Przekrojowe wprowadzenie do uczenia maszynowego

Seminarium Dynamiki PWR

Maciej Ziaja, Inżynier uczenia maszynowego
POLSL, Katedra Algorytmiki i Oprogramowania | KP Labs, dział uczenia maszynowego
marzec 2024

Tło zawodowe

KP Labs i Politechnika Śląska

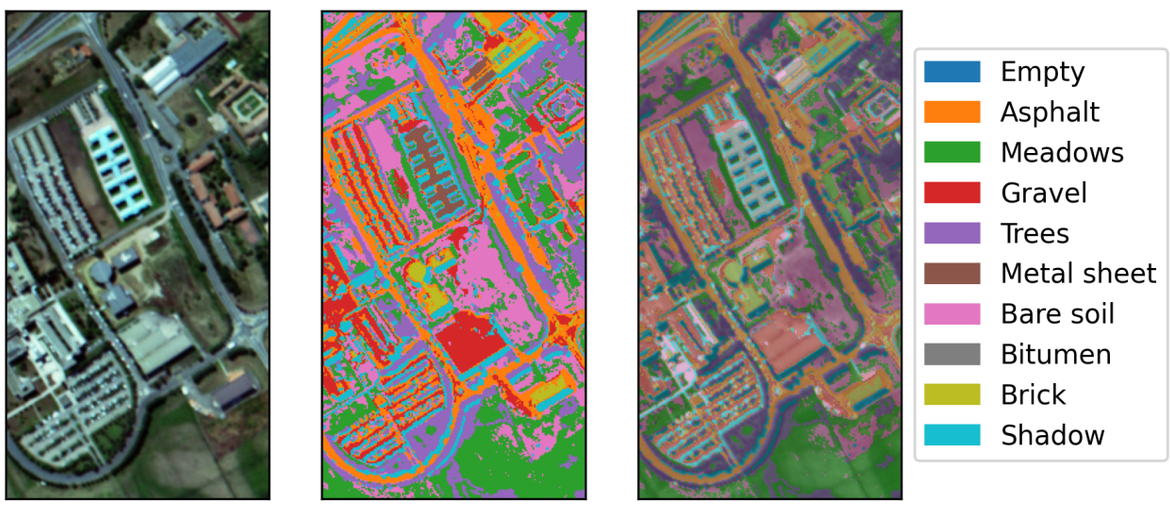
- KP Labs (kplabs.space) – **produkcja modułów sprzętowych oraz oprogramowania do autonomizacji platform kosmicznych** (głównie satelitarnych):
  - **moduły sprzętowe** – komputery pokładowe i jednostki do obliczeń sztucznej inteligencji na pokładzie aparatury kosmicznej,
  - **moduły oprogramowania** – komponenty do elastycznego zarządzania misją oraz modele sztucznej inteligencji do analizy obserwacji satelitarnych Ziemii i telemetrii *on-board* dla urządzeń kosmicznych.
- Sprzedawane moduły są kierowane do innych firm z branży kosmicznej jako komponenty do budowy końcowych rozwiązań takich jak: monitorowanie upraw rolnych, ekosystemów, nadzoru administracyjnego i innych.
- Flagowym projektem firmy jest **satelita Intuition–1 zbudowany z komponentów produkowanych przez firmę**.
- Archiwalny zapis relacji ze startu misji: https://www.youtube.com/watch?v=7xsd-TPYg1Y&t=669s.
- Położenie satelity na żywo: https://satellitetracker3d.com/

Proces przygotowań i startu misji Intuition-1 na pokładzie statku Falcon 9, 11 listopada 2023

Tło zawodowe

Algorytmy uczenia maszynowego na pokładzie Intuition-1

Satelita wykonuje zdjęcie → Zdjęcie jest przetwarzane na orbicie przez sztuczną inteligencję → W czasie sesji komunikacyjnej z bazą naziemną przesyłane są wyniki analizy na orbicie



Segmentacja obrazu wykonana przez Intuition-1 na orbicie okołoziemskiej

Agenda

Przekrój dziedziny uczenia maszynowego

1. Dane i zadania uczenia maszynowego
2. Tradycyjne modele uczenia maszynowego
3. Sztuczne sieci neuronowe
4. Głębokie sieci neuronowe (tak zwana *sztuczna inteligencja*)
5. Pozostałe rodzaje uczenia maszynowego

Dane i zadania uczenia maszynowego

Definicja uczenia maszynowego

W kontraście do programowania imperatywnego

Uczenie maszynowe (*machine learning – ML*), to technika rozwiązywania problemów poprzez definicję modelu o zestawie parametrów i ekspozycję go na dane reprezentujące pewne zależności. Wynikiem procesu uczenia maszynowego jest model, którego parametry są dopasowane do rozwiązania danego problemu.

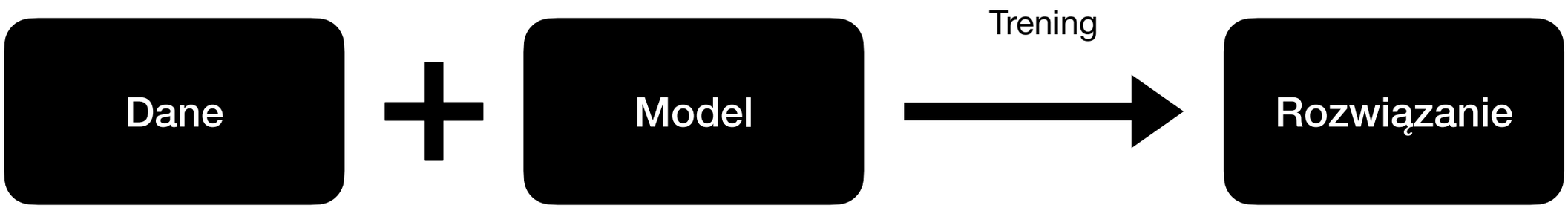
Przeciwnym podejściem (tradycyjnym) jest programowanie imperatywne, w którym rozwiązanie problemu jest opracowane przez człowieka w podejściu analitycznym, a wyniki są uzyskiwane przez realizację zaprogramowanych kroków, które są zdefiniowane przez autora algorytmu.

Definicja uczenia maszynowego

Podejście intuicyjne

Autor przygotowuje

Zbiór danych i sposób ich przetwarzania	Architekturę modelu	Procdurę treningową	Autor **nie** definiuje rozwiązania problemu



Dane muszą być reprezentatywne dla danego problemu	Przy inicjalizacji parametry modelu nie są dostosowane do problemu (zazwyczaj losowe)	Rozwiązaniem jest model o parametrach dostosowanych do konkretnego problemu

Schemat przygotowania rozwiązania problemu z użyciem uczenia maszynowego

Rodzaje uczenia maszynowego

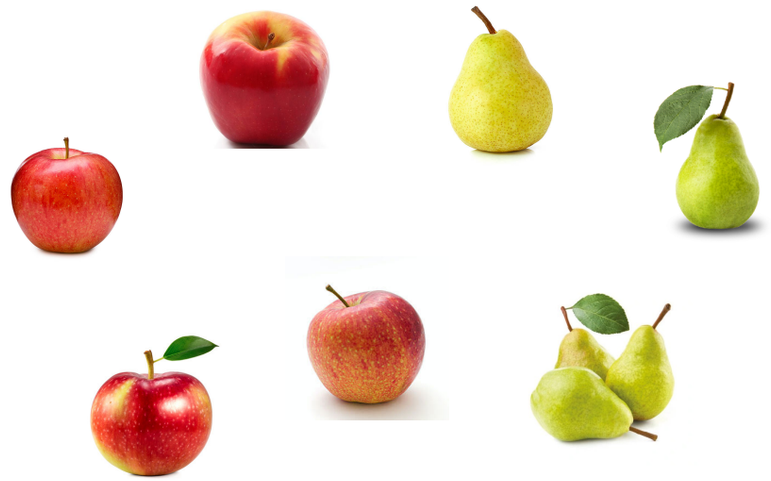
Ze względu na sposób uczenia

- **Uczenie nadzorowane (*supervised learning*)** – w skład zbioru danych treningowych wchodzą dane wraz z przykładami pożądanego wyjścia modelu. Zależność między danymi wejściowymi, a pożądanym wyjściem definiuje problem. Wyjścia modelu są często nazywane **etykietami (label)**, a przykłady etykiet z pożądanymi danymi nazywane **etykietami ground truth (GT)**. Etykiety GT są opracowywane poza procesem uczenia maszynowego, przez *ekspertów*.

- **Uczenie nienadzorowane (*unsupervised learning*)** – w zbiorze danych treningowych znajdują się dane bez przykładów pożądanego wyjścia modelu. Modele uczenia nienadzorowanego uczą się realizować zadania poprzez wyszukiwanie zależności i podobieństwa wewnątrz zbiorów danych.

- Inne rodzaje uczenia:
  - metody hybrydowe,
  - *reinforcement learning* (uczenie przez wzmacnianie).
- Podział i mnogość zadań są bardziej złożone, w trakcie prezentacji przeanalizujemy dużo więcej przypadków.

Zbiór do uczenia nadzorowanego



Zbiór do uczenia nienadzorowanego

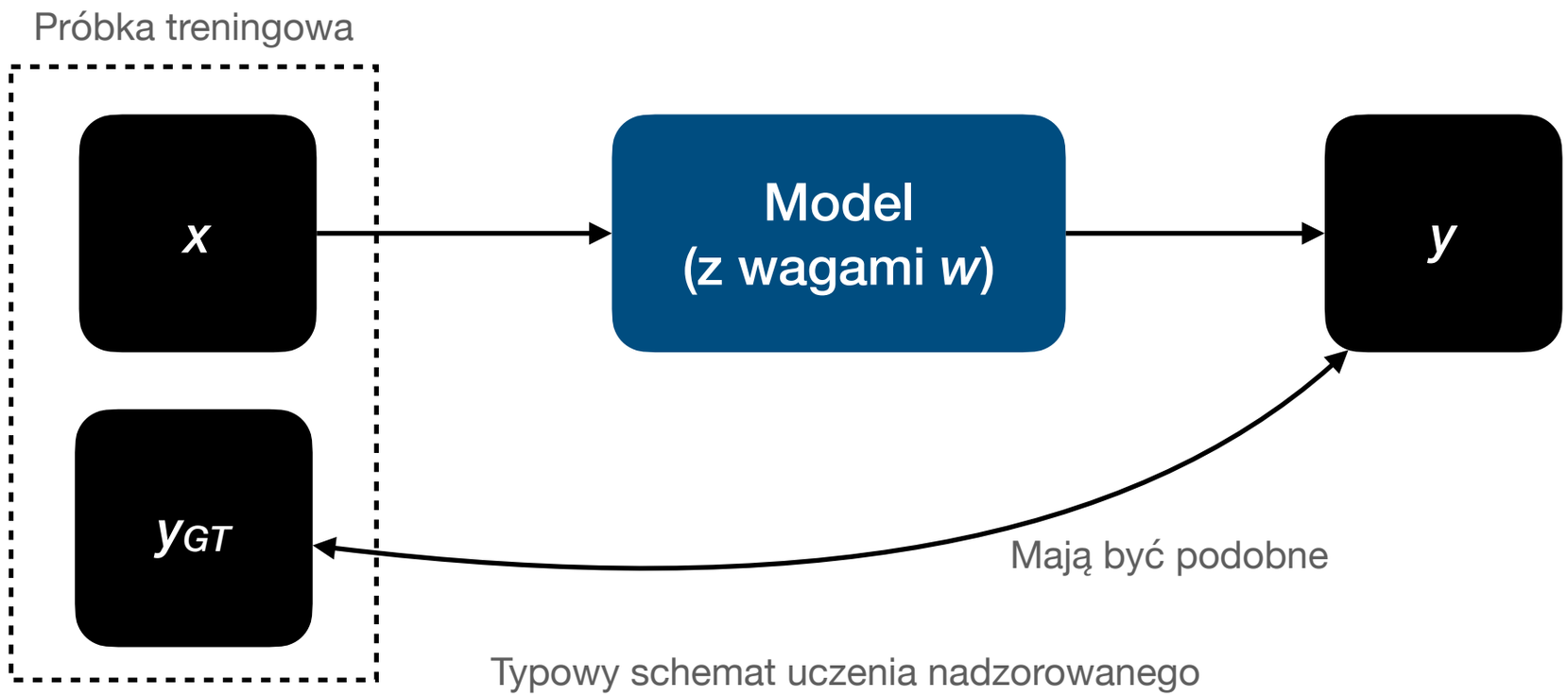
Uczenie nadzorowane

Główny dział uczenia maszynowego

- Model jest (zazwyczaj) funkcją *f(x) = y*, z zestawem parametrów *w* (można zapisać jako *f(x, w) = y*). Celem modelu jest przewidywanie wyjścia *y* dla wejścia *x*. Zbiór treningowy zawiera przykładowe **pary wejść i pożądanych wyjść *(x, y_{GT})***. Trening odbywa się przez ekspozycję modelu na przykładowe wejścia oraz pożądane etykiety *ground truth.* W trakcie uczenia modelu jego parametry są dostosowywane, tak by *f(x, w) ~= y_{GT}.*

- Wyuczony model jest w stanie przewidywać etykiety dla nowych wejść nie wchodzących w skład zbioru treningowego (o ile **nowe dane i dane treningowe reprezentują ten sam problem**). Model trenujemy na zbiorze treningowym ze spreparowanymi przykładami, używamy potem w realnym zastosowaniu na nowych danych (**in the wild**). **Wytrenowany model ocenia się na danych testowych**.

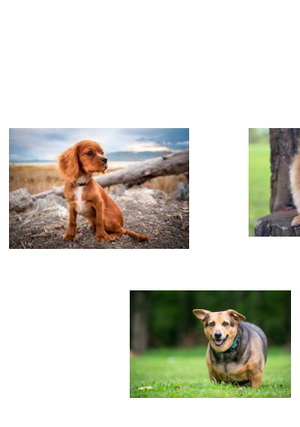
- Kluczowa jest duża **ilość danych oraz ich reprezentatywność**.



Typowy schemat uczenia nadzorowanego

Uczenie nadzorowane

Rodzaje uczenia nadzorowanego



Uczenie nadzorowane

Przewidywanie klasy/typu (np. gatunku zwierzęcia na zdjęciu)

Jaki cel?

Przewidywanie wielkości (np. wartości używanego samochodu na podstawie danych technicznych)

$ 31 000

$ 6 000

$ 50 000

Klasyfikacja

Regresja

Ile klas w zbiorze?

Ile klas na jeden obiekt?

Dwie (np. zdrowy/chory)

Wiele (np. samochód/samolot/ statek)

Jedną (np. dana osoba zatrudniona albo jest bezrobotna)

Wiele (np. dana osoba jest inżynierem i wykładowcą akademickim)

Binarna

Wieloklasowa

Jednoetykietowa

Wieloetykietowa

Rodzaje danych

Mnogość danych cyfrowych

- **Dane tabelaryczne (ustrukturyzowane)** (często finansowe, np. dane Excela oraz dane naukowe dotyczące zachowań społecznych lub pomiary z aparatury naukowej).
- **Dane multimedialne (nieustrukturyzowane)**:
  - **dane tekstowe i sekwencyjne** (język naturalny, dane genetyczne, ciągi czasowe),
  - **dane obrazowe** (obrazy w skali szarości, RGB, hiperspektralne, multispektralne),
  - **dane audio**,
  - **dane audio-wideo**.
- **Dane grafowe** (sieci powiązań i struktury np. chemiczne).
- W uczeniu maszynowym natura danych odgrywa dużą rolę, często mówimy o domenie zastosowania maszynowego i wiedzy domenowej na temat zagadnień, które są rozwiązywane przy pomocy uczenia maszynowego. Inżynier uczenia maszynowego ma wiedzę informatyczną, ale nie zna konkretnych domen (np. finansowej, społecznej, robotyki, chemii, fizyki, uprawy roli, itd.).

