Neuronové sítě a Deep Learning

Pokyny k online schůzce

1. Používejte nejlépe desktopovou aplikaci Teams.

2. Pokud nekladete dotaz nebo se neúčastníte diskuze, mějte prosím vypnutý mikrofon.

3. Pokud jen trochu můžete, mějte puštěnou kameru. Vás výraz tváře pomůže vyučujícímu :)

4. Jak můžete položit dotaz:
   a. Zapněte si mikrofon a rovnou se zeptejte.
      nebo:
   b. Napište dotaz do chatu. nebo:
   c. Použijte tlačítko zvednout ruku.
      (Po vyvolání ruku sundejte)

Agenda

- Deep Learning
- Koncept neuronových sítí
- Perceptron
- Trénování neuronových sítí
- Aktivační funkce
- Reprezentace dat pro neuronové sítě
- Geometrická interpretace tenzorových operací
- Regresní úlohy
- Klasifikační úlohy
- Keras a TensorFlow

Instalace Tensorflow

- pip install tensorflow
  - (instalační soubor má cca 370 MB a nějakou dobu to trvá)
  - pokud máte GPU Nvidia, můžete nainstalovat tensorflow-gpu
    - pip install tensorflow-gpu
    - pro podporu GPU je třeba doinstalovat řadu dalších knihoven
      - https://www.tensorflow.org/install/gpu
      - (jde to rozchodit i na Windows)
- pokud nemáte k dispozici dostatečně výkonný počítač, můžete použít Google Colab
  - https://colab.research.google.com/
  - online hostovaná varianta Jupyter Notebooku
  - vytvoříte si notebook stejně jako lokálně u sebe
  - pro GPU vyberete Runtime - Change Runtime Type - GPU, víc není potřeba nastavovat

Deep Learning

Deep Learning (“hluboké učení”)

- podskupina strojového učení, kde modely jsou tvořeny z více na sebe navazujících vrstev
- každá následující vrstva se učí smysluplnější reprezentace z dat (“destilace informací”)
- příklad: CNN (konvoluční neuronová síť)
  - první vrstva pracuje na úrovni pixelů a zpracovává informace jako jsou např. hrany v obrázku
  - další vrstva skládá hrany např. do čtverců a jiných útvarů
  - každá další vrstva tedy pracuje se stále více abstraktními informacemi (textury, ..., objekty), přičemž jako vstup používá výstup předchozí vrstvy
- hloubka nepředstavuje “hlubší porozumění” problému, ale myšlenku postupných vrstev reprezentací
- Deep Learning má dosud průlomové výsledky například v těchto oblastech:
  - klasifikace obrazu (téměř na úrovni člověka)
  - rozpoznávání řeči (téměř na úrovni člověka)
  - přepis ručně psaného textu
  - výrazné zdokonalení strojového překladu (Google Translate - LSTM rekurentní neuronová síť)
  - autonomní řízení vozidel (téměř na úrovni člověka)
  - vylepšené cílení reklam a vyhledávacích výsledků
  - nadlidská schopnost hraní různých her
  - generování textu, Q&A (téměř na úrovni člověka - OpenAI GPT-3 - Transformer)

Koncept neuronových sítí

Úvod do neuronových sítí

- inspirací pro vznik původních umělých neuronových sítí (ANN - artifical neural network) jsou neurony v biologickém mozku
  - postupným vývojem se však ANN zcela vzdálily od biologického neuronu a řada autorů se dnes snaží této analogii vyhýbat
    - místo výrazu “neuron” pak používají “unit” (výpočetní jednotka)
  - pojem “neuronová síť” budeme dále chápat jako ekvivalent k ANN
- neuronové sítě jsou základním stavebním kamenem Deep Learningu
  - univerzálně použitelné
  - výkonné
  - škálovatelné
- typické příklady použití
  - klasifikace miliard obrázků (Google Images)
  - rozpoznávání hlasu (hlasoví asistenti apod.)
  - doporučovací systémy pro stovky milionů uživatelů (Youtube, Netflix)
  - herní agenti (DeepMind Alpha-Zero)
  - ...

Historie neuronových sítí

- koncept neuronových sítí je poměrně starý: objevil se již ve 40. letech minulého století
  - po prvotním nadšení o ANN poklesl zájem vzhledem k malým výsledkům v praxi
  - takto se to několikrát opakovalo v průběhu dalších desítek let
  - na počátku 90. let se objevilo SVM a další výkonné ML algoritmy a ANN opět stály stranou

