



**AGH**

**AKADEMIA GÓRNICZO-HUTNICZA IM. STANISŁAWA STASZICA W  
KRAKOWIE**

**WYDZIAŁ FIZYKI I INFORMATYKI**

WYDZIAŁ INFORMATYKI STOSOWANEJ

Praca dyplomowa inżynierska

*Wykorzystanie algorytmu sztucznej inteligencji do  
rozwiązania problemu optymalnego wyboru ścieżki lub  
wyjścia ewakuacyjnego*

Autor:

*Amadeusz Hercog*

Kierunek studiów:

*Informatyka stosowana*

Opiekun pracy:

*mgr. inż. Robert Lubaś*

Kraków, 21 listopada 2018

Uprzedzony o odpowiedzialności karnej na podstawie art. 115 ust. 1 i 2 ustawy z dnia 4 lutego 1994 r. o prawie autorskim i prawach pokrewnych (t.j. Dz.U. z 2006 r. Nr 90, poz. 631 z późn. zm.): „Kto przywłaszcza sobie autorstwo albo wprowadza w błąd co do autorstwa całości lub części cudzego utworu albo artystycznego wykonania, podlega grzywnie, karze ograniczenia wolności albo pozbawienia wolności do lat 3. Tej samej karze podlega, kto rozpowszechnia bez podania nazwiska lub pseudonimu twórcy cudzy utwór w wersji oryginalnej albo w postaci opracowania, artystycznego wykonania albo publicznie zniekształca taki utwór, artystyczne wykonanie, fonogram, wideogram lub nadanie.”, a także uprzedzony o odpowiedzialności dyscyplinarnej na podstawie art. 211 ust. 1 ustawy z dnia 27 lipca 2005 r. Prawo o szkolnictwie wyższym (t.j. Dz. U. z 2012 r. poz. 572, z późn. zm.): „Za naruszenie przepisów obowiązujących w uczelni oraz za czyny uchybiające godności studenta student ponosi odpowiedzialność dyscyplinarną przed komisją dyscyplinarną albo przed sądem koleżeńskim samorządu studenckiego, zwanym dalej «sądem koleżeńskim».”, oświadczam, że niniejszą pracę dyplomową wykonałem(-am) osobiście i samodzielnie i że nie korzystałem(-am) ze źródeł innych niż wymienione w pracy.

*Serdecznie dziękuję ...tu ciąg  
dalszych podziękowań np. dla  
promotora, żony, sąsiada itp.*



# 1. Wstęp

*W tym rozdziale zostanie opisane podłoże historyczne, cel i jego założenia projektu. Zaprezentowane zostanie proponowane rozwiązanie postawionego problemu oraz technologie do tego użyte.*

## 1.1. Cel projektu

W wielu miastach, gdzie istnieją duże skupiska ludzi prędzej czy później pojawia się problem alokacji przestrzeni na zabudowę. W tym celu nowoczesne budynki użytku publicznego są projektowane pod kątem wykorzystania jak największej przestrzeni dla celów użytkowych, ale i z drugiej strony - dla zapewnienia jak największego bezpieczeństwa ludziom w nich przebywającym. Jednym z czynników określających wchodzącym w skład szeroko pojętego bezpieczeństwa, jest zapewnienie ludziom ścieżek ewakuacyjnych w razie pożaru lub innych wypadków. Z tego powodu celu postawiłem następujący problem - rozwiązanie problemu optymalizacji wyboru ścieżki lub wyjścia ewakuacyjnego ludzi w budynku. Co więcej, w związku z moim wcześniejszym doświadczeniem z tematem uczenia maszynowego, postanowiłem rozwiązać ten problem za pomocą technik machine learning.

## 1.2. Użyte technologie oraz programy

Wszystkie obliczenia oraz rysowanie wykresów zostało zrealizowane w języku programowania **Python** w wersji 3.6 przy użyciu bibliotek:

- **NumPy** - służąca do obliczeń numerycznych,
- **Pandas** - służąca łatwiejszemu zarządzaniu dużą ilością danych,
- **Matplotlib** - służąca do rysowania wykresów,
- **Tensorflow** - używana do głębokiego uczenia maszynowego.

Aby mieć możliwość symulacji ruchu pieszych w budynku, użyłem symulatora **PSP** (*Pederastian Simulation Project*) autorstwa Roberta Lubaś oraz Wojciecha Myśliwiec - opisany szerzej w rozdziale [2](#).

## 1.3. Proponowane rozwiązanie

Rozwiązanie proponowane przeze mnie zostało podzielone na kilka kroków:

1. Wygenerowanie  $n$  zestawów parametrów wejściowych do symulatora *PSP* w celu pozyskania jego danych wyjściowych.
2. Stworzenie modelu uczenia maszynowego na podstawie wygenerowanych zestawów danych wejściowych oraz wyjściowych symulatora i nauczanie go na danych z punktu [1](#). Daje to możliwość generowania nowych danych wyjściowych bez używania symulatora.
3. Optymalizacja zestawu danych wyjściowych modelu na podstawie jego danych wejściowych używając metod optymalizacji.

## 2. PSP - Pederastian Simulator Project

*W tym rozdziale opisany zostanie symulator ruchu pieszych użyty w projekcie i jego sposób użycia wraz z niezbędnymi dodatkowymi plikami. Opisane zostaną użyte w projekcie parametry wejściowe do symulatora oraz metryki stworzone na podstawie jego danych wyjściowych.*

### 2.1. Opis

Symulator *Pederastian Simulator Project* jest programem napisanym w języku C++ autorstwa Roberta Lubaś oraz Wojciecha Myśliwiec. Służy on do modelowania ruchu agentów (ludzi) podczas ewakuacji w danej przestrzeni. Symulowana przestrzeń jest dwuwymiarowa, z możliwością dodania wielu poziomów odpowiadających kolejnym piętrům budynku. Główne okno programu jest widoczne na rysunku [2.1](#).

### 2.2. Sposób użycia

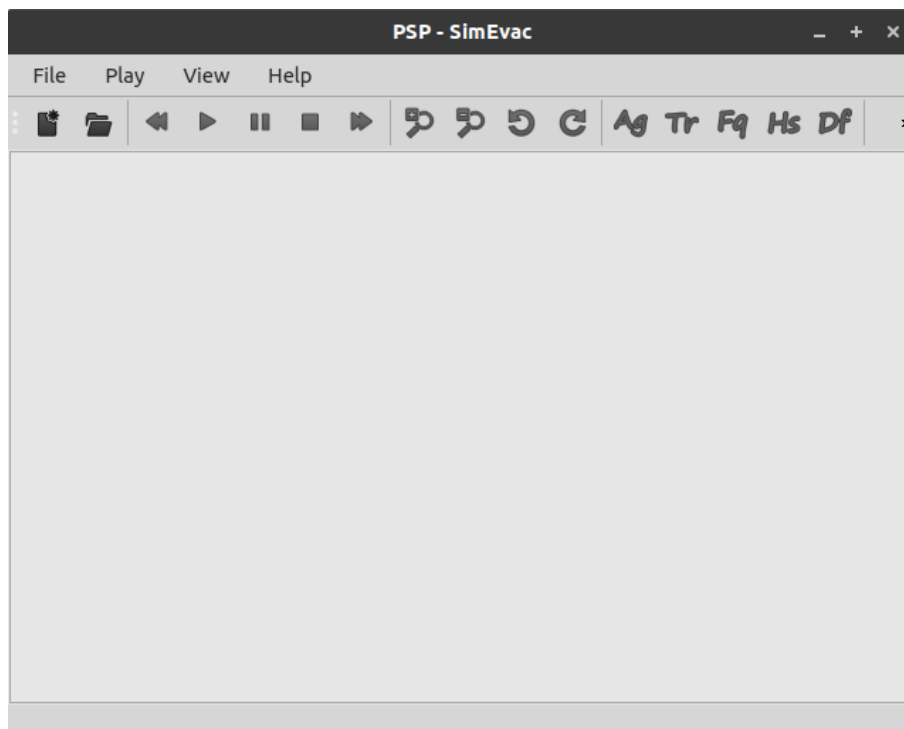
Do przeprowadzenia symulacji potrzebne są następujące rzeczy:

- plik *.xml* zawierający parametry wejściowe symulacji,
- pliki *.jpg* i *.bmp* zawierające przestrzeń użytą do symulacji.

W oknie głównym programu należy wybrać opcję otwarcia pliku *.xml* po czym od razu zaczyna się symulacja w czasie rzeczywistym. Po symulacji, we wskazanym w pliku z parametrami folderze zostaną wygenerowane pliki *.csv* z danymi wyjściowymi symulacji.

### 2.3. Dane wejściowe i wyjściowe

Dane, na których miałem możliwość przeprowadzenia badań znajdowały się w:



**Rys. 2.1.** Główne okno symulatora.

- dane wejściowe - plik *.xml*,
- dane wyjściowe - pliki *.csv*.

