Deep Learning pro texty a sekvenční data

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

- Práce s textem a Word Embeddings
- Rekurentní neuronové sítě
- 1D konvoluční sítě
- LSTM a GRU vrstvy
- Generování textu

Deep Learning pro sekvenční data

- co budeme zpracovávat:
  - text
    - tj. sekvence slov nebo sekvence znaků
  - časové řady
  - sekvenční data obecně

- použitelné algoritmy:
  - rekurentní neuronové sítě
  - 1D konvoluční neuronové sítě

- typické úlohy:
  - klasifikace dokumentů a časových řad
    - např. analýza sentimentu, identifikace tématu článku, identifikace autora textu apod.
  - porovnání sekvenčních dat
    - např. jak moc si jsou podobné dokumenty nebo jak moc si je podobný vývoj ceny akcií
  - překlad sekvence na sekvenci
    - např. strojový překlad z angličtiny do češtiny
  - predikce časových řad
    - např. predikce počasí

- pozn.: žádný ze současných přístupů (ani GPT-3) nechápe text v lidském slova smyslu

Word Embeddings

Práce s textem a Word Embeddings

- zpracování přirozeného jazyka (NLP - natural language processing)
  - rozpoznávání vzorů aplikované na slova, věty a odstavce; obdobně jako počítačové vidění využívá rozpoznávání vzorů při práci s obrazem
- text lze chápat jako sekvence znaků nebo jako sekvence slov
  - většinou ale jako sekvence slov
- modely hlubokého učení nemohou pracovat s textem v surové podobě
  - nutno převést text na číselné tenzory = vektorizace
    - jednotky, na které se text rozpadá (slova, znaky, n-gramy) = tokeny;
    - převod textu na tokeny = tokenizace
  - jednotlivé vektory (představující slova, znaky nebo n-gramy) se následně zabalí do sekvenčních tenzorů

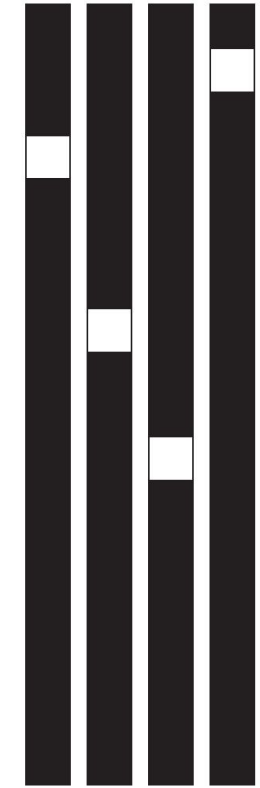
One-hot Encoding a Word Embeddings

- jak převést token na vektor:
  - one-hot encoding
  - word embeddings (“vnoření slov”)

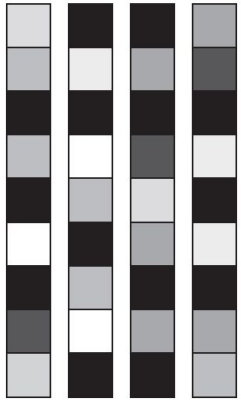
- one-hot encoding
  - každému slovu se přiřadí celočíselný index
  - kolik je různých slov, tolik je různých indexů a tím pádem také dimenzí vektoru
  - vektor pro dané slovo obsahuje samé nuly a jednu jedničku v příslušném indexu
    - podobné jako bag-of-words, ale bag-of-words reprezentuje celý dokument, zatímco one-hot encoding reprezentuje jedno slovo
      - celý dokument je pak zakódován jako sekvence těchto slovních vektorů a nikoliv jako jeden vektor

Word Embeddings

- velmi populární a výkonný způsob, jak převést slovo na vektor
- slovo se zakóduje do tzv. hustého slovního vektoru (dense word vector)
  - one-hot vektory jsou binární a vysoce dimenzionální (sparse vector)
  - word embeddings jsou nízkodimenzionální vektory reálných čísel (dense vector)
- word embeddings jsou natrénované z dat, podobně jako jsou třeba natrénované filtry v konvolučních vrstvách
  - oproti tomu one-hot encoding je dopředu a napevno



