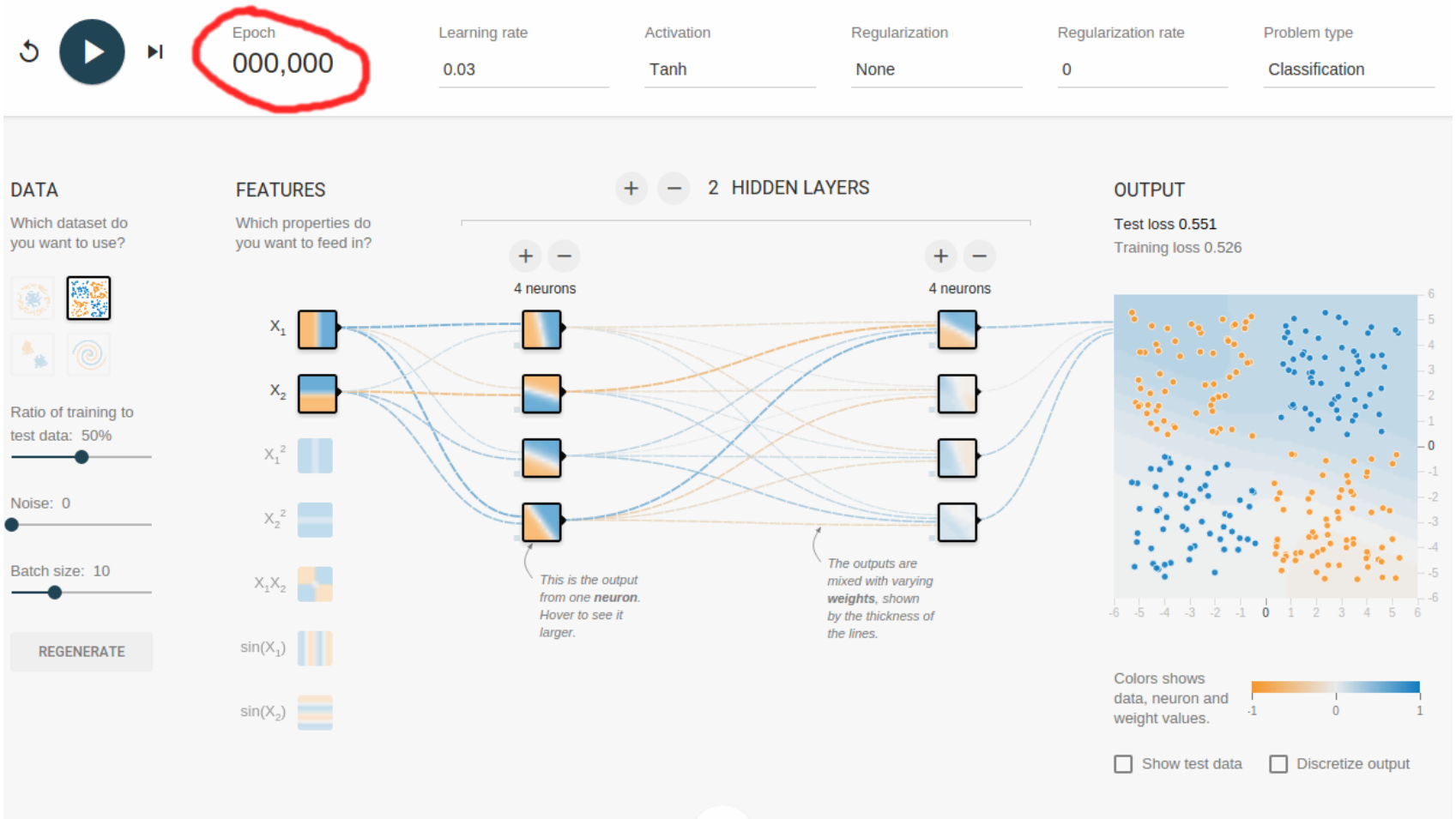
Uczenie maszynowe, różne warianty

Paweł Rychlikowski

Instytut Informatyki UWr

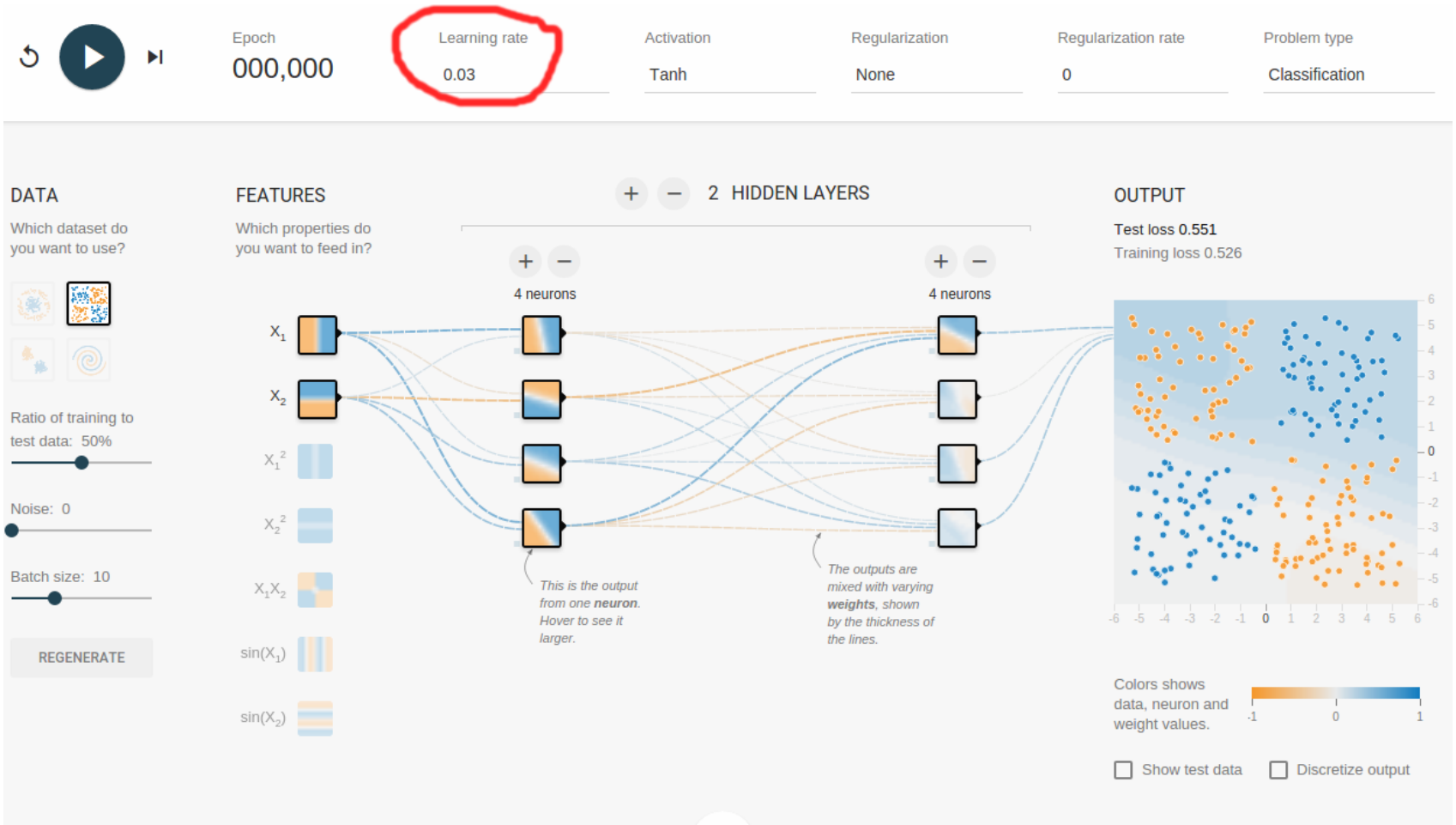
18 maja 2023

Plac zabaw dla tensorflow. Ważne pojęcia (1)



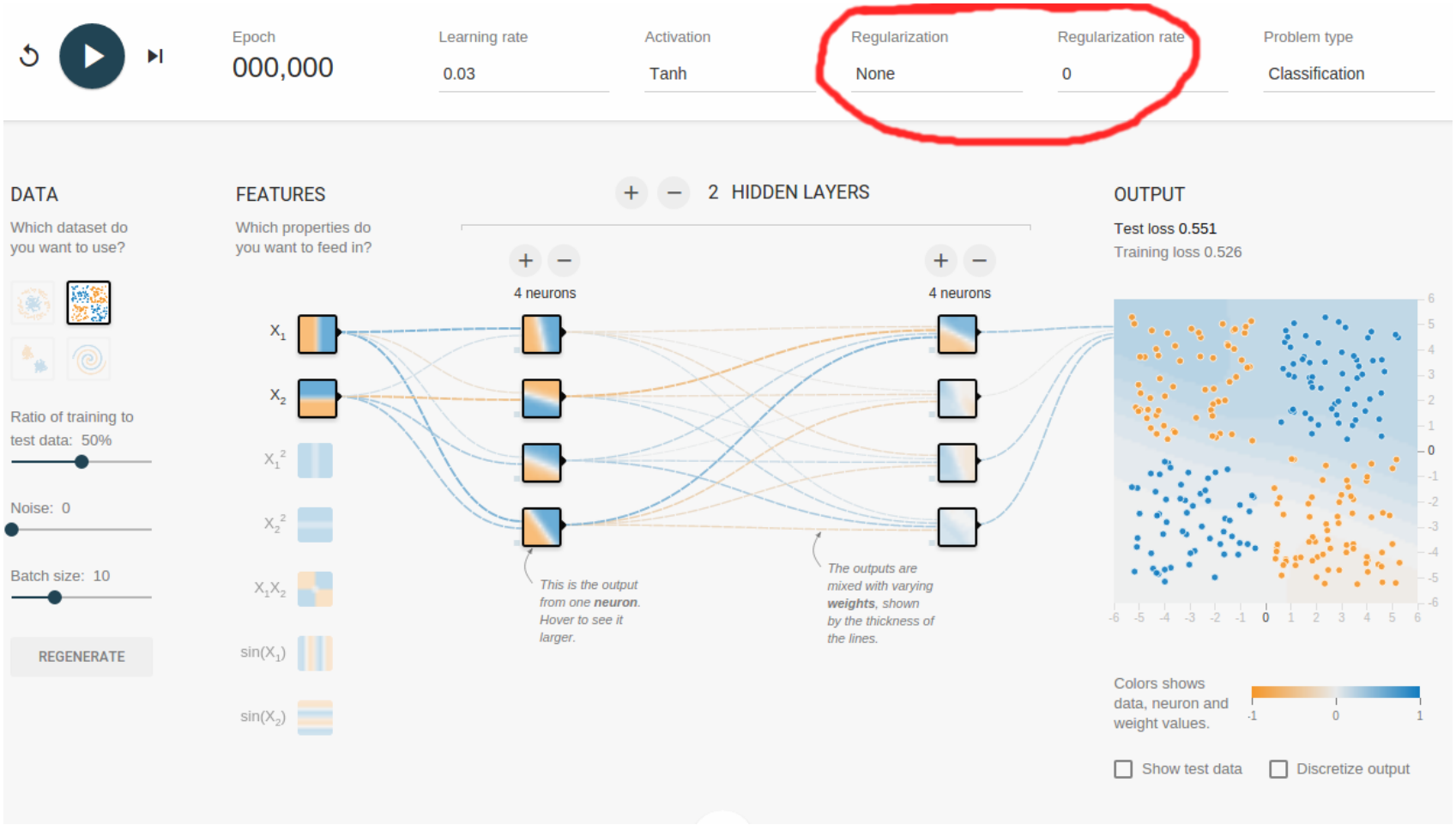
Epoka: etap uczenia, w którym uwzględnione są wszystkie dane uczące.

