DL Sieci neuronowe

DL Sieci neuronowe

Agenda

1. **Struktura sieci: neuron, warstwa**
2. Pojęcie wagi, biasu, funkcji aktywacji i propagacji
3. Narzędzia: Google Colab, Tensorflow, Keras

DL Sieci neuronowe

Agenda

1. Struktura sieci: neuron, warstwa
2. **Pojęcie wagi, biasu, funkcji aktywacji i propagacji**
3. Narzędzia: Google Colab, Tensorflow, Keras

DL Sieci neuronowe

Agenda

1. Struktura sieci: neuron, warstwa
2. Pojęcie wagi, biasu, funkcji aktywacji i propagacji
3. **Narzędzia: Google Colab, Tensorflow, Keras**

DL Sieci neuronowe

Czym jest Deep Learning?

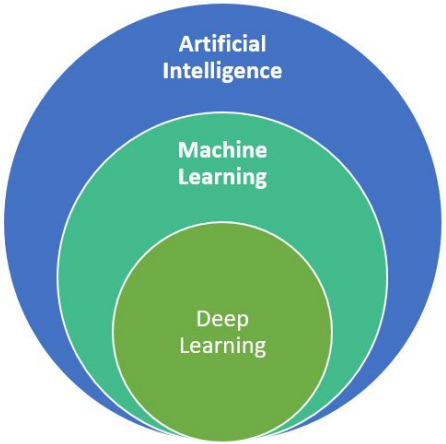
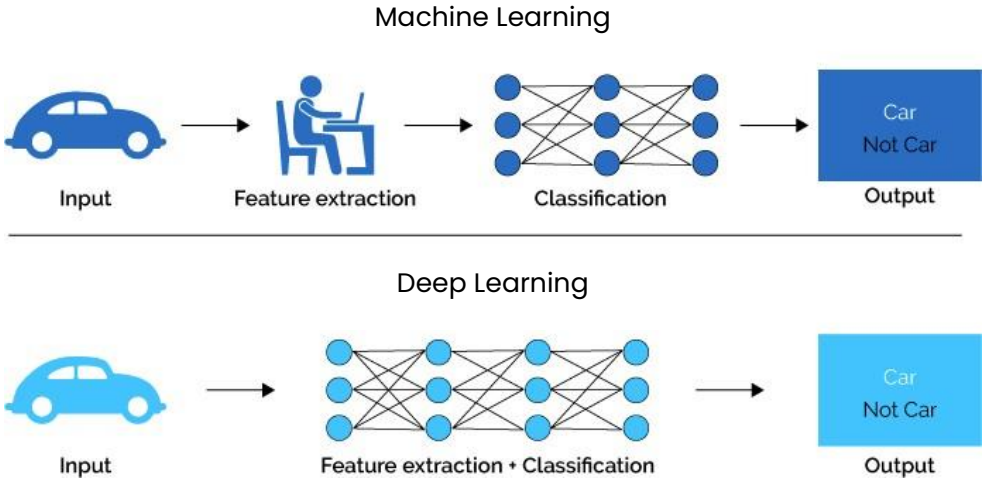
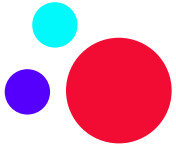


Figure 1: artificial intelligence, machine leaning and deep learning Source: Nadia BERCHANE (M2 IESCI, 2018)

DL Sieci neuronowe

Czym jest Deep Learning?

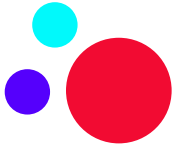




DL Sieci neuronowe

Czym się różni od ML?

- Posiada "uczący się feature extractor".
- Modele bywają ogromne.
- Na ogół potrzebuje ogromną ilość danych.
- Oraz mocny sprzęt do obliczeń ;)
- Do niemal każdego problemu można dostosować odpowiednią architekturę sieci neuronowej.



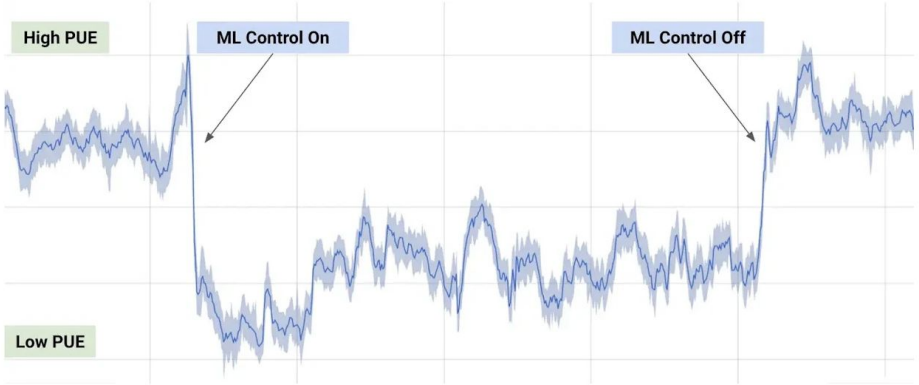
DL Sieci neuronowe

Rodzaje głębokiego uczenia

- Predykcja wartości danych “tabelkowych”.
- Rozpoznawanie obrazów.
- Detekcja obiektów.
- Klasyfikacja tekstu.
- Translacja.
- Wiele wiele innych...

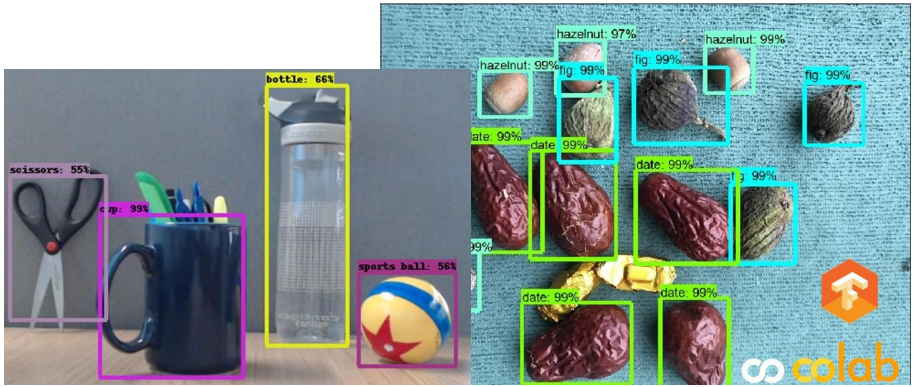
DL Sieci neuronowe

Optymalizacja kosztów zużycia energii



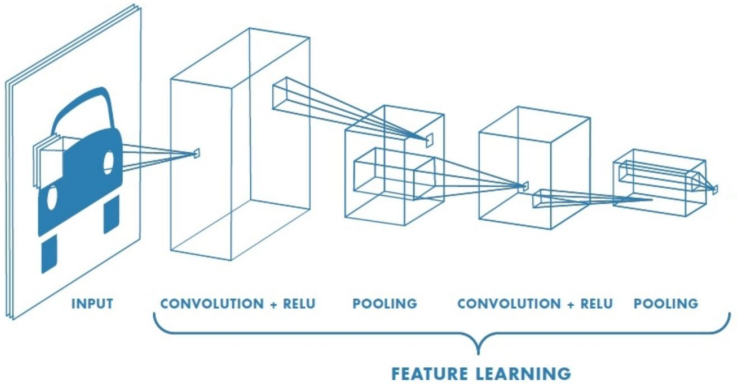
DL Sieci neuronowe

Konwolucyjne sieci neuronowe



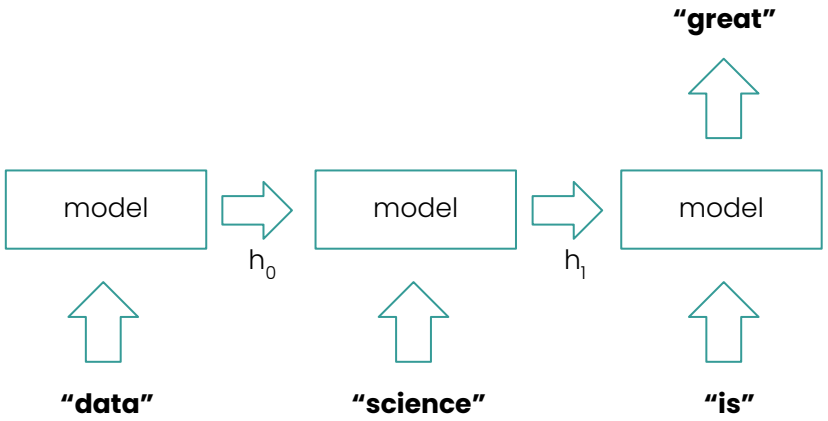
DL Sieci neuronowe

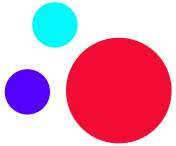
Konwolucyjne sieci neuronowe



DL Sieci neuronowe

Generowanie tekstu – rekurencyjne sieci neuronowe





DL Sieci neuronowe

gpt2

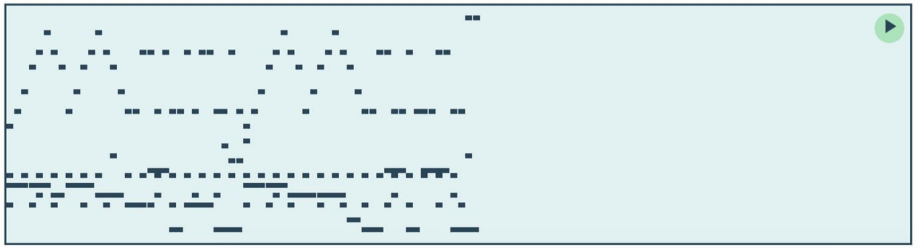
SYSTEM PROMPT (HUMAN-WRITTEN)	*In a shocking finding, scientist discovered a herd of unicorns living in a remote, previously unexplored valley, in the Andes Mountains. Even more surprising to the researchers was the fact that the unicorns spoke perfect English.*
MODEL COMPLETION (MACHINE-WRITTEN, 10 TRIES)	The scientist named the population, after their distinctive horn, Ovid's Unicorn. These four-horned, silver-white unicorns were previously unknown to science. Now, after almost two centuries, the mystery of what sparked this odd phenomenon is finally solved. Dr. Jorge Pérez, an evolutionary biologist from the University of La Paz, and several companions, were exploring the Andes Mountains when they found a small valley, with no other animals or humans. Pérez noticed that the valley had what appeared to be a natural fountain, surrounded by two peaks of rock and silver snow.

