POLITECHNIKA LUBELSKA
WYDZIAŁ ELEKTROTECHNIKI I INFORMATYKI
INFORMATYKA

Zintegrowany Program Rozwoju Politechniki Lubelskiej - część druga

PODSTAWY SZTUCZNEJ INTELIGENCJI W JĘZYKU PYTHON

Podstawy uczenia maszynowego

Paweł Karczmarek

Fundusze Europejskie Wiedza Edukacja Rozwój
Rzeczpospolita Polska
Unia Europejska Europejski Fundusz Społeczny

Agenda

- Sposób zaliczenia
- Efekty uczenia się
- Omówienie sylabusa
- Zarys historii sztucznej inteligencji
- Cele sztucznej inteligencji
- Uczenie maszynowe
- Podział metod uczenia maszynowego

Sposób zaliczenia przedmiotu

Egzamin z wykładów (51%)

Zaliczenie z laboratorium (51%)

Materiały - Moodle

Kurs Podstawy sztucznej inteligencji w języku Python
Wykład Paweł Karczmarek

PSIwJP_W_PK

psiwjp20212022

Efekty uczenia się

Student:

W zakresie wiedzy:

- **Ma wiedzę dotyczącą tworzenia programów w języku Python**
- **Ma wiedzę dotyczącą własności i stosowania wybranych modeli sztucznej inteligencji w zależności od zastosowań**

W zakresie umiejętności:

- Potrafi programować aplikacje w języku Python w zakresie sztucznej inteligencji
- Potrafi umiejętnie stosować wybrane biblioteki języka Python w celu efektywnego tworzenia aplikacji w zależności od zastosowania

W zakresie kompetencji społecznych:

- Jest gotów do współpracy z kolegami i prowadzącym w ramach zajęć laboratoryjnych
- Jest gotów do podejmowania wyzwań związanych z wykorzystaniem nowoczesnych technik programistycznych i krytycznej oceny tych technik

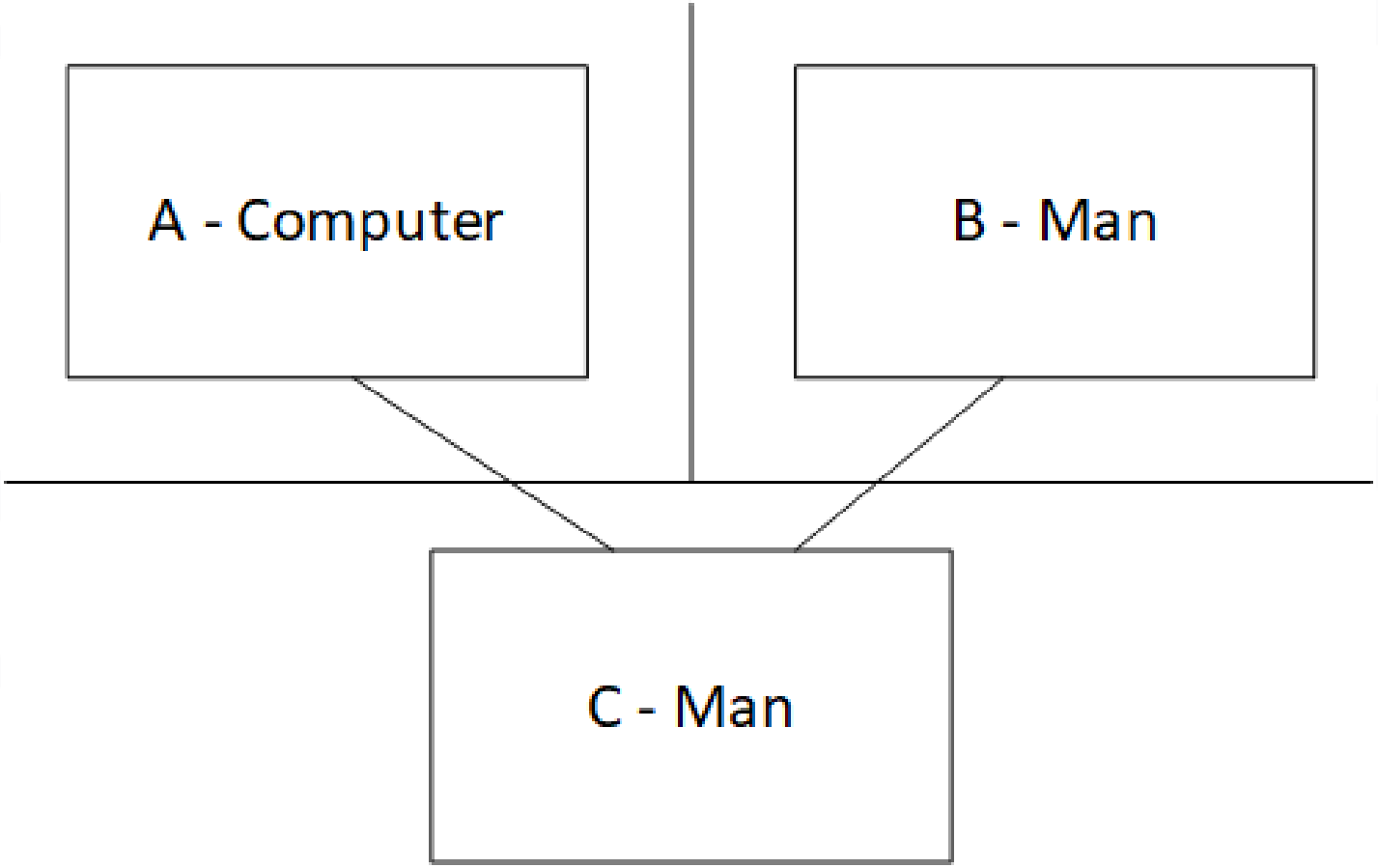
Treści programowe

- Podstawy uczenia maszynowego
- Przegląd środowisk i bibliotek Python. Podstawowe informacje nt języka
- Klasyfikacja najbliższego sąsiada
- Regresja liniowa
- Maszyny wektorów nośnych
- Analiza głównych składowych i liniowa analiza dyskryminacyjna
- Drzewa decyzyjne
- Wprowadzenie do sieci neuronowych
- Uczenie i działanie sieci neuronowej
- Sieci jednowarstwowe, wielowarstwowe i propagacja wsteczna
- Sieci samouczące, samoorganizujące się i rekurencyjne
- Głębokie sieci neuronowe i uczenie głębokich sieci neuronowych
- Głębokie splotowe sieci neuronowe
- Głębokie rekurencyjne sieci neuronowe
- Autoenkodery

Historia (1)

- 1702, G. W. Leibniz: logika w formalnym, matematycznym sensie
- 1854, G. Boole (algebra Boole'a)
- 1810, C. Babbage: wizja obliczeń mechanicznych
- 1840, Ada Lovelace: pierwszy program
- Neurologia: mózg jest elektryczną siecią neuronów
- N. Wiener: podstawy cybernetyki (kontrola i stabilność w sieciach elektrycznych)
- C. Shannon: teoria informacji – sygnały cyfrowe
- A. Turing: teoria obliczeń
- W. Pitts, W. McCulloch: pierwsza sieć neuronowa

Test Turinga



Historia (2)

- 1951, pierwszy program szachowy, poziom amatorski
- Obliczenia symboliczne (programy do dowodzenia twierdzeń logiki)
- Wczesne rozważania filozoficzne nt. SI (czy maszyny mogą posiadać umysły tak, jak ludzie)
- Konferencja w Dartmouth (1956): podstawy SI jako dziedziny (M. Minsky, J. Bigelow, D. M. Mackay, C. Shannon, …)
- Cel: sprecyzować i opracować koncepcję maszyn myślących

Sukcesy (1956-1974) i zima SI (1974-1980)

- ELIZA – pierwszy chatbot
- SHRDLU – wczesny program NLP
- WABOT, WABOT-1 – inteligentny humanoidalny robot/android

- Ograniczona moc komputerów
- Kombinatoryczna eksplozja
- Paradoks Moraveca
- Krytyka ze strony filozofów
- Ograniczenia perceptronów

Lata osiemdziesiąte

- Systemy ekspertowe
  - odpowiadają na pytania lub rozwiązują problemy, dotyczące określonej dziedziny wiedzy, stosując reguły logiczne wywodzące się z wiedzy ekspertów
- Japonia: komputer piątej generacji, Prolog
- Sieć Hopfielda
- Propagacja wsteczna (Hinton & Rumelhart)
- Lata 1987-1993: druga zima SI (bankructwa firm, obcięcie funduszy, brak realizacji założonych celów)

Deep Blue

- Komputer grający w szachy (IBM)
- Deep Blue wygrał pojedynek z Garrym Kasparovem w 1996
- Analiza 200 000 000 ruchów na sekundę
- 2006: Deep Fritz

Prawo Moore'a

- Optymalna liczba tranzystorów w układzie scalonym zwiększa się w kolejnych latach zgodnie z trendem wykładniczym (podwaja się w niemal równych odcinkach czasu)
- Moc obliczeniowa komputerów podwaja się co 24 miesiące (wersja)
- Rozwój od 1993

Głębokie uczenie maszynowe (Deep learning)

- A. G. Ivakhnenko, V. G. Lapa (1967)
- Y. LeCun et al. (1989) standardowy algorytm propagacji wstecznej w głębokiej sieci neuronowej
- G. E. Hinton et al. (2006): Nauka wielu warstw reprezentacji
- GPU – popularyzacja, wpływ rynku gier

Zastosowania sztucznej inteligencji

- Chatboty
- eCommerce
- Komunikacja w miejscu pracy
- Zarządzanie zasobami ludzkimi
- Opieka zdrowotna
- Bezpieczeństwo cybernetyczne
- Logistyka i łańcuch dostaw
- Zakłady sportowe
- Usprawnianie produkcji
- Kasyna i hotele
- Sprzedaż

Cele SI

- Tradycyjne cele badań nad sztuczną inteligencją obejmują rozumowanie, reprezentację wiedzy, planowanie, uczenie się, przetwarzanie języka naturalnego, percepcję oraz zdolność poruszania się i manipulowania obiektami.
- Inteligencja ogólna (umiejętność rozwiązywania arbitralnego problemu) jest jednym z długoterminowych celów tej dziedziny.

Uczenie maszynowe

- Nauka o algorytmach komputerowych, które usprawniają się automatycznie dzięki doświadczeniu i wykorzystaniu danych.
- Jest postrzegane jako część SI. Algorytmy uczenia maszynowego budują model na podstawie danych przykładowych, znanych jako dane uczące, w celu przewidywania lub podejmowania decyzji bez wyraźnego zaprogramowania do tego.
- Są wykorzystywane tam, gdzie opracowanie konwencjonalnych algorytmów do wykonywania potrzebnych zadań jest trudne lub niemożliwe

Tradycyjne podejścia

- Uczenie nadzorowane
- Uczenie nienadzorowane
- Uczenie się ze wzmocnieniem

Uczenie nadzorowane (z nauczycielem)

- Program dostaje przykładowe dane wejściowe i pożądane wyniki, podane przez „nauczyciela”, a celem jest nauczenie się ogólnej zasady, która odwzorowuje dane wejściowe na wyniki.

Uczenie nienadzorowane (bez nauczyciela)

- Algorytm uczący się nie otrzymuje żadnych etykiet, aby znaleźć strukturę danych wejściowych. Nienadzorowane uczenie się może być celem samym w sobie (odkrywanie ukrytych wzorców w danych) lub środkiem do osiągnięcia celu (uczenie się cech).

Uczenie się ze wzmocnieniem

- Program komputerowy wchodzi w interakcję z dynamicznym środowiskiem, w którym musi wykonać określony cel (np. prowadzić pojazd lub grać z przeciwnikiem). Podczas poruszania się po swojej przestrzeni problemowej program otrzymuje informację zwrotną analogiczną do nagród, którą stara się zmaksymalizować.

Typowe modele

- Sieci neuronowe
- Drzewa decyzyjne
- Maszyny wektorów nośnych (SVM)
- Analiza regresji
- Sieci Bayesowskie
- Algorytmy genetyczne
- …

Ograniczenia i kwestie etyczne

- Błędy w danych
- Brak (właściwych) danych
- Brak dostępu do danych
- Stronniczość danych
- Prywatność (np. biometria, dane medyczne, anonimizacja danych)
- Niewłaściwie dobrane zadania i algorytmy
- Niewłaściwi narzędzia i ludzie
- Brak zasobów
- Problemy z oceną
- Bezpieczeństwo (np. samochody autonomiczne)

Fei-Fei Li

„W AI nie ma nic sztucznego… Jest inspirowana przez ludzi, tworzona przez ludzi i – co najważniejsze – wpływa na ludzi. Jest to potężne narzędzie, które dopiero zaczynamy rozumieć, a to jest głęboka odpowiedzialność.”

Literatura podstawowa

- Chollet F., Deep Learning. Praca z językiem Python i biblioteką Keras, Helion, Gliwice 2019
- Raschka S., Mirjalili V., Python. Uczenie maszynowe. Wydanie II, Helion, Gliwice 2019

Literatura uzupełniająca

- Bonaccorso G., Algorytmy uczenia maszynowego. Zaawansowane techniki implementacji, Helion, Gliwice 2019
- Géron A., Uczenie maszynowe z użyciem Scikit-Learn i TensorFlow, Helion, Gliwice 2018
- Gift N., AI - podejście pragmatyczne. Wprowadzenie do uczenia maszynowego opartego na chmurze, Promise, Warszawa 2018
- Goodfellow I., Bengio Y., Courville A., Deep Learning. Współczesne systemy uczące się, PWN, Warszawa 2018
- Joel Grus , Data science od podstaw. Analiza danych w Pythonie, Helion, 2018
- Hearty J., Zaawansowane uczenie maszynowe z językiem Python, Helion, Gliwice 2017
- Karczmarek P., Selected Problems of Face Recognition and Decision-Making Theory, Wydawnictwo Politechniki Lubelskiej, Lublin 2018
- Osinga D., Deep Learning. Receptury, Helion, Gliwice 2019
- Patterson J., Gibson A., Deep Learning. Praktyczne wprowadzenie, Helion, Gliwice 2018
- Rutkowski L., Metody i techniki sztucznej inteligencji, PWN, Warszawa 2012
- Tadeusiewicz R., Sieci neuronowe, Akademicka Oficyna Wydawnicza, Warszawa 1993

Pytania?

Dziękuję za uwagę!

POLITECHNIKA LUBELSKA
WYDZIAŁ ELEKTROTECHNIKI I INFORMATYKI
INFORMATYKA

Zintegrowany Program Rozwoju Politechniki Lubelskiej - część druga

PODSTAWY SZTUCZNEJ INTELIGENCJI W JĘZYKU PYTHON

Przegląd środowisk i bibliotek Python. Podstawowe informacje nt. języka

Paweł Karczmarek

Fundusze Europejskie Wiedza Edukacja Rozwój
Rzeczpospolita Polska
Unia Europejska Europejski Fundusz Społeczny

Agenda

- Python i jego historia
- Środowiska
- Kluczowe biblioteki
- Wybrane informacje nt. składni

Python

Twórca: Gino van Rossum (1991)

Zarządzany jest przez Python Software Foundation

Python

Wspiera paradygmaty

- Obiektowy
- Imperatywny
- Funkcyjny

Posiada:

- Dynamiczny system typów
- Automatyczne zarządzanie pamięcią
- Często używany jako język skryptowy
- Używany w wielu językach operacyjnych
- Interpretery dla różnych platform

Python - zastosowania

Typowe zastosowania

- Analiza i wizualizacja danych
- Aplikacje sieciowe
- Nauka programowania
- Proste gry
- Badania naukowe
- …

Historia wersji

- 0.9.0 (1991)
- 1.0 (1994)
- 2.0 (2000)
- 3.0 (2008) – brak kompatybilności z poprzednimi wersjami!
- Aktualna wersja (od 2020): 3.9

Wybrane środowiska

- PyCharm
- VS Code
- Sublime Text
- Jupyter
- Spyder

NumPy

- Służy do przetwarzania dużych, wielowymiarowych tablic i macierzy, z pomocą dużego zbioru funkcji matematycznych wysokiego poziomu.
- Przydatna do podstawowych obliczeń naukowych w uczeniu maszynowym (algebra liniowa, transformacja Fouriera i liczby losowe).
- Zaawansowane biblioteki (TensorFlow) używają wewnętrznie NumPy do manipulacji tensorami.

SciPy

- Zawiera różne moduły do optymalizacji, algebry liniowej, integracji i statystyki.
- SciPy jest również bardzo przydatna do manipulacji obrazami.

Scikit-learn

- Jedna z bibliotek ML dla klasycznych algorytmów.
- Jest zbudowana na bazie dwóch podstawowych bibliotek (NumPy i SciPy).
- Obsługuje większość nadzorowanych i nienadzorowanych algorytmów uczenia się.
- Scikit-learn może być również używana do eksploracji danych i analizy danych.

Theano

- Służy do efektywnego definiowania, oceny i optymalizacji wyrażeń matematycznych obejmujących wielowymiarowe tablice.
- Osiąga się to poprzez optymalizację wykorzystania CPU i GPU.
- Stosowana do testów jednostkowych i samoweryfikacji w celu wykrywania i diagnozowania różnego rodzaju błędów.
- Używana w dużych, intensywnych obliczeniowo projektach naukowych.
- Może być stosowana w mniejszych projektach.

TensorFlow

- Biblioteka open-source do wysokowydajnych obliczeń numerycznych opracowana przez zespół Google Brain.
- Framework, który obejmuje definiowanie i przeprowadzanie obliczeń z wykorzystaniem tensorów.
- Obsługuje głębokie sieci neuronowe (badania i zastosowania).

Keras

- Interfejs API sieci neuronowych wysokiego poziomu, który może działać w oparciu o TensorFlow, CNTK lub Theano.
- Może działać bezproblemowo zarówno na procesorze, jak i na GPU.
- Dobrze udokumentowana i stosowana przez początkujących.
- Pozwala na łatwe i szybkie prototypowanie.
- Twórca: F. Chollet.

PyTorch

- Biblioteka uczenia maszynowego typu open source oparta na Torch, która jest zaimplementowana w C.
- Posiada szeroki wybór narzędzi i bibliotek obsługujących wizję komputerową, NLP, i in.
- Umożliwia programistom wykonywanie obliczeń na tensorach z akceleracją GPU
- Pomaga w tworzeniu wykresów obliczeniowych.

Pandas

- Biblioteka do analizy danych.
- Została opracowana specjalnie do ekstrakcji i przygotowywania danych.
- Zapewnia struktury danych wysokiego poziomu i szeroką gamę narzędzi do analizy danych.
- Zapewnia wiele wbudowanych metod wyszukiwania, łączenia i filtrowania danych.

Matplotlib

- Biblioteka do wizualizacji danych.
- Przydatna do zwizualizowania wzorców w danych.
- Biblioteka do kreślenia wykresów.
- Moduł pyplot ułatwia programistom kreślenie, ponieważ zapewnia funkcje sterowania stylami linii, właściwościami czcionek, osiami formatowania itp.
- Zapewnia różnego rodzaju wykresy do wizualizacji danych, np. histogram, wykresy słupkowe i in.

Listy

```
mojalista = ["xyz", 1, -2.34, 5+6j, [7, 8]]
print mojalista[2]   # -2.34
```

Reprezentacja danych: wielowymiarowe tablice w NumPy - tensory

- Skalary – tensory 0-wymiarowe
- Wektory – tensory 1-wymiarowe
- Macierze – tensory 2-wymiarowe
- Tensory 3- i wielo-wymiarowe

Przykład 1

```
>>> import numpy as np
>>> x = np.array(10)
>>> x
array(10)
>>> x.ndim
0
```

Przykład 2

```
>>> x = np.array([[[1, 2, 3, 4, 5],
                   [6, 7, 8, 9, 10],
                   [11, 12, 13, 14, 15]],
                  [[16, 17, 18, 19, 20],
                   [21, 22, 23, 24, 25],
                   [26, 27, 28, 29, 30]],
                  [[31, 32, 33, 34, 35],
                   [36, 37, 38, 39, 40],
                   [41, 42, 43, 44, 45]]])
>>> x.ndim
3
```

Właściwości tensorów

- Liczba osi – rząd (0, 1, 2, 3, … dla skalarów, wektorów itd.).
- Kształt – krotka – w odniesieniu do Przykładu 2 jest postaci (3, 3, 5).
- Typ danych (często określany dtype, np. float32, uint8, float64, char).

Pytania?

Dziękuję za uwagę!

POLITECHNIKA LUBELSKA
WYDZIAŁ ELEKTROTECHNIKI I INFORMATYKI
INFORMATYKA

Zintegrowany Program Rozwoju Politechniki Lubelskiej - część druga

PODSTAWY SZTUCZNEJ INTELIGENCJI W JĘZYKU PYTHON

Klasyfikacja najbliższego sąsiada

Paweł Karczmarek

Fundusze Europejskie Wiedza Edukacja Rozwój
Rzeczpospolita Polska
Unia Europejska Europejski Fundusz Społeczny

Agenda

- Klasyfikator najbliższego sąsiada
- Klasyfikator *K* najbliższych sąsiadów
- Wagi
- Miary podobieństwa i niepodobieństwa
- Jakość klasyfikacji

Idea klasifikacji *K* najbliższych sąsiadów (Thomas Cover)

- Dane wejściowe: *K* najbliższych przykładów uczących w przestrzeni cech
- Wyjście: Przynależność do klasy
- Obiekt jest klasyfikowany na podstawie głosowania sąsiadów. Jest przypisywany do klasy najczęściej występującej wśród *K* najbliższych sąsiadów
- *K* jest liczbą naturalną, często *K*=1 (najbliższy sąsiad)

Algorytm (najbliższego sąsiada - NN)

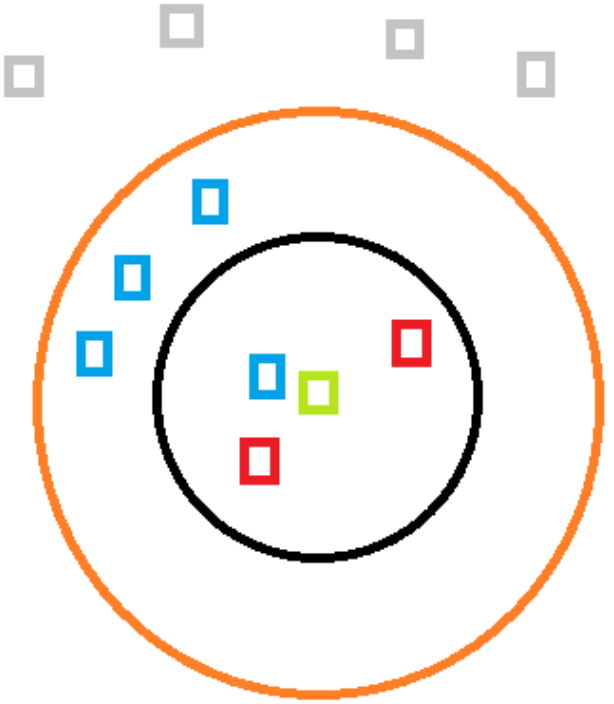
- Dane są przykłady uczące (dane treningowe)
- Uczenie polega jedynie na przechowywaniu wektorów cech i etykiet klas
- Klasyfikacja:
- Nieoznakowany wektor jest klasyfikowany przez przypisanie etykiety, która jest najbliższa temu punktowi
- Zauważmy, że K=1

Algorytm (*K*NN)

- Dane są przykłady uczące (zbiór uczący)
- Uczenie polega jedynie na przechowywaniu wektorów cech i etykiet klas (j.w.)
- Klasyfikacja:
- Wektor testowy jest klasyfikowany przez przypisanie etykiety, która jest najczęstsza wśród *K* prób uczących znajdujących się najbliżej tego punktu. Uwaga: występuje tu głosowanie większością głosów.
- Zamiast określać prawdopodobieństwo klasycznie, za pomocą stosunku liczby sukcesów do liczby wszystkich prób, oblicza się szansę, czyli stosunek prawdopodobieństwa sukcesu do prawdopodobieństwa porażki.

Przykład

- NN: zielony punkt należy do klasy punktów niebieskich
- 3-NN: zielony punkt należy do klasy punktów czerwonych
- 6-NN: zielony punkt należy do klasy punktów niebieskich



Wagi

- Różne elementy wektora mogą być opisywane przez różne wagi
- Jest to ważne, gdy różne cechy (np. w biometrii lub algorytmach podejmowania decyzji, w których kryteria są ważone) mogą mieć różne znaczenie (np. okolice oczu, okolice nosa i okolice ust).
- Suma wag powinna wynosić 1 (normalizacja)

Wagi: Przykład (biometria)

$$d(\boldsymbol{x}, \boldsymbol{y}) = \sqrt{\sum_{i=1}^{10} w_i (x_i - y_i)^2}$$

gdzie parametry oczu (i=1,2,3,4) są ważone przez 0.11, nosa (i=5,6,7) przez 0.09 i ust (i>7) przez 0.096.