Danymi wejściowymi, które wybrałem - na podstawie łatwości modyfikacji - do dalszej analizy były:

- **Panic spread factor** - współczynnik, który decyduje jak duży wpływ ma **tryb paniki** (tryb, w którym agenci zwracają mniejszą uwagę na otoczenie oraz charakteryzują się bardziej chaotycznym ruchem) na agentów.
- **Panic cancel zone** - współczynnik określający odległość od wyjść, w obrębie jakiej agenci mają szansę na deaktywację trybu paniki.
- **Cancel panic chance**- podczas testów na anulowanie trybu paniki określa procent szans na powodzenie.
- **Choosing evacuation path mode** - tryb wyboru drogi ewakuacyjnej przez agentów. Dostępne są 4 tryby:
  - odległości,
  - gęstości przy wyjściu,



- odległości oraz gęstości przy wyjściu,
- odległości, gęstości przy wyjściu oraz popularności wyboru wyjścia.
- **Number of pederastians** - liczba agentów biorąca udział w symulacji.
- **Chaos level** - szansa na aktywację trybu paniki u agentów.
- **Density factor** - określa wpływ współczynnika gęstości wokół agenta na funkcję kary.
- **Frequency factor** - określa wpływ częstości wyboru danego pola na funkcję kary.
- **Panic factor** - potęguje część współczynników biorących udział w obliczaniu funkcji kary.
- **Distance factor** - wpływa na wybór agentów ruchu po skosie lub na wprost.
- **Randomness factor** - czynnik losowości dla wartości kary.
- **Pre-movement time mean value** oraz **Pre-movement time standard deviation** - średnia i odchylenie czasu *Pre-movement*, który ma wpływ na częstotliwość wykonywania testu na zmianę obranego wyjścia przez agentów.
- **Speed distribution mean value** oraz **Speed distribution standard deviation** - średnia i odchylenie prędkości agentów.

Dane wyjściowe wybrane do analizy przeze mnie były następujące:

- czas ewakuacji ostatniego agenta (odpowiadający czasowi ewakuacji wszystkich agentów),
- błąd średniokwadratowy średnich prędkości wszystkich agentów w stosunku do średniej prędkości najszybszego agenta.

Dodatkowe parametry wejściowe, które były mi potrzebne (lecz nie wpływały na sam proces badawczy) to:

- **Repeat number** - ilość powtórzeń symulacji wykonanej na pojedynczym pliku *.xml*.
- **Sim directory** - określa ścieżkę do folderu zawierającego pliki symulowanej przestrzeni.
- **Sim stat directory** - określa ścieżkę do folderu, do którego zostaną zapisane dane wyjściowe symulacji (pliki *.csv*).



## 3. Budowa modelu uczenia maszynowego

*W tym rozdziale opisany będzie proces zbierania danych za pomocą symulatora PSP, tworzenie modelu uczenia maszynowego oraz optymalizacji jego wartości wyjściowych.*

### 3.1. Zebranie danych

W celu zebrania danych, które miałyby trafić później do modelu uczenia maszynowego, stworzyłem skrypt pythonowy, który wygenerował 200 różnych zestawów parametrów wejściowych do symulatora. Rodzaje danych, które wygenerowałem znajdują się w rozdziale 2.3. Dla każdego zestawu danych ustawiłem parametrowi *Repeat number* wartość **10**, dzięki czemu dla każdego zestawu danych wejściowych generowane było 10 zestawów danych wyjściowych. Było to niezwykle cenne z 2 powodów:

- Do symulatora trzeba było ręcznie ładować plik *.xml*, a dzięki temu z jednego pliku otrzymywałem od razu 10 zestawów plików wyjściowych.
- Ruchy agentów, nawet z takim samym zestawem parametrów, różnią się między symulacjami, a dzięki powtórzeniom łatwiej było wyeliminować te różnice.

Dla części symulacji, symulator niespodziewanie wyłączył się podczas pracy, przez co niektórym zestawom danych wejściowych odpowiada mniej niż 10 zestawów danych wyjściowych. Ostatecznie z 200 zestawów parametrów wejściowych uzyskałem 1888 zestawów parametrów wyjściowych. Oznacza to, że 5,6% z symulacji, które miały zostać przeprowadzone nie udało się.

W pierwszym podejściu, aby móc przetestować model w prostych warunkach, wybrałem z każdego zestawu danych: wszystkie dane wejściowe, jedną daną wyjściową. Daną wyjściową, którą wybrałem był czas ewakuacji ostatniego agenta. Każdy zestaw danych zawierał więc **15** parametrów wejściowych oraz **1** parametr

wyjściowy. Z tak przygotowanymi danymi miałem możliwość wybrania pierwszego modelu.

## 3.2. Opis modelu Perceptron

Na początek, wybrałem prosty model uczenia maszynowego - regresję liniową. Stworzyłem go wykorzystując **Perceptron**. Jest to prosta sieć neuronowa, która posiada zestaw neuronów wejściowych, każdy odpowiadający pojedynczej danej wejściowej. Składa się on także z jednego lub wielu niezależnych neuronów wyjściowych. Każdy z nich jest połączony ze wszystkimi neuronami wejściowymi za pomocą krawędzi posiadających wagi - definiują one jak duży wpływ na wyjście ma każda dana wejściowa. Jeśli wyrazimy:

- wejście jako wektor  $X$ ,
- wyjście pojedynczego Perceptronu jako  $a$ ,
- neuron jako wektor wag  $N$ ,
- funkcję aktywacji jako  $\sigma$  - jest to funkcja wprowadzająca nieliniowość danych, najpopularniejsze z nich to *Sigmoid*, *ReLU (Rectified linear unit)* czy też *Softmax*

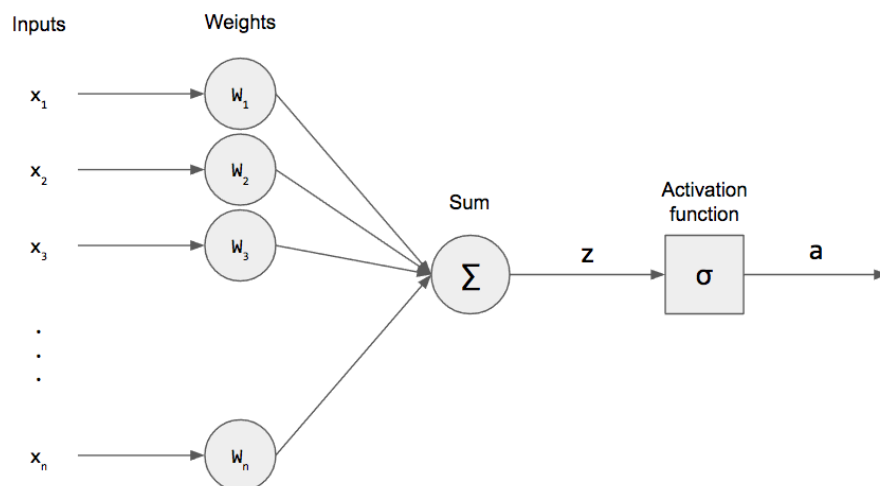
to zachodzi równanie:

$$Y = \sigma(NX) \quad (3.1)$$

Budowa Perceptronu z jednym neuronem wyjściowym jest widoczna na rysunku 3.1<sup>1</sup>.

Podczas procesu uczenia *Perceptronu* ważne jest aby określić **loss function** - metrykę którą chcemy optymalizować. Dla problemów regresji najczęściej używaną funkcją kosztu jest **MSE** (Mean Squared Error). Kolejnym parametrem sieci *Perceptron* jest parametr **learning rate**, który określa jak bardzo będzie się zmieniać macierz/wektor wag na podstawie gradientu podczas każdej iteracji uczącej. Zbyt duży współczynnik *learning rate* może spowodować, że zamiast optymalizować funkcję kosztu będzie wręcz przeciwnie - będzie ona osiągała coraz większe wartości. Gdy jest on zbyt mały, jest on mniej niebezpieczny gdyż nie doprowadzi on do wzrostu wartości funkcji kosztu; problemem jest jednak wolniejszy proces uczący sieci co wymaga więcej epok uczących. Nie ma jednak uniwersalnej metody regulującej sposób dobierania *learning rate* - trzeba to robić eksperymentalnie.

<sup>1</sup> <https://pythonmachinelearning.pro/perceptrons-the-first-neural-networks/>



**Rys. 3.1.** Budowa Perceptronu z jednym neuronem wyjściowym.

### 3.3. Pierwszy model - tworzenie i analiza

Dla powyższych zastosowań stworzyłem więc *Perceptron* posiadający pojedyncze wyjście oraz nie posiadający funkcji aktywacji - jako że model miał służyć do regresji a nie klasyfikacji. Zestawy danych podzieliłem w stosunku 80% : 20% na **dane treningowe** i **dane testowe**, pozwala to na trenowanie sieci na danych treningowych oraz sprawdzanie jej efektywności na danych testowych co daje możliwość walidacji tego, jak dobrze sieć radzi sobie z danymi, z którymi nie miała styczności.

Podsumowując, parametry pierwszego modelu były następujące:

- pojedyncza warstwa składająca się z jednego neuronu,
- **funkcja aktywacji** - brak,
- **learning rate** - 0.1,
- **metryka do optymalizacji** - *MSE*,
- **liczba epok uczących** - 300,
- podział na **dane treningowe** i **dane testowe** - 80% : 20%.