infoShare ACADEMY

DL Sieci neuronowe

Magenta tensorflow

And this is what that sample sounds like:



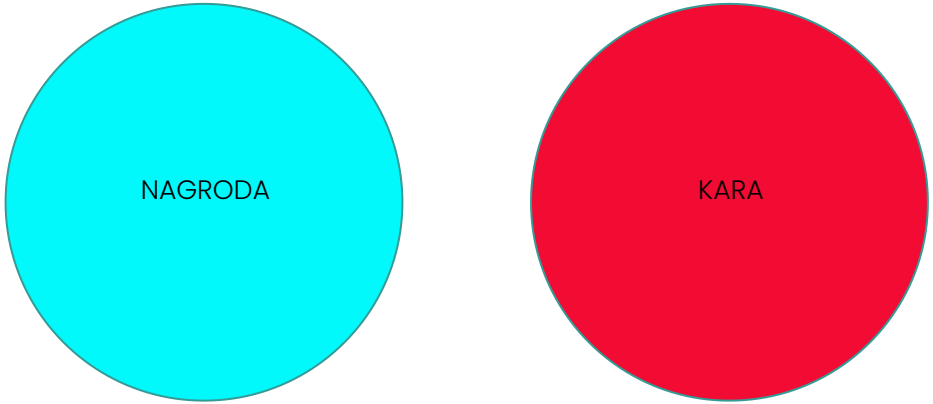
infoShareAcademy.com

DL Sieci neuronowe

Generowanie rzeczywistego obrazu

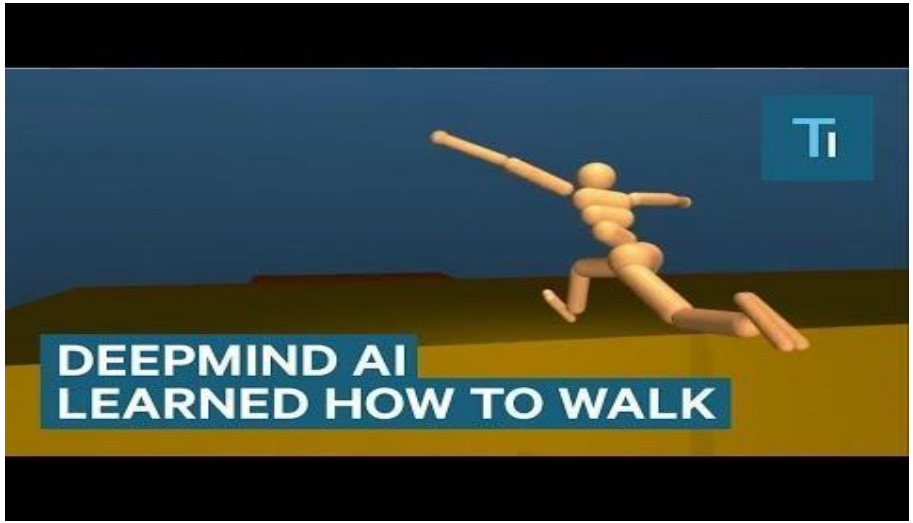
DL Sieci neuronowe

Uczenie ze wzmocnieniem



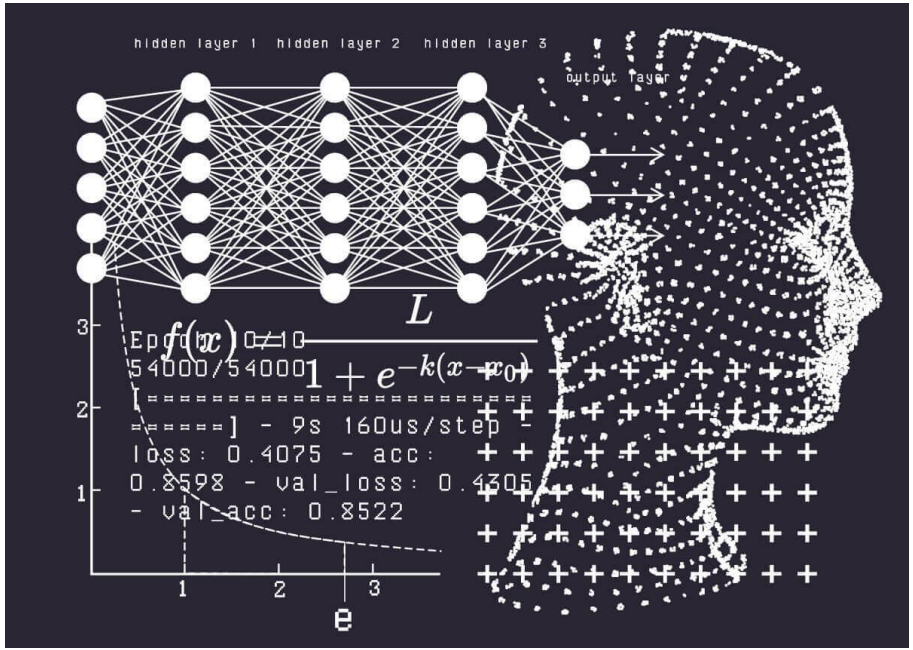
DL Sieci neuronowe

Uczenie ze wzmocnieniem



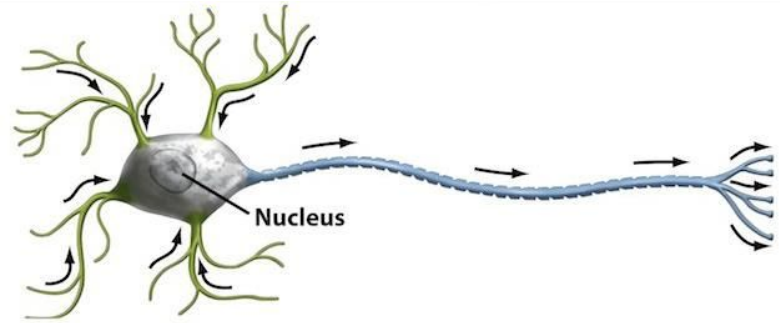
DL Sieci neuronowe

Sieć neuronowa



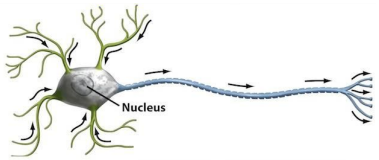
DL Sieci neuronowe

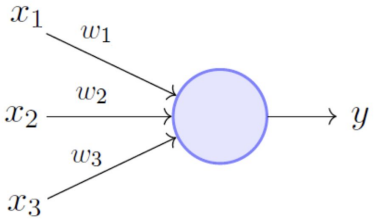
Koncepcja sieci neuronowej



DL Sieci neuronowe

Koncepcja sieci neuronowej

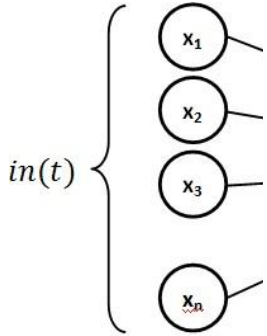




Perceptron Model (Minsky-Papert in 1969)