	Pregnancies	Glucose	BloodPressure	SkinThickness	Insulin	BMI	DiabetesPedigreeFunction	Age	Outcome
758	1	106	76	0	0	37.5	0.197	26	0
759	6	190	92	0	0	35.5	0.278	66	1
760	2	88	58	26	16	28.4	0.766	22	0
761	9	170	74	31	0	44.0	0.403	43	1
762	9	89	62	0	0	22.5	0.142	33	0
763	10	101	76	48	180	32.9	0.171	63	0
764	2	122	70	27	0	36.8	0.340	27	0
765	5	121	72	23	112	26.2	0.245	30	0
766	1	126	60	0	0	30.1	0.349	47	1
767	1	93	70	31	0	30.4	0.315	23	0

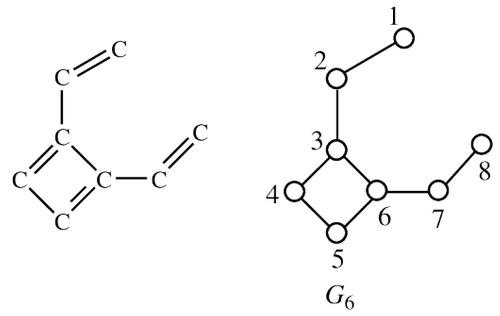
Medyczne dane tabelaryczne Źródło

Dane obrazowe OCR Źródło

No estimates of total proved net oil or natural gas reserves have been filed with or included in reports to any federal authority or agency, other than the SEC, since October 1, 2011 September 30, 2012 Oil, including natural gas liquids (Bbls) 1,078,000 Natural gas (Mcf) 11,109,000 Total (Boe) 2,993,000 During fiscal 2012, Barnwell total net proved developed reserves, including proved developed producing reserves, of oil and natural gas liquids decreased by 106,000 Bbls (9%) and total net proved developed reserves of natural gas decreased by 3,834,000 Mcf (26%). Standardized Measure of Discounted Future Net Cash Flows The following table sets forth Barnwell Estimated Future Net Revenues from total proved oil, natural gas and natural gas liquids reserves and the present value of Barnwell Estimated Future Net Revenues (discounted at 10%).

Biznesowe dane tekstowe Źródło

Finansowe dane ciągów czasowych Źródło



Chemiczne dane grafowe
http://dx.doi.org/10.1201/b18389

Rodzaje danych w uczeniu maszynowym

Podział danych z punktu widzenia modelów uczenia maszynowego

- **Wszystkie dane cyfrowe są reprezentowane liczbowo** (na poziomie maszynowym bitowo), jednak ich sposoby kodowania są bardzo różne (np. kodowanie ASCII, UTF-8, formaty audio-wideo, jpg, png, tif, mp3, mp4, mov, itd.).
- **Wszystkie rodzaje danych należy sprowadzić do formy zdanej do użycia przez modele uczenia maszynowego.** Zazwyczaj konkretny rodzaj danych wiążę się z konkretnym sposobem sprawdzenia go do bardziej standardowej formy.
- W uczeniu maszynowym rozróżniamy dwa główne rodzaje danych:
  - **Dane kategoryczne (categorical)** – dane oznaczające kategorie nazywane też *klasami.*
    - Mogą być nieuporządkowane (np. rodzaj samochodu – sedan, minivan, coupé, SUV; gatunek zwierzęcia – pies, kot), albo uporządkowane (categorical ordinal data) (np. grupa wiekowa – dziecko, dorosły, emeryt).
  - **Dane ciągłe (continuous)** – dane o wartościach ciągłych:
    - Mogą być o **nieograniczonym zakresie** (np. cena nieruchomości nie ma sztywnej górnej granicy), lub **określonym zakresie** (pixele na zdjęciu przyjmują wartości między zerem reprezentującym kolor czarny i wartością maksymalną reprezentującą kolor biały).
    - mogą być ciągłe rzeczywiste, lub skwantyzowane (np. na liczbach całkowitych).
- Modele uczenia maszynowego (typowo) operują tylko na liczbach, **wszystkie typy danych należy przedstawić w formie liczb**, które kodują dane kategoryczne lub ciągłe.

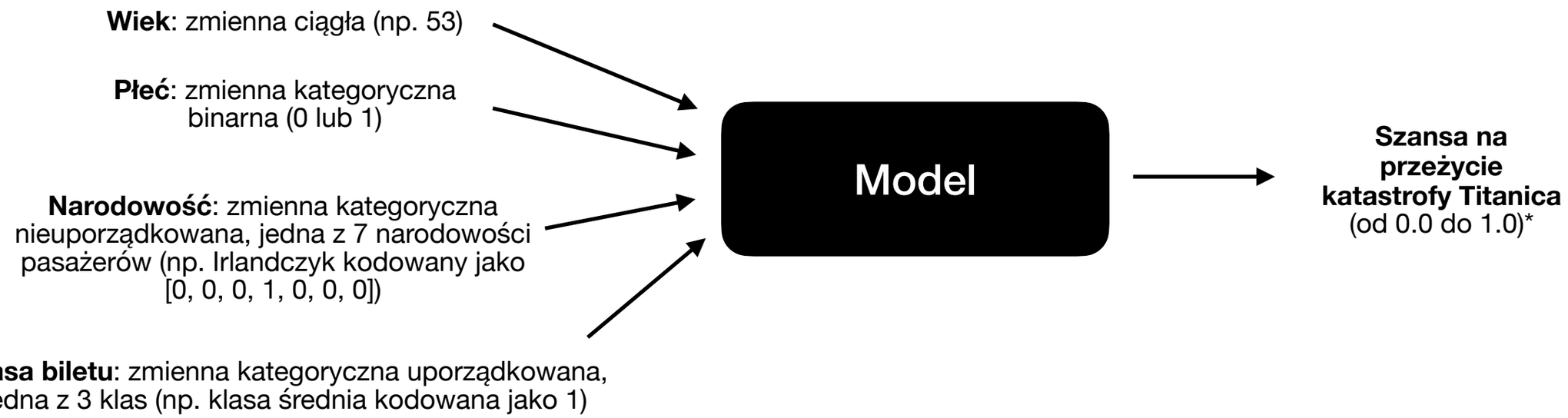
Kodowanie i preprocessing danych

Jak przetwarzać dane do uczenia maszynowego

- **Dane ciągłe** przetwarzamy do:
  - **Niezmienionej postaci –** np. wiek pacjenta do diagnostyki podajemy jako liczbę (np. pacjent w wieku **21** lat).
  - **Postaci znormalizowanej –** dla danych o określonym zakresie dobrze jest znormalizować je w przedziale [0, 1], gdzie 1 to wartość maksymalna (np. poziom napełnienia zbiornika na ciecz, od pustego do pełnego w 100%, zbiorniku pełen w stopniu opisanym liczbą **0.31**).
  - **Postaci poddanej standaryzacji –** dla danych o nieokreślonych wartościach minimalnych i maksymalnych przydane *może być* poddanie ich standaryzacji (**odjęcie wartości średniej i podzielenie względem odchylenia standardowego ze zbioru treningowego**).
- **Dane kategoryczne:**
  - **Dane kategoryczne (porządkowe)** kodujemy liczbami, gdzie dana liczba oznacza pewną kategorię (np. szeregujemy kategorie: dziecko, dorosły, emeryt i przypisujemy im kolejno wartości: **0, 1, 2**).
  - **Dla kategorii nieuporządkowanych** kodowanie na rosnących liczbach może fałszywie sugerować modelowi pewną kolejność klas. Dlatego dla danych nieuporządkowanych stosuje się kodowanie z gorącą jedynką *one-hot-encoding* jako wektory rzadkie, gdzie indeks z wartością 1 koduje klasę (dla kategorii: sedan, minivan, coupe stosujemy kodowanie: **[1, 0, 0], [0, 1, 0], [0, 0, 1]**).

Przykłady modeli i danych uczenia maszynowego

Różne typy danych i zadań



Przykład modelu uczenia maszynowego dla danych tabelarycznych

Przykład modelu uczenia maszynowego dla danych obrazowych

* Przykład inspirowany prawdziwym zbiorem, patrz: https://www.kaggle.com/competitions/titanic/data

Tradycyjne modele uczenia maszynowego

Tradycyjne modele uczenia maszynowego

Modele sprzed rozwoju dziedziny uczenia głębokiego (AI)

- Modele sprzed rewolucji związanej z głęboki sieciami neuronowymi.
- **Bazują na technikach statystycznych** – łatwe w interpretacji (niekoniecznie łatwe w sensie kolokwialny, ale dobrze opisane modele matematycznie o znanych właściwościach).
- **Najlepiej sprawdzają się dla danych tablarycznych, o małych rozmiarach zbiorów**.
- Przykłady:
  - k-nearest neighbours (KNN) – metody oparte na sąsiedztwie (np. odległości euklidesowej) w próbek w przestrzeni cech wejściowych,
  - naiwny klasyfikator bayesowski (Naive Bayes classifier) – opiera się na twierdzeniu o prawdopodobieństwie całkowitym i jego odwrotności, czyli twierdzeniu Bayesa,
  - maszyny wektorów podpierających (SVM – Support vector machines) – opierają się na metodach optymalizacji na płaszczyznach podziału cech wejściowych,
  - drzewa decyzyjne (d*ecision trees*) – opiera się na metodach statystycznych,
  - losowe lasy (*Random Forest*) – ensemble drzew decyzyjnych,
  - **XGBoost** (https://xgboost.ai/) – ensemble drzew decyzyjnych budowane techniką gradient boosting – wiodąca rodzina modeli uczenia tradycyjnego.

* Techniki ensemble to osobna dziedzina zagadnień, zobacz takie hasła jak bootstrap, bagging, gradient boosting

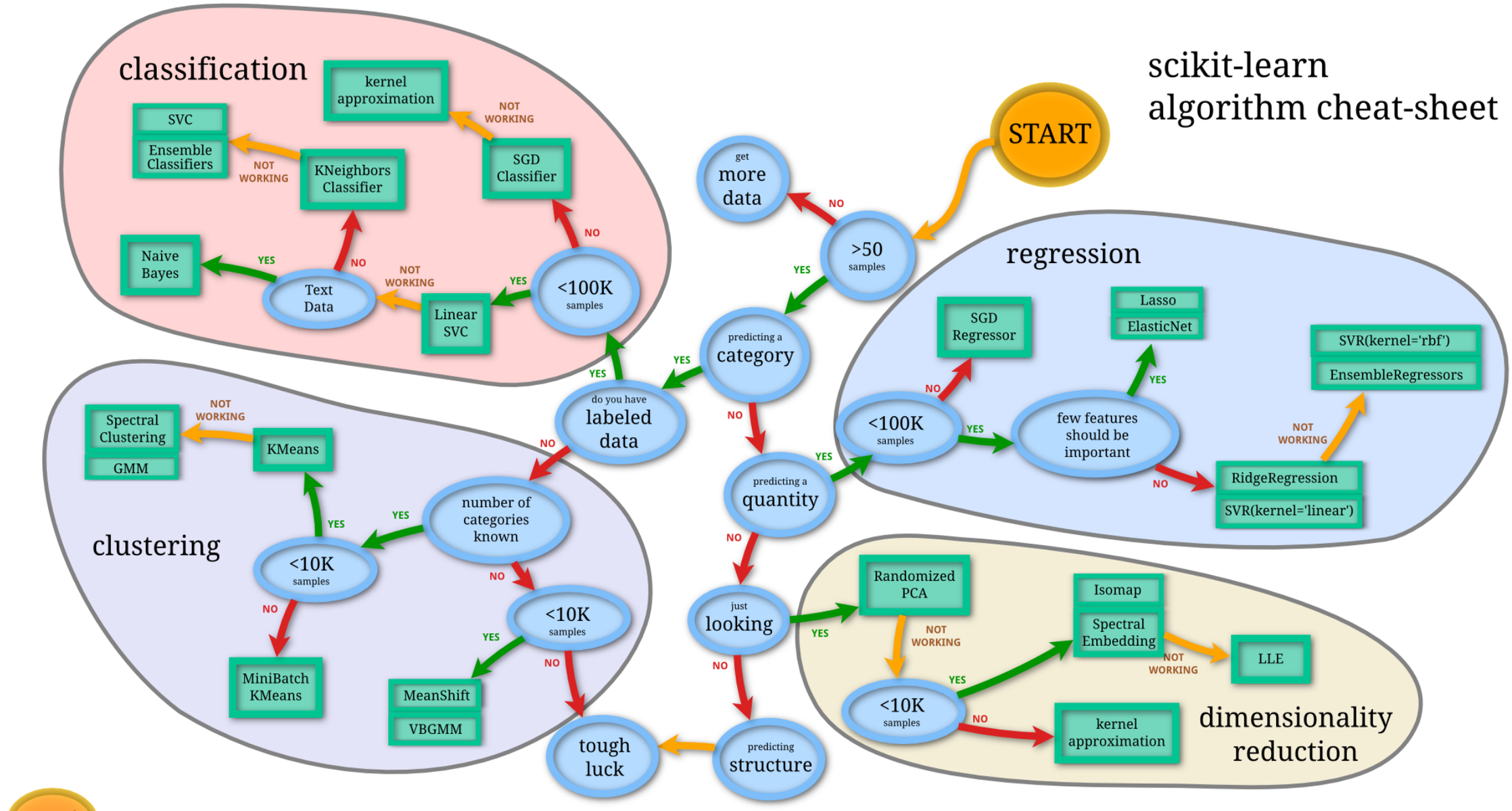
Technologie tradycyjnego uczenia maszynowego

Wprowadzenie do inżynierii uczenia maszynowego

- **Python** jest wiodącym językiem programowania służącym do uczenia maszynowego (alternatywy: Julia – szybsza, nowsza, mniej stabilna, mniej popularna, Matlab – mniej popularny, zamknięty, ograniczony, przyjazny dla osób niezwiązanych z programowaniem).

- W tradycyjnym uczeniu maszynowym wykorzystuje się **biblioteki języka Python**:
  - **NumPy** – obliczenia numeryczne (macierzowe),
  - **SciPy** – obliczenia naukowe i statystyczne,
  - **Pandas** – analiza i przetwarzanie danych tabelarycznych,
  - **Scikit-learn** – tradycyjne uczenie maszynowe,
  - **Matplotlib** – wykresy i wizualizacja.