One-hot word vectors:
- Sparse
- High-dimensional
- Hardcoded



Word embeddings:
- Dense
- Lower-dimensional
- Learned from data

Zdroj: (Chollet 2019)

Word Embeddings

- počet dimenzí ve word embeddings je na našem uvážení (hyperparametr)
  - obvykle se volí hodnoty jako 128, 256, 512 nebo případně 1024 pro velmi velké slovníky
  - oproti tomu one-hot encoding může být 10 tisíc nebo také 100 tisíc dimenzí

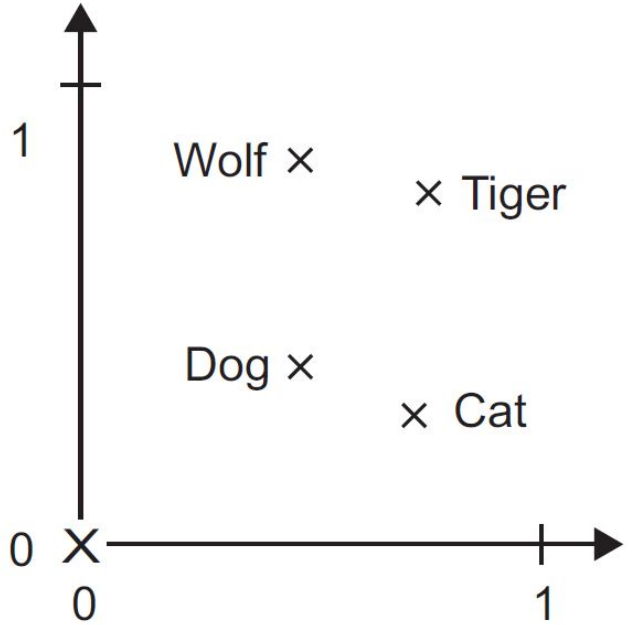
- jak získat word embeddings:
  - natrénovat společně s hlavní úlohou, kterou řešíme
    - začne se s náhodnými slovními vektory a ty se postupně učí (upravují) podobně jako se učí jiné parametry v neuronové síti
  - načíst předtrénované word embeddings z jiné úlohy (pretrained word embeddings)

Word Embeddings

- pokud bychom zvolili vektory náhodně, nebude mít výsledný prostor vnoření žádnou strukturu
  - významově podobná slova by např. byla prostorově daleko od sebe, ačkoliv ve větě by třeba byla zaměnitelná
- proto se word embeddings trénuje tak, aby podobná slova byla blízko u sebe a aby geometrické vztahy mezi slovními vektory reflektovaly významové vztahy mezi slovy

Word Embeddings

- významové vztahy mezi slovy mohou být zakódované jako geometrické transformace
- např. abychom se z kočky dostali k tygrovi by měl být stejný vektor transformace jako od psa k vlkovi
  - vektor “od domácího zvířete k divokému”
  - vektor “od psovité šelmy ke kočkovité”
- příklady geometrických transformací v reálných word embeddings:
  - vektor pohlaví (muž - žena)
  - vektor množného čísla
  - příklady:
    - král + vektor pohlaví + vektor množného čísla = královny
    - Bill Clinton - USA + ČR = Václav Havel



Zdroj: (Chollet 2019)

Word Embeddings

- zatím ale neexistuje nic jako “perfektní” prostor pro word embeddings
  - pro každou úlohu se totiž hodí jiný prostor
    - např. ideální word embeddings pro model analýzy sentimentu anglických hodnocení filmů bude patrně vypadat jinak než pro úlohu klasifikace právních dokumentů, protože důležitost různých sémantických vztahů bude v obou úlohách jiná
- word embeddings se tedy typicky trénují pro každou úlohu od začátku
  - `keras.layers.Embedding(1000, 64)`
    - maximálně 1000 různých tokenů
    - 64 dimenzí
- jak chápat vrstvu `Embedding` v keras modelech:
  - slovník, který mapuje celočíselný index (tedy jedno konkrétní slovo) na hustý vektor
  - vstup: 2D tenzor čísel (samples, sequence_length), každý jeden příklad tedy sekvence čísel
    - všechny sekvence v jedné dávce musejí mít stejnou délku (buď doplnění nulami nebo oříznutí)
  - výstup: 3D tenzor tvaru (samples, sequence_length, embedding_dimensionality)
    - toto už lze použít jako vstup do rekurentní nebo 1D konvoluční neuronové sítě

