Neuronové sítě 1 - Vrstevnaté neuronové sítě

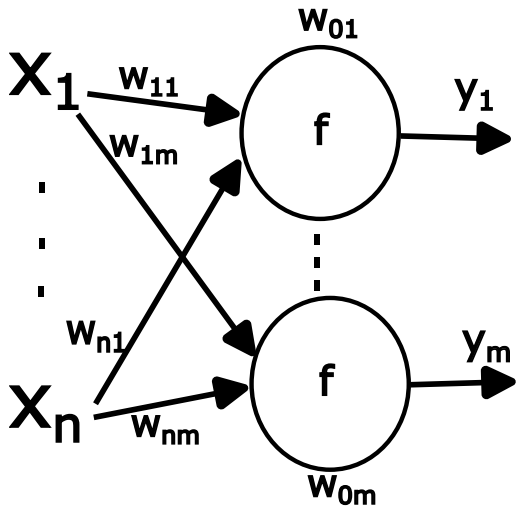
18NES1 - 11. a 12. hodina, LS 2024/25

Zuzana Petříčková

24-27. března 2025

Co jsme probírali minule

1. Neuronová síť s jednou vrstvou neuronů
   - Mnohorozměrná lineární regrese (lineární neuronová síť)
   - Lineární klasifikace do více tříd / rozpoznávání vzorů (jednovrstvý perceptron)



1. Vícevrstvá (vrstevnatá) neuronová síť (MLP)
2. Algoritmus zpětného šíření (backpropagation)

Vrstevnatá neuronová síť (multi-layer perceptron, MLP, 1980)

- hierarchická **sekvenční** architektura: neurony jsou uspořádány do vrstev
- **dense layers (plně propojené vrstvy)**: všechny neurony v jedné vrstvě jsou propojeny právě se všemi neurony z následující vrstvy

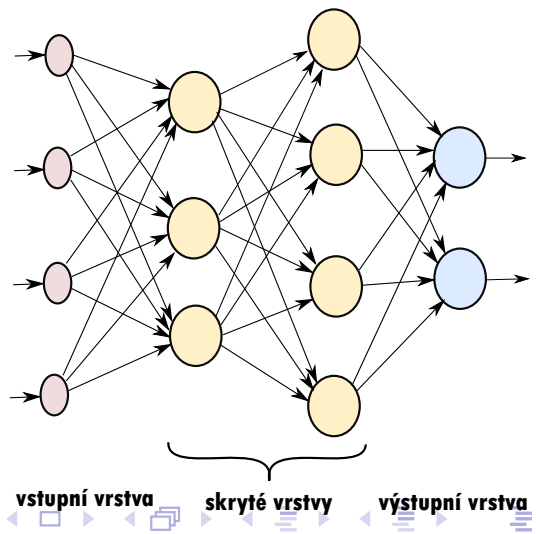
Speciální vstupní vrstva:

- odpovídá vstupům neuronové sítě

Výstupní vrstva

- výstup (odezva) neuronové sítě odpovídá výstupům (aktivitám) výstupních neuronů

Zbylé vrstvy jsou **skryté**.



Algoritmus zpětného šíření (Backpropagation)

Základní princip: je to standardní gradientní metoda

1. náhodně inicializuj parametry modelu (váhy a biasy)
2. opakuj trénovací cyklus:
   - připrav dávku (batch) trénovacích vzorů X a odpovídajících požadovaných výstupů D
   - spočti skutečný výstup (odezvu) modelu Y
   - spočti chybu modelu (jak moc se liší Y a D)
   - aktualizuj parametry (váhy a biasy) tak, aby se chyba modelu o něco zmenšila (tj. proti směru gradientu chybové funkce)