Technologie tradycyjnego uczenia maszynowego



Mapa wyboru algorytmu uczenia maszynowego w ramach biblioteki Scikit-learn [1]

[1] https://scikit-learn.org/stable/tutorial/machine_learning_map/index.html

Przykład procesu uczenia maszynowego

Najprostszy praktyczny przykład klasyfikacji wieloklasowej

- Zbiór Iris zawiera dane w dziedzinie uczenia nadzorowanego: wejścia to 4 liczby opisujące długość i szerokość (w cm) płatków kwiatów irysów, a dołączone etykiety ground truth kodują jeden z trzech podgatunków irysów (Iris-setosa, Iris-versicolor, Iris-virginica).

```python
# Importing necessary libraries
import numpy as np
from sklearn import datasets
from sklearn.model_selection import train_test_split
from sklearn.neighbors import KNeighborsClassifier

# Load the iris dataset
iris = datasets.load_iris()
X = iris.data  # Features
y = iris.target  # Target variable

# Print sample data
print(f'{X.shape=}, {y.shape=}')
print(f'{np.unique(y)=}')
print(f'{X[0]=}, {y[0]=}')

# Split the data into training and testing sets
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2)

# Instantiate the classifier
knn = KNeighborsClassifier(n_neighbors=3)

# Train the model
knn.fit(X_train, y_train)

# Predictions on the testing set
predictions = knn.predict(X_test)

# Evaluating the accuracy
accuracy = sum(predictions == y_test) / len(y_test)
print("Accuracy:", accuracy)
```

Przykładowy program do klasyfikacji podgatunku Irysów w zbiorze Iris

Wizualizacja danych i rozpatrywanych klas w zbiorze Iris [1]

```
X.shape=(150, 4), y.shape=(150,)
np.unique(y)=array([0, 1, 2])
X[0]=array([5.1, 3.5, 1.4, 0.2]), y[0]=0
Accuracy: 0.9666666666666667
```

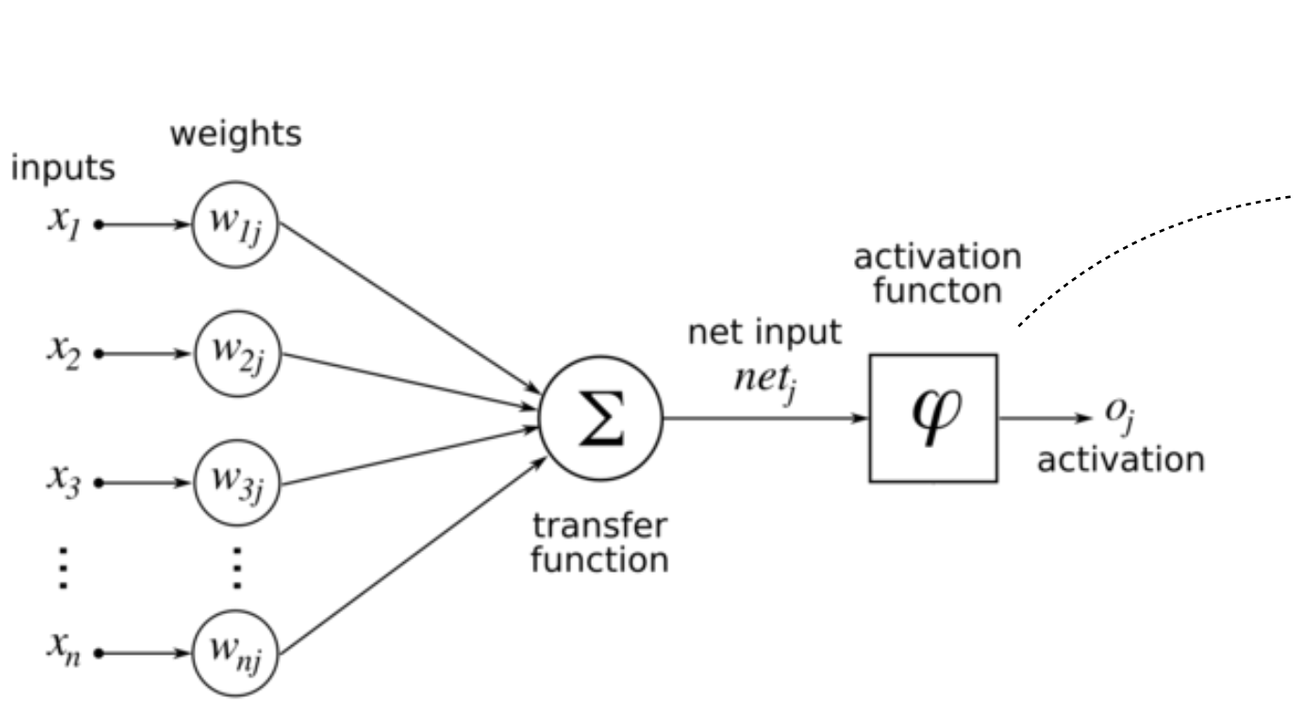
Wyjście przykładowego programu klasyfikacji dla zbioru Iris

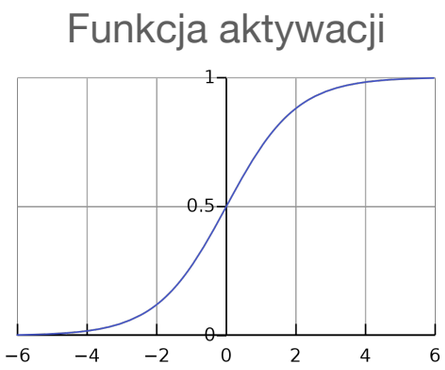
https://www.embedded-robotics.com/iris-dataset-classification/

Sztuczne sieci neuronowe

Perceptron Rosenblatta

Podstawowy budulec sieci neuronowych





Perceptron Rosenblatta
model sztucznego neuronu [1]

$$o_j = \phi\left(\sum_{n}^{i=1} x_{ij} wij + b_i\right)$$

Postać podstawowa

$$o_j = \phi(w^T \cdot x + b)$$

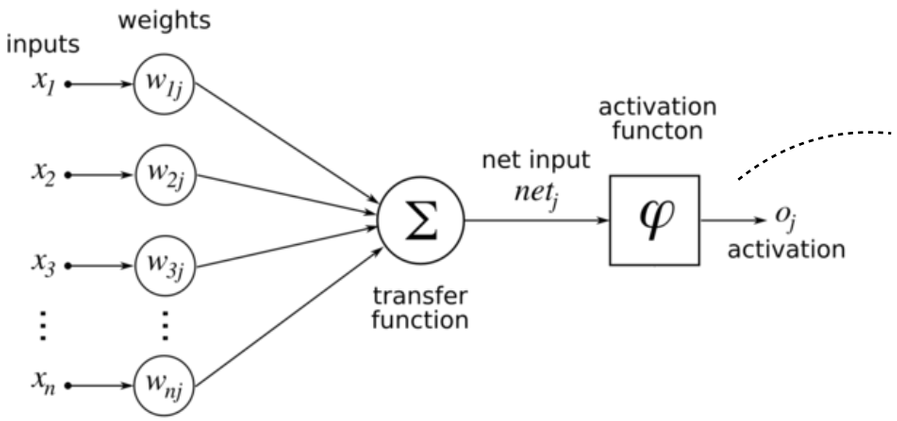
Postać wektorowo macierzowa*

* Istnieją różne konwencje zapisu, czasem można natrafić na wzory bez transpozycji

[1] Zródło obrazu (obraz zmodyfikowano)

Perceptron Rosenblatta

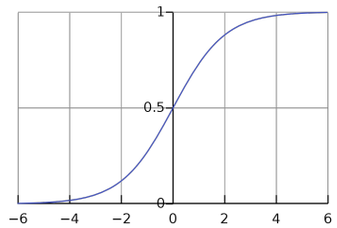
Podstawowy budulec sieci neuronowych



Perceptron Rosenblatta

model sztucznego neuronu

Funkcja aktywacji



$$o_j = \phi\left(\sum_{n}^{i=1} x_{ij} wij + b_i\right)$$

Postać podstawowa

$$o_j = \phi(w^T \cdot x + b)$$

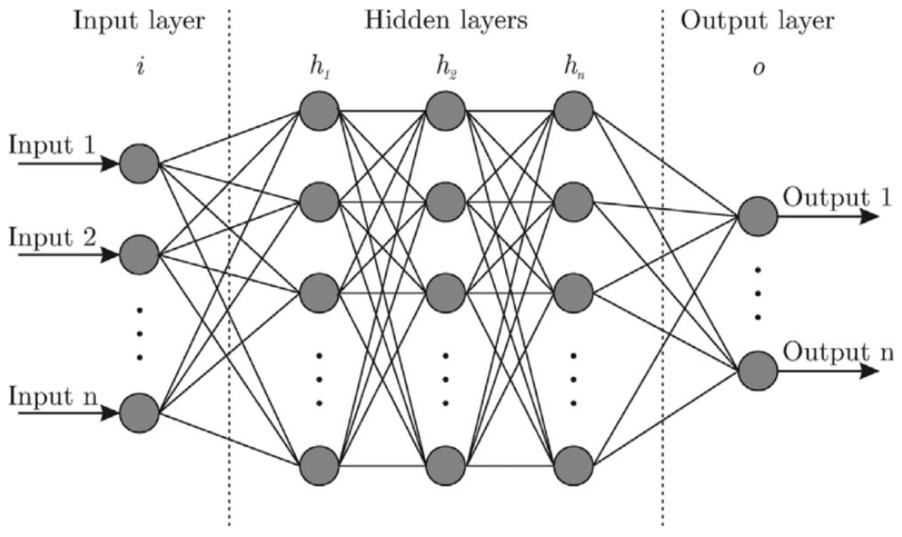
Postać wektorowo macierzowa

- Obliczenie wyjść sieci neuronowej na podstawie wejść nazywamy **inferencją (inference) abo forward pass**.
- **Funkcja aktywacji wprowadza do neuronu nieliniowość** (bez niej wszystkie działania w sekwencji neuronów byłby redukowalne to transformacji liniowej), istnieją różne funkcje aktywacji, współcześnie wewnątrz sieci neuronowych stosuje się funkcję ReLu, a w wyjściowej warstwie funkcje sigmoid/ softmax, która sprowadza wyjścia neuronów (logit) do prawdopodobieństw.
- **Postać macierzowa jest preferowana w implementacji**, ze względu na zwięzłość i powiązanie z optymalizacją obliczeń równoległych
- W praktycznych implementacjach i użyciu podstawowe warstwy o architekturze perceptronu nazywane są często **gęstymi (dense)**, w wariacie bez funkcji aktywacji noszą nazwę projekcji **liniowych (linear)**.

Wielowarstwowy perceptron

Łączenie wielu neuronów w sieć neuronową

- Perceptron jest bardzo prostym modelem, sieci neuronowe buduje się przez łączenie neuronów. Sieci możemy rozbudowywać:
  - w kierunku głębokości – łącząc wyjście jednego neuronu z wejściem drugiego. Każdy kolejny neuron połączony z poprzednim tworzy **warstwę**,
  - w kierunku szerokości – ustawiając neurony równolegle. W ramach **warstwy może istnieć wiele równoległych neuronów**.

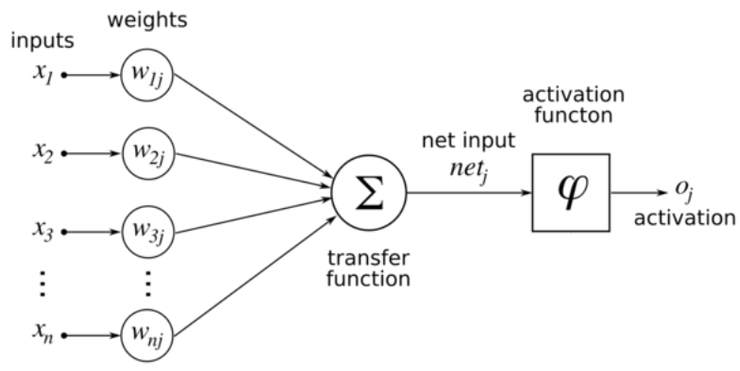


Wielowarstwowy perceptron (sieć neuronowa składająca się z wielu warstw połączonych perceptronów) [1]

[1] https://www.researchgate.net/publication/348049291_Artificial_Neural_Network_Algorithms_for_3D_Printing

Trening sztucznych sieci neuronowych

Jak dopasować wagi modelu do konkretnego zadania



Perceptron Rosenblatta
model sztucznego neuronu

- Sztuczne sieci neuronowe typowo stosuje się w technikach nadzorowanych – zbiór danych treningowych zawiera pary wejść i połączonych z nimi pożądanych wyjść (x, y_{GT}).

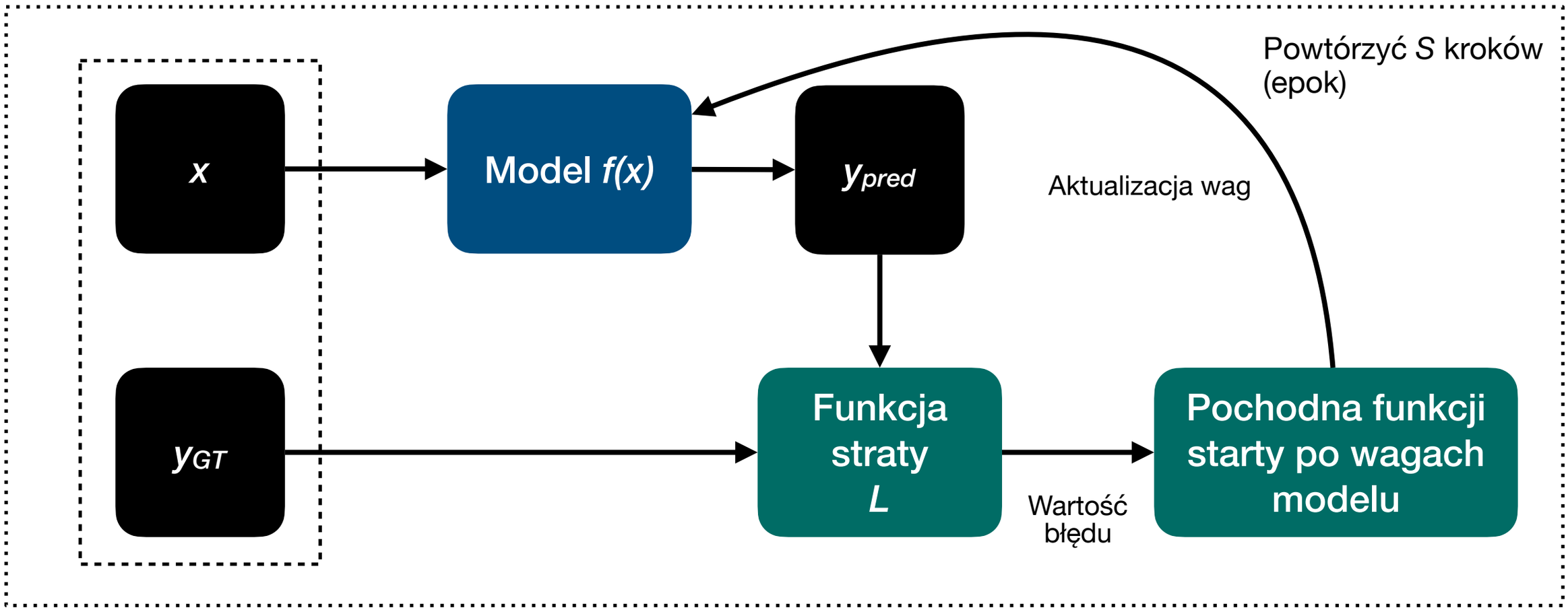
- Proces treningu polega na znalezieniu takich wag w modelu $f(x, w) = y$ by zbliżyć się do stanu $f(x, w) = y_{GT}$.