Miary podobieństwa/niepodobieństwa

$$\boldsymbol{x}, \boldsymbol{y}, \boldsymbol{w} \in \mathbb{R}^n$$

- Braya-Curtisa $m(\boldsymbol{x}, \boldsymbol{y}) = \frac{\sum_{i=1}^{n} |x_i - y_i|}{\sum_{i=1}^{n} |x_i + y_i|}$
- Canberra $m(\boldsymbol{x}, \boldsymbol{y}) = \sum_{i=1}^{n} \frac{|a_i - b_i|}{|a_i| + |b_i|}$
- Czebyszewa $m(\boldsymbol{x}, \boldsymbol{y}) = \max_i |x_i - y_i|$
- χ^2 $m(\boldsymbol{x}, \boldsymbol{y}) = \sum_{i=1}^{n} \frac{(x_i - y_i)^2}{x_i + y_i}$

Miary podobieństwa/niepodobieństwa

- Korelacji

$$m(\boldsymbol{x},\boldsymbol{y}) = 1 - \frac{\sum_{i=1}^{n}\left(\left(x_i - \frac{1}{n}\sum_{j=1}^{n} x_j\right)\left(y_i - \frac{1}{n}\sum_{j=1}^{n} y_j\right)\right)}{\left(\sum_{i=1}^{n}\left(x_i - \frac{1}{n}\sum_{j=1}^{n} x_j\right)^2\right)^{\frac{1}{2}}\left(\sum_{i=1}^{n}\left(y_i - \frac{1}{n}\sum_{j=1}^{n} y_j\right)^2\right)^{\frac{1}{2}}}$$

- Kosinusowa $m(\boldsymbol{x},\boldsymbol{y}) = 1 - \frac{\sum_{i=1}^{n}(x_i y_i)}{\left(\sum_{i=1}^{n} {x_i}^2\right)^{\frac{1}{2}}\left(\sum_{i=1}^{n} {y_i}^2\right)^{\frac{1}{2}}}$
- Euklidesowa $m(\boldsymbol{x},\boldsymbol{y}) = \left(\sum_{i=1}^{n}(x_i - y_i)^2\right)^{\frac{1}{2}}$

Miary podobieństwa/niepodobieństwa

- Manhattan $m(\boldsymbol{x}, \boldsymbol{y}) = \sum_{i=1}^{n} |x_i - y_i|$
- Mediana wartości bezwzględnych $m(\boldsymbol{x}, \boldsymbol{y}) = \text{med}_i |x_i - y_i|$
- Mediana kwadratów różnic $m(\boldsymbol{x}, \boldsymbol{y}) = \text{med}_i (x_i - y_i)^2$
- Zmodyfikowana Euklidesowa $m(\boldsymbol{x}, \boldsymbol{y}) = \frac{\sum_{i=1}^{n} (x_i - y_i)^2}{\sum_{i=1}^{n} x_i^2 \sum_{i=1}^{n} y_i^2}$

Miary podobieństwa/niepodobieństwa

- Zmodyfikowana Manhattan $m(\boldsymbol{x}, \boldsymbol{y}) = \frac{\sum_{i=1}^{n} |x_i - y_i|}{\sum_{i=1}^{n} |x_i| \sum_{i=1}^{n} |y_i|}$
- Kwadrat Euklidesowej $m(\boldsymbol{x}, \boldsymbol{y}) = \sum_{i=1}^{n} (x_i - y_i)^2$
- Ważona kosinusowa $m(\boldsymbol{x}, \boldsymbol{y}) = 1 - \frac{\sum_{i=1}^{n} \frac{x_i y_i}{\sqrt{w_i}}}{\left(\sum_{i=1}^{n} {x_i}^2\right)^{\frac{1}{2}} \left(\sum_{i=1}^{n} {y_i}^2\right)^{\frac{1}{2}}}$

Miary podobieństwa/niepodobieństwa

- Ważona Manhattan $m(\boldsymbol{x}, \boldsymbol{y}) = \sum_{i=1}^{n} \frac{|x_i - y_i|}{\sqrt{w_i}}$
- Ważony kwadrat Euklidesowej $m(\boldsymbol{x}, \boldsymbol{y}) = \sum_{i=1}^{n} \frac{(x_i - y_i)^2}{\sqrt{w_i}}$
- Hellingera (współczynnik Bhattacharyya)

$$m(\boldsymbol{x}, \boldsymbol{y}) = \sqrt{1 - \frac{\sum_{i=1}^{n} \sqrt{x_i y_i}}{\sqrt{\sum_{i=1}^{n} x_i \sum_{i=1}^{n} y_i}}}$$

Zalety i wady metody NN

Zalety

- Intuicyjna
- Sprawdza się w prostych problemach

Wady

- Elementy klasy z wieloma elementami mają tendencję do dominacji w przewidywaniu przynależności nowego wektora testowego
- Wysoka wymiarowość problemów
- Słaba jakość klasyfikacji w niektórych problemach (np. w porównaniu do innych metod)

Redukcja wymiarowości

- Analiza głównych składowych (PCA)
- Liniowa analiza dyskryminacyjna (LDA)
- Analiza korelacji kanonicznej (CCA)
- Ważenie (ocena ekspercka) w oparciu o heurystykę/doświadczenie

Terminologia w problemach klasyfikacji (tablica pomyłek)

- Prawdziwie pozytywna/True positive (TP)
  - równoważne „właściwemu trafieniu”
- Prawdziwie negatywna/True negative (TN)
  - równoważne poprawnemu odrzuceniu
- Fałszywie pozytywna/False positive (FP)
  - równoważne fałszywemu alarmowi, błąd typu I
- Fałszywie negatywna/False negative (FN)
  równoważna pomyłce, błąd typu II

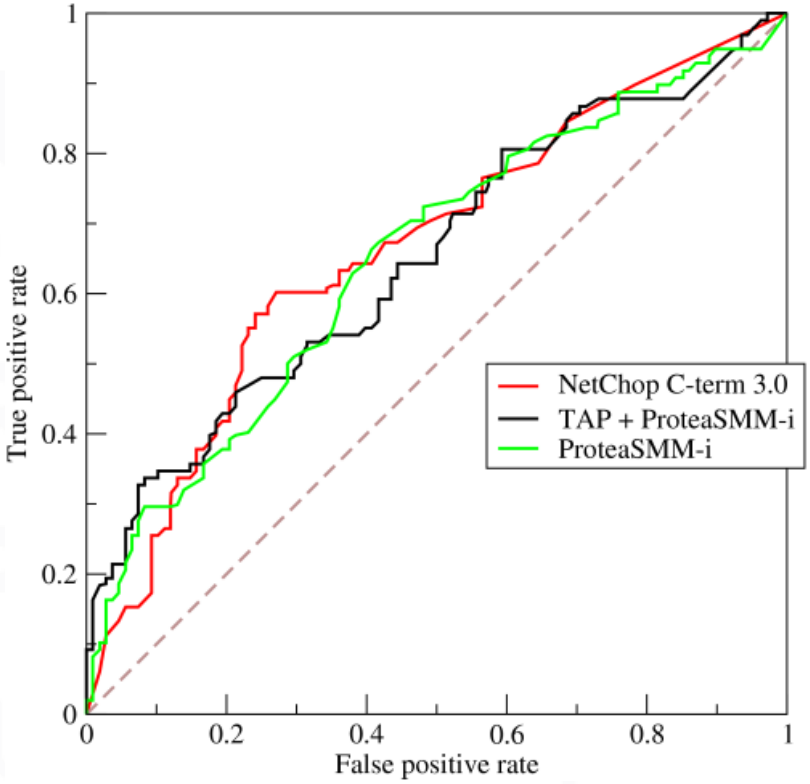
Miary

- Czułość (sensitivity)
  TPR = TP/(TP + FN)
- Swoistość (specificity)
  TNR = TN/(TN+FP)
- Precyzja (precision)
  PPV = TP/(TP + FP)
- Dokładność (accuracy)
  ACC = (TP + TN)/(TP + TN + FP + FN)

Wartości TP, TN, FP, FN rozumie się jako sumy

Inne miary

- Krzywa ROC - to wykres graficzny, który ilustruje zdolność diagnostyczną systemu klasyfikatorów binarnych, ponieważ jego próg dyskryminacji jest zróżnicowany
- AUROC (pole pod krzywą ROC) – im większe, tym lepszy jest klasyfikator
- Miara F1 F1=2TP/(2TP+FP+FN)
- Źródło obrazu: https://commons.wikimedia.org/wiki/File:Roccurves.png
- Licencja: https://en.wikipedia.org/wiki/GNU_Free_Documentation_License



Pytania?

Dziękuję za uwagę!

POLITECHNIKA LUBELSKA
WYDZIAŁ ELEKTROTECHNIKI I INFORMATYKI
INFORMATYKA

Zintegrowany Program Rozwoju Politechniki Lubelskiej - część druga

PODSTAWY SZTUCZNEJ INTELIGENCJI W JĘZYKU PYTHON

Regresja liniowa

Paweł Karczmarek

Fundusze Europejskie Wiedza Edukacja Rozwój
Rzeczpospolita Polska
Unia Europejska Europejski Fundusz Społeczny

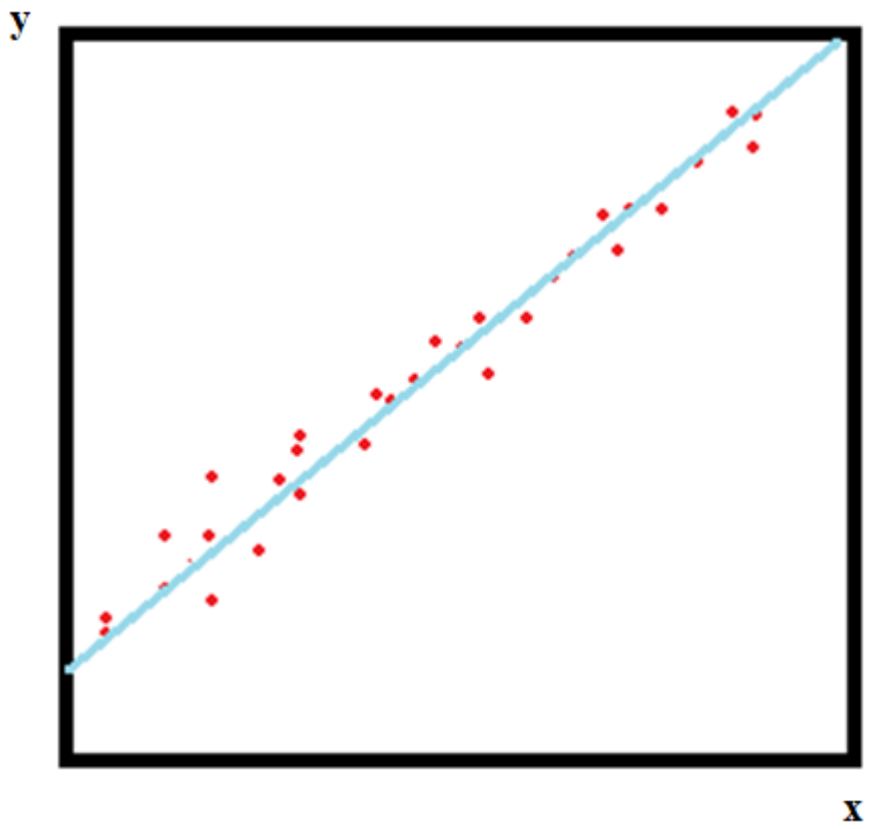
Agenda

- Idea
- Regresja liniowa
- Wielokrotna regresja liniowa
- Zastosowania

Idea

- Wejście:
  Dane z wieloma zmiennymi
- Pytanie:
  Jak są ze sobą powiązane zmienne?
- Regresja to zbiór technik służących do oszacowania zależności

Idea



Linia trendu

Model

- Celem jest dopasowanie linii (prostej)

  $y=a+bx$

- x – zmienna niezależna/zmienna predykcyjna/zmienna objaśniająca
- y – zmienna zależna/zmienna odpowiedzi/zmienna objaśniana

Własności linii

- *b* - nachylenie linii. Jeśli jest bliskie 0, związek jest niewielki lub żaden.
  Jeśli ma duże wartości dodatnie lub ujemne, to istnieje odpowiednio duża dodatnia lub ujemna zależność.
- *a* – przecięcie prostej.

Podejście probabilistyczne

- Jak dobra jest linia?
- Aby to ocenić, możemy dodać tzw. szum Gaussowski:

$$y_i = \mathrm{a} + \mathrm{b}x_i + \varepsilon_i$$
$$\varepsilon \sim N(0, \sigma^2)$$

- a, b i σ są traktowane jak stałe (deterministyczne), ale nieznane.

Rozwiązanie: Regresja metodą najmniejszych kwadratów

- Założenie: Dane $(x_1, y_1), \ldots, (x_n, y_n)$ są wygenerowane
- Wówczas możemy określić linię, dla której prawdopodobieństwo danych jest największe. Rozwiązaniem jest wynik zagadnienia optymalizacyjnego w postaci:

$$\min_{a,b}: \sum_{i=1}^{n} (y_i - (a + bx_i))^2$$

Rozwiązanie

$$\hat{b} = \frac{\sum_{i=1}^{n} x_i y_i - \frac{1}{n}\sum_{i=1}^{n} x_i \sum_{i=1}^{n} y_1}{\sum_{i=1}^{n} {x_i}^2 - \frac{1}{n}\left(\sum_{i=1}^{n} x_i\right)^2}$$

$$\hat{a} = \bar{y} - \hat{b}\bar{x}$$

gdzie $\bar{x}$ oraz $\bar{y}$ są odpowiednio średnimi po wartościach *x* oraz *y*.

Predykcja

$$\bar{y}(x') = \hat{a} + \hat{b}x'$$

gdzie x' jest dowolną wartością.
Jeżeli $x' \in [\min_i x_i, \max_i x_i]$, problem jest nazywany interpolacją.
Przypadek przeciwny jest nazywany ekstrapolacją.

Regresja wielokrotna

Załóżmy teraz, że zamiast pojedynczej, skalarnej, wartości *x,* dysponujemy wektorem $(x_1, \dots, x_p)$ dla każdego punktu *i.*
Zatem istnieje *n* punktów danych, każdy z *p* cechami.
Celem jest znaleźć *y* dla każdego punktu danych takie, że

$$y = B_1 x_1 + B_2 x_2 + \cdots + B_p x_p$$

Właściwości reprezentacji

- Wiele zmiennych zależnych
- Nieliniowość

Model

- Dane wejściowe: X – macierz, w której każdy wiersz odpowiada punktowi danych a kolumna odpowiada cechom
- y – n-elementowy wektor kolumnowy
- Model liniowy:

$$y = XB + \varepsilon$$

- B – p-elementowy wektor współczynników
- ε – wektor zdefiniowany tak, jak poprzednio

Rozwiązanie

- Zagadnienie optymalizacyjne:

$$\min_{B} \sum_{i=1}^{n} (y_i - X_i B)^2$$

(X_i - wiersze macierzy X)

- Rozwiązanie:

$$B = (X^T X)^{-1} X^T y$$

Zastosowania

- Prognozowanie
- Redukcja błędów
- Linie trendu
- Finanse i ekonomia
- Nauka o środowisku
- Epidemiologia itp.

Linia trendu - przykład

```
x = np.array([1, 3, 5, 8])
y = np.array([ 6, 3, 10, 5 ])
plt.plot(x, y, 'o')
m, b = np.polyfit(x, y, 1)
plt.plot(x, m*x + b)
```

Przykład (W3Schools)

```
import pandas
from sklearn import linear_model
df = pandas.read_csv("cars.csv")
X = df[['Weight', 'Volume']]
y = df['CO2']
regr =
linear_model.LinearRegression()
regr.fit(X, y)
```

Przykład (W3Schools)

Toyota	Aygo	1000	790	99
Mitsubishi	Space Star	1200	1160	95
Skoda	Citigo	1000	929	95
Fiat	500	900	865	90
Mini	Cooper	1500	1140	105
VW	Up!	1000	929	105
Skoda	Fabia	1400	1109	90

itd.

Przykład (W3Schools)

```
import pandas
from sklearn import linear_model
df = pandas.read_csv("cars.csv")
X = df[['Weight', 'Volume']]
y = df['CO2']
regr = linear_model.LinearRegression()
regr.fit(X, y)
predictedCO2 = regr.predict([[2300, 1300]])
print("predictedCO2 [weight=2300kg,
    volume=1300ccm]:")
print(predictedCO2) # [107.2087328]#
przewidziano emisję CO2 auta o wadze 2300 kg
# i objętości 1300 cm3
```

Przykład (W3Schools)

Podsumowując, regresja wielokrotna działa w sposób zbliżony do regresji liniowej, ale z więcej niż jedną niezależną wartością, co oznacza, że próbujemy przewidzieć wartość na podstawie dwóch lub więcej zmiennych.

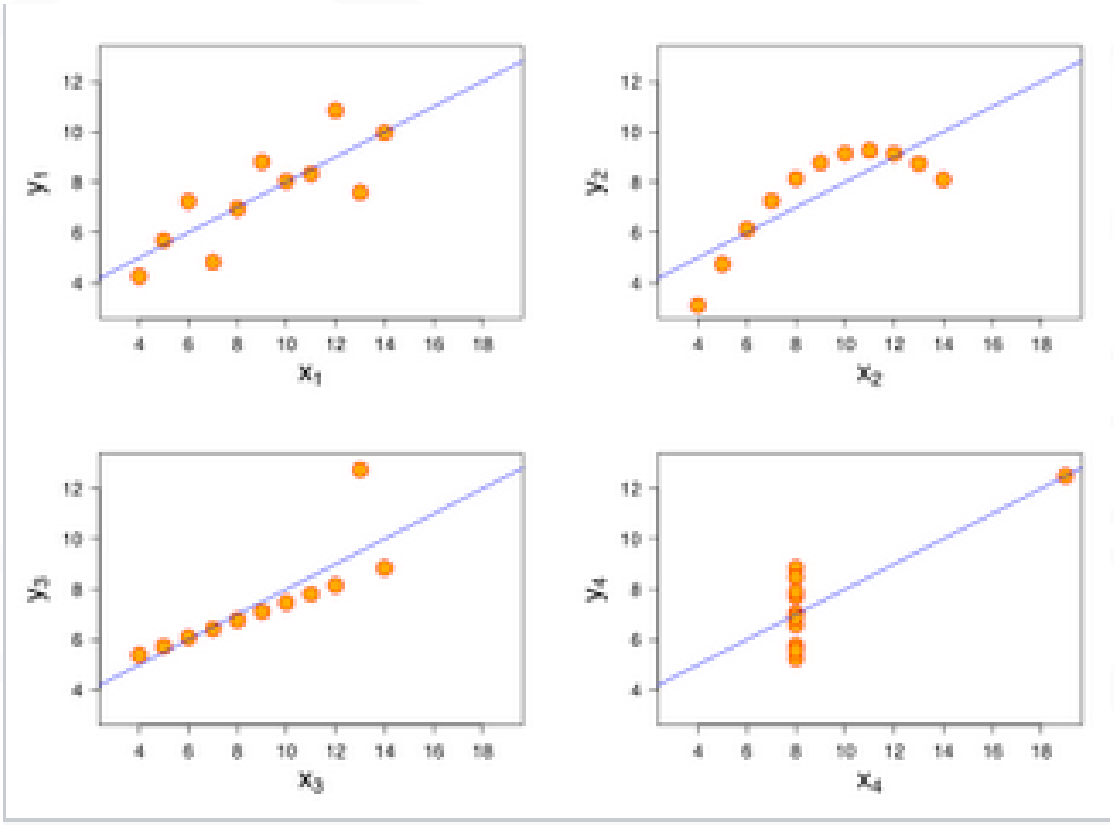
Możemy przewidzieć emisję CO2 przez samochód na podstawie wielkości silnika, ale dzięki regresji wielokrotnej możemy dodać więcej zmiennych, np. wagę samochodu, aby prognoza była dokładniejsza.

Regresja a testy statystyczne

Nazwa zwyczajowa	Równoważny model liniowy	Opis słowny
test *t* Studenta dla jednej próby	$y = \beta_0 + \epsilon$	Czy średnia (lub mediana) obserwacji jest ich dobrym predyktorem?
test Wilcoxona dla jednej próby	$\operatorname{ranga}^+_-(y) = \beta_0 + \epsilon$	
test *t* Studenta dla par obserwacji	$y_2 - y_1 = \beta_0 + \epsilon$	Czy średnia (lub mediana) różnic obserwacji jest ich dobrym predyktorem?
test Wilcoxona dla par obserwacji	$\operatorname{ranga}^+_-(y_2 - y_1) = \beta_0 + \epsilon$	
korelacja *r* Pearsona	$y = \beta_0 + \beta_1 x + \epsilon$	Czy model liniowy jest dobrym predyktorem obserwacji (lub ich rang)?
korelacja Spearmana	$\operatorname{ranga}(y) = \beta_0 + \beta_1 \operatorname{ranga}(x) + \epsilon$	
test *t* Studenta dla dwóch prób	$y = \beta_0 + \beta_1 D + \epsilon$	Czy średnie grup są dobrym predyktorem obserwacji (lub ich rang)?
test Manna-Whitneya	$\operatorname{ranga}^+_-(y) = \beta_0 + \beta_1 D + \epsilon$	
jednoczynnikowa ANOVA	$y = \beta_0 + \beta_1 D_1 + \beta_2 D_2 + \cdots + \beta_n D_n + \epsilon$	
test Kruskala-Wallisa	$\operatorname{ranga}^+_-(y) = \beta_0 + \beta_1 D_1 + \beta_2 D_2 + \cdots + \beta_n D_n + \epsilon$	
jednoczynnikowa ANCOVA	$y = \beta_0 + \beta_1 D_1 + \beta_2 D_2 + \cdots + \beta_n D_n + \beta_x x + \epsilon$	Czy średnie grup oraz ich liniowy model są dobrym predyktorem obserwacji (lub ich rang)?
dwuczynnikowa ANOVA	$y = \beta_0 + \beta_1 D_1 + \beta_2 D_2 + \cdots + \beta_n D_n + \beta_o E_1 + \beta_p E_2 + \cdots + \beta_r E_m + \beta_s D_1 E_1 + \beta_t D_1 E_2 + \cdots + \beta_u D_n E_m + \epsilon$	Czy średnie grup oraz ich iloczynów są dobrym predyktorem obserwacji?

Źródło: wikipedia

Niedoskonałość regresji – kwartet Anscombe'a

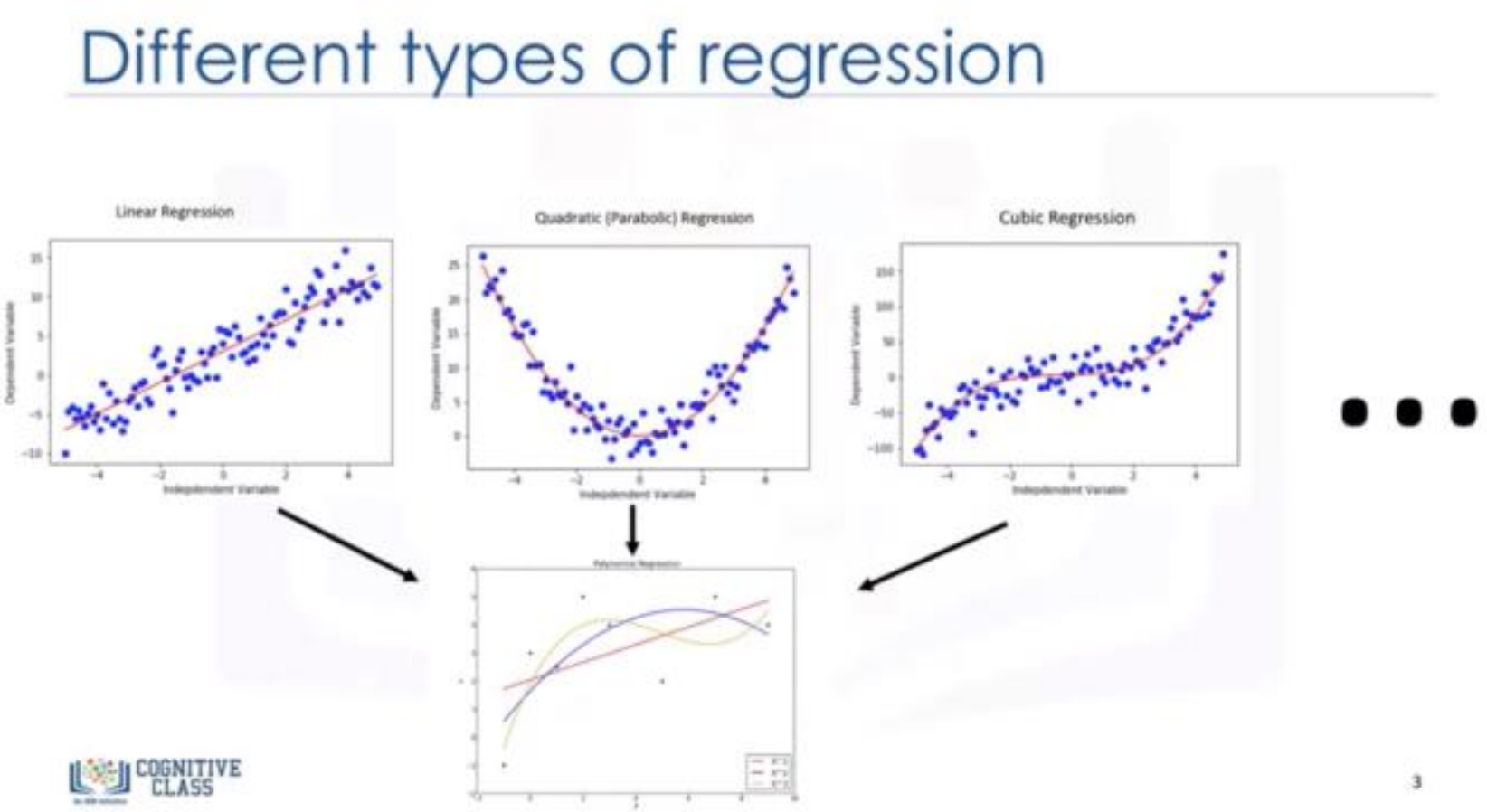


Źródło: wikipedia

Uwaga

Regresja jest problemem interpolacyjnym i nie powinna być (chyba, że jest to uzasadnione) stosowana do problemów ekstrapolacyjnych

Regresja nieliniowa



Źródło: https://medium.com/@toprak.mhmt/non-linear-regression-4de80afca347

Regresja wielomianowa

What is polynomial regression?

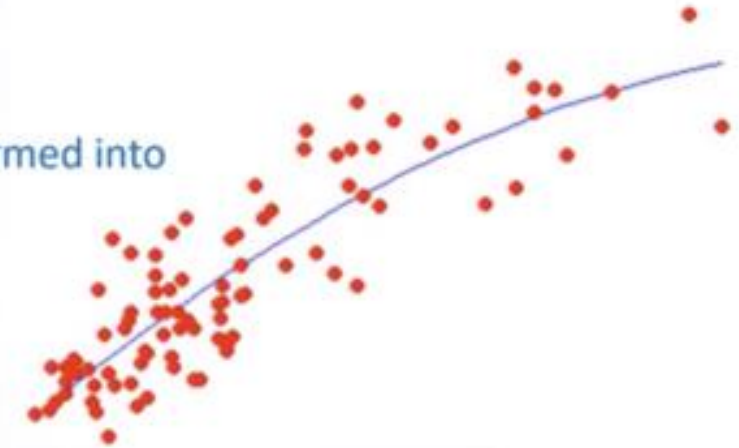
- Some curvy data can be modeled by a polynomial regression
- For example:

$$\hat{y} = \theta_0 + \theta_1 x + \theta_2 x^2 + \theta_3 x^3$$

- A polynomial regression model can be transformed into linear regression model.

$$x_1 = x$$
$$x_2 = x^2$$
$$x_3 = x^3$$

$$\hat{y} = \theta_0 + \theta_1 x_1 + \theta_2 x_2 + \theta_3 x_3$$



COGNITIVE CLASS

4

Źródło: https://medium.com/@toprak.mhmt/non-linear-regression-4de80afca347

Pytania?

Dziękuję za uwagę!

