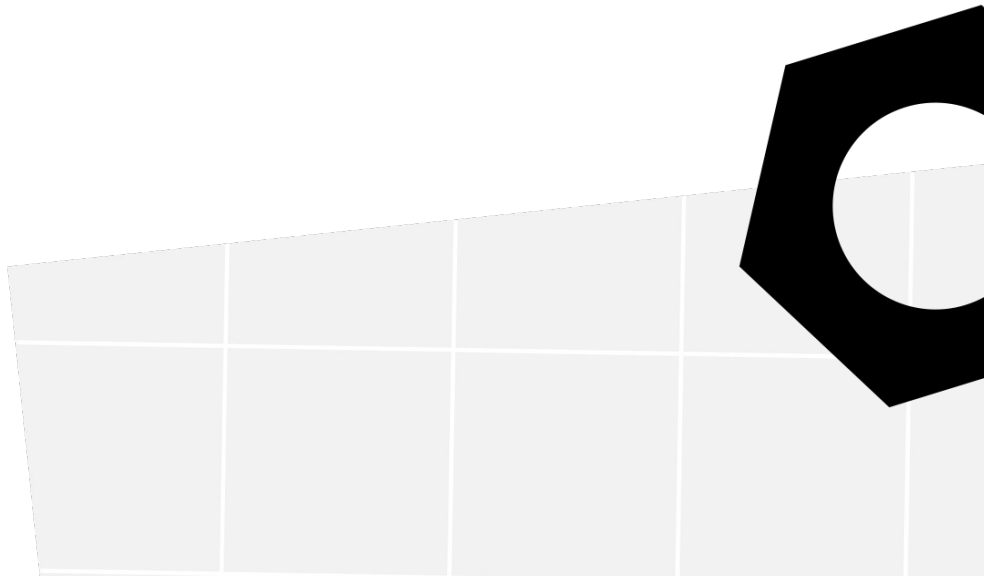
Praca z sekwencjami – rekurencyjne sieci neuronowe

Kurs Data Science



software
development
academy

Rekurencja – czyli co ma wspólnego matrioszka z algorytmami?

- Tak jak rosyjska lalka ma w sobie mniejszą lalkę, a ta posiada jeszcze mniejszą i tak dalej aż trafimy na laleczkę, która jest zbyt mała, aby pomieścić następną, tak my zobaczymy jak zaprojektować algorytm, który rozwiąże problem, rozwiązując najpierw mniejszy przypadek tego samego problemu, dopóki ten problem nie jest tak mały, aby rozwiązać go bezpośrednio.
- Taką technikę nazywamy rekurencją.

Przykłady użycia rekurencji

- Napiszmy program rekurencyjny wykonujący bardzo popularne obliczenia matematyczne. Weźmy silnię liczby całkowitej n zapisywaną jako n! (wymawiamy „n silnia”), która jest wynikiem iloczynu:

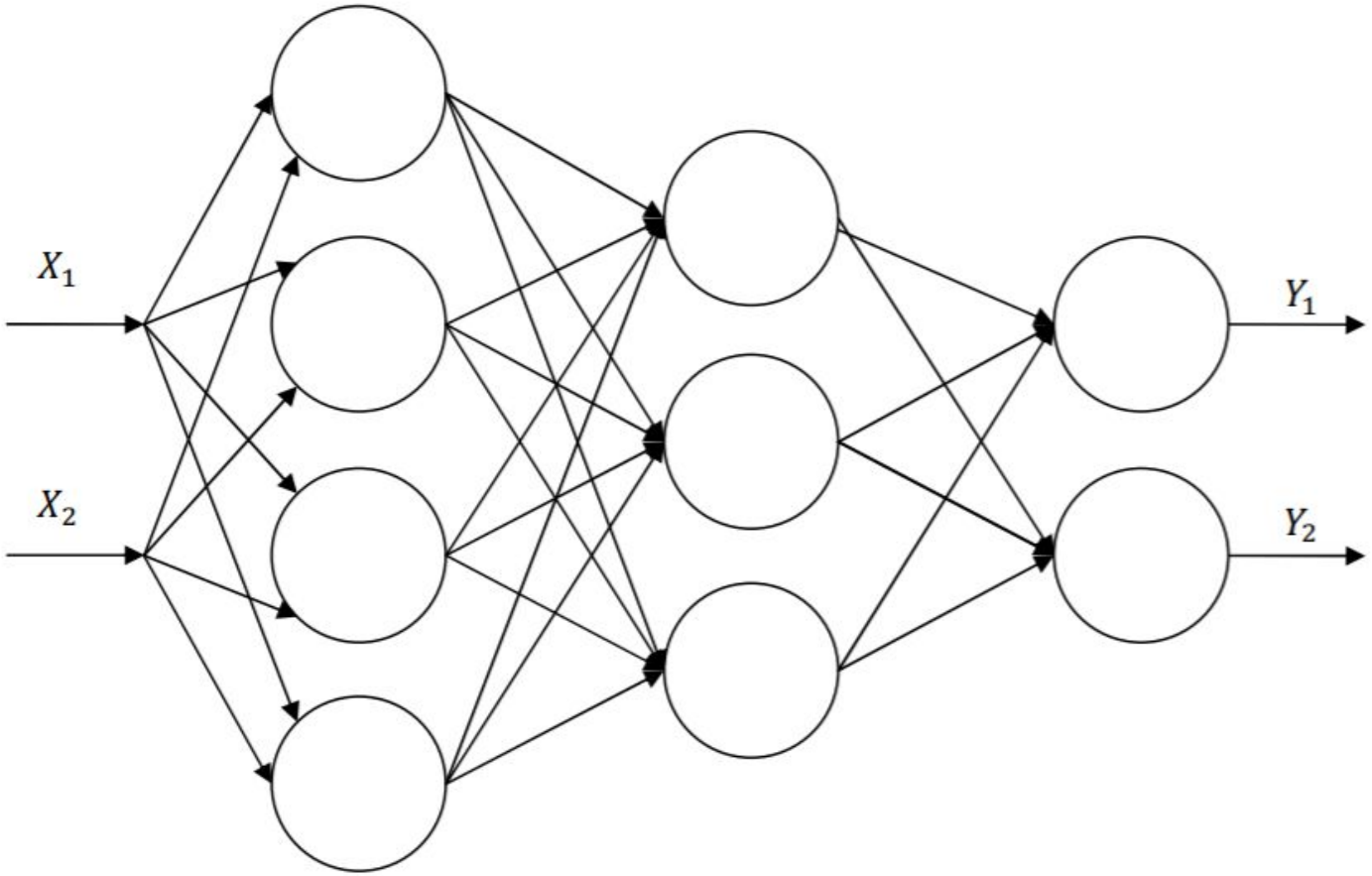
$$n \cdot (n - 1) \cdot (n - 2) \cdot \ldots \cdot 1$$

gdzie 1! wynosi 1, a 0! jest zdefiniowane jako 1. Oznacza to, że 5! to wynik iloczynu $5 \cdot 4 \cdot 3 \cdot 2 \cdot 1$, czyli 120.