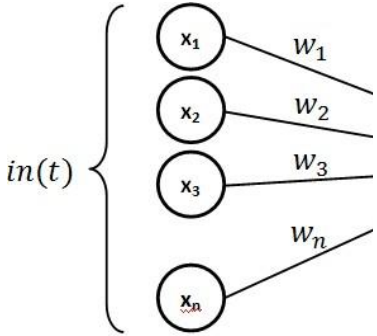
DL Sieci neuronowe

Koncepcja sieci neuronowej - perceptron



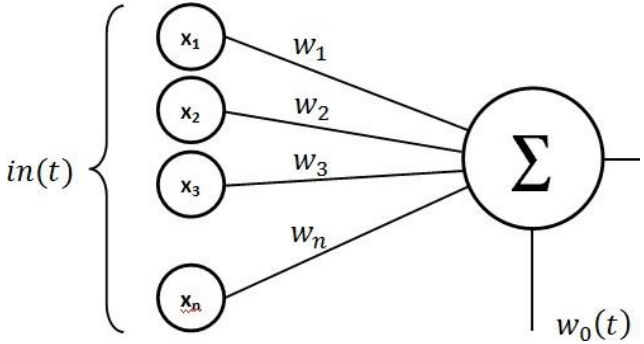
DL Sieci neuronowe

Koncepcja sieci neuronowej - perceptron



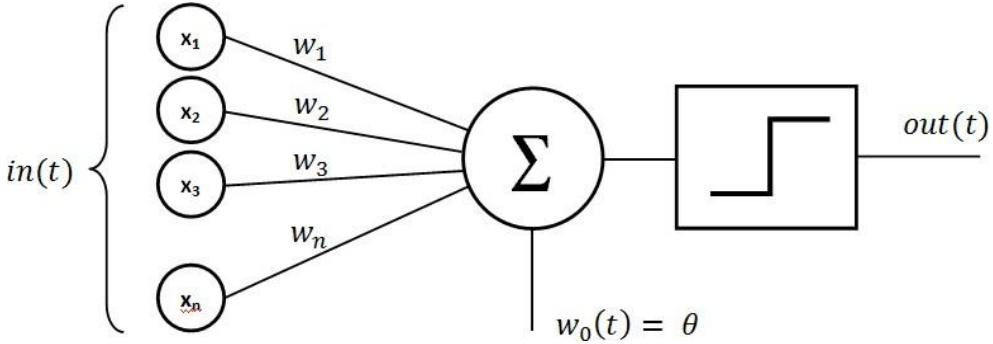
DL Sieci neuronowe

Koncepcja sieci neuronowej - perceptron



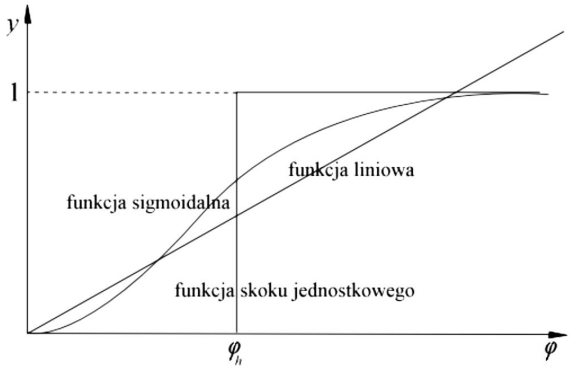
DL Sieci neuronowe

Koncepcja sieci neuronowej - perceptron



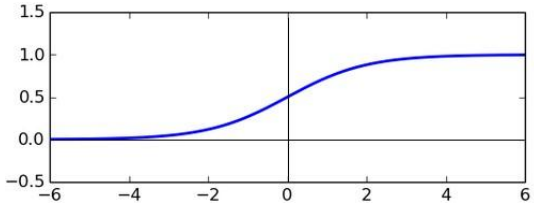
DL Sieci neuronowe

Funkcje aktywacji



DL Sieci neuronowe

Funkcje aktywacji

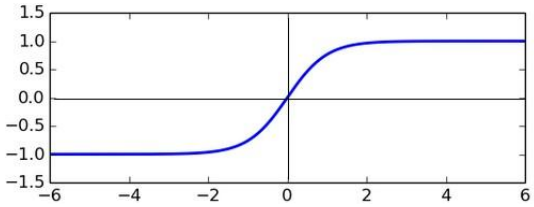


Sigmoid

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

DL Sieci neuronowe

Funkcje aktywacji

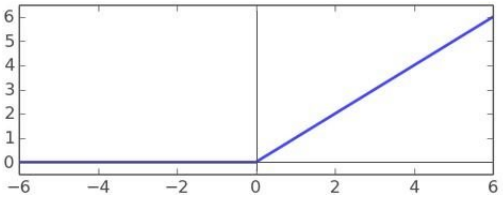


Hyperbolic Tangent

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

DL Sieci neuronowe

Funkcje aktywacji

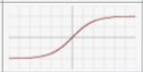
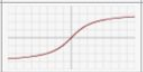


Rectified Linear

$$\phi(z) = \begin{cases} 0 & if\ z < 0 \\ z & if\ z \geq 0 \end{cases}$$

DL Sieci neuronowe

Funkcje aktywacji

Name	Plot	Equation	Derivative
Identity		$f(x) = x$	$f'(x) = 1$
Binary step		$f(x) = \begin{cases} 0 & \text{for } x < 0 \\ 1 & \text{for } x \geq 0 \end{cases}$	$f'(x) = \begin{cases} 0 & \text{for } x \neq 0 \\ ? & \text{for } x = 0 \end{cases}$
Logistic (a.k.a Soft step)		$f(x) = \frac{1}{1+e^{-x}}$	$f'(x) = f(x)(1-f(x))$
TanH		$f(x) = \tanh(x) = \frac{2}{1+e^{-2x}} - 1$	$f'(x) = 1 - f(x)^2$
ArcTan		$f(x) = \tan^{-1}(x)$	$f'(x) = \frac{1}{x^2+1}$
Rectified Linear Unit (ReLU)		$f(x) = \begin{cases} 0 & \text{for } x < 0 \\ x & \text{for } x \geq 0 \end{cases}$	$f'(x) = \begin{cases} 0 & \text{for } x < 0 \\ 1 & \text{for } x \geq 0 \end{cases}$
Parameteric Rectified Linear Unit (PReLU) [2]		$f(x) = \begin{cases} \alpha x & \text{for } x < 0 \\ x & \text{for } x \geq 0 \end{cases}$	$f'(x) = \begin{cases} \alpha & \text{for } x < 0 \\ 1 & \text{for } x \geq 0 \end{cases}$
Exponential Linear Unit (ELU) [3]		$f(x) = \begin{cases} \alpha(e^x - 1) & \text{for } x < 0 \\ x & \text{for } x \geq 0 \end{cases}$	$f'(x) = \begin{cases} f(x) + \alpha & \text{for } x < 0 \\ 1 & \text{for } x \geq 0 \end{cases}$
SoftPlus		$f(x) = \log_e(1 + e^x)$	$f'(x) = \frac{1}{1+e^{-x}}$

Zadanie 18.1 (instrukcja)

Napisz poniższe funkcje aktywacji w języku Python i zobrazuj na wykresach ich funkcjonowanie:

$$\text{sig}(x) = \frac{1}{1+\exp^{-x}}$$

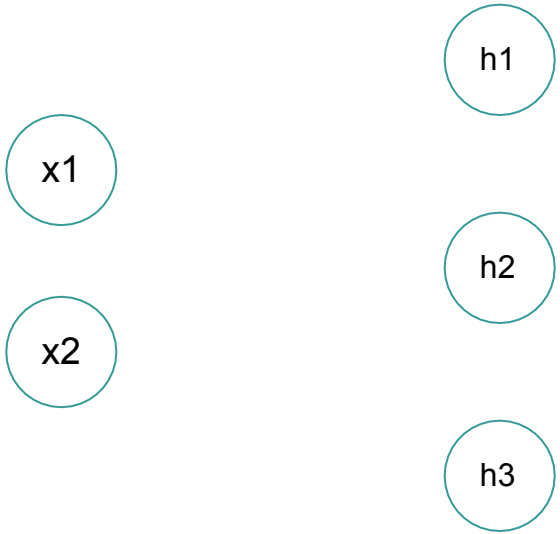
$$\text{relu}(x) = \begin{cases} 0, x \leq 0 \\ x, x > 0 \end{cases}$$