POLITECHNIKA LUBELSKA
WYDZIAŁ ELEKTROTECHNIKI I INFORMATYKI
INFORMATYKA

Zintegrowany Program Rozwoju Politechniki Lubelskiej - część druga

PODSTAWY SZTUCZNEJ INTELIGENCJI W JĘZYKU PYTHON

Maszyny wektorów nośnych

Paweł Karczmarek

Fundusze Europejskie Wiedza Edukacja Rozwój
Rzeczpospolita Polska
Unia Europejska Europejski Fundusz Społeczny

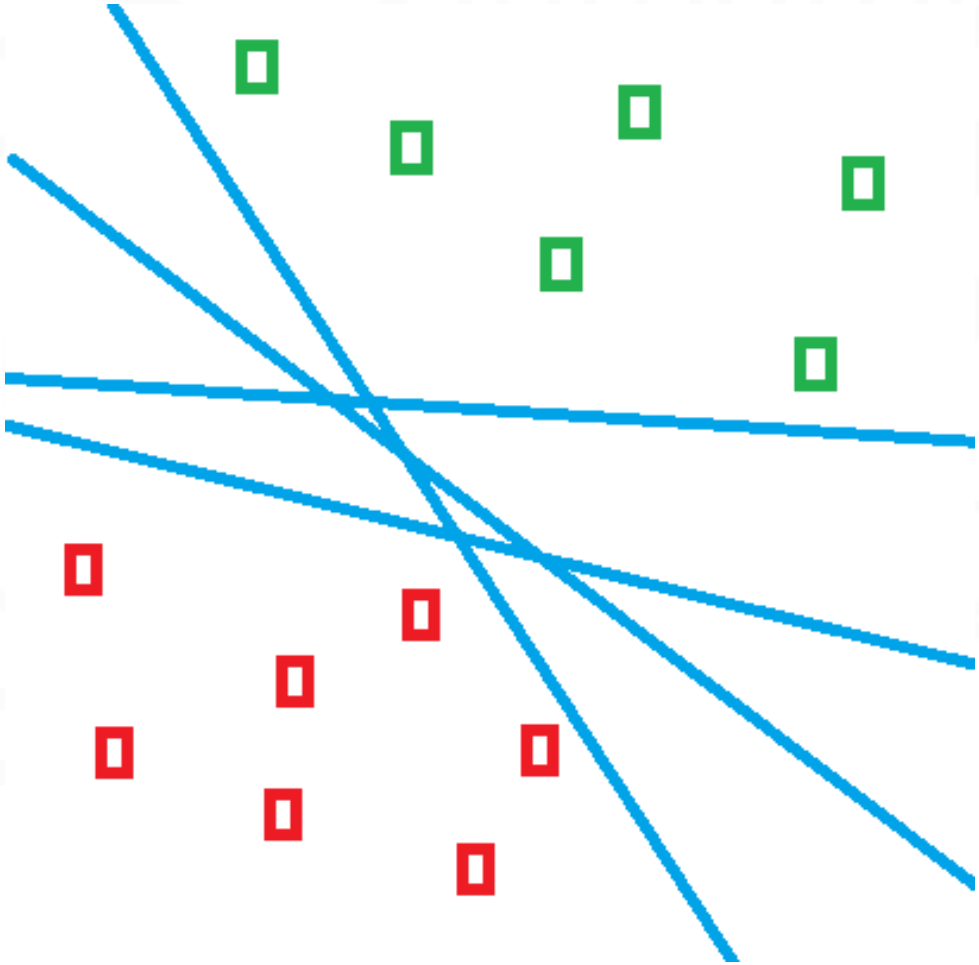
Agenda

- Koncepcja
- Twardy margines
- Miękki margines

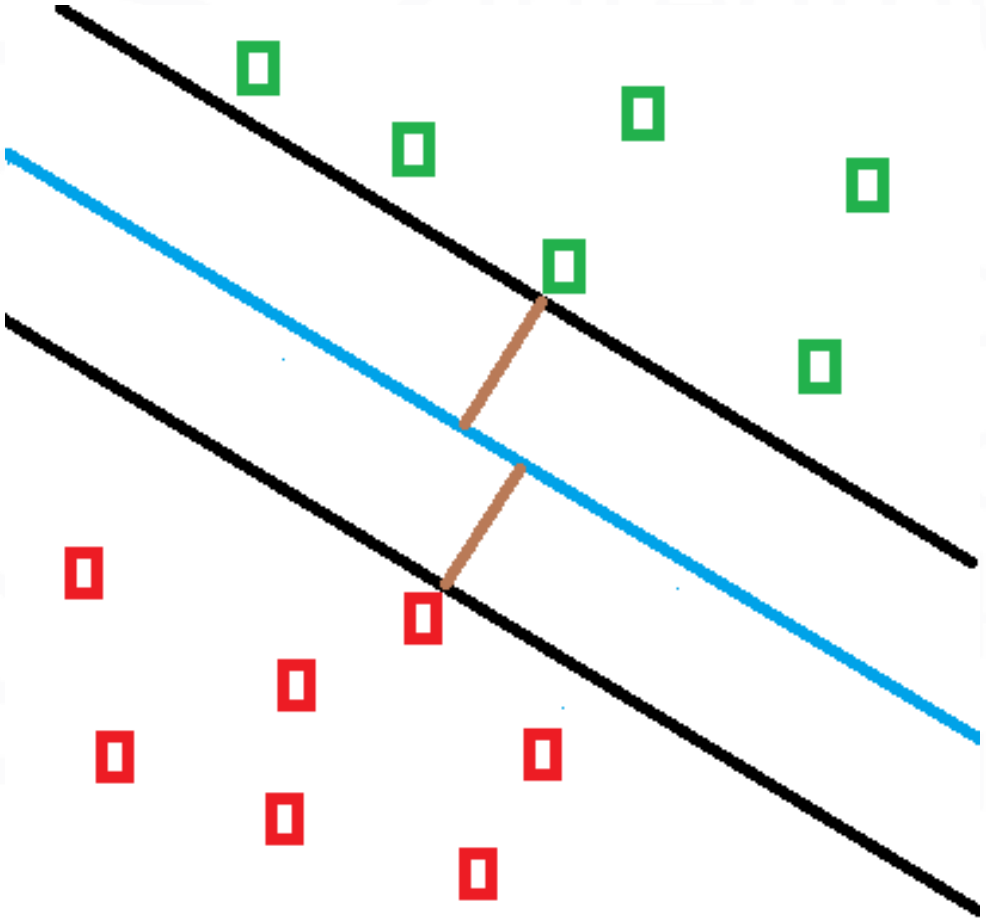
Ogólna idea

- Dane są punkty, z których każdy należy do jednej z dwóch klas
- Cel: Zdecydować, w której klasie będzie znajdować się nowy punkt danych
- Punkt danych jest *p*-wymiarowym wektorem
- Czy możemy oddzielić takie punkty za pomocą (*p*-1)-wymiarowej hiperpłaszczyzny?
- (tj. klasyfikatora liniowego)

Kilka możliwych hiperpłaszczyzn spełniających rozwiązanie



Maksymalizacja marginesu (kanoniczne hiperpłaszczyzny)



Teoria

Dane:

- Zbiór punktów $(x_i, y_i), x_i \in \mathbb{R}^p, y_i \in \{-1,1\}$ – dwie klasy

Hiperpłaszczyzna:

- Zbiór punktów $\boldsymbol{x}$ spełniających $\boldsymbol{w} \cdot \boldsymbol{x} - b = 0$ ($\boldsymbol{w}$ jest wektorem normalnym do hiperpłaszczyzny)
- Wartość $\frac{b}{\|\boldsymbol{w}\|}$ określa przesunięcie hiperpłaszczyzny od początku wzdłuż wektora normalnego

Klasyfikacja liniowa

- Przypadek danych liniowo separowalnych
- Wybierz dwie równoległe hiperpłaszczyzny, które oddzielają dwie klasy, tak aby odległość między nimi była jak największa
- Region przez nie ograniczony nazywa się marginesem
- Hiperpłaszczyzna maksymalnego marginesu to hiperpłaszczyzna leżąca w połowie odległości między nimi

Klasyfikacja liniowa (2)

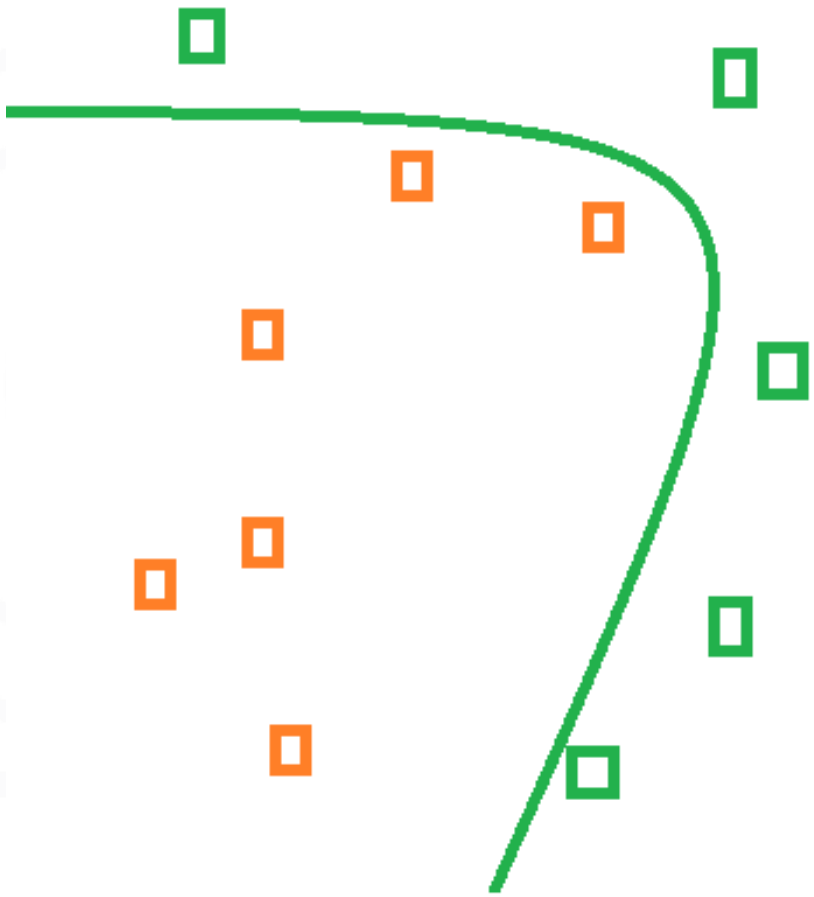
- Dwie hiperpłaszczyzny mogą być zdefiniowane jako $\boldsymbol{w} \cdot \boldsymbol{x}_i - b = 1$ oraz $\boldsymbol{w} \cdot \boldsymbol{x}_i - b = -1$
  (pierwsza i druga klasa)
- Zapobiegamy wypadaniu punktów na margines: $\boldsymbol{w} \cdot \boldsymbol{x}_i - b \geq 1$ oraz $\boldsymbol{w} \cdot \boldsymbol{x}_i - b \leq -1$
- Dlatego: $\boldsymbol{y}_i(\boldsymbol{w} \cdot \boldsymbol{x}_i - b) \geq 1$ dla wszystkich i *
- Zagadnienie optymalizacyjne: Zminimalizować $\|\boldsymbol{w}\|$ przy warunku *
- Rozwiązenie: $\boldsymbol{w}$ i b, które spełniają problem
- Klasyfikator: $\boldsymbol{x} \mapsto \operatorname{sgn}(\boldsymbol{w} \cdot \boldsymbol{x} - b)$

Klasyfikacja liniowa (3)

- Problem nazywa się twardą marżą (marginesem)
- Wektory określające marginesy nazywane są wektorami nośnymi
- Aby zastosować SVM do problemów, w których dane nie są liniowo oddzielone, używa się funkcji straty, tzw. hinge loss:

$$l(y) = \max(0,1 - \mathrm{ty}), \mathrm{t} = -1 \text{ lub } \mathrm{t} = 1$$

Idea klasyfikacji nieliniowej



Miękki margines/miękka marża

- Pozwolić SVM popełnić kilka błędów
- Utrzymywać margines jak najszerzej tak, aby pozostałe punkty były właściwie sklasyfikowane
- Formalnie:
- Zminimizować $\frac{\|\boldsymbol{w}\|^2}{2} + C$ (C–liczba pomyłek)
- C – hiperparametr decydujący o kompromisie

Miękki margines/miękka marża

- Dla każdego punktu danych $\boldsymbol{x}_i$ wprowadzamy parametr „luzu” $\boldsymbol{\xi}_i$.
- Jego wartość to dystans $\boldsymbol{x}_i$ od odpowiedniego marginesu klasy, jeżeli $\boldsymbol{x}_i$ jest po niewłaściwej stronie marginesu, w przeciwnym wypadku ma on wartość zerową.
- Punkty, które są daleko od marginesu po niewłaściwej stronie, mogą otrzymać większą „karę”
- Zatem każdy punkt danych $\boldsymbol{x}_i$ musi spełniać warunek $\boldsymbol{y}_i(\boldsymbol{w} \cdot \boldsymbol{x}_i - b) \geq 1 - \boldsymbol{\xi}_i$

Ostateczny problem

- $L = \frac{\|\boldsymbol{w}\|^2}{2} + C\sum_i \boldsymbol{\xi}_i + \sum_i \lambda_i \left(\boldsymbol{y}_i(\boldsymbol{w}\cdot\boldsymbol{x}_i + b) - 1 - \boldsymbol{\xi}_i\right)$
- Tutaj wykorzystano mnożniki Lagrange'a
- Aby go rozwiązać, można użyć triku z jądrem (tj. użyć dowolnej funkcji jądra zamiast iloczynu skalarnego)

Uwagi

Literatura uzupełniająca (tematy poszukiwań):

- Problem pierwotny
- Problem dualny
- Funkcje jądra
- Mnożniki Lagrange'a

Podstawy teorii: V. Vapnik

Biblioteki Pythona:

- scikit-learn (import svm)
- SVM^{struct} Python
- LIBSVM

Przykład

```
>>> from sklearn import svm
>>> X = [[0, 0], [1, 1]]
>>> y = [0, 1]
>>> clf = svm.SVC()
>>> clf.fit(X, y)
SVC()

#Predykcja:

>>> clf.predict([[2., 2.]])
array([1])
```

Zalety SVM

- Skuteczny w przestrzeniach wielowymiarowych.
- Skuteczny w przypadkach, gdy liczba wymiarów jest większa niż liczba próbek.
- Wykorzystuje podzbiór punktów treningowych w funkcji decyzyjnej (zwanych wektorami wsparcia), dzięki czemu jest również wydajny pamięciowo.
- Wszechstronny: dla funkcji decyzyjnej można określić różne funkcje jądra.
- Dostępne są wspólne jądra, ale możliwe jest również określenie niestandardowych jąder.

Wady SVM

- Jeśli liczba funkcji jest znacznie większa niż liczba próbek, należy unikać nadmiernego dopasowania przy wyborze funkcji jądra
- SVM nie dostarcza bezpośrednio szacunków prawdopodobieństwa, są one obliczane przy użyciu kosztownej walidacji krzyżowej

Większa liczba klas

- SVC i NuSVC implementują podejście „one vs one” w klasyfikacji wieloklasowej.
- W sumie konstruuje się *n_klas* * (*n_klas* - 1) / 2 klasyfikatorów i każdy z nich uczy dane z dwóch klas
- Aby zapewnić spójny interfejs z innymi klasyfikatorami, opcja decision_function_shape umożliwia monotoniczną transformację wyników klasyfikatorów „one vs one” na funkcję decyzyjną kształtu „one vs rest” (n_prób, n_klas).

Przykład

```
>>> X = [[0], [1], [2], [3]]
>>> Y = [0, 1, 2, 3]
>>> clf =
svm.SVC(decision_function_shape='ovo')
>>> clf.fit(X, Y)
SVC(decision_function_shape='ovo')
>>> dec = clf.decision_function([[1]])
>>> dec.shape[1] # 4 classes: 4*3/2 = 6
6
>>> clf.decision_function_shape = "ovr"
>>> dec = clf.decision_function([[1]])
>>> dec.shape[1] # 4 classes
4
```

Dodatkowe uwagi

- Uwaga na problemy niezbalansowane (ważenie klas)
- SVM może być również stosowany w problemie regresji
- Dokumentacja scikit-learn:

https://scikit-learn.org/stable/modules/svm.html

Pytania?

Dziękuję za uwagę!

POLITECHNIKA LUBELSKA
WYDZIAŁ ELEKTROTECHNIKI I INFORMATYKI
INFORMATYKA

Zintegrowany Program Rozwoju Politechniki Lubelskiej - część druga

PODSTAWY SZTUCZNEJ INTELIGENCJI W JĘZYKU PYTHON

Analiza głównych składowych i liniowa analiza dyskryminacyjna

Paweł Karczmarek

Fundusze Europejskie Wiedza Edukacja Rozwój
Rzeczpospolita Polska
Unia Europejska Europejski Fundusz Społeczny

Agenda

- Koncepcja
- PCA (Principal Component Analysis)
- LDA (Linear Discriminant Analysis)

Wielowymiarowość

- Przekleństwo wymiarowości
  - Odnosi się do sytuacji, gdy poprawna klasyfikacja obiektów, wykorzystując pełen zbiór danych, jest niemal niemożliwa, a wielość charakterystyk w wektorze skutkuje wzrostem liczby parametrów, co skutkuje wzrostem złożoność klasyfikatora. Rośnie również ryzyko przeuczenia i tym samym spadku zdolności uogólniających klasyfikatora.

Wielowymiarowość

Dla różnych typów rozkładów losowych zbiorów danych różnica pomiędzy najmniejszą, a największą odległością pomiędzy zbiorami staje się nieznaczącą w porównaniu do najmniejszej odległości, gdy zwiększa się miara (innymi słowy: najmniejsze i największe odległości różnią się tylko stosunkowo niewielkim wielkościami), a zatem wyniki funkcji odległości i algorytmów na nich opartych przestają być użyteczne dla rozkładów w przestrzeniach wielowymiarowych.

Redukcja wymiaru

- Selekcja cech - ograniczenie zbioru zmiennych wedle jednej lub kilku reguł
  - odrzucanie cech nadmiernie skorelowanych ze sobą
  - odrzucanie cech nieistotnych statystycznie
  - odrzucanie cech, które nie poprawiają wyników modelu
  - odrzucanie cech według wiedzy eksperckiej
  - istotność cech – wagi
- Ekstrakcja cech - tworzenie cech pochodnych z początkowego zestawu danych celem uzyskania mniej obszernego zbioru zmiennych, który jak najlepiej odzwierciedlać będzie zależności w danych

Ogólna koncepcja PCA

- Statystyczna metoda analizy czynnikowej
- Zbiór danych składający się z *N* obserwacji, każda z nich zawiera *K* zmiennych, może być interpretowany jako chmura *N* punktów w *K*-wymiarowej przestrzeni
- Celem jest obrócenie układu współrzędnych w taki sposób, aby zmaksymalizować najpierw wariancję pierwszej współrzędnej, następnie wariancję drugiej współrzędnej itd.

Ogólna koncepcja PCA (2)

- Przekształcone w ten sposób wartości współrzędnych nazywane są składowymi głównymi
- Powstaje nowa przestrzeń obserwacji, w której największą zmienność wyjaśniają czynniki początkowe
- PCA można wykorzystać do zmniejszenia rozmiaru zbioru danych poprzez odrzucenie ostatnich czynników lub do merytorycznej interpretacji czynników
- Obliczanie: przy użyciu macierzy kowariancji lub macierzy korelacji

Algorytm (obliczanie z zastosowaniem macierzy kowariancji)

- Cel:
  Przekształcić zadany zbiór danych ***X*** wymiaru *p* w alternatywny zbiór danych ***Y*** o mniejszym wymiarze *L*
- *Dane są:*
  Zbiór *n* wektorów danych $\boldsymbol{x}_1, \ldots, \boldsymbol{x}_n$ reprezentujących pojedyncze pogrupowane obserwacje *p* zmiennych

Algorytm (organizacja danych i uśrednienie)

- Zapisz $\boldsymbol{x}_1, \ldots, \boldsymbol{x}_n$ jako wektory wierszami, z których każdy ma *p* kolumn
- Umieść te wektory w pojedynczej macierzy ***X*** wymiaru $n \times p$
- Znajdź średnią po każdej kolumnie *j*=1, …, *p* i umiejść ją w wektorze średnich ***u*** wymiaru $p \times 1$

$$u_j = \frac{1}{n}\sum_{i=1}^{n} X_{ij}$$

Algorytm (odchylenia od średniej)

- Odejmij średnią $\boldsymbol{u}^T$ od każdego wiersza macierzy $\boldsymbol{X}$
- Zmagazynuj wyniki w macierzy $\boldsymbol{B}$ wymiaru $n \times p$

$$\boldsymbol{B} = \boldsymbol{X} - \boldsymbol{h}\boldsymbol{u}^T$$

gdzie $\boldsymbol{h}$ jest *n*-wymiarowym wektorem samych jedynek:

$$h_i = 1$$

dla *i=1, …, n*

Algorytm (macierz kowariancji)

- Znajdź macierz kowariancji $\boldsymbol{C}$ wymiaru $p \times p$ na podstawie macierzy $\boldsymbol{B}$:

$$\boldsymbol{C} = \frac{1}{n-1} \boldsymbol{B}^* \boldsymbol{B}$$

- * oznacza sprzężony operator transpozycji (w przypadku niezespolonym jest to zwykła transpozycja)
- Wartość *n*-1 jest stosowana w miejscu *n*, aby obliczyć kowariancję z poprawką Bessela

Algorytm (wektory własne i wartości własne)

- Wyznacz macierz $\boldsymbol{V}$ wektorów własnych, która diagonalizuje macierz kowariancji C, tj.

$$\boldsymbol{V}^{-1}\boldsymbol{C}\boldsymbol{V} = \boldsymbol{D}$$

- Macierz $\boldsymbol{D}$ przyjmie postać macierzy diagonalnej wymiaru $p \times p$, gdzie

$$D_{kl} = \lambda_k \text{ dla } k = l$$

jest k-tą wartością własną macierzy $\boldsymbol{C}$ oraz

$$D_{kl} = 0 \text{ dla } k \neq l$$

Algorytm (sortowanie i energia skumulowana)

- Posortuj kolumny ***V*** oraz ***D*** malejąco względem wartości własnych
- Te wartości własne reprezentują rozkład energii źródła danych względem każdego wektora własnego, natomiast wektory własne tworzą bazę dla danych
- Skumulowana wartość energii dla *j*-tego wektora własnego jest sumą wartości energii po wartościach własnych od pierwszej do *j*-tej:

$$g_j = \sum_{k=1}^{j} D_{kk} \text{ for } j = 1, \ldots, p$$

Algorytm (wybór podzbioru wektorów bazowych)

- Zapisz pierwsze *L* kolumn macierzy ***V*** jako macierz ***W*** wymiaru $p \times L$**:**

$$W_{kl} = V_{kl} \text{ dla } k = 1, \dots, p, l = 1, \dots, L$$

gdzie $L \leq p$

- Na podstawie wektora **g** wybierz odpowiednią wartość *L* w taki sposób, aby była jak najmniejsza, osiągając rozsądnie wysoką wartość g, np. dla t=90%, 95%, 99% itp.:

$$\frac{g_L}{g_p} \geq t$$

Algorytm (rzutowanie)

- Zrzutuj dane na nową bazę. Zrzutowane wektory są kolumnami macierzy

$$\boldsymbol{T} = \boldsymbol{B} \cdot \boldsymbol{W}$$

- Wiesze $\boldsymbol{T}$ przedstawiają przekształcenie Karhunena–Loève'a wektorów danych

LDA

- Uogólnienie liniowego dyskryminatora Fishera
- Technika redukcji wymiarowości oparta na minimalizacji macierzy rozproszeń wewnątrzklasowych i maksymalizacji macierzy rozproszenia międzyklasowego
- PCA można rozumieć jako algorytm nienadzorowany (ignoruje etykiety klas, jego celem jest znalezienie głównych składników, które maksymalizują wariancję w zbiorze danych)
- LDA jest nadzorowany. Oblicza liniowe dyskryminatory reprezentujące osie, które maksymalizują separację między wieloma klasami

LDA (Cel i pierwsze przekształcenia)

- Dane:
- *d*-wymiarowe wektory pewnego zbioru danych $\boldsymbol{X}$
- Cel:
- Zredukować wymiar zbioru *d*-wymiarowego poprzez zrzutowanie go na *k*-wymiarową podprzestrzeń, *k*<*d*
- Pierwszy krok: Oblicz *d*-wymiarowe wektory średnie $\boldsymbol{m}_k$ dla różnych klas D_k:

$$\boldsymbol{m}_k = \frac{1}{n_k} \sum_{\boldsymbol{x} \in D_k} \boldsymbol{x}$$

Macierz rozproszeń wewnątrzklasowych

$$S_W = \sum_{k=1}^{c} S_k$$

gdzie S_k jest macierzą dla każdej klasy:

$$S_k = \sum_{\boldsymbol{x} \in D_k} (\boldsymbol{x} - \boldsymbol{m}_i)(x - \boldsymbol{m}_i)^T$$

i c jest liczbą klas

Macierz rozproszenia międzyklasowego i uogólnione zagadnienie znalezienia wartości własnych

$$S_A = \sum_{i=1}^{c} C_i(\boldsymbol{m}_i - \boldsymbol{m})(\boldsymbol{m}_i - \boldsymbol{m})^T$$

m – wektor średni względem całego zbioru

C_i - liczba elementów *i*-tej klasy

- Zagadnienie uogólnione:

$$S_w{}^{-1} S_A \boldsymbol{v} = \lambda \boldsymbol{v}$$

Rankingowanie wartości własnych i wektorów własnych

- Posortuj wartości własne i odpowiadające im wektory własne w porządku malejącym (od najbardziej do najmniej „informatywnych”)
- Wybierz *r* wektorów własnych (podobnie, jak w przypadku PCA)
- Ostateczne przekształcenie to

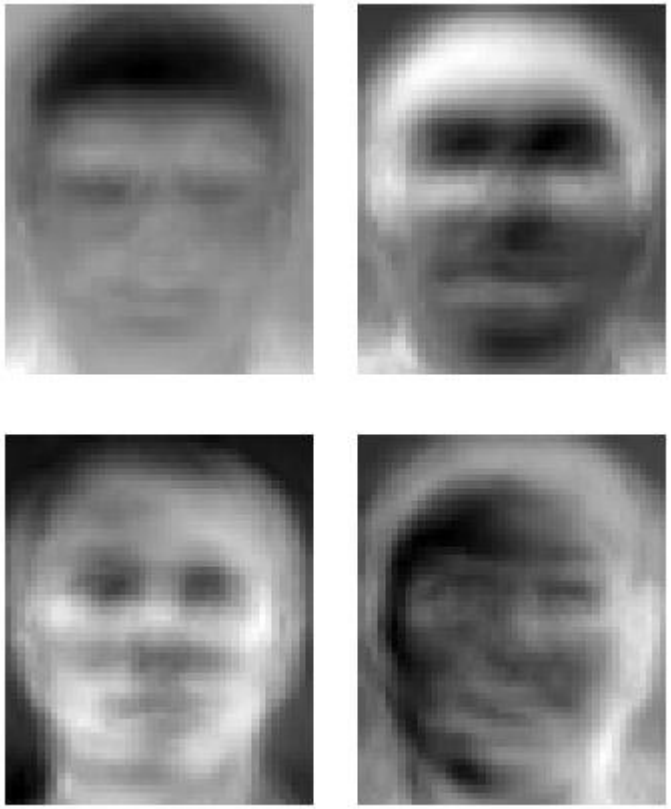
$$Y = XW$$

gdzie *X* jest macierzą wymiaru $n \times d$ próbek, *W* – macierzą wymiaru $d \times r$ wektorów własnych

Zastosowania

- Finanse (PCA)
- Neurobiologia (PCA)
- **Eigenfaces (Turk & Pentland, 1991)**
- **Fisherfaces (Belhumeur et al., 1997)**
- Biomedycyna (LDA)
- Nauki o ziemi (LDA)
- Przewidywanie bankructwa (LDA)
- Marketing (LDA)

Eigenfaces (twarze własne)



Źródło:
https://upload.wikimedia.org/wikipedia/commons/6/67/Eigenfaces.png
ATT&T Laboratories Cambridge

Przykładowe zastosowania bibliotek Pythona

- PCA
  https://towardsdatascience.com/pca-using-python-scikit-learn-e653f8989e60
- LDA
  https://machinelearningmastery.com/linear-discriminant-analysis-with-python/

Pytania?