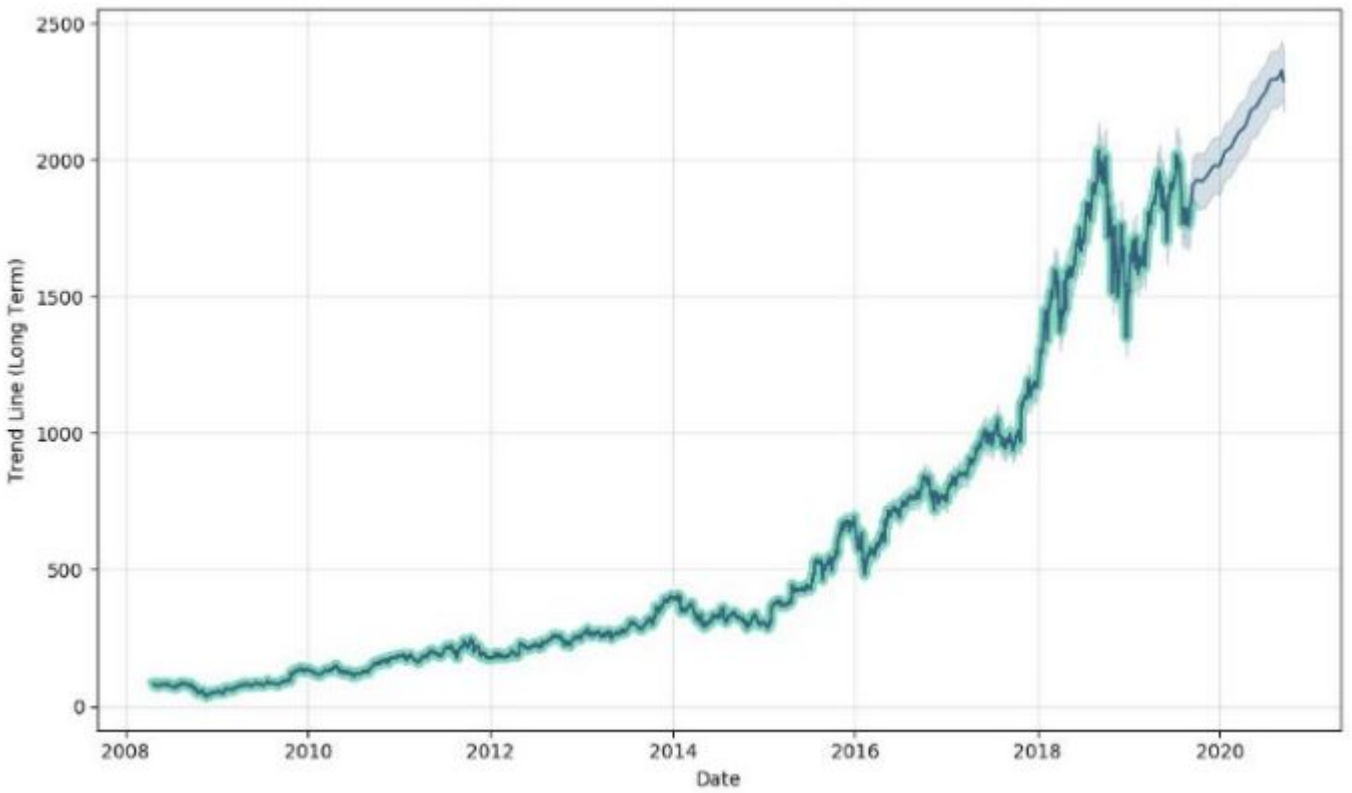
Silnię liczby całkowitej number (gdzie number 0) jesteśmy w stanie wyliczyć iteracyjnie (bez rekurencji) za pomocą instrukcji for w sposób następujący:

```
factorial=1;
for (int counter = number; counter >= 1; counter--) {
    factorial *= counter;
}
```

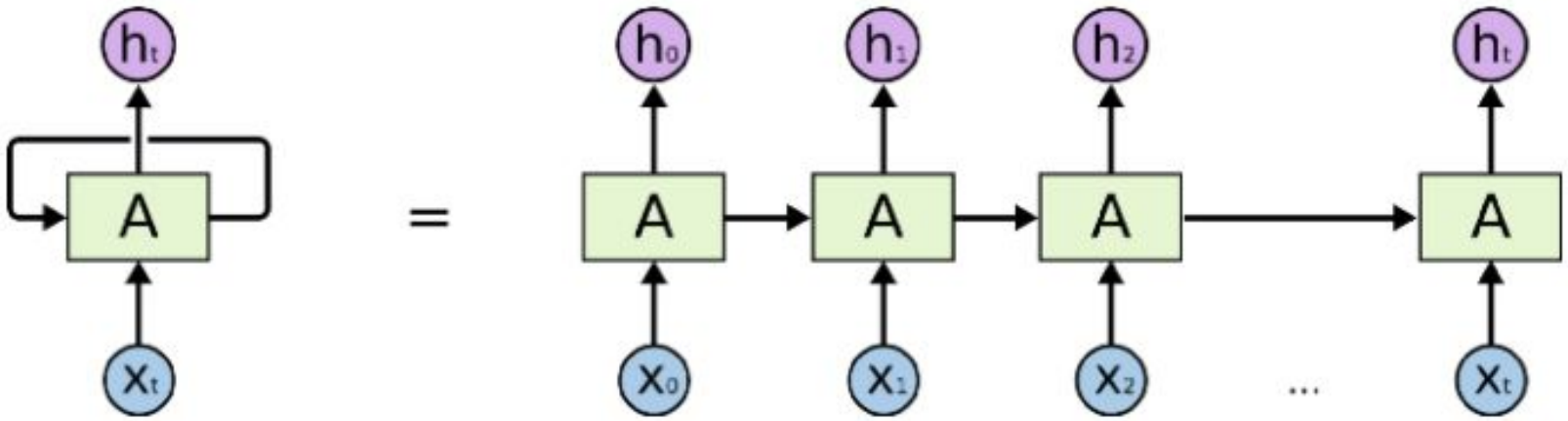
Przypomnienie – struktura perceptronu wielowarstwowego



Sieci rekurencyjne - wprowadzenie



Sieci rekurencyjne - wprowadzenie



A Recurrent Neural Network

Sieci rekurencyjne - wprowadzenie

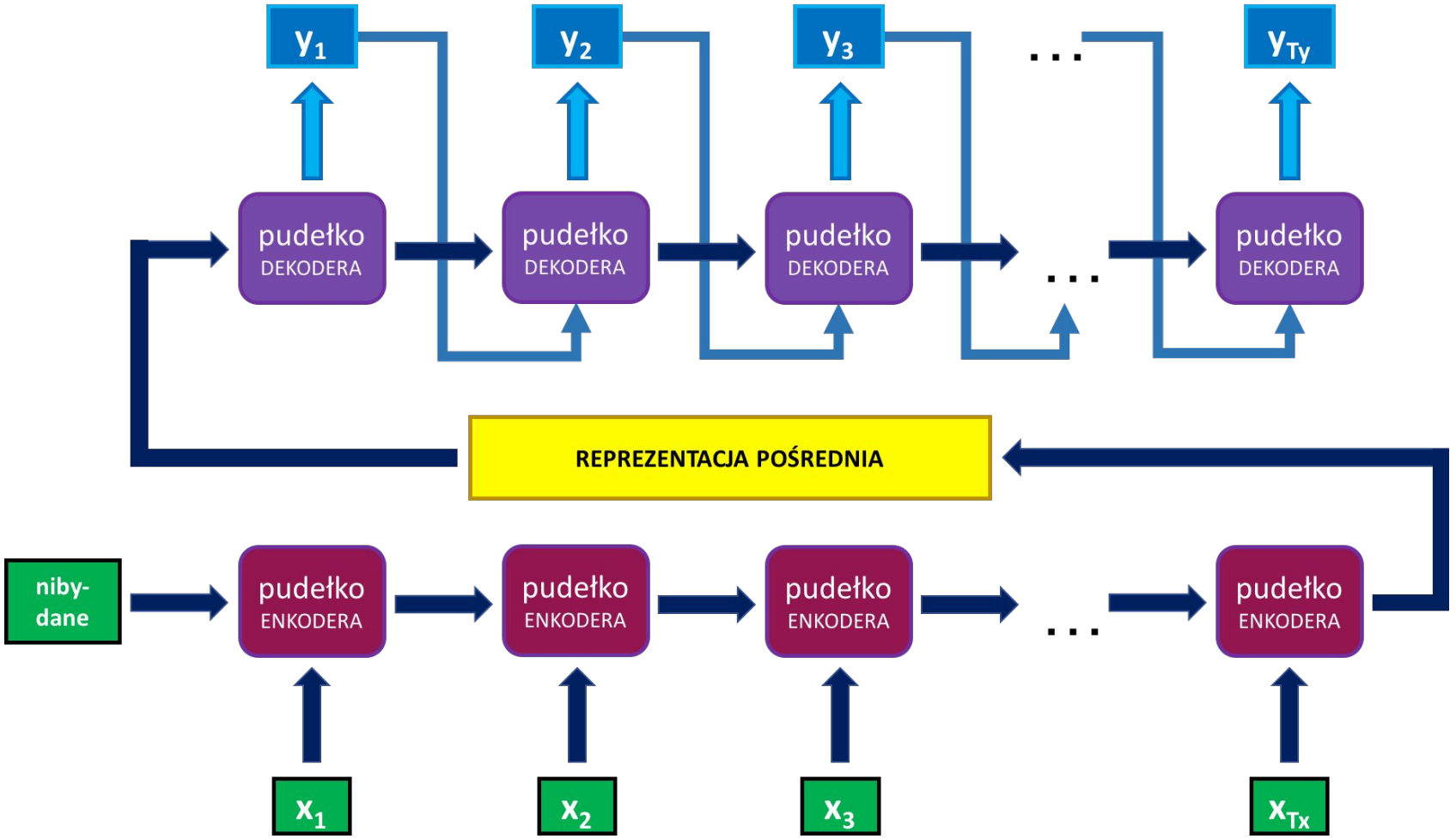
Charakterystyka:

- Istnienie sprzężeń zwrotnych między wejściem a wyjściem
- Dwukierunkowy przepływ informacji

Do najczęściej spotykanych sieci rekurencyjnych należą:

- Sieć Hopfielda
- Maszyna Boltzmana

Przykładowa architektura sieci rekurencyjnej



„Pseudokod” sieci rekurencyjnej

```
state_t = 0
for input_t in input_sequence:
    output_t = f(input_t, state_t)
    state_t = output_t
```

Implementacja prostej rekurencyjnej sieci neuronowej w pakiecie numpy

```
import numpy as np

timesteps = 100
input_features = 32
output_features = 64

inputs = np.random.random((timesteps, input_features))

state_t = np.zeros((output_features,))

W = np.random.random((output_features, input_features))
U = np.random.random((output_features, output_features))
b = np.random.random((output_features,))

successive_outputs = []
for input_t in inputs:
    output_t = np.tanh(np.dot(W, input_t) + np.dot(U, state_t) + b)

    successive_outputs.append(output_t)

    state_t = output_t

final_output_sequence = np.concatenate(successive_outputs, axis=0)
```

Liczba kroków czasu sekwencji wejściowej.

Liczba wymiarów przestrzeni cech wejściowych.

Liczba wymiarów przestrzeni cech wyjściowych.

Dane wejściowe: w tym przypadku jest to tylko losowy szum.

Stan początkowy: wektor składający się z samych

Tworzenie losowej macierzy wag.

Obiekt input_t jest wektorem o kształcie (input_features,).

Łączenie danych wejściowych z obecnym stanem (poprzednimi danymi wyjściowymi) w celu uzyskania bieżącej wartości wyjściowej.

Zapisywanie wartości wyjściowej w formie listy.

Aktualizacja stanu sieci w celu przygotowania jej do przetworzenia kolejnego kroku czasu.

Ostateczny obiekt wyjściowy jest dwuwymiarowym tensorem o kształcie (timesteps, output_features).

Źródło: Francois Chollet, Deep Learning. Praca z językiem Python i biblioteką Keras

Sekwencja

- Sekwencja to uporządkowana lista zbiorów zdarzeń, oznaczana jako:

$$S = (i_1, \ldots, i_{N_1})(i_1, \ldots, i_{N_2}) \ldots (i_1, \ldots, i_{N_T}).$$

- Przykładem takiej sekwencji są zakupy dokonywane przez jednego klienta, w pewnym sklepie komputerowym:

$$S = (Komputer, Monitor)(Mysz, Klawiatura)(Papier, Drukarka)$$

Zastosowania danych sekwencyjnych

- Dane biologiczne (wskazanie czy w danym fragmencie DNA jest zakodowana informacja genetyczna)
- Klasyfikacja dokumentów:

✔ Klasyfikacja pod względem treści

✔ Klasyfikacja dokumentów XML pod względem struktury

✔ Analiza ruchu sieciowego

✔ Klasyfikacja użytkowników

Metody klasyfikacji danych sekwencyjnych

- Klasyfikacja za pomocą drzew przyrostowych
- Klasyfikacja w oparciu o ukryty model Markova
- Metoda najbliższych sąsiadów do klasyfikacji:

☐ Tekstów

☐ Protein

☐ Łancuchów DNA

Założenia

- Sieć neuronowa przyjmuje na wejściu odpowiednio zakodowaną sekwencję, a na wyjściu zwraca zakodowany identyfikator klasy.
- W procesie projektowania sieci rekurencyjnej najważniejszy jest:

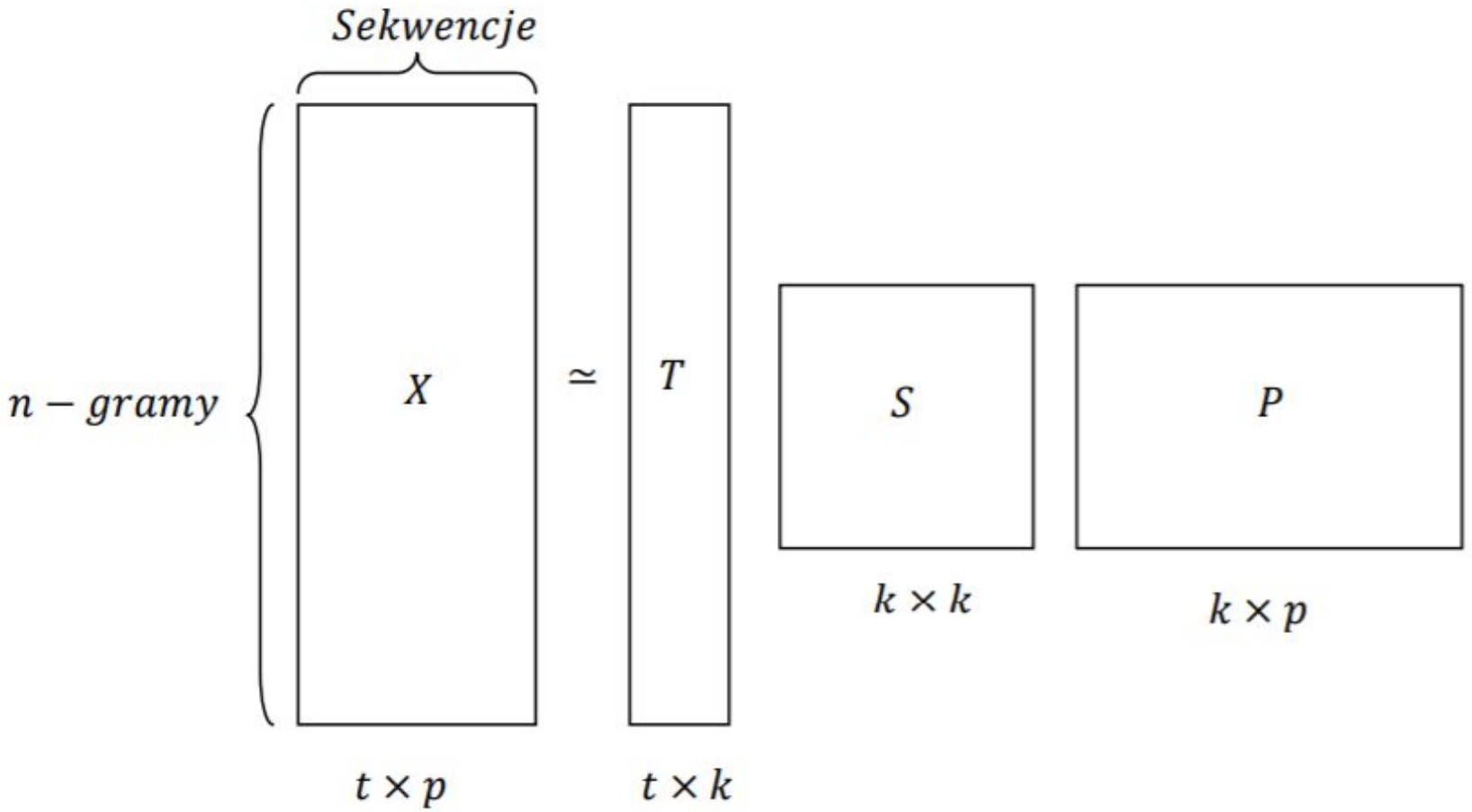
Wybór architektury oraz sposób kodowania sekwencji.