DL Sieci neuronowe

Warstwy

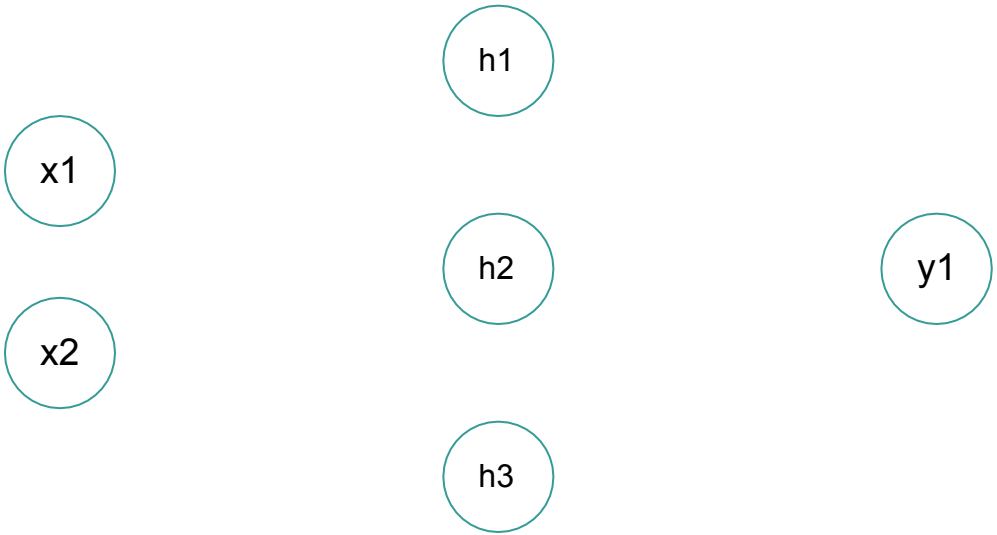
DL Sieci neuronowe

Warstwy



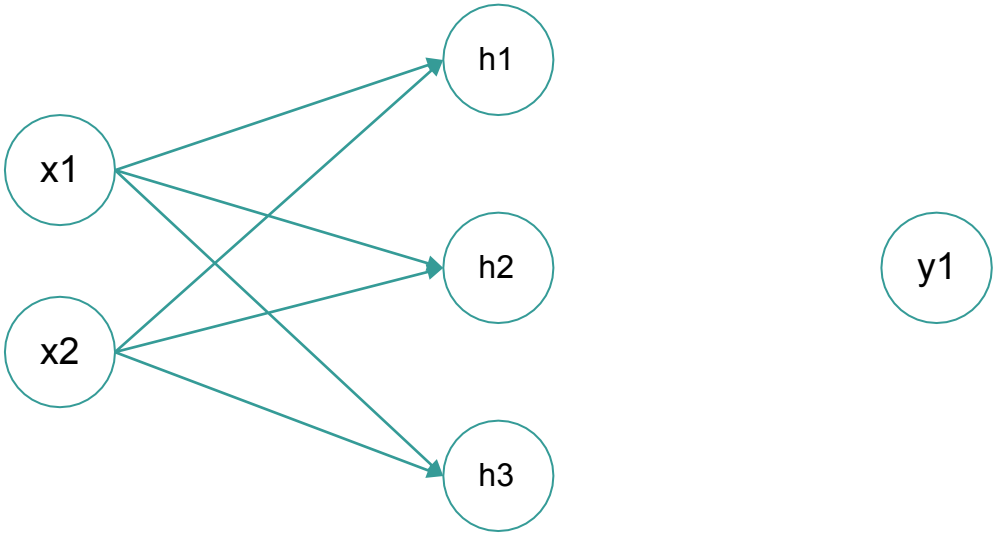
DL Sieci neuronowe

Warstwy



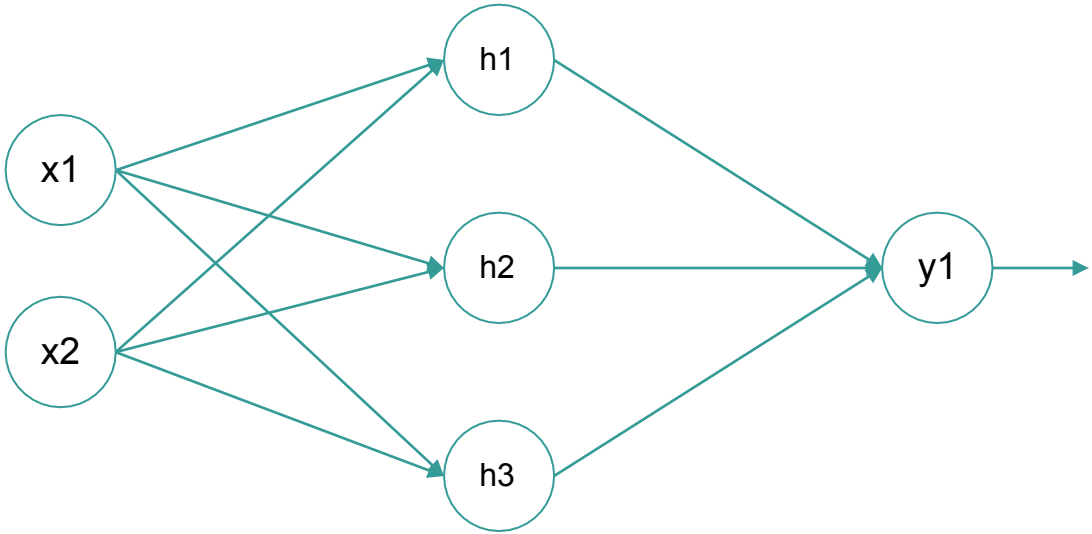
DL Sieci neuronowe

Warstwy



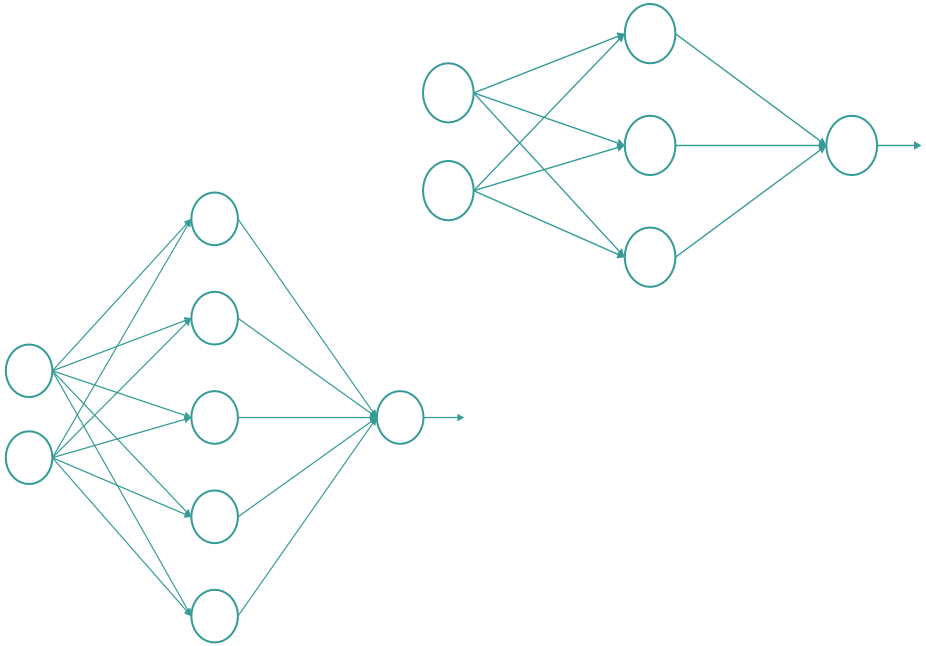
DL Sieci neuronowe

Warstwy



DL Sieci neuronowe

Modyfikowalność



infoShareAcademy.com

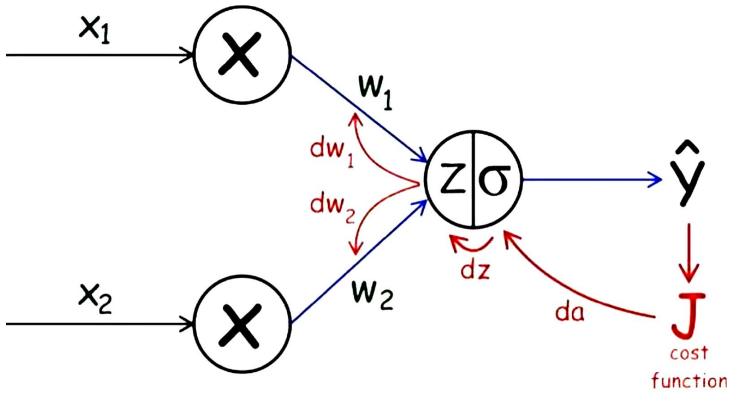
DL Sieci neuronowe

Optymalizacja. Funkcja kosztu

$$E_{\text{total}} = \sum \frac{1}{2}(\text{ target } - \text{ output })^2$$

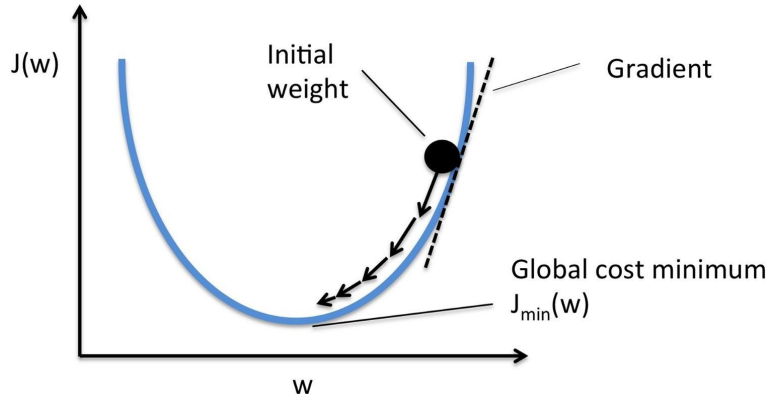
DL Sieci neuronowe

Optymalizacja. Wsteczna propagacja



DL Sieci neuronowe

Optymalizacja. Gradient descent



DL Sieci neuronowe

Optymalizacja. Korekta wag

Wnew = Wcurr - lr*(dErr/Wcurr)

(dErr/Wcurr) – pochodna błędu po danej wadze, w tensorflow będzie to tzw. gradient.

DL Sieci neuronowe

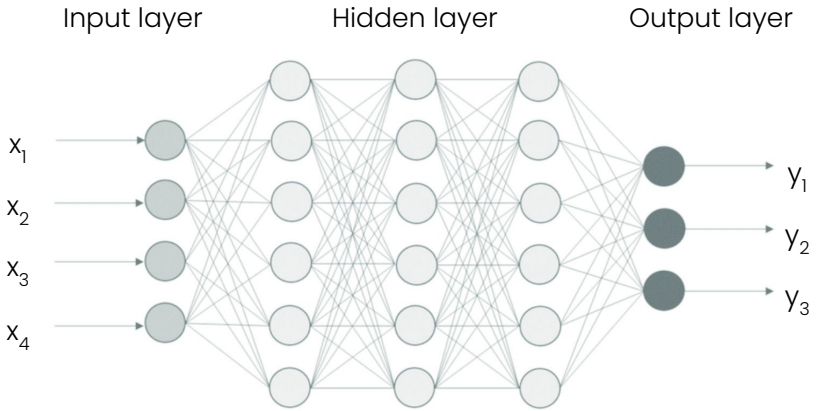
Optymalizacja. Wsteczna propagacja. Chain rule

$$F(x) = f(g(x))$$

$$F'(x) = f'(g(x))g'(x)$$

DL Sieci neuronowe

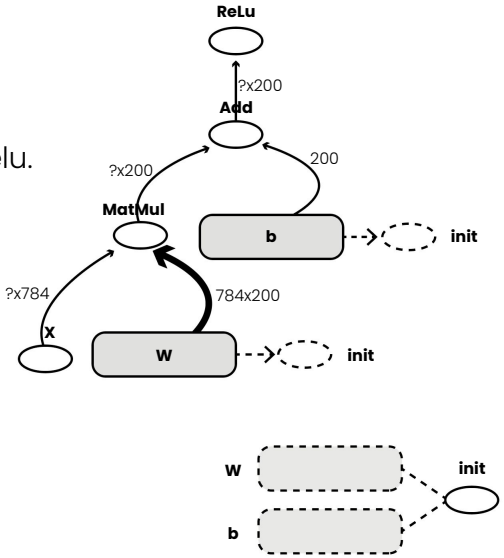
Podsumowanie



DL Sieci neuronowe

Tensorflow

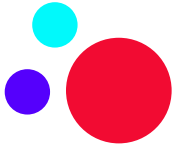
- Tworzenie modeli DL.
- Bazuje na tensorach.
- Tworzy z operacji graf modelu.
- Zoptymalizowane pod gpu.



DL Sieci neuronowe

Tensorflow - Eager execution

Praca z modelem sieci w locie.
Oznacza to, że definiując konkretne zmienne wchodzące w skład modelu, tensorflow sam dołącza je do reszty.

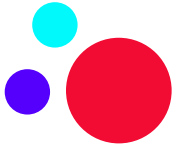


DL Sieci neuronowe

Tensorflow - Eager execution


```
import tensorflow as tf

# Sprawdzenie dostępnych GPU
gpus = tf.config.list_physical_devices('GPU')

print("Num GPUs Available: ", len(gpus))
```

Num GPUs Available: 1



DL Sieci neuronowe

Tensorflow - Eager execution

Stała: wartość której zmienić nie możemy w trakcie jej istnienia.

