Neuronové sítě 1 - Vrstevnaté neuronové sítě

18NES1 - 13. a 14. hodina, LS 2024/25

Zuzana Petříčková

31. března a 3. dubna 2025

Co jsme probírali minule

Vrstevnatá neuronová síť (MLP) a algoritmus zpětného šíření (backpropagation)

- Přehled pythonovských knihoven pro hluboké učení, ukázky kódu, návod k instalaci
- Interaktivní vizualizace v Tensorflow Playground
- Stručná analýza modelu, nastavení hyperparametrů v závislosti na typu úlohy – teoreticky i prakticky
- Podrobný popis procesu učení MLP v Kerasu na příkladu binární klasifikace – nestihli jsme dokončit

Tento týden

1. Dokončení ukázky v Kerasu (včetně ladění hyperparametrů)
2. Nastavení hyperparametrů - pokračování
   - Algoritmy učení vrstevnatých neuronových sítí
3. Ukázky různých typů úloh:
   - Binární klasifikace (již bylo), klasifikace do více tříd, regrese, predikce časové řady
   - Specifika jednotlivých úloh a příprava dat

Opakování z minule: Učení krok za krokem

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

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

- **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/

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 nebo RMSE) - pro běžná data
- **mean_absolute_error** (MAE) - pro data s outliery

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

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/

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
→ 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 - bylo za 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 uvedeny 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).

Tento týden

1. Dokončení ukázky v Kerasu (včetně ladění hyperparametrů)
2. **Analýza modelu a nastavení hyperparametrů - pokračování**
   - **Algoritmy učení vrstevnatých neuronových sítí**
3. Ukázky různých typů úloh:
   - Binární klasifikace (již bylo), klasifikace do více tříd, regrese, predikce časové řady
   - Specifika jednotlivých úloh a příprava dat
4. Zobecňování vrstevnatých neuronových sítí a techniky pro prevenci přeučení (včetně ukázek a příkladů)

Problém saturace neuronů

- Jsou-li váhy a biasy moc malé, šíří se sítí příliš malá chyba a učení je moc pomalé.
- Moc velké váhy naopak vedou k tzv. **saturaci** neuronů
  - Neurony jsou hodně aktivní nebo hodně pasivní pro všechny trénovací vzory a nelze je dále učit, protože derivace přenosové funkce je téměř nulová
    $\rightarrow$ **paralýza sítě** a nekontrolovaný růst vah

Jak snížit riziko saturace?

- relu přenosová funkce ve skrytých vrstvách namísto tanh
- normalizace trénovacích dat, další normalizační techniky (batch normalization, layer normalization)
- vhodná inicializace vah a biasů: He inicializace (pro ReLU) nebo Xavier (Glorot) inicializace (pro sigmoid/tanh)
- snížit parametr učení nebo použít algoritmus s adaptivním parametrem učení

Vrstevnaté neuronové sítě - analýza modelu
Rychlost učení a aproximační schopnosti

- Algoritmus zpětného šíření je poměrně pomalý.
- Špatná volba hyperparametrů ho ještě zpomalí.
- Přesto dosahuje často lepších výsledků než mnohé „rychlé algoritmy“
  (hlavně v případě, že má úloha realistickou úroveň složitosti a velikost trénovací množiny přesáhne kritickou mez)

Vrstevnaté neuronové sítě - analýza modelu
Rychlost učení a aproximační schopnosti

Jak urychlit učení a zároveň zajistit dobrou aproximaci

- Vhodná inicializace vah a biasů
- Předzpracování a normalizace vstupních dat
- Vhodná volba parametru učení
- Použití rychlého algoritmu učení
- Současná adaptace parametrů (vah, biasů) i architektury sítě.