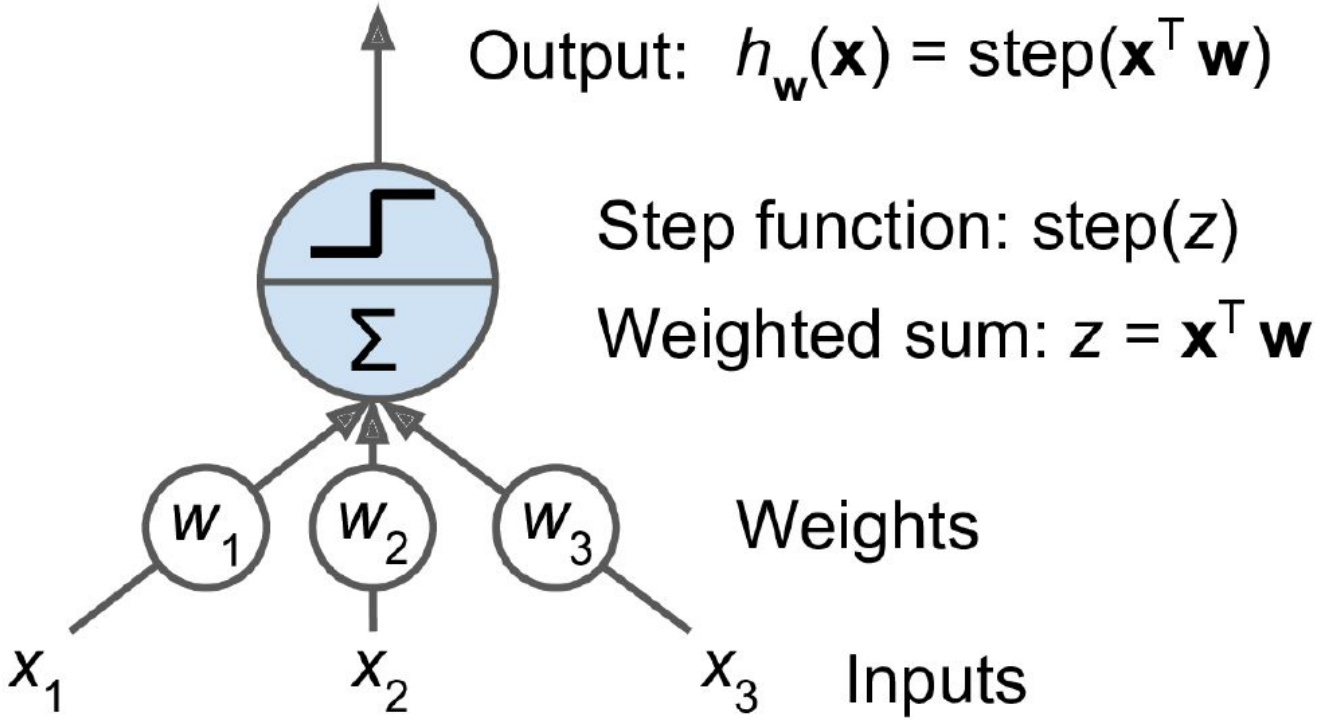
- poslední vlna zájmu o ANN (cca 2000+) je však jiná:
  - obrovské množství dostupných dat, což je pro neuronové sítě klíčové
    - na velkých a komplexních datasetech neuronové sítě excelují v porovnání s jinými ML alg.
  - obrovský vzestup dostupného výpočetního výkonu umožňuje, aby se i složité sítě daly natrénovat v rozumném čase
    - hlavní zásluhu na tomto nese herní průmysl, protože průlom způsobily moderní GPU čipy
  - zlepšily se algoritmy pro trénování neuronových sítí
    - ačkoliv se jedná o drobné změny, mají zásadní vliv na výsledek
  - některé z teoretických omezení neuronových sítí se v praxi ukázaly jako nepříliš významné
    - uvíznutí v lokálním minimu při trénování (důsledek použití metody Gradient Descent) se příliš často nestává, nebo je toto minimum blízko globálního minima
  - moderní neuronové sítě plní titulky novin
    - trvale o ně roste zájem i z hlediska dalších investic a rozvoje

Od biologických neuronů k umělým neuronům

- proč “neuronová” síť:
  - biologický neuron přijímá skrz synapse krátké elektrické impulsy od ostatních neuronů a když získá dostatečné množství signálů během několika milisekund, tak vyšle sám svůj signál dalším neuronům
  - síť miliard propojených (jednoduchých) neuronů je schopná řešit komplexní úlohy
- umělý neuron
  - obecně: výpočetní jednotka, která má jeden nebo více vstupů a jeden výstup
- základní architektura ANN: Perceptron
  - postavena na umělých neuronech typu TLU (threshold logic unit)
    - jeden nebo více číselných vstupů, kdy každý vstup má přiřazenou váhu, kterou je násoben
    - uvnitř TLU je spočtena vážená suma těchto vstupních hodnot ($z = w_1x_1 + w_2x_2 + \dots + w_nx_n = x^Tw$)
    - na tuto sumu je aplikována aktivační funkce a její výsledek je pak vrácen jako výstup TLU
      - aktivační funkce v perceptronu: jednotkový skok (step function)
        - $H_p(x) = 0$ pokud $z < 0$
        - $H_p(x) = 1$ pokud $z \geq 0$

Perceptron

Perceptron



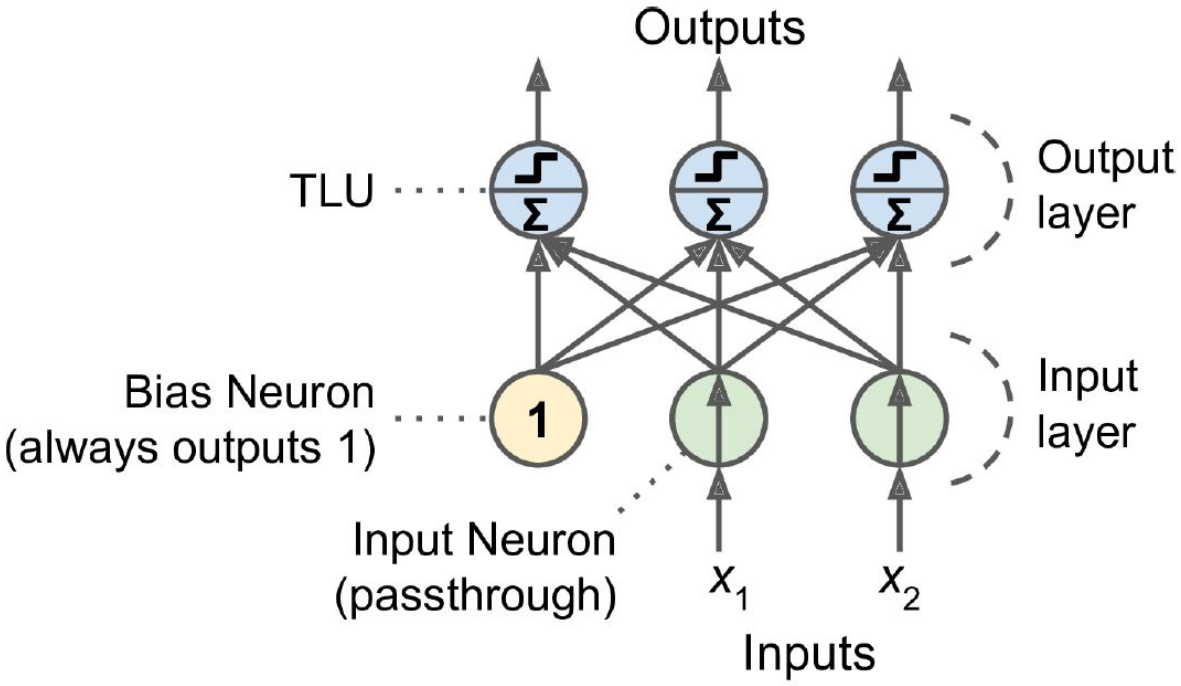
Zdroj: (Géron 2019)