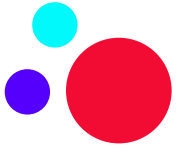
```
tf_const = tf.constant(1)
```

```
tf_const
```

```
<tf.Tensor: shape=(), dtype=int32, numpy=1>
```

```
type(tf_const)
```

```
tensorflow.python.framework.ops.EagerTensor
```

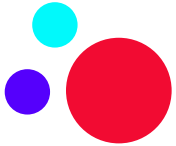


DL Sieci neuronowe

Tensorflow - Eager execution

```
# Tworzenie stałej w TensorFlow
    someConst = tf.constant([[1,2],[3,4],[5,6]], dtype="float64")
    someConst

        <tf.Tensor: shape=(3, 2), dtype=float64, numpy=
        array([[1., 2.],
                    [3., 4.],
                    [5., 6.]])>
# Konwersja stałej do tablicy NumPy
    np_array=someConst.numpy()
    np_array

        array([[1., 2.],
                    [3., 4.],
                    [5., 6.]])

    someConst.shape.as_list()
                [3, 2]
```


infoShare ACADEMY



DL Sieci neuronowe

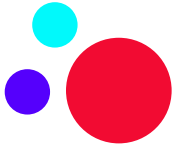
Tensorflow - Eager execution

Zmienna: możemy ją modyfikować. Mogą to być na przykład wagi danego modelu.

```
np.arange(6).reshape(2,3)

  array([[0, 1, 2],
         [3, 4, 5]])
```

```
someVar = tf.Variable(np.arange(6).reshape(2,3), dtype="float", name="wagi")
someVar
```

```
<tf.Variable 'wagi:0' shape=(2, 3) dtype=float32, numpy=
array([[0., 1., 2.],
       [3., 4., 5.]], dtype=float32)>
```



DL Sieci neuronowe

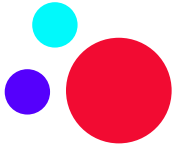
Tensorflow - Eager execution

someVar.dtype

```
tf.float32
```

someVar.numpy()

```
array([[0., 1., 2.],
       [3., 4., 5.]], dtype=float32)
```



DL Sieci neuronowe

Tensorflow - Eager execution

```
someVar.assign(np.arange(6,12).reshape(2,3))
someVar
```

```
<tf.Variable 'wagi:0' shape=(2, 3) dtype=float32, numpy=
array([[ 6.,  7.,  8.],
       [ 9., 10., 11.]], dtype=float32)>
```

```
np.ones(shape=(2,3))
```

```
array([[1., 1., 1.],
       [1., 1., 1.]])
```

```
someVar.assign_add(np.ones(shape=(2,3)))
someVar
```

```
<tf.Variable 'wagi:0' shape=(2, 3) dtype=float32, numpy=
array([[14., 16., 18.],
       [20., 22., 24.]], dtype=float32)>
```


infoShare ACADEMY

DL Sieci neuronowe

Tensorflow – obliczenia macierzowe

Obliczenia macierzowe:

```
a = tf.constant(np.arange(10))
b = tf.constant(np.arange(10))
c = tf.add(a, b)
c
```

```
<tf.Tensor: shape=(10,), dtype=int32, numpy=array([ 0,  2,  4,  6,  8, 10, 12, 14, 16, 18])>
```

```
print(f"{a}\n+\n{b}\n=\n{c}")
```

```
[0 1 2 3 4 5 6 7 8 9]
+
[0 1 2 3 4 5 6 7 8 9]
=
[ 0  2  4  6  8 10 12 14 16 18]
```

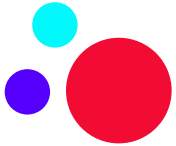
DL Sieci neuronowe

Tensorflow – obliczenia macierzowe

```
a = np.arange(10).reshape(10,1)
b = np.arange(10).reshape(1,10)
d = tf.matmul(a ,b)
print(f"{a}\n*\n{b}\n=\n{d.numpy()}")
```

```
[[0]
 [1]
 [2]
 [3]
 [4]
 [5]
 [6]
 [7]
 [8]
 [9]]
*
[[0 1 2 3 4 5 6 7 8 9]]
=
[[ 0  0  0  0  0  0  0  0  0  0]
 [ 0  1  2  3  4  5  6  7  8  9]
 [ 0  2  4  6  8 10 12 14 16 18]
 [ 0  3  6  9 12 15 18 21 24 27]
 [ 0  4  8 12 16 20 24 28 32 36]
 [ 0  5 10 15 20 25 30 35 40 45]
 [ 0  6 12 18 24 30 36 42 48 54]
 [ 0  7 14 21 28 35 42 49 56 63]
 [ 0  8 16 24 32 40 48 56 64 72]
 [ 0  9 18 27 36 45 54 63 72 81]]
```


infoShare ACADEMY



DL Sieci neuronowe

Tensorflow – obliczenia macierzowe

```
a = np.arange(10)
b = np.arange(10)
d = tf.multiply(a,b)
print(f"{a}\n*\n{b}\n=\n{d}")
```

```
[0 1 2 3 4 5 6 7 8 9]
*
[0 1 2 3 4 5 6 7 8 9]
=
[ 0  1  4  9 16 25 36 49 64 81]
```

```
type(d)
```

```
tensorflow.python.framework.ops.EagerTensor
```

DL Sieci neuronowe

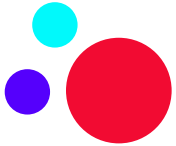
Tensorflow – gradient type

Mechanizm automatycznego wyznaczania gradientów z podanego fragmentu modelu.

Równanie wielomianu:

$$poly(w1,w2)=3*w21+4*w32$$

```
w1 = tf.Variable([2.0])
w2 = tf.Variable([3.0])

def create_poly(w1, w2):
    return 3*w1**2 + 4*w2**3
```

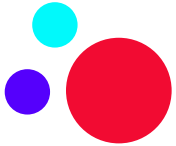


DL Sieci neuronowe

Tensorflow – gradient type

```
with tf.GradientTape() as tape:
    part1 = 4*w2**3
    poly = 3*w1**2 + part1
    #poly = create_poly(w1, w2)
    # 3*w1^2 + 4*w2^3

grad = tape.gradient(poly, [w1, w2])
# po w1 -> 3*2*w1^1 + 0 -> 3*2*2 = 12
# po w2 -> 0 + 4*3*w2^2 -> 4*3*3^2 = 108
print(f"grad = {grad[0].numpy()}, {grad[1].numpy()}")


    grad = [12.], [108.]
```


infoShare ACADEMY



DL Sieci neuronowe

Tensorflow – gradient type


```
s = 3
x_init = [1,2,3]
tf_vars = []
for i in range(s):
    tf_vars.append(tf.Variable([x_init[i]]))
tf_vars
```

```
[<tf.Variable 'Variable:0' shape=(1,) dtype=int32, numpy=array([1])>,
 <tf.Variable 'Variable:0' shape=(1,) dtype=int32, numpy=array([2])>,
 <tf.Variable 'Variable:0' shape=(1,) dtype=int32, numpy=array([3])>]
```

Zadanie 18.2 (instrukcja)

Napisz funkcję która przyjmuje jako parametr:

- stopień wielomianu,
- wartość inicjalizującą X,
- wektor stałych współczynników przy X.

Zwracać funkcja powinna wartość gradientu.
Przykładowo dla wielomianu trzeciego rzędu: a+bx+cx^2 pochodna po x zwrócić powinna wynik z: 0+b+2cx.

Wskazówki:

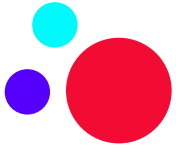
- x to zmienne tensorflowa,
- współczynniki (a, b, c, ...) to stałe tensorflowa.

DL Sieci neuronowe

Tensorflow - podsumowanie

TensorFlow

infoShareAcademy.com



DL Sieci neuronowe

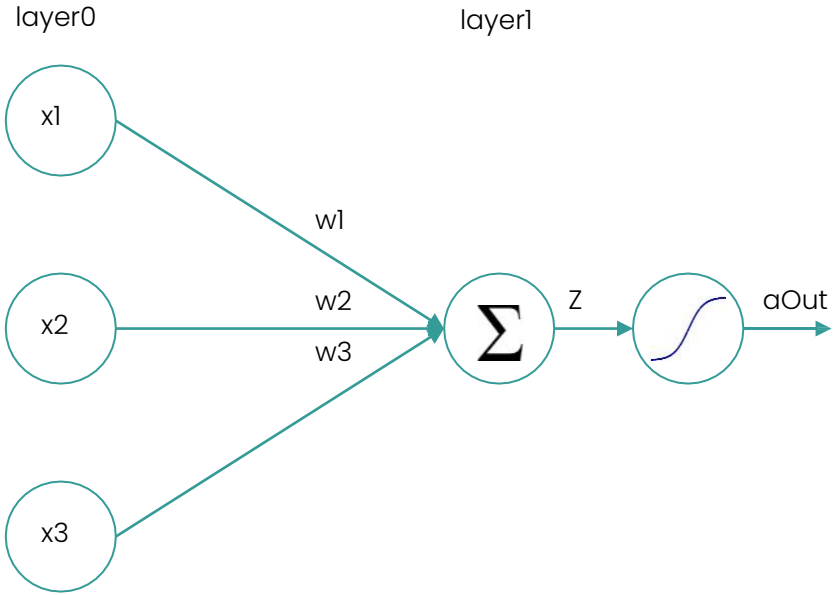
Implementacja sieci

TensorFlow

infoShareAcademy.com

DL Sieci neuronowe

Implementacja sieci neuronowej: dwuwarstwowa



DL Sieci neuronowe

Implementacja sieci neuronowej: dwuwarstwowa

Forward:

$$Z = input * W$$

$$y = f_{activation}(Z)$$

DL Sieci neuronowe

Implementacja sieci neuronowej: dwuwarstwowa

Backward:

$$W_{new} = W_{current} - lr * \frac{dError}{dW}$$

$$\frac{dError}{dW} = \frac{dError}{dActivationOut} * \frac{dActivationOut}{dZ} * \frac{dZ}{dW}$$