Volba vhodného parametru učení

- pro učení je zásadní správné nastavení parametru učení α (learning rate)

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
- adaptivní parametr učení

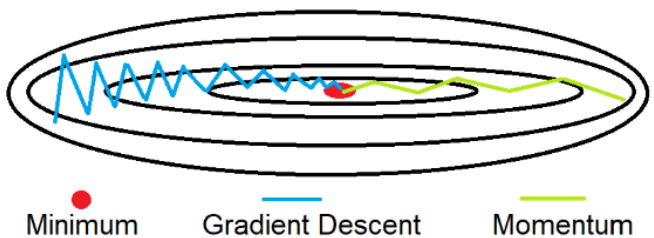
Algoritmus zpětného šíření s momentem

Problém: V úzkých údolích chybové funkce může vést sledování gradientu k náhlým, velkým a častým oscilacím během učení

- Gradient kolísá $\rightarrow$ zpomaluje konvergenci

Řešení: přidání **momentu**

- Kromě aktuální hodnoty gradientu chybové funkce bereme v úvahu i předchozí změny vah
- Efekt: Větší setrvačnost, umožňuje udržovat směr, oslabuje oscilace během učení



https://www.andreaperlato.com/aipost/gradient-descent-with-momentum/

Algoritmus zpětného šíření s momentem

Nové adaptační pravidlo

- Změna váhy hrany z neuronu i do neuronu j v čase (t+1):

$$\begin{aligned} \triangle w_{ij}(t+1) &= -\alpha \frac{\partial E_t}{\partial w_{ij}} + \alpha_m \Delta w_{ij}(t) \\ &= -\alpha \frac{\partial E}{\partial w_{ij}} + \alpha_m (w_{ij}(t) - w_{ij}(t-1)) \end{aligned}$$

- α ... parametr učení
- α_m ... moment učení

Algoritmus zpětného šíření s momentem

Moment učení

$$\triangle w_{ij}(t+1) = -\alpha \frac{\partial E_t}{\partial w_{ij}} + \alpha_m \triangle w_{ij}(t)$$

- Udržuje směr v úzkých údolích chybové funkce
- Snižuje nebezpečí ustálení v nestabilních stavech (lokální minima, sedla)
- Zvyšuje rychlost konvergence (delší úseky beze změny směru přírůstku vah)
- Moc velký α_m - moc velká setrvačnost, mohu přeběhnout minimum chybové funkce

Nesterov momentum

Vylepšení klasického momentu:

- „Nahlíží dopředu“ – počítá gradient v předpokládané nové pozici
- Lepší stabilita a rychlejší konvergence než obyčejný moment

Výpočet změny váhy:

1. Nejprve spočteme přechod do **předpokládané pozice**:

$$\tilde{w}_{ij}(t) = w_{ij}(t) + \alpha_m \cdot \Delta w_{ij}(t-1)$$

2. Pak spočteme gradient chyby vůči této pozici:

$$\Delta w_{ij}(t) = -\alpha \cdot \left.\frac{\partial E}{\partial w_{ij}}\right|_{w=\tilde{w}(t)} + \alpha_m \cdot \Delta w_{ij}(t-1)$$

3. A aktualizujeme váhu:

$$w_{ij}(t+1) = w_{ij}(t) + \Delta w_{ij}(t)$$

Nesterov momentum

Výpočet změny váhy:

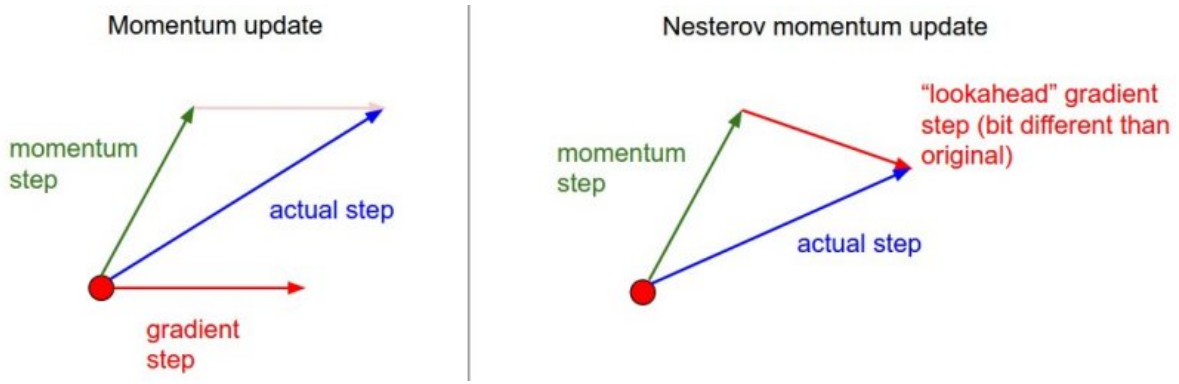
1. Nejprve spočteme přechod do **předpokládané pozice**:

$$\tilde{w}_{ij}(t) = w_{ij}(t) + \alpha_m \cdot \Delta w_{ij}(t-1)$$

2. Pak spočteme gradient chyby vůči této pozici:

$$\Delta w_{ij}(t) = -\alpha \cdot \left.\frac{\partial E}{\partial w_{ij}}\right|_{w=\tilde{w}(t)} + \alpha_m \cdot \Delta w_{ij}(t-1)$$

3. A aktualizujeme váhu: $w_{ij}(t+1) = w_{ij}(t) + \Delta w_{ij}(t)$



https://github.com/cs231n/cs231n.github.io/blob/master/assets/nn3/nesterov.jpeg

Moment vs. Nesterov momentum – Shrnutí

Klasický moment:

- Reaguje až po provedení kroku
- Snazší implementace

Nesterov momentum:

- Předvídá budoucí pozici – přepočítává gradient dřív
- Lepší konvergence v praxi

V Kerasu: oba varianty dostupné v rámci SGD

optimizer = SGD(learning_rate=0.01, momentum=0.9, nesterov=True)

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

Nastavení parametrů učení

- **Parametr učení** α
  - Malý $\alpha \rightarrow$ pomalé učení, riziko uváznutí v lokálním minimu
  - Velký $\alpha \rightarrow$ rychlé učení, ale riziko oscilací a nestability
- **Moment** α_m
  - Pomáhá překonávat plošiny a stabilizuje učení ve strmých oblastech
  - Příliš velký $\rightarrow$ síť může přeběhnout přes minimum
  - Typicky volíme $0.8 \leq \alpha_m \leq 0.95$
- **Použít Nesterov momentum?**
  - **Ano, pokud:** model je hlubší, terén chybové funkce složitější
  - Užitečné tam, kde klasický moment osciluje nebo zpomaluje
  - Vhodné jako výchozí volba u SGD v moderních knihovnách

Problém: Správné nastavení těchto parametrů bývá obtížné a závisí na konkrétní úloze.

Řešení: Adaptivní řízení parametru učení

Adaptivní řízení parametru učení

- existují různé strategie, od jednoduchých heuristik po sofistikované metody

Primitivní heuristika (již známe):

- Parametr učení by měl klesat s rostoucím počtem epoch
- **Počáteční parametr učení** $0 << \alpha < 1$
  - Umožňuje přeskočit mělká lokální minima
  - Rychlý vývoj vah sítě
- **Závěrečný parametr učení** $\alpha \sim 0$
  - Zabraňuje oscilacím
  - Neměl by klesat moc rychle, stačí:
    $\sum_{t=0}^{\infty} \alpha_t = \infty$,
    $\sum_{t=0}^{\infty} \alpha_t^2 < \infty$

Adaptivní parametr učení

Lokální parametr učení pro každou váhu

- Změna váhy z neuronu i do neuronu j v čase (t+1):

$$\triangle w_{ij}(t+1) = -\alpha_{ij,t+1}\frac{\partial E_t}{\partial w_{ij}} + \alpha_m \triangle w_{ij}(t)$$

Resilent propagation (Rprop) - Silva & Almeida

Algoritmus založený na změně znaménka parciální derivace:

- Inicializuj $\alpha_{ij,0}$ malými náhodnými hodnotami
- Zrychluj učení, pokud se za poslední dvě po sobě jdoucí iterace nezměnilo znaménko parciální derivace $\frac{\partial E}{\partial w_{ij}}$
- Zpomaluj, pokud se znaménko změnilo

Více variant:

- základní (Silva & Almeida)
- Super SAB
- Rprop+
- ...