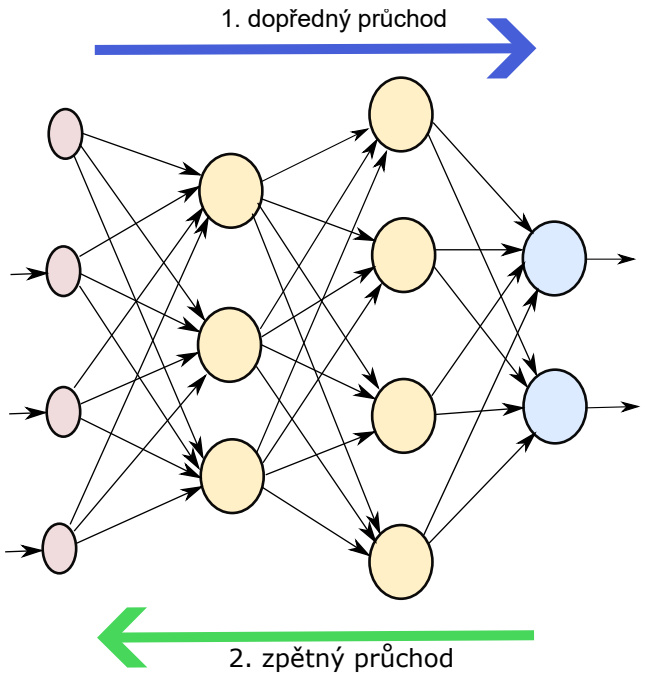
$$w_i(t+1) = w_i(t) - \alpha_t \frac{\partial E_t}{\partial w_i}$$

Hezké vizualizace hladin chybové funkce:
https://jithinjk.github.io/blog/nn_loss_visualized.md.html
https://izmailovpavel.github.io/curves_blogpost/

Algoritmus zpětného šíření (Backpropagation)

Základní princip (backpropagation)

1. Spočteme skutečnou odezvu sítě pro daný trénovací vzor (nebo dávku trénovacích vzorů).
   - jedním průchodem od vstupní vrstvy směrem k výstupní (forward pass)
2. Porovnáme skutečnou a požadovanou odezvu sítě.
3. Adaptujeme váhy a prahy:
   - proti směru gradientu chybové funkce
   - jedním průchodem od výstupní vrstvy směrem ke vstupní (backward pass)



Hlavní frameworky pro hluboké učení v Pythonu

- **TensorFlow**: Open-source knihovna od Googlu.
  - Výkonný framework pro AI aplikace (mobil, server).
  - Umožňuje jak dynamické, tak statické grafy výpočtů.
- **PyTorch**: Open-source knihovna od Meta (Facebooku).
  - Flexibilní a intuitivní, ideální pro výzkum a akademickou sféru.
  - Dynamické výpočtové grafy, snadné debugování.
- **Keras**: Univerzální High-level API.
  - Srozumitelné a přístupné začátečníkům.
  - Ideální pro rychlé prototypování, běží nad TensorFlow, JAX nebo PyTorch.
- **PyTorch Lightning**: High-level rozšíření PyTorchu.
  - Odstraňuje opakující se kód při učení modelů.
  - Podporuje více GPU, škálování modelů a snadnou reprodukovatelnost.
- **JAX**: Optimalizovaný pro výpočetní rychlost a výzkumné experimenty.
- Dříve populární **Theano**, dnes již nepodporovaný.

Srovnání TensorFlow a PyTorch

TensorFlow - stabilní systém, těžkopádnější

- součást většího ekosystému (TensorBoard,TF Lite...)
- velmi efektivní (napůl C++/Python), nativní distribuované učení, nativní podpora TPU
- optimalizovaný pro produkci, podpora kompilace, mobilního nasazení (TF Lite)
- méně programátorsky příjemná práce: složitěji se vytvářejí custom modely, zbytečně moc psaní kódu
- obtížné ladění u složitých modelů (uvnitř C++ knihovny)

PyTorch - poměrně mladý, rychle se vyvíjející systém

- příjemnější programování (pythonovský styl, méně kódu pro totéž), postupně dohání funkcionalitu
- výkonný (ale méně, používá čistý Python)
- flexibilnější, velmi snadno se vytvářejí custom modely, výborný pro výzkum, snadněji se ladí

Další užitečné knihovny:

Manipulace s daty a numerické výpočty

- **Scikit-learn(sklearn)**: Univerzální knihovna pro klasické algoritmy strojového učení. Nástroje pro práci s daty a vyhodnocování modelů.
- **NumPy**: Efektivní numerické výpočty, práce s poli a tensory.
- **Pandas**: Výkonná knihovna pro práci se strukturovanými daty (kategoriální, chybějící hodnoty). Změna reprezentace dat, filtrování, agregace.