Dziękuję za uwagę!

POLITECHNIKA LUBELSKA
WYDZIAŁ ELEKTROTECHNIKI I INFORMATYKI
INFORMATYKA

Zintegrowany Program Rozwoju Politechniki Lubelskiej - część druga

PODSTAWY SZTUCZNEJ INTELIGENCJI W JĘZYKU PYTHON

Drzewa decyzyjne

Paweł Karczmarek

Fundusze Europejskie Wiedza Edukacja Rozwój
Rzeczpospolita Polska
Unia Europejska Europejski Fundusz Społeczny

Agenda

- Idea
- Uczenie
- Lasy losowe

Drzewo decyzyjne

- Narzędzie wspomagające podejmowanie decyzji
- Wykorzystuje podobny do drzewa model decyzji i ich możliwych konsekwencji, w tym wyników zdarzeń losowych, kosztów zasobów i użyteczności
- Służy do opisu algorytmu, który zawiera tylko warunkowe instrukcje sterujące
- Zastosowania: Klasyfikacja lub regresja

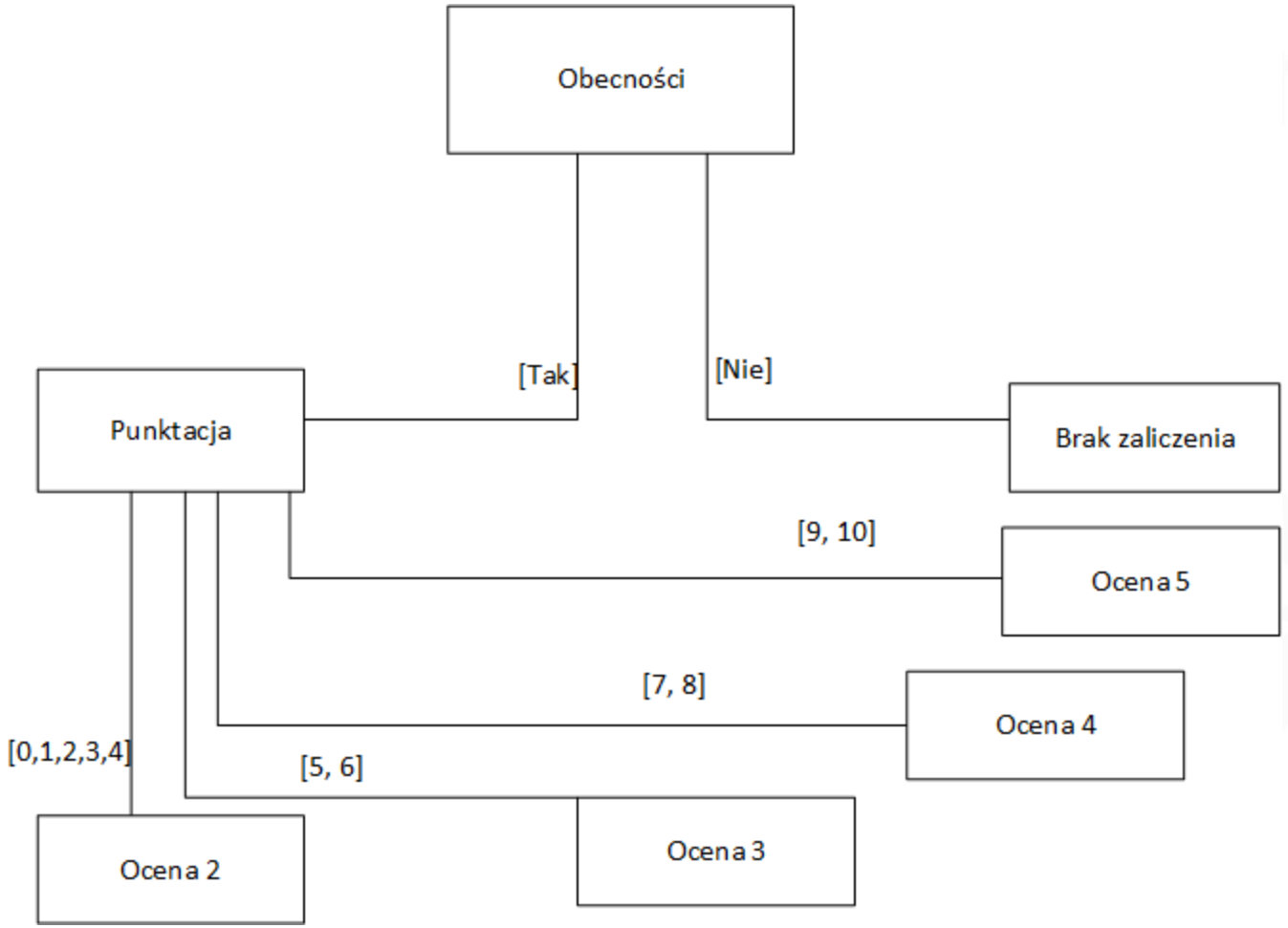
Drzewo decyzyjne (2)

- Struktura drzewa jest podobna do schematu blokowego
- Węzeł wewnętrzny reprezentuje test na atrybucie
- Gałąź reprezentuje wynik testu
- Węzeł liścia reprezentuje etykietę klasy (decyzja podjęta po obliczeniu)
- Ścieżki od korzenia do liścia reprezentują reguły klasyfikacji

Węzły

- Węzły decyzyjne
  – zwykle przedstawiane jako kwadraty
- Węzły szansy
  – zwykle przedstawiane jako okręgi
- Węzły końcowe
  – zwykle przedstawiane jako trójkąty

Przykład


Obecności
[Tak]
[Nie]
Punktacja
Brak zaliczenia
[9, 10]
Ocena 5
[7, 8]
Ocena 4
[0,1,2,3,4]
[5, 6]
Ocena 2
Ocena 3

Rodzaje drzew decyzyjnych

- Drzewo klasyfikacji:
  Przewidywany wynik to klasa (dyskretna), do której należą dane
- Drzewo regresji:
  Przewidywany wynik: liczba rzeczywista

Konstruowanie drzew decyzyjnych

- W przypadku prostych decyzji – ręcznie
- Zazwyczaj:
  Korzysta się z algorytmów
- Działają metodą top-down na podstawie wyboru w każdym kroku zmiennej, która najlepiej dzieli zbiór elementów
- Wykorzystywana jest miara (nie)jednorodności zmiennej w podzbiorach (indeks Giniego, redukcja wariancji)

Uczenie drzewa – algorytm CART

- Classification And Regression Tree (CART)
- Stosowany w Scikit-Learn
- Idea:
  Najpierw dzieli zbiór uczący na dwa podzbiory korzystając z cechy *k* i progu t_k
- Szuka par (k, t_k) zwracających „najczystsze” podzbiory (ważone ich rozmiarem)

Uczenie drzewa – algorytm CART

Funkcja kosztu, którą należy zminimalizować to

$$F(k, t_k) = \frac{m_L}{m} G_L + \frac{m_R}{m} G_R$$

G_L - miara niejednorodności lewego podzbioru
G_R - miara niejednorodności prawego podzbioru
m_L - liczba prób w lewym podzbiorze
m_R - liczba prób w prawym podzbiorze

Miara niejednorodności

Zwana indeksem (wskaźnikiem) Giniego:

$$G = 1 - \sum_{i=1}^{n} {p_i}^2$$

p_i - stosunek liczby instancji *i*-tej klasy do uczących instancji w danym węźle.

CART (3)

- Po pomyślnym podzieleniu zestawu uczącego na dwie części dzieli podzbiory rekursywnie przy użyciu tej samej logiki itd.
- Przestaje osiągając maksymalną głębokość lub jeśli nie może znaleźć podziału, który zmniejszy niejednorodność
- Algorytm jest zachłanny
- Znalezienie optymalnego drzewa to problem NP-zupełny

Alternatywy

- Miara entropii zamiast wskaźnika Giniego
- Algorytmy:
  ID3
  C4.5
  CHAID
  MARS
- Rozmyte drzewa decyzyjne (fuzzy decision trees)

Lasy losowe

- Metoda zespołowego uczenia się, która działa poprzez konstruowanie wielu drzew decyzyjnych w czasie uczenia i wyznaczanie klasy, która jest **modą** klas (klasyfikacja) lub **średnią** (regresja) poszczególnych drzew
- Losowe lasy decyzyjne naprawiają tendencję drzew decyzyjnych do nadmiernego dopasowania do zestawu treningowego

Lasy losowe

Ważna cecha:

- Wprowadzają dodatkową losowość podczas tworzenia drzew zamiast poszukiwania najlepszej funkcji podczas dzielenia węzła
- Wyszukują najlepszą funkcję wśród losowego podzbioru funkcji
- W efekcie uzyskujemy większą różnorodność drzew

Przykład 1

```
>>> from sklearn import tree
>>> X = [[0, 0], [1, 1]]
>>> Y = [0, 1]
>>> clf =
tree.DecisionTreeClassifier()
>>> clf = clf.fit(X, Y)

>>> clf.predict([[2., 2.]])
array([1])
```

Przykład 2

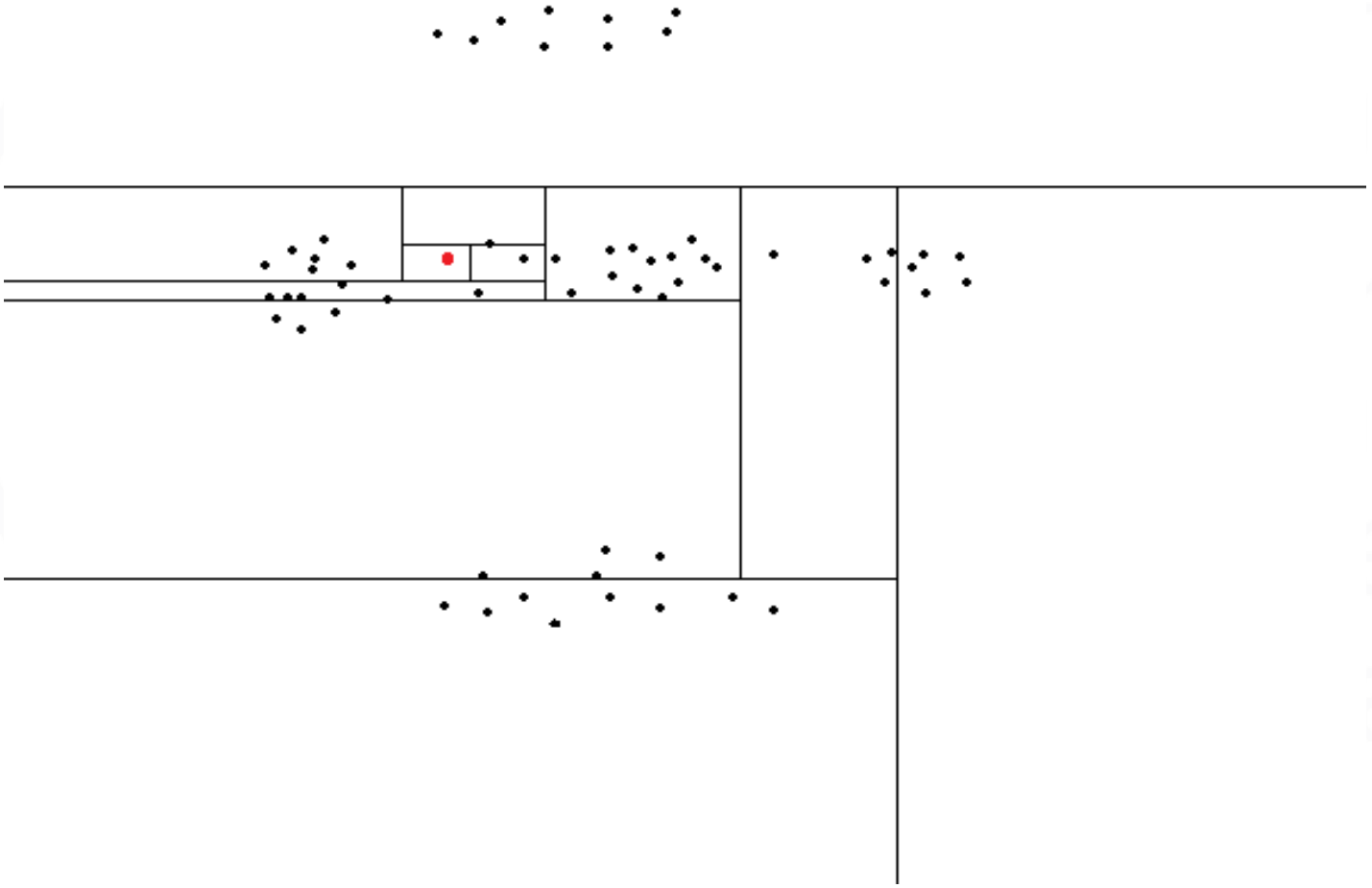
```
>>> from sklearn.datasets import
load_iris
>>> from sklearn import tree
>>> iris = load_iris()
>>> X, y = iris.data, iris.target
>>> clf =
tree.DecisionTreeClassifier()
>>> clf = clf.fit(X, y)

#dokumentacja scikit-learn
```

Wykrywanie anomalii w bazach danych

- Isolation Forest (Liu et al. 2008, 2012)
- Metoda oparta na drzewach poszukiwań binarnych (zwykle 100)
- Drzewa poszukiwań binarnych: 128/256 liści (2 do potęgi głębokości)
- Dla każdego rekordu w bazie oblicza się wartość anomalii względem drzewa i sumuje (ew. średnia)

Las izolacji (Isolation Forest) - koncepcja



Miara anomalii

$$s(x) = 2^{\frac{c(R)}{E(t,M,k)}}$$

gdzie

$$c(R) = 2H(R-1) - 2(R-1)/R$$

jest średnią długością nieudanego procesu przeszukania BST,

$$H(R-1) = \ln(R-1) + 0.5772156649$$

R jest liczbą rekordów w bazie

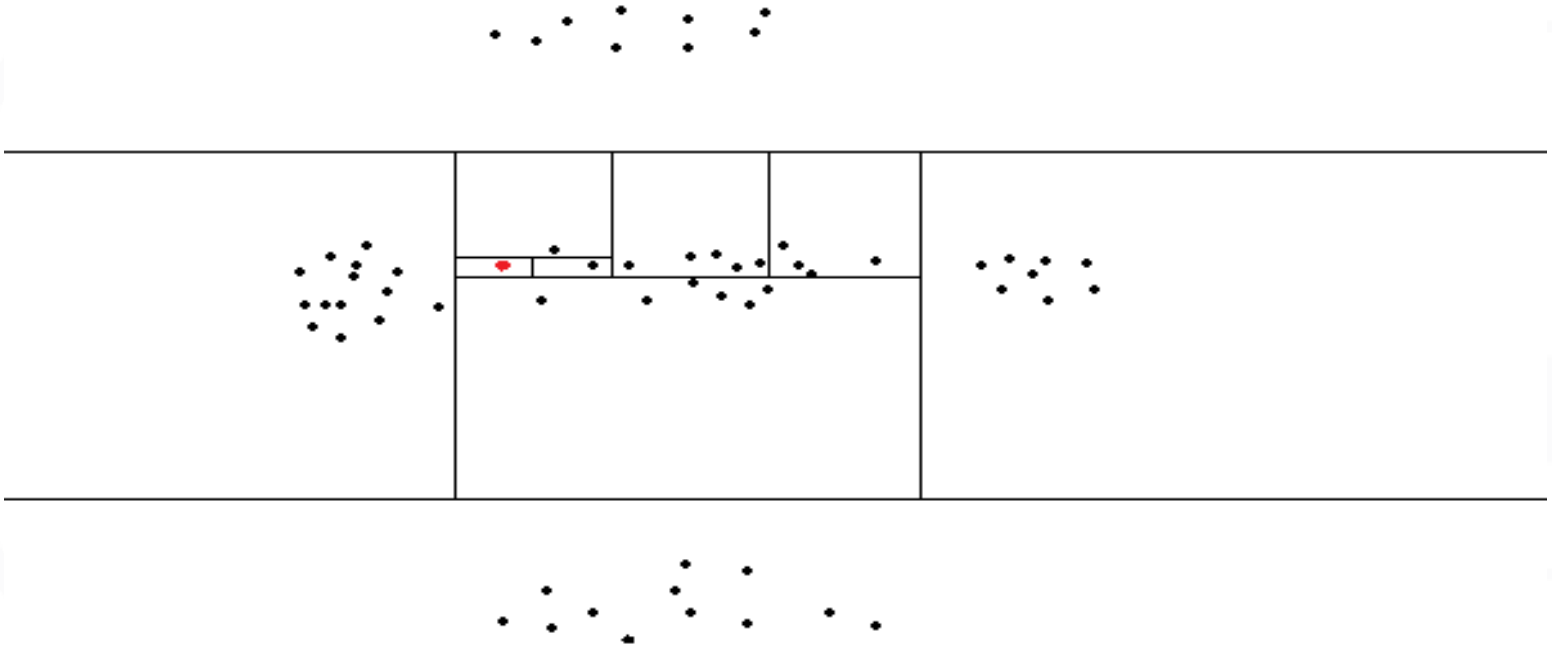
Miara anomalii (2)

$$E(t,M,k) = \frac{1}{t}\sum_{i=1}^{t} \begin{cases} \sum_{j=1}^{M} 1 \text{ dla } k = 1 \\ \sum_{j=1}^{M} 1 + c(k) \text{ w poz. przyp.} \end{cases}$$

t jest liczbą drzew, M jest liczbą wszystkich podziałów binarnych podczas przeszukiwania drzewa, k jest liczbą elementów na ostatnim węźle.

Ogólnie, jeśli miara jest bliska 1, rekord jest prawdopodobnie anomalią. Jeżeli ta wartość jest mniejsza od 0.5, rekord można uznać za „normalny”.

K-Means-Based IF (2020) - koncepcja



Miara anomalii

W każdym podziale

$$s = 1 - d(x, c_q)/d(c_l, c_q)$$

gdzie c_q jest środkiem klastra, c_l jest granicą klastra. *s* należy rozumieć jako przynależność *x* do klastra (liczba klastrów – reguła łokcia). Ostateczna miara dla każdego rekordu bazy to suma wszystkich przynależności w podziałach (w każdym *j*-tym podziale, $j = 1, 2, \dots, M$) podzielona przez liczbę drzew

$$a(x) = 1 - \frac{1}{t}\sum_{i=1}^{t}\sum_{j=1}^{M} s_j(x)$$

Pytania?

Dziękuję za uwagę!

POLITECHNIKA LUBELSKA
WYDZIAŁ ELEKTROTECHNIKI I INFORMATYKI
INFORMATYKA

Zintegrowany Program Rozwoju Politechniki Lubelskiej - część druga

PODSTAWY SZTUCZNEJ INTELIGENCJI W JĘZYKU PYTHON

Wprowadzenie do sieci neuronowych

Paweł Karczmarek

Fundusze Europejskie Wiedza Edukacja Rozwój
Rzeczpospolita Polska
Unia Europejska Europejski Fundusz Społeczny

Agenda

- Koncepcja
- Podstawowe pojęcia
- Przykłady

Neuron mózgowy

- Typowy neuron jest zbudowany z ciała komórki (perikarion) oraz odchodzących od niego wypustek: aksonu i dendrytów.
- Neurony kontaktują się między sobą poprzez synapsy, tworząc sieci neuronowe.
- Informacje od innych neuronów są odbierane przez synapsy położone na dendrytach, przewodzone wzdłuż neuronu i przekazywane dalej do synaps na zakończeniach aksonu.

Neuron mózgowy (2)

- Przewodzenie informacji w postaci sygnału elektrycznego jest możliwe dzięki temu, że wszystkie neurony są elektrycznie pobudliwe, czyli zdolne do generowania i przewodzenia potencjałów elektrycznych.
- Niepobudzony neuron utrzymuje potencjał spoczynkowy (będący różnicą między potencjałem elektrycznym wnętrza neuronu a zewnętrznej powierzchni błony) dzięki działaniu leżących w błonie pomp jonowych, które przenoszą określone jony przez błonę i generują różnicę w stężeniu tych jonów po obu stronach błony.

Neuron mózgowy (3)

- Pod wpływem dostatecznie silnego bodźca dochodzi do zmian w przepuszczalności określonych jonów przez błonę, co prowadzi do powstania potencjału czynnościowego – sygnału elektrycznego, który rozprzestrzenia się wzdłuż aksonu do synaps znajdujących się w zakończeniach aksonu.

Sztuczna sieć neuronowa (artificial neural network, ANN)

- System obliczeniowy luźno inspirowany biologicznymi sieciami neuronowymi, które tworzą mózgi zwierząt
- Polega na połączeniu zbioru połączonych węzłów zwanych sztucznymi neuronami
- Każde połączenie może przesyłać sygnał do innych neuronów
- Neuron, który odbiera sygnał, przetwarza go i może sygnalizować dalej do podłączonych do niego neuronów

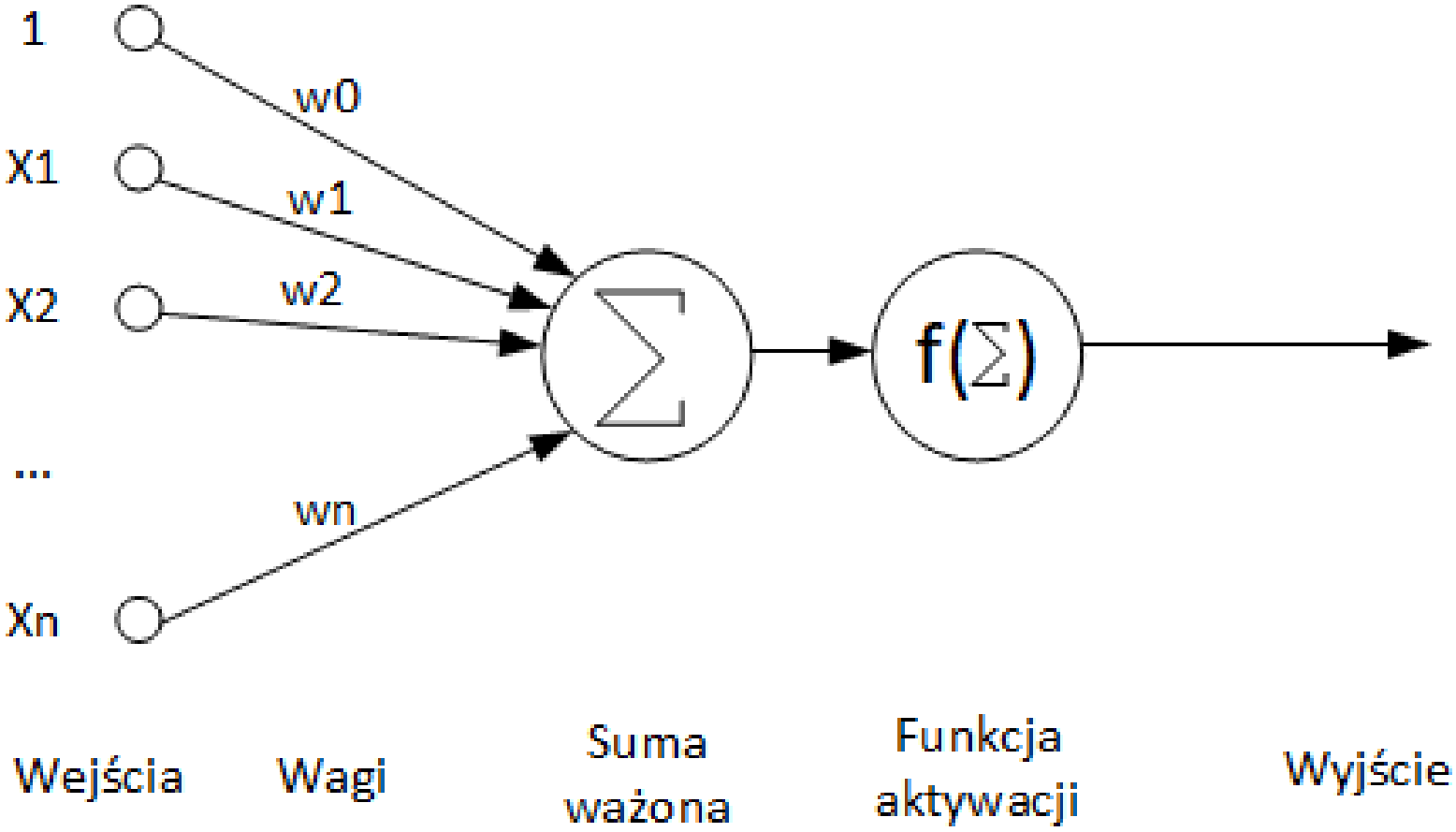
ANN (2)

- Sygnał jest liczbą rzeczywistą
- Wyjście każdego neuronu jest obliczane przez pewną nieliniową funkcję sumy jego wejść
- Połączenia są czasem nazywane krawędziami
- Neurony i połączenia zazwyczaj mają wagę, która dostosowuje się wraz z postępem uczenia się
- Waga zwiększa lub zmniejsza siłę sygnału na połączeniu

ANN (3)

- Neurony mogą mieć próg (threshold) ustawiony w taki sposób, że sygnał jest wysyłany tylko wtedy, gdy sygnał zagregowany przekroczy ten próg
- Neurony są pogrupowane (zagregowane) w warstwy
- Różne warstwy mogą wykonywać różne przekształcenia na swoich wejściach
- Sygnały wędrują od pierwszej warstwy (warstwy wejściowej) do ostatniej warstwy (warstwy wyjściowej), często po wielokrotnym przejściu przez warstwy

Neuron McCullocha-Pittsa



Wejścia i bias (wyraz wolny)

- Dane wejściowe są liczbami rzeczywistymi
- Bias (zwykle o wartości 1) pozwala przesunąć funkcję aktywacji poprzez dodanie stałej (tj. danego biasu) do wejścia.
- Można jego rolę traktować jako analogiczną do roli stałej b w funkcji liniowej $y = ax + b$, gdzie linia jest efektywnie transponowana przez stałą wartość

Funkcja aktywacji (przejścia)

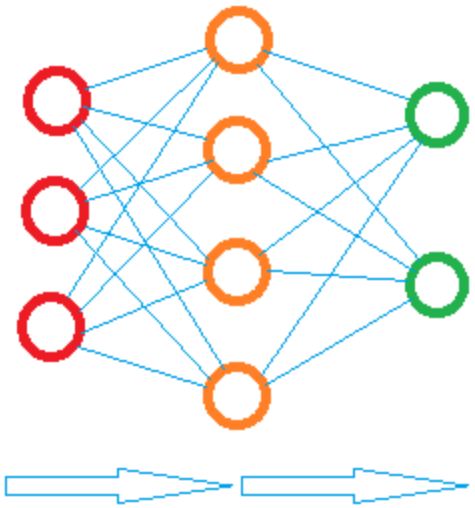
- Funkcja aktywacji węzła definiuje wyjście tego węzła przy danym wejściu lub zbiorze wejść Przykłady:
- Liniowa
- Gaussa
- Sigmoidalna
- Krok binarny (0 lub 1)

Wyjście

- Wynik sieci neuronowej
- Zależy on od zadania
  - Klasyfikacja
  - Wartość funkcji
  - Prawdopodobieństwo
  - …

Wyjście

- Warstwa wejściowa
- Warstwa ukryta
- Warstwa wyjściowa



Wartości

- Dane wejściowe:
  rekordy, wektory, obrazy, cechy, ceny itp.
- Dane wyjściowe:
  kategorie, nazwy, członkostwa itp.
- Miara jakości:
  jak bliska jest odległość między działaniem algorytmu a oczekiwanym wynikiem?