Resilent propagation (Rprop) - Silva & Almeida

Adaptace parametru učení v čase (t+1)

- $\alpha_{ij,t+1} = u \cdot \alpha_{ij,t}$, jestliže $\frac{\partial E_t}{\partial w_{ij}} \cdot \frac{\partial E_{t-1}}{\partial w_{ij}} > 0$
- $\alpha_{ij,t+1} = d \cdot \alpha_{ij,t}$, jestliže $\frac{\partial E_t}{\partial w_{ij}} \cdot \frac{\partial E_{t-1}}{\partial w_{ij}} < 0$
- Konstanty u, d jsou pevně zvolené tak, že $u > 1$, $d < 1$

Problémy

- Parametr učení roste i klesá exponenciálně vzhledem k u a d
  $\rightarrow$ Problémy mohou nastat, jestliže po sobě následuje mnoho urychlovacích kroků.

Resilent propagation (Rprop) - Super SAB

Algoritmus

- Nastav všechny α_{ij}^0 na počáteční hodnotu α_{start}.
- Proveď krok (t) algoritmu zpětného šíření s momentem.
- Pokud $\frac{\partial E_t}{\partial w_{ij}} \cdot \frac{\partial E_{t-1}}{\partial w_{ij}} > 0$: $\alpha_{ij,t+1} = u \cdot \alpha_{ij,t}$.
- Pokud $\frac{\partial E_t}{\partial w_{ij}} \cdot \frac{\partial E_{t-1}}{\partial w_{ij}} < 0$:
  - Anuluj předchozí změnu vah: $w_{ij}(t+1) = w_{it}(t) - \triangle w_{ij}(t)$
  - Zmenši parametr učení: $\alpha_{ij,t+1} = d \cdot \alpha_{ij,t}$

Vlastnosti

- Řádově rychlejší než standardní algoritmus zpětného šíření
- Poměrně stabilní
- Robustní k volbě počátečních parametrů

Resilent propagation - Rprop+

- Parametr učení není řízený znaménkem parciální derivace chybové funkce, ale změnou velikosti chyby
- Rychlejší než Super SAB

Algoritmus

- Nastav všechny α_{ij}^0 na počáteční hodnotu α_{start}.
- Proveď krok (t) algoritmu zpětného šíření s momentem.
- Pokud $E_t < E_{t-1}$: $\alpha_{ij,t+1} = u \cdot \alpha_{ij,t}$.
- Pokud $E_t > c \cdot E_{t-1}$:
  - Anuluj předchozí změnu vah: $w_{ij}(t+1) = w_{it}(t) - \triangle w_{ij}(t)$
  - Zmenši parametr učení: $\alpha_{ij,t+1} = d \cdot \alpha_{ij,t}$
- Konstanty c, u, d jsou pevně zvolené tak, že $c > 1$, $u > 1$, $d < 1$

Moderní optimalizační algoritmy

Základ:

- **SGD (Stochastic Gradient Descent)** – základní algoritmus, vylepšený o setrvačnost (**Momentum / Nesterov momentum**)

Moderní algoritmy v knihovnách jako Keras, PyTorch:

- **RMSprop** – adaptivní parametr učení, vhodný pro rekurentní modely (RNN)
- **Adam (Adaptive Moment Estimation)** – dnes nejčastěji používaný
- **NAdam** – Adam + Nesterov momentum
- a mnoho dalších (**AdaGrad, Adadelta, AdaMax, FTRL,...**)

Typický je pro ně rychlejší učení díky adaptivnímu parametru učení (a menší citlivost vůči volbě jejich parametrů)

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

RMSprop – Root Mean Square Propagation

Myšlenka:

- V různých směrech může být chybová funkce různě „prudká“ („samá díra a výmol”) nebo „plochá“
- Nechceme ve všech směrech používat stejný krok
- RMSprop upravuje krok zvlášť pro každou váhu podle toho, jak moc se v daném směru mění gradient
- Chová se jako **adaptivní tlumič** v divokém terénu

Jak to algoritmus pozná?

- Sleduje se klouzavý průměr čtverců gradientů pro každou váhu:

$$v_{ij}(t) = \beta \cdot v_{ij}(t-1) + (1-\beta) \cdot \left(\frac{\partial E_t}{\partial w_{ij}} \right)^2$$

RMSprop – Root Mean Square Propagation

Jak to algoritmus pozná?