Plac zabaw dla tensorflow. Ważne pojęcia (2)



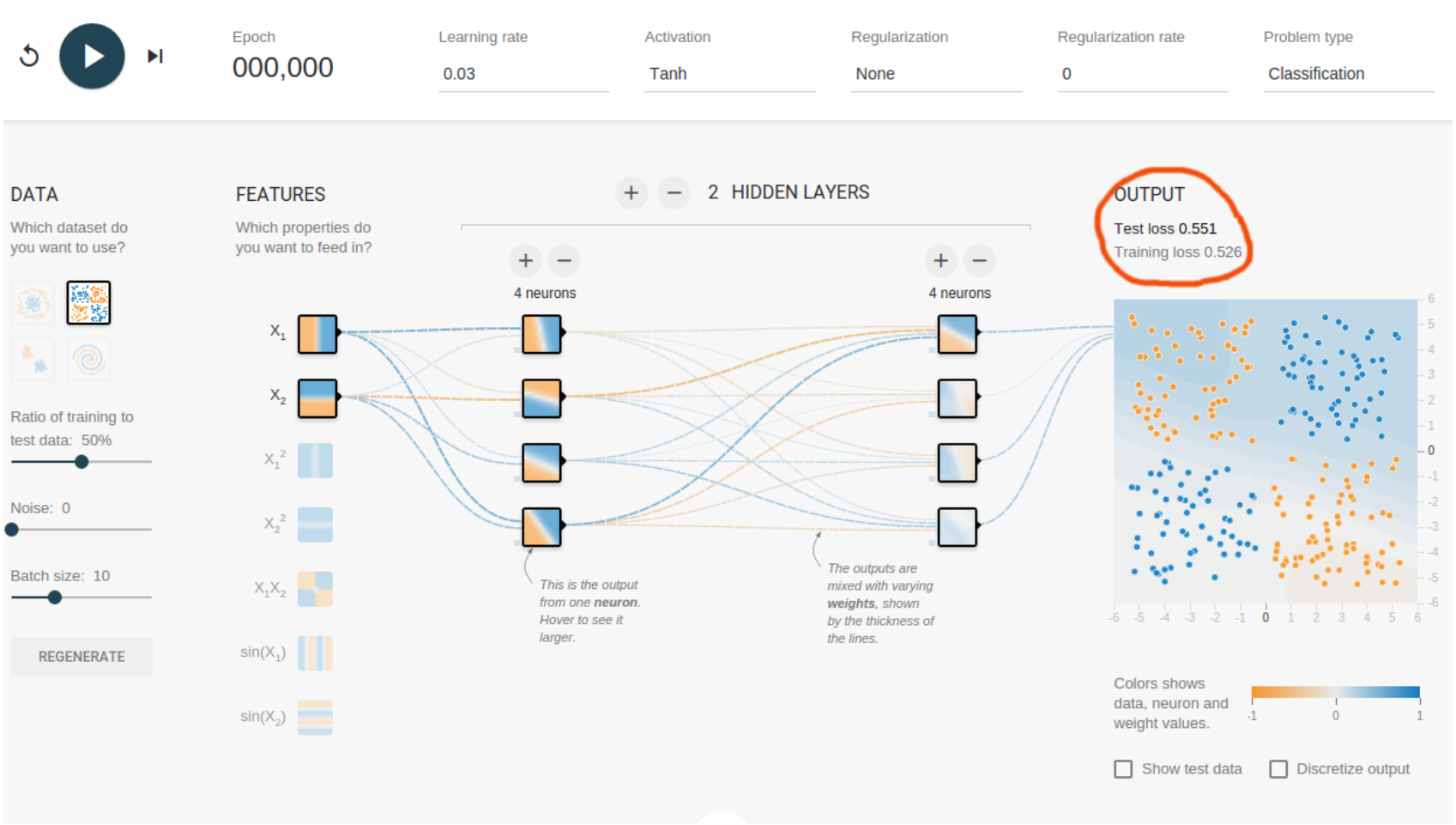
Learning rate: stała przez którą mnożone są delty wag. Za duża może dać chaotyczne zachowanie, za mała: bardzo wolny postęp.

Plac zabaw dla tensorflow. Ważne pojęcia (3)



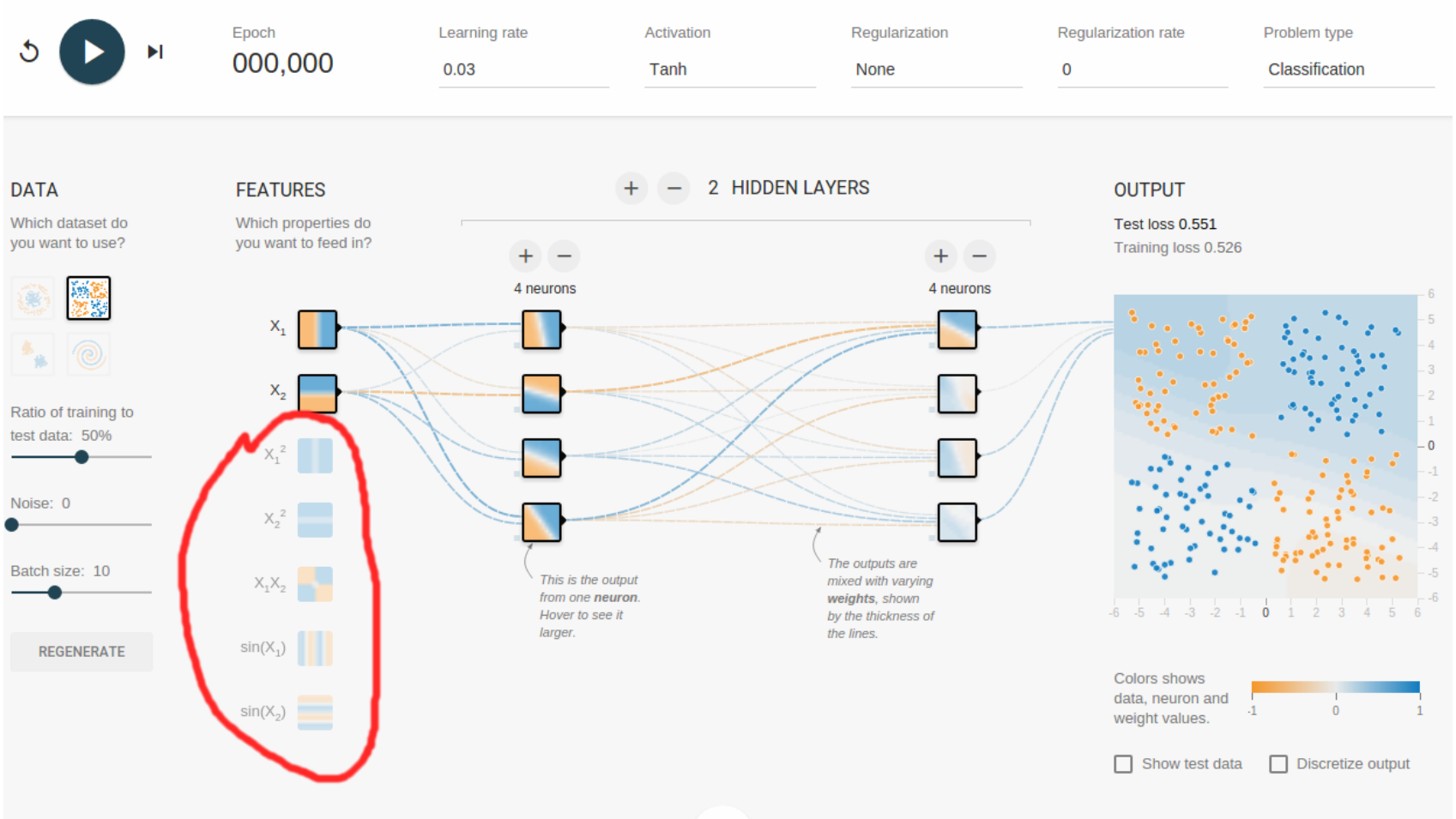
Regularyzacja: dołożenie do uczenia wymagania, by wagi nie były zbyt duże. Może dać większą stabilność uczenia (zob. tablica).