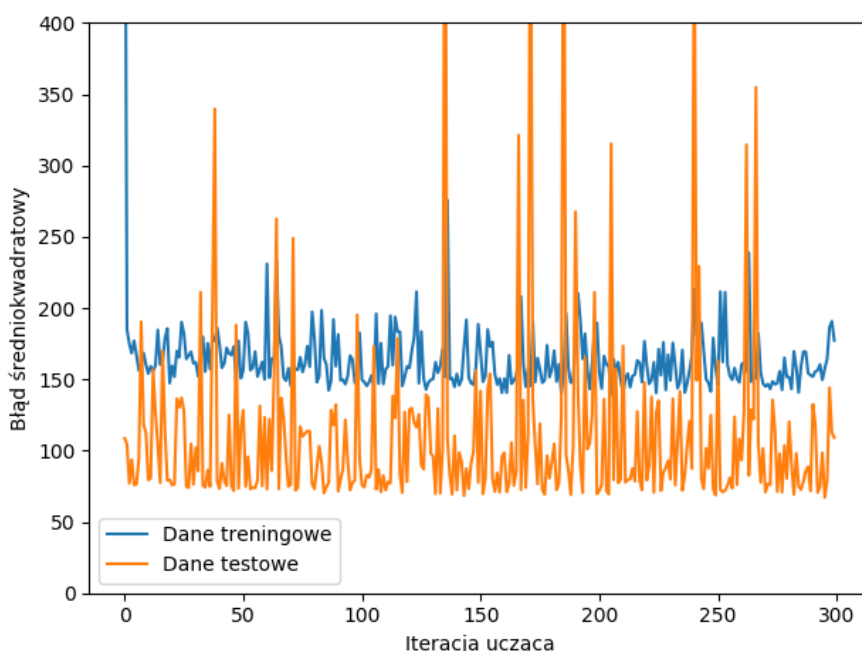
Proces uczenia został wykonany 10 razy od nowa na nienauczonym modelu, za każdym razem wykonując permutację danych. Średnie *MSE* danych wyglądają następująco:

- **średnie MSE danych treningowych** - 147,77 ( $\pm 16,04$ ),

- **średnie MSE danych testowych** - 166,18 ( $\pm 50,77$ ).

Wykres zależności *MSE* danych od iteracji uczącej najlepszego przypadku (biorąc pod uwagę *MSE* danych testowych) jest widoczna na rysunku 3.2. Taki rodzaj wykresów będzie nazywany dalej **wykresami krzywych uczących dla najlepszego przypadku**. Analizując go, doszedłem do następujących wniosków:

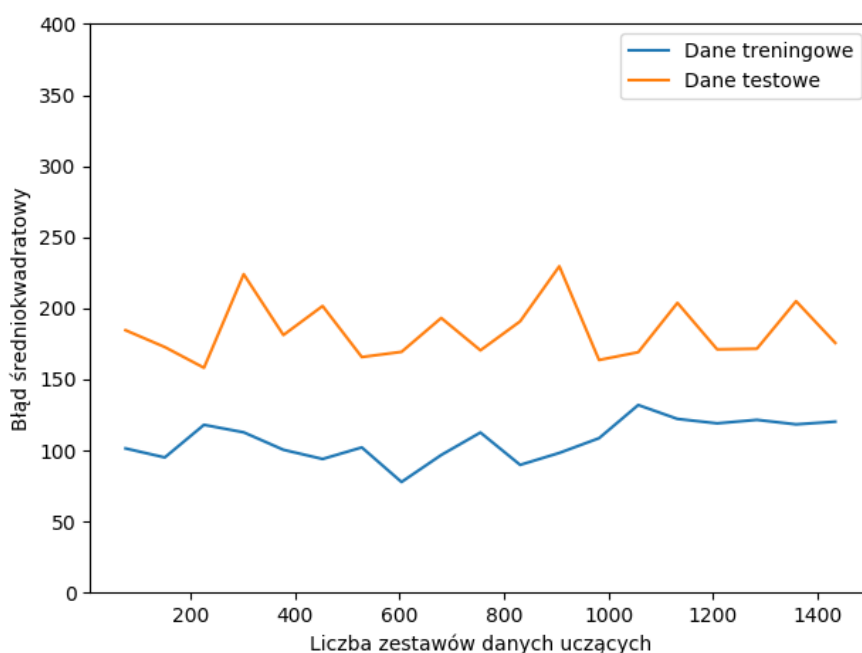
- *MSE* danych treningowych jak i testowych jest na wysokim poziomie co oznacza, że model ma **wysoki bias**. Oznacza to, że użyty model jest zbyt prosty i nie ważne jak dużo danych zostanie do niego podanych i jak długo będzie się on uczył - *MSE* danych nie obniży się poniżej pewnego poziomu.
- Podczas uczenia *MSE* danych jest obarczony dużym szumem, co oznacza niestabilność danych lub modelu.
- *MSE* danych testowych jest przez większość iteracji mniejsze niż treningowych co oznacza, że dla danych których sieć nie widziała, przewiduje ona wartości bliższe prawdzie niż na danych na których się uczyła. Jest to niecodzienne zachowanie co również dowodzi niestabilności modelu lub danych.



**Rys. 3.2.** Krzywe uczące dla najlepszego przypadku w pierwszym modelu.

Aby mieć szerszy pogląd na to, jaki wpływ ma ilość danych na których się uczy sieć na *MSE* tych danych - postanowiłem uruchomić proces uczenia kolejne 20 razy

dla pojedynczej permutacji danych wejściowych, lecz przy zwiększającej się ilości danych treningowych. Wykres obrazujący wynik tych badań jest widoczny na rysunku 3.3. Ten rodzaj wykresów dalej nazywany będzie **wykresami krzywych uczących dla zmiennej ilości danych**. Informacją jaką można z niego wyciągnąć jest ilość danych potrzebna do tego aby osiągnąć dany pułap  $MSE$  danych. Widać że podczas gdy przy danych treningowych, większa ich ilość spowodowała spadek ich  $MSE$  to przy danych testowych nie spowodowało to większej różnicy.



**Rys. 3.3.** Krzywe uczące dla zmiennej ilości danych w pierwszym modelu.

Podsumowując, obecny model ma wysoki bias - jest zbyt prosty dla danych, przez co większa ich ilość oraz dłuższy proces uczący nie dają żadnych efektów. Na wykresach widać niestabilność modelu lub danych, co powoduje bardzo duże szумы na wykresach  $MSE$ . Co więcej - niepokojący jest fakt, że  $MSE$  danych testowych był często niższy niż  $MSE$  danych treningowych.

### 3.4. Normalizacja danych

Mając na uwadze problemy stworzonego modelu, w pierwszej kolejności postanowiłem przejrzeć dane wejściowe do modelu. Ze wszystkich zestawów danych obliczyłem więc średnią każdego z parametrów wejściowych do sieci neuronowej.

**Tabela 3.1.** Średnie wartości parametrów wejściowych do modelu.

Nazwa parametru	Średnia wartość
Panic spread factor	1.74
Panic cancel zone	0.53
Cancel panic chance	51.59
Choosing evacuation path mode	2.53
Number of pederastian	271.8
Chaos level	53.9
Density factor	5.26
Frequency factor	5.33
Panic factor	1.45
Distance factor	2.51
Randomness factor	0.49
Pre-movement time mean value	5.44
Pre-movement time standard deviation	0.51
Speed distribution mean value	5.62
Speed distribution standard deviation	0.54

Wyniki są widoczne w tabeli 3.1. Widać w niej, że niektóre parametry mają wartości 2 rzędy wielkości większe niż inne. Takie różnice pomiędzy różnymi parametrami wejściowymi powodują faworyzowanie przez sieć jednych parametrów ponad drugie. Postanowiłem więc zastosować normalizację danych według wzoru:

$$X_{new} = \frac{X_{old} - \mu}{\sigma} \quad (3.2)$$

Gdzie:

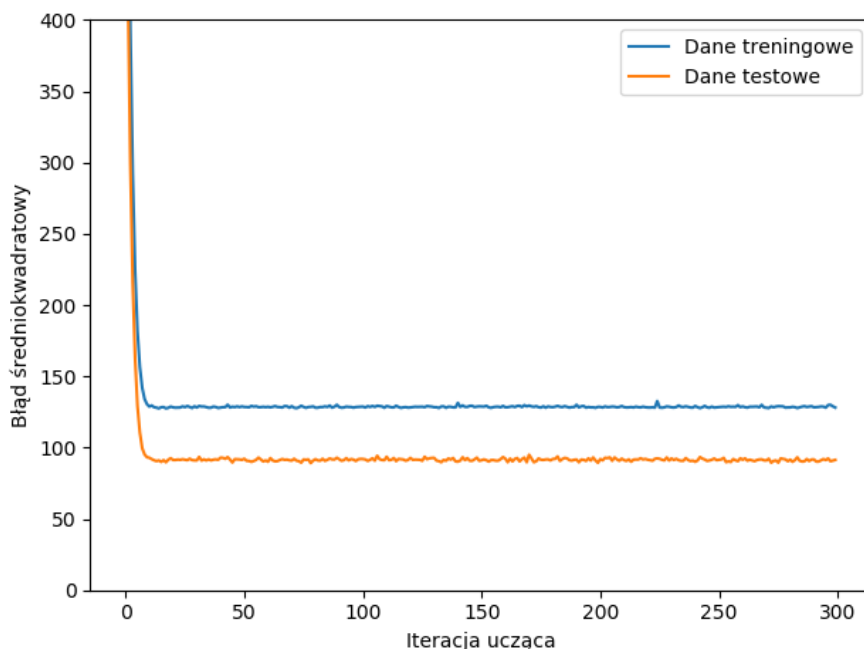
- $\mu$  - wektor średniej parametrów,
- $\sigma$  - wektor odchylenia parametrów.