Uczenie/walidacja/zbiór testowy

- Zbiór uczący (treningowy) to zbiór przykładów używanych podczas procesu uczenia się, który służy do dopasowania parametrów (np. wag), np. klasyfikatora
- Zbiór danych walidacyjnych to zbiór danych przykładów używany do dostrajania hiperparametrów (tj. architektury) klasyfikatora
- Testowy zbiór danych to zbiór danych, który jest niezależny od treningowego zbioru danych, ale ma taki sam rozkład prawdopodobieństwa, jak treningowy zbiór danych.

Uczenie nadzorowane (z nauczycielem)

- Jest stosowane, gdy istnieje możliwość weryfikacji poprawności odpowiedzi sieci neuronowej.
  Schemat:
- Podaj wektor wejściowy
- Znajdź błąd sieci
- Użyj błędu, aby poprawić wagę siatki

Uczenie nienadzorowane (bez nauczyciela)

- Stosuje się je, gdy nie znamy odpowiedzi na dany wzór
- Przykład: mapa samoorganizująca się (Self-Organizing Map, SOM)

Propagacja wsteczna

- Propagacja wsteczna oblicza gradient w przestrzeni wag sieci neuronowej ze sprzężeniem do przodu, w odniesieniu do funkcji straty
- Przykładowe funkcje straty (f. minimalizowana w procesie dopasowywania modelu - reprezentuje wybraną miarę rozbieżności wartości dopasowanej funkcji dla danych obserwowanych i „prognozowanych”):
  - Funkcja logistyczna
  - Softmax
  - Funkcja sigmoidalna

Perceptron

Najprostsza sieć neuronowa, składająca się z jednego lub więcej niezależnych neuronów McCullocha-Pittsa, realizująca nadzorowany algorytm uczenia klasyfikatorów binarnych.

Perceptron wielowarstwowy (Multilayer Perceptron, MLP)

- Klasa sztucznej sieci neuronowej ze sprzężeniem do przodu (ANN).
- Składa się z co najmniej trzech warstw węzłów:
  - warstwy wejściowej,
  - warstwy ukrytej,
  - warstwy wyjściowej.
- Z wyjątkiem węzłów wejściowych, każdy węzeł jest neuronem, który wykorzystuje nieliniową funkcję aktywacji i propagację wsteczną do uczenia.

Przykład: Perceptron - ewaluacja

```
from numpy import mean
from numpy import std
from sklearn.datasets import make_classification
from sklearn.model_selection import cross_val_score
from sklearn.model_selection import RepeatedStratifiedKFold
from sklearn.linear_model import Perceptron

X, y = make_classification(n_samples=1000, n_features=10,
n_informative=10, n_redundant=0, random_state=1)

model = Perceptron()

cv = RepeatedStratifiedKFold(n_splits=10, n_repeats=3,
random_state=1)

scores = cross_val_score(model, X, y, scoring='accuracy', cv=cv,
n_jobs=-1)

print('Mean Accuracy: %.3f (%.3f)' % (mean(scores), std(scores)))
```

Źródło: https://machinelearningmastery.com/perceptron-algorithm-for-classification-in-python/

Przykład: Perceptron - predykcja

```
from sklearn.datasets import make_classification
from sklearn.linear_model import Perceptron

X, y = make_classification(n_samples=1000, n_features=10,
n_informative=10, n_redundant=0, random_state=1)

model = Perceptron()

model.fit(X, y)

row = [0.12777556,-3.64400522,-2.23268854,-
1.82114386,1.75466361,0.1243966,1.03397657,2.35822076,1.010
01752,0.56768485]

yhat = model.predict([row])

print(Predykowana klasa: %d' % yhat)
```

Źródło: https://machinelearningmastery.com/perceptron-algorithm-for-classification-in-python/

Pytania?

Dziękuję za uwagę!

POLITECHNIKA LUBELSKA
WYDZIAŁ ELEKTROTECHNIKI I INFORMATYKI
INFORMATYKA

Zintegrowany Program Rozwoju Politechniki Lubelskiej - część druga

PODSTAWY SZTUCZNEJ INTELIGENCJI W JĘZYKU PYTHON

Uczenie i działanie sieci neuronowej

Paweł Karczmarek

Fundusze Europejskie Wiedza Edukacja Rozwój
Rzeczpospolita Polska
Unia Europejska Europejski Fundusz Społeczny

Agenda

- Ogólne strategie uczenia sieci
- Funkcje aktywacji
- Uczenie perceptronu
- Reguły uczenia sieci

Uczenie sieci

- Sieci neuronowe mają zdolność do uczenia się, czyli zdolność do samodzielnego dostosowywania współczynników wagowych – uczenie sieci jest to wymuszenie na niej określonego zareagowania na sygnały wejściowe.
- Dzięki temu mówi się, że mają one właśnie charakter SI, bo potrafią samodzielnie dostosować się do zmieniających się warunków.
- Celem uczenia jest taki dobór wag w poszczególnych neuronach, aby sieć mogła rozwiązywać stawiane przed nią problemy.

Uczenie sieci

- Pod nadzorem
- Nienadzorowane
- Z krytykiem

Uczenie sieci

- Działanie sieci neuronowych jest uzależnione od modelu neuronu, który jest zastosowany, ale też od wartości poszczególnych parametrów topologii sieci.
- Ogólnie można stwierdzić, że „wiedza” sieci jest zawarta w parametrach (wagach), gdyż od nich zależy, z jaką siłą dana zmienna będzie wprowadzona do neuronu.
- Nie ma ogólnej reguły w odniesieniu do tego, czy lepiej stosować ujemne, czy dodatnie wagi.
- Neurony komunikują się pomiędzy warstwami, a nie w danej warstwie (do której należą)

Uczenie nadzorowane (z nauczycielem)

- Uczenie maszynowe zakładające obecność ludzkiego nadzoru nad tworzeniem funkcji odwzorowującej wejście systemu na jego wyjście
- Nadzór polega na utworzeniu zestawu danych uczących w postaci par: wejściowy obiekt uczący – pożądana przez nauczyciela (nadzorcę) odpowiedź

Uczenie nadzorowane (z nauczycielem) II

- Celem systemu jest tu uczenie się predykcji prawidłowej odpowiedzi na zadane przez pobudzenie oraz uogólnienie przypadków nauczonych na przypadki, z którymi system się jeszcze nie zetknął.

Uczenie nadzorowane (z nauczycielem) III

- Schemat postępowania:
  - Podaj wektor wejściowy
  - Wyznacz błąd popełniany przez sieć
  - Wykorzystaj otrzymany błąd do korekty wag sieci

Uczenie nienadzorowane

Uczenie maszynowe zakładające brak obecności nadzoru nad tworzeniem odwzorowania wejścia systemu na jego wyjście.

Warunki poprawnego uczenia nienadzorowanego

- Sygnały wejściowe powinny dać się jakoś sklasyfikować (powinny być podobne do pewnych wzorców – atraktorów)
- Neuronów musi być znacznie więcej niż atraktorów
- Sieć musi mieć neurony mające różnorodne początkowe preferencje

Kolejność prezentacji wzorców uczących

- Do każdej prezentacji należy losować wzorzec z jednakowym prawdopodobieństwem

Kolejność II

- Gdyby w kolejnych epokach przypadki uczące pokazywać stale w tej samej kolejności – to istniałaby obawa, że proces uczenia może zmieniać wagi w kółko, powracając po każdym cyklu do punktu wyjścia.
- Zapętleniu uczenia można zapobiec poprzez randomizację zbioru uczącego, to znaczy poprzez zmianę kolejności pokazywania poszczególnych przypadków uczących w kolejnych epokach. Wtedy proces zmiany wag w trakcie uczenia porządkuje się i wyraźnie widać, że zmierza do określonego celu, odpowiadającego optymalnemu zestawowi wag zapewniającemu rozwiązywanie stawianych sieci zadań z minimalnym błędem.

Przypomnienie

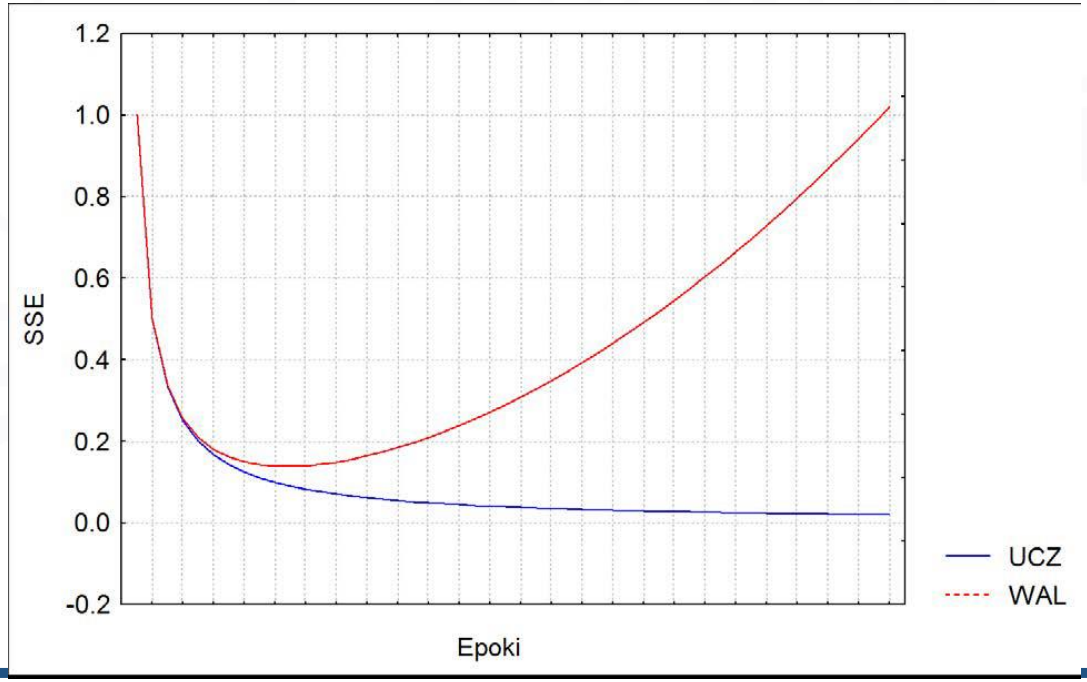
- Zbiór uczący: Zbiór przypadków uczących (zwykle ok.80%), czyli zadań zawierających dane wejściowe oraz skojarzone z nimi odpowiedzi wzorcowe
- Zbiór walidacyjny: Część zbioru uczącego (zwykle losowo wybranych około 20% przypadków uczących) przeznaczona do przeprowadzania w trakcie uczenia okresowej walidacji mającej na celu zapobieganie zjawisku przeuczenia.
- Przeuczenie: Zbyt długie uczenie sieci neuronowej powoduje, że sieć nadmiernie uzależnia swoje działanie od cech użytych do uczenia przypadków uczących – w tym także od cech drugorzędnych, nie dających podstaw do generalizacji.

Przypomnienie II

- Zbiór testowy: Część zbioru uczącego przeznaczona do przeprowadzenia po zakończeniu uczenia jednorazowej kontroli (czy w wyniku zbiegu okoliczności mimo okresowej walidacji nie doszło w trakcie uczenia do utraty zdolności generalizacji). Zbiór testowy tworzy się często w przypadku posiadania bardzo dużego zbioru uczącego, włączając do niego np. około 10% losowo wybranych przypadków uczących.
- Epoka: jednorazowe użycie w procesie uczenia wszystkich przypadków uczących zawartych w zbiorze uczącym. Po wykonaniu wszystkich kroków należących do jednej epoki algorytm uczący dokonuje oceny zdolności sieci do generalizacji wyników uczenia przy wykorzystaniu zbioru walidacyjnego. Po stwierdzeniu, że zarówno błąd obliczany na zbiorze uczącym, jak i błąd wyznaczony dla zbioru walidacyjnego nadal jeszcze obiecująco maleją – algorytm uczący wykonuje następną epokę. W przeciwnym przypadku proces uczenia zostaje zatrzymany.

Walidacja

Moment zapoczątkowania wzrostu błędu dla zbioru walidacyjnego stanowi sygnał do zakończenia szkolenia sieci i odzyskania najlepszych jej wag z epoki poprzedzającej początek wzrostu tego błędu. Źródło rys.: R. Tadeusiewicz, M. Szaleniec, Leksykon sieci neuronowych, Projekt Nauka, Wrocław 2015



Uczenie przez wzmocnienie (z krytykiem)

- Polega na interakcji ze środowiskiem za pomocą polityki na podstawie zebranych przez nią informacji.
- Nie przygotowuje się zestawu danych uczących, tylko środowisko, z którego model będzie zbierał dane automatycznie.
- Celem jest zmaksymalizowanie zwracanej przez nie nagrody.
- Większość algorytmów uczenia przez wzmacnianie polega na przygotowaniu polityki, zebraniu za jej pomocą danych o środowisku do bufora, wytrenowaniu jej na ich podstawie i powtarzaniu tego procesu do osiągnięcia zamierzonego skutku.

Reguły uczenia

- **Hebba** (bez nauczyciela, sygnał uczący to sygnał wyjściowy)
- **Perceptronowa** (z nauczycielem, sygnał uczący to różnica między wartością rzeczywistą a pożądaną)
- **Delta** (dla neuronów z ciągłymi funkcjami aktywacji i nadzorowania trybu uczenia). Polega na minimalizacji kwadratowego kryterium błędu

Reguły uczenia (2)

- **Korelacyjna** (poprawka każdej składowej wektora wag jest proporcjonalna do iloczynu odpowiedniej składowej obrazu wejściowego i pożądanego przy tym wzorca wyjścia)
- **Wygrywający bierze wszystko:** przykład nauki z rywalizacją stosowanej zazwyczaj do poznawania własności statystycznych sygnałów wejściowych w trybie bez nauczyciela

Funkcje aktywacji

- Identycznościowa

$$\phi(x) = x$$

- Pochodna

$$\phi'(x) = 1$$

- Zakres wartości

$$(-\infty, \infty)$$

Funkcje aktywacji

- Funkcja skokowa Heaviside'a, skok jednostkowy

$$\phi(x) = \begin{cases} 0 \text{ dla } x < 0 \\ 1 \text{ dla } x \geq 0 \end{cases}$$

- Pochodna

$$\phi'(x) = \begin{cases} 0 \text{ dla } x \neq 0 \\ \text{niezdefiniowana dla } x = 0 \end{cases}$$

- Zakres wartości

$$\{0,1\}$$

Funkcje aktywacji

- Logistyczna, sigmoidalna

$$\phi(x) = \frac{1}{1 + e^{-x}}$$

- Pochodna

$$\phi'(x) = \phi(x)\big(1 - \phi(x)\big)$$

- Zakres wartości

$$(0,1)$$

Funkcje aktywacji

- Tangens hiperboliczny

$$\phi(x) = \frac{e^{x} - e^{-x}}{e^{x} + e^{-x}}$$

- Pochodna

$$\phi'(x) = 1 - \phi^{2}(x)$$

- Zakres wartości

$$(-1,1)$$

Funkcje aktywacji

- ReLU (rectified linear unit)

$$\phi(x) = \max(0, x)$$

- Pochodna

$$\phi'(x) = \begin{cases} 0 \text{ dla } x < 0 \\ 1 \text{ dla } x > 0 \\ \text{niezdefiniowana dla } x = 0 \end{cases}$$

- Zakres wartości

$$[0, \infty)$$

Funkcje aktywacji

- Gaussowska

$$\phi(x) = e^{-x^2}$$

- Pochodna

$$\phi'(x) = -2xe^{-x^2}$$

- Zakres wartości

$$(0, 1]$$

Funkcje aktywacji

- Softmax

$$\phi(x) = \frac{e^{x_i}}{\sum_{j=1}^{J} e^{x_j}}$$

- Pochodna

$$\phi'(x) = \phi_i(\vec{x})\left(\delta_{ij} - \phi_j(\vec{x})\right)$$

- Zakres wartości

$$(0,1)$$

Pytania?

Dziękuję za uwagę!

POLITECHNIKA LUBELSKA
WYDZIAŁ ELEKTROTECHNIKI I INFORMATYKI
INFORMATYKA

Zintegrowany Program Rozwoju Politechniki Lubelskiej - część druga

PODSTAWY SZTUCZNEJ INTELIGENCJI W JĘZYKU PYTHON

Sieci jednowarstwowe, wielowarstwowe i propagacja wsteczna

Paweł Karczmarek

Fundusze Europejskie Wiedza Edukacja Rozwój
Rzeczpospolita Polska
Unia Europejska Europejski Fundusz Społeczny

Agenda

- Sieci jednokierunkowe
- Sieci jednowarstwowe
- Sieci wielowarstwowe
- Propagacja wsteczna

Sieć jednokierunkowa

- Sieć, w której przepływ sygnałów odbywa się wyłącznie w kierunku od wejścia (poprzez ewentualne warstwy ukryte) do wyjścia.
- Wykluczony jest przepływ sygnałów w drugą stronę, co powoduje, że sieci tego typu są przeciwstawiane sieciom rekurencyjnym.
- Nazywane są sieciami typu feedforward.
- Często są utożsamiane z sieciami typu MLP, ale należą do nich również inne rodzaje sieci (np. radialne – RBF, uogólnionej regresji – GRNN, probabilistyczne – PNN).

Perceptron

- Składa się z pojedynczej warstwy węzłów wyjściowych.
- Wejścia są podawane bezpośrednio na wyjścia z uwzględnieniem wag.
- Suma iloczynów wag i wejść jest obliczana w każdym węźle, a jeśli wartość jest powyżej pewnego progu (zwykle 0), neuron aktywuje się i przyjmuje aktywowaną wartość (zwykle 1). W przeciwnym razie przyjmuje dezaktywowaną wartość (zwykle -1).

Perceptron

- Perceptron można utworzyć przy użyciu dowolnych wartości dla stanów aktywowanych i dezaktywowanych, o ile wartość progowa znajduje się pomiędzy nimi.
- Perceptrony można trenować za pomocą prostego algorytmu uczenia, który zwykle nazywa się regułą delta.
- Oblicza błędy między obliczonymi danymi wyjściowymi a danymi wyjściowymi próbki i wykorzystuje je do stworzenia korekty wag, realizując w ten sposób formę metody gradientu prostego.

Perceptron

- Perceptrony jednowarstwowe są zdolne do uczenia się tylko liniowo rozdzielonych wzorów.
- Jednowarstwowa sieć perceptronowa nie jest w stanie nauczyć się funkcji XOR.
- Jest w stanie aproksymować każdą ciągłą funkcję określoną na przedziale zwartym o wartościach w [-1, 1].
- Często wybierana funkcja aktywacji to funkcja logistyczna.

Perceptron - uczenie

- Perceptron możemy zapisać jako

$$y = f\left(\sum_{i=1}^{n} w_i x_i + \theta\right)$$

- Przyjmijmy, że funkcja aktywacji jest bipolarna, tj.

$$f(s) = \begin{cases} 1 \text{ dla } s > 0 \\ -1 \text{ dla } s \leq 0 \end{cases}$$

Algorytm uczenia perceptronu

1. Wybieramy losowo wagi początkowe.
2. Na wejścia podajemy wektor uczący $\boldsymbol{x}$, przy czym $\boldsymbol{x} = \boldsymbol{x}(t) = [x_0(t), x_1(t), \dots, x_n(t)]^T, t = 1,2, \dots$
3. Obliczamy wartość wyjściową *y.*
4. Porównujemy wartość wyjściową $y(t)$ z wartością wzorcową $d = d\big(\boldsymbol{x}(t)\big)$ znajdującą się w ciągu uczącym.
5. Dokonujemy modyfikacji wag:
   1. Jeżeli $y\big(\boldsymbol{x}(t)\big) \neq d\big(\boldsymbol{x}(t)\big)$, to $w_i(t+1) = w_i(t) + d\big(\boldsymbol{x}(t)\big)x_i(t)$
   2. W przeciwnym wypadku – bez zmian
6. Wracamy do punktu 2.

Algorytm uczenia perceptronu

Algorytm powtarza się tak długo, aż dla wszystkich wektorów uczących błąd na wyjściu będzie mniejszy od założonej tolerancji.

Epoka

Epoka są to dane tworzące ciąg uczący.

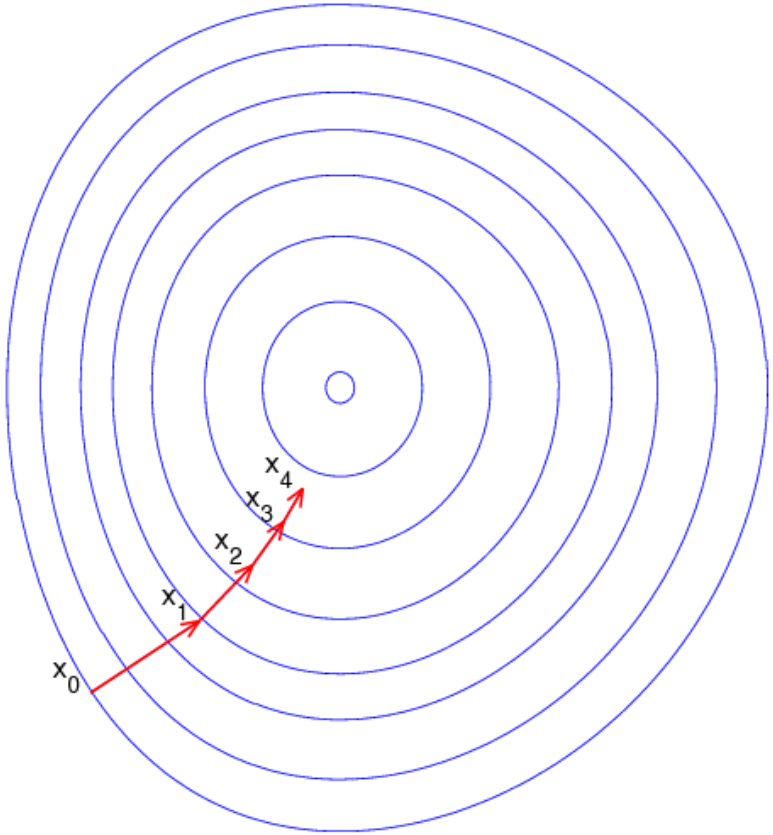
Błąd jest obliczany dla całej epoki.

Jeżeli jest on większy od dopuszczalnego, należy powtórzyć algorytm uczenia.

Metoda gradientu prostego (gradient descent)

- Iteracyjny algorytm wyszukiwania minimum zadanej funkcji celu $f: D \to \mathbb{R}$
- $D \subset \mathbb{R}^n$
- f jest ciągła i różniczkowalna
- f jest wypukła w analizowanej dziedzinie
- Metoda polega na poszukiwaniu minimum (począwszy od pewnego punktu początkowego $\boldsymbol{x}_0$) w kierunku określanym przez antygradient funkcji celu, tj. $-\nabla f(\boldsymbol{x}_k)$

Metoda gradientu prostego



- Źródło: https://commons.wikimedia.org/wiki/File:Gradient_descent.png

Metoda gradientu prostego

Algorytm

1. Wybierz $\boldsymbol{x}_0$
2. $\boldsymbol{x}_{k+1} = \boldsymbol{x}_k - \alpha_k \nabla f(\boldsymbol{x}_k)$
3. Sprawdź kryterium stopu, jeśli jest spełnione, to zakończ algorytm z ostatnio wyznaczonym $\boldsymbol{x}$. Możliwe kryteria: $\|\boldsymbol{x}_{k+1} - \boldsymbol{x}_k\| < \varepsilon$ albo $\|\nabla f(\boldsymbol{x}_k)\| < \varepsilon$
4. Jeżeli $f(\boldsymbol{x}_{k+1}) > f(\boldsymbol{x}_k)$, to zmniejsz α_k
5. Powtórz punkt 2 dla następnego kroku

Metoda jest zbieżna liniowo

Perceptron wielowarstwowy

- Składa się z wielu warstw neuronów, zwykle połączonych w sposób jednokierunkowy.
- Każdy neuron w jednej warstwie ma skierowane połączenia do neuronów kolejnej warstwy.
- W wielu aplikacjach neurony tych sieci stosują funkcję sigmoidalną jako funkcję aktywacji.
- Jedną z najczęstszych metod uczenia MLP jest algorytm propagacji wstecznej (back-propagation).

Wsteczna propagacja błędów

- Nauczyciel nie wie, jaka powinna być wartość na wyjściu warstw ukrytych.
- Funkcją błędu jest funkcja, której zmiennymi są wszystkie wagi wielowarstwowej sieci neuronowej
- Oznacznie: $Q(\boldsymbol{w})$
- $\boldsymbol{w}$ jest wektorem wszystkich wag sieci
- Dążymy do znalezienia minimum funkcji Q względem wektora $\boldsymbol{w}$

Rozwijamy Q w szereg Taylora w najbliższym sąsiedztwie znanego aktualnego rozwiązania $\boldsymbol{w}$ w kierunku $\boldsymbol{p}$

$$Q(\boldsymbol{w}+\boldsymbol{p}) = Q(\boldsymbol{w}) + [\boldsymbol{g}(\boldsymbol{w})]^T\boldsymbol{p} + \frac{1}{2}\boldsymbol{p}^T\boldsymbol{H}(\boldsymbol{w})\boldsymbol{p} + \cdots \quad (1)$$

Wsteczna propagacja błędów (2)

gdzie $\boldsymbol{g}(\boldsymbol{w})$ jest gradientem

$$\boldsymbol{g}(\boldsymbol{w}) = \left[\frac{\partial Q}{\partial w_1}, \frac{\partial Q}{\partial w_2}, \ldots, \frac{\partial Q}{\partial w_n}\right]^T \quad (2)$$

zaś $\boldsymbol{H}(\boldsymbol{w})$ hesjanem

$$\boldsymbol{H}(\boldsymbol{w}) = \left[\begin{pmatrix} \frac{\partial^2 Q}{\partial w_1 \partial w_1} & \cdots & \frac{\partial^2 Q}{\partial w_1 \partial w_n} \\ \vdots & \ddots & \vdots \\ \frac{\partial^2 Q}{\partial w_n \partial w_1} & \cdots & \frac{\partial^2 Q}{\partial w_n \partial w_n} \end{pmatrix}\right] \quad (3)$$

Modyfikujemy wagi:

$$\boldsymbol{w}(t+1) = \boldsymbol{w}(t) + \eta(t)\boldsymbol{p}(t) \quad (4)$$

gdzie η jest współczynnikiem uczenia.