Plac zabaw dla tensorflow. Ważne pojęcia (4)



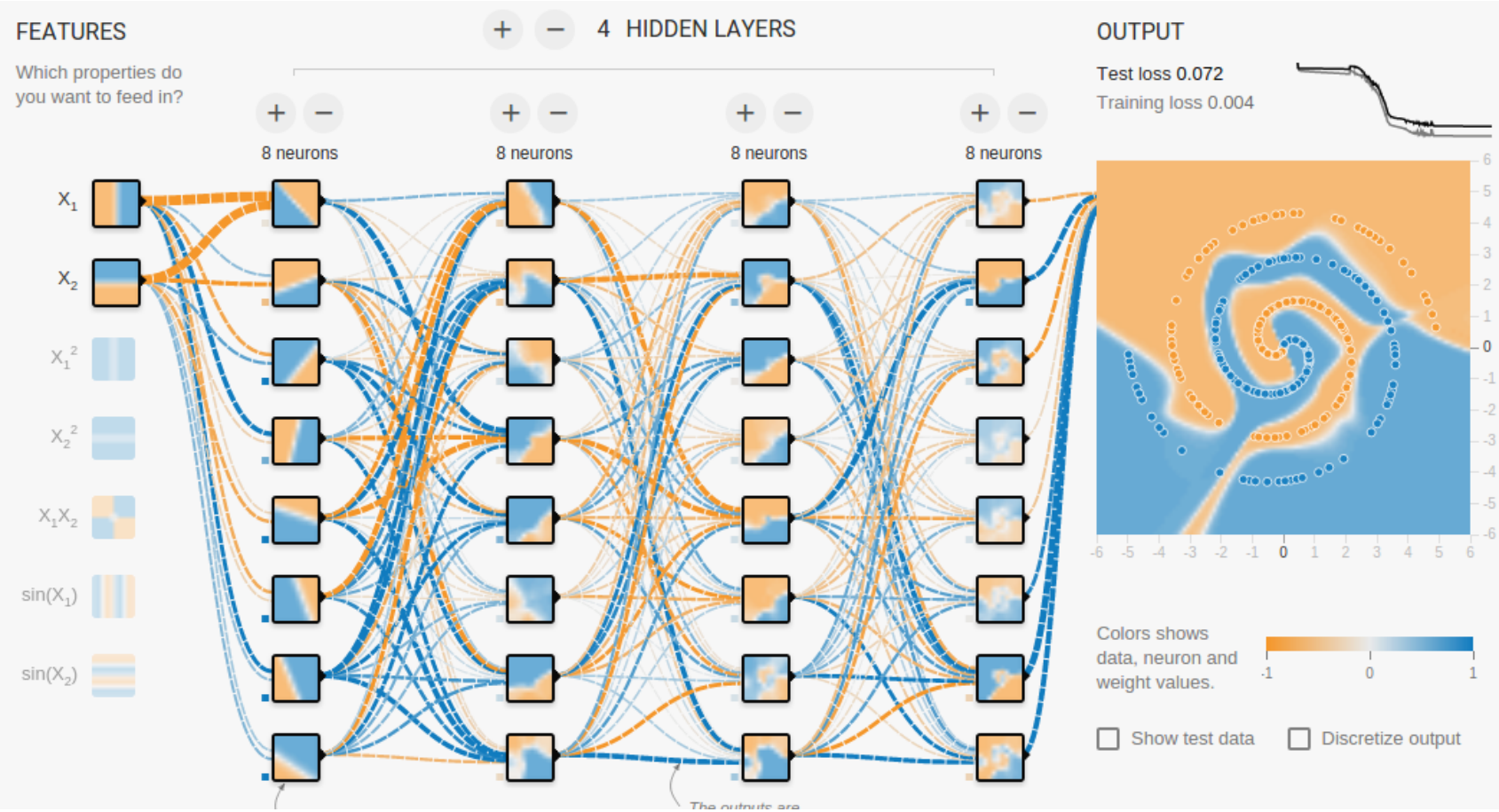
Test loss/training loss: wartość funkcji straty dla zbioru testowego i uczącego (oczywiście pierwsza zawsze większa).

Plac zabaw dla tensorflow. Ważne pojęcia (5)



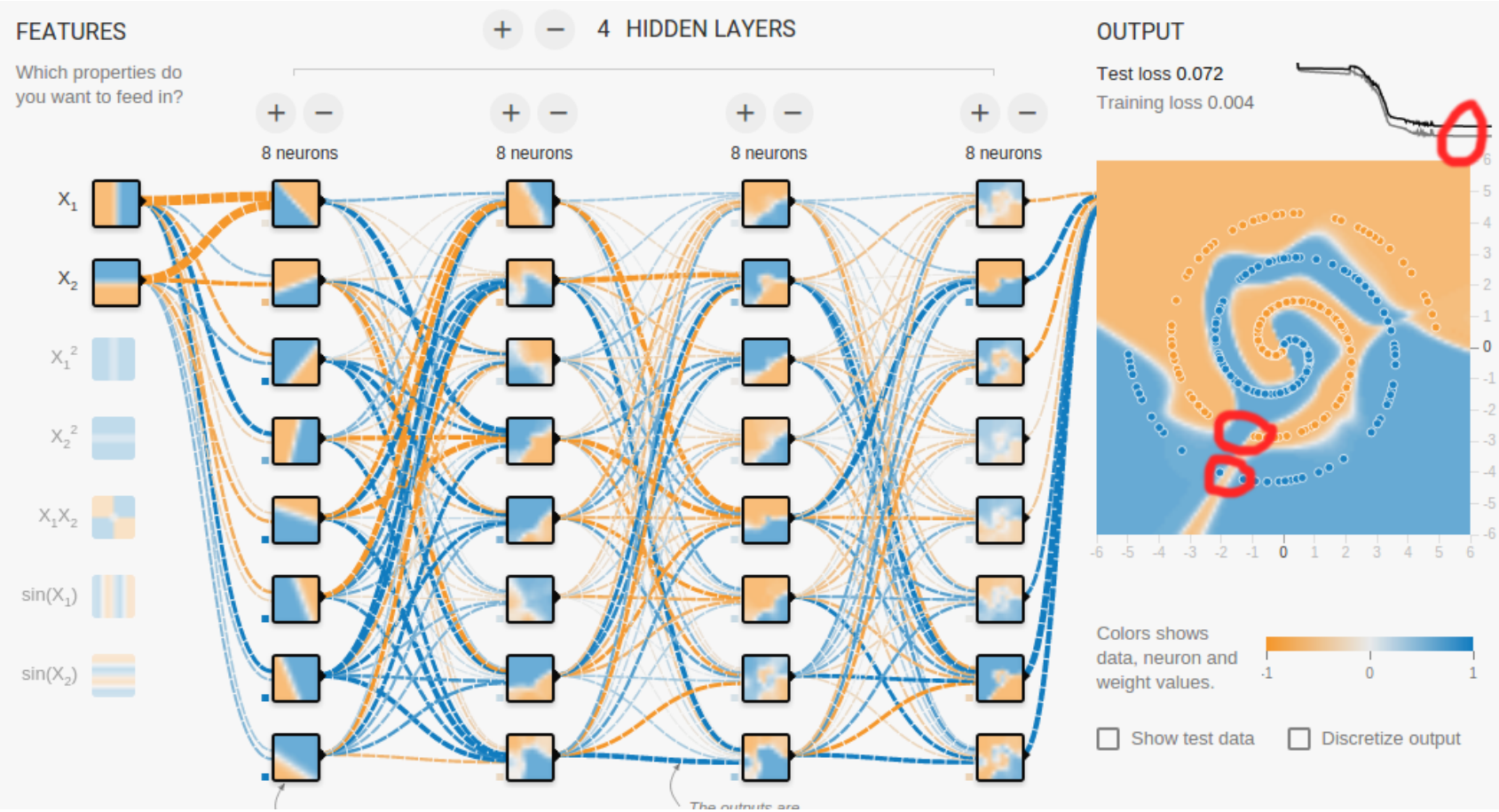
Feature engineering: proces tworzenia własnych cech dla konkretnych przypadków. Dobre cechy mają związek z zadaniem.

Plac zabaw dla tensorflow. Ważne pojęcia (6)



Przeuczenie (overfiting): sytuacja, w której sieć dostosowuje się do nieistotnych fluktuacji danych uczących, co pogarsza generalizację.

Plac zabaw dla tensorflow. Ważne pojęcia (6)



Przeuczenie (overfiting): sytuacja, w której sieć dostosowuje się do nieistotnych fluktuacji danych uczących, co pogarsza generalizację.

Kodowanie wejścia

- Wejściem do sieci jest **wektor** (czyli ciąg liczb o ustalonej długości)
- W tym wektorze możemy zakodować wszystko:
  - obrazki (jak?)
  - teksty o ustalonej długości (jak?)
  - sytuację na planszy w Reversi (jak?)

Kodowanie **one-hot**

Sieci neuronowe lubią *rozwlekłe* kodowanie, w którym liczbę $i \in \{0, \ldots, N-1\}$ kodujemy jako $(0, 0, 0, \ldots, 1, \ldots, 0, 0)$ (jedynka na i-tej pozycji).