Perceptron

- jedna výpočetní jednotka typu TLU je schopna lineární binární klasifikace
  - spočte lineární kombinaci vstupů a pokud výsledek překročí určitou hranici, vrátí pozitivní třídu
    - stejně jako logistická regrese nebo lineární SVM
  - trénování TLU tedy znamená nalezení správných hodnot vah pro jednotlivé vstupy
- perceptron je tvořen jednou vrstvou jednotek TLU, kdy každá jednotka je napojena na všechny vstupy
  - hustě propojená vrstva (dense layer) - všechny neurony v této vrstvě jsou propojeny se všemi neurony z předchozí vrstvy (resp. vstupy v případě jediné vrstvy)
  - každá vrstva má kromě vstupů z předchozí vrstvy přidaný ještě bias neuron (má konstantní hodnotu 1)
  - viz obrázek

Perceptron

- perceptron na obrázku dokáže klasifikovat do tří různých tříd na základě dvou vstupů (features)
- např. dataset Iris se zvolenými dvěma features a třemi třídami



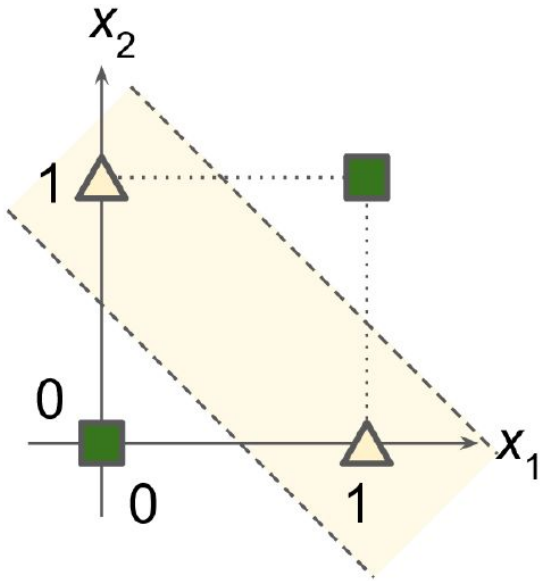
Zdroj: (Géron 2019)

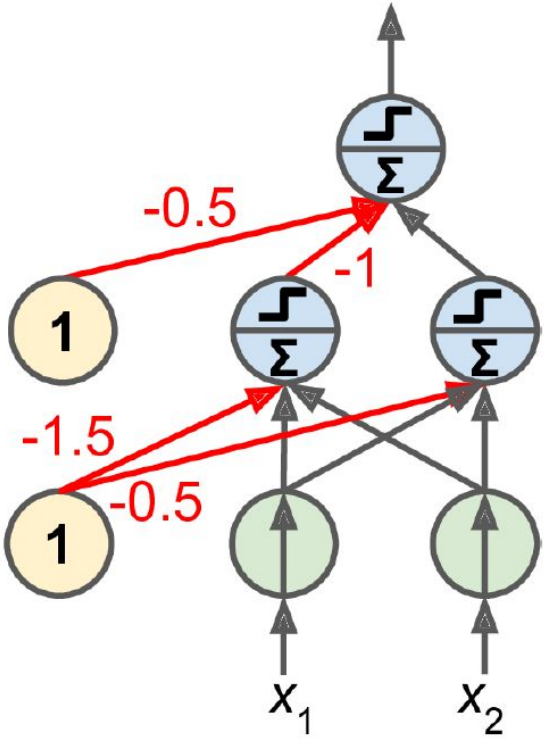
Perceptron vs. vícevrstvý perceptron (MLP)

- perceptron má určitá omezení:
  - obsahuje jen jednu vrstvu neuronů
    - rozhodovací hranice každého neuronu je ale lineární, takže si neporadí s komplexními nelineárními datasety (stejně jako logistická regrese nebo jiné lineární algoritmy)
  - vzhledem ke skokové aktivační funkci (0 nebo 1) nedokáže vrátit pravděpodobnost, ale jen “tvrdou” hodnotu (oproti např. logistické regresi)

- vícevrstvý perceptron MLP (Multi-Layer Perceptron)
  - výpočetní jednotky TLU jsou ve více vrstvách nad sebou
  - díky tomu lze řešit i nelineární úlohy, viz obrázek