Wsteczna propagacja błędów (3)

Należy znaleźć taki wektor $\boldsymbol{p}$, aby w kolejnych krokach algorytmu błąd na wyjściu sieci malał, tj. $Q\big(\boldsymbol{w}(t+1)\big) < Q\big(\boldsymbol{w}(t)\big)$ w kolejnych krokach iteracji. Ograniczamy szereg Taylora do rozwinięcia liniowego:

$$Q(\boldsymbol{w}+\boldsymbol{p}) = Q(\boldsymbol{w}) + [\boldsymbol{g}(\boldsymbol{w})]^T \boldsymbol{p} \quad (5)$$

Ponieważ $Q(\boldsymbol{w})$ zależy od wagi z kroku t, a $Q(\boldsymbol{w}+\boldsymbol{p})$ od wag z kroku t+1, więc $\boldsymbol{p}$(t) należy dobrać tak, aby $[\boldsymbol{g}(\boldsymbol{w})]^T \boldsymbol{p}(t) < 0$. Warunek zostanie spełniony, jeśli przyjmiemy

$$\boldsymbol{p}(t) = -\boldsymbol{g}(\boldsymbol{w}(t)) \quad (6)$$

Podstawiając (6) do (4), otrzymujemy wzór określający sposób zmiany wag:

$$w(t+1) = w(t) - \eta\, \boldsymbol{g}(\boldsymbol{w}(t)) \quad (7)$$

Wsteczna propagacja błędów (4)

Jest to tzw. reguła najmniejszego spadku (steepest descent). Załóżmy teraz, że w każdej z L warstw sieci znajduje się N_k $(k = 1, \dots, L)$ elementów, oznaczonych jako ${N_i}^k$ $(i = 1, \dots, N_k)$, tj. neuronów. Każdy z nich może być neuronem sigmoidalnym. Omawiana sieć ma N_0 wejść, na które są poustawiane sygnały $x_1(t), \dots, x_{N_0}(t)$:

$$\boldsymbol{x} = \left[x_1(t), \dots, x_{N_0}(t)\right]^T, t = 1,2, \dots \quad (8)$$

Sygnał wyjściowy *i*-tego neuronu w warstwie *k*-tej oznaczmy przez ${y_i}^{(k)}(t), i = 1, \dots, N_k, k = 1, \dots, L.$

${N_i}^k$ ma N_k wejść tworzących wektor

$$\boldsymbol{x}^{(k)}(t) = \left[{x_1}^{(k)}(t), \dots, {x_{N_{k-1}}}^{(k)}(t)\right]^T \quad (9)$$

Wsteczna propagacja błędów (5)

Ponadto $x_i{}^{(k)}(t) = 1$ dla $i = 0, k = 1, \dots, L$. Sygnał wejściowy neuronu $N_i{}^k$ jest związany z sygnałem wyjściowym warstwy $k-1$ w sposób następujący:

$$x_i{}^{(k)}(t) = \begin{cases} x_i(t), k = 1 \\ y_i{}^{(k-1)}(t), k = 2, \dots, L \\ +1, i = 0, k = 1, \dots, L \end{cases} \quad (10)$$

$w_{ij}{}^{(k)}(t)$ jest wagą *i*-tego neuronu, $i = 1, \dots, N_k$, łączącą ten neuron z *j*-tym sygnałem wejściowym $x_j{}^{(k)}(t), j = 0,1, \dots, N_k$. Wektor wag neuronu $N_i{}^k$ to

$$\boldsymbol{w}_i{}^{(k)}(t) = \left[w_{i0}{}^{(k)}(t), \dots, w_{iN_{k-1}}{}^{(k)}(t)\right]^T, k = 1, \dots, L, i = 1, \dots, N_k \quad (11)$$

Wsteczna propagacja błędów (6)

Sygnał wyjściowy neuronu ${N_i}^k$ w chwili t definiujemy jako

$${y_i}^{(k)}(t) = f\left({s_i}^{(k)}(t)\right) \quad (12)$$

gdzie

$${s_i}^{(k)}(t) = \sum_{j=0}^{N_{i-1}} {w_{ij}}^{(k)}(t){x_j}^{(k)}(t) \quad (13)$$

Zauważmy, że sygnały wyjściowe w L-tej warstwie, tj.

$${y_1}^L(t), \ldots, {y_{N_L}}^L(t) \quad (14)$$

są też sygnałami wyjściowymi całej sieci. Są one porównywane z sygnałami wzorcowymi sieci

$${d_1}^L(t), \ldots, {d_{N_L}}^L(t) \quad (15)$$

Błąd na wyjściu sieci Q można zdefiniować jako

$$Q(t) = \sum_{i=1}^{N_L} {\varepsilon_i}^{(L)^2}(t) = \sum_{i=1}^{N_L}\left({d_i}^L(t) - {y_i}^L(t)\right)^2 \quad (16)$$

Wsteczna propagacja błędów (7)

Na mocy (7) i (16) dostajemy

$$w_{ij}^{(k)}(t+1) = w_{ij}^{(k)}(t) - \eta \frac{\partial Q(t)}{\partial w_{ij}^{(k)}(t)} \qquad (17)$$

Zauważmy, że

$$\frac{\partial Q(t)}{\partial w_{ij}^{(k)}(t)} = \frac{\partial Q(t)}{\partial s_i^{(k)}(t)} \frac{\partial s_i^{(k)}(t)}{\partial w_{ij}^{(k)}(t)} = \frac{\partial Q(t)}{\partial s_i^{(k)}(t)} x_j^{(k)}(t) \qquad (18)$$

Oznaczmy

$$\delta_i^{(k)}(t) = -\frac{1}{2} \frac{\partial Q(t)}{\partial s_i^{(k)}(t)} \qquad (19)$$

Otrzymamy

$$\frac{\partial Q(t)}{\partial w_{ij}^{(k)}(t)} = -2\delta_i^{(k)}(t) x_j^{(k)}(t) \qquad (20)$$

A zatem (17) zapisujemy w postaci

Wsteczna propagacja błędów (8)

$$w_{ij}^{(k)}(t+1) = w_{ij}^{(k)}(t) + 2\eta\delta_i^{(k)}(t)x_j^{(k)}(t) \qquad (21)$$

Dla warstwy ostatniej $\delta_i^{(k)}$ wyznaczamy następująco:

$$\delta_i^{(L)}(t) = -\frac{1}{2}\frac{\partial Q(t)}{\partial s_i^{(L)}(t)} = -\frac{1}{2}\frac{\partial \sum_{m=1}^{N_L} Q_m^{(L)^2}(t)}{\partial s_i^{(L)}(t)} =$$

$$-\frac{1}{2}\frac{\partial Q_i^{(L)^2}(t)}{\partial s_i^{(L)}(t)} = -\frac{1}{2}\frac{\partial\left(d_i^{(L)}(t) - y_i^{(L)}(t)\right)^2}{\partial s_i^{(L)}(t)} =$$

$$Q_i^{(L)}(t)\frac{\partial y_i^{(L)}(t)}{\partial s_i^{(L)}(t)} = Q_i^{(L)}(t)f'\left(s_i^{(L)}(t)\right) \qquad (22)$$

Dla dowolnej warstwy *k* różnej od *L*, otrzymujemy

Wsteczna propagacja błędów (9)

$$\delta_i^{(L)}(t) = -\frac{1}{2}\frac{\partial Q(t)}{\partial s_i^{(L)}(t)} =$$

$$-\frac{1}{2}\sum_{m=1}^{N_{k+1}}\frac{\partial Q(t)}{\partial s_m^{(k+1)}(t)}\frac{\partial s_m^{(k+1)}(t)}{\partial s_i^{(k)}(t)} =$$

$$\sum_{m=1}^{N_{k+1}}\delta_i^{(k+1)}(t)w_{mi}^{(k+1)}(t)f'\left(s_i^{(k)}(t)\right) =$$

$$f'\left(s_i^{(k)}(t)\right)\sum_{m=1}^{N_{k+1}}\delta_i^{(k+1)}(t)w_{mi}^{(k+1)}(t) \quad (23)$$

Możemy zdefiniować błąd w warstwie k-tej (oprócz ostatniej, $k = 1, \ldots, L-1$) dla i-tego neuronu

$$\varepsilon_i^{(k)}(t) = \sum_{m=1}^{N_{k+1}}\delta_m^{(k+1)}(t)w_{mi}^{(k+1)}(t) \quad (24)$$

Podstawiając (24) do (18) mamy

$$\delta_i^{(k)}(t) = \varepsilon_i^{(k)}(t)f'\left(s_i^{(k)}(t)\right) \quad (25)$$

Algorytm

$$y_i^{(k)}(t) = f\left(s_i^{(k)}(t)\right), s_i^{(k)}(t) = \sum_{j=0}^{N_{i-1}} w_{ij}^{(k)}(t) x_j^{(k)}(t) \quad (26)$$

$$Q_i^{(L)}(t) = \begin{cases} d_i^{(L)}(t) - y_i^{(L)}(t) \text{ dla } k = L \\ \sum_{m=1}^{N_{k+1}} \delta_m^{(k+1)}(t) w_{mi}^{(k+1)}(t) \text{ dla } k = 1, \ldots, L-1 \end{cases} \quad (27), (28)$$

$$\delta_i^{(k)}(t) = \varepsilon_i^{(k)}(t) f'\left(s_i^{(k)}(t)\right) \quad (29)$$

$$w_{ij}^{(k)}(t+1) = w_{ij}^{(k)}(t) + 2\eta \delta_i^{(k)}(t) x_j^{(k)}(t) \quad (30)$$

Uwagi

- Zauważmy, że wykorzystując regułę delta, możemy zmodyfikować wagi neuronów ostatniej warstwy na podstawie wzorów (27), (29) i (30).
- Jednak w ten sposób nie możemy zmodyfikować neuronów w warstwach ukrytych – nie znamy $\delta_i^{(k)}$ dla neuronów tych warstw.
- Dlatego błąd wyjściowy jest propagowany od tyłu zgodnie z połączeniami neuronów i uwzględnieniem ich funkcji aktywacji – wzory (28), (29) i (30).

Uwagi (2)

- W dyskutowanej metodzie uczenie sieci przeprowadza się metodą przyrostowego uaktualnienia wag, tzn. każdorazowo po podaniu na wejście wektora uczącego.
- Można też zastosować kumulacyjne uaktualnianie wag, tzn. sumować otrzymane w kolejnych iteracjach błędy w ramach jednej epoki.

Uwagi (3)

- Inicjalizacja wag (problem wielu minimów lokalnych)
- Jedno z rozwiązań: różne wartości wag początkowych
- Dobór współczynnika uczenia η z zakresu $(0, 1)$ – w dużej mierze zależy od doświadczenia
- Ogólna zasada: płaska funkcja celu $=>$ większe η, stroma funkcja celu $=>$ mniejsze η
- Wykład opracowano na podstawie L. Rutkowski, *Metody i techniki sztucznej inteligencji*, PWN, Warszawa 2012

Pytania?

Dziękuję za uwagę!

POLITECHNIKA LUBELSKA
WYDZIAŁ ELEKTROTECHNIKI I INFORMATYKI
INFORMATYKA

Zintegrowany Program Rozwoju Politechniki Lubelskiej - część druga

PODSTAWY SZTUCZNEJ INTELIGENCJI W JĘZYKU PYTHON

Sieci samouczące, samoorganizujące się i rekurencyjne

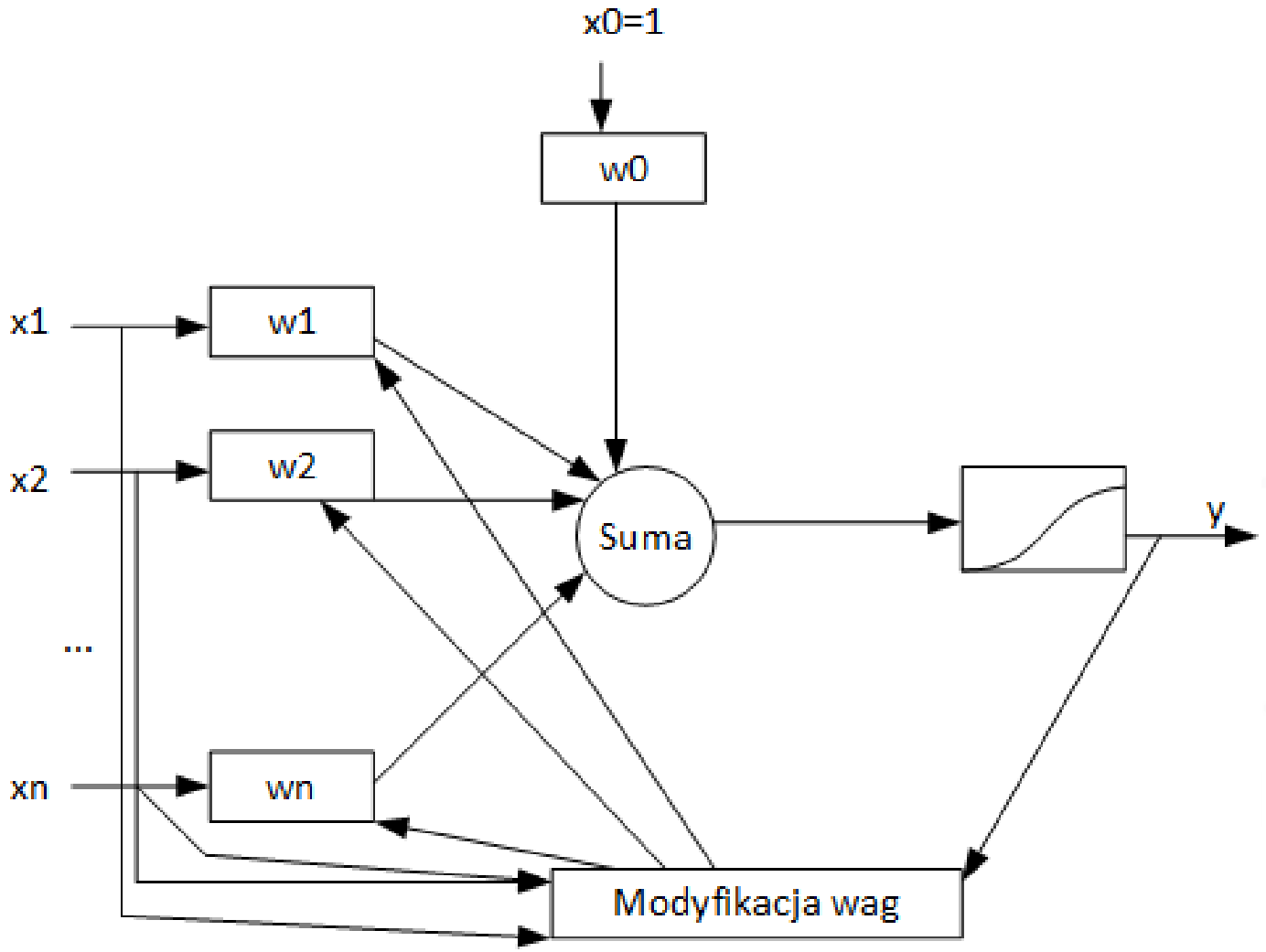
Paweł Karczmarek

Fundusze Europejskie Wiedza Edukacja Rozwój
Rzeczpospolita Polska
Unia Europejska Europejski Fundusz Społeczny

Agenda

- Model neuronu Hebba
- Wybrane zagadnienia doboru architektury sieci
- Sieć Hopfielda
- Sieć Kohonena

Neuron Hebba



Neuron Hebba

- Hebb zaobserwował, że połączenie pomiędzy dwoma komórkami jest wzmacniane, jeżeli w tym samym czasie obie komórki stają się aktywne.
- Zaproponował algorytm modyfikacji wag:

$$w_i(t+1) = w_i(t) + \Delta w_i$$

- zaś

$$\Delta w_i = \eta y x_i$$

Neuron Hebba

- W przypadku pojedynczego neuronu w trakcie uczenia modyfikuje się wartość wagi w_i proporcjonalnie zarówno do wartości sygnału podanego na *i*-te wejście, jak i sygnału wyjściowego *y* z uwzględnieniem współczynnika uczenia η (nie podajemy tu wartości wzorcowej – metoda bez nauczyciela)
- Modyfikacja dla metody z nauczycielem:

$$\Delta w_i = \eta x_i d$$

  Gdzie d oznacza sygnał wzorcowy
- Wada metody: Wagi mogą dowolnie rosnąć

Twierdzenie Kołmogorowa

Dowolną funkcję rzeczywistą $f(x_1, \dots, x_n)$, określoną na $[0,1]^n, n > 1$, można aproksymować za pomocą funkcji

$$F(\boldsymbol{x}) = \sum_{j=1}^{2n+1} g_j \left(\sum_{i=1}^{n} \phi_{ij}(x_i) \right)$$

gdzie $\boldsymbol{x} = [x_1, \dots, x_n]^T, g_j, j = 1, \dots, 2n+1$ są funkcjami ciągłymi jednej zmiennej, natomiast $\phi_{ij}, i = 1, \dots, n, j = 1, \dots, 2n+1$ są ciągłe i monotonicznie rosnące i nie zależą od *f*.

Twierdzenie Kołmogorowa - wnioski

- Zauważmy, że strukturę odpowiadającą zależności z twierdzenia tworzy sieć neuronowa dwuwarstwowa o n wejściach, $2n+1$ neuronach w warstwie ukrytej i jednym neuronie z liniową funkcją aktywacji w warstwie wyjściowej.
- Twierdzenie Kołmogorowa to jednak wynik czysto teoretyczny – nie podaje rodzaju funkcji nieliniowych oraz metod uczenia sieci.

Twierdzenie

Załóżmy, że ϕ jest dowolną funkcją ciągłą sigmoidalną. Wtedy dla każdej ciągłej funkcji f zdefiniowanej na $[0,1]^n, n > 1$, i dla dowolnego $\varepsilon > 0$ istnieje liczba całkowita N i zbiór stałych $\alpha_i, \theta_i, w_{ij}, i = 1, \dots, N, j = 1, \dots, n$, takich, że funkcja

$$F(x_1, \dots, x_n) = \sum_{i=1}^{N} \alpha_i \phi \left(\sum_{i=1}^{n} w_{ij} x_j - \theta_i \right)$$

aproksymuje funkcję f, tzn.

$$|F(x_1, \dots, x_n) - f(x_1, \dots, x_n)| < \varepsilon$$

dla wszystkich $\{x_1, \dots, x_n\} \in [0,1]^n$.

Wniosek

Sieć neuronowa z liniowym neuronem wyjściowym oraz jedną warstwą ukrytą o neuronach z sigmoidalną funkcją aktywacji może aproksymować dowolną funkcję ciągłą zdefiniowaną na $[0,1]^n$.

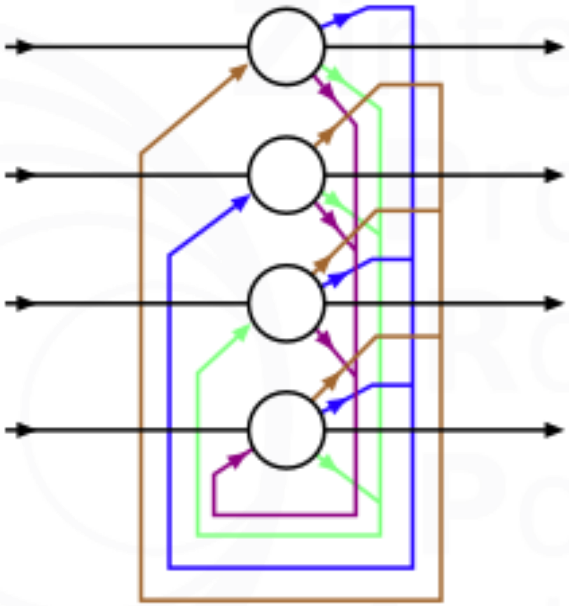
Uwaga

- Na działanie sieci ma znaczenie liczba neuronów w poszczególnych warstwach.
- Zbyt duża liczba neuronów wydłuża proces uczenia.
- Jeżeli liczba próbek uczących jest niewielka w porównaniu z rozmiarem sieci, strukturę można przeuczyć, tzn. straci ona możliwość uogólniania wiedzy – sieć nauczy się na pamięć ciągu uczącego i będzie poprawnie odwzorowywać tylko próbki w nim zawarte.

Sprzężenie zwrotne i sieci rekurencyjne

- Sytuacja, w której sygnał otrzymany na wyjściu sieci z powrotem trafia na jej wejście, nazywana jest sprzężeniem zwrotnym
- Sieci neuronowe, które oparte są na takim schemacie, nazywane są sieciami rekurencyjnymi.
- Ogólnie można powiedzieć, że rekurencyjne sieci neuronowe to sieci, w których połączenia pomiędzy neuronami tworzą graf z cyklami.

Sieć Hopfielda



Źródło:
https://commons.wikimedia.org/wiki/File:Hopfield-net-vector.svg
Autor: Zawersh

Sieć Hopfielda

- Sieć Hopfielda jest jednowarstwowa i ma regularną budowę
- Składa się z wielu neuronów połączonych każdy z każdym
- Nie istnieją sprzężenia zwrotne obejmujące ten sam neuron
- Wobec tego sygnał wyjściowy danego neuronu nie trafia na jego wejście, tj. waga $w_{ii} = 0$
- Wagi są symetryczne, tzn. waga w_{kj}, łącząca neuron *k* z neuronem *j,* jest równa wadze w_{jk}, która łączy neuron *j* z neuronem *k*

Sieć Hopfielda

- Podczas uczenia modyfikuje swoje wagi w zależności od wartości wektora uczącego ***x***
- W trybie odtworzeniowym wagi nie ulegają modyfikacjom, natomiast sygnał wejściowy pobudza sieć, która poprzez sprzężenie zwrotne wielokrotnie przyjmuje na swoje wejście sygnał wyjściowy, aż do ustabilizowania odpowiedzi

Sieć Hopfielda

- Daje możliwość rekonstrukcji i rozpoznawania wcześniej zapamiętanych wzorców na podstawie skojarzeń, bazując na dostępnym fragmencie wzorca lub wzorca podobnego do niego
- Stosuje się do modelowania pamięci skojarzeniowej. W tych sieciach nie ma wyszczególnionych warstw, każda jednostka przetwarzająca jest złączona ze wszystkimi innymi jednostkami z wyjątkiem samej siebie. Połączenie pomiędzy dwiema jednostkami jest symetryczne, co oznacza, że ma taką samą siłę w obie strony

Sieć Hopfielda

- Procesy obliczeniowe polegają na nauczeniu sieci wzorcowych danych, a w dalszej kolejności na prezentacji dowolnych danych na wejściu
- Sygnał rozprzestrzenia się po sieci aż do momentu samoczynnego uzyskania stanu stabilnego, kiedy już nie zachodzą zmiany aktywacji żadnych jednostek
- Stan aktywacji opisany na zbiorze jednostek przetwarzających jest przesyłany na wyjście systemu i jest wzorcem najbardziej zbliżonym do danych wejściowych

Sieć Kohonena

- Jeden z podstawowych typów sieci samoorganizujących się.
- Dzięki zdolności samoorganizacji otwierają się zupełnie nowe możliwości - adaptacja do wcześniej nieznanych danych wejściowych, o których bardzo niewiele wiadomo.
- Wydaje się to naturalnym sposobem uczenia, który jest używany chociażby w mózgach, którym nikt nie definiuje żadnych wzorców, tylko muszą się one krystalizować w trakcie procesu uczenia, połączonego z normalnym funkcjonowaniem.

Sieć Kohonena

- Sieci Kohonena stanowią synonim całej grupy sieci, w których uczenie odbywa się metodą samoorganizującą typu konkurencyjnego.
- Polega ona na podawaniu na wejścia sieci sygnałów, a następnie wybraniu w drodze konkurencji zwycięskiego neuronu, który najlepiej odpowiada wektorowi wejściowemu.
- Dokładny schemat konkurencji i późniejszej modyfikacji wag synaptycznych może mieć różną postać. Podtypy sieci opartych na konkurencji różnią się dokładnym algorytmem samoorganizacji.

Etapy działania

- Konstrukcja
- Uczenie
- Rozpoznawanie

Sieć Kohonena

- System realizujący powinien składać się z kilku podstawowych elementów.
- Pierwszym z nich jest macierz neuronów pobudzanych przez sygnały wejściowe.
- Sygnały te powinny opisywać pewne charakterystyczne cechy zjawisk zachodzących w otoczeniu tak, aby na ich podstawie sieć była w stanie je pogrupować.

Sieć Kohonena

- Informacja o zdarzeniach jest przekładana na bodźce pobudzające neurony.
- Zbiór sygnałów przekazywanych do każdego neuronu nie musi być identyczny, nawet ich ilość może być różna.
- Muszą one jednak spełniać pewien warunek, a mianowicie jednoznacznie określać dane zdarzenia.

Sieć Kohonena

- Kolejną częścią składową sieci jest mechanizm, który dla każdego neuronu określa stopień podobieństwa jego wag do danego sygnału wejściowego oraz wyznacza jednostkę z największym dopasowaniem - zwycięzcę.
- Obliczenia zaczynamy dla wag równych małym liczbom losowym, przy czym ważne jest, aby nie zachodziła żadna symetria.
- W trakcie uczenia wagi te są modyfikowane w taki sposób, aby najlepiej odzwierciedlać wewnętrzną strukturę danych wejściowych.
- Istnieje jednak niebezpieczeństwo, że związą się one z pewnymi wartościami zanim jeszcze grupy zostaną prawidłowo rozpoznane i wtedy trzeba ponawiać uczenie z innymi wagami.

Sieć Kohonena

- Koniecznym do przeprowadzenia samoorganizacji jest, aby sieć była wyposażona w zdolność do adaptacji wartości wag neuronu zwycięzcy i jego sąsiadów w zależności od siły, z jaką odpowiedział on na dane wejście.
- Topologię sieci można w łatwy sposób określić poprzez zdefiniowanie sąsiadów dla każdego neuronu.
- Załóżmy, że jednostkę, której odpowiedź na dane pobudzenie jest maksymalna, będziemy nazywali "obrazem" tego pobudzenia. Wtedy możemy przyjąć, że sieć jest uporządkowana, jeśli topologiczne relacje między sygnałami wejściowymi i ich obrazami są takie same.