Vizualizace

- **Matplotlib**: Obsáhlá knihovna pro tvorbu statických, animovaných a interaktivních vizualizací.
- **Plotly**: Interaktivní vizualizace
- **Seaborn**: Vizualizace složitějších vztahů mezi více proměnnými.
- **Tensorboard**: Vizualizace průběhu učení nejen pro (TensorFlow)

Tento týden

Pondělní hodina

1. Dokončení ukázek pythonovských frameworků pro neuronové sítě
2. O lokální instalaci knihoven

Čtvrteční hodina

1. Příklady v Tensorflow Playground
2. Stručná analýza modelu vrstevnaté neuronové sítě s algoritmem zpětného šíření
3. Nastavení hyperparametrů modelu vrstevnaté sítě - teoreticky a na praktické úloze v Kerasu

Příklady

NN_libraries.ipynb

- Okomentované ukázky předních pythonovských frameworků pro (hluboké) neuronové sítě (Keras, Tensorflow, PyTorch, Lightning) a jejich srovnání na jednoduché úloze binární klasifikace
- Automatické symbolické derivování tensorů v Tensorflow a PyTorch
- Frameworky a podpora GPU

NN_libraries_installation.ipynb

- Stručný návod na instalaci pro případ, že budete chtít programy spouštět lokálně na svých počítačích

Ukázka vrstevnaté neuronové sítě na různých jednoduchých úlohách:

https://playground.tensorflow.org/

- pět úloh (různě složité, nejtěžší je spirála)
- možnost navolit architekturu a další parametry
- moc pěkné vizualizace, včetně průběhu chyby, velikosti a znamének vah a toho, které příznaky jednotlivé neurony reprezentují
- můžeme experimentovat s tím, kolik vrstev a neuronů stačí na kterou úlohu,...
- **nejtěžší úkol:** naučit spirálu

Vrstevnatá neuronová síť učená gradientní metodou - analýza modelu

Výhody:

- Jednoduchý **univerzální** model s poměrně dobrými aproximačními schopnostmi
  - Vhodný pro úlohy klasifikace i regrese, včetně úlohy predikce časových řad.
  - Schopný zachytit i složité nelineární vztahy.
  - Dobře zobecňuje.
- Využívá backpropagation pro efektivní učení gradientní metodou.
- Univerzální aproximátor – zvládá aproximaci jakékoliv spojité funkce (pro některé nelineární přenosové funkce stačí jedna nebo dvě skryté vrstvy). Ale problém učení je NP-úplný.

NP-úplnost problému učení

Věta

- Obecný problém učení umělých neuronových sítí je NP-úplný. Výpočetní náročnost roste exponenciálně s počtem parametrů.

Poznámky

1. Věta platí i pro model vrstevnaté neuronové sítě a problém učení logických funkcí.
2. Pro některé speciální typy jednoduchých neuronových sítí je problém učení řešitelný v polynomiálním čase (pomocí metod lineárního programování).

$\rightarrow$ proto nám nezbývá než se smířit s lokálními metodami učení (gradientní metoda,...)

Vrstevnatá neuronová síť učená gradientní metodou - analýza modelu

Nevýhody:

- Model je velmi citlivý na inicializaci vah, trénovací data a hyperparametry, které je třeba vyladit.
- Omezení na tvar vstupních a výstupních dat (trénovací vzory = číselné vektory)
- Pomalá konvergence - ale existují rychlejší varianty algoritmu (např. Adam)
- Lokální metoda učení – nalezne suboptimální řešení.
- Náchylnost k přeučení – lze řešit pomocí různých technik, jako je regularizace, early stopping.
- Nemá zabudované mechanismy pro zohlednění prostorové struktury dat
- ,,Černá skříňka" - interní reprezentace znalostí pomocí vah a biasů je pro člověka špatně čitelná

Vrstevnatá neuronová síť učená gradientní metodou - analýza modelu

Chceme, aby učení lokální (gradientní) metodou bylo úspěšné:

- rychlost učení
- schopnost naučit se danou úlohu (správně zachytit skryté vzory v datech)
- schopnost zobecňovat (dávat správné výstupy i pro vzory co nebyly v trénovací množině)

Co je zásadní pro úspěch algoritmu zpětného šíření?

