

Uczenie maszynowe

PRZYGOTOWAŁ MICHAŁ TRACEWICZ

Czym jest uczenie maszynowe?

Uczenie maszynowe (ang. Machine Learning w skrócie ML) - jedna z dziedzin będąca częścią zagadnienia sztucznej inteligencji (ang. Artificial Intelligence w skrócie AI).

Jej zadaniem jest tworzenie modeli, które na podstawie historycznych danych wejściowych (ang. input) i wyjściowych (ang. output) są w stanie utworzyć reguły, które przewidzą wyjście dla nowego, nie znanego wcześniej wejścia.

Definicja według Donalda Michie (1991):

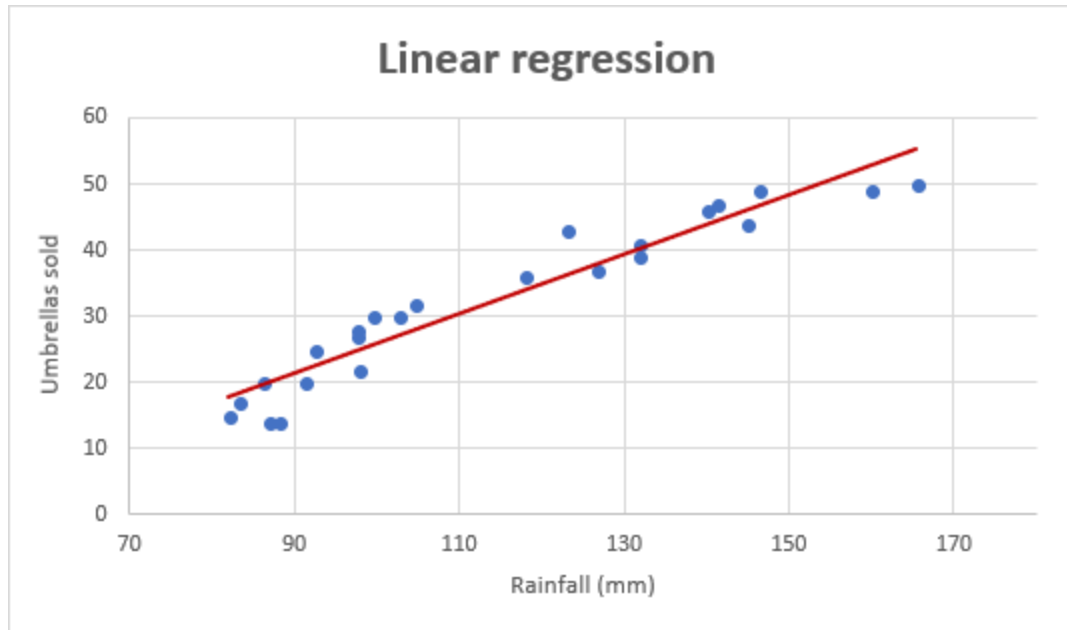
"System uczący się wykorzystuje zewnętrzne dane empiryczne w celu tworzenia i aktualizacji podstaw dla udoskonalonego działania na podobnych danych w przyszłości oraz wyrażania tych podstaw w zrozumiałej i symbolicznej postaci".

(źródło: https://pl.wikipedia.org/wiki/Uczenie_maszynowe)

Uczenie maszynowe jest także czasami nazywane "statystyką stosowaną" (bibliografia 8.)

Podstawowe pojęcia

Regresja liniowa (ang. Linear regression)



Jest to proces dopasowanie funkcji liniowej aby jak najlepiej reprezentowała nasze dane.

W uczeniu maszynowym funkcja ta przyjmuje postać:

$$y' = b + \sum_{i \in [0, N-1]} w_i x_i$$

Gdzie:

y' – predykcja naszego modelu

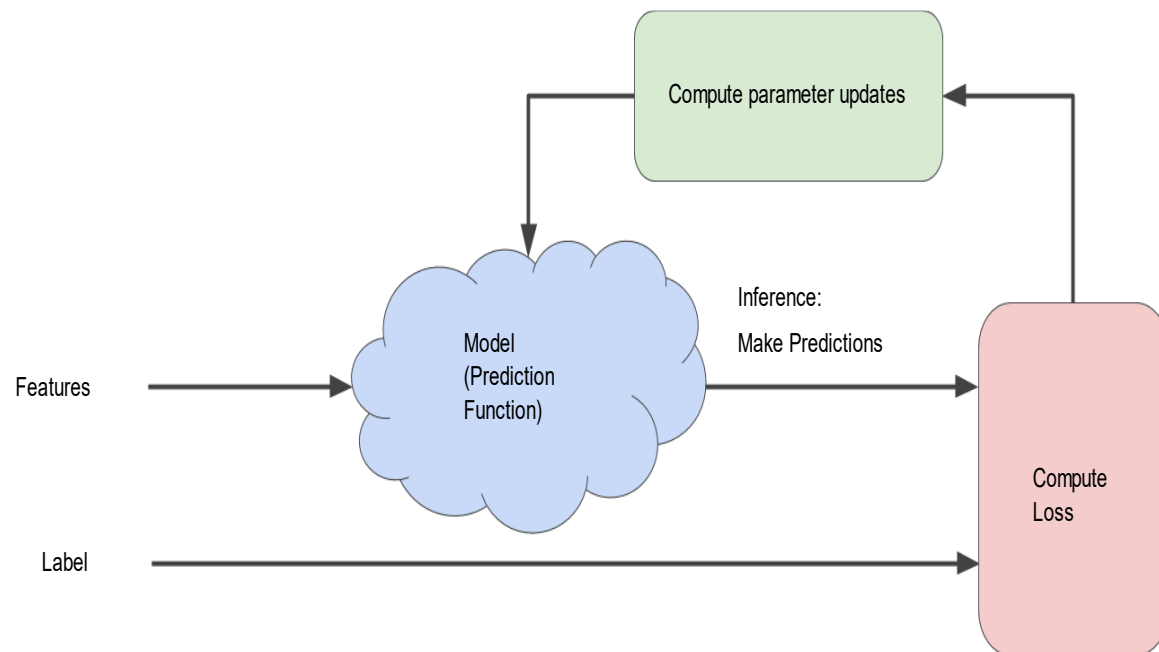
b – stronniczość (ang. bias)

x_i – i -ta dana wejściowa

w_i – i -ta waga

N – liczba danych wejściowych

Proces uczenia



Nasz model zaczyna od losowych wag, następnie zgaduje jakie będą wyniki dla danych wejść. W kolejnym kroku obliczana jest funkcja straty i na jej podstawie wagi zostają poprawione. Proces ten powtarzamy wybraną ilość razy (najczęściej definiujemy ją jako wielokrotność przejrzania całego zbioru danych, jedno takie przejście nazywamy "epoch")

<https://developers.google.com/machine-learning/crash-course/reducing-loss/an-iterative-approach>

Dokładność (ang. accuracy)

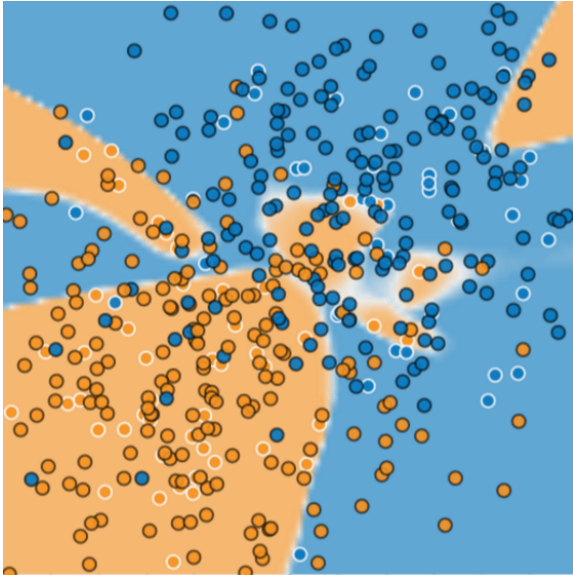
Jedną z podstawowych własności każdego modelu jest dokładność. Jest ona mierzona poprzez podzielenie liczby podanych przez model poprawnych odpowiedzi przez liczbę wszystkich odpowiedzi. Intuicyjnie im większa dokładność tym lepiej, jednak należy na to uważać.