Kodowanie sekwencji

- Przetworzenie sekwencji wejściowych na wektor liczb, który traktujemy jako sygnał wejściowy sztucznej sieci neuronowej.
- Każda sekwencja musi być zakodowana tą samą ilością symboli.
- Są 2 metody kodowania:

- Metoda n-gramowa
- Metoda rozkładu wartości osobliwych (ang.SVD)

Rozkład wartości osobliwych



Architektura sieci

- Ilość wejść jest wyznaczona przez rozmiar wektora (zależna od sposobu kodowania sekwencji)
- Ilość wyjść jest równa liczbie możliwych klas
- Ilość neuronów dobierana jest heurystycznie
- Najczęściej dobierana wartość pomiędzy ilością wejść a wyjść
- Wytrenowanie kilku sieci neuronowych

Budowa sieci rekurencyjnej w pakiecie Keras

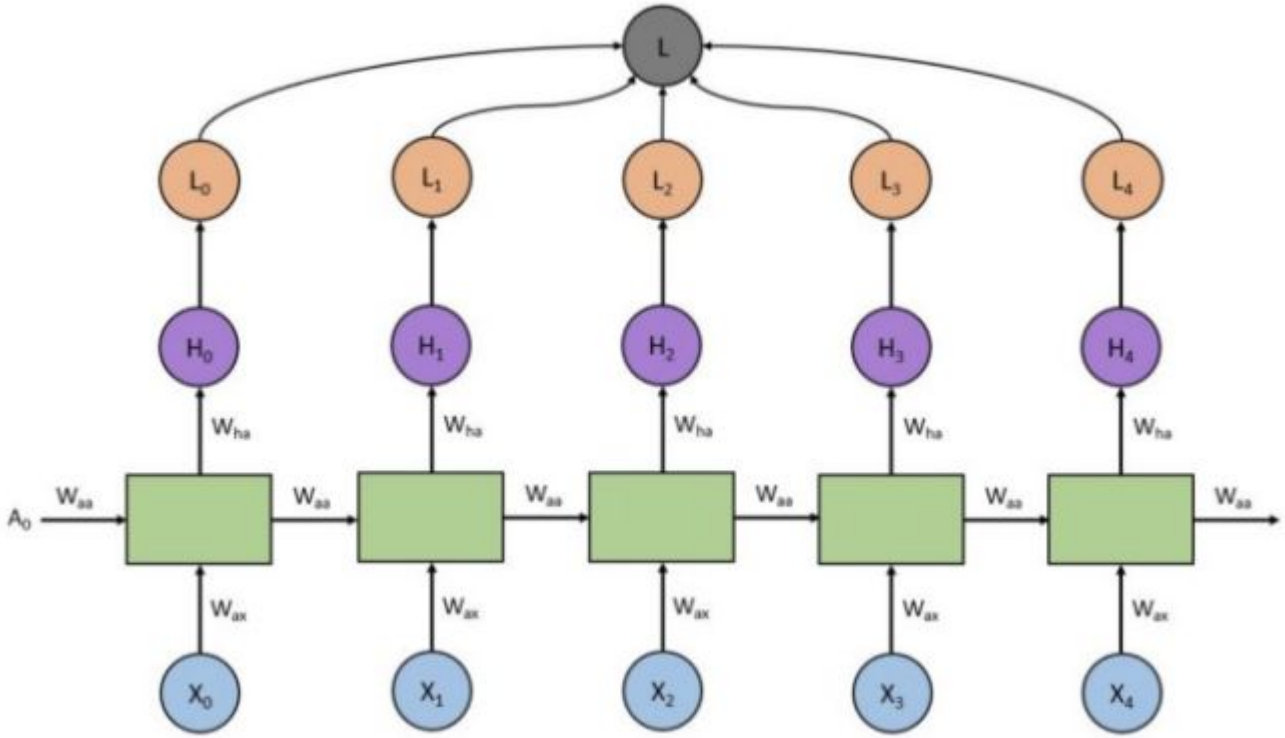
Będziemy korzystać z danych, które znajdziesz w zeszycie z sieci rekurencyjnych.

Możesz go pobrać z sekcji
“Dodatkowe materiały do bloku” tutaj.

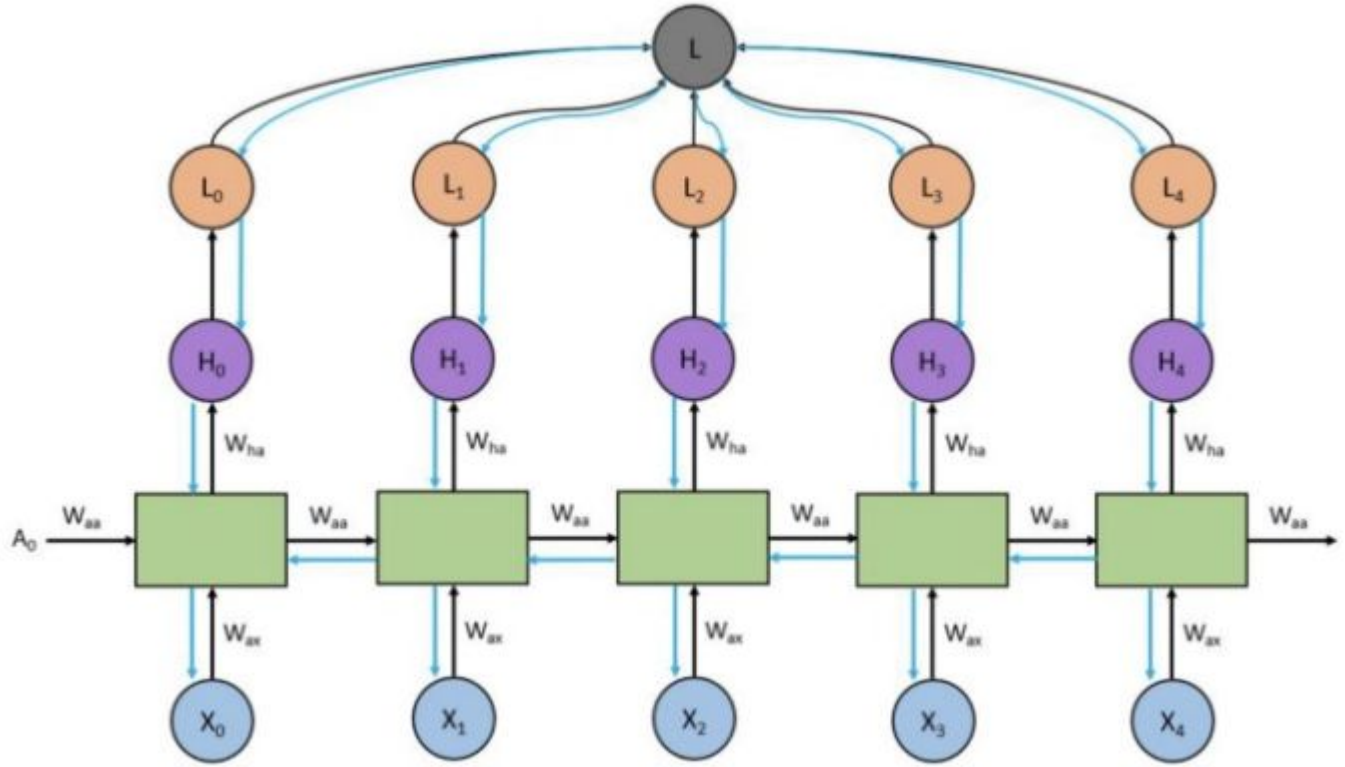
Komórki rekurencyjne

- Niestabilność gradientu
- Długa pamięć krótkotrwała (LSTM)
- Optymalizacja sieci rekurencyjnych
- Bramkowane jednostki rekurencyjne (GRU)