Word Embeddings

- ukázka viz jupyter imdb_embedding

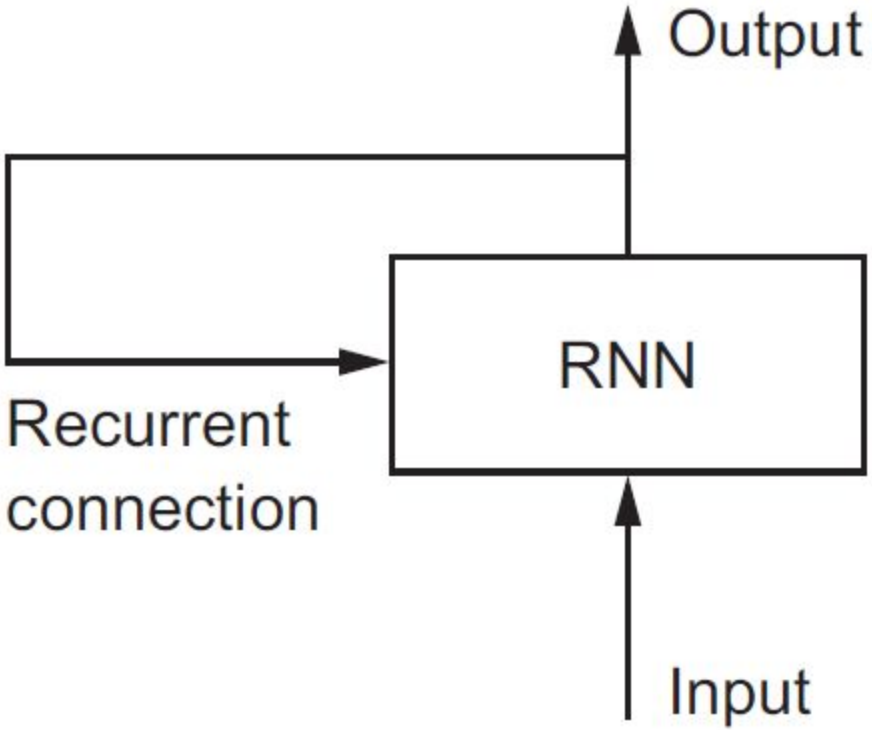
Předtrénovaná word embeddings

- pokud máme příliš málo dat na to, abychom si natrénovali word embeddings pro naši úlohu, můžeme využít předtrénovaného modelu
  - předpokladem je, že předtrénovaný model reprezentuje obecné charakteristiky v textu
  - přelomový model Tomáše Mikolova word2vec z roku 2013
  - od té doby celá řada dostupných předtrénovaných modelů
    - Glove https://nlp.stanford.edu/projects/glove/ (korpus z Wikipedie)
    - NNLM https://tfhub.dev/google/tf2-preview/nnlm-en-dim50 (korpus z Google News)