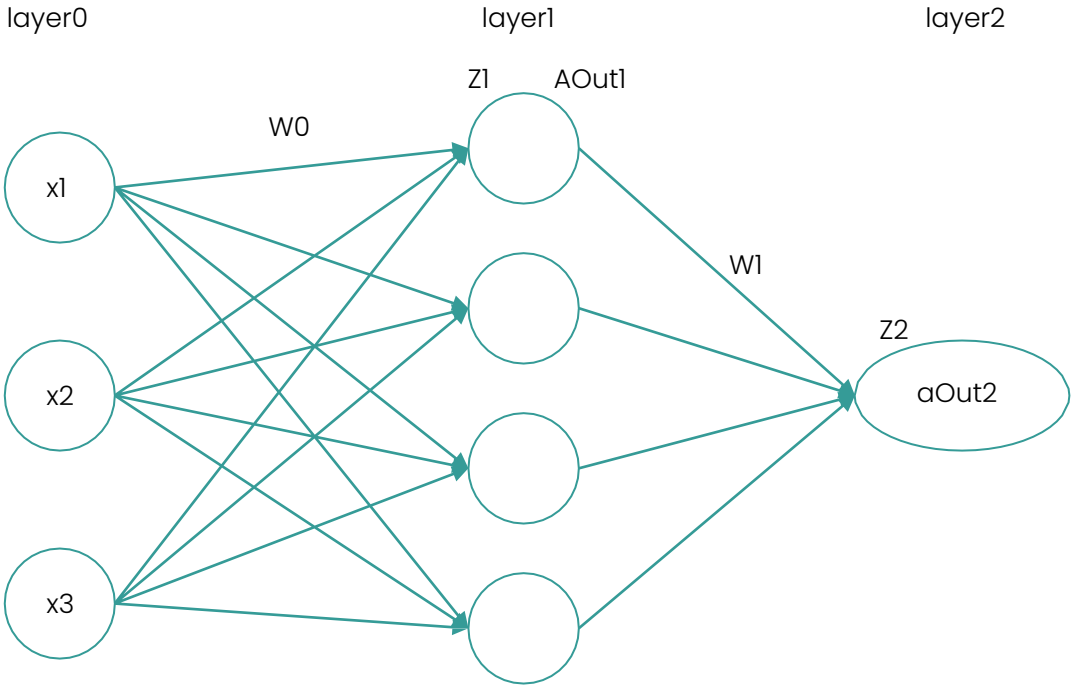
$$\frac{dError}{dActivationOut} = activationOut - y_{reference}$$

$$\frac{dActivationOut}{dZ} = ActivationFunctionDerivaive(Z)$$

$$\frac{dZ}{dW} = activationOut_{previousLayer}$$

DL Sieci neuronowe

Implementacja sieci neuronowej: trójwarstwowa



infoShare ACADEMY

DL Sieci neuronowe

Implementacja sieci neuronowej: trójwarstwowa

Forward:

$$Z_1 = input * W_0$$
$$activationOut_1 = f_{activation}(Z_1)$$
$$Z_2 = activationOut_1 * W_1$$
$$y = f_{activation}(Z_2)$$

DL Sieci neuronowe

Implementacja sieci neuronowej: trójwarstwowa

Backward:

third layer/output(2)

$$W_{1new} = W_{1current} - lr * \frac{dError}{dW_1}$$

$$\frac{dError}{dW_1} = \frac{dError}{dActivationOut_2} * \frac{dActivationOut}{dZ_2} * \frac{dZ_2}{dW_1}$$

$$\frac{dError}{dActivationOut_2} = activationOut_2 - y_{reference}$$

$$\frac{dActivationOut}{dZ_2} = ActivationFunctionDerivaive(Z_2)$$

$$\frac{dZ_2}{dW_1} = activationOut_1^T$$

DL Sieci neuronowe

Implementacja sieci neuronowej: trójwarstwowa

Second layer(1)

$$W_{0new} = W_{0current} - lr * \frac{dError}{dW_0}$$

$$\frac{dError}{dW_0} = \frac{dError}{dActivationOut_1} * \frac{dActivationOut}{dZ_1} * \frac{dZ_1}{dW_0}$$

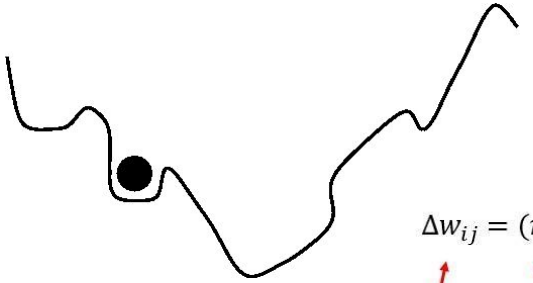
$$\frac{dError}{dActivationOut_1} = \frac{dError}{dActivationOut_2} * \frac{dActivationOut}{dZ_2} * W_1^T$$

$$\frac{dActivationOut}{dZ_1} = ActivationFunctionDerivaive(Z_1)$$

$$\frac{dZ_1}{dW_0} = input^T$$

DL Sieci neuronowe

Co możemy do sieci dołączyć?
Człon momentum



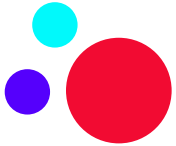
$$\Delta w_{ij} = (\eta * \frac{\partial E}{\partial w_{ij}})$$

weight increment — learning rate — weight gradient

$$\Delta w_{ij} = (\eta * \frac{\partial E}{\partial w_{ij}}) + (\gamma * \Delta w_{ij}^{t-1})$$

momentum factor — weight increment, previous iteration

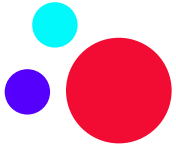
infoShare ACADEMY



DL Sieci neuronowe

Zalety

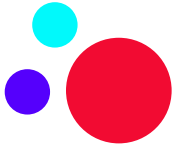
- Teoretycznie do każdego problemu można dostosować odpowiednią sieć neuronową.
- Radzi sobie z dużą ilością danych.
- Dowolne wejście/wyjście.
- Skuteczne zarówno dla regresji jak i klasyfikacji.



DL Sieci neuronowe

Wady

- Zanikający gradient.
- Eksplodujący gradient.
- Złożoność obliczeniowa.
- Wielkość.



DL Sieci neuronowe

Głęboka sieć neuronowa

- Zadaniem będzie rozwiązanie problemu klasyfikacji raka: sklearn.datasets.load_breast_cancer()
- Model sieci będzie wielowarstwowy.
- Ustawiać będziemy mogli dowolną liczbę warstw ukrytych.
- Funkcja błędu dla klasyfikacji - softmax_cross_entropy_with_logits.
- Dodamy optymalizator.
- Dodajemy warstwę (mechanizm) dropout.

DL Sieci neuronowe

Dane

```
from sklearn.datasets import load_breast_cancer
data = load_breast_cancer()

data.feature_names
```

```
array(['mean radius', 'mean texture', 'mean perimeter', 'mean area',
       'mean smoothness', 'mean compactness', 'mean concavity',
       'mean concave points', 'mean symmetry', 'mean fractal
dimension',
       'radius error', 'texture error', 'perimeter error', 'area error',
       'smoothness error', 'compactness error', 'concavity error',
       'concave points error', 'symmetry error',
       'fractal dimension error', 'worst radius', 'worst texture',
       'worst perimeter', 'worst area', 'worst smoothness',
       'worst compactness', 'worst concavity', 'worst concave points',
       'worst symmetry', 'worst fractal dimension'], dtype='<U23')
```


infoShare ACADEMY

DL Sieci neuronowe

Dane

data.target_names

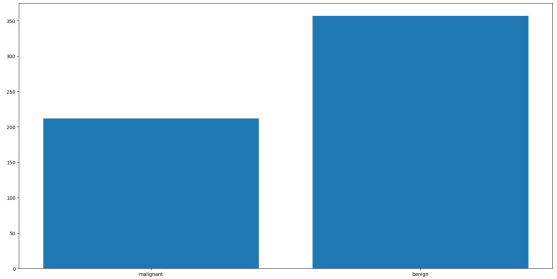
```
array(['malignant', 'benign'], dtype='<U9')
```

```
x = data.data.astype(np.float32)
y = data.target.astype(np.float32)
```

```
classVals, classCnts = np.unique(y, return_counts=True)
for oneName, oneVal, oneCnt in zip(data.target_names, classVals, classCnts):
    print(f"{oneName} {oneVal}: {oneCnt}")
```

malignant 0.0: 212

benign 1.0: 357



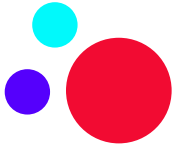
DL Sieci neuronowe

Dane

```
from sklearn.preprocessing import MinMaxScaler

x = MinMaxScaler().fit_transform(x)

for i in range(x.shape[1]):
    print(f"{i} mean: {x[:,i].mean():.2} std: {x[:, i].std():.2}")
```

```
0 mean: 0.34 std: 0.17
1 mean: 0.32 std: 0.15
2 mean: 0.33 std: 0.17
3 mean: 0.22 std: 0.15
4 mean: 0.39 std: 0.13
5 mean: 0.26 std: 0.16
6 mean: 0.21 std: 0.19
7 mean: 0.24 std: 0.19
8 mean: 0.38 std: 0.14
9 mean: 0.27 std: 0.15
10 mean: 0.11 std: 0.1
```


infoShare ACADEMY



DL Sieci neuronowe

Dane

```
from sklearn.preprocessing import OneHotEncoder


y = y.reshape(-1, 1)
y = OneHotEncoder(sparse=False).fit_transform(y)
y

        array([[1., 0.],
               [1., 0.],
               [1., 0.],
               ...,
               [1., 0.],
               [1., 0.],
               [0., 1.]])
```


infoShare ACADEMY

DL Sieci neuronowe

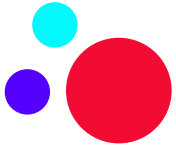
Dane

```
from sklearn.model_selection import train_test_split

x_train, x_test, y_train, y_test = train_test_split(x, y, test_size=0.2, random_state=42)

x_train.shape
```

```
(455, 30)
```

```
x_test.shape
```

```
(114, 30)
```