- Trening można zdefiniować jako **problem optymalizacji z funkcją straty (loss function)** $L(y, y_{GT})$ gdzie za L może posłużyć funkcja błędu średniokwadratowego $L_{MSE}(y, y_{GT}) = (y_{GT} - y)^2$. Proces treningowy ma na celu **minimalizację funkcji starty**.

- Optymalizację przeprowadza się gradientowymi metodami iteracyjnymi. Model jest inicjalizowany losowymi wagami, a algorytm treningowy w kolejnych krokach oblicza gradienty funkcji starty i dąży w kierunku minimum. W podstawowym wariancie taki algorytm nazywa się stochastycznym spadkiem wzdłuż gradientu (**stochastic gradient descent (SGD)**).

Trening sztucznych sieci neuronowych

Algorytm stochastycznego spadku wzdłuż gradientu



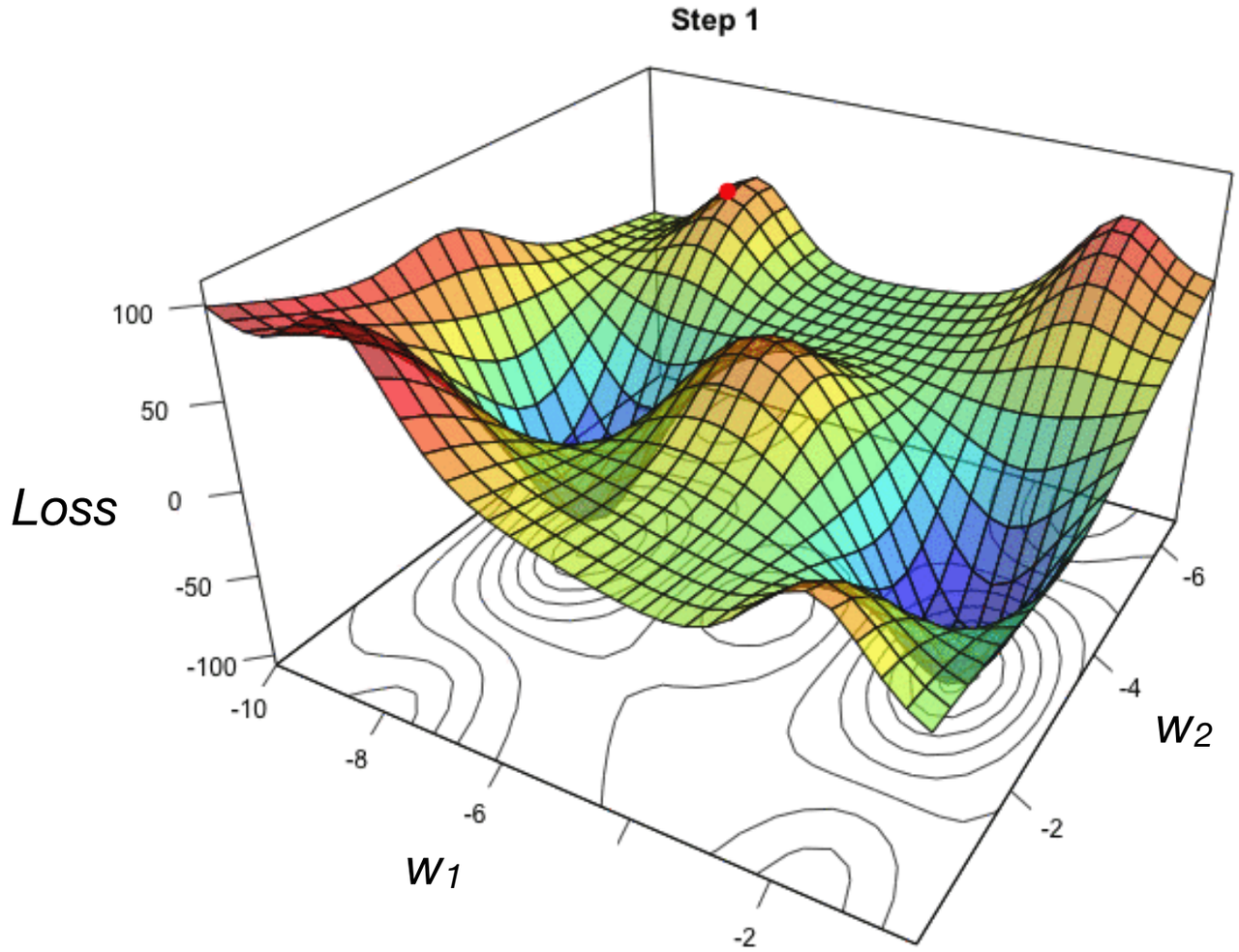
Schemat blokowy algorytmu spadku wzdłuż gradientu

$$W_{i+1} = W_i - \alpha \frac{\partial L(f(W_i, x), y_{GT})}{\partial W}$$

Formuła matematyczna aktualizacji wag modelu w algorytmie spadku wzdłuż gradientu

Trening sztucznych sieci neuronowych

Algorytm stochastycznego spadku wzdłuż gradientu



Wizualizacja minimalizacji funkcji starty w dwóch wymiarach [1]

[1] https://egallic.fr/Enseignement/ML/ECB/book/gradient-descent.html

Trening sztucznych sieci neuronowych

Algorytm propagacji wstecznej (backpropagation)

- Algorytm SGD jest wydajny i nie narzuca ograniczeń co do rodzaju optymalizowanej funkcji.
- We wzorze na krok algorytmu SGD widnieje jeden problematyczny element – **liczenie pochodnej funkcji starty z wyjścia modelu po wagach modelu**.
- Różniczkowanie numeryczne jest trudnym zadaniem, jak obliczać pochodne w trakcie treningu sieci neuronowych?
  - Biblioteki uczenia sieci neuronowych zawierają podstawowe bloku budulcowe (warstwy) do konstrukcji sieci, takie bloki są opatrzone odgórnie podanymi wzorami na ich pochodne.
  - Pochodne złożonych modeli komputer oblicza ze wzoru mówiącego że *pochodna funkcji złożonej jest iloczynem pochodnych funkcji składowych (chain rule)*. **Model jest traktowany jako funkcja złożona swoich warstw**.
  - Można tworzyć swoje moduły oraz podawać ich wzory na wykonanie inferencji (forward) oraz pochodne (backward).
  - W konwencji uczenia sieci neuronowych dane przetwarzane przez sieci nazywamy **tensorami**.

Trening sztucznych sieci neuronowych

Wsparcie bibliotek do uczenia głębokiego w języku Python

- Wszystkie wymienione wcześniej biblioteki do tradycyjnego uczenia maszynowego są przydatne jako narzędzia pomocnicze przy trenowaniu sieci neuronowych.

- Same sieci trenujemy w bibliotekach wspierających **różniczkowalne obliczenia tensorowe**:

  - **Tensorflow** – pierwsza *duża* biblioteka do uczenia sieci neuronowych, obecnie jej popularność drastycznie maleje.

  - **PyTorch** – obecnie najpopularniejsza biblioteka do uczenia sieci neuronowych, zdetronizowała Tensorflow ze względu na dużą elastyczność.

  - **Keras** – Abstrakcyjny interfejs do nauki sieci, którego popularność i podejście do rozwiązywania kwestii technicznych mocno zmieniało się na przestrzeni czasu, obecnie w zaniku, jednak ma szanse na powrót do popularności.

Przykład procesu uczenia sieci neuronowej

Najprostszy praktyczny przykład klasyfikacji przy użycia Pytorch

```python
import torch
import torch.nn as nn
import torch.optim as optim
import numpy as np
from sklearn import datasets
from sklearn.model_selection import train_test_split
from sklearn.preprocessing import StandardScaler
from sklearn.metrics import accuracy_score

# Load the iris dataset
iris = datasets.load_iris()
X = iris.data.astype(np.float32)  # Features
y = iris.target  # Target variable

# Normalize the features
scaler = StandardScaler()
X = scaler.fit_transform(X)

# Split the data into training and testing sets
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_state=42)

# Convert data to PyTorch tensors
X_train = torch.tensor(X_train)
X_test = torch.tensor(X_test)
y_train = torch.tensor(y_train)
y_test = torch.tensor(y_test)

# Define the neural network model
class IrisClassifier(nn.Module):
    def __init__(self):
        super(IrisClassifier, self).__init__()
        self.fc1 = nn.Linear(4, 8)  # 4 input features, 8 hidden units
        self.fc2 = nn.Linear(8, 3)  # 8 hidden units, 3 output classes

    def forward(self, x):
        x = torch.relu(self.fc1(x))
        x = self.fc2(x)
        return x
```

Fragment programu przygotowujący dane treningowe i definiujący model

```python
# Instantiate the model
model = IrisClassifier()

# Define the loss function and optimizer
criterion = nn.CrossEntropyLoss()
optimizer = optim.Adam(model.parameters(), lr=0.01)

# Training the model
epochs = 1000
for epoch in range(epochs):
    optimizer.zero_grad()
    outputs = model(X_train)
    loss = criterion(outputs, y_train)
    loss.backward()
    optimizer.step()
    if (epoch+1) % 100 == 0:
        print(f'Epoch [{epoch+1}/{epochs}], Loss: {loss.item():.4f}')

# Testing the model
with torch.no_grad():
    outputs = model(X_test)
    _, predicted = torch.max(outputs, 1)
    accuracy = accuracy_score(y_test.numpy(), predicted.numpy())
    print('Accuracy:', accuracy)
```

Fragment programu przeprowadzający procedurę treningową

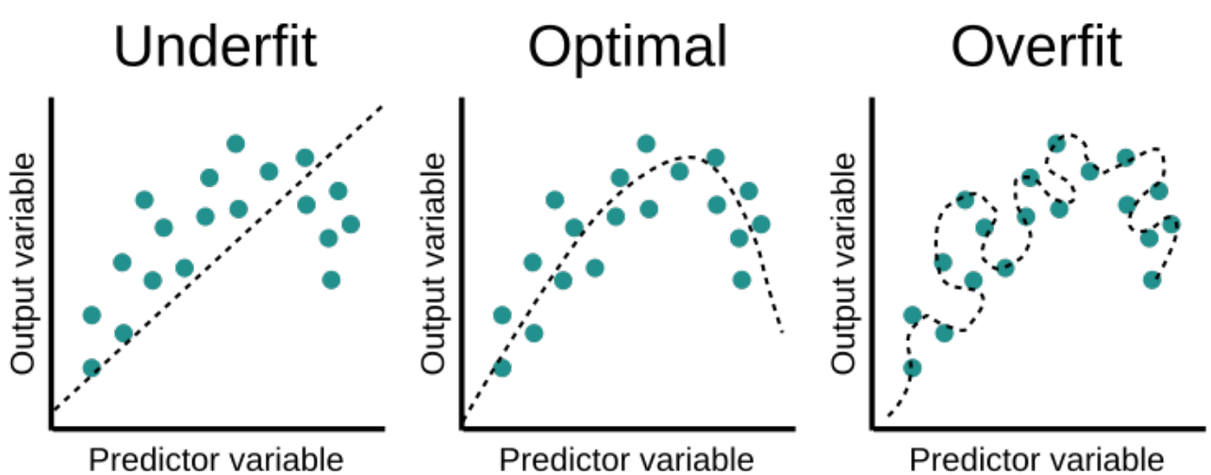
```
Epoch [100/1000], Loss: 0.1138
Epoch [200/1000], Loss: 0.0622
Epoch [300/1000], Loss: 0.0528
Epoch [400/1000], Loss: 0.0495
Epoch [500/1000], Loss: 0.0481
Epoch [600/1000], Loss: 0.0474
Epoch [700/1000], Loss: 0.0471
Epoch [800/1000], Loss: 0.0469
Epoch [900/1000], Loss: 0.0468
Epoch [1000/1000], Loss: 0.0468
Accuracy: 1.0
```

Wyjście procedury treningowej

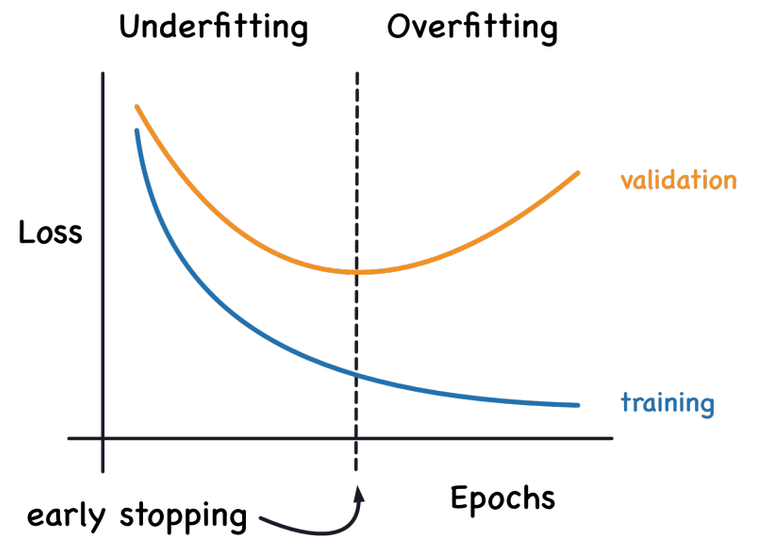
Problem generalizacji

Przeuczanie sieci neuronowych i walidacja

- Rzeczywiste problemy uczenia maszynowego są złożone, dane skomplikowane a modele mają duże rozmiary.
- W trakcie iteracyjnego procesu uczenia sieci neuronowej istnieje ryzyko, że **nadmiernie dopasuje się ona do zbioru treningowego**. Zbyt długa ekspozycja na dane treningowe (szczególnie złożonej sieci) może spowodować, że nauczy się ona wykrywać zjawiska charakterystyczne dla próbek treningowych, a nie dla ogólnego zjawiska.
- Zdolność sieci neuronowej do poprawnego działania na danych innych niż treningowe nazywamy generalizacją. Zjawisko, w którym sieć jest nadmiernie dopasowana do danych treningowych i **traci zdolność generalizacji nazywamy przetrenowaniem (overfitting)**.
- Najbardziej skrajny przypadek przetrenowania zachodzi gdy siec nauczy się dokładnie wszystkich próbek treningowych *na pamięć*.
- W celu testowania zdolności generalizacji każdy zbiór dzielimy na zbiór treningowy i testowy. **Zbiór testowy nie bierze udziału w optymalizacji modelu, a jedynie w określeniu jego zdolności generalizacji na końcu procesu tworzenia rozwiązania**. Jeżeli model działa dobrze na danych treningowych, a źle na testowych to jest przetrenowany.
- W celu uniknięcia przetrenowania ze zbioru treningowego wydziela się podzbiór walidacyjny. Nie bierze on bezpośrednio udziału w procesie optymalizacji, ale w każdej iteracji treningu jest używany do sprawdzania czy model nie traci zdolności generalizacji. **Jeżeli wyniki na zbiorze treningowym się poprawiaja, a na walidacyjnym pogarszają, należy przerwać procedurę treningową**. Ostatecznie model należy przetestować na danych testowych, aby sprawdzić pełną możliwość generalizacji modelu.