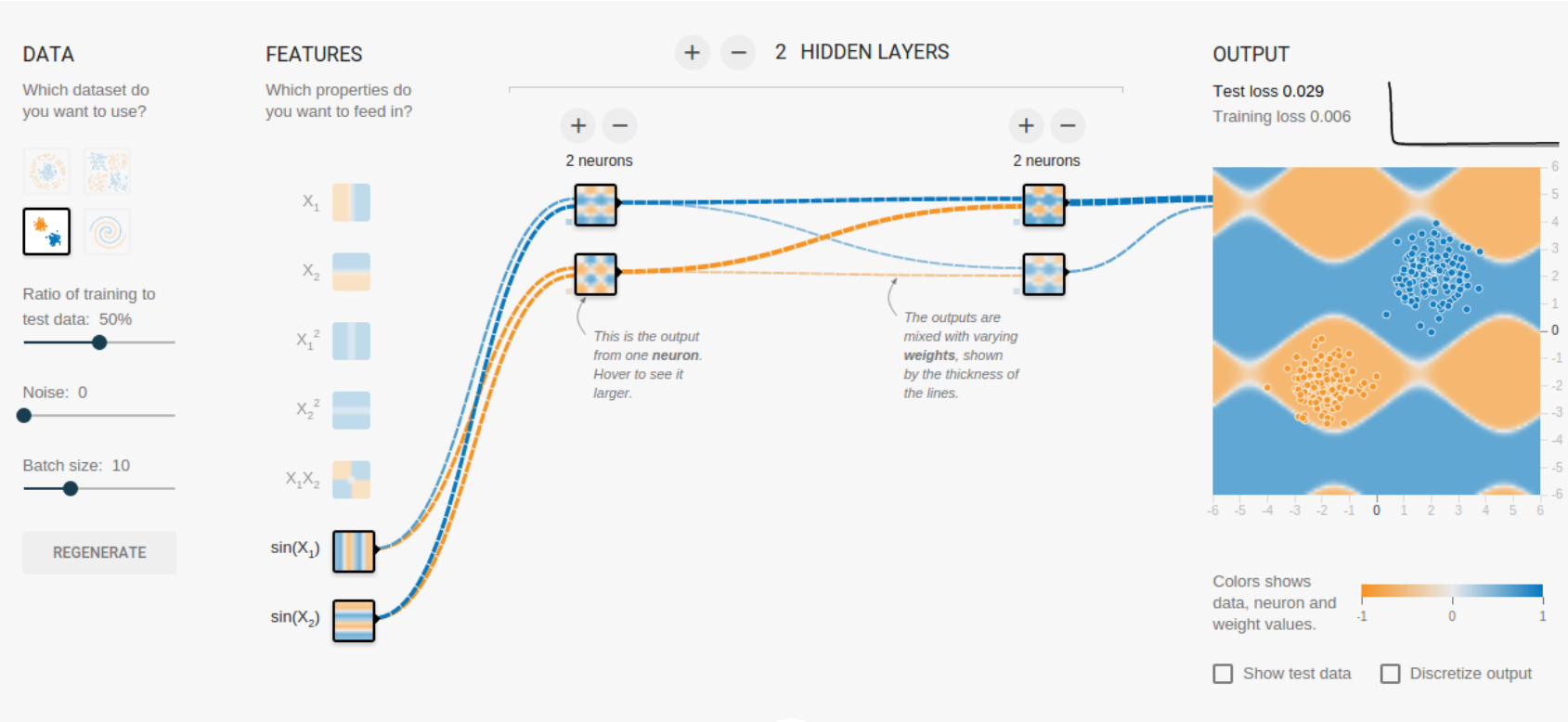
Kodowanie wejścia

- Pamiętajmy, że możemy dowolnie tworzyć cechy dla przypadków testowych:
  - Kwantyzacja dla obrazów
  - Analiza Fouriera dla dźwięków
  - Tworzenie *pseudosłów* (rzeczownik, a-cja, ...)
  - ...

Uwaga

Dodając cechy możemy przyśpieszyć uczenie, ale możemy też *zasugerować* sieci naszą wizję świata. Np. cecha w Reversi: wynik jakiejś funkcji heurystycznej.

Sugerowanie cykliczności



Sieć w miarę poprawnie sklasyfikowała zbiór uczący, dobrze też go uogólnia, ale jest przekonana, że świat jest mozaiką. Nikt z nas, widząc te dane nie wyrobił sobie tego poglądu.

Softmax

- Często chcemy, żeby sieć decydowała o jednej z K opcji (zadanie klasyfikacji).
- Rozmywamy ten wybór, prosząc o podanie rozkładu prawdopodobieństwa dla wszystkich K opcji.
- To tzw. Softmax layer, która przypisuje prawdopodobieństwo zależne od wielkości pobudzenia.

Wzór:

$$\text{Sotfmax}(z_i) = \frac{e^{z_i}}{\sum_{j=1}^{d} e^{z_j}}$$

Popatrzmy na to, jak działa funkcja Softmax.

Softmax

```
>>> softmax([1,2,3])
[0.09003, 0.24472, 0.66524]

>>> softmax([1,2,3,10]
[0.00012, 0.00033, 0.00091, 0.99863]

>>> softmax([3,4,4])
[0.15536, 0.42231, 0.42231]
```

Super łatwe sieci neuronowe

- Można wykorzystać bibliotekę **sklearn** (lub analogiczną), która implementuje **MLP** (czyli wielowarstwowy perceptron)
- Sieć definiujemy jednym konstruktorem z dużą liczbą parametrów (ale ufamy, że wartości domyślne są ok)

Super łatwe sieci neuronowe

Przygotowanie danych

```
from sklearn.neural_network import MLPClassifier
import random, pickle

# data: list of pairs (X,y)
# X: vector of floats/ints
# y in [v1,...,vk]

random.shuffle(data)
N = len(data) / 6
test_data = data[:N]
dev_data  = data[N:]

X = [x for (x,y) in dev_data]
y = [y for (x,y) in dev_data]
X_test = [x for (x,y) in test_data]
y_test = [y for (x,y) in test_data]
```

Super łatwe sieci neuronowe (2)

Uczenie sieci

```
# creating model
nn = MLPClassifier(hidden_layer_sizes=(60,60,10))

# training model
nn.fit(X,y)

print ('Dev score', nn.score(X,y))
print ('Test score', nn.score(X_test, y_test))

# writing model
with open('nn_weights.dat', 'w') as f:
    pickle.dump(nn, f)
```

Super łatwe sieci neuronowe (3)

Korzystanie z sieci

```
from sklearn.neural_network import MLPClassifier
import pickle

with open('nn_weights.dat') as f:
    nn = pickle.load(open(f))

x = data_vector

probabilities = nn.predict_proba([x])

prob0 = ys[0][0]
prob1 = ys[0][1]
```

Podsumowanie

Cons

- Oczywiście daje dużo mniejszą swobodę niż bardziej specjalizowane biblioteki.
- Nadaje się do tworzenia niezbyt dużych sieci
- Nie ma sieci splotowych, sieci rekurencyjnych, ...

Pros

- Bardzo prosta w użyciu i wystarczająco szybka
- Ten sam (prawie) interfejs dla różnych mechanizmów:
  - `from sklearn.neighbors import KNeighborsClassifier as Classifier`
  - `from sklearn.tree import DecisionTreeClassifier as Classifier`
  - `from sklearn.svm import SVC as Classifier`
  - ... (i jeszcze kilkanaście innych)

Inne rodzaje uczenia

Uwaga

Rozważaliśmy uczenie z nadzorem, czyli taki wariant, w którym dysponujemy dodatkowymi danymi (dotyczącymi np. prawidłowej klasyfikacji każdej próbki).

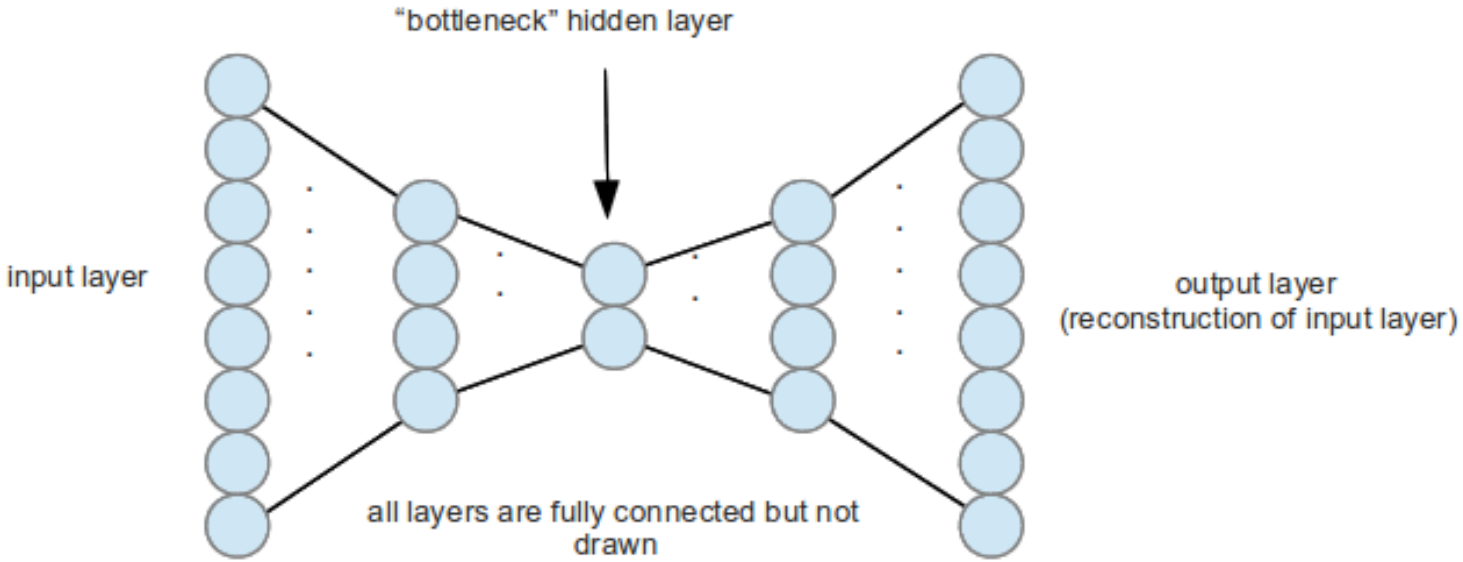
A co można zrobić, jeżeli mamy same próbki?

1. Nauczyć się generować podobne próbki (**autoenkoder**).
2. Pogrupować próbki (algorytmy **klasteryzacji**)
3. Narysować próbki (algorytmy **wizualizacji**)
4. Znaleźć **dziwne** próbki (algorytmy **wykrywania nieprawidłowości**, czyli anomaly detection)

Z wizualizacją i autoenkoderami związana jest **redukcja wymiarowości**

Autoenkodery

- Tworzymy zadanie uczenia się z nadzorem (funkcji identycznościowej)
- Wariant jednowarstwowej funkcji liniowej jest skrajnie nieciekawy (bo?)
- Wielowarstwowa sieć, która ma część redukującą wymiar (coraz mniejsze warstwy) i analogiczną grupę warstw zwiększającą wymiar
- Może być użyteczna, bo tworzy wewnętrzną reprezentację obrazu



Bardziej skomplikowane autoenkodery (NVIDIA Celebrities)

Ci ludzie nie dadzą Ci autografu (przykładowe twarze dla losowego zacisku)