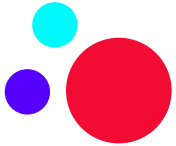


DL Sieci neuronowe

Model

```
class modelDNN(object):
    def __init__(self, inSize, outSize, outAct, hiddenNum, hiddenAct, opt, job_type):
        self.lr = tf.Variable(0.0, dtype="float")
self.W_h = []
        self.b_h = []
#pierwsza ukryta
        self.W_h.append(tf.Variable(tf.random.normal([inSize, hiddenNum[0]])))
        self.b_h.append(tf.Variable(tf.zeros([hiddenNum[0]])))
        # kolejne ukryte
        for i in range(1, len(hiddenNum)):
            self.W_h.append(tf.Variable(tf.random.normal([hiddenNum[i-1], hiddenNum[i]])))
            self.b_h.append(tf.Variable(tf.zeros([hiddenNum[i]])))
# warstwa wyjściowa
        self.W_out = tf.Variable(tf.random.normal([hiddenNum[-1], outSize]))
        self.b_out = tf.Variable(tf.zeros([outSize]))
                self.inSize = inSize
        self.outSize = outSize
        self.outAct = outAct
        self.hiddenAct = hiddenAct
                self.optimizer = opt
self.job_type = job_type
```


infoShare ACADEMY



DL Sieci neuronowe

Model

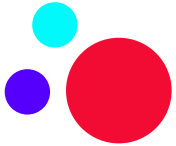
```
def predict(self, x):
    layer_in = x

    # pierwsza warstwa ukryta
    layer_hidden = tf.add(tf.matmul(layer_in, self.W_h[0]), self.b_h[0])
    layer_hidden = self.hiddenAct[0](layer_hidden)

    # kolejne warstwy ukryte
    for i in range(1, len(self.W_h)):
        layer_hidden = tf.add(tf.matmul(layer_hidden, self.W_h[i]), self.b_h[i])
        layer_hidden = self.hiddenAct[i](layer_hidden)

    # warstwa wyjściowa
    layer_out = tf.add(tf.matmul(layer_hidden, self.W_out), self.b_out)
    if self.outAct != None:
        layer_out = self.outAct(layer_out)

    return layer_out
```


infoShare ACADEMY

DL Sieci neuronowe

Model

```
def lossFun(self, y_pred, y_true):
        y_true = tf.reshape(y_true, (-1, self.outSize))
        if self.job_type == "class":
            return
tf.reduce_mean(tf.nn.softmax_cross_entropy_with_logits(y_true, y_pred))
        elif self.job_type == "regr":
            return tf.reduce_mean(tf.square(y_pred - y_true))
```



DL Sieci neuronowe

Model

```
def fit(self, x, y_true, lr=None):
    if lr != None:
        self.optimizer.learning_rate.assign(lr)

    with tf.GradientTape() as t:
        y_pred = self.predict(x)
        current_loss = self.lossFun(y_pred, y_true)

    gradient = t.gradient(
        current_loss,
        [*self.W_h, *self.b_h, self.W_out, self.b_out]
    )

    self.optimizer.apply_gradients(zip(
        gradient,
        [*self.W_h, *self.b_h, self.W_out, self.b_out]
    ))

    return current_loss
```


info Share ACADEMY

DL Sieci neuronowe

Dobór funkcji kosztu

Dobór funkcji kosztu uzależniony jest od rozkładu wartości wyjściowych i typu neuronów wyjściowych

Output Type	Output Distribution	Output Layer	Cost Function
Output Type	Bernoulli	Sigmoid	Binary cross-entropy
Discrete	Multinoulli	Softmax	Binary cross-entropy
Continuous	Gaussian	Linear	Gaussian cross-entropy (MSE)

DL Sieci neuronowe

Model

```
inSize = x.shape[1]
outSize = classNum
outAct = None
hiddenNum = [16, 24, 32]
hiddenAct = [tf.nn.relu for i in range(len(hiddenNum))]
opt = tf.optimizers.Adam()
job_type = "class"
nowyModel = modelDNN(inSize, outSize, outAct, hiddenNum, hiddenAct, opt, job_type)
lr = None #0.001 - tf.optimizers.Adam().learning_rate
epochsNum = 150
lossList = []

for i in range(epochsNum):
    tmpLoss = nowyModel.fit(
                    x_train,
                    y_train,
                    lr=lr
                    )
    #print(f"curr loss: {tmpLoss}")
    lossList.append(tmpLoss)
```


info Share ACADEMY

DL Sieci neuronowe

Model

```
# prezentacja na mniejszym zbiorze z testu
y_pred = tf.nn.softmax(
        nowyModel.predict(
            x_test[:25]
        )
    ).numpy().argmax(axis=1)

print(f"Wyjścia z sieci:\n{y_pred}")
print(f"Co powinno być:\n{y_test[:25].argmax(axis=1)}")
```

Wyjścia z sieci:

[1 0 0 0 1 0 0 0 0 1 1 0 1 0 1 0 1 1 1 0 0 1 0 1 1]

Co powinno być:

[1 0 0 1 1 0 0 0 1 1 1 0 1 0 1 0 1 1 1 0 0 1 0 1 1]

infoShare ACADEMY

DL Sieci neuronowe

Model

```
# sprawdzamy na zbiorze treningowym
print("\nRaport klasyfikacji na zbiorze treningowym")
y_pred = tf.nn.softmax(
        nowyModel.predict(
          x_train)).numpy().argmax(axis=1)
print(classification_report(y_train.argmax(axis=1), y_pred))
```

Raport klasyfikacji na zbiorze treningowym

	precision	recall	f1-score	support
0	0.81	0.85	0.83	169
1	0.91	0.88	0.89	286
accuracy			0.87	455
macro avg	0.86	0.86	0.86	455
weighted avg	0.87	0.87	0.87	455

Raport klasyfikacji na zbiorze testowym

	precision	recall	f1-score	support
0	0.73	0.81	0.77	43
1	0.88	0.82	0.85	71
accuracy			0.82	114
macro avg	0.80	0.82	0.81	114
weighted avg	0.82	0.82	0.82	114

infoShare ACADEMY

DL Sieci neuronowe

Model

```
# testujemy na całości
print("\nRaport klasyfikacji na zbiorze testowym")
y_pred = tf.nn.softmax(
        nowyModel.predict(
            x_test
        )
    ).numpy().argmax(axis=1)
print(classification_report(y_test.argmax(axis=1), y_pred))
print("\n")
fig = plt.figure(figsize=(20,10))
plt.plot(lossList)

plt.show()
```

Wyjścia z sieci:

[1 0 0 1 1 0 0 0 0 1 1 0 1 0 1 0 1 1 1 0 0 1 0 1 1]

Co powinno być:

[1 0 0 1 1 0 0 0 1 1 1 0 1 0 1 0 1 1 1 0 0 1 0 1 1]

DL Sieci neuronowe

Wyniki

Wyjścia z sieci:
[1 0 0 1 1 0 0 0 0 1 1 0 1 0 1 0 1 1 1 0 0 1 0 1 1]
Co powinno być:
[1 0 0 1 1 0 0 0 1 1 1 0 1 0 1 0 1 1 1 0 0 1 0 1 1]

DL Sieci neuronowe

Model

Raport klasyfikacji na zbiorze treningowym

	precision	recall	f1-score	support
0	0.78	0.82	0.80	169
1	0.89	0.87	0.88	286
accuracy			0.85	455
macro avg	0.84	0.84	0.84	455
weighted avg	0.85	0.85	0.85	455

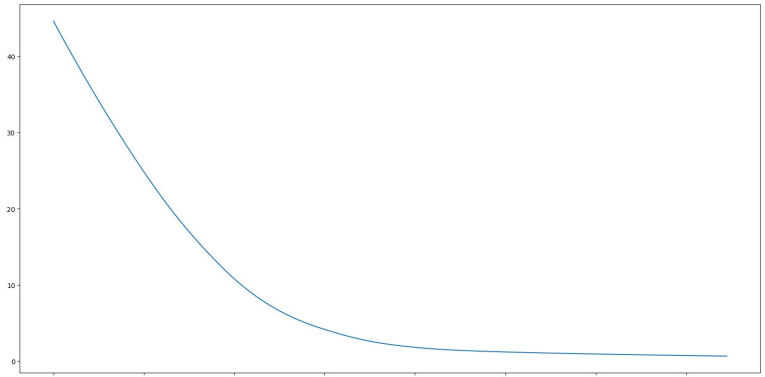
Raport klasyfikacji na zbiorze testowym

	precision	recall	f1-score	support
0	0.79	0.86	0.82	43
1	0.91	0.86	0.88	71
accuracy			0.86	114
macro avg	0.85	0.86	0.85	114
weighted avg	0.86	0.86	0.86	114

infoShare ACADEMY

DL Sieci neuronowe

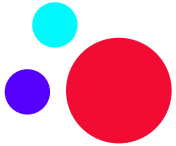
Wyniki



DL Sieci neuronowe

Keras

infoShareAcademy.com



DL Sieci neuronowe

Komponenty API Keras

- Warstwy
- Modele

DL Sieci neuronowe

Warstwy

Klasa tf.keras.layers.Layer jest podstawową abstrakcją w Keras.
Layer zawiera stan (wagi) i niektóre obliczenia (zdefiniowane w metodzie tf.keras.layers.Layer.call).

Ciężary tworzone przez warstwy mogą być trenowane lub nie. Warstwy można komponować rekurencyjnie: jeśli przypiszesz instancję warstwy jako atrybut innej warstwy, warstwa zewnętrzna zacznie śledzić wagi utworzone przez warstwę wewnętrzną.

Warstwy można również używać do obsługi zadań wstępnego przetwarzania danych, takich jak normalizacja i wektoryzacja tekstu. Warstwy przetwarzania wstępnego można włączyć bezpośrednio do modelu podczas szkolenia lub po jego zakończeniu, dzięki czemu model jest przenośny.

infoShare ACADEMY

DL Sieci neuronowe

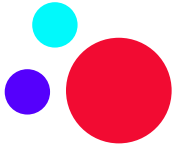
Modele

Model to obiekt, który grupuje warstwy i którego można uczyć na danych.

Najprostszym typem modelu jest model Sequential , który jest liniowym stosem warstw. W przypadku bardziej złożonych architektur można albo użyć funkcjonalnego interfejsu API Keras , który umożliwia budowanie dowolnych wykresów warstw, albo użyć podklas do napisania modeli od zera.

Klasa tf.keras.Model posiada wbudowane metody uczenia i ewaluacji:

tf.keras.Model.fit: Uczy model dla ustalonej liczby epok.
tf.keras.Model.predict: Generuje prognozy wyjściowe dla próbek wejściowych.
tf.keras.Model.evaluate: Zwraca wartości strat i metryk dla modelu; skonfigurowany za pomocą metody tf.keras.Model.compile.