Rekurentní neuronové sítě

Úvod do rekurentních neuronových sítí

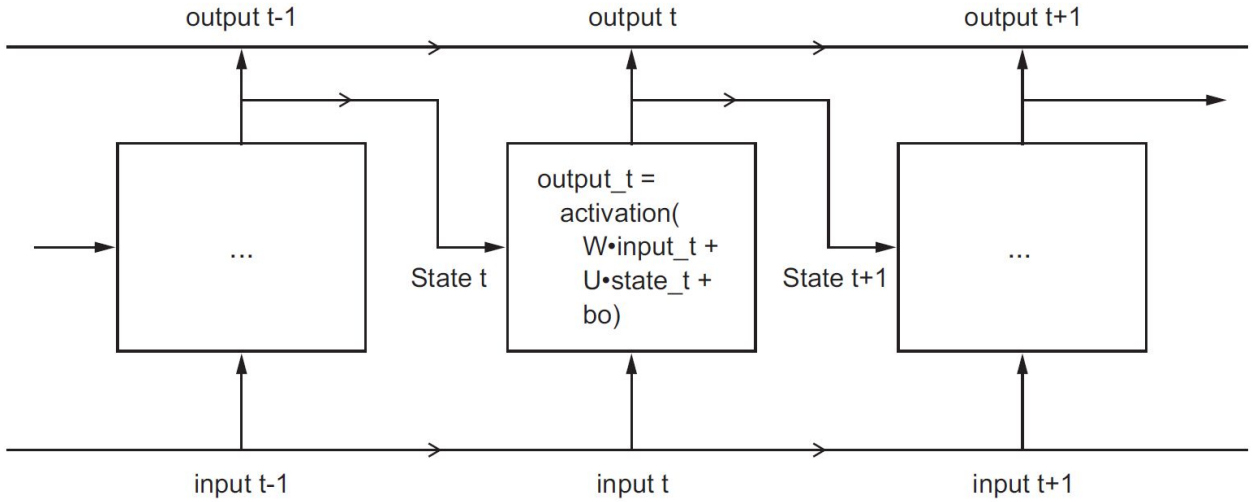
- sítě, které jsme viděli doteď, neměly žádnou formu paměti
  - sekvence se takové síti musí předat jako jeden datový bod
  - “sítě s dopředným chodem” (feedforward networks)
- naproti tomu biologická inteligence zpracovává informace inkrementálně a udržuje si interní model, který je vytvořen z minulých informací a průběžně aktualizován
- rekurentní neuronové sítě (RNN - recurrent neural network) používají podobný princip, nicméně velmi zjednodušeně:
  - zpracovávají sekvence postupně po prvcích a udržují stav o tom, co viděly doposud
    - RNN tedy mají vnitřní cyklus
  - stav se udržuje pouze pro konkrétní instanci; jakmile je instance zpracovaná, stav se resetuje



Zdroj: (Chollet 2019)

Rekurentní neuronové sítě

- RNN tedy využívá výstupy z předchozí iterace ve svém vnitřním cyklu
- jednoduchá RNN rozbalená v čase:
  - finální výstup této sítě: (timesteps, output_features), tedy pro každý timestep máme výstup (většinou nám ale stačí až poslední výstup, tedy (output_features))



Zdroj: (Chollet 2019)