Posiadając tak przygotowane dane wykonałem - tak jak w punkcie 3.3 - 10 cykli nauczania z różnymi permutacjami danych oraz 20 cykli z jedną permutacją, lecz zmieniającym się rozmiarem zestawu danych treningowych. Średnie *MSE* danych są następujące:

- **średnia MSE danych treningowych** - 118,64 ( $\pm 6,07$ ),
- **średnia MSE danych testowych** - 130,70 ( $\pm 23,47$ ).



Wykres krzywych uczących dla najlepszego przypadku widoczny jest na rysunku 3.4. Dodatkowo patrząc na wykres krzywych uczących dla zmiennej ilości danych z rysunku 3.5 - widać zależność  $MSE$  danych od liczby danych treningowych.

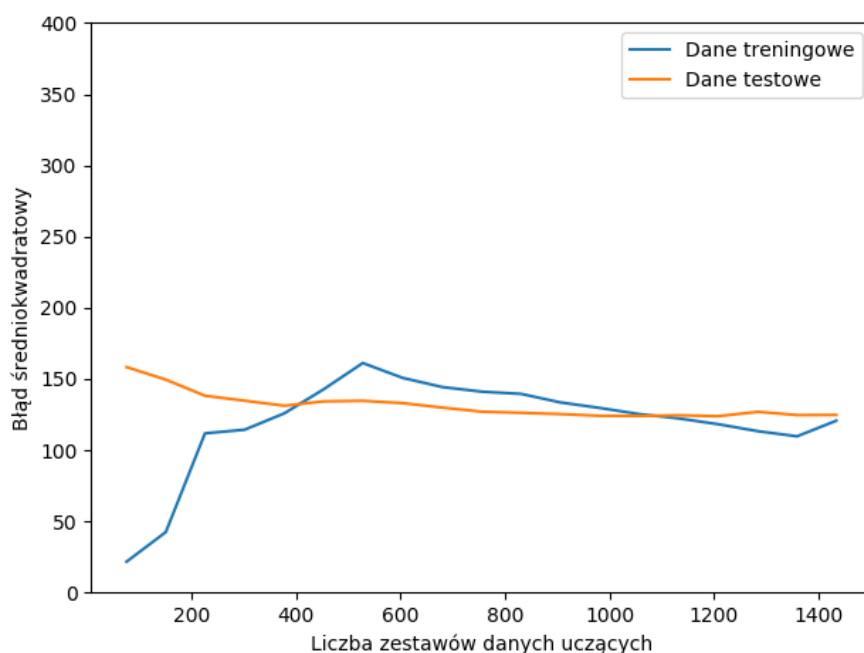


**Rys. 3.4.** Krzywe uczące dla najlepszego przypadku w pierwszym modelu ze znormalizowanymi danymi.

Podsumowując, pierwszą rzeczą rzucającą się w oczy jest duża stabilność wartości na wykresach - eliminuje to sytuacje w których wynik uczenia był winą przypadku, jako że kilka iteracji wcześniej lub później *Perceptron* dawał zupełnie inny rezultat. Dzięki większej stabilności sieci łatwiej jest także wyciągać wnioski na podstawie wykresów. Lepiej widoczny jest problem zauważony wcześniej - wysoki bias modelu. Objawia się on tym, że na początku  $MSE$  danych maleje, aż do pewnego momentu gdzie jest stały i żadna ilość danych albo długość uczenia nie spowoduje jego zmniejszenia. Ostatnią rzeczą, która jest taka sama jak w rozdziale 3.3 -  $MSE$  danych testowych często jest mniejsze niż danych treningowych.

### 3.5. Drugi model - tworzenie i analiza

Chcąc rozwiązać główny problem poprzedniego modelu - wysoki bias - potrzebowałem modelu, który potrafi nauczyć się bardziej skomplikowanych funkcji. Takim modelem była sieć neuronowa z kilkoma warstwami. Dzięki każdej następnej



**Rys. 3.5.** Krzywe uczące dla zmiennej ilości danych w pierwszym modelu ze znormalizowanymi danymi.

warstwie ma ona możliwość uczenia się coraz bardziej skomplikowanych funkcji. Sieć jaką stworzyłem miała 1 **warstwę ukrytą** - warstwę neuronów pomiędzy wejściami a warstwą wyjść - z 30 neuronami. Jako funkcję aktywacji wybrałem funkcję *ReLU*. Podsumowując, parametry drugiego modelu były następujące:

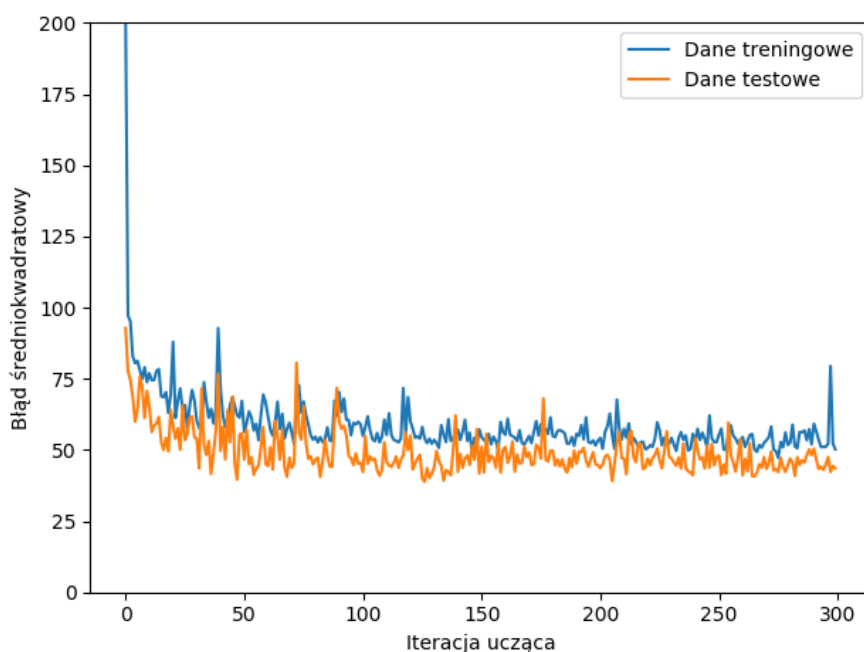
- 1 warstwa ukryta z 30 neuronami i funkcją aktywacji ReLU,
- warstwa wyjściowa składająca się z jednego neuronu bez funkcji aktywacji,
- **learning rate** - 0.1,
- **metryka do optymalizacji** - MSE,
- **liczba epok uczących** - 300,
- podział na **dane treningowe** i **dane testowe** - 80% : 20%.

Proces uczący przeprowadziłem identycznie jak poprzednio - wykonałem 10 osobnych cykli uczących z różnymi permutacjami danych oraz 20 osobnych cykli uczących na zmieniającej się wielkości zestawu danych treningowych. Średnie *MSE* danych z 10 cykli są następujące:

- **średnia MSE danych treningowych** - 47,47 ( $\pm 6,08$ ),

- **średnia MSE danych testowych** - 73,66 ( $\pm 25,77$ ).

Wykres krzywych uczących dla najlepszego przypadku widoczny jest na rysunku 3.6 a krzywych uczących dla zmiennej ilości danych - na rysunku 3.7.



**Rys. 3.6.** Krzywe uczące dla najlepszego przypadku w drugim modelu.

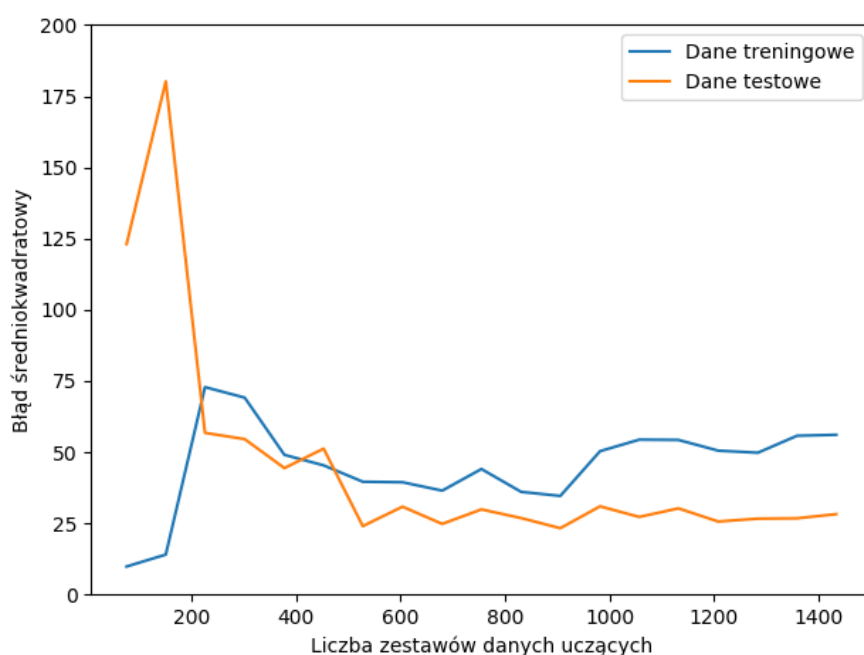
Porównując te dane z tymi z sekcji 3.4 widać znaczącą poprawę modelu - trzykrotny spadek średnich *MSE* danych treningowych oraz testowych. Mimo tego widać, *MSE* po osiągnięciu pewnego pułapu przestaje maleć - oznacza to, że model dalej ma duży bias.