- vhodné předzpracování trénovacích dat
- vhodná inicializace vah a biasů (např. $\sim N(0, 1)$)
- nastavit správně hyperparametry pro danou úlohu

Příklad

keras_simple_example.ipynb

- Podrobnější příklad v Kerasu - postup učení krok za krokem (na úloze binární klasifikace)
- Analýza a předzpracování dat. Vytvoření modelu a nastavení jeho hyperparametrů. Průběh učení. Vizualizace. Zhodnocení.
- Ladění hyperparametrů
- Vizualizace pomocí TensorBoard

Mezi příkladem a slidy budeme průběžně přepínat

Předzpracování trénovacích dat pro MLP

Klíčové kroky předzpracování dat:

- **Serializace**
  - Převedení vstupních i výstupních dat na 2D tenzory tvaru (vzory, číselné příznaky)
  - Je třeba se vypořádat s kategorickými proměnnými (ordinal encoding, one-hot encoding)
- **Zabezpečení konzistence dat:**
  - Zkontrolujeme, že všechny vstupní vektory jsou stejně dlouhé, žádné hodnoty nechybí.
  - Chybějící data je třeba nahradit pomocí průměru, mediánu nebo s využitím sofistikovanějších metod.

Předzpracování trénovacích dat pro MLP

Klíčové kroky předzpracování dat:

- **Normalizace/Standardizace vstupů:**
  - **Normalizace**: Škálování hodnot na interval [0, 1] nebo [-1, 1] (v závislosti na zvolené přenosové funkci - relu vs. tanh).
  - **Standardizace**: typicky aby data měla střední hodnotu 0 a směrodatnou odchylku 1.
  - Normalizace je velmi důležitá pro to, aby se model dobře učil
- **Trénovací množina by měla být dostatečně velká a vyvážená.**
  - Někdy je třeba přistoupit k augmentaci trénovací množiny (zvětšení počtu trénovacích vzorů)
- **Rozdělení dat na trénovací, validační a testovací množinu:**
  - Obvyklé rozdělení je například 70% trénovací, 15% validační, 15% testovací sada.

Klíčové hyperparametry modelu MLP

Architektura

- **Velikost modelu**: Počet skrytých vrstev a neuronů v nich
- **Přenosové (aktivační) funkce na jednotlivých vrstvách**: relu, sigmoid, tanh, softmax,...

Další klíčové hyperparametry

- **Chybová funkce (loss function)**: MSE, binary crossentropy,...
- **Metriky pro vyhodnocení modelu**: accuracy, MSE, precission,..
- **Učící algoritmus (optimizer)**: SGD, Adam, RMSProp,...
- **Rychlost učení (learning rate)**, popř. další parametry učícího algoritmu
- **Batch size (velikost dávky)**
- **Počet epoch**
- **Inicializace vah** Typicky malé náhodné hodnoty
- **Regularizace**: L2, Dropout, Early stopping...

Architektura vrstevnaté neuronové sítě

Vytvoření modelu v Kerasu

- **Sequential** – nejjednodušší tvorba modelu, jednoduché skládání vrstev po sobě
- **Input** – vstupní vrstva (někdy se vynechává)
- **Dense** – plně propojená (fully connected) vrstva
  - Počet neuronů
  - Přenosová funkce: **activation='relu'**, **'sigmoid'**, **'linear'** (default), ...
  - Metoda inicializace vah: **kernel_initializer='glorot_uniform'**, **bias_initializer='zeros'** (default), ...
  - Příklad: **Dense(10, activation='relu', kernel_initializer='he_normal')**

Oficiální dokumentace:
https://keras.io/api/layers/core_layers/dense/

Architektura vrstevnaté neuronové sítě

- **Velikost modelu:** Je daná počtem vrstev a počtem neuronů v nich

Jak zvolit velikost modelu?

- **Vstupní a výstupní vrstva**: počty neuronů jsou dané tvarem dat
- **Skryré vrstvy:**
  - Větší model = větší kapacita $\rightarrow$ lepší schopnost naučit se složité vzory
  - Malý model $\rightarrow$ podučení (underfitting), nezachytí složitější vztahy
  - Příliš velký model bez dostatku dat $\rightarrow$ přeučení (overfitting)
  - Správný počet skrytých vrstev a neuronů v nich závisí na složitosti úlohy a velikosti dat. Obvykle ho určíme experimentálně.