źródło: http://research.nvidia.com/

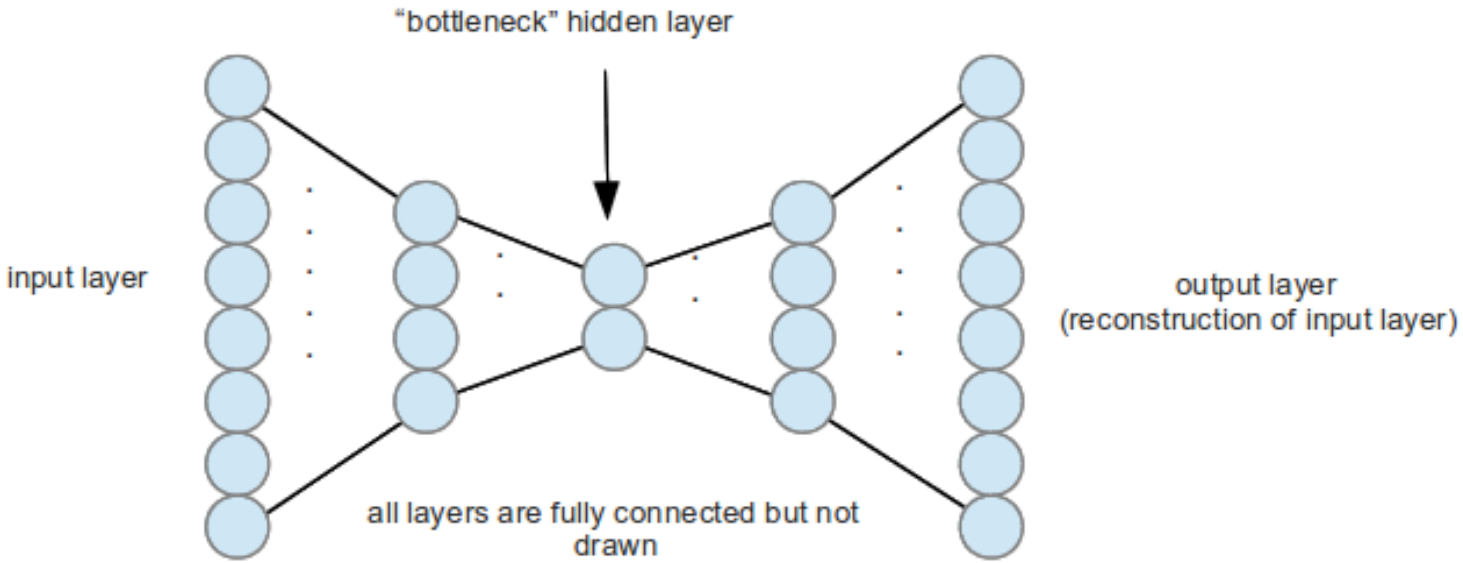
O twarzach (2)

Oczywiście nie zawsze jest idealnie, bo:

źródło: https://nerdist.com/nvidia-ai-headshots-fake-celebrities/

Anomalie (1)

Autoenkoder może być użyty do wykrywania anomalii.
Autoenkodery radzą sobie dobrze z **typowymi próbkami**.



Nietypowe próbki będą miały duży błąd rekonstrucji!

Klasteryzacja

Definicja

Klasteryzacja (grupowanie) to zadanie identyfikacji w próbce uczącej naturalnych grup związanych ze sobą obiektów.

Obiekt = wektor w $\mathcal{R}^n$

- Najprostszy wariant: chcemy otrzymać konkretną liczbę grup, powiedzmy K
- Najprostszy algorytm: K-średnich (k-means)

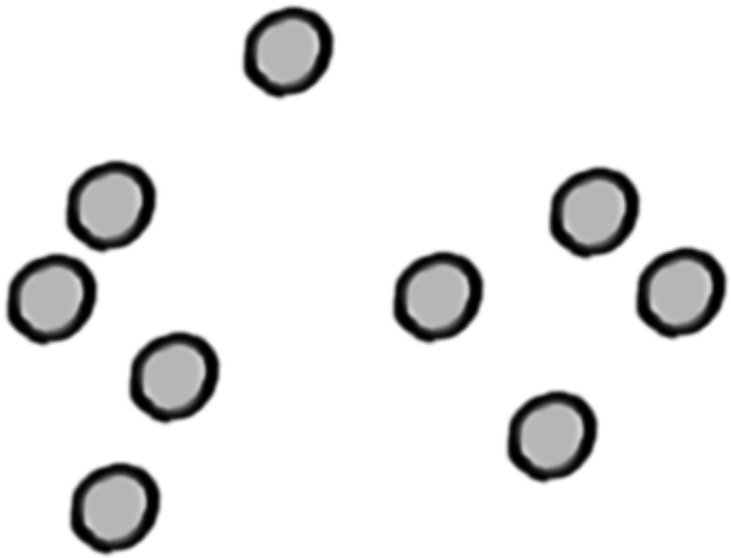
Algorytm K-średnich

Przez cały czas działania algorytmu pamiętamy K prototypów (czyli punktów będących reprezentantami grupy)

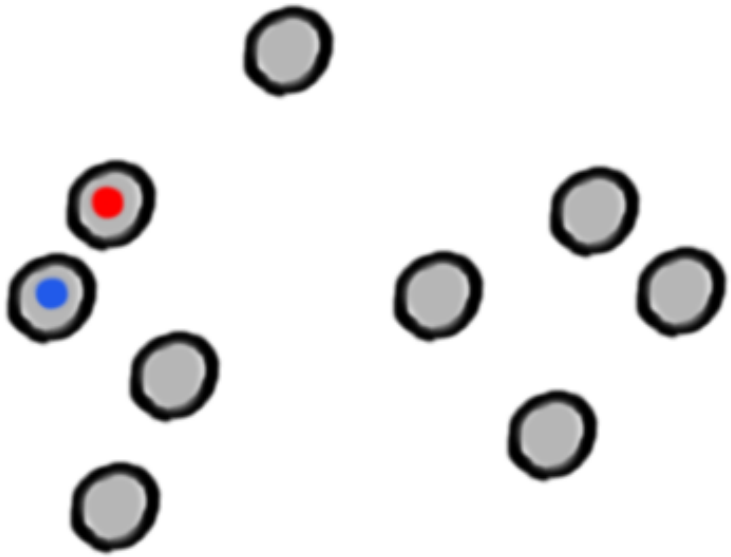
Algorytm przeplata dwie fazy:

1. Przypisanie każdego punktu do najbliższego mu prototypu
2. Obliczenie nowych prototypów jako **średnich** wszystkich punktów przypisanych do tego samego prototypu