### 3.6. Zwiększenie stopnia skomplikowania modelu

Próbując zmniejszyć bias sieci neuronowej, postanowiłem przeprowadzić eksperymenty z innymi parametrami sieci, a mianowicie:

- liczba warstw ukrytych - **1**, liczba neuronów w warstwie - **40**;
- liczba warstw ukrytych - **2**, liczba neuronów w każdej warstwie - **20**;
- liczba warstw ukrytych - **2**, liczba neuronów w każdej warstwie - **40**;

Zestawienie wyników z sekcji 3.5 oraz powyższych kombinacji parametrów jest następujące:



**Rys. 3.7.** Krzywe uczące dla zmiennej ilości danych w drugim modelu.

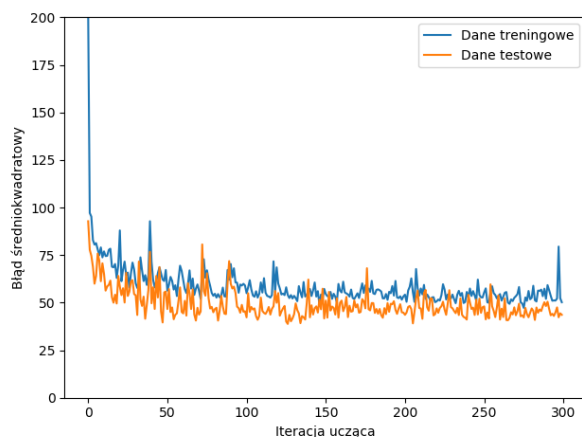
**Tabela 3.2.** Średnie wartości *MSE* danych dla różnych kombinacji parametrów sieci.

Ilość warstw	Ilość neuronów w warstwie	MSE danych treningowych	MSE danych testowych
1	20	47,47 ( $\pm 6,08$ )	73,66 ( $\pm 25,77$ )
1	40	53,16 ( $\pm 4,93$ )	47,87 ( $\pm 20,13$ )
2	20	59,49 ( $\pm 10,81$ )	89,52 ( $\pm 22,04$ )
2	40	64,13 ( $\pm 7,30$ )	70,68 ( $\pm 20,42$ )

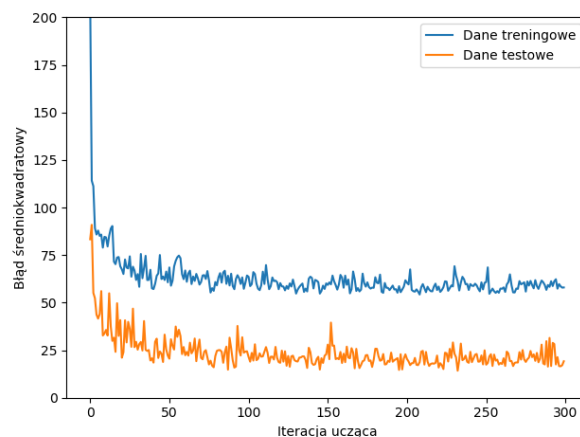
- krzywe uczące dla najlepszych przypadków są widoczne na rysunku 3.8,
- krzywe uczące dla zmiennej ilości danych pokazane są na rysunku 3.9.

Dodatkowo średnie *MSE* z 10 różnych permutacji danych zestawilem w tabeli 3.2.

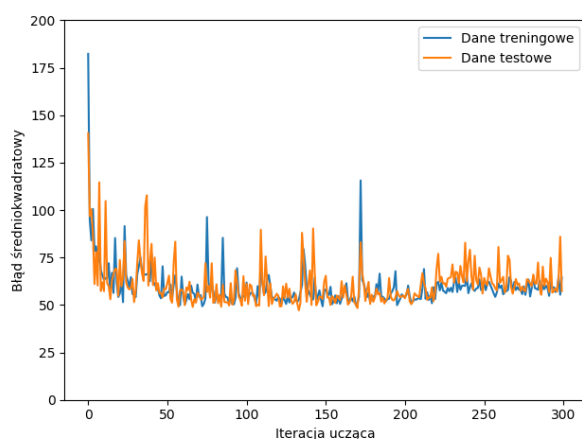
Mimo zwiększenia stopnia skomplikowania sieci, *MSE* danych treningowych i testowych nie wykazywały tendencji spadkowej. To co powinno się stać w takim przypadku, to *MSE* danych treningowych powinien osiągać coraz niższe wartości wraz ze skomplikowaniem modelu. Na wykresach również widac, że dane testowe miały czasem *MSE* mniejszych od zestawu treningowego. Wnioskiem tej analizy jest to, że problem z uczeniem sieci leży nie w modelu, a w danych.



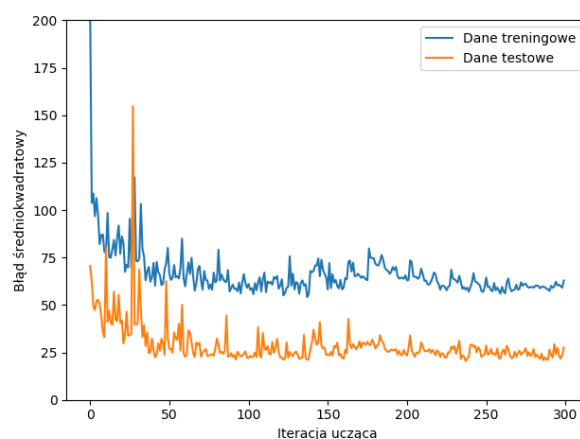
(a) 1 warstwa ukryta, 20 neuronów w każdej warstwie



(b) 1 warstwa ukryta, 40 neuronów w każdej warstwie



(c) 2 warstwy ukryte, 20 neuronów w każdej warstwie

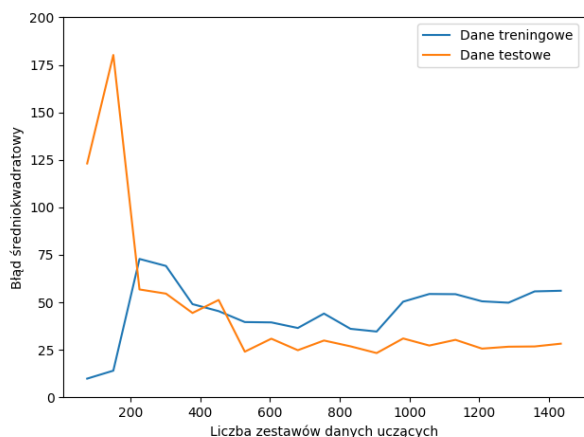


(d) 2 warstwy ukryte, 40 neuronów w każdej warstwie

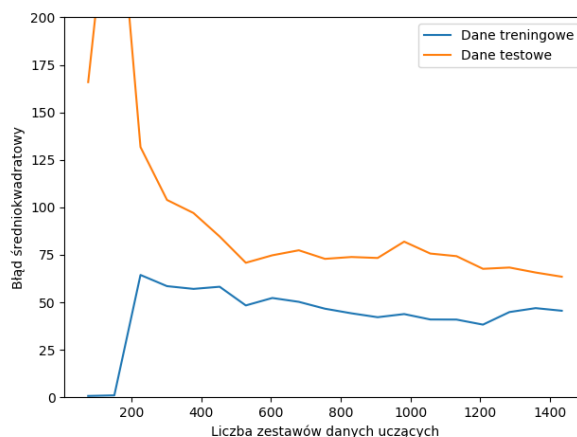
**Rys. 3.8.** Krzywe uczące dla najlepszych przypadków przy użyciu różnych kombinacji parametrów w drugim modelu.

## 3.7. Uśrednianie danych

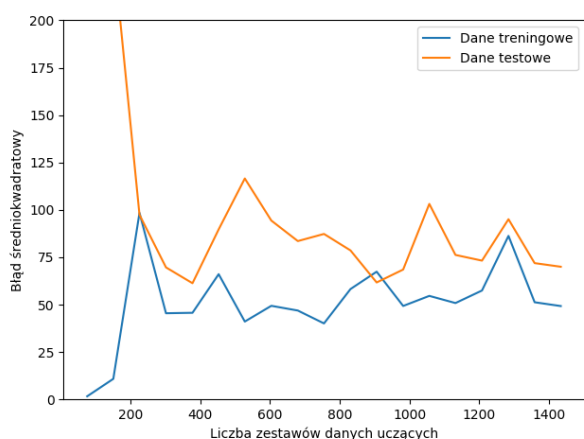
Wiedząc że problem leży w danych postanowiłem ulepszyć istniejące dane, uśredniając je. W sekcji 3.1 napisałem, że dla każdego zbioru parametrów wejściowych wygenerowałem do 10 zbiorów parametrów wyjściowych. Dla każdego zestawu wejściowego obliczyłem średnią zestawów parametrów wyjściowych co zmniejszyło ilość danych prawie 10-krotnie. Ostatecznie mój nowy zbiór zawierał 200 zestawów danych (przy czym, podczas procesu uczenia zostały one podzielone na dane treningowe i testowe).