Velikost modelu a její vliv na výkon MLP

Praktické doporučení:

- Začneme s menším modelem a postupně přidávej vrstvy/neurony podle potřeby.
- Použijeme validační data pro monitorování výkonu modelu a vyhnutí se přeučení.
- Pokud se model přeučuje, použijeme techniky jako je regularizace, early stopping nebo dropout.
- Volíme vhodný kompromis mezi šířkou (počet neuronů) a hloubkou (počet vrstev)
- U menších datasetů preferujeme menší modely

Architektura vrstevnaté neuronové sítě

- **mělká (shalllow)** - model s jednou skrytou vrstvou
  - je vhodnější pro jednodušší úlohy - model se pro ně učí rychleji a lépe zobecňuje
  - vystačí si s menšími trénovacími daty (naopak velká trénovací data mu nesvědčí)
  - je snazší ho pochopit a interpretovat
  - složité úlohy se učí pomalu a s obtížemi, potřebuje hodně neuronů
- **hluboká (deep)** - model s více (nebo i mnoha) skrytými vrstvami
  - je vhodnější pro složitější úlohy s velkými trénovacími daty - učí se pro ně lépe
  - je schopna zachytit i složité vztahy mezi daty
  - vyžaduje jiné učící mechanismy a potýká se v praxi s jinými problémy než mělký model

Jaké zvolit přenosové (aktivační) funkce?

Jakou přenosovou funkci použít ve výstupní vrstvě?

- regresní úloha: lineární (**linear**)
- klasifikace do dvou tříd: sigmoidální (**sigmoid**)
- klasifikace do k tříd: **softmax**

Jakou přenosovou funkci použít ve skrytých vrstvách?

- hyperbolický tangens (**tanh**) - stabilní, symetrická, ale možné problémy se saturací neuronů, oblíbená u rekurentních modelů
- v případě hlubokých sítí se často používá **ReLU** (pozitivně lineární) - rychlejší, efektivnější pro hluboké sítě, ale asymetrie a omezená schopnost reprezentace dat (i zde hrozí saturace)

https://keras.io/api/layers/activations/

Vhodná inicializace vah a biasů

Pravidlo

- Váhy a biasy by měly být malé, náhodné, rovnoměrně rozdělené, se střední hodnotou 0.

Proč střední hodnota 0?

- Očekávaná hodnota potenciálu každého neuronu bude 0.
- Derivace logsig / tanh je maximální pro 0 (~ 0.25) $\rightarrow$ výraznější změny vah na začátku
- Menší riziko saturace.

Proč náhodné?

- Snaha omezit symetrii. Skryté neurony by neměly mít navzájem podobnou funkci

Vhodná inicializace vah a prahů

Např. podle nějaké heuristiky:

- Základ: $w_{ij}(0) \sim N(0, 1)$
- Dobrá heuristika, např. metoda Nguyen-Widrow: snaží se neurony z dané vrstvy rozhodit ve vstupním prostoru zhruba rovnoměrně
- Glorot et al. (2010) : $w_{ij}(0) \sim N(0, \sqrt{\frac{6}{n_i+n_j}})$ pro matici vah tvaru $n_i \times n_j$ mezi dvěma vrstvami, snaží se, aby rozptyl výstupů byl přibližně stejný jako vstupů

Doporučení:

- **Pro ReLU: He inicializace (HeUniform, HeNormal)** - snaží se udržet rozptyl výstupů neuronů při průchodu sítě
- **Pro sigmoid/tanh/linear: Glorot (Xavier) inicializace (GlorotUniform, GlorotNormal)**

https://keras.io/api/layers/initializers/

Kompilace modelu v Kerasu (`model.compile`)

Slouží k definici způsobu učení modelu.

Základní parametry:

- **optimizer** – algoritmus pro učení (např. **'adam'**, **'sgd'**, **Adam(learning_rate=0.001)**)
- **loss** – chybová funkce, ta se bude během učení minimalizovat
- **metrics** – metriky pro průběžné sledování a závěrečné vyhodnocení vyhodnocení výkonu modelu (např. **['accuracy']**, **['mae']**)

Příklad: model.compile(optimizer='adam', loss='binary_crossentropy', metrics=['accuracy'])

Jakou zvolit chybovou funkci (loss)?