Wizualizacja ideowa działania przetrenowanego modelu [1]



Wizualizacja funkcji straty z przetrenowaniem [2]

[1] https://www.fastaireference.com/overfitting
[2] https://www.kaggle.com/code/ryanholbrook/overfitting-and-underfitting

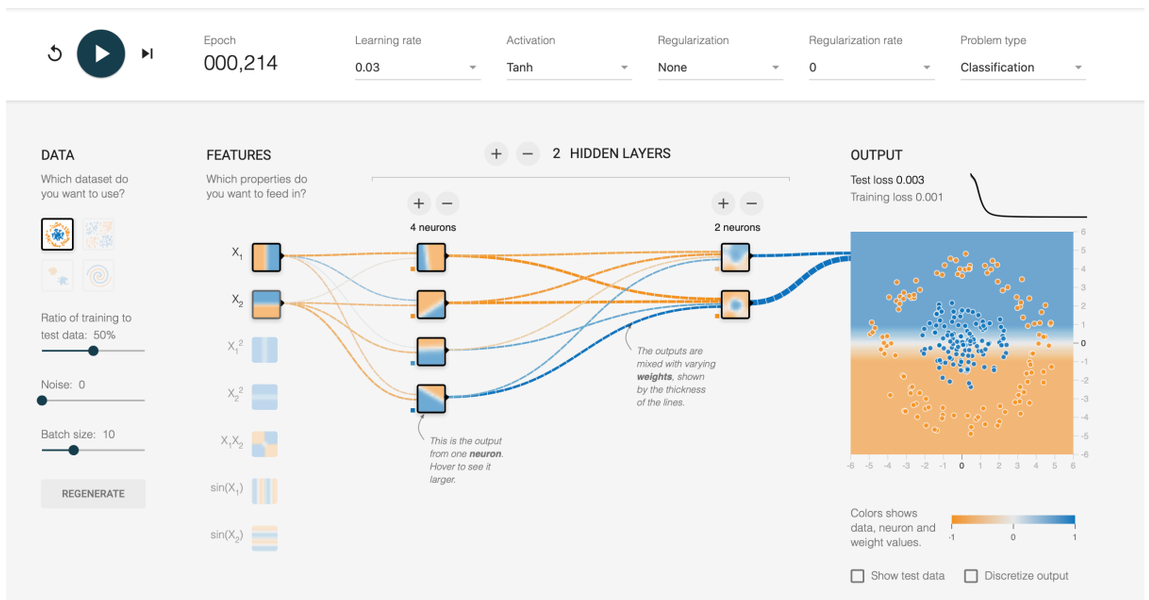
Interaktywne demo wielowarstwowego perceptronu

Do własnej eksploracji

- Na stronie https://playground.tensorflow.org można pobawić się treningiem perceptronu danych o różnym kształcie.
- Strona pokrywa wszystkie dotychczas omawiane zagadnienia:
  - architektura sieci – wiele neuronów w warstwie i wiele warstw w sieci,
  - zdolność sieci do modelowania złożonych funkcji z prostych neuronów,
  - wpływ funkcji aktywacji na zdolność sieci do modelowania nieliniowych zależności (kliknięcie *Activation* → *Linear* wyłącza nieliniowe funkcje aktywacji),
- wartość funkcji starty na zbiorę treningowym i walidacyjnym/testowym (zadanie: spróbować wywołać przeuczenie).

$$o_j = \phi\left(\sum_{n}^{i=1} x_{ij} wij + b_i\right)$$

Przypomnienie – wzór na wyjście komórki neuronowej



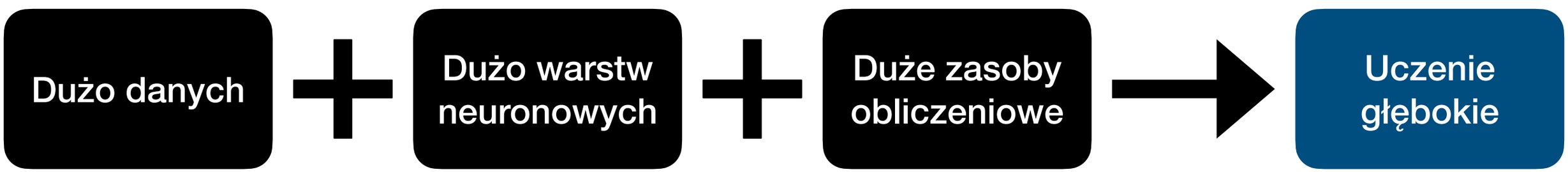
Panel na stronie Tensorflow Playground

Głębokie sieci neuronowe (tak zwana *sztuczna inteligencja)*

Głębokie sieci neuronowe

Fenomen uczenia głębokiego

- Tradycyjne uczenie maszynowe (wliczając w to proste sieci neuronowe typu perceptron) ma ograniczone zastosowania.
- Ewolucja sieci neuronowych w złożone struktury o wielu (często specjalnych) warstwach trenowanych na dużych zbiorach danych stworzyła nową dziedziną uczenia głębokiego.
- **Uczenie głębokie doprowadziło do przełomu w przetwarzaniu danych multimedialnych.**
- **Fenomen uczenia głębokiego leży nie w jakości lecz w ilości.**
- Sieci głębokie składają się z wielu warstw neuronów, często zawierają one wyspecjalizowane warstwy związane z konkretnym typem danych.



Uczenie głębokie

Uczenie głębokie vs. uczenie tradycyjne

- Uczenie głębokie zrewolucjonizowało przetwarzanie złożonych danych multimedialnych (tak zwane dane nieustrukturyzowane), których struktura wychodzi poza dane tabelaryczne (tak zwane dane ustrukturyzowane).

Uczenie tradycyjne	Uczenie głębokie
Modele oparte na własnościach statystycznych lub proste sieci perceptron	Sieci neuronowe o wielu złożonych warstwach
Różne sposoby uczenia modeli	Dominuje algorytm SGD
Skonwencjonalizowane i dobrze opracowane modele	Elastyczność w budowie architektur sieci neuronowych, emergentne cechy modeli
Stosunkowo małe zbiory danych	Duże i ogromne zbiory danych (co najmniej setki a najlepiej tysiące, próbek)
Małe wymagania obliczeniowe	Duże i ogromne wymagania obliczeniowe (RTX xx70 to minimum)
Zazwyczaj duży nacisk na analizę statystyczną danych	Szeroki przekrój danych zamiast ich analizy

Rodzaje sieci w uczeniu głębokim

Dane nieustrukturyzowane i złożone sieci

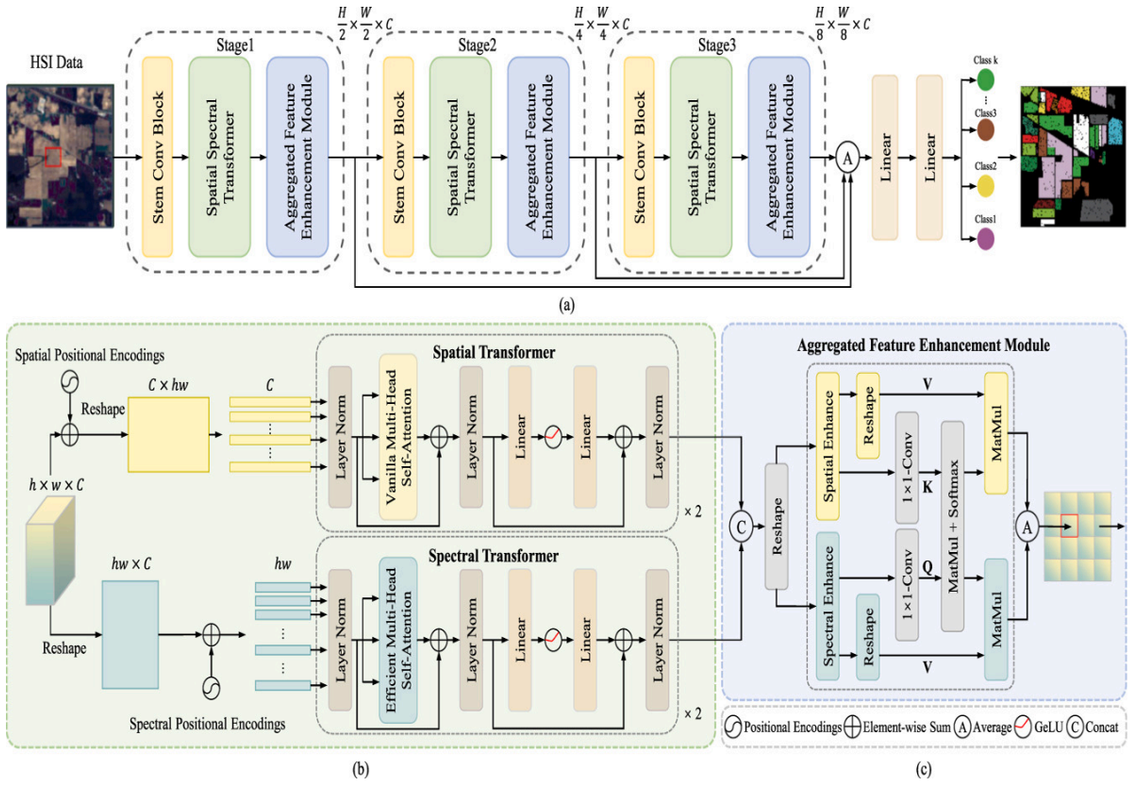
- Aby przetwarzać złożone dane multimedialne/ nieustrukturyzowane (obrazy, tekst, struktury chemiczne) potrzeba wyspecjalizowanych warstw neuronowych, a dane musza być poddane **wektoryzacji** – zamianie do postaci tensora (najlepiej zmiennoprzecinkowego).

- Rodzaje głębokich sieci neuronowych i ich danych docelowych*:

    1. **konwolucyjne** (dane obrazowe),
    2. **rekurencyjne** (dane sekwencyjne),
    3. sieci **transformer** (sieci GPT),
    4. sieci **grafowe** (grafy, np. modele chemiczne, sieci zależności).

- **Złożone sieci neuronowe w ogólności podlegają tym samym regułom co perceptron** (tensory, chain rule, SGD, funkcja straty, trening/walidacja/test, przeuczenie, itd.).

- Postaramy się przejrzeć wszystkie ważne typy sieci, ale możemy to zrobić bardzo powierzchownie.



Przykład złożonej sieci neuronowej [1]

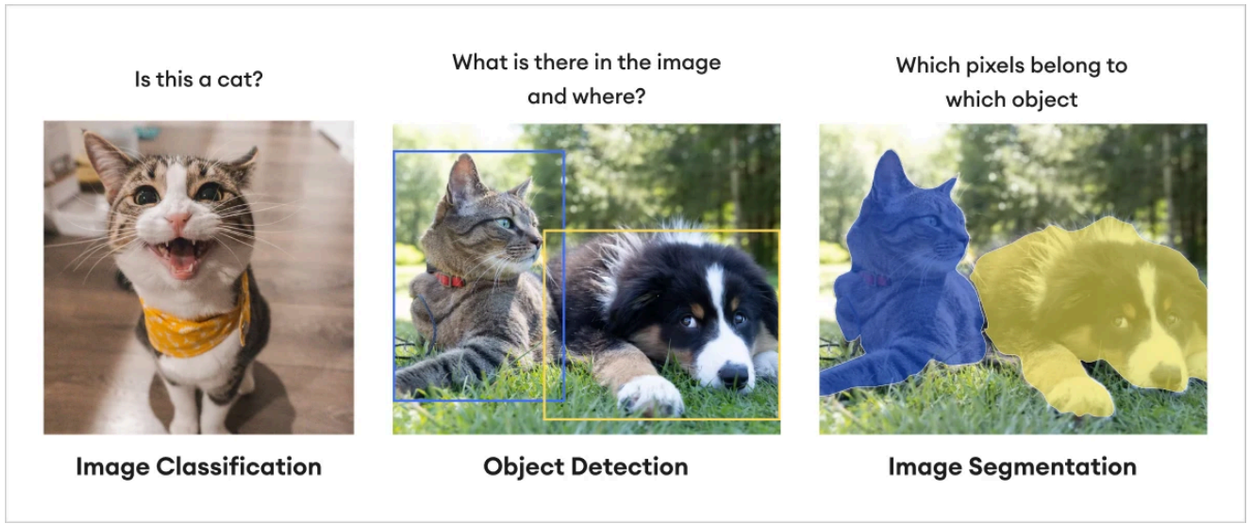
* Zaprezentowany podział jest bardzo zgrubny

[1] https://www.sciencedirect.com/science/article/pii/S1569843222001935

Sieci konwolucyjne

Przetwarzanie obrazów

- Sieci konwolucyjne **(convolutional neural network CNN)** były pierwszym typem sieci, który wyszedł poza ramy tradycyjnych perceptronów i szeroko zademonstrował rewolucyjny potencjał uczenia głębokiego (poczytaj więcej o historii zbioru ImageNet i związanych z nim konkursów).
- Z pomocą uczenia głębokiego i sieci (typowo) konwolucyjnych można realizować zadania takie jak:
    - **Klasyfikacja obrazu** (np. odróżnianie psów od kotów, wykrywanie chorych narządów na obrazach medycznych, rozróżnianie materiałów po strukturze powierzchni).
    - **Segmentacja obrazu** (np. wyróżnienie części obrazu zajętego przez klasę kota, zaznaczenie chorych tkanek w obrazowaniu medycznym USG, RTG, MRI itd. – inaczej klasyfikacja per piksel).
    - **Detekcja obiektów** (np. lokalizacja różnych obiektów, ich położenia i typu w ruchu drogowym – samochód, pieszy, rower, itd.).
    - **Generowanie obrazów** (np. zastępowanie obiektów tłem – *inpainting*, transfer stylu obrazu na zdjęcie).
    - **Określanie podobieństwa obrazów** (np. w celach marketingowych, podobieństwa materiałów, ludzi, itd.).
    - **Znajdowanie anomalii, punktów zainteresowania (ROI)** (np. w obrazach naukowych, wykrywanie wadliwych produktów przemysłowych).



Wizualizacja różnic między klasyfikacją, segmentacja i detekcją
https://www.superannotate.com/blog/image-segmentation-for-machine-learning

Sieci konwolucyjne

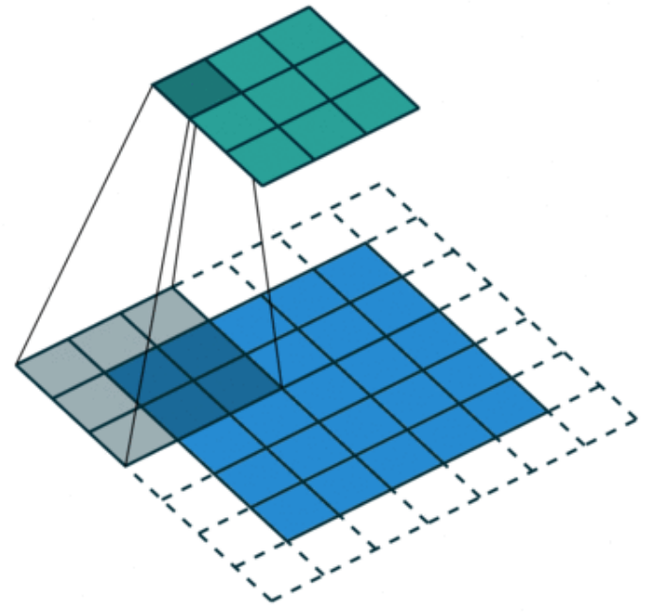
Warstwa konwolucyjna kontra perceptron

- Tradycyjny neuron ma sztywno określoną liczbę wejść, których zasięg jest globalny.