Uczenie sieci



Uczenie sieci



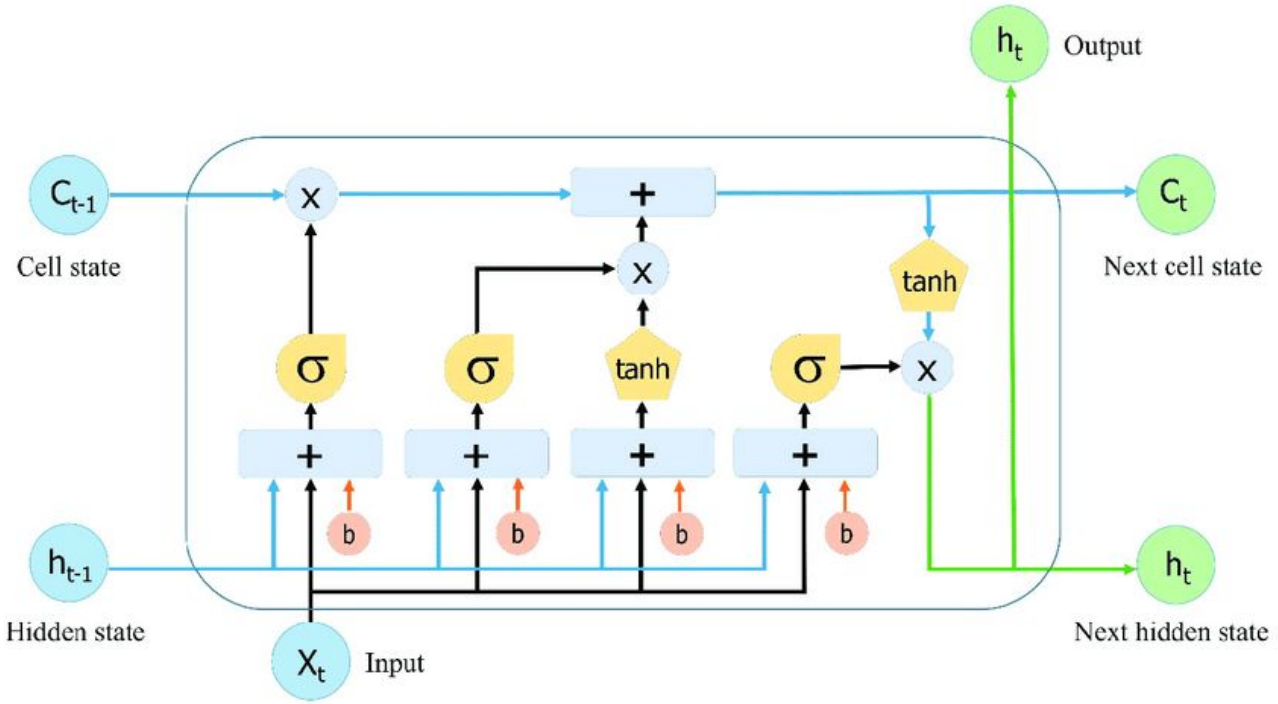
Niestabilność gradientu

- Sieci neuronowe mają tendencje do degradacji sygnału w czasie
- Przykładowo po 50 krokach sygnał jest zdegradowany
- Dlatego trenowanie sieci neuronowych na dłuższych sygnałach czasowych jest trudnym zadaniem
- Istnieje wiele metod zwalczania tej niestabilności

Długa pamięć krótkoterminowa

- W standardowej komórce rekurencyjnej sygnały z odległej przeszłości ulegają wytłumieniu
- Przez to RNN nie mogą się uczyć skomplikowanych zależności
- Problem zauważalny szczególnie w NLP
- Rozwiązaniem może być umożliwienie przechodzenia niezmienionych stanów z przeszłości
- Architektura LSTM umożliwia przejście ze stanu przeszłego do teraźniejszego z lekkimi modyfikacjami
- Oferują wyższą wartość uczenia niż standardowe komórki RNN

Architektura komórki LSTM



Inputs:

- X_t Current input
- C_{t-1} Memory from last LSTM unit
- h_{t-1} Output of last LSTM unit

Outputs:

- C_t New updated memory
- h_t Current output

Nonlinearities:

- σ Sigmoid layer
- tanh Tanh layer
- b Bias

Vector operations:

- x Scaling of information
- \+ Adding information

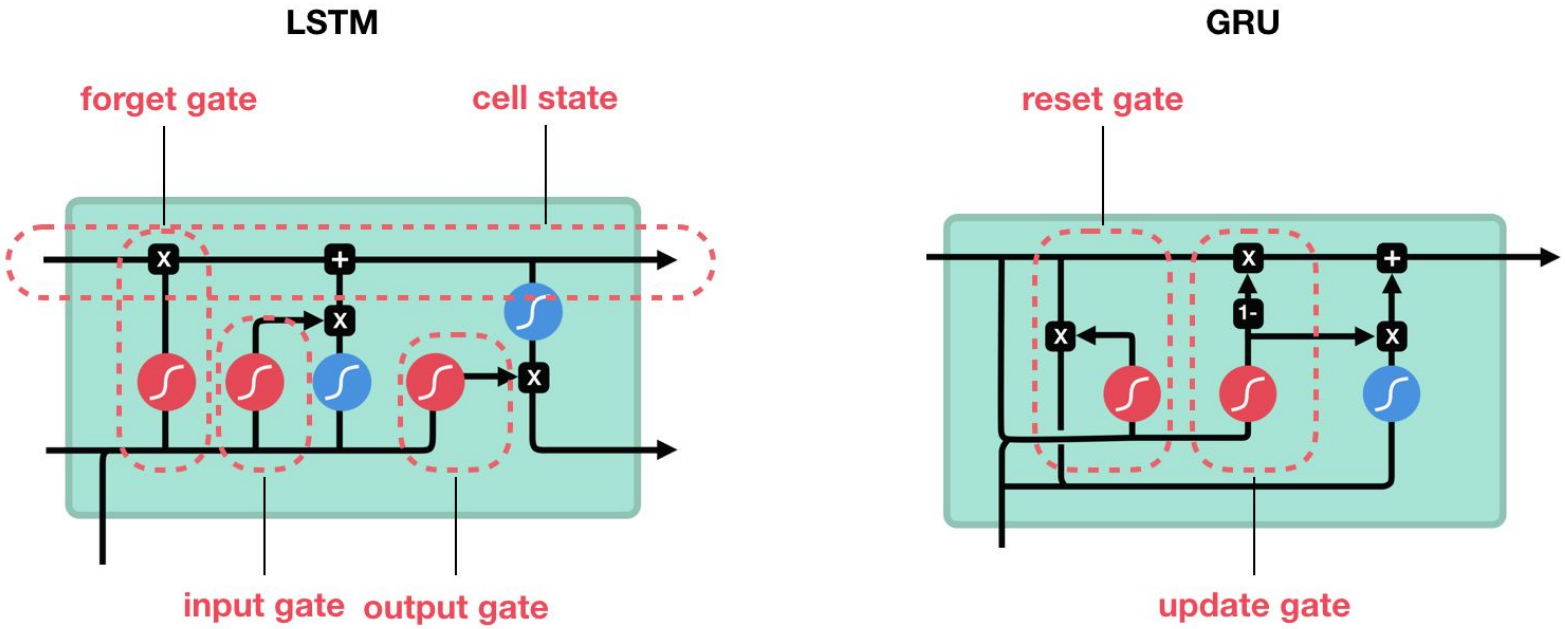
Optymalizacja sieci LSTM

- Komórki LSTM w odróżnieniu od MLP czy CNN wymagają skomplikowanych operacji matematycznych oraz sterowania przepływem
- Trenowanie na dużą skalę bywa wyzwaniem
- Możliwość trenowania z użyciem procesorów graficznych (GPU)

Bramkowane jednostki rekurencyjne (GRU)

- Złożoność obliczeniowa LSTM skłoniła wielu badaczy to stworzenia prostszej obliczeniowo sieci, ale z podobnymi efektami pod względem skuteczności
- GRU usuwa jeden ze składowych elementów LSTM, ale empirycznie wydaje się uzyskiwać wydajność zbliżoną do LSTM

Porównanie GRU i LSTM



sigmoid

tanh

pointwise multiplication

pointwise addition

vector concatenation

Zastosowania sieci rekurencyjnych

- Generowanie danych
- Modele seq2seq
- Przetwarzanie korpusu językowego Penn Treebank
- Przewidywanie indeksów giełdowych
- Zaawansowane modele bazujące na tekście

Zaawansowane zastosowania sieci rekurencyjnych

- Odrzucanie rekurencyjne - wbudowany algorytm mający na celu zapobieganie przeuczaniu się warstw rekurencyjnych
- Tworzenie stosu warstw rekurencyjnych - technika ma na celu zwiększenie mocy tworzenia reprezentacji przez sieć kosztem konieczności wykonywania bardziej złożonych obliczeń
- Dwukierunkowe warstwy rekurencyjne - warstwy przedstawiające te same informacje w sieci rekurencyjnej, rozwiązanie zwiększające dokładność i rozwiązujące problemy ginięcia informacji

Zastosowania sieci rekurencyjnych w praktyce

Będziemy korzystać z danych, które znajdziesz w zeszycie z zaawansowanych zastosowań.