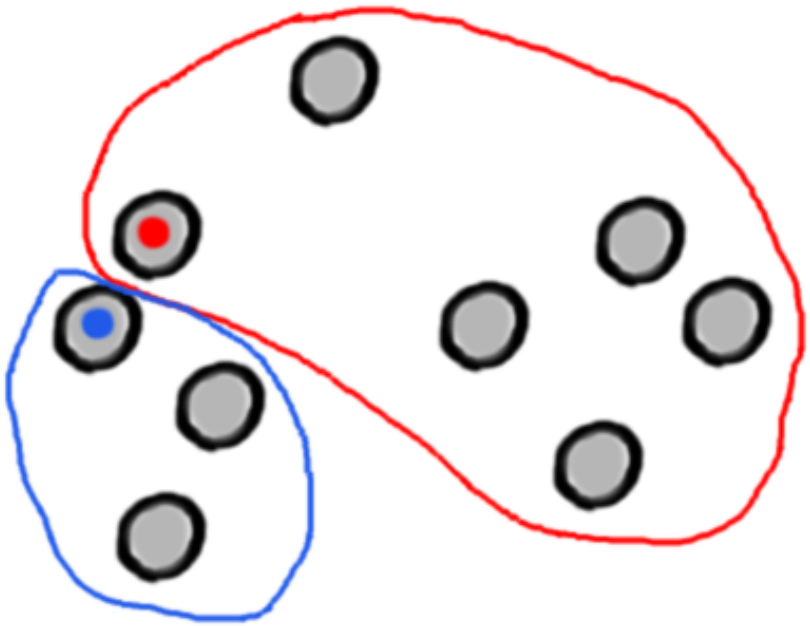
Algorytm K-średnich. Przykład: K=2



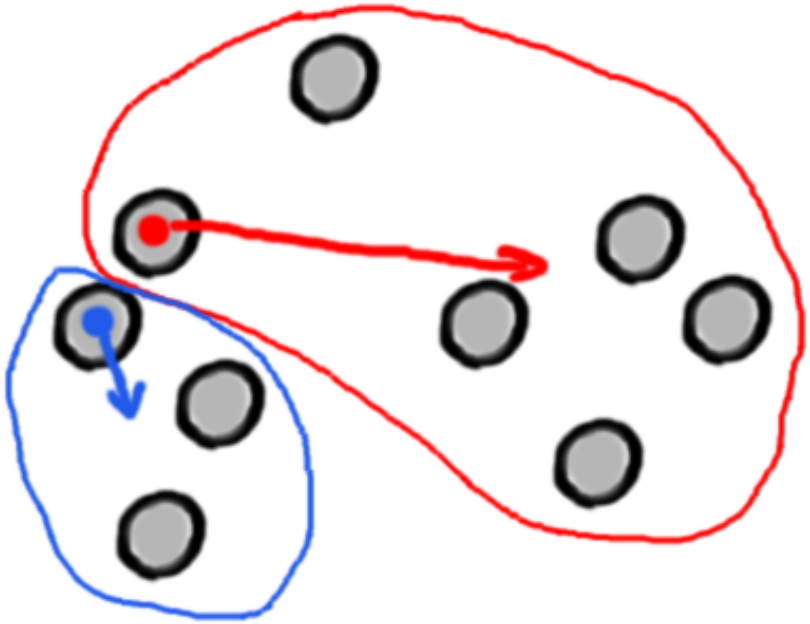
Algorytm K-średnich. Przykład: K=2



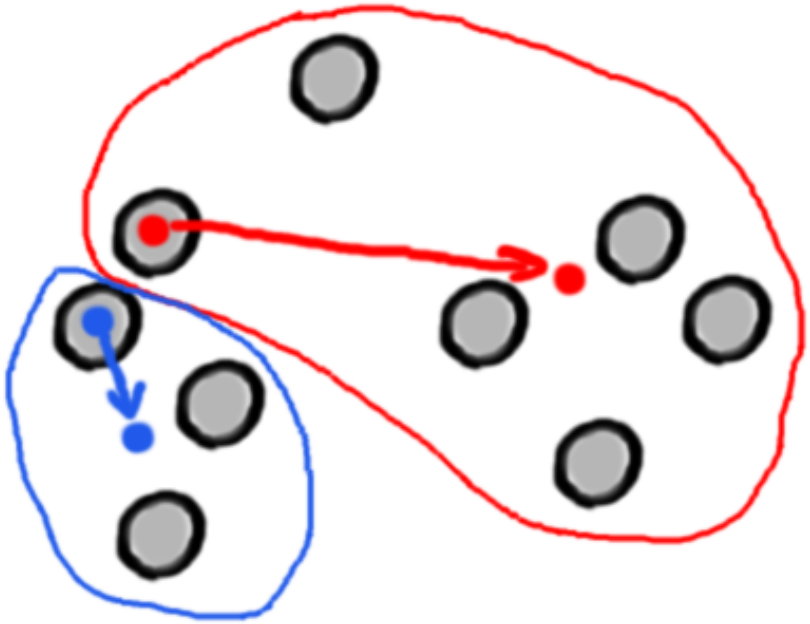
Algorytm K-średnich. Przykład: K=2



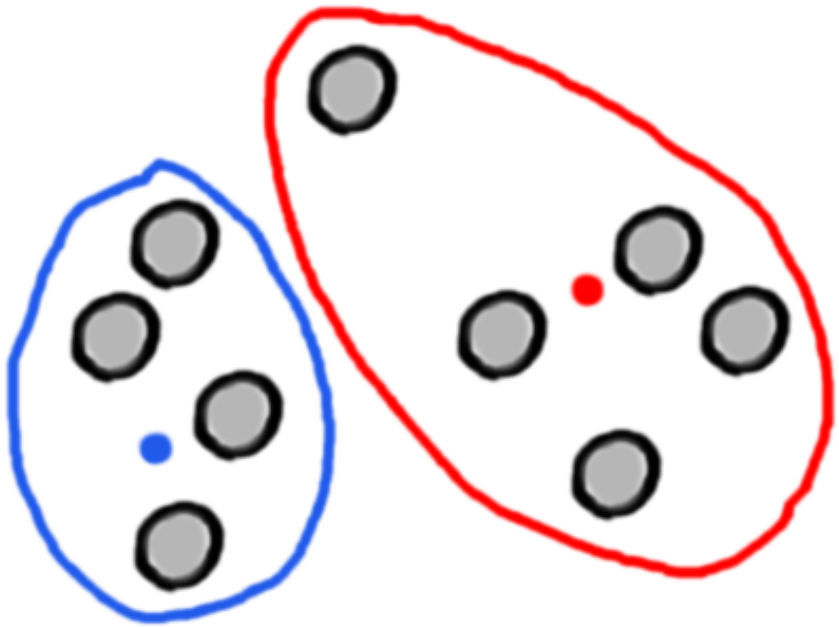
Algorytm K-średnich. Przykład: K=2



Algorytm K-średnich. Przykład: K=2



Algorytm K-średnich. Przykład: K=2



Algorytm K-średnich. Demonstracja)

- Losujemy pewną liczbe punktow na plaszczyźnie, tak aby w naturalny sposób tworzyły klastry.
- Wybieramy początkowe centra z populacji punktów
- Obserwujemy, jak działa algorytm

Popatrzmy na demonstrację `kmeans.py`
Dodatkowe funkcje: `restart`, `new`

Co oblicza algorytm K-średnich?

Cel

Chcemy, żeby prototyp przybliżał wszystkie przypisane mu elementy.

Daleka analogia: Prototyp jest takim elektorem, który przybliża poglądy swoich wyborców.
Każdy wyborca jest zadowolony, jeżeli ma elektora dobrze rozumiejącego jego preferencje.

Co oblicza algorytm K-średnich? (2)

- Interesuje nas, aby **każdy element, jak najmniej się różnił od swojego reprezentanta** (czyli średniej, prototypu, centroidu)
- Miara: błąd średniokwadratowy, czyli

$$\sum_{x \in \text{Dane}} (x - \text{reprezentant}(x))^2$$

Definicja

Powyżej zdefiniowaną wielkość nazwiemy błędem klasteryzacji.

K-średnich jako minimalizacja błędu

Etap przypisywania

Prototypy ustalone. Każdy egzemplarz trafia do bliższego prototypu (czyli błąd maleje).

Etap liczenia średnich

Patrzymy na 1 klaster. Policzmy pochodną po c dla

$$\frac{1}{N}\sum_{i=1}^{N}(x_i - c)^2$$

Wychodzi:

$$\frac{2}{N}\sum_{i=1}^{N}(c - x_i)$$

Błąd klasteryzacji dla jednego klastra osiąga minimum w punkcie będącym średnią punktów klastra.

K-średnich jako minimalizacja błędu

Wniosek

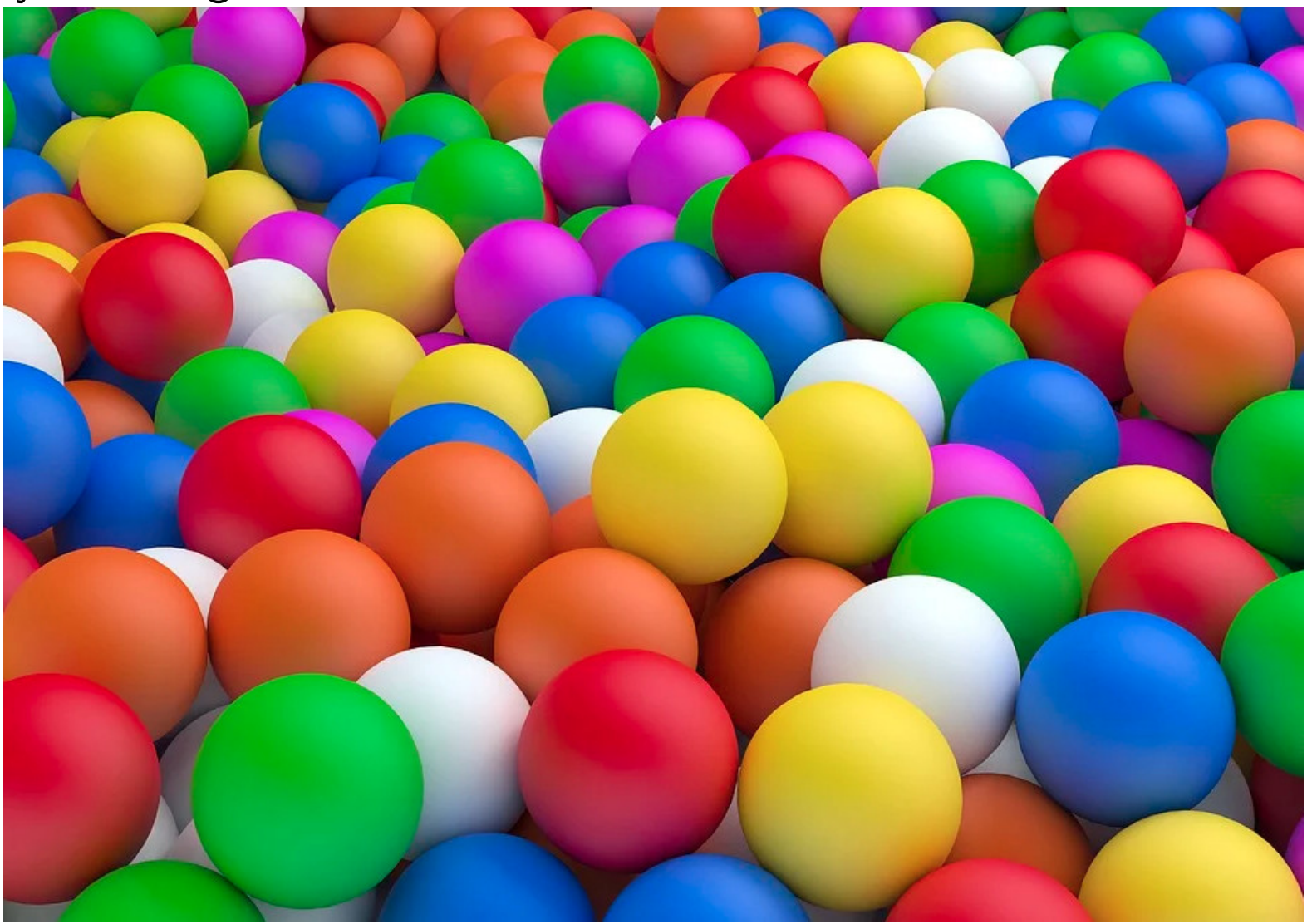
Oba etapy nie zwiększają błędu (tzn. jeżeli coś robią, to błąd maleje). Czyli osiągamy lokalne minimum.

K-średnich. Przykładowe zastosowania

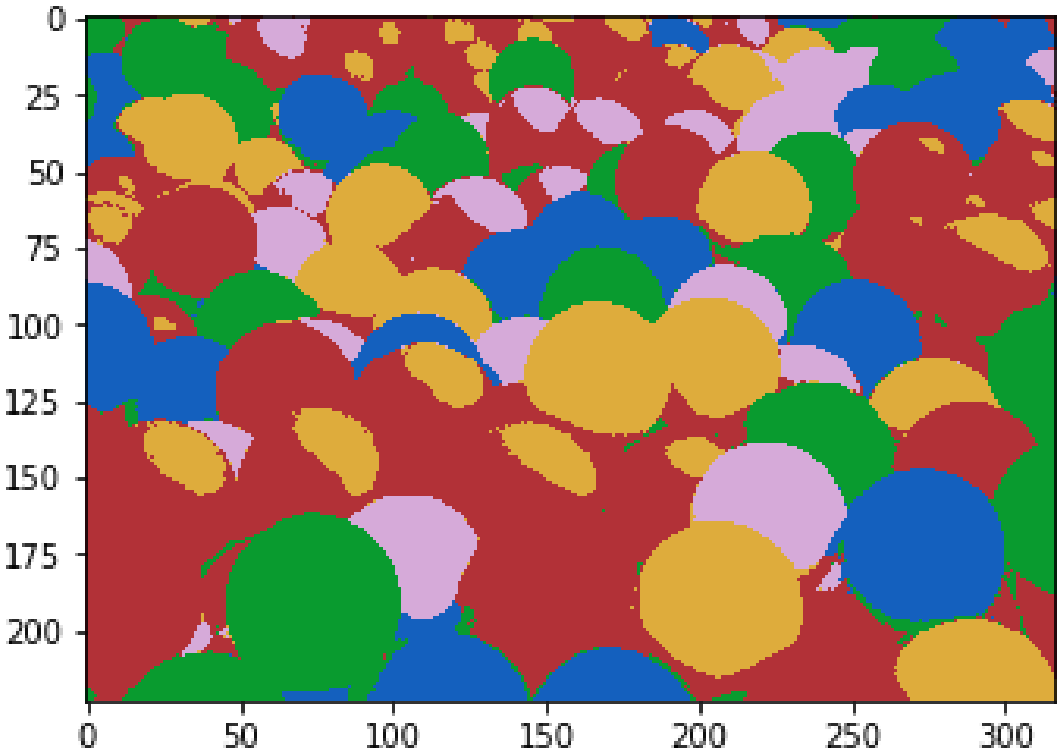
1. Wybór palety K kolorów, dostosowanej do obrazka.
2. Sklejanie obrazka z kwadratów (kompresja wektorowa)
3. Grupowanie słów podobnych (jeżeli reprezentujemy słowa jako wektory)
   - Po zastosowaniu do fraz: odpoczynek w pięknym miasteczku $\approx$ wypoczynek w uroczym kurorcie