Příklad vícevrstvého perceptronu



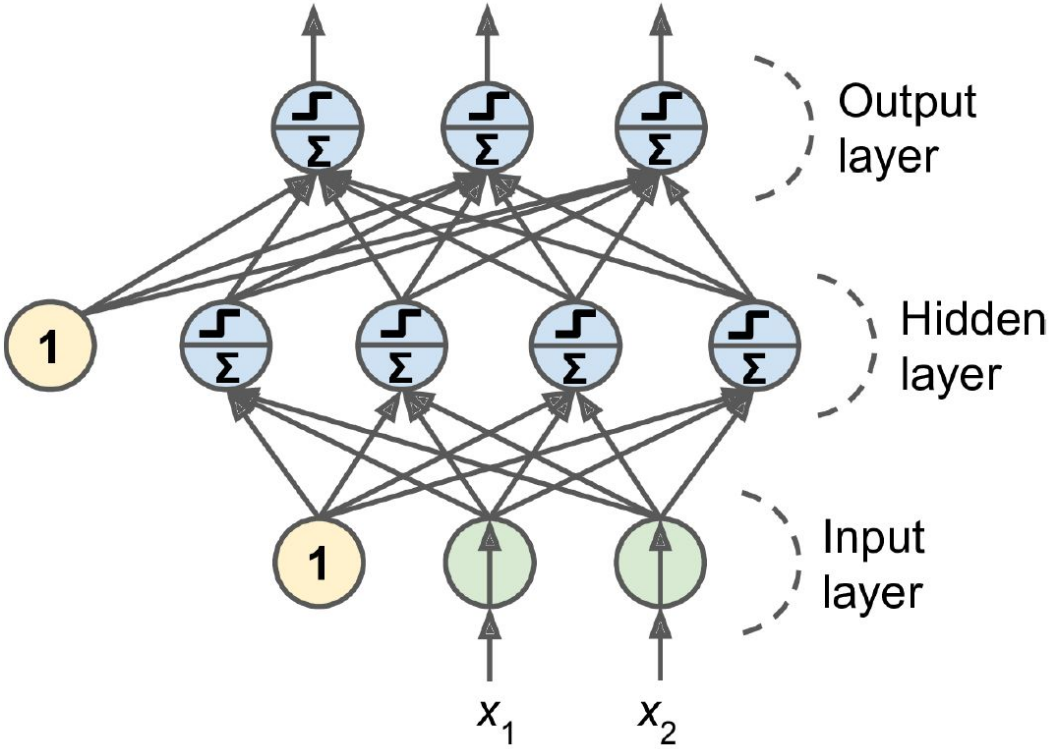


Zdroj: (Géron 2019)

Vícevrstvý perceptron

- obecná struktura:
  - vstupní vrstva (input layer)
    - pouze předává hodnoty příznaků
  - jedna nebo více vrstev jednotek TLU
    - tzv. skryté vrstvy (hidden layers)
  - výstupní vrstva jednotek TLU (output layer)
- signál putuje pouze jedním směrem, od vstupů k výstupům
  - dopředná neuronová síť (FNN - Feedforward Neural Network)
- pokud síť obsahuje hluboký stoh vrstev, jedná se o hlubokou neuronovou síť => Deep Neural Network => Deep Learning
  - Deep Learning se tedy zaměřuje na hluboké neuronové sítě, resp. obecně na modely s více výpočetními vrstvami

Vícevrstvý perceptron obecně



Zdroj: (Géron 2019)

Trénování neuronových sítí

Trénování vícevrstvého perceptronu

- algoritmus zpětného šíření chyby (backpropagation)
  - metoda využívající Gradient Descent
    - připomenutí - podle parciálních derivací ztrátové funkce se upravují hodnoty jednotlivých vah tak, aby se hodnota ztrátové funkce v postupných krocích snižovala
  - získání gradientů (parciálních derivací), tedy jakým směrem se při úpravách parametrů vydat, je klíčovou schopností tohoto algoritmu
    - stačí jeden dopředný a jeden zpětný chod sítí k tomu, aby algoritmus získal gradient pro každý jeden parametr sítě
    - tedy dokáže zjistit, jak má být váha každého spojení upravena, aby se snížila chyba sítě

Backpropagation

- nastaví se náhodné výchozí hodnoty vah (parametrů) v síti
- zpracovává mini-dávku v jednom kroku (mini-batch, velikost např. 32 instancí)
  - celý trénovací dataset tak projde v několika krocích podle velikosti dávky
  - jeden průchod celým datasetem = jedna epocha
  - celým datasetem prochází několikrát - typicky jsou tedy řádově desítky epoch
- dávka je poslána na vstupní vrstvu, která ji jen přepošle do první skryté vrstvy
- spočítá se výstup všech neuronů v této vrstvě (pro každou instanci v dávce) a výstup je poslán do další skryté vrstvy
  - takto se pokračuje, dokud se nedostaneme až na výstupní vrstvu
  - = dopředný chod sítí
- spočítá se výstupní chyba sítě (pomocí ztrátové funkce - MSE, crossentropy, ...)
- spočítá se, jak moc které výstupní spojení přispělo k chybě
  - pomocí “řetízkového pravidla” (pravidlo o derivaci složené funkce)
- spočítá se, jak moc které spojení v nižší vrstvě přispělo k chybě
  - opět pomocí řetízkového pravidla
  - opakuje se to tak dlouho, dokud se nedojde ke vstupní vrstvě
  - = zpětný chod sítí (výstupem je tedy gradient chyby pro všechny spojení, tj. váhové parametry, v síti)
- provede se krok metody Gradient Descent, tedy upravení hodnot vah podle spočtených gradientů