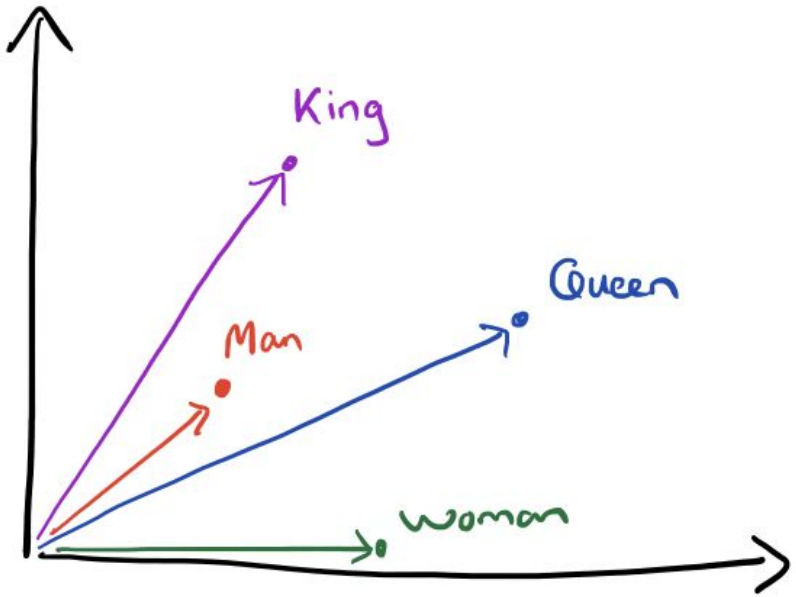
Możesz go pobrać z sekcji
“Dodatkowe materiały do bloku” tutaj.

Osadzanie słów

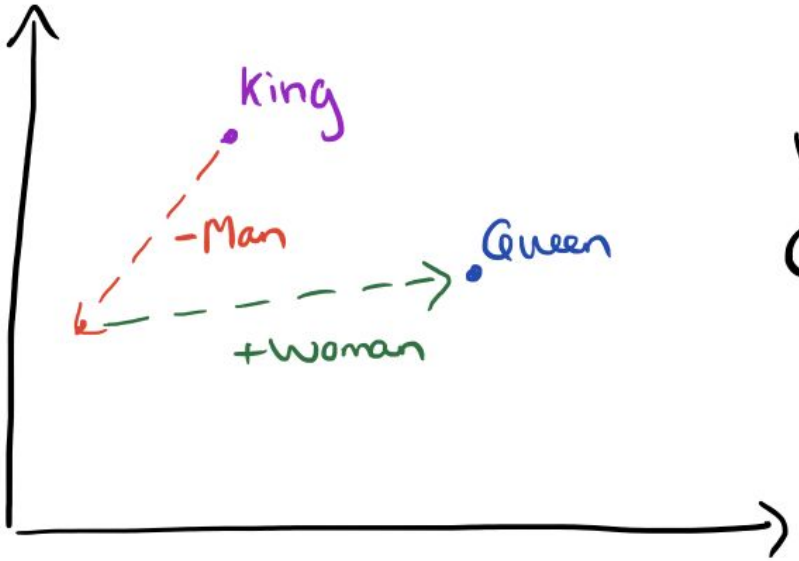
- Częściej można spotkać się z angielskim odpowiednikiem terminu osadzanie słów - word embedding
- Wektory, które tworzymy używając metody one-hot-encoding są binarne, składają się głównie z zer oraz liczba ich wymiarów jest równa liczbie słów wchodzących w skład słownika
- Osadzenia natomiast charakteryzują się niską liczbą wymiarów, są gęstymi wektorami zmiennoprzecinkowymi
- W przeciwieństwie do wektorów pochodzących z one-hot-encoding są uczone na danych

Osadzanie słów

Osadzanie słów

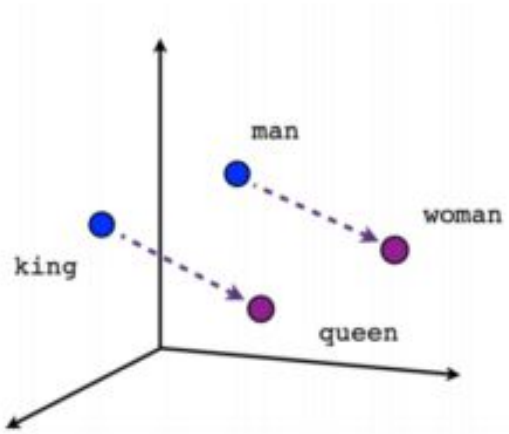


Word Vectors

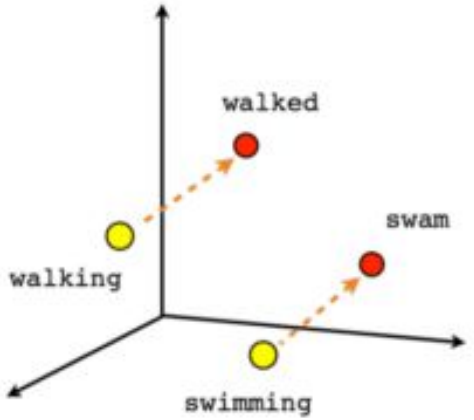


Vector Composition

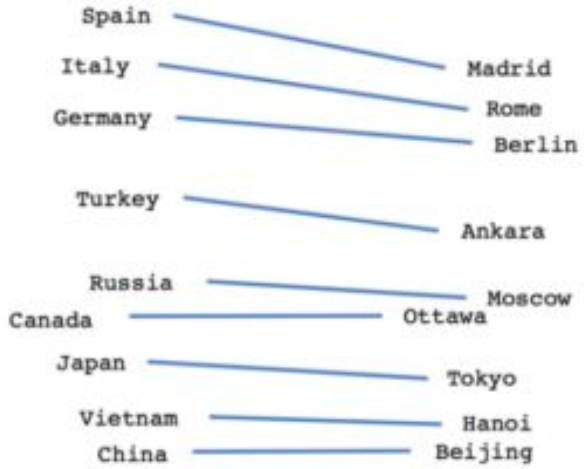
Osadzanie słów



Male-Female



Verb tense



Country-Capital

Tworzenie osadzeń słów

- Mogą być one uczone wraz z głównym zadaniem (np. analizą sentymentu), zaczynamy od losowych wektorów słów, a następnie zawartość tych wektorów jest uczona w sposób taki sam jak wagi sieci neuronowej
- Możemy użyć “uprzednio wytrenowanych osadzeń słów”, które zostały utworzone podczas pracy nad innym zagadnieniem uczenia maszynowego

Uczenie osadzeń słów przy użyciu warstwy osadzającej

- Na początku wybieramy losowy wektor
- Geometryczna zależność między wektorami słów powinna odzwierciedlać semantyczną zależność
- Osadzenia słów powinny reprezentować język w przestrzeni geometrycznej
- W poprawnej przestrzeni synonimy powinny być osadzone w podobnych wektorach słów
- Odległość geometryczna między słowami powinna odzwierciedlać różnice między nimi

Uczenie osadzeń słów przy użyciu warstwy osadzającej

- W przestrzeni osadzeń nie tylko odległość ma znaczenie, ale też kierunek
- Np. mając osadzone słowa: kot, pies, tygrys jako jedna reprezentacja wektorowa
- Ta reprezentacja pozwala nam zakodować pewne zależności semantyczne między słowami np. jako wektor przejścia od zwierzęcia domowego do dzikiego
- Innym przykładem jest dodanie rodzaju żeńskiego do wektora “król” wówczas otrzymujemy wektor królowa

Ładowanie zbioru danych IMDB w celu użycia warstwy embedding

```
from keras.datasets import imdb
from keras import preprocessing
max_features = 10000
 maxlen = 20
(x_train, y_train), (x_test, y_test) =
imdb.load_data(num_words=max_features)
x_train = preprocessing.sequence.pad_sequences(x_train, maxlen=maxlen)
x_test = preprocessing.sequence.pad_sequences(x_test, maxlen=maxlen)
```

Łączenie wszystkich technik - schemat postępowania

- Pobieranie zbioru danych IMDB w postaci tekstowej
- Tokenizacja danych
- Pobieranie osadzeń słów Glove
- Wstępne przetwarzanie osadzeń
- Definiowanie modelu
- Ładowanie osadzeń glove do modelu
- Trenowanie i ewaluacja modelu

Osadzanie słów w praktyce

Będziemy korzystać z danych, które znajdziesz w zeszycie z osadzaniem słów.