K-średnik dla kolorów

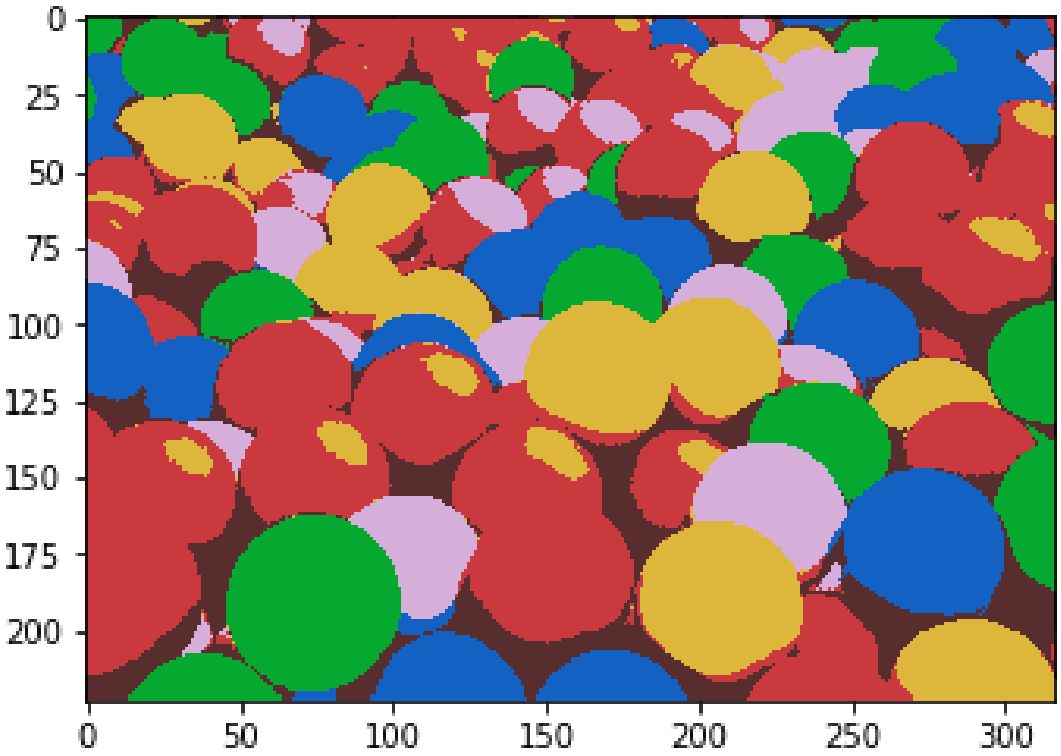
Popatrzymy, jak wybór palety i kompresja wektorowa działają dla przykładowego obrazka:



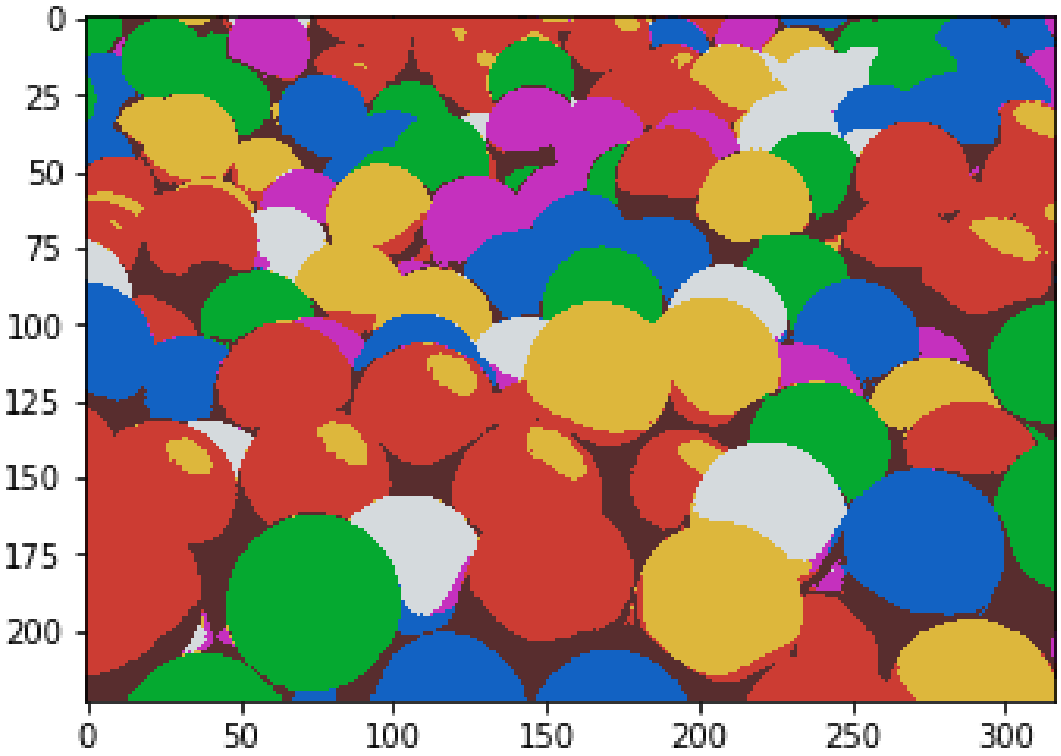
K-średnich. Wybór reprezentatywnych pikseli, K=5



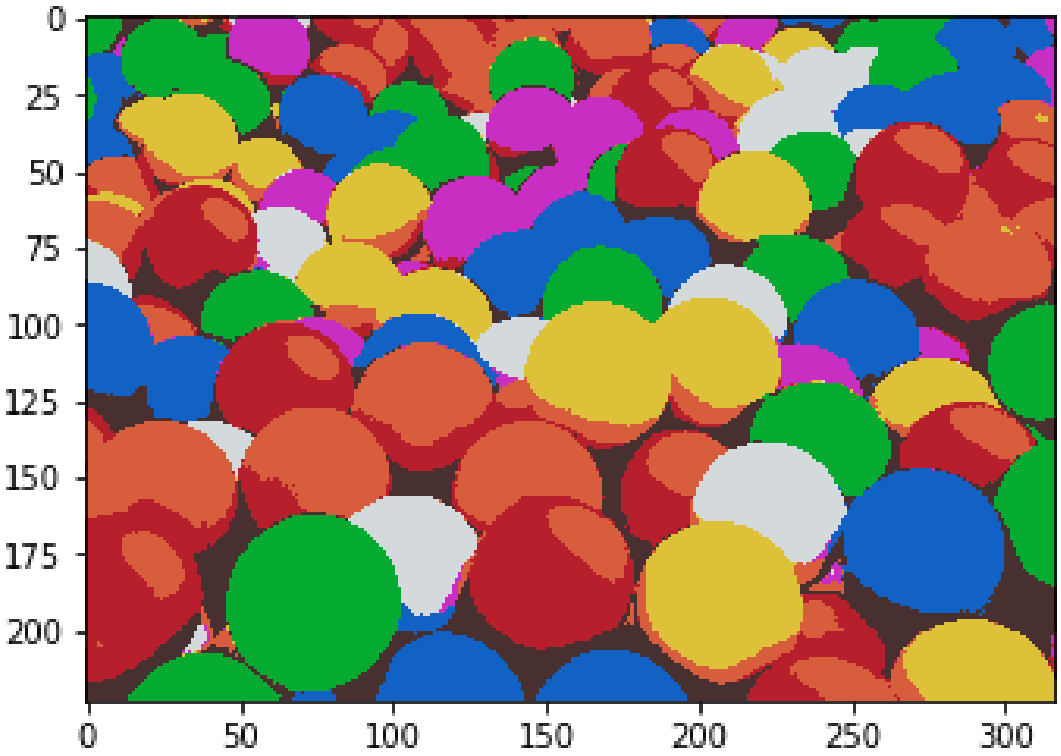
K-średnich. Wybór reprezentatywnych pikseli, K=6



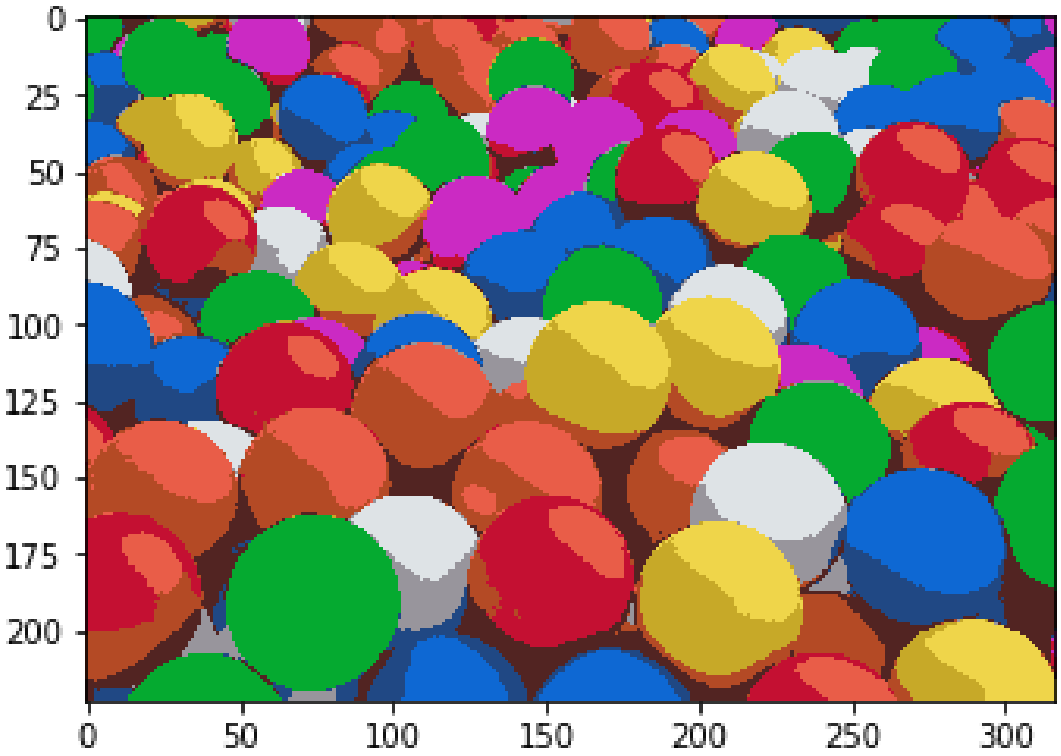
K-średnich. Wybór reprezentatywnych pikseli, K=7