Pro regresní úlohu:

- **MSE (loss='mean_squared_error')**
  - velmi vhodná pro regresní úlohy, nejčastěji používaná
  - citlivá k odlehlým hodnotám
- **MAE (loss='mean_absolute_error')** - robustnější k odlehlým hodnotám
- **Huber Loss (loss='huber')** - kombinace předchozích
- ...

https://keras.io/api/losses/

Jakou zvolit chybovou funkci (loss)?

Pro klasifikaci:

- **Binární Crossentropy (loss = 'binary_crossentropy')**
  - vhodná pro klasifikaci do dvou tříd společně se sigmoidální přenosovou funkcí ve výstupní vrstvě
- **Kategorická Crossentropy (loss = 'categorical_crossentropy')**
  - vhodná pro klasifikaci do více tříd ve spojení se softmax přenosovou funkcí ve výstupní vrstvě
  - pro one-hot-coded výstupy
- **Řídká Kategorická Crossentropy (loss = 'sparse_categorical_crossentropy')**
  - jako kategorická crossentropy, ale pracuje s číselnými indexy tříd místo one-hot encodingu

Jakou zvolit metriku pro sledování výkonu modelu?

Klasifikace:

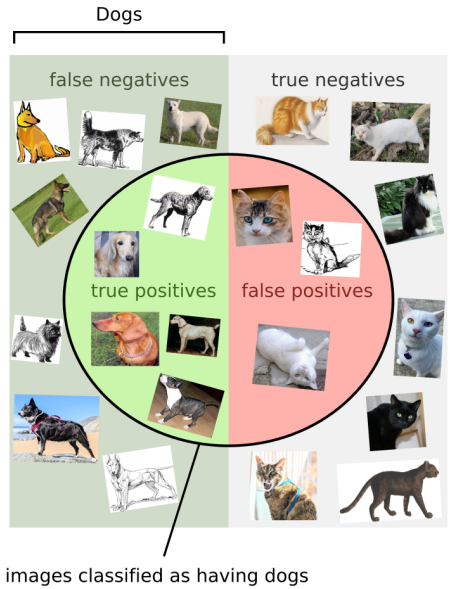
- **Accuracy (Přesnost)** - podíl správně klasifikovaných vzorů
  - **binární klasifikace**: **accuracy**, **binary_accuracy**
  - **klasifikace do více tříd: categorical_accuracy** (pro one-hot), **sparse_categorical_accuracy** (pro číselné indexy)
- **Další metriky pro binární klasifikaci**:
  - **AUC** - sleduje plochu pod ROC křivkou, vhodné pro nevyvážená data.
  - **Precision**, **Recall**, **F1** -vhodné při potřebě minimalizovat falešně pozitivní nebo negativní.

Regrese

- **mean_squared_error** (MSE) - pro běžná data
- **mean_absolute_error** (MAE) - pro data s outliery

https://keras.io/api/metrics/

Jakou zvolit metriku pro sledování výkonu modelu?



$$\text{Precision} = \frac{\text{5 true pos.}}{\text{8 total pos.}} \qquad \text{Recall} = \frac{\text{5 true pos.}}{\text{12 total dogs}}$$

$$\text{Prevalence} = \frac{\text{12 total dogs}}{\text{22 total images}}$$

$$\text{Accuracy} = \frac{\text{5 true pos. + 7 true neg.}}{\text{22 total images}}$$

zdroj: https://en.wikipedia.org/wiki/Precision_and_recall, licence CC0

Algoritmy učení (optimalizátory) hlubokých neuronových sítí

- vycházejí z gradientní metody a často používají adaptivní a lokální parametr učení
  - **SGD (Stochastic Gradient Descent)** (základní algoritmus, stochastický algoritmus zpětného šíření využívající mini-batche, stabilní)
  - **Adam** (aktuálně zřejmě nejpopulárnější, adaptivní parametr učení, rychlejší učení)
  - **RMSprop** (vhodný pro sekvenční a online data)
  - AdaGrad, Adadelta, AdaMax, NAdam, FTRL,...
- každý optimalizátor má další parametry (např. SGD: **learning_rate, momentum, nesterov**)
  často můžeme zachovat defaultní nastavení

https://keras.io/api/optimizers/

Volba vhodného parametru učení

- pro učení algoritmem SGD je zásadní správné nastavení parametru učení (learning_rate)
- parametr řídí, jak rychle se model učí

Jak volit parametr učení?