Możesz go pobrać z sekcji
“Dodatkowe materiały do bloku” tutaj.

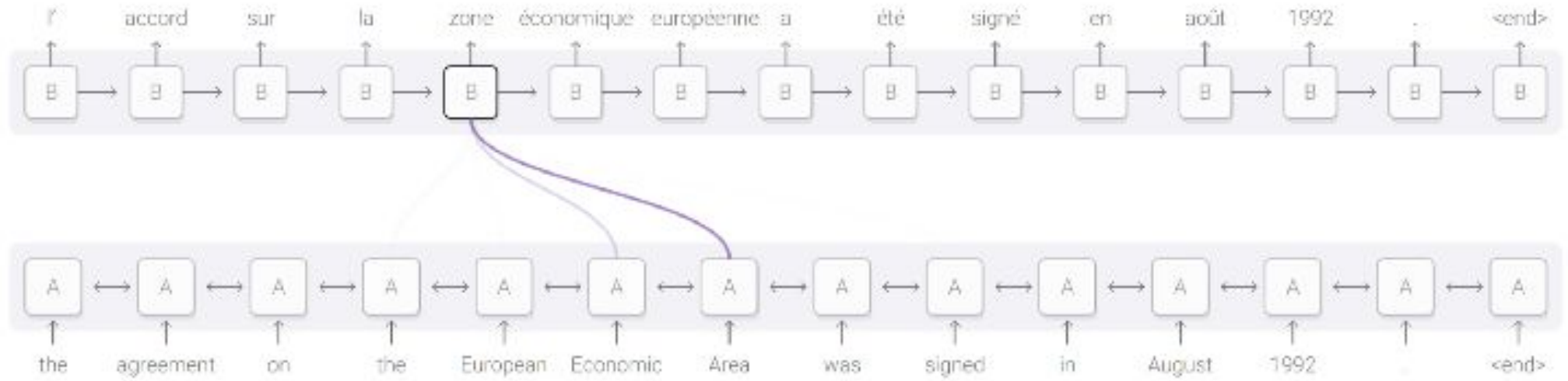
Mechanizm uwagi

“Uważaj na to, co robisz!”

To zdanie, które z pewnością słyszałeś wielokrotnie w dzieciństwie. Rzeczywiście, w życiu codziennym **umiejętności uwagi wzrokowej, słuchowej i poznawczej mają znaczenie dla zrozumienia sytuacji**.

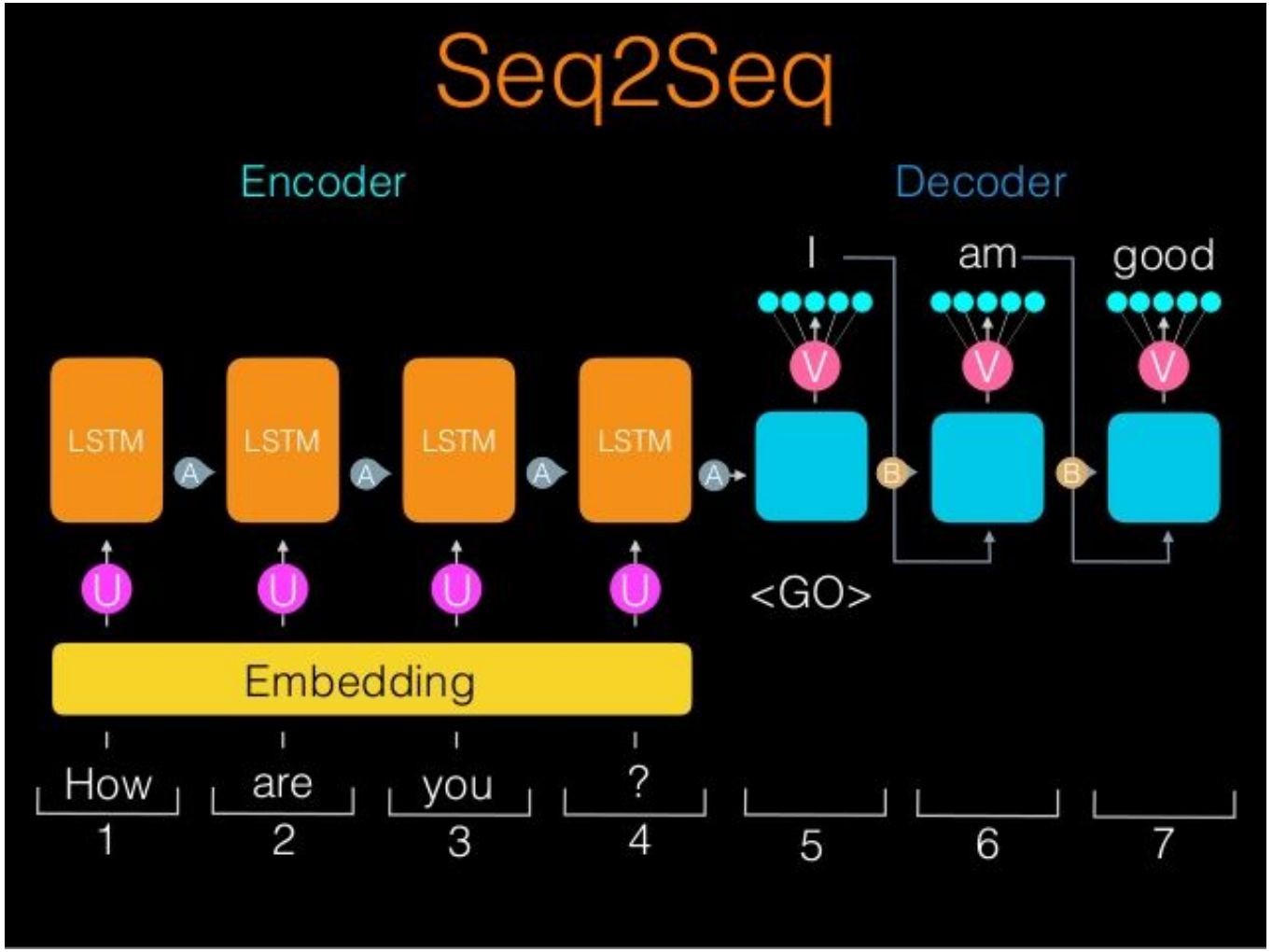
Zwrócenie uwagi na sytuację, a następnie wykorzystanie wspomnień pozwala być bardziej wydajnym, na przykład przy tłumaczeniu tekstu z jednego języka na inny, ponieważ nie tylko wszystkie słowa pierwotnego tekstu do przetłumaczenia nie mają tego samego znaczenia, ale dodatkowo kolejność przetłumaczonych słów niekoniecznie odpowiada kolejności słów w tekście początkowym.

Mechanizm uwagi - działanie



Poniższy obraz dobrze ilustruje tę zasadę: aby wygenerować token „strefy” w sekwencji wyjściowej, dekoder B wykorzystuje względne znaczenie słów „obszar”, a następnie „ekonomiczny”, które są jednak tokenami o wyższej pozycji w sekwencja wejściowa enkodera A.