Aktivační funkce

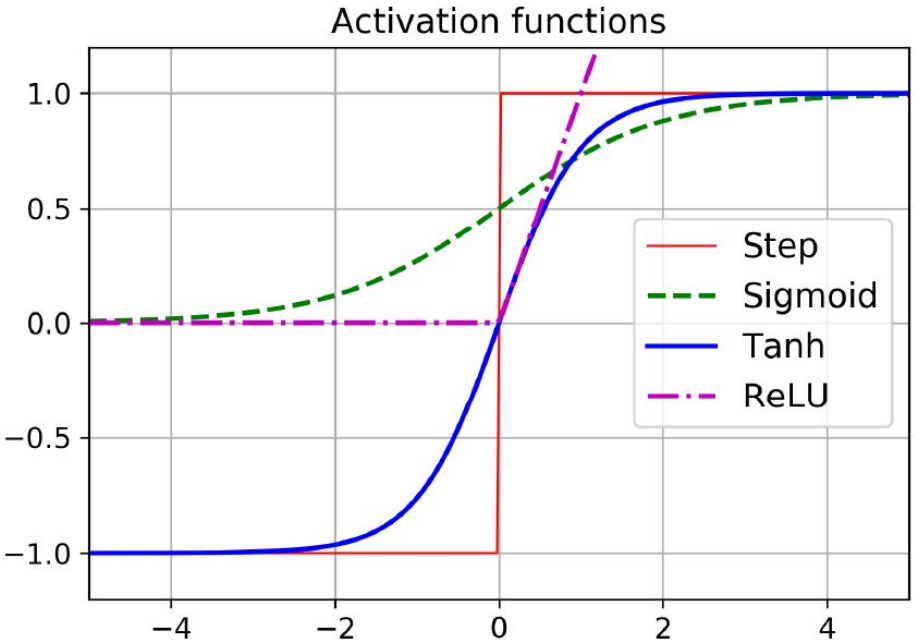
Aktivační funkce - proč je potřeba?

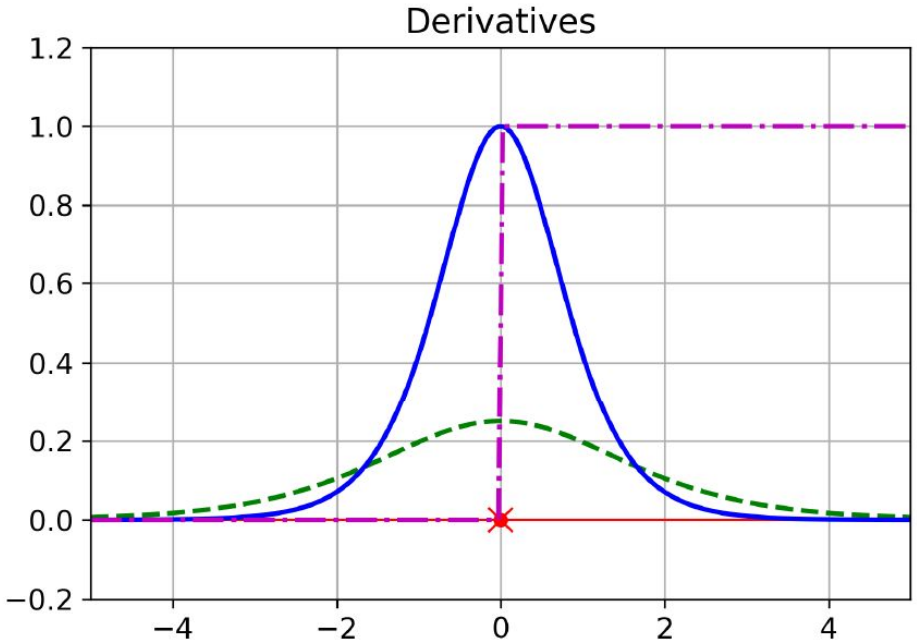
- bez aktivační funkce by každý neuron vracel pouze lineární kombinaci svých vstupů
- pokud bychom takto spojili řadu neuronů, dostali bychom zase pouhou lineární kombinaci vstupů
- síť by pak nedokázala řešit nelineární problémy => potřebujeme do ní vnést nějaký nelineární prvek, což je právě aktivační funkce

Typy aktivační funkce

- aby mohl algoritmus backpropagation správně fungovat, musí být aktivační funkce dobře derivovatelná
  - což původní step funkce (jednotkový skok) není, proto byla nahrazena logistickou funkcí (sigmoid)
- kromě logistické funkce se (dnes častěji) používají jiné aktivační funkce:
  - tanh
    - hyperbolický tangens
    - křivka ve tvaru S stejně jako logistická funkce, ale s rozsahem hodnot -1 až 1 (oproti 0 až 1)
    - výstupy jsou pak centrovány okolo 0, což v praxi urychluje konvergenci k řešení
  - relu
    - REctified Linear Unit function
    - lomená křivka (v nule), proto v nule není diferencovatelná, což může způsobit “poskakování” metody Gradient Descent
    - v praxi ale funguje velmi dobře a je na rozdíl od ostatních velmi rychlá na výpočet

Aktivační funkce





Zdroj: (Géron 2019)