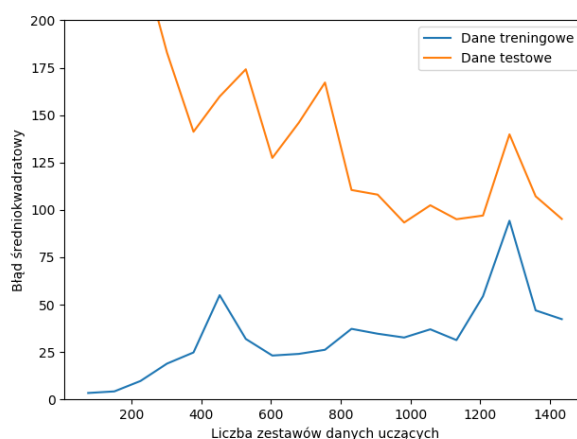
(a) 1 warstwa ukryta, 20 neuronów w każdej warstwie



(b) 1 warstwa ukryta, 40 neuronów w każdej warstwie



(c) 2 warstwy ukryte, 20 neuronów w każdej warstwie



(d) 2 warstwy ukryte, 40 neuronów w każdej warstwie

**Rys. 3.9.** Krzywe uczące dla zmiennej ilości danych przy różnych kombinacjach parametrów w drugim modelu.

Po przetworzeniu danych postanowiłem powtórzyć badanie z sekcji 3.6, używając kombinacji parametrów:

- ilość warstw ukrytych - **1 i 2**,
- ilość neuronów w każdej warstwie - **20 i 40**.

Po powtórzeniu poprzedniego badania na uśrednionych danych, wyniki widać na rysunkach 3.15 i 3.11. Średnie  $MSE$  danych dla kombinacji parametrów jest umieszczone w tabeli 3.3.

Analizując dane, można zauważyć że po uśrednieniu zachowują się one zgodnie z przewidywaniami - większe skomplikowanie sieci neuronowej zmniejszyło

**Tabela 3.3.** Średnie wartości *MSE* danych przy uśrednionych danych.

Ilość warstw	Ilość neuronów w warstwie	MSE danych treningowych	MSE danych testowych
1	20	1,38 ( $\pm 0,81$ )	189,06 ( $\pm 60,12$ )
1	40	2,21 ( $\pm 2,02$ )	182,91 ( $\pm 50,58$ )
2	20	6,06 ( $\pm 8,93$ )	113,56 ( $\pm 55,77$ )
2	40	2,02 ( $\pm 4,38$ )	108,03 ( $\pm 40,39$ )

dramatycznie *MSE* danych treningowych. Co było również spodziewane - *MSE* danych testowych pozostał wysoki; jednakże na wykresach 3.11 (c) i (d) widać ciekawą tendencję - wygląda na to, że przy większych rozmiarach zbioru danych nawet *MSE* zbioru testowego zaczyna się obniżać. Podobną tendencję, lecz słabszą można też zauważyć na wykresach 3.11 (a) i (b) po przekroczeniu pułapu 140 zestawów danych uczących. Gdy model jest zbyt dobrze przyuczony do danych treningowych i źle generalizuje na danych testowych, mówi się że posiada on **wysoką wariancję**.

Ze względu na duży czas oczekiwania oraz potrzebę ciągłego nadzoru nad symulatorem, postanowiłem w pierwszej kolejności poszukać innych sposobów na zwiększenie efektywności modelu.

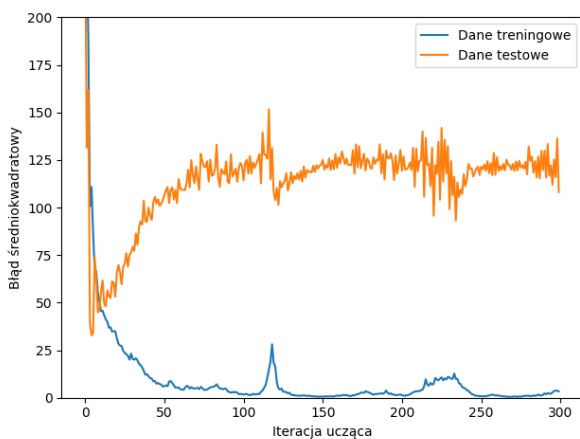
### 3.8. Regularyzacja modelu

Chcąc obniżyć wariancję modelu uczenia maszynowego stosuje się techniki zwane regularyzacją. Najprostsze z nich czyli regularyzacja **L1** i **L2** polegają na dodaniu współczynnika kary do funkcji kosztu. Karze ona model za posiadanie wysokich wag na połączeniach pomiędzy neuronami. Dla porównania mając funkcję kosztu 3.3, używając regularyzacji *L1* będzie ona wyglądać jak na równaniu 3.4 a regularyzacji *L2* - tak jak na równaniu 3.5. Współczynnik  $\lambda$  to tak zwany **współczynnik regularyzacji** - odpowiada on za siłę, z jaką wagi będą zmniejszane.

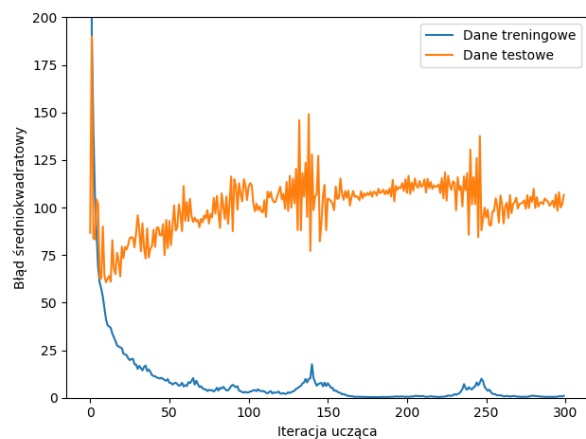
$$S = \sum_{i=1}^n (y_i - f(x_i))^2 \quad (3.3)$$

$$S = \sum_{i=1}^n (y_i - f(x_i))^2 + \lambda \sum_{i=1}^k |w_i| \quad (3.4)$$

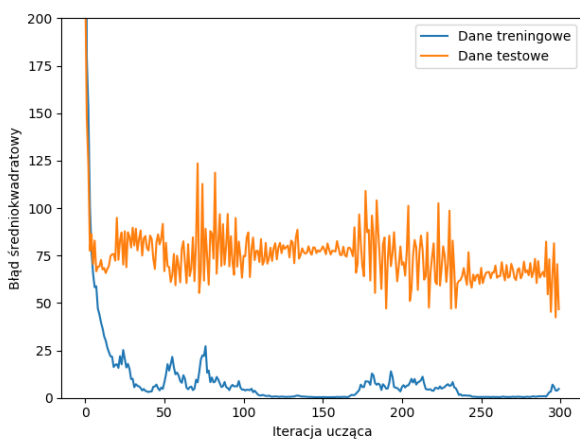
$$S = \sum_{i=1}^n (y_i - f(x_i))^2 + \lambda \sum_{i=1}^k w_i^2 \quad (3.5)$$



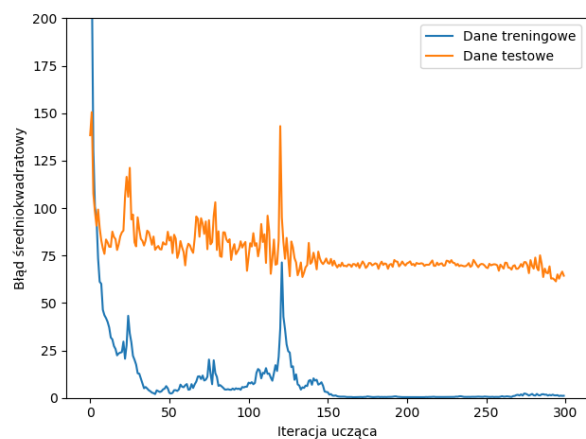
(a) 1 warstwa ukryta, 20 neuronów w każdej warstwie



(b) 1 warstwa ukryta, 40 neuronów w każdej warstwie



(c) 2 warstwy ukryte, 20 neuronów w każdej warstwie



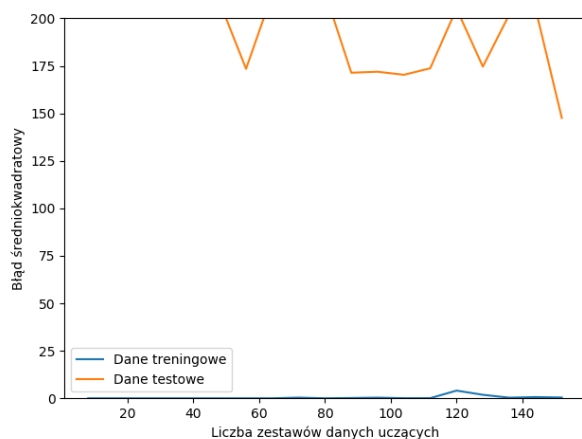
(d) 2 warstwy ukryte, 40 neuronów w każdej warstwie

**Rys. 3.10.** Krzywe uczące dla najlepszych przypadków przy uśrednionych danych w drugim modelu.

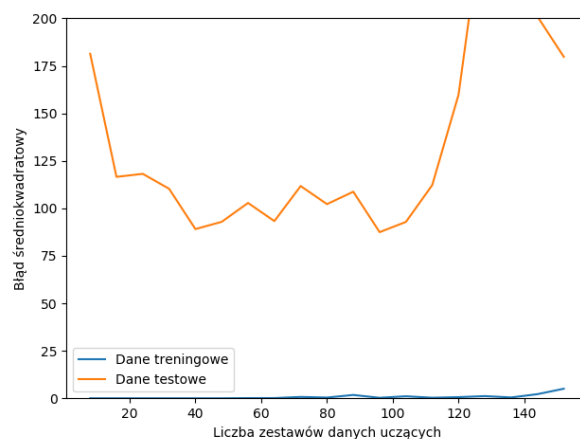
Bazując na badaniach z poprzedniego rozdziału, wybrałem model z jedną 20-neuronową warstwą ukrytą aby przeprowadzić na nim eksperymenty z regularyzacją. Mój wybór był umotywowany tym, że był to najprostszy z 4 modeli, a jednocześnie miał możliwość prawie idealnego nauczenia się danych treningowych. Parametry regularyzacji jakie wybrałem do badań to:

- funkcje regularyzujące - **L1** i **L2**,
- parametr regularyzacji  $\lambda$  - **0,1; 0,3; 1; 3**.