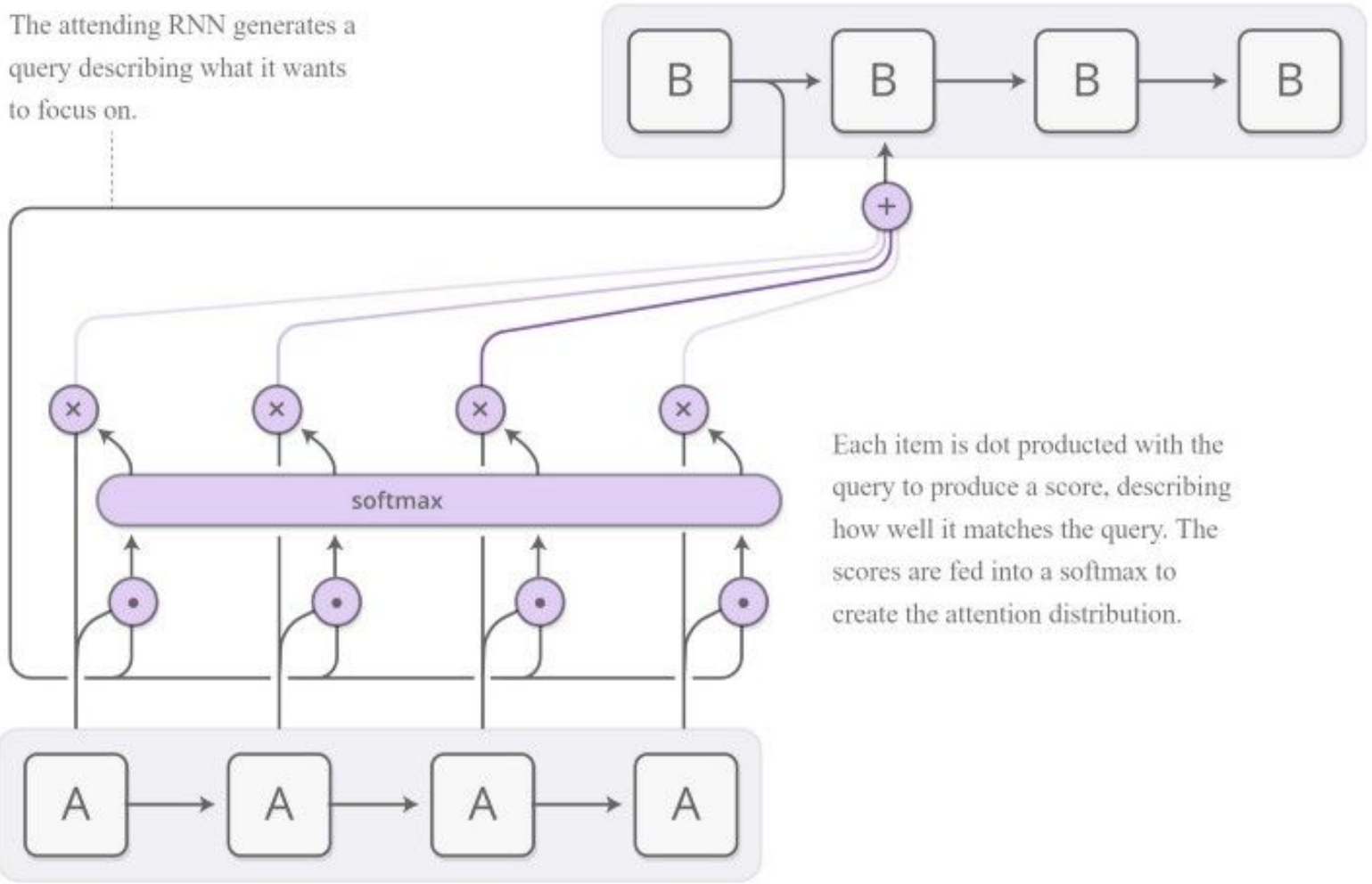
Model seq2seq



Zastosowanie mechanizmu uwagi w seq2seq

- Na podstawie żądania wysłanego przez dekoder (przy użyciu jego ostatniego wektora stanu) każdy wektor stanu wygenerowany przez koder podczas kodowania tokenów sekwencji wejściowej jest odnotowywany (punktowany) przy użyciu w pełni połączonej sieci neuronowej.
- Moglibyśmy wtedy podsumować mechanizm uwagi na sekwencji wejściowej modelu Seq2Seq jako metodę notacji (punktacji) każdego wektora stanu emitowanego przez koder w celu poprawy generowania przez dekoder każdego nowy token sekwencji wyjściowej.
- W pełni połączona sieć neuronowa to sieć neuronowa z ukrytą warstwą, której wartości parametrów są klasycznie uczone podczas uczenia całego modelu Seq2Seq (tj. przez wsteczną propagację gradientu funkcji błędu). Nie ma zatem niezależnego uczenia tej sieci neuronowej ani konieczności modyfikowania funkcji błędu modelu Seq2Seq.

Kodowanie mechanizmu uwagi



Zaawansowane algorytmy NLP - BERT

Google opisuje BERTa jako największą zmianę w systemie wyszukiwania od czasu wprowadzenia prawie pięć lat temu RankBraina, a nawet jedną z największych zmian algorytmicznych w historii wyszukiwania. Informacja o „przybyciu" BERTa i jego zbliżającym się wpływie wywołała niemałe poruszenie w społeczności SEO. Wraz z tym poruszeniem pojawiły się również pewne niejasności co do tego, co właściwie robi BERT, jak działa i co to oznacza dla całej branży. Poniżej postaramy odpowiedzieć na te pytania.

Czym jest BERT?

- **BERT jest przełomowym technicznie modelem służącym do przetwarzania języka naturalnego**, który podbija świat odkąd został wydany w postaci pracy naukowej napisanej przez Jacoba Devlina i współtwórców (Min-Wei Chang, Kenton Lee, Kristina Toutanova; https://arxiv.org/abs/1810.04805) w 2018 r.
- BERT (z języka angielskiego *Bidirectional Encoder Representations from Treansformers*) to wstępnie przeszkolona platforma do nauki języka naturalnego, która jeszcze na etapie badań dała najnowocześniejsze wyniki w przetwarzaniu 11 przydzielonych zadań językowych. Zadania te obejmują między innymi **oznaczanie ról semantycznych, klasyfikację tekstu, przewidywanie kolejnego zdania** etc. BERT pomaga również w ujednoznacznieniu słów polisemicznych w kontekście. Pierwotnie był wyszkolony na całej angielskiej Wikipedii i na zbiorze dzieł związanych ze współczesnym j. ang. Uniwersytetu Brown.

Okoliczności powstania BERTa

- Wyzwania jakie stawia język naturalny (niemal co 2 słowo w języku angielskim ma wiele znaczeń)
- Niejasności leksykalne (słowa o wielu znaczeniach łączą się, by stworzyć wieloznaczeniowe zdania i frazy, które stają się trudniejsze do zrozumienia)
- Polisemia i homonimia
- Homografy i homofony
- Anafora i katafora

Jak wyszukiwarki uczą się języka?

- Współwystępowanie (słowa o podobnym znaczeniu mają tendencję do tego, by były używane w pobliżu siebie w języku naturalnym)
- Podobieństwa i pokrewieństwa (**Słowa, które są podobnymi „rodzajami rzeczy”, mają podobieństwo semantyczne)**
- Grafy wiedzy i repozytoria (dane uporządkowane ułatwiają wyszukiwarkom zrozumienie języka naturalnego)
- Oznaczanie części mowy

Jak BERT pomaga wyszukiwarce zrozumieć język?

- Przeszkolenie w zakresie tekstu nieotagowanego ręcznie
- Dwukierunkowość (możliwości jednoczesnego zobaczenia wszystkich słów w zdaniu i zrozumieniu, w jaki sposób wszystkie te słowa wpływają na kontekst innych wyrażeń w zdaniu)
- Wykorzystanie architektury na wzór transformera (modelu tłumaczeniowego AI - Na wzór transformerów tłumaczeniowych **każde zdanie jest rozłożone na czynniki pierwsze i połączone z kontekstem**.

Polecane materiały związane z tematyką bloku

- https://ksopyla.com/pytorch/siec-rekurencyjna-lstm-do-zliczania-znakow-wprowadzenie/
- https://www.petrospsyllos.com/pl/prezentacje/blog/598-myslaca-maszyna-wstep-do-sztucznej-inteligencji-cz-2-wideo
- https://colah.github.io/posts/2015-08-Understanding-LSTMs/