

Metody arbitrażu statystycznego

Michał Sierakowski

IBM, University of Warsaw

m.sierakowski@mimuw.edu.pl

Slack channel: <https://ibm.enterprise.slack.com/archives/C09KE8UFH33>

4 listopada 2025r.

Przegląd

- 1 Zasady
- 2 Wykład
- 3 Laby
 - Lab I.50
 - Lab I.51
- 4 Praca domowa
 - Lab H.I.52

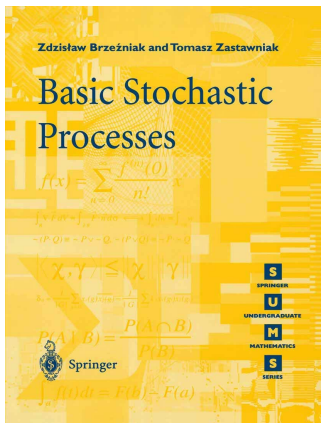
Zasady

- **ocena końcowa** = 40% ćwiczenia + 10% kolokwium + 15% prezentacja + 35% projekt
 - [0 – 50%): ocena 2
 - [50% – 60%): ocena 3
 - [60% – 70%): ocena 3,5
 - [70% – 80%): ocena 4
 - [80% – 90%): ocena 4,5
 - [90% – 100%]: ocena 5

Zasady: ćwiczenia (waga 40%)

- **indywidualne zadania w trakcie zajęć**
 - punktowane: $\{0, 1\}$
 - zrzut ekranu i wyjaśnienie na zajęciach
- **grupowe zadania w trakcie zajęć**
 - punktowane: $\{0, 1\}$ tylko dla **obecnych na zajęciach członków grupy** (co należy zaznaczyć zgłaszając rozwiązanie)
 - zrzut ekranu i wyjaśnienie na zajęciach
- **indywidualne zadania domowe**
 - punktowane: $\{0, 1\}$
 - **czas wykonania: do rozpoczęcia kolejnych zajęć**
 - zrzut ekranu lub listing
- **grupowe zadania domowe**
 - punktowane: $\{0, 1\}$ dla **wszystkich członków grupy**
 - **czas wykonania: do rozpoczęcia kolejnych zajęć**
 - zrzut ekranu lub listing

Motywacja



Pragmatycznie o AI

✓ Machine Learning

✓ Supervised ML

✓ Unsupervised ML

✓ Deep Learning

✓ Reinforcement Learning

Example: handwritten digits

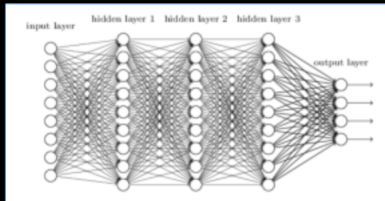
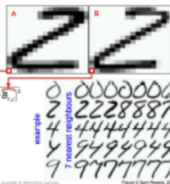
- 16x16 bitmaps
- 8-bit grayscale
- Euclidian distance

– over raw pixels

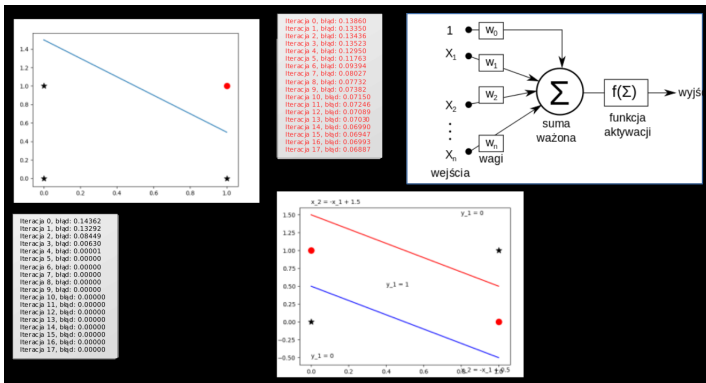
$$D(A,B) = \sqrt{\sum_x \sum_y (A_{x,y} - B_{x,y})^2}$$

- Accuracy:

- 7-NN ~ 95.2%
- SVM ~ 95.8%
- humans ~ 97.5%



Pragmatycznie o AI



Pragmatycznie o ML

Diagram of a simple neural network with an input layer (2 nodes), a hidden layer (3 nodes), and an output layer (2 nodes).

Listing 7.2 neuron.py

```
from typing import List, Callable
from util import dot_product

class Neuron:
    def __init__(self, weights: List[float], learning_rate: float, activation_function: Callable[[float], float], derivative_activation_function: Callable[[float], float]) -> None:
        self.weights: List[float] = weights
        self.activation_function: Callable[[float], float] = activation_function
        self.derivative_activation_function: Callable[[float], float] = derivative_activation_function
        self.learning_rate: float = learning_rate
        self.output_cache: float = 0.0
        self.delta: float = 0.0

    def output(self, inputs: List[float]) -> float:
        self.output_cache = dot_product(inputs, self.weights)
```

Listing 7.3 layer.py

```
from typing import List, Callable, Optional
from random import random
from neuron import Neuron

class Layer:
    def __init__(self, previous_layer: Optional[Layer], num_neurons: int, learning_rate: float, activation_function: Callable[[float], float], derivative_activation_function: Callable[[float], float]) -> None:
        self.previous_layer: Optional[Layer] = previous_layer
        self.neurons: List[Neuron] = []

    def add_neuron(self, num_neurons: int, learning_rate: float, activation_function: Callable[[float], float], derivative_activation_function: Callable[[float], float]) -> None:
        for i in range(num_neurons):
            weights = [random() for _ in range(len(self.previous_layer.neurons) + 1)]
            self.neurons.append(Neuron(weights, learning_rate, activation_function, derivative_activation_function))
```

delta is stored in the neuron.