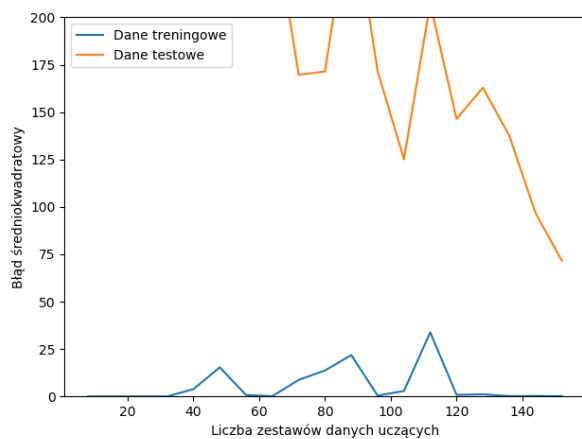




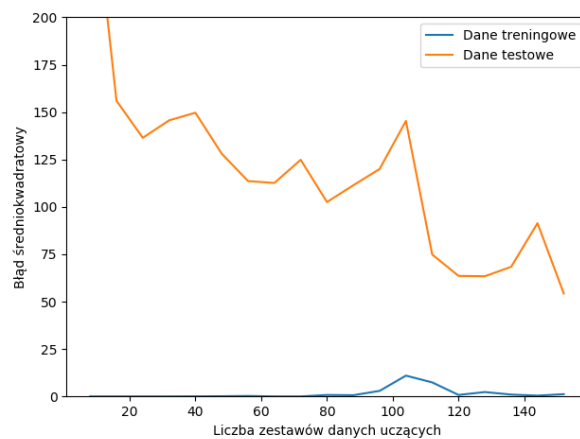
(a) 1 warstwa ukryta, 20 neuronów w każdej warstwie



(b) 1 warstwa ukryta, 40 neuronów w każdej warstwie



(c) 2 warstwy ukryte, 20 neuronów w każdej warstwie

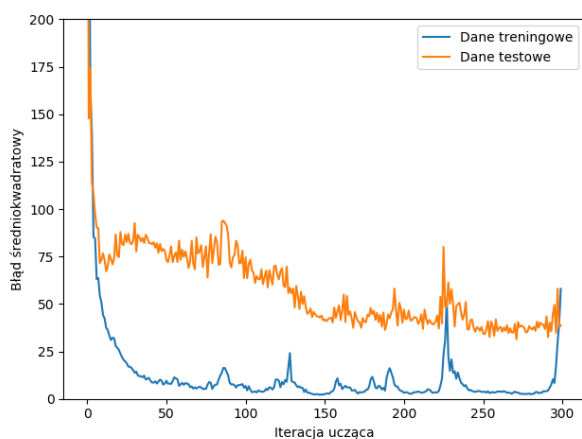
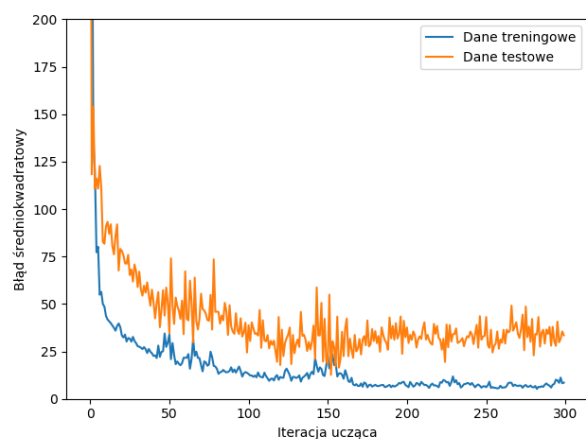
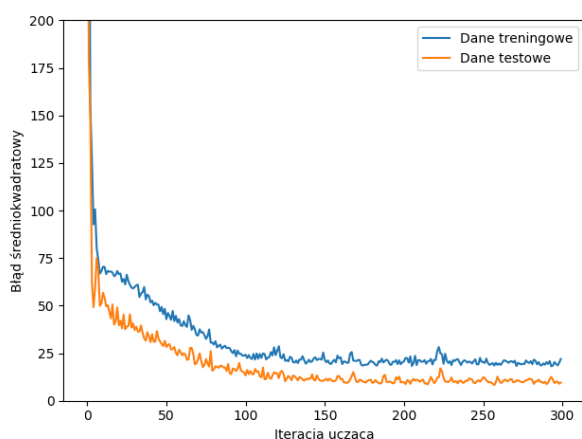
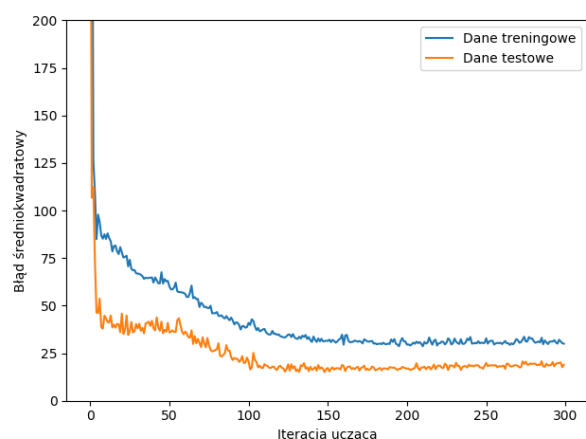


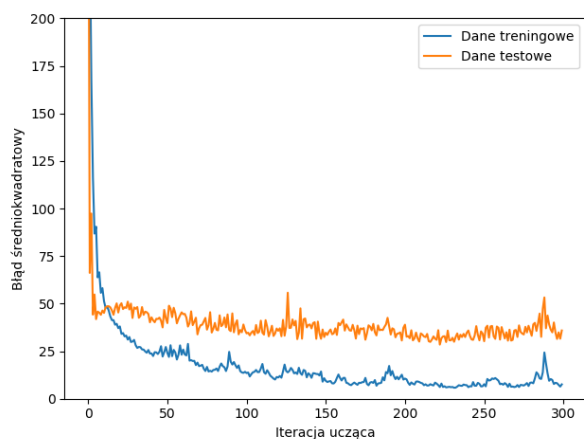
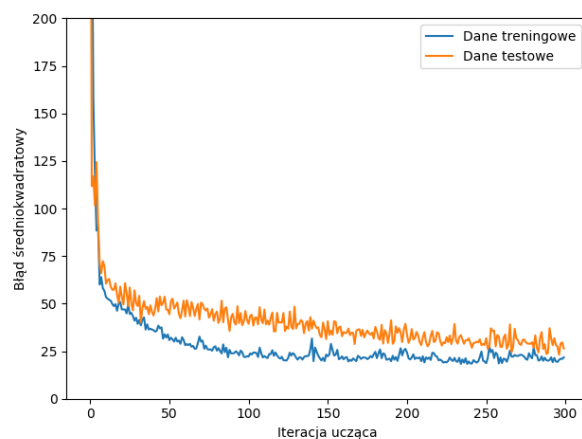
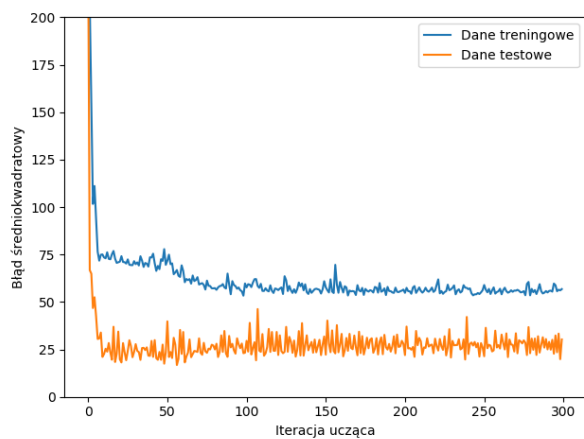
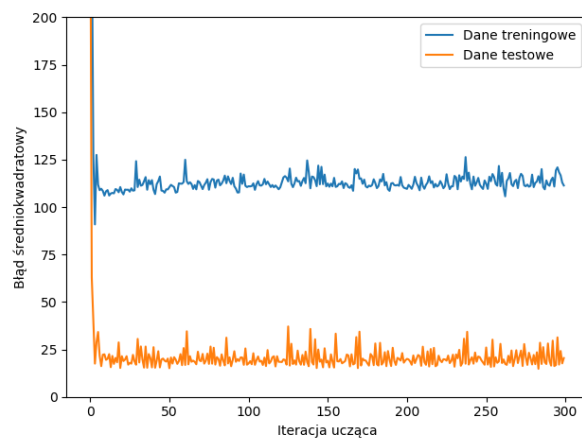
(d) 2 warstwy ukryte, 40 neuronów w każdej warstwie