Rekurentní vrstvy v Kerasu

- nejjednodušší vrstva SimpleRNN, která odpovídá uvedenému popisu
  - vstupy: (batch_size, timesteps, input_features)
  - výstup: stejně jako jiné RNN vrstvy, může vracet buď celou sekvenci (batch_size, timesteps, output_features) nebo jen poslední výstup pro každou sekvenci (batch_size, output_features)
    - parametr return_sequences v konstruktoru
- vrstvy lze stohovat stejně jako v případě Dense nebo Conv vrstev
  - může se tím zvýšit reprezentační síla sítě
  - všechny mezivrstvy pak musejí vracet plné sekvence, aby další vrstva mohla pracovat s jednotlivými timestepy
  - nejdřív se zpracuje celá nižší vrstva a ta pak předá všechny výstupy vyšší vrstvě
  - ```
    model = Sequential([
        Embedding(10000, 32),
        SimpleRNN(32, return_sequences=True),
        SimpleRNN(32, return_sequences=True),
        SimpleRNN(32, return_sequences=True),
        SimpleRNN(32),
        Dense(1, activation='sigmoid'),
    ])
    ```
    ```

Rekurentní vrstvy v Kerasu

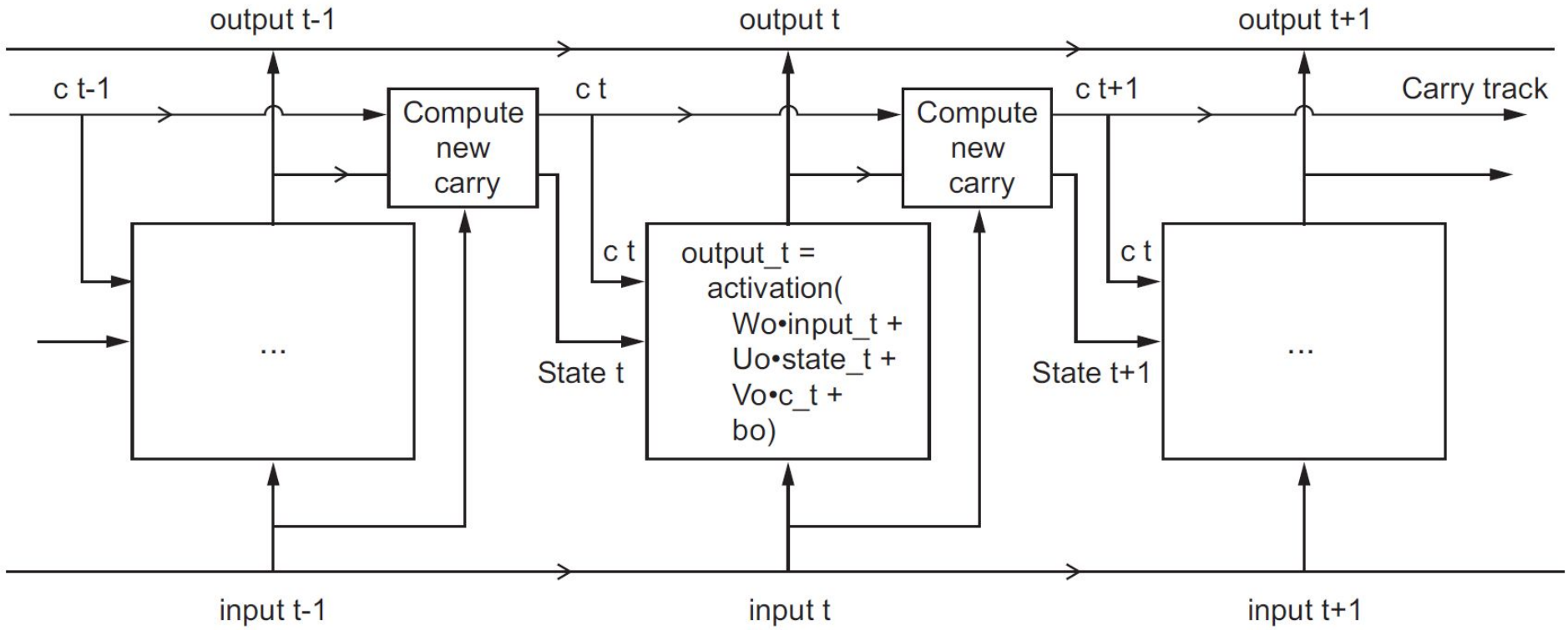
```
>>> model.summary()
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
embedding_24 (Embedding)     (None, None, 32)          320000
_________________________________________________________________
simplernn_12 (SimpleRNN)     (None, None, 32)          2080
_________________________________________________________________
simplernn_13 (SimpleRNN)     (None, None, 32)          2080
_________________________________________________________________
simplernn_14 (SimpleRNN)     (None, None, 32)          2080
_________________________________________________________________
simplernn_15 (SimpleRNN)     (None, 32)                2080
=================================================================
Total params: 328,320
Trainable params: 328,320
Non-trainable params: 0
```

Ukázka rekurentní sítě

viz jupyter imdb_rnn

Lepší typy rekurentních vrstev v Kerasu

- SimpleRNN bývá příliš jednoduchá pro reálné použití, proto se výhradně používají některé z pokročilejších vrstev: LSTM a GRU
  - SimpleRNN mají problém s mizejícím gradientem, díky čemu se paměť nedaří zachovat na dostatečně dlouhé sekvence
- LSTM
  - Long Short-Term Memory
  - rozšiřuje architekturu SimpleRNN o způsob, jak nést informaci napříč mnoha časovými kroky
    - “dopravní pás”, který běží paralelně se sekvencí; informace na něj mohou v libovolném místě “naskočit” a v libovolném místě být zase neporušeně “vyloženy”
    - LSTM tedy umožňuje uložit libovolné informace na pozdější použití


output t-1
output t
output t+1
c t-1
Compute new carry
c t
Compute new carry
c t+1
Carry track
...
c t