Dla przykładu gdy jakaś choroba ma prawdopodobieństwo wystąpienia u pacjenta 0.001% nasz model może osiągnąć dokładność 99.999% poprzez każdorazowe odpowiadanie, że pacjent nie ma danej choroby.

Dodatkowo bardzo duża dokładność może sygnalizować zjawisko zbytniego dopasowania.

Zbytnie dopasowanie (ang. Overfitting)

Czasami gdy nasz model jest trenowany zbyt długo na tych samych danych może nauczyć się odpowiedzi do naszych danych historycznych "na pamięć". Osiągniemy wtedy na nich wprowadzie wielką dokładność ale nasz model zupełnie nie poradzi sobie na nowym nieznanym dotąd przykładzie



Przykład ilustrujący zbytnie dopasowanie.
Dane na których model ćwiczył mają białą otoczkę,
a nowe dane wejściowe otoczkę czarną.

<https://developers.google.com/machine-learning/crash-course/generalization/peril-of-overfitting>

Podział danych

Jednym z głównych sposobów walki ze zbytnim dopasowaniem jest podzielenie naszych danych.

Istnieją dwa główne podziały:

- Dane treningowe + testowe -> w tym sposobie dzielimy nasze dane na dwie części, dbając o to aby danych treningowych było więcej niż danych testowych np. 70 – 30. W tej metodzie nasz model uczy się używając danych treningowych a następnie przed liczeniem funkcji straty jest testowany na nowych nieznanych wcześniej danych
- Dane treningowe + testowe + walidacyjne -> metoda ta różni się tym, że dodajemy dodatkową trzecią grupę danych, której nie używamy zupełnie w procesie uczenia. Dane, które się w niej znajdują zostają użyte dopiero do testu gdy nasz model jest już nauczony i chcemy sprawdzić jak poradzi sobie zupełnie nowymi danymi.

Aby podziały były skuteczne należy zapewnić przemieszanie danych wejściowych, żeby np. w grupie treningowej nie było wszystkich danych o studentach matematyki, a w testowej o studentach informatyki

Dane wejściowe (ang. features)

Dane wejściowe mogą być pod bardzo różnymi postaciami np. zdjęcie, ramka danych, ciąg liczb. Jednak w większości muszą być takie same dla danego modelu.

To znaczy, że jeżeli nasz model uczył się na zdjęciach rozmiaru 256x256 w zapisie RGB(x3) to nie będzie w stanie obsłużyć nowych danych w postaci zdjęcia monochromatycznych(x1) rozmiaru 512x512

Czyszczenie danych

Ważne jest, żeby nasze dane zostały odpowiednio przygotowane przed rozpoczęciem pracy nad nimi. Przykładowe operacje, które możemy przeprowadzić na naszych danych to:

- Usunięcie wartości skrajnych
- Sprawdzenie brakujących danych
- Zeskalować dane (zmniejsza to koszt komputacji) np. Zamiast zapisu 78 000 000 możemy zapisać 78 jeżeli większość danych ma taki rząd wielkości. Należy jednak pamiętać, że musimy zeskalować każdą wartość wchodzącą tej kategorii.
- Grupowanie danych w grupy i np. zamiast traktować każdą szerokość geograficzną stworzyć przedziały co stopień, co prawdopodobnie zwiększy ilość informacji w naszych danych.

One hot encoding

Jest to sposób kodowania danych, gdzie bierzemy wektor samych zer a następnie zaznaczamy dokładnie jedną jedynkę. Np. $[0, 0, 0, 1, 0, 0]$

Może nam posłużyć np. do wskazania które słowo w zdaniu to podmiot. Innym przykładem może być np. Zaznaczenie, który z trzech kolorów podstawowych dominuje w obrazie zamiast pisanie: "czerwony", "zielony", "niebieski", co byłoby trudne do liczenia dla naszych sieci neuronowych. Możemy mieć natomiast wektor $[0, 1, 0]$ oznaczający, że dominuje kolor zielony.