- Sleduje se klouzavý průměr čtverců gradientů pro každou váhu:

$$v_{ij}(t) = \beta \cdot v_{ij}(t-1) + (1-\beta) \cdot \left(\frac{\partial E_t}{\partial w_{ij}} \right)^2$$

- Pokud jsou gradienty často velké $\rightarrow$ krok se zmenší (zpomalení učení)
- Pokud jsou gradienty malé $\rightarrow$ krok zůstane větší (zrychlení učení)

Aktualizace váhy:

$$\Delta w_{ij}(t) = -\frac{\alpha}{\sqrt{v_{ij}(t)} + \varepsilon} \cdot \frac{\partial E_t}{\partial w_{ij}}$$

$$w_{ij}(t+1) = w_{ij}(t) + \Delta w_{ij}(t)$$

RMSprop – Root Mean Square Propagation

Výsledek:

- Stabilnější učení, zejména u sítí s různými měřítky gradientu (např. RNN)
- Méně závislé na ručním ladění parametru učení α

Využití:

- Často používaný pro rekurentní neuronové sítě (RNN)
- Dobře zvládá nestacionární gradienty

Adam – Adaptive Moment Estimation

Princip:

- Kombinuje výhody momentu (1. moment) a RMSprop (2. moment)
- Odhad průměru a rozptylu gradientů (exponenciálně klouzavé průměry)
- Sleduje:
  - $m_{ij}(t)$ – klouzavý průměr gradientů (odhad 1. momentu)
  - $v_{ij}(t)$ – klouzavý průměr čtverců gradientů (odhad 2. momentu)
- Oprava zkreslení na začátku (bias correction)

Adam – Adaptive Moment Estimation

Aktualizace vah:

$$\Delta w_{ij}(t) = -\alpha \cdot \frac{\hat{m}_{ij}(t)}{\sqrt{\hat{v}_{ij}(t)} + \varepsilon} \qquad w_{ij}(t+1) = w_{ij}(t) + \Delta w_{ij}(t)$$

Kde:

$$g_{ij}(t) = \frac{\partial E_t}{\partial w_{ij}}$$

$$m_{ij}(t) = \beta_1 \cdot m_{ij}(t-1) + (1-\beta_1) \cdot g_{ij}(t)$$

$$v_{ij}(t) = \beta_2 \cdot v_{ij}(t-1) + (1-\beta_2) \cdot g_{ij}^2(t)$$

$$\hat{m}_{ij}(t) = \frac{m_{ij}(t)}{1-\beta_1^t}, \quad \hat{v}_{ij}(t) = \frac{v_{ij}(t)}{1-\beta_2^t}$$

Typické hodnoty: $\alpha = 0.001$, $\beta_1 = 0.9$, $\beta_2 = 0.999$, $\varepsilon = 10^{-7}$

Adam – Adaptive Moment Estimation

- **Univerzální a spolehlivý algoritmus** – funguje dobře ve většině situací bez složitého ladění parametrů
- **Kombinuje výhody:**
  - momentu (hladší a stabilnější změny vah)
  - adaptivního kroku (RMSprop)
- Standardní výchozí volba v knihovnách jako Keras nebo PyTorch
- Pro některé typy úloh (např. konvexní tvar chybové funkce) tíhne k suboptimálním řešením, pak je lepší sáhnout po SGD, RMSprop

Keras:
optimizer = Adam(learning_rate=0.001, beta_1=0.9, beta_2=0.999, epsilon=1e-7)

Nadam – Nesterov-accelerated Adam

Kombinuje výhody:

- Adam (adaptivní učení + moment)
- Nesterov momentum (náhled do budoucí pozice)

Efekt:

- Rychlejší a stabilnější konvergence než běžný Adam
- Vhodné pro hluboké sítě a složité úlohy

Shrnutí moderních optimalizátorů

Algoritmus	**Výhody**	**Kdy použít**
SGD	jednoduchý, přehledný	malé modely, ladění
SGD + Momentum	rychlejší konvergence	hlubší sítě
RMSprop	adaptivní krok, stabilní	RNN, sekvenční data
Adam	robustní, bez ladění	univerzální volba
Nadam	ještě stabilnější	hluboké modely

