží-li vedle sebe scheduler typu reduce-on-plateau a early stopping se stejnou trpělivostí, ukončí se trénink zrovna ve chvíli, kdy se krok snížil a model by z toho teprve začal těžit.

Snadno se také přehlédne, že po skončení tréninku nechceme poslední epochu, ale nejlepší; checkpoint se proto musí skutečně ukládat.

L. Prechelt: Early Stopping — but When? In: Neural Networks: Tricks of the Trade. Springer (1998).

Implicitní regularizace

I bez explicitního členu $\Omega(\boldsymbol{w})$ nevybírá optimalizátor řešení náhodně. Inicializace, SGD, pořadí dat a architektura vytvářejí implicitní preferenci.

Relevantní faktory:

- měřítko počátečních vah,
- velikost mini-batche a šum gradientu,
- počet kroků a plán učicího kroku,
- tvar architektury: sdílení vah, konvoluce, zúžená místa, zkratky (přednášky 7 a 8),
- předtrénování a to, jak volně se smí model při doladění měnit (přednáška 8).

Nevýhodou je, že se implicitní regularizace nedá nastavit jediným parametrem. Zjistit její vliv lze jen ablací: měnit vždy jednu věc, zachovat stejný rozpočet a měření zopakovat s několika seedy.

Průměrování modelů a vah

- **Ensemble** průměruje predikce několika modelů a snižuje varianci. Modely musí dělat alespoň částečně odlišné chyby; několik kopií téhož checkpointu nepřinese nic.
- **EMA** udržuje během trénování vyhlazené váhy $\overline{\boldsymbol{w}}_t = \beta \overline{\boldsymbol{w}}_{t-1} + (1-\beta)\boldsymbol{w}_t$.
- **SWA** průměruje checkpointy z pozdější části trajektorie a může tak nalézt širší oblast s dobrou ztrátou.

Váhy jen z jedné trajektorie

Dva nezávisle natrénované modely mají tytéž neurony zpravidla v jiném pořadí, takže se při průměrování sečte první neuron jednoho s nesouvisejícím prvním neuronem druhého. Průměr jejich *predikcí* je naproti tomu v pořádku.

P. Izmailov a kol.: Averaging Weights Leads to Wider Optima and Better Generalization. UAI (2018).

Generalizace není totéž co robustnost

Vše dosavadní mlčky předpokládá, že testovací data pocházejí ze stejného světa jako trénovací.

Vyměníme-li kameru, přestěhujeme-li model do jiné země nebo přijde-li někdo, kdo se jej záměrně snaží zmást, přestává to platit; ani pečlivě naladěný dropout tu nepomůže.

Těmto situacím je věnována L12. Augmentace pomáhá jen vůči transformacím, které skutečně zahrnuje nebo z nichž model dokáže odvodit obecnější invarianci.