output_t = activation(Wo•input_t + Uo•state_t + Vo•c_t + bo)
State t
c t
...
State t+1
input t-1
input t
input t+1

GRU - Gated Recurrent Unit

- zjednodušená a rychlejší varianta LSTM
- ukázka viz jupyter

Obousměrné RNN

- RNN jsou obecně zcela závislé na pořadí, v jakém danou sekvenci zpracovávají
  - podle toho se také učí veškeré reprezentace
- pokud pro danou úlohu nezáleží na tom, zda je sekvence zpracovávaná zleva nebo zprava, můžeme zkusit použít obousměrnou vrstvu
  - např. v oblasti zpracování přirozeného jazyka je důležité, že jsou slova u sebe, ale moc nezáleží na tom, z jaké strany
  - díky tomu se při průchodu zleva i zprava může síť naučit odlišné reprezentace, které dohromady mohou být užitečnější
    - analogie ensemble learningu

Rekurentní neuronové sítě a GPU

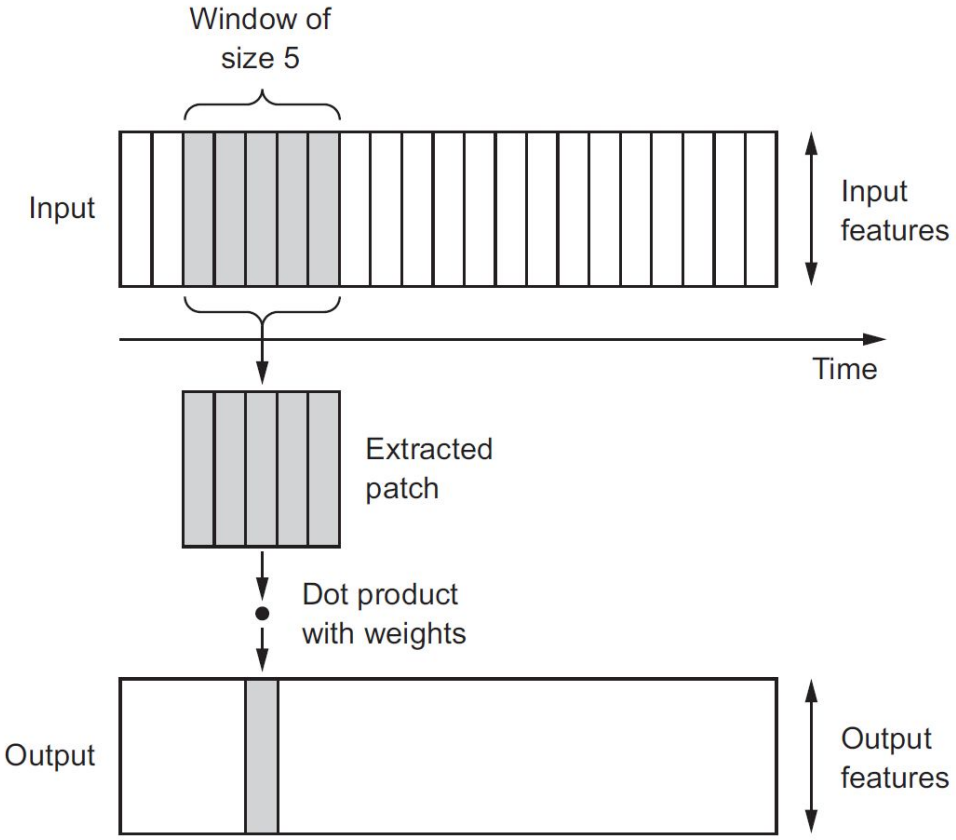
- rychlé zpracování na GPU lze využít pouze pro určité kombinace hyperparametrů rekurentních vrstev
  - typicky výchozí hodnoty fungují, viz dokumentace

1D konvoluční neuronové sítě

Zpracování sekvencí pomocí konvolučních sítí

- princip konvolučních sítí lze aplikovat i na sekvenční data
  - na čas lze nahlížet jako na prostorovou dimenzi
- 1D konvoluční sítě mohou soupeřit s RNN při řešení určitých úloh a to za využití menšího výpočetního výkonu
- konvoluční vrstvy v případě zpracování obrazu extrahovaly 2D oblasti z obrazu
  - 1D konvoluční vrstvy dělají to samé - extrahují 1D oblasti vstupních dat
- 1D konvoluční vrstva dokáže rozpoznat lokální vzor v sekvenci
  - stejná transformace je použita pro každou oblast, proto vzor, který se síť naučí na určitém místě, lze aplikovat i kdekoliv jinde

Conv1D



Zdroj: (Chollet 2019)

Conv1D a MaxPooling1D

- Conv1D
  - 1D konvoluční sítě umožňují použít větší “okna” než v případě 2D
    - 3x3 ve 2D = 9 feature vektorů
    - v 1D si tedy můžeme dovolit okna velikosti 7 nebo 9 bez větších problémů

- MaxPooling1D
  - stejně jako se maxpooling používá při zpracování obrazu, použije se i u sekvenčních dat, aby se zmenšila velikost mapy příznaků a aby vyšší vrstvy viděly větší část vstupních dat

- Con1D lze použít zejména tam, kde sice záleží na lokálním pořadí, ale globální pořadí nehraje takovou roli
  - analýza textu, kde řešíme např. sousedící slova, ale nezáleží na celkovém pořadí ve větě, je vhodná úloha
  - naopak časové řady je lepší řešit pomocí RNN

Ukázka Conv1D

- viz jupyter

Časové řady

Časové řady

- úloha predikce časové řady: předpověď teploty
  - máme k dispozici údaje z meteostanice o teplotě, tlaku, vlhkosti, … a budeme predikovat, jaká bude teplota za 24 hodin