Dane wyjściowe (ang. labels)

Zależą od typu naszej sieci. Mogą to być dla przykładu:

- Przewidywana cena mieszkania
- Czy na obrazku jest kot/pies/chomik
- Prawdopodobieństwo, że dany email to spam

Funkcja strat (ang. Loss function)

Aby móc poprawić nasze wyniki musimy mieć jakiś sposób oceny tego jak poszło naszemu modelowi. W tym celu liczymy funkcję strat i staramy się ją zminimalizować. Popularne funkcje strat to:

$L1 = \sum_{i \in [0, N-1]} |y_i - y_i'|$, gdzie y_i - wartość oczekiwana, y_i' - wartość przewidziana

$L2 = \sum_{i \in [0, N-1]} (y_i - y_i')^2$, gdzie y_i - wartość oczekiwana, y_i' - wartość przewidziana

Metoda gradientu prostego (ang. Gradient Descent)

Do minimalizowania funkcji strat posłuży nam metoda gradientu prostego, która powie nam w jaką stronę powinniśmy się kierować aby zbliżyć się do lokalnego minimum.

To jak "duże" kroki we wskazaną przez gradient stronę wykona nasz model ustalamy poprzez parametr o nazwie learning rate.

Nasze uczenie powinno kończyć w momencie, w którym nasza funkcja osiągnie minimum (ang. converge).

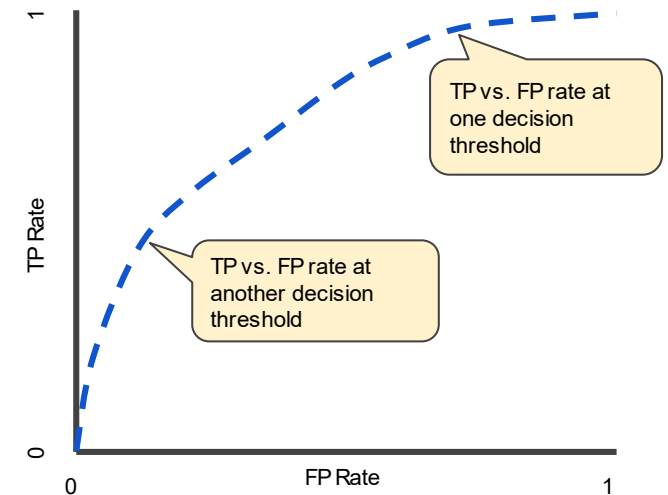
Ustawiając zbyt małego learning rate może spowodować, że nasze obliczenia będą trwać wieki, podczas gdy zbyt duży parametr może spowodować, że nasz model nigdy nie osiągnie lokalnego minimum.

Macierz pomyłek

<p>True Positive (TP) Pasterz krzyczy "WILK!" Wilk podchodzi do wioski Wynik : Pasterz został bohaterem</p>	<p>False Positive (FP) Pasterz krzyczy "WILK!" Wilka nigdzie nie ma Wynik : Pasterz został skrzyczany</p>
<p>False Negative (FN) Pasterz nie krzyczy "WILK!" Wilk podchodzi do wioski Wynik : Wieśniacy stracili owce</p>	<p>True Negative (TN) Pasterz nie krzyczy "WILK!" Wilka nigdzie nie ma Wynik : Wszystko jest w porządku</p>

Mierzenie poprawności naszego modelu

- Precyzja (ang. Precision) = $TP/(TP+FP)$ – jaki procent naszych stwierdzeń, że wynik jest pozytywny był poprawny
- Przywracanie(ang. Recall) = $TP/(TP+FN)$ – jaki procent wszystkich pozytywnych instancji był przez nas skategoryzowany jako pozytywny
- Krzywa ROC to wykres gdzie:
oś y: True Positive Rate (TPR) = $TP/(TP+FN)$
oś x: False Positive Rate (FPR) = $FP/(FP + TN)$
na obu osiach mamy wartości z przedziału $[0, 1]$.
- AOC ("area under the ROC curve") - pole pod krzywą (całka) interpretowane jako prawdopodobieństwo, że losowy przykład pozytywny zostanie oceniony wyżej niż przykład negatywny.



Regularyzacja

Gdy mamy bardzo dużo danych wejściowych ale wiemy, że niektóre z nich są mało istotne albo gdy chcemy zaoszczędzić miejsce na dysku oraz czas procesora, możemy postarać się wprowadzić regularyzację danych w naszym modelu.