**Rys. 3.11.** Krzywe uczące dla zmiennej ilości uśrednionych danych w drugim modelu.

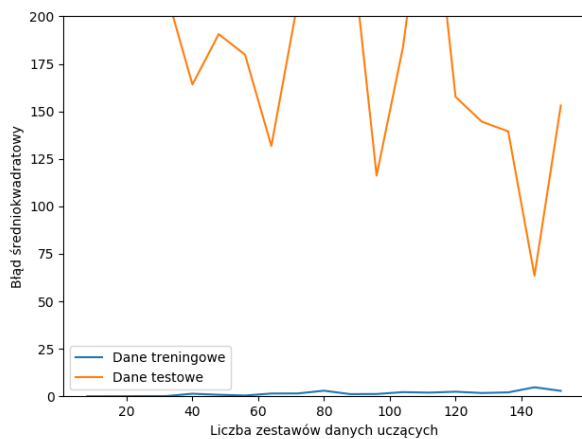
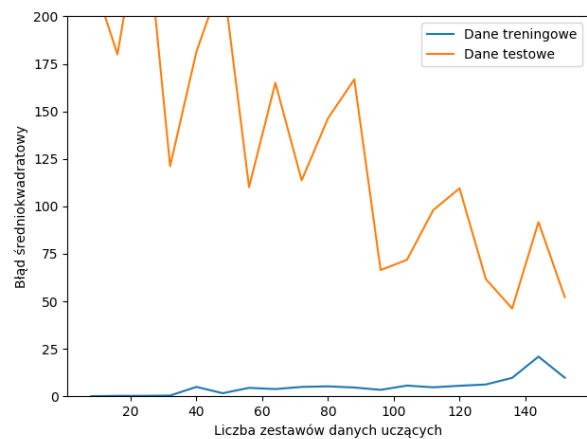
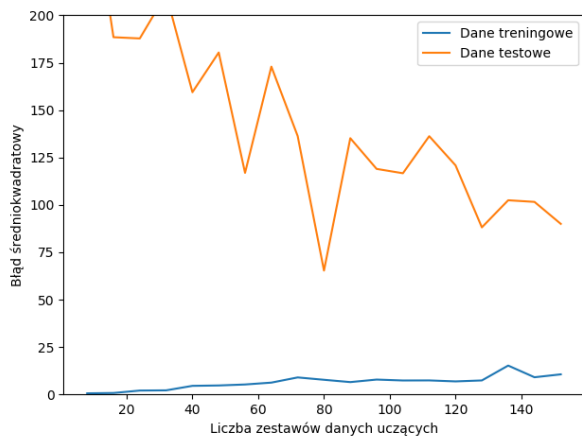
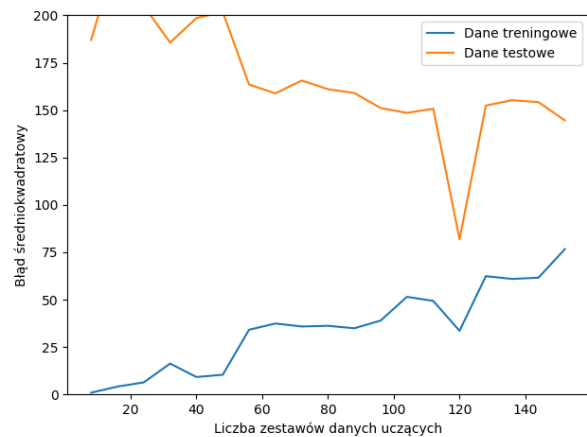
**Tabela 3.4.** Średnie wartości *MSE* danych przy regularyzacji.

Rodzaj regularyzacji	Ilość neuronów w warstwie	MSE danych treningowych	MSE danych testow
L1	0.1	11,65 ( $\pm 15,70$ )	99,61 ( $\pm 43,27$ )
L1	0.3	7,68 ( $\pm 1,04$ )	82,24 ( $\pm 39,32$ )
L1	1	15,89 ( $\pm 3,44$ )	36,21 ( $\pm 24,35$ )
L1	3	40,44 ( $\pm 21,46$ )	64,26 ( $\pm 39,62$ )
L2	0.1	7,42 ( $\pm 1,74$ )	102,93 ( $\pm 47,46$ )
L2	0.3	20,04 ( $\pm 3,16$ )	76,49 ( $\pm 36,76$ )
L2	1	53,44 ( $\pm 4,77$ )	81,18 ( $\pm 43,06$ )
L2	3	103,04 ( $\pm 7,29$ )	62,86 ( $\pm 34,93$ )

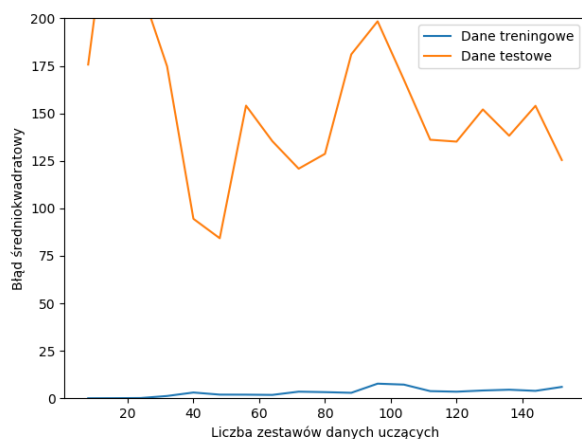
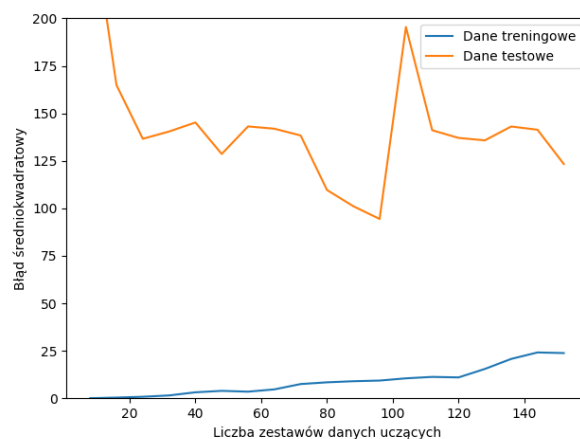
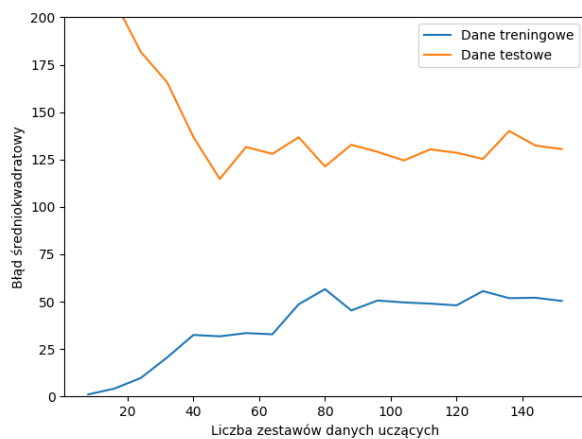
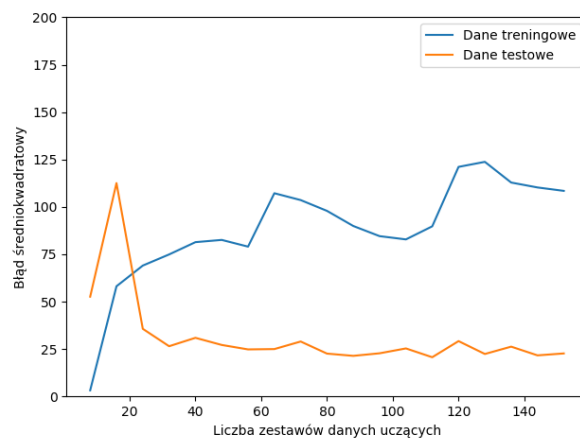
(a) parametr  $\lambda - 0.1$ (b) parametr  $\lambda - 0.3$ (c) parametr  $\lambda - 1$ (d) parametr  $\lambda - 3$ **Rys. 3.12.** Krzywe uczące dla najlepszych przypadków używając regularyzacji L1 w drugim modelu.

(a) parametr  $\lambda - 0.1$ (b) parametr  $\lambda - 0.3$ (c) parametr  $\lambda - 1$ (d) parametr  $\lambda - 3$ 

**Rys. 3.13.** Krzywe uczące dla najlepszych przypadków używając regularyzacji L2 w drugim modelu.

(a) parametr  $\lambda - 0.1$ (b) parametr  $\lambda - 0.3$ (c) parametr  $\lambda - 1$ (d) parametr  $\lambda - 3$ 

**Rys. 3.14.** Krzywe uczące dla zmiennej ilości danych używając regularyzacji L1 w drugim modelu.

(a) parametr  $\lambda - 0.1$ (b) parametr  $\lambda - 0.3$ (c) parametr  $\lambda - 1$ (d) parametr  $\lambda - 3$ 

**Rys. 3.15.** Krzywe uczące dla zmiennej ilości danych używając regularyzacji L2 w drugim modelu.



## Bibliografia

- [1] L. Lamport. *LaTeX system przygotowywania dokumentów*. Kraków: Wydawnictwo Ariel, 1992.
- [2] *Ada Reference Manual ISO/IEC 8652:200y(E) Ed. 3*. Ada Europe. 2006.
- [3] A. Burns i B. Dobbing. „The Ravenscar Profile for Real–Time and High Integrity Systems”. W: *CrossTalk* 16.11 (2003), s. 9–12.