- Warstwa konwolucyjna jest modyfikacją perceptronu – wejścia neuronu operują na małym sąsiedztwie fragmentów obrazu wejściowego, a neuron jest poddany konwolucji z całym zdjęciem wejściowym.

- Warstwy konwolucyjne znajdują zastosowanie nie tylko w przetwarzaniu obrazu – są najlepszym wyborem tam, gdzie liczy się niezależne i dokładne przetwarzanie fragmentów większej całości, np. lokalnej struktury większego obiektu.

- W bibliotece PyTorch warstwa typowa warstwa konwolucyjna to *Conv2D* https://pytorch.org/docs/stable/generated/torch.nn.Conv2d.html.



Wizualizacja działania okna konwolucji [1]

[1] https://hannibunny.github.io/mlbook/neuralnetworks/convolutionDemos.html

Architektura sieci konwolucyjnych

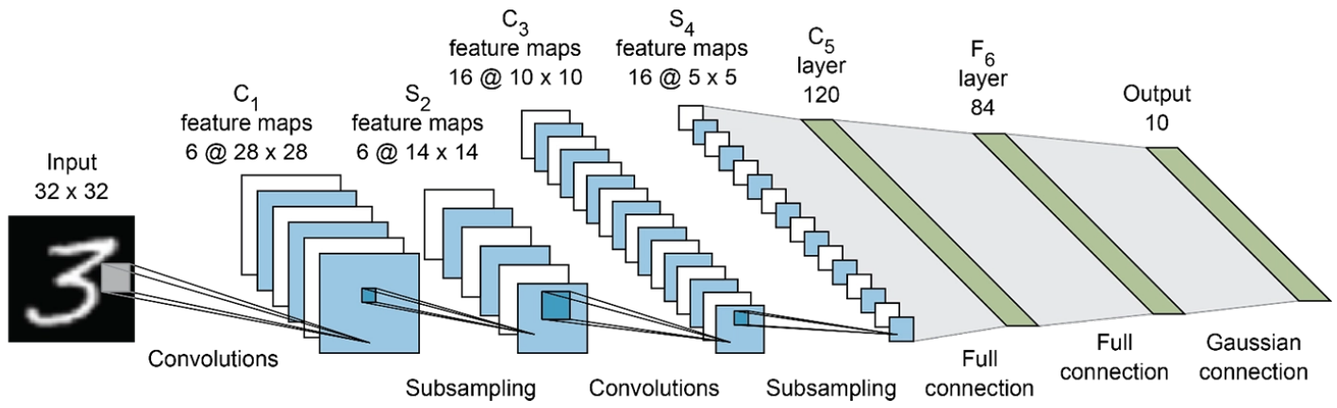
Idea modułu enkoder

- Przygotowanie obrazów do postaci wejściowej do sieci jest proste, ponieważ obrazy z natury są macierzami (pikseli) – jedyny konieczny zabieg to zadbanie o poprawne kodowanie wartości zmiennoprzecinkowych (najlepiej w zakresie 0.0–1.0).

- W sieciach konwolucyjnych warstwy konwolucyjne ułożone są tak, że wraz z przetwarzaniem pomniejszany jest rozmiar przestrzenny obrazu, natomiast powiększany jest wymiar *głębokości* obrazu, w którym gromadzone są abstrakcyjne informacje.

- Zdecydowana większość sieci neuronowe (lub ich fragmenty) działają jak kompresory danych z podanej formy do abstrakcyjnej postaci rozumianej wewnętrznie jedynie przez sieć neuronową. Taki moduł ma na schemacie kształt **zwężającego się lejka** (w wymiarze zrozumiałym dla człowieka) i nosi nazwę **enkodera** – koduje dane do postaci **niejawnej (latent space)**.

- **Postać latentna może być użyta do analizy** przez perceptron w celu wykonania zadania np. klasyfikacji lub regresji, albo rozkodowana do innej postaci, kiedy sieć ma służyć do generowania innych obrazów.



Sieć konwolucyjna do detekcji cyfr – model typu enkoder
https://www.superannotate.com/blog/guide-to-convolutional-neural-networks

Sieci konwolucyjne generujące obrazy

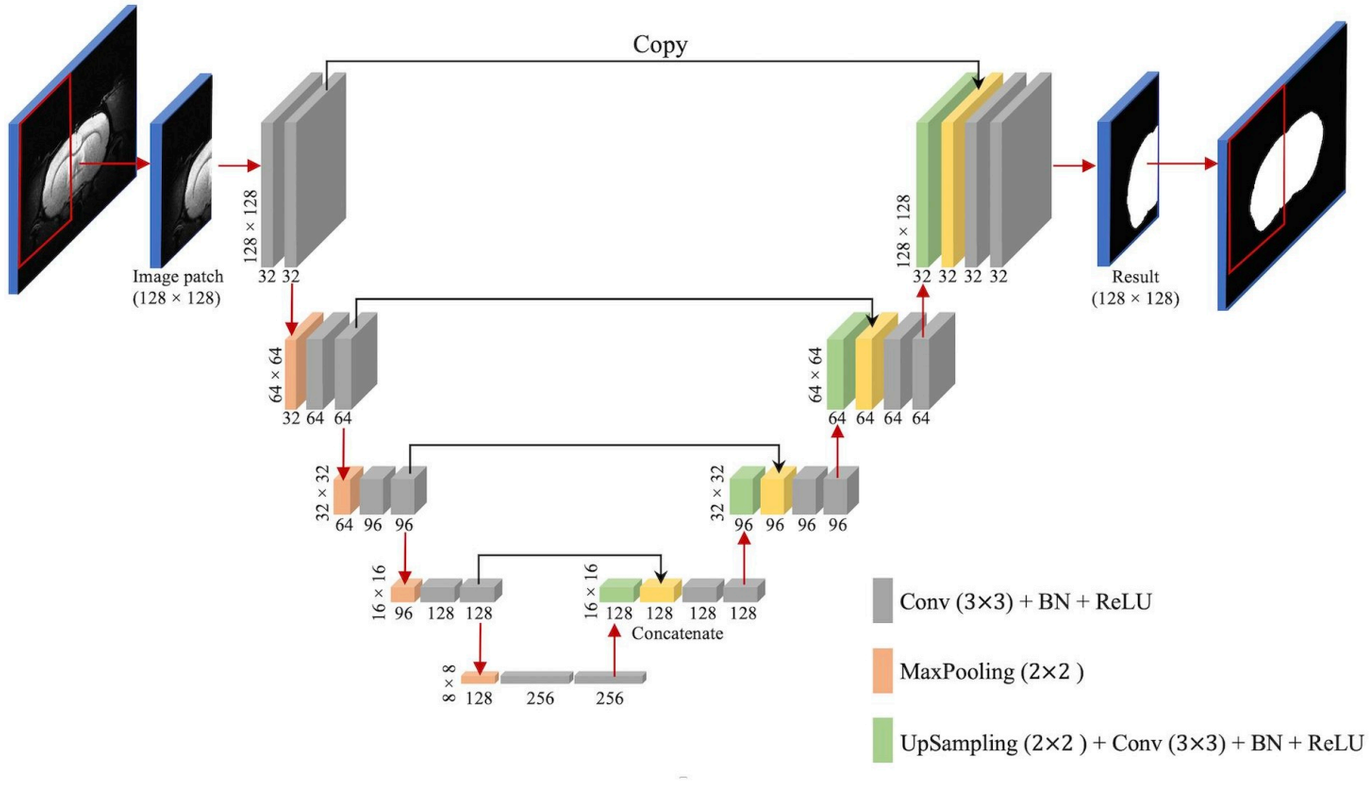
Idea modułu enkoder-dekoder

- Enkoder koduje dane do postaci abstrakcyjnej, jeśli sieć ma zwrócić przetworzony obraz, postać *latentnej* musi być rozkodowana do innej reprezentacji.

- Jeśli chcemy wygenerować mapę chorych tkanek, moduł enkodera musi zakodować obraz medyczny do abstrakcyjnej postaci, a potem moduł **dekodera** musi rozkodować postać abstrakcyjna do innego obrazu – mapy choroby.

- Sieć enkoder-dekoder jest powtarzającym się wzorcem w wielu rodzajach modeli (nie tylko obrazowych i konwolucyjnych). Jej schemat ma kształt klepsydry.

- Sieci do segmentacji bazują na modelu **UNet**, który jest modyfikacją architektury enkoder-dekoder z dodatkowymi połączeniami, które noszą nazwę rezydualnych.



Sieć typu UNet do segmentacji medycznej [1]

[1] Źródło

Sieci rekurencyjne

Przetwarzanie języka naturalnego i szeregów czasowych

- Dane w postaci **szeregów danych** stanowią istotne pole aplikacji modeli uczenia głębokiego; np. szeregi danych pomiarowych, telemetria, a także dane w postaci języka naturalnego (mówionego) oraz języków opisu zjawisk fizyko-chemicznych (język opisu substancji chemicznych).

- Dane pomiarowe w postaci **ciągów liczb są zdatne do przetwarzania przez modele uczenia głębokiego** bez aplikowania złożonych przekształceń.

- W przeciwieństwie do danych obrazowych, które ze swojej natury są macierzami zdatnymi do uczenia maszynowego bez złożonych modyfikacji, **dane tekstowe wymagają specjalnego przetwarzania wstępnego**.

- Komputer koduje tekst w postaci ASCII/UTF-8, gdzie **każdy znak (np. litera) jest wewnętrznie reprezentowany jako liczba** (np. w ASCII „A” to 41).

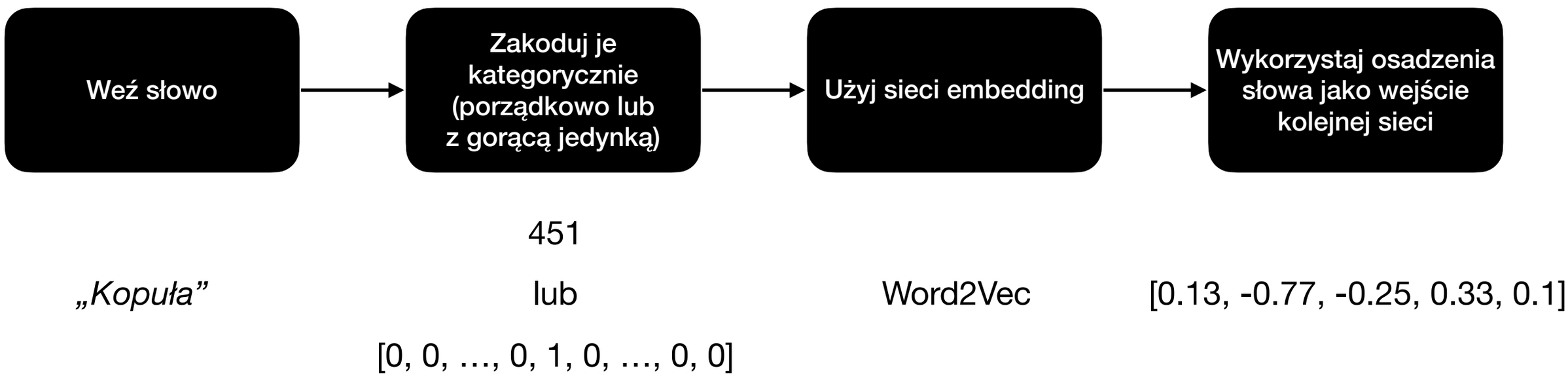
- **Modele do przetwarzania języka nie operują na literach, a na tokenach (tzn. słowach)** (podobnie jak ludzie składają słowa z liter, ale sens zdania budują ze słów, które znają). Tekst analizowany przez sztuczną inteligencję **musi zostać poddany wektoryzacji tzn. słowa muszą zostać zakodowane do *latentnych* wektorów znaczeń (word embedding)**.

Sieci rekurencyjne

Osadzenia języka

- Istnieje dużo metod budowania osadzeń słów/tokenów (kodowania słów do postaci wektorów), jako domyślną (nie najlepszą, ale dobrą i popularną) metodę można uznać podejście z modelami typu **Word2Vec** – sieciami neuronowymi do zamiany słów kodowanych kategorycznie do postaci wektorów znaczeń.

- Sieci **word2vec wytrenowane dla konkretnych zastosowań (np. język polski, albo pierwiastki chemiczne) są gotowe i dostępne do ogólnego użytku**. Własne sieci osadzeń można budować za pomocą warstw takich jak *Embedding* w PyTorch (https://pytorch.org/docs/stable/generated/torch.nn.Embedding.html).



Schemat użycia sieci do wektoryzacji słów

Osadzenia języka

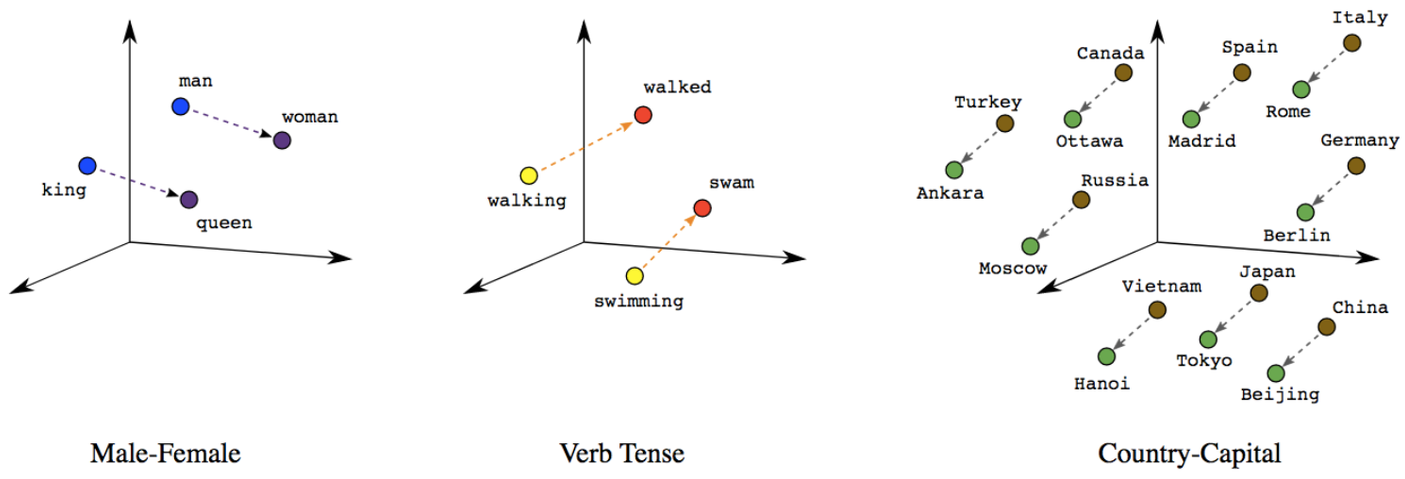
Demonstracja

```python
import torch
from torchtext.vocab import FastText

# Load pretrained word vectors
word_vectors = FastText()

# Sample word
word = 'example'

# Encode the word
embedding_vector = word_vectors[word]

# Print the embedding vector
print(embedding_vector)
```