Wtedy zamiast minimalizować wyłącznie funkcję strat minimalizujemy jej sumę z funkcją regularyzującą pomnożoną przez parametr λ .

Parametr ten powinien przyjmować wartość z przedziału $[0,1]$.

Wskazuje on jak duży wpływ na wagi ma mieć regularyzacja.

Dwie efektywne funkcje regularyzacji to:

- L1 regularyzacja – redukujemy niektóre wagi do 0.

Wzór: $L_1 = |x_1| + |x_2| + \dots + |x_n|$

- L2 regularyzacja - zachęcamy model do posiadania wag jak najbliższych 0.

Wzór: $L_2 = x_1^2 + x_2^2 + \dots + x_n^2$

Regresja logistyczna (ang. logistic regression)

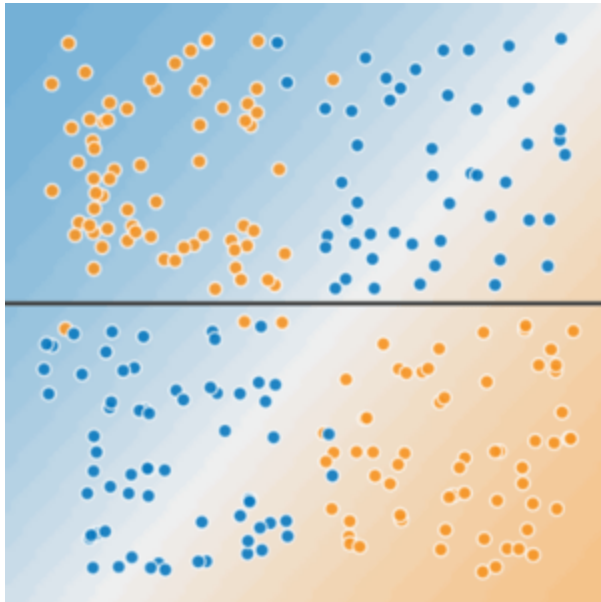
W regresji liniowej wskazywaliśmy konkretną wartość dla danych wejściowych np. w zagadnieniu czy dany email to spam dostawaliśmy odpowiedź tak/nie (1/0).

Regresja logistyczna zwraca nam zamiast tego prawdopodobieństwo, czyli dla danego emaila zwróci nam np. 0,98 (na 98% jest to spam).

Wzór na regresję logistyczną to:

$$F(z) = 1 / (1 + e^{-z}), \text{gdzie } z = w_0x_0 + \dots + w_nx_n$$

Nieliniowość

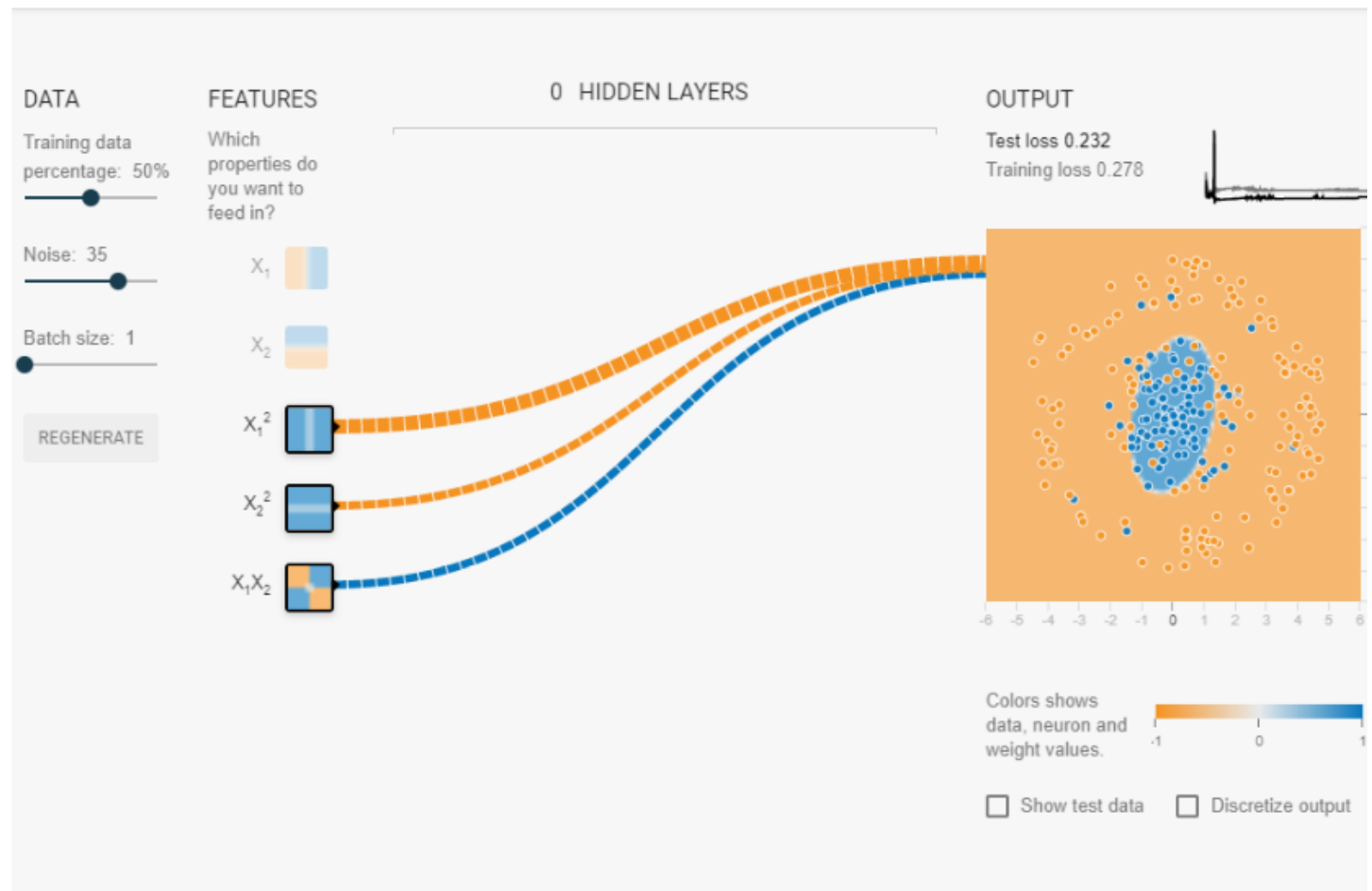


W rzeczywistości dane bardzo często nie przyjmują postaci liniowej. Ilustracja pokazuje przykładowy zbiór danych, którego nie jesteśmy w stanie podzielić jedną prostą. Niezależnie od jej umiejscowienia w najlepszym wypadku otrzymamy dokładność $\pm 50\%$.

Feature crosses

W sytuacji gdzie nie możemy policzyć naszej funkcji jako pewnej wartościowanej sumy naszych wejść możemy się zastanowić nad dodaniem dodatkowych parametrów np. gdy mamy dane x_1 i x_2 możemy dodać $x_3 = x_1 * x_2$.

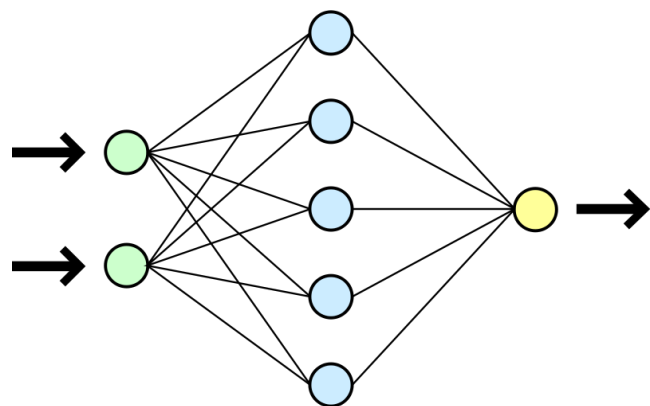
Przykład na obrazku pokazuje jak przy pomocy x_1^2 , x_2^2 oraz $x_1 * x_2$ uzyskaliśmy na naszym modelu elipsę



Sieci neuronowe

A solid dark gray horizontal bar spanning the width of the slide, positioned at the bottom.

Czym są sieci neuronowe?



Poprzednio patrzyliśmy na dane, które byliśmy w stanie bez większych problemów opisać używając połączenia wejść i ich kombinacji.