DL Sieci neuronowe

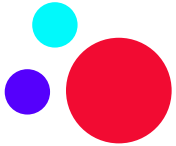
Model sekwencyjny

Sequential model jest odpowiedni dla zwykłego stosu warstw, przy czym każda warstwa ma dokładnie jeden tensor tensor wejście i jedno wyjście.

DL Sieci neuronowe

Model sekwencyjny

```
# Define Sequential model with 3 layers
model = keras.Sequential(
    [
        layers.Dense(2, activation="relu", name="layer1"),
        layers.Dense(3, activation="relu", name="layer2"),
        layers.Dense(4, name="layer3"),
    ]
)
# Call model on a test input
x = tf.ones((3, 3))
y = model(x)

# Create 3 layers
layer1 = layers.Dense(2, activation="relu", name="layer1")
layer2 = layers.Dense(3, activation="relu", name="layer2")
layer3 = layers.Dense(4, name="layer3")

# Call layers on a test input
x = tf.ones((3, 3))
y = layer3(layer2(layer1(x)))
```

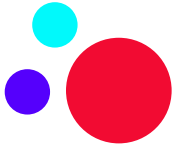


DL Sieci neuronowe

Model sekwencyjny

Model sekwencyjny nie jest odpowiedni, gdy:

- twój model ma wiele wejść lub wiele wyjść,
- każda z twoich warstw ma wiele wejść lub wiele wyjść,
- musisz udostępnić warstwy,
- potrzebujesz nieliniowej topologii (np. połączenie resztkowe, model wielorozgałęziony).



DL Sieci neuronowe

Tworzenie modelu sekwencyjnego

```
model = keras.Sequential(
    [
        layers.Dense(2, activation="relu"),
        layers.Dense(3, activation="relu"),
        layers.Dense(4),
    ]
)
```

lub

```
model = keras.Sequential()
model.add(layers.Dense(2, activation="relu"))
model.add(layers.Dense(3, activation="relu"))
model.add(layers.Dense(4))
```


infoShare ACADEMY

DL Sieci neuronowe

Tworzenie modelu sekwencyjnego

```
layer = layers.Dense(3)

x = tf.ones((1, 4))
y = layer(x)
layer.weights
```

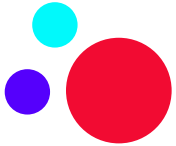
```
[<tf.Variable 'dense_6/kernel:0' shape=(4, 3) dtype=float32,
numpy=
 array([[ 0.5319189 , -0.8767905 , -0.63919735],
        [-0.6276014 ,  0.1689707 , -0.57695866],
        [ 0.6710613 ,  0.5354214 , -0.00893992],
        [ 0.15670097, -0.15280598,  0.8865864 ]], dtype=float32)>,
 <tf.Variable 'dense_6/bias:0' shape=(3,) dtype=float32,
numpy=array([0., 0., 0.], dtype=float32)>]
```

DL Sieci neuronowe

Tworzenie modelu sekwencyjnego

```
model = keras.Sequential(
    [
        layers.Dense(2, activation="relu"),
        layers.Dense(3, activation="relu"),
        layers.Dense(4),
    ]
)
# model.weights

# model.summary()

x = tf.ones((1, 4))
y = model(x)
print("Number of weights after calling the model:", len(model.weights))
```

```
Number of weights after calling the model: 6
```

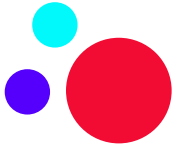


DL Sieci neuronowe

Tworzenie modelu sekwencyjnego

model.summary()

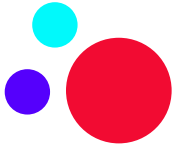
```
Model: "sequential_3"
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
dense_7 (Dense)              (1, 2)                    10
_________________________________________________________________
dense_8 (Dense)              (1, 3)                    9
_________________________________________________________________
dense_9 (Dense)              (1, 4)                    16
=================================================================
Total params: 35
Trainable params: 35
Non-trainable params: 0
_________________________________________________________________
```



DL Sieci neuronowe

Model sekwencyjny - implementacja

```
import tensorflow as tf
from sklearn.datasets import load_iris
from sklearn.model_selection import train_test_split
from sklearn.preprocessing import StandardScaler


iris = load_iris()
X_train, X_test, y_train, y_test = train_test_split(iris.data,
iris.target, test_size=0.2, random_state=42)

scaler = StandardScaler()
X_train_scaled = scaler.fit_transform(X_train)
X_test_scaled = scaler.transform(X_test)
```



DL Sieci neuronowe

Model sekwencyjny - implementacja

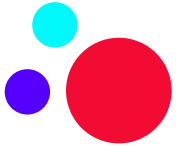
```
model = tf.keras.Sequential([
  tf.keras.layers.Dense(8, input_dim=4, activation='relu'),
  tf.keras.layers.Dense(3, activation='softmax')
])

model.compile(loss='sparse_categorical_crossentropy', optimizer='adam', metrics=['accuracy'])

model.fit(X_train_scaled, y_train, epochs=50, batch_size=10, validation_split=0.1)

test_loss, test_accuracy = model.evaluate(X_test_scaled, y_test)
print(f'Test accuracy: {test_accuracy * 100:.2f}%')
```

```
1/1 [==============================] - 0s 133ms/step - loss: 0.3110 - accuracy: 0.9333
Test accuracy: 93.33%
```

Zadanie 18.3 (instrukcja)

Zadaniem jest zbudowanie i wytrenowanie modelu sekwencyjnego w Tensorflow do przewidywania wystąpienia cukrzycy na podstawie zbioru danych dostępnego w bibliotece scikit-learn.

```
from sklearn.datasets import load_diabetes
```



DL Sieci neuronowe

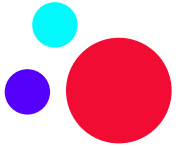
Podsumowanie

DL Sieci neuronowe

Google Colab

Google Colaboratory

infoShareAcademy.com



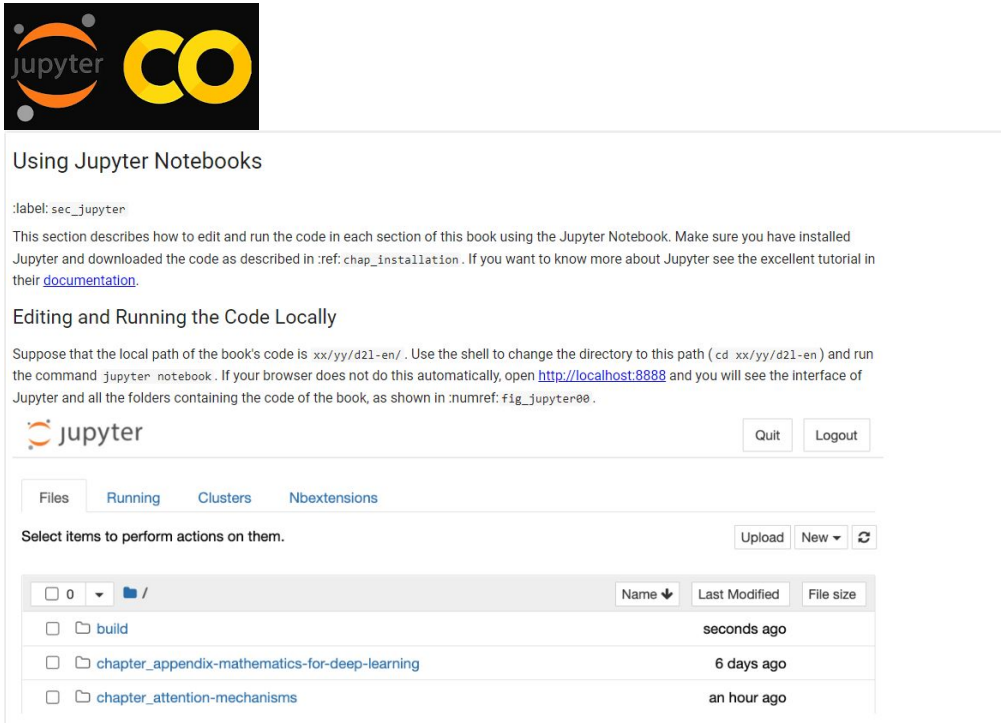
DL Sieci neuronowe

Google Colab – środowisko pracy

- Dostęp do GPU
- Notatniki Jupyter w Chmurze
- Łatwa Instalacja bibliotek
- Duża Ilość pamięci RAM
- Integracja z Google Drive
- Darmowy dostęp do zasobów obliczeniowych
- Wsparcie dla innych technologii
- Dodatkowe zasoby i tutoriale

DL Sieci neuronowe

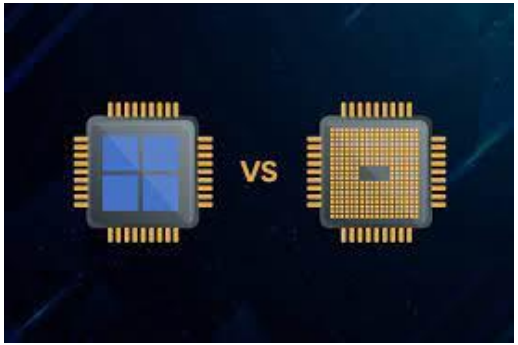
Google Colab – Jupyter Notebook



infoShareAcademy.com

DL Sieci neuronowe

Google Colab – zasoby obliczeniowe



DL Sieci neuronowe

Google Colab – integracja z Google Drive

infoShareAcademy.com

DL Sieci neuronowe

Google Colab – biblioteki

infoShareAcademy.com

DL Sieci neuronowe

Google Colab – bezpieczeństwo

Zadanie 18.4 (instrukcja)

Utwórz notatnik w Google Colab, w którym zbudujesz prosty model klasyfikacyjny przy użyciu biblioteki Tensorflow. Wykorzystaj zbiór danych Breast Cancer, aby stworzyć model, który będzie klasyfikował przypadki raka piersi na podstawie cech komórek.

```
from sklearn.datasets import load_breast_cancer
```

DL Sieci neuronowe

Podsumowanie