- moc malý ... pomalé učení (malé změny vah) - nebezpečí uvíznutí v suboptimálním lokálním minimu
- moc velký ... velké skoky - nebezpečí oscilací, mohu přeskočit lokální minimum chybové funkce

Co pomůže?

- nastavení optimální hodnoty parametru učení pro danou úlohu
- učení s momentem (parametry: momentum, nesterov)
- adaptivní parametr učení (Adam, RMSProp)

→ **více o algoritmech učení příště**

Učení modelu v Kerasu (`model.fit`)

Základní parametry:

- **x**, **y** – vstupní vzory a požadované výstupy
- **batch_size** – počet vzorů zpracovaných najednou (např. 32)
- **epochs** – počet průchodů celým datasetem
- **validation_data** – validační data, např. **(x_val, y_val)**
- **callbacks** – funkce volané během učení (např. **EarlyStopping**)
- **shuffle=True** – náhodné zamíchání dat před každou epochou

Příklad: model.fit(x_train, y_train, batch_size=32, epochs=10, validation_data=(x_val, y_val), shuffle=True)

Dokumentace:
https://keras.io/api/models/model_training_apis/#fit-method

Další klíčové hyperparametry modelu MLP

- **Batch size (velikost dávky)** -počet vzorů zpracovaných v jedné dávce.
  - obvykle mocnina 2 (8, 16, 32, 64,...)
  - malé dávky: pomalejší, méně stabilní učení, ale větší šance, že se model lépe se naučí a zobecňuje
  - velké dávky (512+): rychlejší učení, větší paměťové nároky, hrozí přeučení
  - **Doporučení:** pro malé datasety preferujeme menší velikost dávek, pro velké datasety zkoušíme co největší velikosti, ale hlídáme paměť a přeučení
- **Počet epoch**
  - určíme experimentálně, využijeme early-stopping

Callbacky v Kerasu

Funkce volané během trénování modelu
$\rightarrow$ umožňují sledovat průběh učení, uložit model, zastavit učení apod.

Nejčastější callbacky:

- **EarlyStopping** – zastaví trénování při nelepšícím se výkonu
- **ModelCheckpoint** – ukládá nejlepší model podle metriky
- **ReduceLROnPlateau** – sníží learning rate, pokud se metrika nezlepšuje
- **TensorBoard** – záznam průběhu trénování pro vizualizaci

Použití: viz příklad

Dokumentace:
https://keras.io/api/callbacks/

Keras model – metody pro vyhodnocení, predikci a uložení do souboru

evaluate() – vyhodnocení modelu na testovacích datech

- Vrací tuple obsahující hodnotu chybové funkce a všech metrik: **loss, accuracy = model.evaluate(x_test, y_test)**

predict() – výpočet výstupů modelu

- Např. pro klasifikaci: **y_pred = model.predict(x_test)**

save() – uložení modelu do souboru

- **model.save("model.eras")**

load_model() – načtení uloženého modelu

- **model = load_model("model.keras")**

TensorBoard

- Nástroj pro vizualizaci průběhu učení neuronových sítí.
- Umožňuje sledovat chybu, metriky, graf modelu, distribuce vah aj.
- Lze pozorovat průběžný vývoj během učení.
- Možnost porovnat více modelů mezi sebou.

Použití v Kerasu:

```
from keras.callbacks import TensorBoard
log_dir = "logs/fit/" + ...
tensorboard_callback = TensorBoard(log_dir=log_dir,...)
model.fit(..., callbacks=[tensorboard_callback,...])
```

Spuštění z příkazové řádky:

```
tensorboard --logdir=logs/fit
```

Dokumentace: tensorflow.org/tensorboard

Příklad - domácí úkol

keras_simple_example.ipynb

- Na příkladu si vyzkoušejte, jak změna různých hyperparametrů (architektura, algoritmus učení,...) ovlivní průběh a výsledek učení
- Nápady na to, co zkoušet, jsou úvedeny přímo v notebooku.
- Vyzkoušejte si práci s TensorBoard (případně i puštěný lokálně na svém počítači)
- Případně modifikujte kód, a snažte se naučit MLP na jiných datech ze scikit-learn repozitáře (např. iris, diabetes, wine).