Często jednak rzeczywiste dane są dużo bardziej skomplikowane. W takim wypadku możemy dodać do naszego modelu warstw pośrednie (nazywane ukrytymi) między wejściem a wyjściem. Posłużą one do wyłapania pewnych zależności w naszych danych. Każda z tak dodanych warstw składa się z neuronów.

Dla przykładu nasz model może się nauczyć, że jeden z tych neuronów jest aktywny wtedy i tylko wtedy, gdy dom znajduje się w "dobrej dzielnicy".

<https://visionhelp.wordpress.com/2011/07/25/practical-wisdom-part-5-the-neural-architecture-of-pattern-recognition/neural-network-basic-photo/>

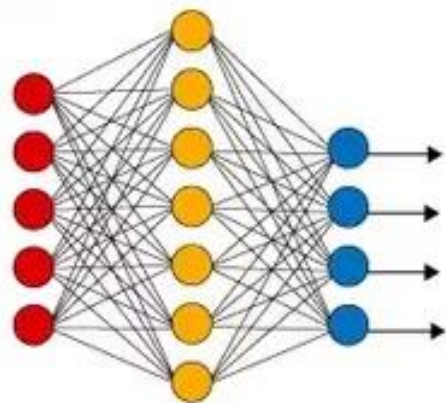
Funkcja aktywacji

Wartość neuronu jest ustalana poprzez policzenie wartości funkcji aktywacyjnej. Funkcje aktywacyjne to funkcje, które skalują naszą sumę danych wejściowych do danego neuronu pomnożonych przez ich wagi. Przykładem może być funkcja sigmoid, która opisana jest wzorem: $F(z) = 1 / (1 + e^{-z})$, gdzie $z = w_0x_0 + \dots + w_nx_n$. Skaluje ona wyniki do przedziału $[0,1]$. Inną popularną funkcją aktywacyjną jest RELU, której wzór wygląda tak: $F(x) = \max(0, x)$.

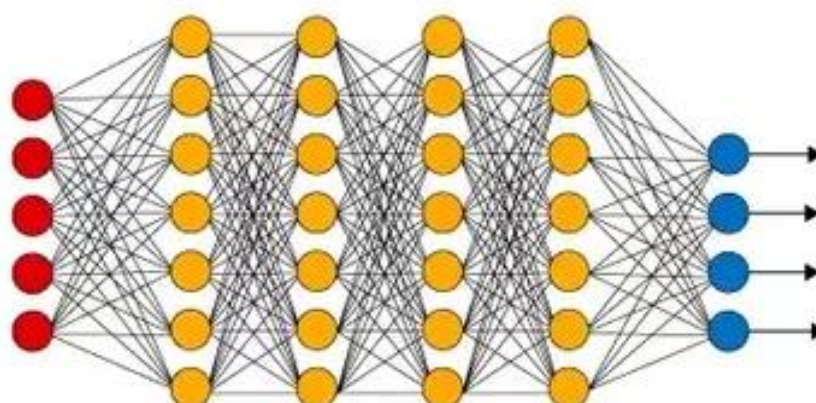
Dodatkowo możemy chcieć aby nasz neuron aktywował się tylko jeżeli $z > b$. Aby do tego doszło musimy po prostu użyć funkcji $\sigma(z - b)$, gdzie σ to nasza funkcja aktywacyjna. W tym wypadku b jest naszą stronniczością (ang. bias).

Istnieją bardzo wiele funkcji aktywacji. Ich listę można znaleźć pod adresem:
https://en.wikipedia.org/wiki/Activation_function

Simple Neural Network



Deep Learning Neural Network



● Input Layer ● Hidden Layer ● Output Layer

Głębokie sieci neuronowe

JEST TO RODZAJ SIECI
NEURONOWYCH, KTÓRE
POSIADAJĄ DWIE LUB
WIĘCEJ WARSTW
UKRYTYCH.

<https://www.quora.com/What-is-the-difference-between-Neural-Networks-and-Deep-Learning>

Convolutional Neural Networks

Jest to specyficzny typ głębokich sieci neuronowych, który specjalizuje się w zadaniach polegających na klasyfikacji/znajdowaniu wzorów. Z tego powodu jest bardzo popularny w przetwarzaniu obrazów.