Další finty pro zlepšení učení

Výpočet lze pustit opakovaně

- pro různé počáteční váhy, vyberu síť s nejmenší výslednou chybou

Pokud síť nekonverguje nebo konverguje velmi pomalu

- zkusit více neuronů ve vrstvách, popř. více vrstev

Častější předkládání důležitých vzorů

- snížení chyby pro důležité vzory

Další finty pro zlepšení učení

'Žíhání sítě' = přidání náhodného šumu

- při ustálení vah a biasů s vysokou chybou
- Adaptace váhy hrany z neuronu i do neuronu j podle:

$$w_{ij}(t+1) = w_{ij}(t) + N(0, \epsilon)$$

- Parametr ϵ je třeba volit opatrně:
  - moc malé ϵ ... vrátí se zpět
  - moc velké ϵ ... musí se znova dlouho adaptovat

Praktické ukázky různých typů úloh

- binární klasifikace: Breast Cancer (již bylo)
- klasifikace do více tříd: MNIST
- regresní úloha: Wine quality
- Predikce časové řady: Denní minimální teploty v Melbourne

Ukázka binární klasifikace: Breast Cancer

- již bylo minule

Nastavení modelu

- **sigmoid** přenosová funkce ve výstupní vrstvě
- **ReLU** (nebo **tanh**) přenosová funkce ve skrytých vrstvách
- chybová funkce **BinaryCrossentropy**, metriky **Accuracy** (přesnost), **Precision**, **Recall**,...

Ukázka úlohy klasifikace do více tříd: MNIST

- 60000 obrázků ručně psaných číslic (odstíny šedé, 28x28)
- obrázky jsou vycentrované a všechny číslice mají cca. stejnou velikost
- 10000 testovacích obrázků číslic napsaných jinými lidmi
- výstupní příznak: číslo (číslice) 0,...,9 (10 tříd)
- vlastnosti dat:
  - všechny obrázky mají stejnou velikost $\rightarrow$ nemusíme jejich velikost standardizovat
  - vstupní data jsou 3D $\rightarrow$ musíme je vektorizovat
  - vstupní příznaky nabývají hodnot 0...255 $\rightarrow$ musíme je normalizovat do intervalu $[0, 1]$ nebo $[-1, 1]$

Ukázka úlohy klasifikace do více tříd: MNIST

Nastavení modelu

- **softmax** přenosová funkce ve výstupní vrstvě
- **ReLU** (nebo **tanh**) přenosová funkce ve skrytých vrstvách
- chybová funkce **SparseCategoricalCrossentropy**, metrika **SparseCategoricalAccuracy** (přesnost) (pokud labely jsou čísla)
- chybová funkce **CategoricalCrossentropy**, metrika **CategoricalAccuracy** (přesnost) (pokud labely jsou one-hot vektory)

Pozorování

- přesnost na testovacích datech je okolo 85%
- chyby na trénovací a validační množině jsou podobné $\rightarrow$ model dobře zobecňuje
- přesnost je možné ještě zlepšit, pokud zvýšíme parametr učení, zvýšíme počet epoch, změníme učící algoritmus apod.

Ukázka regresní úlohy: Wine Quality data

- 11 číselných vstupních příznaků (např. kyselost, obsah alkoholu, síry,...)
- 1 výstupní příznak – kvalita vína (hodnocení 0–10, většinou 3–8)

Vlastnosti dat:

- Příznaky mají různé rozsahy hodnot $\rightarrow$ normalizace (např. pomocí StandardScaler)
- Vzorků je dostatek (4900)
  1. lze trénovat i větší model bez okamžitého rizika přeučení
  2. možnost rozdělit data na trénovací, validační a testovací množinu

Ukázka regresní úlohy: Wine Quality data

Nastavení modelu pro regresi

- ReLU (nebo tanh) aktivační funkce ve skrytých vrstvách
- Lineární aktivační funkce (identita) ve výstupní vrstvě
- Chybová funkce: **MeanSquaredError (MSE)**
- Metriky: **MeanAbsoluteError (MAE)**, **RootMeanSquaredError (RMSE)**

Ukázka regresní úlohy: Wine Quality data

Pozorování:

- Model se lépe naučí hodnoty ze středu intervalu než z okrajů (kde je málo dat) → řešení: oversampling (častější předkládání daných vzorů)
- Bez early stopping můžeme sledovat přeučení (obzlášť pro větší architekturu)
- Výkon modelu může být citlivý na normalizaci vstupních příznaků

Shrnutí

1. Pro regresní úlohy volíme lineární výstupní vrstvu a chybovou funkci MSE.
2. Příznaky s různým rozsahem hodnot je potřeba normalizovat – zlepšuje učení modelu.
3. Kromě MSE lze použít další metriky jako MAE pro lepší interpretaci výsledků.