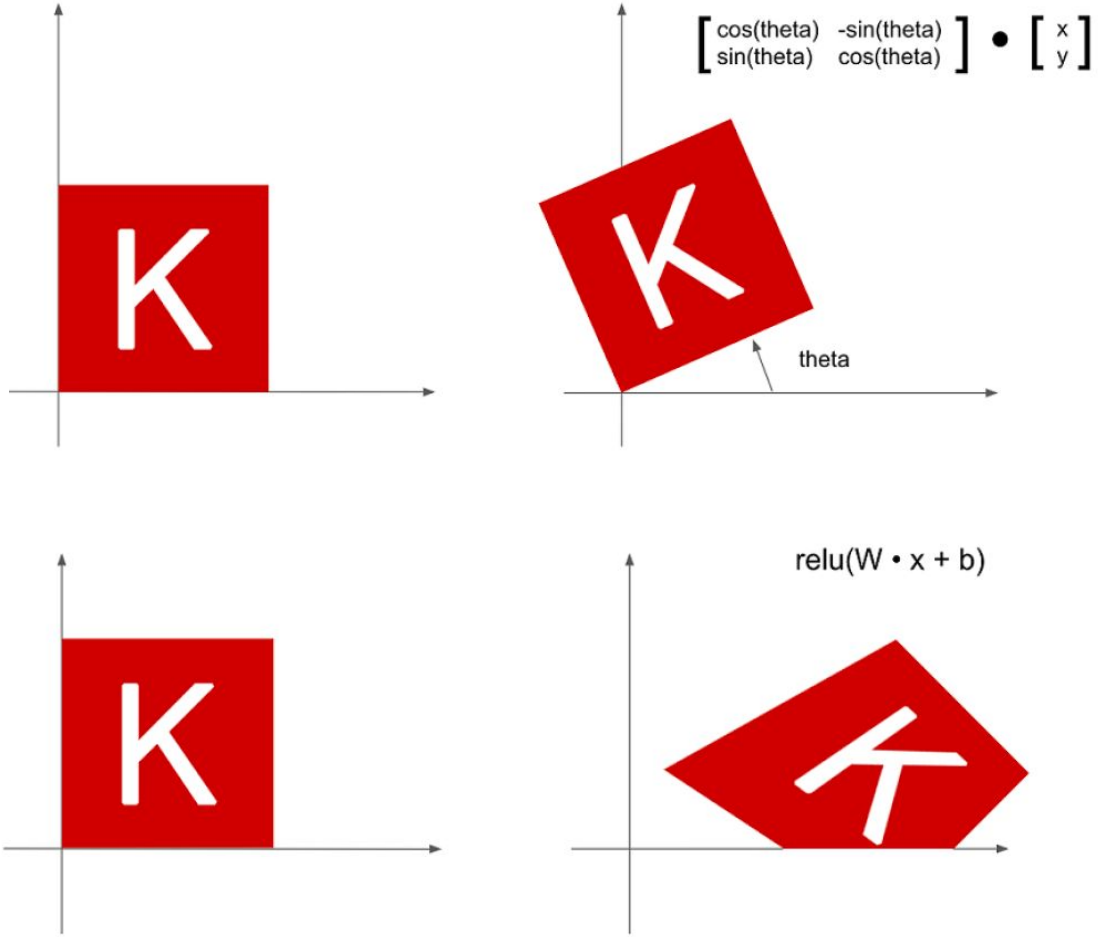
Reprezentace dat pro neuronové sítě

Data pro neuronové sítě

- neuronové sítě dokáží pracovat jen s číselnými hodnotami, které by navíc měly být škálované
  - data pro neuronové sítě se nazývají “tenzory” (tensor)
- tenzor = zobecnění matice do libovolného počtu dimenzí:
  - 0. řád: skaláry (čísla)
  - 1. řád: vektory
  - 2. řád: matice
- příklady tvaru tenzorů podle vstupních dat:

vektorová data:	2. řád	(samples, features)
časové řady nebo sekvence:	3. řád	(samples, timesteps, features)
obrázky:	4. řád	(samples, height, width, channels)
video:	5. řád	(samples, frames, height, width, channels)

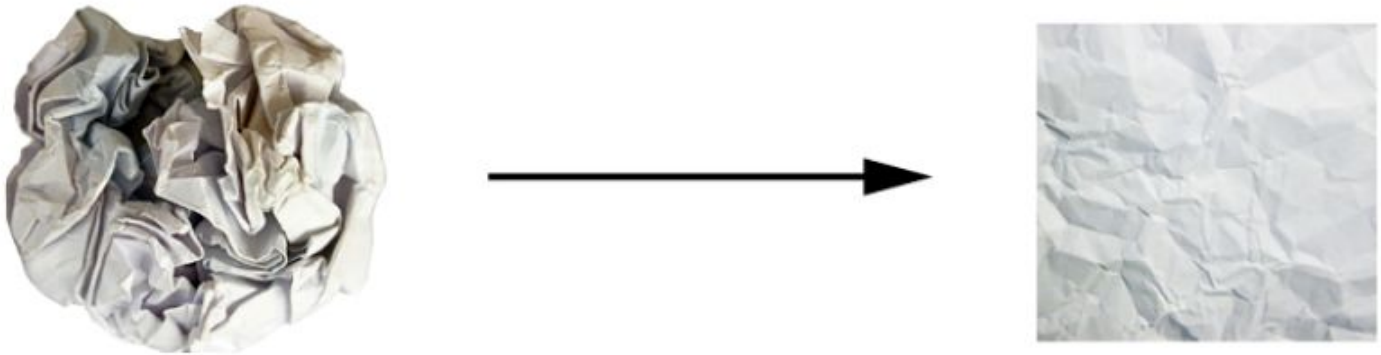
Geometrická interpretace tenzorových operací



Zdroj: (Chollet 2021)

Geometrická interpretace Deep Learningu

- neuronová síť představuje řadu tenzorových operací, přičemž všechny je možné chápat jako jednoduché geometrické transformace vstupních dat
- neuronovou síť je pak možné chápat jako velmi komplexní geometrickou transformaci v mnohodimenzionálním prostoru, která je implementovaná skrz dlouhou řadu jednoduchých kroků