Ich elementem charakterystycznym jest to, że przynajmniej jedna z warstw ukrytych dokonuje konwolucji (ang. Kernel convolution). Oznacza to, że otrzymuje ona filtr np. macierz 3x3 i liczy jej iloczyn skalarny z fragmentem wejścia (iteracyjnie przebiega w ten sposób po całym obrazie) po czym uzyskany wynik zapisuje tworząc w ten sposób nowy obraz z np. wyostrozonymi krawędziami.

Modyfikując filtry, które wpuszczamy do naszej sieci modyfikujemy jakie wzorce odnajduje.

Bardzo dobre wytłumaczenie ogólne można znaleźć pod linkami:

<https://www.youtube.com/watch?v=py5by00HZM8>,

https://www.youtube.com/watch?v=BFdMrDOx_CM,

<https://www.youtube.com/watch?v=TJIAxW-2nml>,

https://www.youtube.com/watch?v=YRhxdVk_sls

Technologie



FileEditViewInsertRuntimeToolsHelp

Table of contents

<>

Getting started

Data science

Machine learning

More Resources


Machine Learning Examples

Section

+ Code+ TextCopy to Drive

ConnectedEditing

↑↓↶↷🗑️⋮

What is Colaboratory?

Colaboratory, or "Colab" for short, allows you to write and execute Python in your browser, with

- Zero configuration required
- Free access to GPUs
- Easy sharing

Whether you're a **student**, a **data scientist** or an **AI researcher**, Colab can make your work easier. Watch [Introduction to Colab](#) to learn more, or just get started below!

Getting started

The document you are reading is not a static web page, but an interactive environment called a **Colab notebook** that lets you write and execute code.

For example, here is a **code cell** with a short Python script that computes a value, stores it in a variable, and prints the result:

```
[ ] seconds_in_a_day = 24 * 60 * 60
seconds_in_a_day
```

86400

To execute the code in the above cell, select it with a click and then either press the play button to the left of the code, or use the keyboard shortcut "Command/Ctrl+Enter". To edit the code, just click the cell and start editing.

Variables that you define in one cell can later be used in other cells:

```
[ ] seconds_in_a_week = 7 * seconds_in_a_day
seconds_in_a_week
```

604800

Google Colaboratory

<https://colab.research.google.com/notebooks/intro.ipynb>

Bibliografia

1. <http://neuralnetworksanddeeplearning.com/chap1.html> - Książka ilustrująca uczenie maszynowe na klasycznych przykładach
2. <https://developers.google.com/machine-learning/crash-course/ml-intro> - Kurs wprowadzający do uczenia maszynowego od Google
3. <https://www.kaggle.com/learn/intro-to-machine-learning> - Kurs wprowadzający do uczenia maszynowego od Kaggle (znany organizator konkursów uczenia maszynowego i data science)
4. https://www.youtube.com/playlist?list=PLZHQOb0WTQDNU6R1_67000Dx_ZCJB3pi - Playlista wizualizująca zasady działania sieci neuronowych
5. https://www.youtube.com/playlist?list=PLQVvva00QuDfKTOs3Keg_kaG2P55YRn5v - Playlista o działaniu i implementacji uczenia maszynowego z użyciem języka Python
6. https://www.youtube.com/playlist?list=PLtBw6njQRU-rwp5_7C0olVt26ZgiG9NI - Playlista zawierająca wprowadzenie do uczenia maszynowego z uczelni MIT
7. <https://www.youtube.com/user/consumerchampion> - Kanał na YT o Python, Data Science i ML
8. <http://ocdevel.com/mlg> - Podcast o ML
9. <https://www.tensorflow.org/tutorials/quickstart/beginner> - Oficjalny tutorial do biblioteki Tensorflow
10. <https://keras.io/> - Dokumentacja biblioteki Keras
11. "Matematyka i już" Richard Elwes ISBN 978-83-7705-311-9 Rozdział "Jak stworzyć cyfrowy mózg"
12. <https://www.kdnuggets.com/2018/10/simple-neural-network-python.html> - sieć neuronowa napisana jedynie z użyciem biblioteki NumPy