Jiný přístup: klasifikace kvality vína

Stejný dataset, jiný přístup: Předpověď kvality vína jako klasifikační úloha místo regrese.

- Kvalitu (0–10) převedeme na třídy, např.:
  - **0–4**: nízká kvalita
  - **5–6**: střední kvalita
  - **7–10**: vysoká kvalita

Pokud bychom změnili regresní model na klasifikační, jak se změní výstupní vrstva, loss, metriky?

Jiný přístup: klasifikace kvality vína

Nastavení modelu pro klasifikaci

- Výstupní vrstva: 3 neurony se **softmax** aktivací
- Chybová funkce: **categorical_crossentropy** (nebo **sparse_categorical_crossentropy**, pokud používáme číselné třídy)
- Metriky: **accuracy**, případně **precision**, **recall**, **F1-score**

Poznámka:

- Rozložení tříd je nevyvážené → je vhodné sledovat víc metrik než jen přesnost.

Srovnání klasifikace a regrese:

- Srovnání klasifikace a regrese: která formulace dává v tomto konkrétním případě větší smysl?

Ukázka úlohy s časovou řadou: Min. denní teploty

- Dataset: Denní minimální teploty v Melbourne, roky 1981–1990
- Cíl: předpovědět teplotu následujícího dne na základě předchozích hodnot

Vlastnosti dat:

- 1 příznak (časová řada – teplota)
- Více než 3600 záznamů – dostatečné pro učení i testování
- Nejde o nezávislé vzory – potřebujeme zachytit časovou závislost

Použití posuvného okénka (sliding window):

- Pro každý vzor použijeme např. posledních 10 dní k predikci následujícího
- Tím vznikne tabulková reprezentace vhodná i pro MLP

Min. denní teploty – model a nastavení

Způsob přípravy vstupu:

- Výroba trénovací, validační a testovací množiny pomocí posuvného okénka (pro každou sadu jiné časové okno!)
- Normalizace vstupních hodnot (např. MinMaxScaler)

Nastavení modelu:

- Vstupní vrstva: počet neuronů odpovídá délce okénka (např. 10)
- Skryté vrstvy s ReLU / tanh
- Výstupní vrstva s lineární přenosovou funkcí (1 hodnota – předpověď teploty)
- Chybová funkce: MSE, metriky: MSE / MAE / RMSE

Pozorování:

- Abychom překonali referenční model, je třeba MLP vyladit
- Volba délky okénka ovlivňuje kvalitu predikce
- Model se může přeučit, pokud použijeme příliš komplexní architekturu

Shrnutí: Časová řada a metoda posuvného okénka

1. Časové řady neobsahují nezávislé vzory – při vytváření trénovací, validační a testovací množiny musíme respektovat časový kontext.
2. Metoda posuvného okénka umožní převést časovou řadu na trénovací množinu pro MLP.
3. Délka okénka (počet vstupních hodnot) je důležitý hyperparametr.
4. Na výstupní vrstvě používáme lineární funkci, jako u regrese.
5. Predikce časových řad lépe zvládají specializované architektury (RNN, LSTM), ale MLP s metodou posuvného okénka je jednoduchý a srozumitelný start.

Shrnutí: Časová řada a metoda posuvného okénka

Možnosti rozšíření

- Vstupem modelu nemusí být pouze minulé hodnoty výstupu, ale i další hodnoty (např. jiné naměřené hodnoty - tlak vzduchu, vlhkost,...)
- Predikovat nemusíme jen jedno číslo (následující hodnota), ale hodnotu za delší časový interval / více hodnot