Kod wykorzystujący gotową sieć osadzeń do wektoryzacji słowa „example” (wagi modelu zajmują ponad 6.5 GB)

```
tensor([ 0.0144,  0.1337, -0.1489,  0.0887, -0.1557,  0.2726, -0.1041, -0.3372,
         0.0417,  0.1062,  0.1022, -0.0453, -0.1542, -0.1053,  0.0697, -0.1869,
        -0.1088,  0.2406,  0.1998,  0.0595, -0.0025,  0.1973, -0.0408, -0.2250,
        -0.0567, -0.2271,  0.0972,  0.0579, -0.2189,  0.0471, -0.0033,  0.1534,
        -0.0562,  0.2991,  0.0233, -0.0728,  0.0187,  0.2056, -0.0789,  0.0379,
        -0.0765, -0.0664, -0.0558, -0.1638,  0.0485,  0.0996,  0.0361, -0.1286,
         0.0232, -0.0946,  0.1334, -0.0198,  0.0813, -0.0804, -0.0900,  0.0593,
         0.2583,  0.0009,  0.0832,  0.1229,  0.0440, -0.1987,  0.1704, -0.2049,
         0.0016,  0.2148,  0.0618, -0.2408,  0.1432, -0.1245, -0.1967,  0.1754,
         0.1480, -0.0399, -0.2774,  0.1578,  0.3308,  0.0178,  0.0667,  0.0173,
         0.3053,  0.1277, -0.1219, -0.1543,  0.0535,  0.0126,  0.0023,  0.1098,
        -0.1017, -0.0724,  0.2588,  0.0406,  0.4015, -0.3056,  0.1164, -0.2622,
         0.1799,  0.2706, -0.0886, -0.1797, -0.1471, -0.2946,  0.0117, -0.0610,
        -0.2033, -0.0645,  0.0095,  0.0616,  0.1442,  0.0520, -0.0088,  0.0486,
        -0.3130, -0.0474, -0.0401, -0.0084,  0.0647,  0.0105, -0.0828,  0.0216,
         0.2487,  0.1871, -0.0116,  0.3250, -0.0615,  0.1366,  0.1154, -0.0725,
        -0.0096,  0.2809,  0.0968,  0.0528, -0.0595, -0.0361, -0.0758,  0.0090,
        -0.0470, -0.0319,  0.1222,  0.1935,  0.0965,  0.0186,  0.2381, -0.0009,
         0.0236,  0.1260, -0.0808, -0.1309, -0.2132,  0.0318, -0.0931, -0.3316,
         0.0198, -0.3221, -0.0666, -0.0333,  0.1107,  0.0120,  0.1428, -0.0606,
        -0.2131,  0.2728, -0.2364, -0.2447, -0.0173, -0.0681,  0.0105, -0.2829,
        -0.0765, -0.0140, -0.3232, -0.1974, -0.0302,  0.0424, -0.0288,  0.1261,
         0.0010,  0.1209,  0.0822, -0.0941, -0.0154, -0.0119,  0.1039,  0.0890,
        -0.1505,  0.0696, -0.0875, -0.2301,  0.1016, -0.0831,  0.0901, -0.3297,
        -0.0358, -0.0262,  0.0174, -0.2262,  0.1110,  0.1466, -0.2466,  0.0669,
         0.1420, -0.1002,  0.1251, -0.0419,  0.2271, -0.0396, -0.1309, -0.0226,
        -0.0844,  0.0776,  0.3040,  0.1906,  0.0999,  0.0694,  0.2593, -0.1199,
        -0.0009, -0.1772, -0.3719,  0.0553, -0.2080, -0.1086, -0.1272,  0.0024,
         0.0174,  0.1825, -0.1258, -0.2401,  0.0277, -0.0983, -0.0445, -0.0208,
        -0.1330, -0.1748, -0.2140,  0.0474, -0.0374, -0.2243,  0.1562,  0.0592,
         0.0849, -0.0859,  0.1303, -0.1565,  0.0405,  0.1615,  0.1747,  0.0838,
        -0.0306,  0.0365,  0.1126, -0.1067, -0.1675,  0.1395,  0.0307, -0.2114,
         0.0537,  0.0747, -0.0853,  0.0831, -0.1252,  0.1119, -0.1096,  0.2488,
        -0.3404, -0.2142, -0.1687, -0.0764, -0.1123,  0.1380, -0.1585, -0.1414,
         0.0978, -0.1022, -0.0050, -0.1105,  0.1088,  0.0665, -0.0778,  0.0314,
         0.1988,  0.0537, -0.0767,  0.1467,  0.1651, -0.2075,  0.1049,  0.1471,
        -0.2238,  0.1961,  0.0190, -0.0388,  0.1029,  0.0275, -0.2925, -0.2639,
         0.0005, -0.0202,  0.0657, -0.0029])
```

Efekt działania przykładowego kodu do wektoryzacji słowa „example”

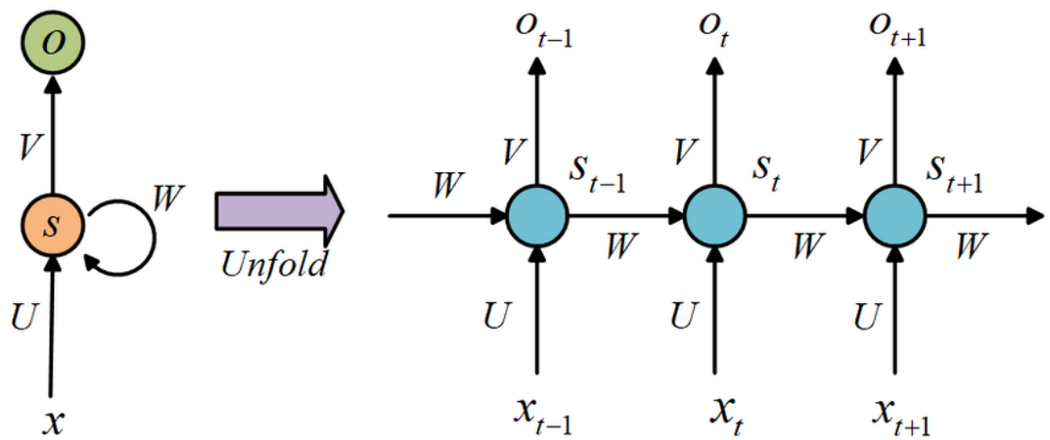


Ilustracja podobieństwa osadzeń słów w przestrzeni cech w różnych przekrojach
https://towardsdatascience.com/a-guide-to-word-embeddings-8a23817ab60f

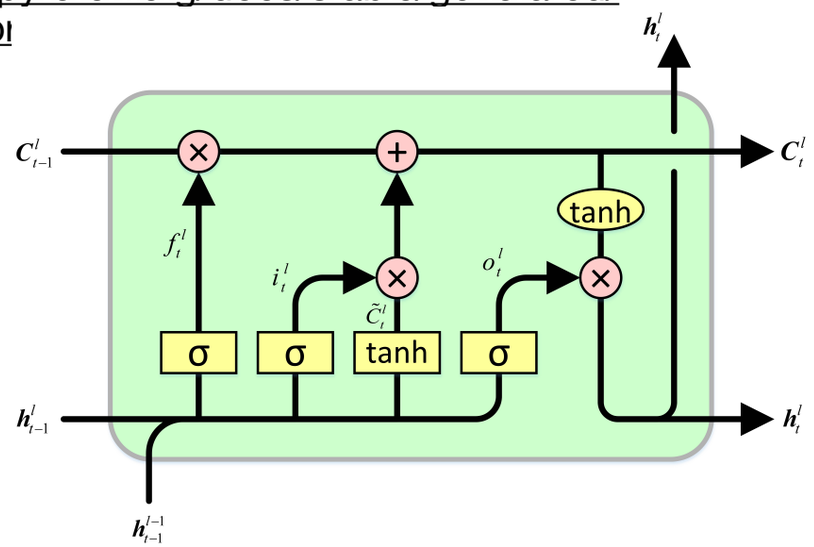
Warstwy rekurencyjne

Przetwarzanie osadzeń języka naturalnego i ciągów danych

- W przypadku gdy dane są ciągiem liczb, bądź tekstem sprowadzonym do postaci ciągu wektorów osadzeń, możemy użyć na nich **sieci rekurencyjnych RNN (recurrent neural networks)**.
- Sieci rekurencyjne to moduły neuronowe **posiadające stan (pamięć) budowany na podstawie historii wejść** (ale nie jest to pamięć rozumiana tak jak pamięć ludzka – stan komórek jest budowany dla każdego tekstu wejściowego, a nie w ogólnym sensie).
- Warstwy rekurencyjne przyjmują na swoje wejście **kolejne elementy ciągu wejściowego i wypracowują w pamięci jego abstrakcyjną reprezentację**.
- **Współcześnie używa się zaawansowanych implementacji sieci rekurencyjnych typu LSTM (long-short-term memory) lub GRU (gated rectified unit)**.
- Warstwy rekurencyjne w PyTorch są zaimplementowane jako *LSTM* https://pytorch.org/docs/stable/generated/torch.nn.LSTM.html oraz *GRU* https://pytorch.org/docs/stable/generated/to



Ogólny schemat działania komórki RNN, w formie wizualizacji ze sprzężeniem zwrotnym i w postaci rozwiniętej [1]



Schemat komórki LSTM o wejściu *h* oraz stanie ukrytym komórki *C*

[1] Źródło

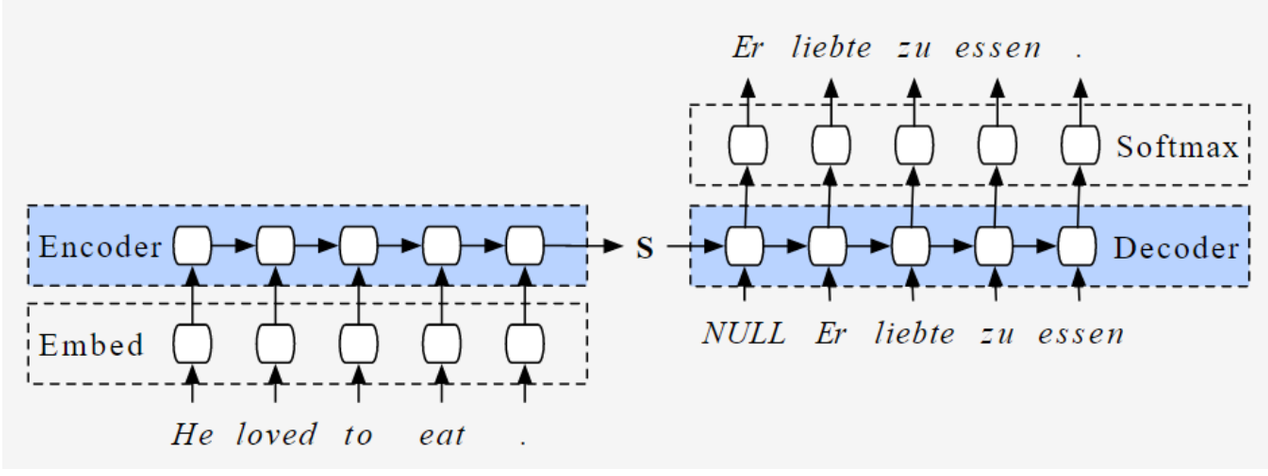
Sieci rekurencyjne

Budowanie sieci rekurencyjnych

- Warstwy LSTM/GRU podelgają takim samym zasadom jak zwykłe neurony i sieci konwolucyjne, tzn. układamy je w warstwy, a jedna warstwa może zawierać równoległe komórki LSTM.

- Podobnie jak w przypadku sieci konwolucyjnych, sieci **LSTM można wykorzystać do budowy enkoderów**, które mają kształt *lejka*, zwężają szereg tokenów do postaci mniejszego wektora stanu ukrytego w pamięci komórki, która stanowi przestrzeń latentną. Wektor w tej przestrzeni można poddać analizie przy pomocy perceptronu w celu klasyfikacji (np. analizy sentymentu) lub regresji (estymacji oceny produktu na podstawie opisu w skali 0.0 do 1.0).

- **Dla sieci RNN możemy także budować modele typu enkoder-dekoder** w celu zamiany jednej sekwencji w drugą, np. do tłumaczenia maszynowego języków. Typowym modelem tego typu jest architektura Seq2seq.



Schemat architektury Seq2seq do tłumaczenia maszynowego (dowolny typ komórki, forma rozwinięta)
https://www.analyticsvidhya.com/blog/2020/08/a-simple-introduction-to-sequence-to-sequence-models/

Warstwa attention

Budowanie sieci rekurencyjnych

- Sieci RNN ograniczone są przez rozmiar ich pamięci – przy przetwarzaniu długiego tekstu gubią główny wątek przetwarzanych danych.

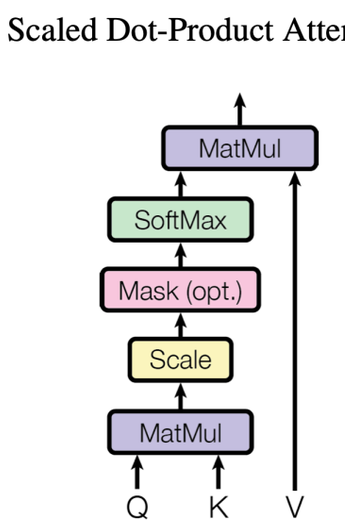
- Odpowiedzią na ten problem i najnowszą rewolucją w świecie sieci neuronowych – warstwa attention, która jest głównym budulcem np. modeli GPT.

- Warstwa attention ma za zadanie wyszukiwanie dalekich i złożonych zależności oraz skojarzeń w dużych zestawach danych.

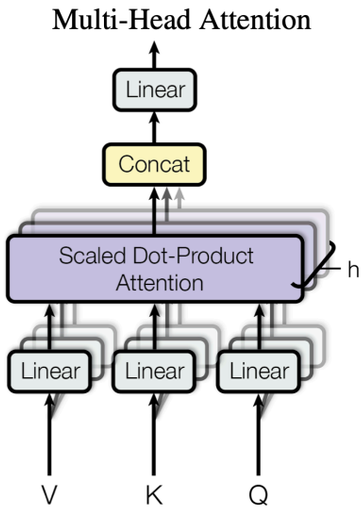
- Ta zaawansowana operacja jest realizowana wyłącznie poprzez użycie prostych perceptronów i odpowiednie mnożenie trzech macierzy: Q (query/zapytanie), K (key/klucz), V (value/wartość).