Zdroj: (Chollet 2021)

Regresní úlohy

Regresní úlohy s neuronovými sítěmi

- chceme predikovat číselnou hodnotu (např. cena nemovitosti na základě mnoha příznaků)
  - použije se výstupní neuron bez aktivační funkce => hodnota pak není omezena
  - kolik potřebujeme výstupů, tolik použijeme výstupních neuronů
    - např. chceme zjistit, kde se objekt na obrazu nachází, tak potřebujeme dvě souřadnice a tím pádem dva výstupní neurony
      - pokud chceme přidat i ohraničující rámeček, tak potřebujeme další dva číselné údaje (výška, šířka) a tím pádem čtyři výstupní neurony
- pokud chceme výstupní hodnoty ohraničit minimem a maximem, můžeme použít aktivační funkci (logistická nebo tanh) a výstupy naškálovat
- ztrátová funkce: typicky MSE

Typická architektura regresní neuronové sítě

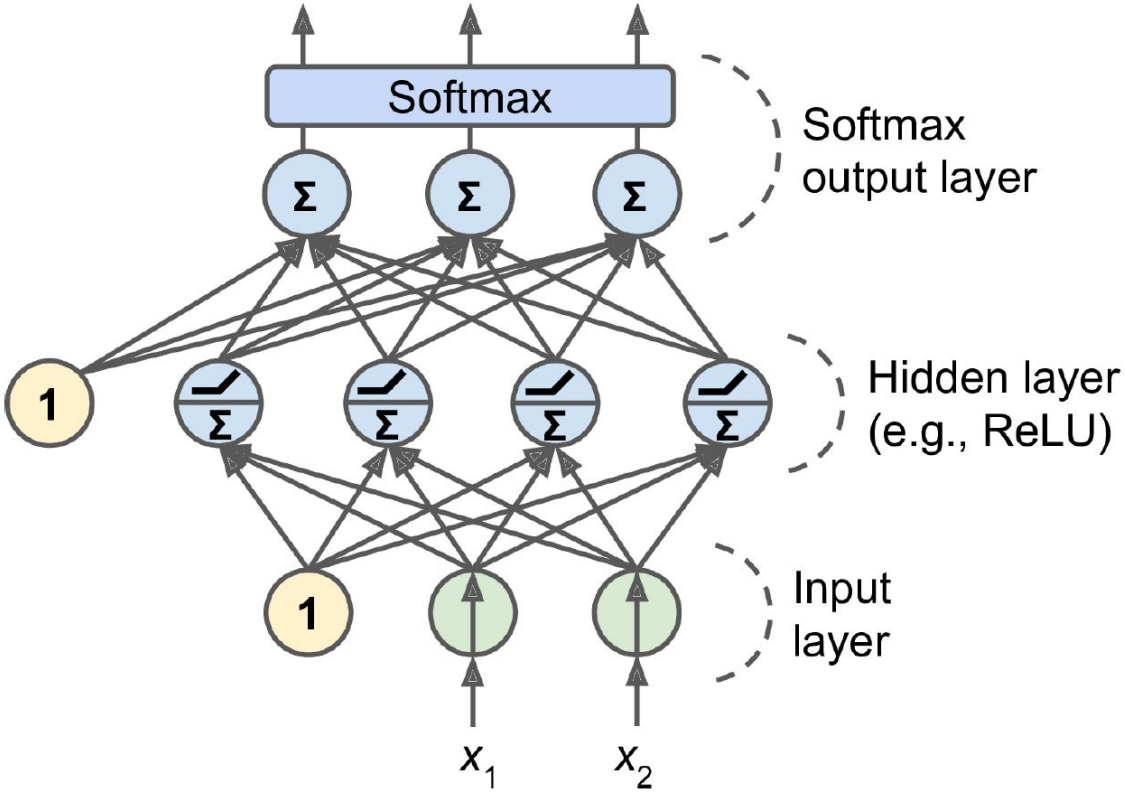
počet vstupních neuronů:	jeden na jeden příznak (např. 784 pro MNIST)
počet skrytých vrstev:	podle konkrétního problému, typicky 1 až 5
počet neuronů na skrytou vrstvu:	podle konkrétního problému, typicky 10 až 100
počet výstupních neuronů:	jeden na každou predikovanou hodnotu
aktivační funkce skrytých vrstev:	relu
aktivační funkce výstupní vrstvy:	žádná, případě logistická/tanh
ztrátová funkce:	MSE, případně MAE (outliers)

Klasifikační úlohy

Klasifikační úlohy s neuronovými sítěmi

- binární klasifikace
  - jeden výstupní neuron s logistickou aktivační funkcí
  - výstup pak bude 0 až 1, což lze interpretovat jako pravděpodobnost příslušnosti ke třídě
- klasifikace pomocí více labelů
  - např. zda e-mail je spam/ham a zda je či není urgentní
  - pro každý label jeden výstupní neuron s logistickou aktivační funkcí
- klasifikace do více (exkluzivních) tříd
  - např. MNIST klasifikace číslic do 10 tříd
  - jeden výstupní neuron pro každou třídu, navíc softmax aktivační funkce pro celou vrstvu
    - softmax (viz přednáška o logistické regresi) zajistí, že pravděpodobnosti budou 0 až 1 a že součet všech pravděpodobností bude roven 1
- ztrátová funkce: cross-entropy (také pod názvem log loss, viz přednáška o klasifikaci)

MLP pro klasifikační úlohu



Zdroj: (Géron 2019)