Uczenie

- Algorytmem, od którego nazwę wzięła cała klasa sieci są samoorganizujące się mapy Kohonena.
- Kohonen zaproponował dwa rodzaje sąsiedztwa: prostokątne i gaussowskie:

$$G(i,x) = \begin{cases} 1 \text{ dla } d(i,w) \leq \lambda \\ 0 \text{ w przeciwnym przypadku} \end{cases}$$

$$G(i,x) = e^{-\frac{d^2(i,w)}{2\lambda^2}}$$

- λ to promień sąsiedztwa malejący w czasie

Schemat działania

Źródło
https://commons.wikimedia.org/wiki/File:Somtraining.svg

Opis

- Niebieska plama jest rozkładem przykładów zestawu uczącego sieć, biała kropka obrazuje aktualnie wylosowany przypadek z tej dystrybucji.
- W pierwszej fazie (od lewej) sieć jest losowo umieszczona w przestrzeni danych.
- Węzeł najbliższy wylosowanemu przypadkowi (podświetlony na żółto) jest przesuwany w jego kierunku.
- Jego sąsiedzi także w mniejszym stopniu zmieniają swoje pozycje (ma to na celu zachowanie równomiernego rozkładu siatki).
- Po wielu krokach siatka ma tendencję do odwzorowania analizowanych przykładów (po prawej)

Przykłady implementacji

- Sieć Hopfielda:

https://programmersought.com/article/8678739126/

- Sieć Kohonena:

https://rubikscode.net/2021/07/06/implementing-self-organizing-maps-with-python-and-tensorflow/

Pytania?

Dziękuję za uwagę!

POLITECHNIKA LUBELSKA
WYDZIAŁ ELEKTROTECHNIKI I INFORMATYKI
INFORMATYKA

Zintegrowany Program Rozwoju Politechniki Lubelskiej - część druga

PODSTAWY SZTUCZNEJ INTELIGENCJI W JĘZYKU PYTHON

Głębokie sieci neuronowe i uczenie głębokich sieci neuronowych

Paweł Karczmarek

Fundusze Europejskie Wiedza Edukacja Rozwój
Rzeczpospolita Polska
Unia Europejska Europejski Fundusz Społeczny

Agenda

- Ogólna koncepcja
- Struktura
- Uczenie
- Funkcja aktywacji
- Przygotowywanie danych
- Ogólny schemat pracy z głębokimi sieciami neuronowymi

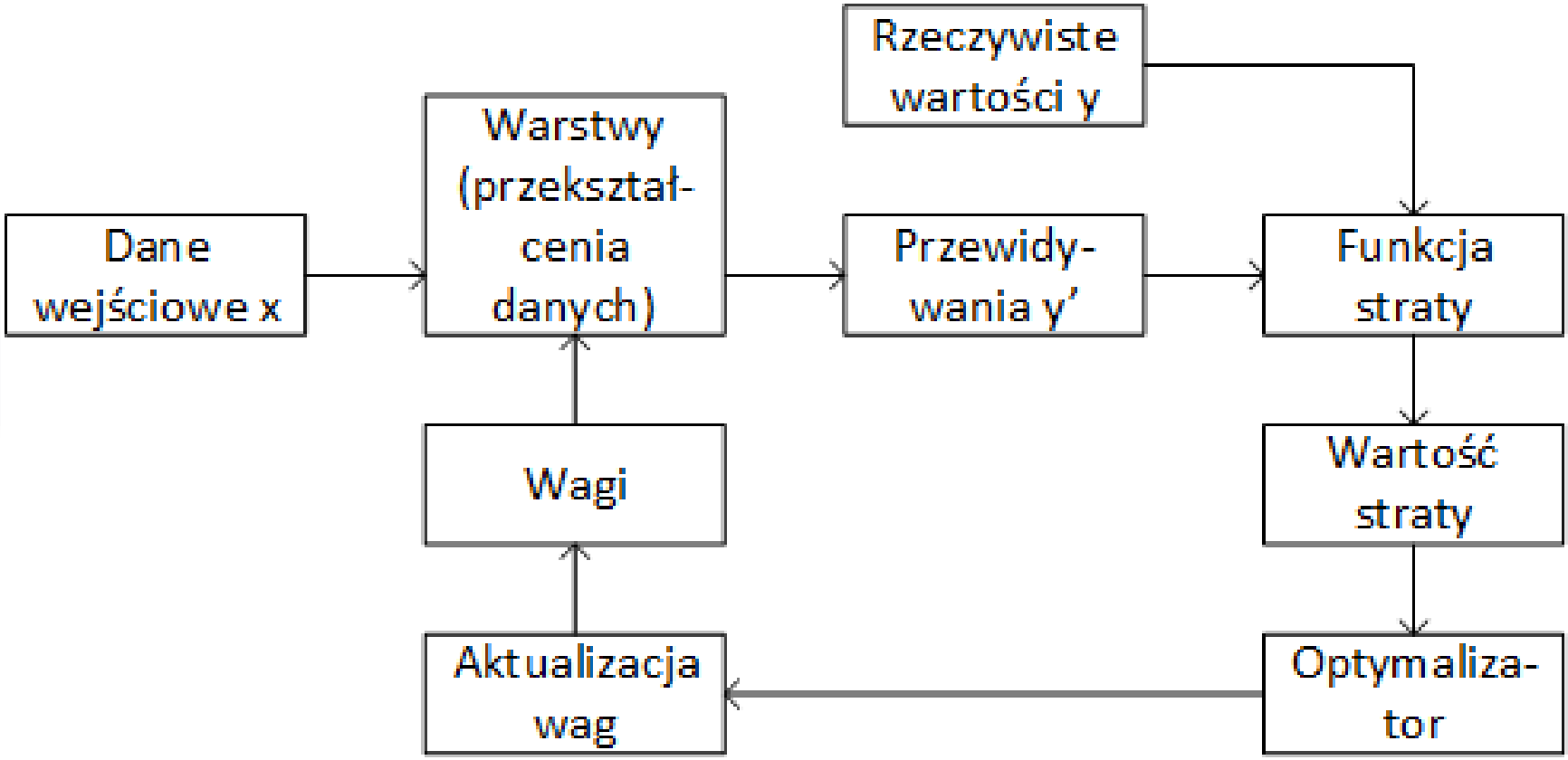
Głębokie sieci neuronowe

- Sztuczna sieć neuronowa (ANN) z wieloma warstwami między warstwą wejściową i wyjściową
- Składniki
  - Neurony
  - Synapsy
  - Wagi
  - Błędy (bias)
  - Funkcje
- Sprzęt: głównie procesory graficzne (GPU), ale też CPU

Ogólna budowa - struktura

- Warstwy: połączone w model/sieć
- Dane wejściowe
- Odpowiadające cele
- Funkcja straty: określa sygnał sprzężenia zwrotnego używany do uczenia się
- Optymalizator: określa, w jaki sposób przebiega nauka

Uproszczona struktura (F. Chollet, Praca z językiem Python i biblioteką Keras, Helion 2019)



Szczegóły

- Funkcja straty (funkcja celu)
- Funkcja ta definiuje sposób pomiaru wydajności sieci podczas przetwarzania uczącego (treningowego) zbioru danych, a więc pozwala na dostrajanie parametrów sieci we właściwym kierunku
- Jej wartość jest minimalizowana w procesie uczenia

Szczegóły (2)

- Optymalizator
- Mechanizm dostrajania sieci na podstawie danych zwracanych przez funkcję straty
- Implementuje określony wariant algorytmu stochastycznego spadku wzdłuż gradientu

Szczegóły (3)

- Metryki monitorowania podczas trenowania i uczenia
- Metryką taką jest m.in. dokładność, np. część obrazów, która została właściwie sklasyfikowana. Inna miara to pole pod wykresem krzywej ROC

Uczenie

- Oznacza znalezienie kombinacji parametrów modelu minimalizującej funkcję straty dla danego zestawu próbek danych treningowych i odpowiadających im etykiet
- Nauka polega na losowaniu zestawów próbek wraz z ich etykietami i obliczaniu gradientu parametrów sieci z uwzględnieniem straty wynikłej podczas przetwarzania tego zbioru próbek. Parametry sieci są następnie nieco poprawiane – ich wartości są zmieniane w kierunku przeciwnym do kierunku gradientu.

Uczenie

- Proces ten jest możliwy, ponieważ sieci neuronowe są łańcuchami różniczkowalnych operacji tensorowych, a także dzięki temu, że możliwe jest stosowanie reguły łańcuchowej różniczkowania w celu określenia funkcji gradientu, przypisującej bieżące parametry i bieżącą próbkę danych do wartości gradientu.

Funkcje aktywacji

- Przypomnijmy, że funkcja aktywacji węzła definiuje wyjście tego węzła, biorąc pod uwagę dane wejściowe lub zestaw danych wejściowych
- Przykłady:
  - Tożsamość
  - Krok binarny (funkcja progowa)
  - Logistyczna
  - Tangens hiperboliczny
  - ReLU
  - Gaussa
  - Softmax

Zbiory

- Uczący (treningowy)
  Służy do trenowania sieci
- Walidacyjny
  Służy do oceny pracy modelu, dostrajania jego hiperparametrów
- Testowy
  Służy do ostatecznej weryfikacji (dane te nie mogą służyć do budowy modelu)

Techniki ewaluacji

- Prosta walidacja na odłożonych danych
- *k*-krotna walidacja krzyżowa
- *k*-krotna walidacja z losowaniem

k-krotna walidacja krzyżowa

- Zbiór danych jest losowo dzielony na *k* części o równej wielkości
- Spośród *k* części pojedyncza część jest zachowywana jako dane walidacyjne do testowania modelu, a pozostałe *k* - 1 części są używane jako dane uczące
- Proces walidacji krzyżowej jest powtarzany k-krotnie, przy czym każda z *k* próbek jest używana dokładnie jeden raz jako dane walidacyjne
- *k* wyników można następnie uśrednić w celu uzyskania pojedynczego oszacowania

Przygotowanie danych

- Zebranie danych
- Eksploracja i profilowanie danych
- Formatowanie danych (wektoryzacja, normalizacja, obsługa brakujących wartości – np. zerami)
- Inżynieria cech (być może nie warto stosować skomplikowanych sieci, jeżeli potrafimy z modelu wyciągnąć informacje w sposób prostszy, np. godzina na zegarze może być odczytana bez stosowania CNN)
- Podział danych na zbiory uczące i ewaluacyjne

Nadmierne dopasowanie

- Odnosi się do modelu, który zbyt dobrze opisuje dane treningowe.
- Ma miejsce, gdy model uczy się szczegółów i szumu w danych uczących do tego stopnia, że ma to negatywny wpływ na wydajność modelu na nowych danych
- Objawia się, gdy np. wydajność modelu podczas przetwarzania odłożonego na bok walidacyjnego zbioru danych po kilku epokach osiąga wartość szczytową, a następnie ulega degradacji

Słabe dopasowanie

- Na początku optymalizacja i uogólnianie są skorelowane – im mniejsza strata na danych treningowych, tym mniejsza strata na danych testowych
- Po kilku iteracjach algorytmu uczącego uogólnianie przestaje ulegać poprawie

Sposoby zapobiegania

- Redukcja rozmiaru sieci
  Model dysponujący większą liczbą parametrów charakteryzuje się większą pojemnością pamięci, a więc może łatwiej uczyć się doskonałego mapowania danych, przypominającego swym działaniem słownik, bez żadnej zdolności uogólniania
- Regularyzacja wag (koszt jest dodawany proporcjonalnie do wartości bezwzględnej współczynników wag/ich kwadratów)
- Porzucanie (wstawianie zer w losowo wybranych miejscach warstw)

Funkcje aktywacji ostatniej warstwy i straty (Keras)

- Klasyfikacja binarna: sigmoid (af), binary_crossentropy (lf)
- Klasyfikacja jednoetykietowa: softmax (af), categorical_crossentropy (lf)
- Regresja: brak, sigmoid (af), mse, binary_crossentropy (lf)

Ogólne podejście

- Zdefiniuj problem
- Określ miarę sukcesu
- Wybierz technikę walidacji
- Zwektoryzuj dane
- Opracuj model
- Stopniowo usprawniaj architekturę modelu poprzez dostrajanie hiperparametrów i dodawanie regularyzacji

Wybrane biblioteki i pakiety

- Flux
- Keras
- Matlab + Deep Learning Toolbox
- Apache MXNet
- PlaidML
- PyTorch
- TensorFlow
- Wolfram Mathematica

Pytania?

Dziękuję za uwagę!

POLITECHNIKA LUBELSKA
WYDZIAŁ ELEKTROTECHNIKI I INFORMATYKI
INFORMATYKA

Zintegrowany Program Rozwoju Politechniki Lubelskiej - część druga

PODSTAWY SZTUCZNEJ INTELIGENCJI W JĘZYKU PYTHON

Głębokie splotowe sieci neuronowe

Paweł Karczmarek

Fundusze Europejskie Wiedza Edukacja Rozwój
Rzeczpospolita Polska
Unia Europejska Europejski Fundusz Społeczny

Agenda

- Ogólna idea
- Rodzaje warstw
- Zastosowania

Sieci splotowe

- Nazywane też konwolucyjnymi
- Twórca: Yann LeCun (1988, LeNet)

Operacja splotu w analizie obrazów

$$c[m,n] = (a * h)[m,n]$$
$$= \sum_j \sum_k h[j,k]a[m-j,n-k]$$

Przykład:

	1	2	3			1	0	1
x	4	5	6	*	h	0	1	0
	7	8	9			1	0	1

	1	2	4	2	3
	4	6	12	8	6
= y	8	14	25	16	12
	4	12	18	14	6
	7	8	16	8	9

Źródło:

https://pl.wikipedia.org/wiki/Splot_(analiza_matematyczna)

Sieci konwolucyjne

- Splotowe
- CNN (Convolutional NN)
- CovNet
- SIANN (shift invariant or space invariant artificial neural networks – z uwagi na ich architekturę współdzielonych wag i charakterystykę niezmienności translacji)

Określenie

- Nazwa wskazuje, że sieć wykorzystuje operację matematyczną zwaną splotem
- Sieci splotowe to wyspecjalizowany rodzaj sieci neuronowych, w których w co najmniej jednej warstwie stosuje się splot zamiast ogólnego mnożenia macierzy
- Splot (ogólnie) definiuje się jako całkę: Niech f i g będą całkowalne w sensie Lebesgue'a na całej prostej rzeczywistej. Splot definiuje się jako

$$(f * g)(t) = \int_{-\infty}^{\infty} f(t-\tau)g(\tau)d\tau$$

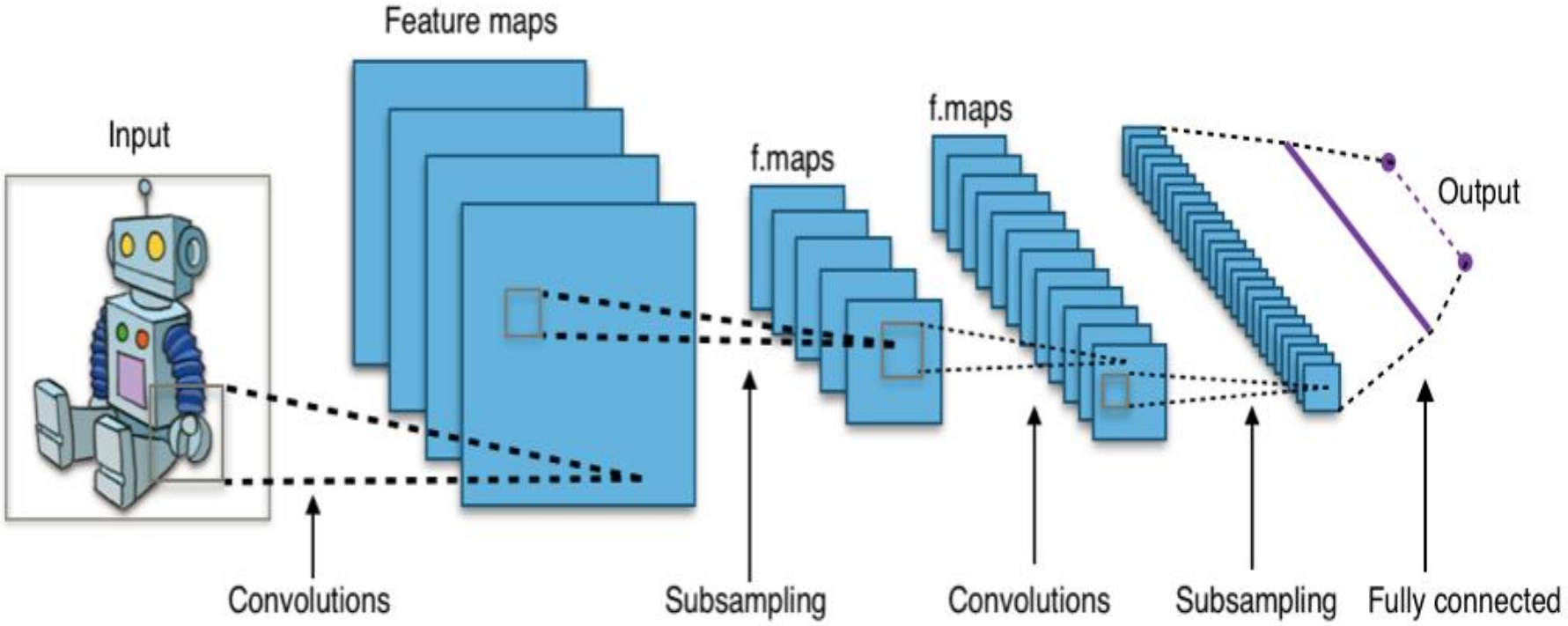
Podobieństwo do MLP

- CNN to uregulowane wersje MLP
- MLP zwykle oznaczają w pełni połączone sieci: Każdy neuron w jednej warstwie jest połączony ze wszystkimi neuronami w następnej warstwie
- Sprawia to, że są podatne na nadmierne dopasowanie danych
- Typowe sposoby regularyzacji obejmują dodanie jakiejś formy pomiaru wielkości wag do funkcji straty

Architektura

- Warstwa wejściowa i wyjściowa
- Wiele ukrytych warstw
- Ukryte warstwy: seria splotowych warstw (stosuje się mnożenie lub iloczyn skalarny)
- Funkcja aktywacji: Zwykle warstwa ReLU, po której następują dodatkowe konwolucje: pooling, w pełni połączone warstwy, warstwy normalizacji (nazywane warstwami ukrytymi, ponieważ ich wejścia i wyjścia są maskowane przez funkcję aktywacji i końcowy splot)

Architektura



Źródło:

https://en.wikipedia.org/wiki/Convolutional_neural_network#/media/File:Typical_cnn.png - Aphex34 - licencja: CC BY-SA 4.0

Warstwa wejściowa

- Reprezentuje obraz wejściowy do sieci.
- W przypadku kolorowych zdjęć używa się trzech kanałów wejściowych RGB
- W przypadku skali szarości jest jeden kanał

Warstwa konwolucyjna (splotowa)

- Zawiera wyuczone filtry (kernele), które wyodrębniają cechy odróżniające od siebie różne obrazy
- Wartości w filtrach są dobierane i optymalizowane podczas trenowania sieci tak, aby minimalizować błąd sieci przy rozwiązywaniu zdefiniowanego problemu
- Filtry są współdzielone na całym zdjęciu. Zatem wagi dobierane są do filtra, który jest następnie przesuwany po całym zdjęciu

Warstwa konwolucyjna (splotowa)

- Nawet jeżeli dodanych jest np. n filtrów (3×3), to ostatecznie będziemy dobierać 9*n* wag
- W porównaniu z liczbą wag przy sieci MLP, gdzie mogło być ich nawet około miliarda, jest to zasadnicza różnica.
- Ponadto rozmiar zdjęcia nie wpływa na liczbę wag w warstwie konwolucyjnej

Hiperparametry warstwy konwolucyjnej

- **Rozmiar jądra (rozmiar filtra)** - wymiar przesuwanego okna nad wejściem.
- Ma znaczny wpływ na zadanie klasyfikacji obrazów.
- Na przykład małe jądra są w stanie wydobyć z danych wejściowych znacznie większą ilość informacji zawierających wysoce lokalne funkcje
- Mniejszy rozmiar jądra prowadzi również do mniejszego zmniejszenia wymiarów warstw, co pozwala na głębszą architekturę

Hiperparametry warstwy konwolucyjnej

- Duży rozmiar jądra wyodrębnia mniej informacji, co prowadzi do szybszego zmniejszenia wymiarów warstw, czego skutkiem może być gorsza wydajność
- Można jednak zyskać lepszą generalizację problemu - duże jądra lepiej nadają się do wyodrębniania większych elementów
- Ostatecznie wybór odpowiedniego rozmiaru jądra powinien zależeć od zadania i zestawu danych, mimo to w ogólności mniejsze rozmiary jądra prowadzą do lepszej wydajności zadania klasyfikacji obrazów

Hiperparametry warstwy konwolucyjnej

- **Wypełnienie – padding**
- Określa sposób obsługi obramowania próbki.
- Umożliwia to otrzymanie rozmiaru wyjścia takiego samego jak rozmiar wejścia. Osiąga się to kosztem dodania dodatkowych (sztucznych) wag na krawędziach (najczęściej z wartością zero) (przy założeniu, że strides jest przesunięciem o jeden).
- Wartości zerowe są wydajne, ponieważ zapewniają prostotę i wydajność obliczeniową. Przykład: AlexNet.

Hiperparametry warstwy konwolucyjnej

- **Krok – stride**
- Definiuje, o ile pikseli jądro powinno zostać przesunięte. Innymi słowy jest to krok przesunięcia okna filtra.
- Najczęściej używa się kroku wynoszącego 1 dla warstw splotowych, tzn. iloczyn skalarny jest wykonywany w oknie wejściowym o zadanym rozmiarze, a następnie jest przesuwany o jeden piksel dla każdej kolejnej operacji.

Hiperparametry warstwy konwolucyjnej

- **Funkcje aktywacji**
- ReLU (nieliniowość)
- Softmax – zapewnia sumowanie wyjścia warstwy do 1, co umożliwia określenie prawdopodobieństwa przynależności np. obrazu do klasy.

Warstwy typu pooling (*łączenie*)

- Celem tej warstwy jest stopniowe zmniejszanie rozmiaru obrazu, prowadzące do zmniejszenia parametrów do wytrenowania.
- Przykładowe „poolingi”:
  - Maksimum z sąsiednich pikseli
  - Średnia z sąsiednich pikseli

Warstwy porzucenia (dropout)

- Przypomnijmy, że jest to jeden ze sposobów unikania przetrenowania sieci.
- Polega na losowym ustawieniu wychodzących krawędzi ukrytych jednostek (neuronów tworzących ukryte warstwy) na 0 przy każdej aktualizacji fazy uczenia.
- Metoda ta jest bardzo efektywna, ponieważ w każdym przejściu losowo wyłączane są połączenia.

Warstwa spłaszczająca (flatten)

- Przekształca wielowymiarową warstwę w sieci w jednowymiarowy wektor.
- Ma to na celu dopasowanie danych wejściowych w pełni połączonej warstwy do klasyfikacji.

Zastosowania

- Rozpoznawanie obrazów
- Analizy wideo
- NLP
- Detekcja anomalii
- Opracowanie leków
- Szacowanie ryzyka utraty zdrowia
- Gry
- Przewidywanie szeregów czasowych
- …

Wybrane architektury

- LeNet-5
- AlexNet
- GoogLeNet
- ResNet

Źródła wykładu i przykłady

- M. Mamczur, https://miroslawmamczur.pl/jak-dzialaja-konwolucyjne-sieci-neuronowe-cnn/
- F. Chollet, Deep learning. Praca z językiem Python i biblioteką Keras, Helion, Gliwice 2019
- https://www.tensorflow.org/tutorials/images/cnn?hl=en
- https://towardsdatascience.com/building-a-convolutional-neural-network-cnn-in-keras-329fbbadc5f5
- https://keras.io/api/layers/convolution_layers/

Pytania?

Dziękuję za uwagę!

POLITECHNIKA LUBELSKA
WYDZIAŁ ELEKTROTECHNIKI I INFORMATYKI
INFORMATYKA

Zintegrowany Program Rozwoju Politechniki Lubelskiej - część druga

PODSTAWY SZTUCZNEJ INTELIGENCJI W JĘZYKU PYTHON

Głębokie rekurencyjne sieci neuronowe

Paweł Karczmarek

Fundusze Europejskie Wiedza Edukacja Rozwój
Rzeczpospolita Polska
Unia Europejska Europejski Fundusz Społeczny

Agenda

- Idea
- Rodzaje połączeń
- Komórka LSTM
- Komórka GRU
- Zastosowania

Neuron rekurencyjny

- Przypomnijmy, że w przypadku neuronu rekurencyjnego, w każdym takcie (czyli kroku czasowym, time step), zwanym też ramką (frame), neuron rekurencyjny (recurrent neuron) oprócz danych wejściowych otrzymuje również własne wyniki z poprzedniego taktu

Sekwencje danych

- Sieci rekurencyjne mogą służyć do analizy sekwencji danych. Możliwe sytuacje to:
  - Wiele do jednego
  - Jeden do wielu
  - Wiele do wielu

Wiele do wielu

- Dane wejściowe stanowią sekwencję
- Dane wyjściowe mają postać wektora lub skalara, a nie sekwencji
- Np. wejściem jest recenzja filmu, a wyjściem etykieta klasy (np. recenzentowi film się nie podobał)

Jeden do wielu

- Dane wejściowe nie są sekwencyjne
- Dane wyjściowe są sekwencyjne
- Np. podpisywanie obrazów: jeden obraz jest opisany przez sekwencję zdań

Wiele do wielu

- Dane wejściowe i wyjściowe są sekwencyjne
- Przykłady:
  - (Synchronizacja) Klasyfikacja filmów, gdzie każda klatka filmu jest oznaczona
  - (Opóźnienie) Tłumaczenie z języka A na język B – potrzebne jest do wczytania całe zdanie

Komórki pamięci

- Ponieważ wyjście neuronu rekurencyjnego w takcie t stanowi funkcję wszystkich danych wejściowych z poprzednich ramek czasowym, dlatego można stwierdzić, że sieć rekurencyjna zawiera pamięć.
- Fragment takiej sieci nazywa się komórką (komórką pamięci).
- Neuron lub warstwa są przykładem komórki podstawowej.