O1w1 is the weight coming into O1 that corresponds to the signal coming from N1.

temp = O1Delta * O1Deriv * O1w1

O1, O2, and O3 had their deltas calculated in Figure 7.4.

The delta of other hidden neurons are calculated the same way with their respective next layer weights and output caches.

Listing 7.4 network.py

```
from typing import List, Callable, Tuple
from random import random
from layer import Layer
from util import sigmoid, derivative_sigmoid

t = TypeVar("T") # output type of interpretation of neural network

class Network:
    def __init__(self, layer_structure: List[int], learning_rate: float, activation_function: Callable[[float], float], derivative_activation_function: Callable[[float], float]) -> None:
        if len(layer_structure) < 3:
            raise ValueError("Error: Should be at least 3 layers (1 input, 1 hidden, 1 output)")
        self.layers: List[Layer] = []
        # input layer
        input_layer: Layer = Layer(None, layer_structure[0], learning_rate, activation_function, derivative_activation_function)
        self.layers.append(input_layer)
        # hidden layers and output layer
        for previous, num_neurons in enumerate(layer_structure[1:]):
            next_layer: Layer = Layer(self.layers[previous], num_neurons, learning_rate, activation_function, derivative_activation_function)
            self.layers.append(next_layer)
```


Osadzanie słów

Word embeddings

Predicting POS/NEG

```
print(similar('terrible'))
```

```
[('terrible', -0.0),  
 ('dull', -0.760788602671491),  
 ('lacks', -0.76706470275372),  
 ('boring', -0.7682894961694),  
 ('disappointing', -0.768657),  
 ('annoying', -0.78786389931),  
 ('poor', -0.825784172378292),  
 ('horrible', -0.83154121717),  
 ('laughable', -0.8340279599),  
 ('badly', -0.84165373783678)]
```

Fill In The Blank

```
print(similar('terrible'))
```

```
[('terrible', -0.0),  
 ('horrible', -2.79600898781),  
 ('brilliant', -3.3336178881),  
 ('pathetic', -3.49393193646),  
 ('phenomenal', -3.773268963),  
 ('masterful', -3.8376122586),  
 ('superb', -3.9043150978490),  
 ('bad', -3.914673639585237),  
 ('marvelous', -4.0470804427),  
 ('dire', -4.178749691835959)]
```

```
print(similar('beautiful'))
```

```
[('beautiful', -0.0),  
 ('atmosphere', -0.70542101298),  
 ('heart', -0.7339429768542356),  
 ('tight', -0.7470388145765346),  
 ('fascinating', -0.7549291974),  
 ('expecting', -0.759886970744),  
 ('beautifully', -0.7603669338),  
 ('awesome', -0.76647368382398),  
 ('masterpiece', -0.7708280059),  
 ('outstanding', -0.7740642167)]
```

```
print(similar('beautiful'))
```

```
[('beautiful', -0.0),  
 ('lovely', -3.0145597243116),  
 ('creepy', -3.1975363066322),  
 ('fantastic', -3.2551041418),  
 ('glamorous', -3.3050812101),  
 ('spooky', -3.4881261617587),  
 ('cute', -3.592955888181448),  
 ('nightmarish', -3.60663813),  
 ('heartwarming', -3.6348147),  
 ('phenomenal', -3.645669007)]
```

king - man + woman ==



```
king = [0.6, 0.1]  
man = [0.5, 0.0]  
woman = [0.0, 0.8]  
queen = [0.1, 1.0]  
king - man = (0.1, 0.1)  
queen - woman = (0.1, 0.2)
```



Rysunek: Osadzanie słów

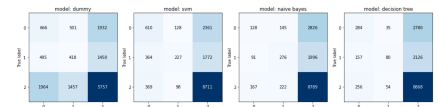
Współczesne modele NLP

	Word order awareness	Context awareness (cross-words interactions)
Bag-of-unigrams	No	No
Bag-of-bigrams	Very limited	No
RNN	Yes	No
Self-attention	No	Yes
Transformer	Yes	Yes

Rysunek: Modele NLP

Klasyfikacja wieloklasowa

```
model=dummy, f1_micro: mean=0.47, std=0.01  
model=svm, f1_micro: mean=0.65, std=0.0  
model=naive bayes, f1_micro: mean=0.63, std=0.0  
model=decision tree, f1_micro: mean=0.63, std=0.0  
model=random forest, f1_micro: mean=0.63, std=0.0
```



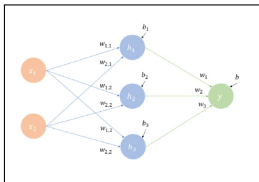
Lab I.50 klasyfikacja wieloklasowa

Wykonaj polecenia zawarte w dołączonym notatniku

Lab.I.50.raw.ipynb.

Uwaga: Przejrzyj również plik `helper.py` w celu poprawienia wyników zawartych w przykładach z powyższego notatnika.

Prosta sieć neuronowa



Lab I.51 Prosta sieć neuronowa

Po prostu wykonaj polecenia w notatniku Lab.I.51.raw.ipynb i pobierz utworzony w pakiecie Keras model sieci neuronowej.

Wykorzystaj bibliotekę `ibm_watson_studio_lib`.

Zamiast Lab I.32

Lab H.I.52 Proszę poprawnie wykonać wszystkie w kroki w dołączonym notatniku Lab.H.I.52.raw.ipynb wykorzystując inny niż w kodzie model LLM dostępny w IBM Cloud.

The End