    - úlohu lze ale modifikovat a místo jedné hodnoty (teplota za 24 hod), můžeme predikovat třeba teploty po hodinách na příštích 24 hodin nebo cokoliv jiného
      - není tedy nutné predikovat jen jednu hodnotu

- v rámci této úlohy si ukážeme některé pokročilejší koncepty RNN:
  - rekurentní výpadek (recurrent dropout)
  - stohování rekurentních vrstev

- viz jupyter

Generování textu

Generování textu

- použijeme předtrénovaný model a necháme ho generovat další text
  - alternativně bychom si mohli na textovém korpusu natrénovat vlastní model, nicméně je to výpočetně hodně náročné (řádově hodiny na jednu epochu)
  - trénování probíhá tak, že jedna instance je sada několika po sobě jdoucích slov a cílová hodnota je další slovo v textu
    - takto se model natrénuje, jaká slova predikovat po jiných slovech

- vyzkoušíme si OpenAI GPT-2 model
  - využívá architekturu nazvanou Transformer
    - moderní architektura, která je založená na principu encoder-decoder, přičemž nevyužívá rekurentní vrstvy
    - podařilo se díky ní zlepšit kvalitu strojových překladů a úloh generování textu
    - více případně formou samostudia

Strojový překlad

Strojový překlad

- zatím jsme sekvence převáděli na vektor
  - IMDB: text na třídu
  - Jena: časová řada na číslo / vektor

- sekvence jdou ale převádět opět na sekvence
  - lze využít jak rekurentní neuronové sítě (zejména LSTM a GRU vrstvy), tak velmi moderní architektury založené na Transformer architektuře
  - viz např. DeepL https://www.deepl.com/translator (dle ohlasů citelně lepší než Google Translate)
  - viz https://www.tensorflow.org/tutorials/text/nmt_with_attention
  - viz https://www.tensorflow.org/tutorials/text/transformer

Úkoly

- zkuste natrénovat model, který bude predikovat teploty pro příštích 24 hodin (tedy 24 údajů)
  - jak na to:
    - jako cílové hodnoty předejte modelu ne jednu teplotu, ale všech 24
    - upravte strukturu sítě, aby na výstupu měla 24 hodnot

- zkuste porovnat různé přístupy k analýze sentimentu textu podle počtu trénovacích instancí
  - porovnejte čisté word embedding, předtrénované word embedding, původní bag-of-words
    - pokud to výpočetní výkon dovolí, zkuste porovnat modely GRU, Conv1D a Dense
  - zkuste např. 100, 200, 500, 1000, 2000, 5000, 10000, 20000 trénovacích instancí
  - jak se vyvíjí správnost jednotlivých modelů podle počtu instancí?

- projděte si tutoriály Attention a Transformer na TensorFlow (viz předchozí slide)

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. (2019) Deep Learning v jazyku Python. Knihovny Keras, TensorFlow. Grada Publishing, a.s. ISBN 978-80-247-3100-1.

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