Warstwa attention

Mechanizm analizy złożonych powiązań



Schemat działania mechanizmu attention [1]



Schemat warstwy attention z wagami projekcji liniowych [1]

[1] https://arxiv.org/pdf/1706.03762.pdf

Sieci Transformer

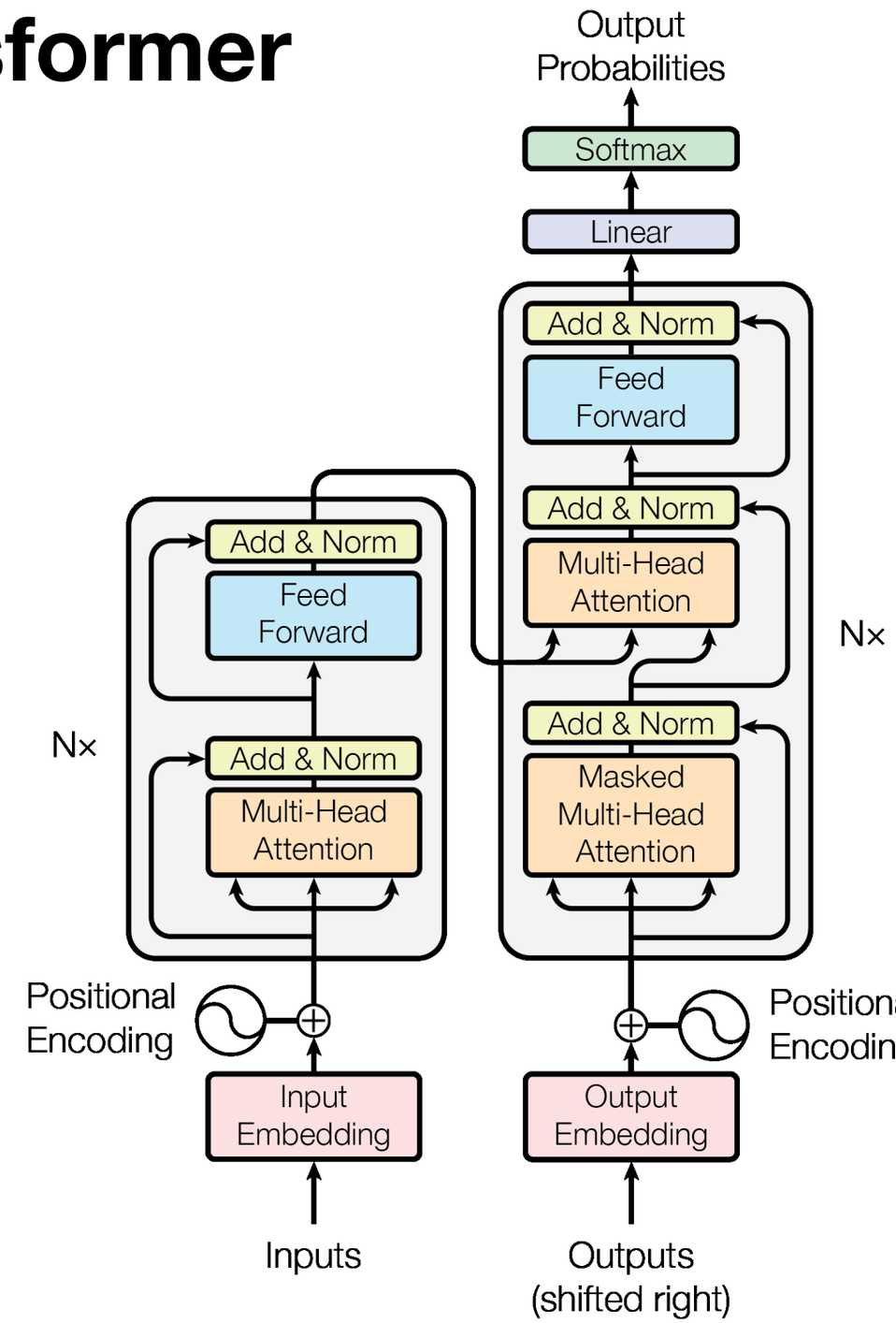
Architektura skojarzeń Transformer

- **Najczęściej warstw attention używa się do budowy architektur Transformer**. Warstwa attention we współczesnej formie oraz architektura Transformer zostały zaproponowane w 2017 roku w artykule *Attention Is All You Need* (współautorstwo Łukasz Kaiser Uniwersytet Wrocławski).

- Transformer to moduł enkoder-dekoder złożony z warstw attention.

- **Modele GPT (Generative Pre-trained Transformer), które są podstawą Chatu GPT są złożone z bloków Transformer**. Moduły Transformer mogą być użyte dla dowolnych danych, gdzie ważne jest znajdywanie powiązań o *dalekim zasięgu*. Duże modele Transformer do przetwarzania języka trenowana na ogromnych zestawach danych i z użyciem potężnej architektury noszą nazwę **Large Language Models (LLM)**.

- Ciekawa wizualizacja sieci LLM: https://bbycroft.net/llm.



Schemat modułu Transformer [1]

[1] https://arxiv.org/pdf/1706.03762.pdf

Sieci LLM

Ograniczenia obecnych technologii dużych modeli językowych

- Warstwy attention i sieci Transformer podlegają tym samym regułom co wszystkie inne sieci neuronowe: często układa się je w schemacie enkoder, enkoder-dekoder, wejścia się wektoryzuje, uczenie następuje w sposób nadzorowany, obowiązuje problematyka przeuczenia sieci itd.

- Sieci LLM to modele zmieniające tekst w tekst, realizują podobne zadania jak sieci LSTM/GRU, ale w inny sposób (lepszy). Transformery opierają się na prostych warstwach neuronowych i mnożeniu macierzy. Próbki treningowe dla treningów sieci LLM są konstruowane z rzeczywistych tekstów, które są dzielone na fragmenty, a sieć uczy się przewidywać następny token (słowo) po danym fragmencie treningowym.

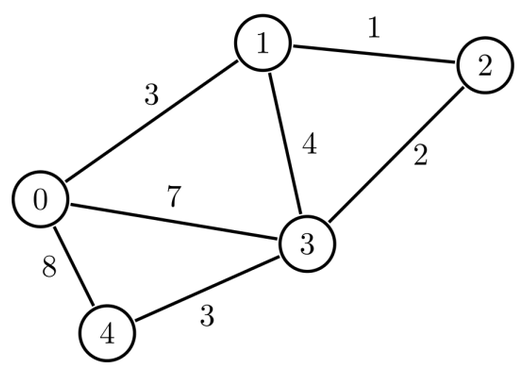
- Sieci LLM nie posiadają decyzyjności, kontynuują jedynie tekst wprowadzony przez użytkownika.

- Sieci LLM nie aktualizują swojej wiedzy na podstawie tekstu wejściowego. Jeśli użytkownik poda nowe informacje to model językowy wygeneruje tekst, który będzie z nimi spójny, ale nie nauczy się niczego nowego.

Dane grafowe

Struktury danych do reprezentacji złożonych struktur i zależności

- **Grafy to struktury danych pozwalające na modelowanie sieci zależności** (np. sieci społecznościowych, struktur chemicznych, sieci połączeń komunikacyjnych, sieci przepływów, itd.).
- **Graf składa się z zestawu wierzchołków/komórek (vertices/nodes) połączonych krawędziami (edges/links/connections).**
- **Zarówno wierzchołki jak i krawędzie mogą mieć przypisane wartości**. Grafy mogą mieć wiele rodzajów i szczególnych cech (np. połączenia mogą być jedno lub dwukierunkowe).
- Przykłady grafów:
  - Grafy reprezentujące sieci miast, gdzie wagi wierzchołków to odległości w kilometrach.
  - Struktury chemiczne, gdzie wartości wierzchołków reprezentują współrzędne pierwiastka w układzie okresowym, a wartości krawędzi kodują kategorycznie typ wiązania (kowalencyjne vs. jonowe).
  - Sieci społecznościowe, gdzie wartości wierzchołków opisują cechy osoby (wiek, kategoria płci, kategorie zainteresowań, kategoria kraju zamieszkania), a wartości krawędzi ilość wiadomości wysłanych do innego użytkownika.

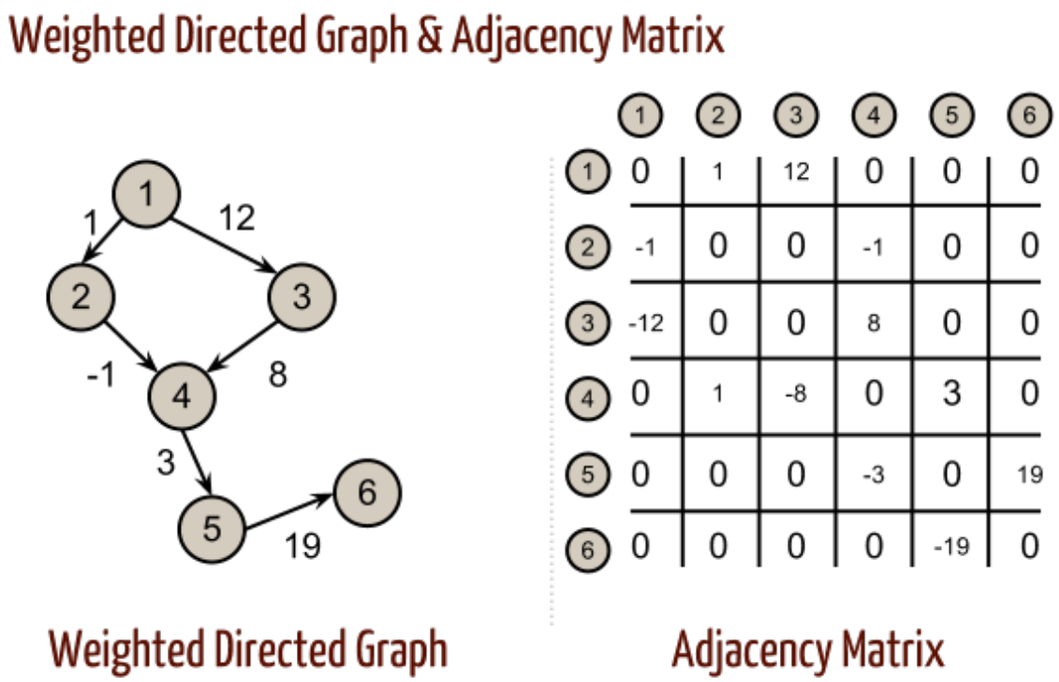


Przykładowy graf o pięciu wierzchołkach i siedmiu wierzchołkach
https://hyperskill.org/learn/step/5645

Wektoryzacja grafów

Reprezentacja liczbowa struktur grafowych

- **Grafy, podobnie jak wszystkie inne typy danych, należy zwektoryzować** w celu użycia ich z algorytmami uczenia maszynowego.
- Istnieje wiele reprezentacji liczbowych grafów, najpopularniejsze to **macierz sąsiedztwa (adjacency matrix) i lista sąsiedztwa (adjacency list)**.
- W ramach demonstracji skupimy się na postaci macierzowej, gdzie **wiersze/kolumny reprezentują wierzchołki, a wartości na ich przecięciu wagi połączeń** (gdzie 0 to brak połączenia).



Reprezentacja grafu w postaci macierzy sąsiedztwa
https://hyperskill.org/learn/step/5645

Grafowe sieci neuronowe

Reprezentacja liczbowa struktur grafowych

- Sieci grafowe są potężnym i ciekawym narzędziem, ale stanowią pewną niszę świata uczenia maszynowego. Istnieje wiele rozwiązań i specyficznych sieci grafowych, ale **są one mniej ustandaryzowane niż w przypadku sieci do przetwarzania tekstu/obrazów**. Potencjał sieci grafowych jest również mniej zbadany i wykorzystany w mniejszym stopniu niż dla bardziej standardowych sieci.

- **Elementy grafów (wierzchołki/krawędzie) oraz całe grafy mogą mieć przypisane wartości takie jak inne dane uczenia maszynowego, czyli wektory wartości ciągłych albo kategorycznych**. Sieci neuronowe mogą uczyć się przewidywać cechy elementów grafów (wartości liczbowe/kategoryczne wierzchołków/krawędzi/całych grafów).

- Sieci grafowe mają swoje odpowiedniki standardowych warstw sieci neuronowych:

  - **perceptronu** – podstawowej sieci o sztywno ustalonych rozmiarach grafu wejściowego,

  - **konwolucyjnej warstwy grafowej (GCNN)** – zdolnej do lokalnego przetwarzania fragmentów grafów,

  - **atencji grafowej** – zdolnej do analizy skojarzeniowej odległych zależności.

- **Sieci grafowe podlegają tym samym regułom co standardowe sieci neuronowe**. Cechy grafów można poddać mechanizmowi osadzeń i kodować wierzchołki/krawędzie w przestrzeniach cech. Warstwy grafowe można łączyć w schematach enkoder i enkoder/dekoder.

- Demonstracja sieci grafowych dla danych chemicznych: https://distill.pub/2021/gnn-intro/.

Podsumowanie

Pominięte tematy

Istotne problemy, których nie poruszono w prezentacji

- Prezentacja miała charakter przekrojowy, ale kosztem tego jak dogłębnie omówiono zagadnienia.
- Terminy związane z pominiętymi zagadnieniami:
  - mnogość tradycyjnych modeli uczenia tradycyjnego i metod redukcji wymiarowości,
  - metody uczenia nienadzorowanego,
  - **transfer learning** – przenoszenie wytrenowanych sieci do nowych zastosowań (zero-shot, one-shot, few-shot),
  - specyficzne, ale bardzo przydatne modele uczenia głębokiego do **zastosowań generatywnych** typu **GAN (sieci adwersaryjne)** oraz **stable diffusion**,
  - optymalizacja parametrów sieci (**fine-tuning**),
  - metody oceny działania modeli uczenia maszynowego,
  - uczenie przez wzmacnianie.

Perspektywy rozwoju

Uczenie maszynowe i głębokie dla zastosowań naukowych

- Obecnie tradycyjne metody uczenia maszynowego są stosowane z zastosowaniach biznesowych dla danych tablarycznych.
- Uczenie głębokie powoduje rewolucję w wielu dziedzinach życia, jednak największą popularność osiągają rozwiązania do zastosowań *miękkich* i generatywnych (stable diffusion – generowanie obrazów i Chat GPT – generowanie tekstu).
- Uczenie głębokie jest również z powodzeniem stosowane w nauce i zastosowaniach technicznych, ale jego adopcja jest wolniejsza niż w przypadku zastosowań *miękkich*. Jest tak dlatego, że:
    - dane reprezentujące zjawiska naukowe lub specyficzne środowiska inżynieryjne są droższe w pozyskaniu, szczególnie w dużej ilości,
    - mniejsza baza użytkowników naukowych i inżynieryjnych powoduje, że mniej osób jest zainteresowanych rozwojem specyficznych rozwiązań i mniej osób wie o potencjalnych problemach, w których może pomóc sztuczna inteligencja.
    - Naukowcy domenowi wiedza mało o uczeniu maszynowym, a inżynierowie/naukowcy uczenia maszynowego mało wiedzą o domenach zastosowań naukowych.
- W czym uczenie maszynowe/głębokie może pomóc w badaniach nauk ścisłych:
    - Modelowanie (bardzo złożonych) zależności fizycznych na podstawie pomiarów empirycznych.
    - Optymalizacja przestrzeni poszukiwań badań na podstawie wykonanych eksperymentów (sieć neuronowa uczy się z przykładów symulować eksperymenty i pomaga znaleźć obiecujące pomiary do wykonania).
    - Automatyzacja procesów badawczych, analiza danych pomiarowych (obrazów, serii pomiarowych itd.).
    - Fuzja i analiza bardzo złożonych danych pomiarowych, poszukiwanie nieoczywistych zależności w złożonych danych.
    - Budowa (pół)autonomicznych maszyn/robotów działających we współpracy z pewnymi zjawiskami chemicznymi.

Dziękuję za uwagę