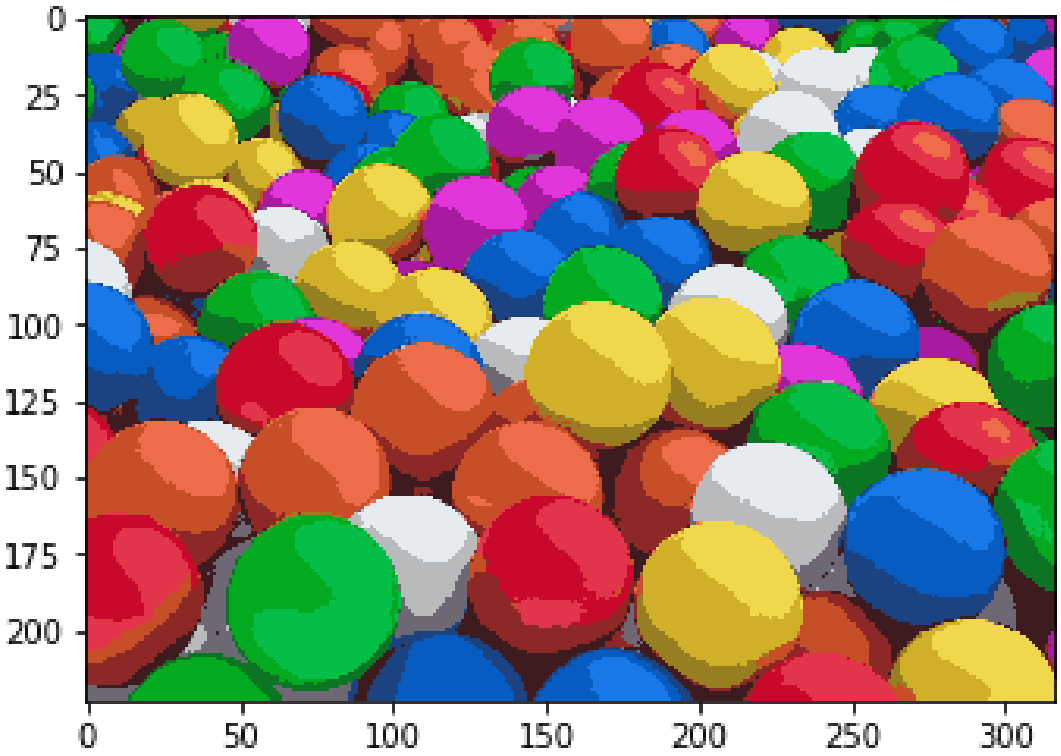
K-średnich. Wybór reprezentatywnych pikseli, K=8



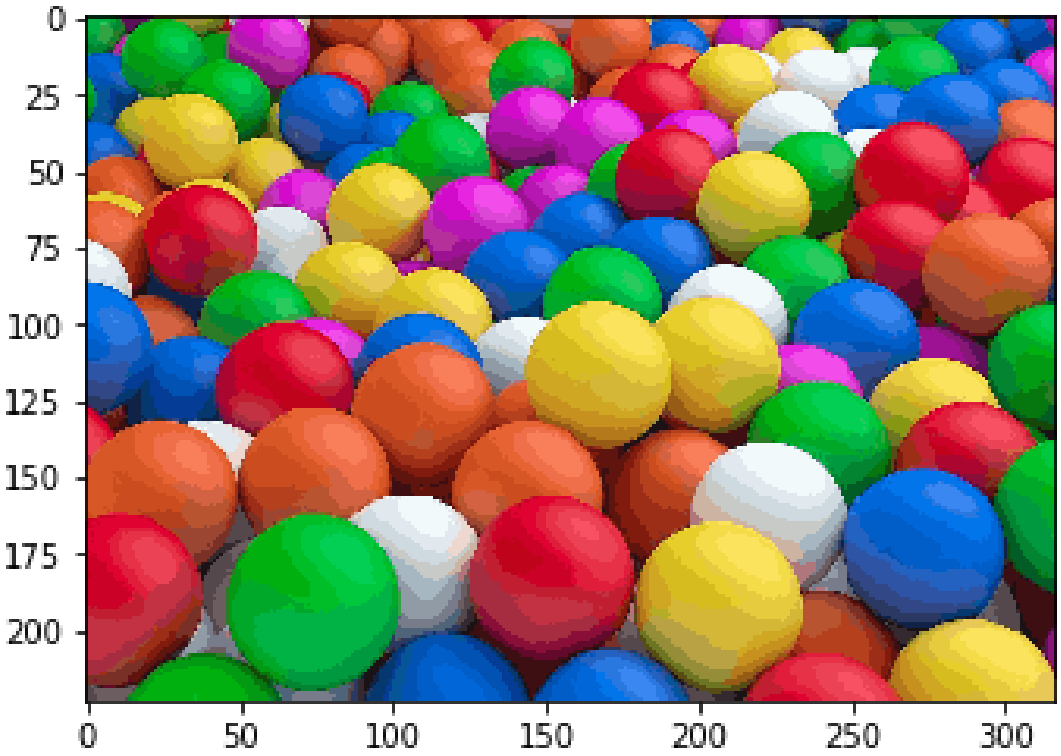
K-średnich. Wybór reprezentatywnych pikseli, K=12



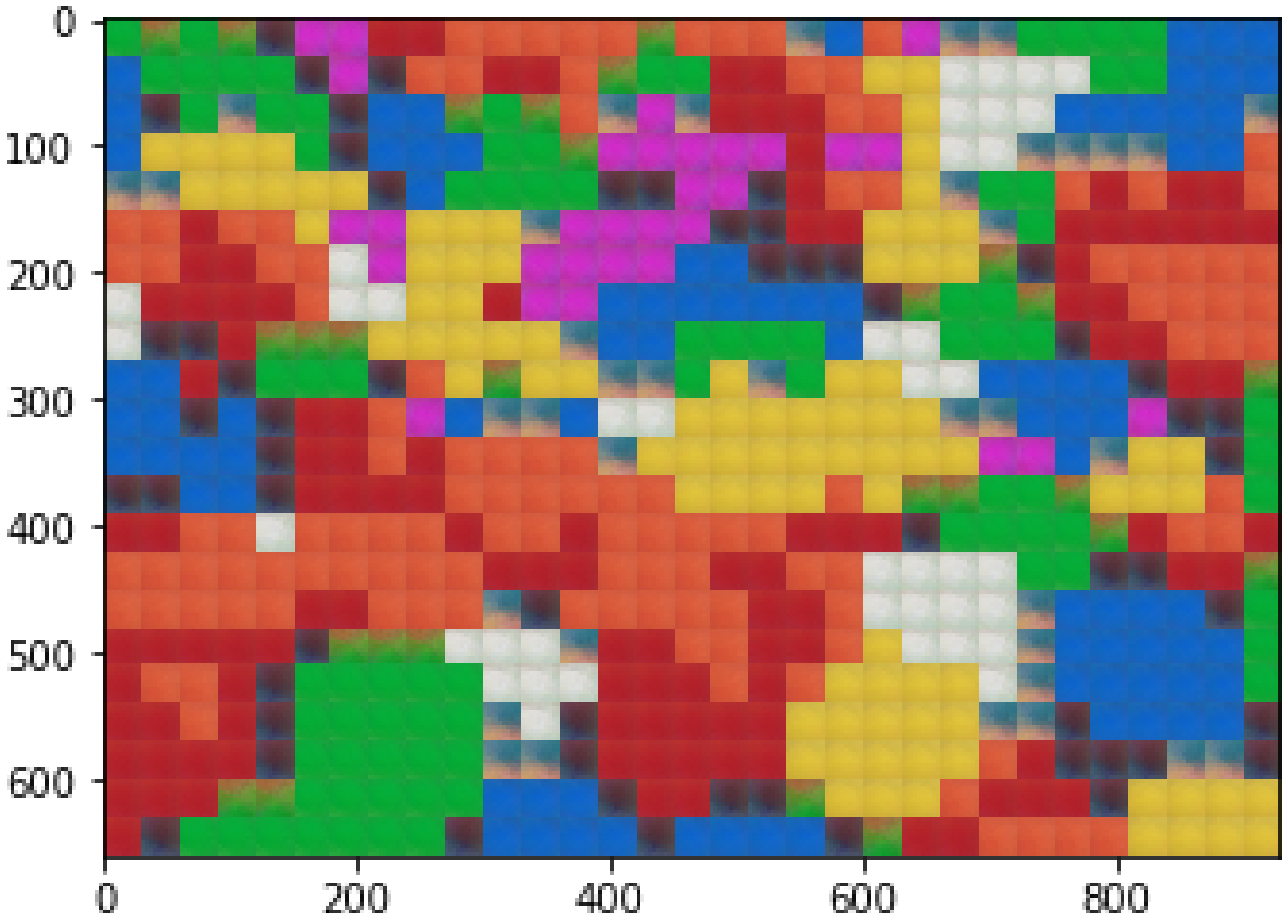
K-średnich. Wybór reprezentatywnych pikseli, K=20



K-średnich. Wybór reprezentatywnych pikseli, K=100



Na deser – wynik kompresji wektorowej, K=10



Na deser – wynik kompresji wektorowej, K=100