Jednostki LSTM

- Long Short-Term Memory
- Twórcy Sepp Hochreiter i Jürgen Schmidhuber (1997)
- Jej głównym składnikiem jest komórka pamięci, która w istocie zastępuje lub reprezentuje warstwę ukrytą typowej sieci rekurencyjnej
- W każdej komórce występuje krawędź rekurencji mająca pożądaną wagę w=1, która służy do przezwyciężania problemów z zanikającymi albo eksplodującymi gradientami
- Wartości powiązane z tą krawędzią rekurencji nazywa się stanem komórki

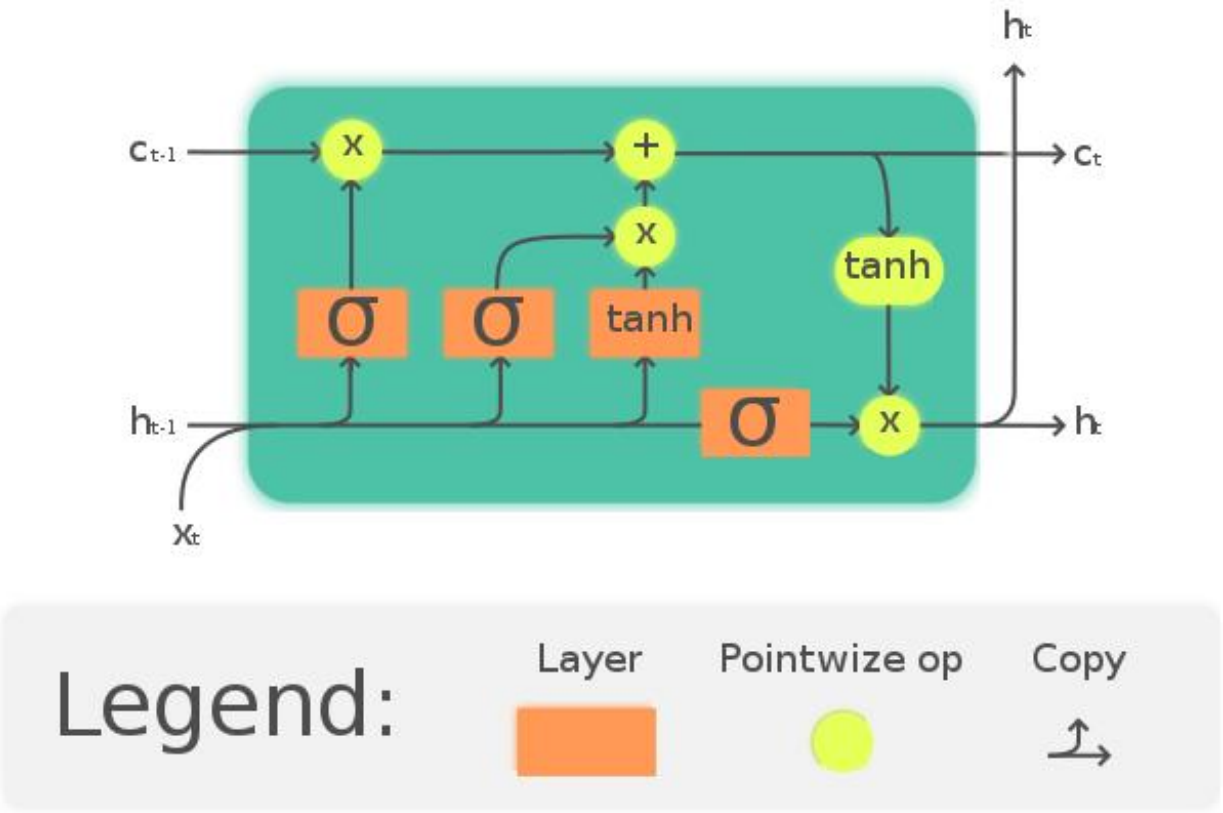
Problem zanikającego gradientu

- Pojawia się podczas uczenia sztucznych sieci neuronowych za pomocą metod opartych na uczeniu gradientowym i propagacji wstecznej.
- Każda z wag sieci neuronowych otrzymuje aktualizację proporcjonalną do pochodnej cząstkowej funkcji błędu względem wagi bieżącej w każdej iteracji uczenia.
- W niektórych przypadkach nachylenie jest znikomo małe, skutecznie uniemożliwiając zmianę wartości wagi. W najgorszym przypadku może to całkowicie uniemożliwić dalsze trenowanie sieci neuronowej.

Problem zanikającego gradientu (2)

- Np. tradycyjne funkcje aktywacji (np. tangens hiperboliczny),mają gradienty w zakresie (0, 1), a propagacja wsteczna oblicza gradienty zgodnie z regułą łańcucha.
- Powoduje to pomnożenie *n* małych liczb w celu obliczenia gradientów wczesnych warstw w sieci *n*-warstwowej, co oznacza, że gradient (sygnał błędu) zmniejsza się wykładniczo wraz z *n*, a wczesne warstwy uczą się bardzo powoli

Jednostki LSTM



Źródło: C. Chevalier, **LARNN: Linear Attention Recurrent Neural Network** https://arxiv.org/abs/1808.05578 & Wikipedia

Rodzaje bramek

- Zapominająca
- Wejściowa
- Wyjściowa

Bramka zapominająca

- Forget gate
- Umożliwia wyzerowanie komórki pamięci bez ryzyka jej nieskończonego wzrostu.
- Decyduje, które informacje mają być przekazywane dalej, a które mają być blokowane.
- Nie była częścią pierwotnej komórki LSTM, została dodana kilka lat później w celu usprawnienia modelu.

Bramka wejściowa i wartość kandydująca

- Input gate, candidate value
- Odpowiadają za aktualizowanie stanu komórki.

Bramka wyjściowa

- Output gate
- Określa sposób aktualizowania wag jednostek ukrytych.

Równania

$$f_t = \sigma_g\left(W_f x_t + U_f h_{t-1} b_f\right)$$
$$i_t = \sigma_g(W_i x_t + U_i h_{t-1} b_i)$$
$$\tilde{c}_t = \sigma_c(W_c x_t + U_c h_{t-1} b_c) o_t$$
$$= \sigma_g(W_o x_t + U_o h_{t-1} b_o)$$

$$c_t = f_t \circ c_{t-1} + i_t \circ \tilde{c}_t$$
$$h_t = o_t \circ \sigma_h(c_t)$$

$$c_0 = 0$$
$$h_0 = 0$$

Oznaczenia

- W_q i U_q - macierze zawierające wagi wejściowych i rekurencyjnych połączeń, indeks *q* może oznaczać bramkę wejściową *i*, wyjściową *o*, bramkę zapominającą *f* lub komórkę pamięci *c* w zależności od wyznaczonej aktywacji.
- $\circ$ - produkt Hadamarda – wynikiem są iloczyny wartości z odpowiadającymi sobie indeksami

Oznaczenia

- $x_t \in \mathbb{R}^d$ - wektor wejściowy do jednostki LSTM
- $f_t \in \mathbb{R}^h$ - wektor aktywacji bramki zapominającej
- $i_t \in \mathbb{R}^h$ - wektor aktywacji bramki wejściowej
- $o_t \in \mathbb{R}^h$ - wektor aktywacji bramki wyjściowej
- $h_t \in \mathbb{R}^h$ - ukryty wektor stanu/wektor wyjściowy jednostki LSTM
- $\tilde{c}_t \in \mathbb{R}^h$ - wektor aktywacji wejścia komórki
- $c_t \in \mathbb{R}^h$ - wektor stanu komórki
- $W \in \mathbb{R}^{h \times d}, U \in \mathbb{R}^{h \times h}, b \in \mathbb{R}^h$ - macierze wag i wektor parametrów biasów, które trzeba wytrenować
- d i h – liczby wejściowych cech i ukrytych jednostek

Oznaczenia

- Funkcje aktywujące:
  - σ_g - funkcja sigmoidalna
  - σ_h - tangens hiperboliczny, ew. tożsamość – dla tzw. wariantu LSTM z wizjerem (peephole LSTM)
- Inny wariant: Konwolucyjna sieć LSTM z wizjerem

Uczenie

- RNN korzystająca z jednostek LSTM może być trenowana w sposób nadzorowany, na zestawie sekwencji treningowych, przy użyciu algorytmu optymalizacji, takiego jak metoda gradientu prostego w połączeniu z propagacją wsteczną, aby obliczyć gradienty potrzebne podczas procesu optymalizacji, co jest potrzebne do zmiany każdej wagi sieci LSTM proporcjonalnie do pochodnej błędu (w warstwie wyjściowej sieci LSTM) w odniesieniu do odpowiedniej wagi.
- W przypadku jednostek LSTM, gdy wartości błędów są wstecznie propagowane z warstwy wyjściowej, błąd pozostaje w komórce jednostki LSTM. Ta „karuzela błędów” nieprzerwanie przesyła błąd z powrotem do każdej z bramek jednostki LSTM, dopóki nie nauczą się one odcinać wartości.

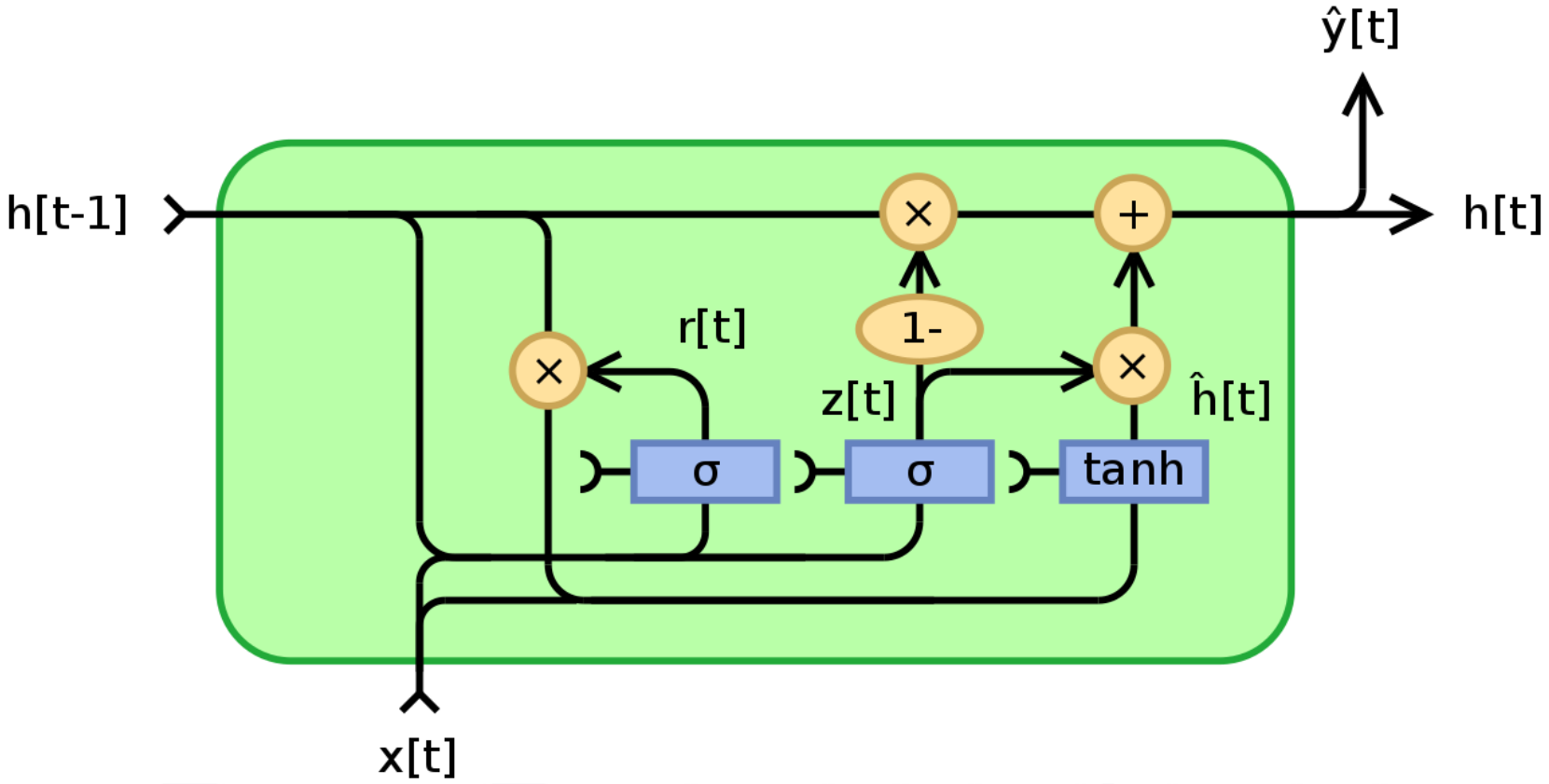
Zastosowania LSTM

- Przewidywanie sentymentów (sentiment analysis)
- NLP
- Szeregi czasowe – predykcja i wyszukiwanie anomalii
- Rozpoznawanie mowy
- Rozpoznawanie tekstu pisanego odręcznie
- Produkcja leków
- Rozpoznawanie ludzkich ruchów
- Komponowanie muzyki
- Przewidywanie korków
- …

Gated Recurrent Unit (GRU)

- Bramkowa jednostka rekurencyjna
- Ma prostszą budowę
- Wydajniejsza obliczeniowo
- W niektórych zastosowaniach porównywalna pod względem skuteczności z LSTM (np. modelowanie muzyki polifonicznej)
- Inne zastosowania: NLP, rozpoznawanie mowy
- Cecha: brak bramki wyjściowej

GRU



- Źródło: https://en.wikipedia.org/wiki/Gated_recurrent_unit#/media/File:Gated_Recurrent_Unit,_base_type.svg
- Autor: Jeblad

Przykładowe implementacje

- https://towardsdatascience.com/implementation-of-rnn-lstm-and-gru-a4250bf6c090
- https://towardsdatascience.com/understanding-lstm-and-its-quick-implementation-in-keras-for-sentiment-analysis-af410fd85b47
- F. Chollet, Deel Learning. Praca z językiem Python i biblioteką Kera, Helion, Gliwice 2019
- A. Géron, Uczenie maszynowe z użyciem Scikit-Learn i TensorFlow, Helion, Gliwice 2018
- S. Raschka, V. Mirjalili, Python. Machine learning i deep learning, Helion, Gliwice 2021

Pytania?

Dziękuję za uwagę!

POLITECHNIKA LUBELSKA
WYDZIAŁ ELEKTROTECHNIKI I INFORMATYKI
INFORMATYKA

Zintegrowany Program Rozwoju Politechniki Lubelskiej - część druga

PODSTAWY SZTUCZNEJ INTELIGENCJI W JĘZYKU PYTHON

Autoenkodery

Paweł Karczmarek

Fundusze Europejskie Wiedza Edukacja Rozwój
Rzeczpospolita Polska
Unia Europejska Europejski Fundusz Społeczny

Agenda

- Idea
- Architektura
- Rodzaje autoenkoderów
- Zastosowania

Autoenkoder (ew. Autokoder)

- Używane do uczenia się wydajnego kodowania danych nieoznakowanych (w uczeniu nienadzorowanym)
- Kodowanie jest sprawdzane i udoskonalane przez próbę ponownego wygenerowania danych wejściowych z kodowania
- A. uczy się reprezentacji (kodowania) zbioru danych, zwykle w celu redukcji wymiarowości, poprzez uczenie sieci ignorowania nieistotnych danych („szumu”).
- Celem niektórych wariantów jest zmuszenie wyuczonych reprezentacji do przyjęcia użytecznych właściwości

Autoenkoder

- Autoenkoder przekształca dane wejściowe w efektywną reprezentację wewnętrzną, a następnie umieszcza na wyjściu wynik przypominający informacje otrzymane na wejściu.

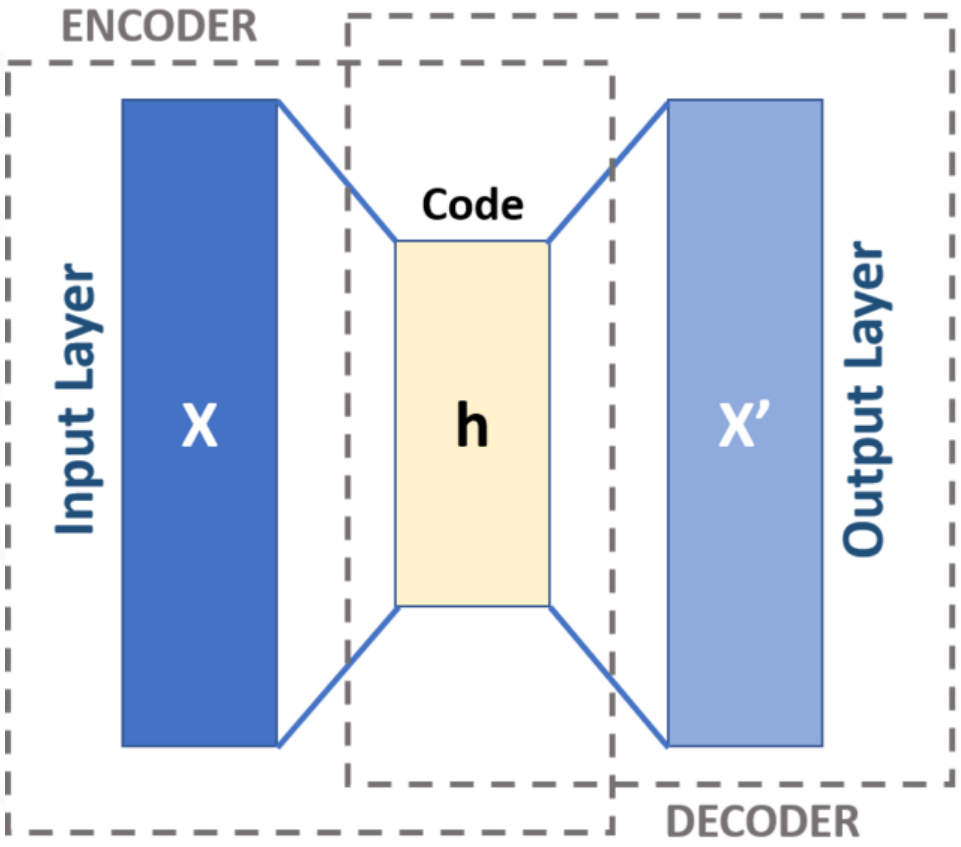
Autoenkoder

- Składa się z dwóch części:
- Koder (sieć rozpoznawnia) – przekształcający dane wejściowe do postaci reprezentacji wewnętrznej
- Dekoder (sieć generatywna) – którego zadaniem jest konwersja reprezentacji wewnętrznej na dane wyjściowe

Podobieństwo do MLP

- Autoenkoder ma strukturę podobną do MLP
- Różnica polega na tym, że liczba neuronów na wyjściu musi być taka sama, jak na wejściu
- Dane wyjściowe są nazywane rekonstrukcjami, ponieważ dekoder stara się zrekonstruować dane wejściowe
- Funkcja kosztu zawiera stratę rekonstrukcji, karzącą modę w sytuacji, gdy rekonstrukcje różnią się od danych wejściowych

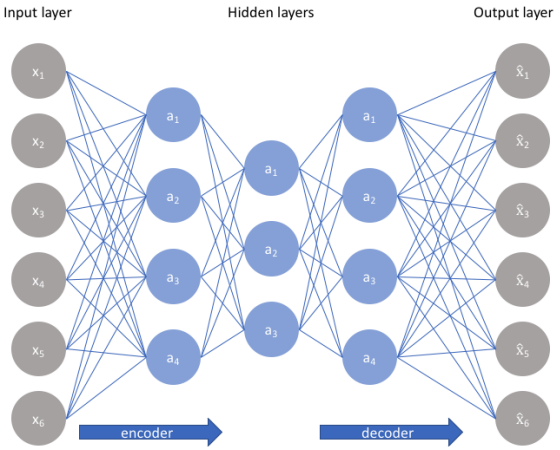
Podobieństwo do MLP



- Źródło: https://commons.wikimedia.org/wiki/File:Autoencoder_schema.png
- Autorka: Michela Massi

Autoenkoder niedopełniony

- Undercomplete autoencoder
- Liczba neuronów w warstwach ukrytych jest mniejsza niż w warstwie wejściowej.
- Wymusza to, że nie może po prostu skopiować danych i przekazać ich dalej, ale szuka sposobu by odtworzyć dane na wyjściu



Źródło rysunku: https://www.jeremyjordan.me/autoencoders/

Autoenkoder przepełniony

- Overcompleted autoencoder
- Sytuacja, w której warstwy ukryte są większe lub równe warstwie wejściowej lub jeśli ukryte jednostki mają wystarczającą pojemność
- Autokoder może potencjalnie nauczyć się funkcji tożsamości i stać się bezużyteczny.
- Jednak wyniki eksperymentalne wykazały, że przepełnione autokodery mogą nadal uczyć się przydatnych funkcji (przypadek tzw. regularyzowanych autoenkoderów)

Autoenkoder stosowy

- Inaczej głęboki
- Posiada wiele warstw ukrytych
- Kolejne warstwy pozwalają autoenkoderowi uczyć się bardziej skomplikowanych kodowań
- Trzeba pamiętać, aby nie przesadzić z „głębokością”
- Zbyt głęboki autoenkoder może mieć problem z nauczeniem przydatnej reprezentacji danych, co prowadzi do trudności z generalizacją przewidywań dla nowych próbek

Wizualizacja cech: wybrane metody

- Przyjrzenie się neuronowi i znalezienie próbek uczących, które najbardziej go uaktywniają
- Dla każdego neuronu w pierwszej warstwie ukrytej można utworzyć obraz, na którym poziom szarości każdego piksela odpowiada wadze połączenia z danym pikselem
- Wczytanie losowego obrazu, zmierzenie aktywności wybranego neuronu i przeprowadzenie wstecznej propagacji modyfikującej ten obraz tak, aby pobudzał on jeszcze bardziej obserwowany neuron – obraz będzie się stopniowo przekształcał w formę maksymalnie pobudzającą dany neuron

Autoenkodery odszumiające

- Polegają na tym, że autoenkoder jest zmuszany do poznawania przydatnych cech poprzez dodawanie szumu do danych wejściowych i uczenie go odzyskiwania pierwotnych, niezaszumionych informacji.
- Zaszumienie można generować np. szumem gaussowskim albo wyłączać losowo wybrane wejścia za pomocą metody porzucania (dropout)

Autoenkodery rzadkie (sparse)

- Modyfikowane przez dodanie odpowiedniego członu do funkcji kosztu
- W ten sposób autoenkoder zostaje zmuszony do zmniejszenia liczby aktywnych neuronów w warstwie kodowania
- Na zbyt aktywne neurony zostaje nałożona kara poprzez dodanie straty rzadkości (sparsity loss) do funkcji kosztu

Autoenkodery wariacyjne

- Są one autoenkoderami probabilistycznymi, tzn. generują częściowo losowe wyniki, nawet po wyuczeniu modelu
- Stanowią klasę autoenkoderów generatywnych, tzn. tworzą nowe próbki przypominające dane zawarte w zbiorze uczącym
- Podczas uczenia koder generuje nie bezpośrednie kodowanie danej próbki wejściowej, ale uśrednione kodowanie i odchylenie standardowe. Rzeczywiste kodowanie jest następnie próbkowane losowo z rozkładu gaussowskiego przy użyciu tej średniej i odchylenia

Autoenkodery kurczliwe

Zostają ograniczone podczas procesu uczenia tak, aby pochodne kodowań miały małe wartości w porównaniu z danymi wejściowymi. Prowadzi to do tego, że dwa podobne przykłady wejściowe mają podobne kodowania.

Stosowe autoenkodery splotowe

Autoenkodery uczące się wydobywać cechy wizualne poprzez rekonstruowanie obrazów przetwarzanych przez warstwy splotowe

Generatywne sieci stochastyczne

Są one uogólnieniem autoenkoderów odszumiających z dodatkiem możliwości generowania danych.

Autoenkodery WTA (winner take all)

Podczas uczenia, na etapie wyliczania aktywacji wszystkich neuronów warstwy kodowania, zostaje zachowanych jedynie *p*% pobudzeń każdego neuronu wobec grupy danych uczących. Pozostałe są zastąpione zerami. W naturalny sposób otrzymuje się tak kodowanie rzadkie.

Autoenkodery antagonistyczne

Polegają na tym, że jedna sieć jest uczona rekonstruowania danych wejściowych, natomiast druga uczy się wyszukiwać dane, których pierwsza sieć nie jest w stanie prawidłowo zrekonstruować. Prowadzi to do tego, że pierwsza z sieci uczy się różnorodnych kodowań.

Zastosowania

- Redukcja wymiarowości
- Detekcja anomalii
- Przetwarzanie obrazów
- Opracowywanie leków
- Predykcja popularności postów w social media
- Tłumaczenie

Przykładowe kody

- https://blog.keras.io/building-autoencoders-in-keras.html
- https://analyticsindiamag.com/guide-to-autoencoders-with-python-code/
- https://machinelearningmastery.com/autoencoder-for-classification/
- https://rubikscode.net/2018/11/26/3-ways-to-implement-autoencoders-with-tensorflow-and-python/
- https://www.tensorflow.org/tutorials/generative/autoencoder
- A. Géron, Uczenie maszynowe z użyciem Scikit-Learn i TensorFlow, Helion, Gliwice 2018

Pytania?

Dziękuję za uwagę!

POLITECHNIKA LUBELSKA
WYDZIAŁ ELEKTROTECHNIKI I INFORMATYKI
INFORMATYKA

Zintegrowany Program Rozwoju Politechniki Lubelskiej - część druga

PODSTAWY SZTUCZNEJ INTELIGENCJI W JĘZYKU PYTHON

GPT-3 - podsumowanie

Paweł Karczmarek

Fundusze Europejskie Wiedza Edukacja Rozwój
Rzeczpospolita Polska
Unia Europejska Europejski Fundusz Społeczny

GPT-3

- Generative Pre-trained Transformer 3
- Generator tekstu, wykorzystujący model DL typu Transformer
- Twórcy: OpenAi (współzałożyciel E. Musk)

Dane uczące

- 60 % Common Crawl
- 22% WebText2
- 8% Books1
- 8% Books2
- 3% Wikipedia

Działanie

- Model języka naturalnego uczy się przewidywać, jakie frazy czy zdania będę najprawdopodobniej występować po danych słowach.
- Uczenie nienadzorowane

Zastosowania

- Odpowiedzi na pytania
- „Czaty” z postaciami historycznymi
- Codex i autokompletowanie kodu
- Tworzenie kodu
- Poradnia medyczna
- Przekształcanie mowy prawniczej na prostszy język mówiony
- Generator chwytów gitarowych
- Generator brakujących grafik
- Dialogi w grach komputerowych
- …

Krytyka

- Błąd algorytmiczny, np. skojarzenie Islamu z terroryzmem
- Ograniczenie dostępu do kodu – tylko Microsoft
- Kwestie sensowności tekstów i plagiatu
- Uczona na publicznych i danych z prawami autorskimi

Pytania?

Dziękuję za uwagę!

POLITECHNIKA LUBELSKA
WYDZIAŁ ELEKTROTECHNIKI I INFORMATYKI

INFORMATYKA

Zintegrowany Program Rozwoju Politechniki Lubelskiej - część druga

Materiały zostały opracowane w ramach projektu
„Zintegrowany Program Rozwoju Politechniki Lubelskiej – część druga”,
umowa nr **POWR.03.05.00-00-Z060/18-00**
w ramach Programu Operacyjnego Wiedza Edukacja Rozwój 2014-2020
współfinansowanego ze środków Europejskiego Funduszu Społecznego

Fundusze Europejskie Wiedza Edukacja Rozwój

Rzeczpospolita Polska

Unia Europejska Europejski Fundusz Społeczny