Typická architektura klasifikační neuronové sítě

	binární	multilabel	multiclass
vstupní a skryté vrstvy:	stejně jako pro regresní úlohy		
počet výstupních neuronů:	1	1 na label	1 na třídu
aktivační funkce výstupní vrstvy:	logistická	logistická	softmax
ztrátová funkce:	binary_crossentropy	crossentropy	crossentropy

Ukázka na TensorFlow Playground

https://playground.tensorflow.org/

Keras a TensorFlow

Knihovny pro práci s neuronovými sítěmi

- Keras
  - high level knihovna pro práci s neuronovými sítěmi
  - jednoduše použitelná
  - velmi flexibilní, lze s ní řešit jednoduché i velmi pokročilé úlohy
  - postavena nad low level výpočetním backendem pro práci s tenzory
    - TensorFlow, Microsoft CNTK, Theano
    - dnes je možné ji použít i v Javascriptu přímo v browseru nebo jako součást mobilních nebo embedded zařízení

- TensorFlow
  - low level knihovna pro práci s tenzory
  - obsahuje Keras již přímo jakou součást svého pythoního balíčku

Ukázka práce s knihovnou Keras - klasifikace

- dataset Fashion MNIST
  - přímá náhrada MNIST datasetu
    - 70 tisíc obrázků 28x28 pixelů
    - 10 tříd
  - obrázky ale reprezentují kusy oblečení a ne číslice
    - výrazně obtížnější na klasifikaci
  - lineární klasifikátor na číslicovém MNISTu dosahuje okolo 92% správnosti, zde jen 83% správnost
- viz Jupyter

Label	Description
0	T-shirt/top
1	Trouser
2	Pullover
3	Dress
4	Coat
5	Sandal
6	Shirt
7	Sneaker
8	Bag
9	Ankle boot

Zdroj: (Géron 2019)

Ukázka práce s knihovnou Keras - regrese

- dataset California housing
  - setkali jsme se s ním hned na začátku semestru (predikce cen nemovitostí)
- viz Jupyter

Early Stopping

- čím déle se neuronová síť trénuje, tím spíše se začne přeučovat
- z grafu vývoje ztrátové funkce na trénovacím a validačním setu lze vyčíst optimální počet epoch, aby se model nezačal přeučovat
- řešeno pomocí callbacků při trénování modelu
  - třída keras.callbacks.EarlyStopping
  - parametr patience
    - ukončí trénování, když nebude docházet ke zlepšení po X epoch
- ukázka viz Jupyter

Počet skrytých vrstev

- pro většinu problémů by teoreticky měla stačit jediná skrytá vrstva
  - pokud bude mít hodně neuronů, bude schopná řešit i velmi komplexní problémy
- hluboká síť má ale mnohem větší efektivitu z hlediska počtu parametrů
  - stačí jí exponenciálně méně neuronů než mělké síti, aby dokázala namodelovat komplexní problém
  - dokáže totiž využívat výstupů nižších vrstev, které obecně modelují struktury nižší úrovně
    - např. nižší vrstvy modelují hrany a linky, střední vrstvy tyto prvky skládají do obrazců (čtverce, kružnice, ...) a nejvyšší vrstvy spolu s výstupní vrstvou pak skládají tyto obrazce do objektů (např. obličej)
    - skutečná data jsou často strukturovaná takto v různých úrovních, proto hluboké neuronové sítě v praxi dobře fungují
  - transfer learning: natrénuje se síť a pak se pro nový problém použijí natrénované nižší vrstvy a jen se vymění “hlava” sítě, která se dotrénuje na specifických datech, pro které má fungovat
- jak zvolit počet vrstev:
  - začít s jednou či dvěma vrstvami
  - pokud se síť nepřeučuje, přidávat další vrstvy, dokud se nezačne přeučovat (potřebujeme přeučení dosáhnout, jinak nepoznáme, že už dosáhla svého maxima)
  - velmi komplexní úlohy (rozpoznávání obrazu či hlasu) potřebují i desítky skrytých vrstev, tím pádem ale potřebují také obrovské množství trénovacích dat

Počet neuronů na skrytou vrstvu

- ve vstupní a výstupní vrstvě je počet neuronů vcelku jasný
  - MNIST: 28x28 vstupních neuronů a 10 výstupních
- dříve se neuronové sítě strukturovaly typicky ve formě pyramidy
  - každá další vrstva měla méně neuronů, např. 300 - 200 - 100
  - odůvodněním byl předpoklad, že se mnoho různých vlastností nižších úrovní skládá do menšího počtu vlastností vyšších úrovní
  - v praxi se však ukázalo, že pokud se použije stejný počet neuronů ve všech vrstvách, dosahuje se minimálně stejných výsledků a není pak třeba ladit tolik hyperparametrů
- pokud se síť nepřeučuje, musíme zvýšit počet neuronů ve vrstvách
- obecně je efektivnější zvyšovat počet vrstev než počet neuronů ve vrstvě
- typický přístup je tedy ten, že se vezme model, který je větší, než potřebujeme a využijí se různé regularizační techniky, aby se nepřeučil (early stopping, dropout - viz později)

Zdroje

- Coelho, L. P.; Richert, W. (2013) Building machine learning systems with Python. Birmingham: Packt Publishing. ISBN 978-1-78216-140-0.

- Géron, A. (2019) Hands-On Machine Learning with Scikit-Learn, Keras, and TensorFlow, 2nd Edition. O'Reilly Media, Inc. ISBN 9781492032649.

- Chollet, F. (2021) Deep Learning with Python, Second Edition. Manning Publications. ISBN 9781617296864.

- Segaran, T. (2007) Programming collective intelligence: building smart web 2.0 applications. Beijing: O'Reilly Media. ISBN 0-596-52932-5.