Spis treści

1 Wstęp **2**
- 1.1 Cel projektu . . . 2
- 1.2 Motywacja . . . 2

2 Problem medyczny **3**
- 2.1 Opis chorób . . . 3
- 2.2 Opis cech . . . 3
- 2.3 Selekcja cech . . . 6
  - 2.3.1 Test chi2 . . . 6

3 Techologie **9**
- 3.1 Python . . . 9
- 3.2 NumPy . . . 10
- 3.3 matplotlib . . . 10
- 3.4 pandas . . . 10
- 3.5 Git . . . 11
- 3.6 Docker . . . 12

4 Sieć neuronowa **13**
- 4.1 Wprowadzenie . . . 13
- 4.2 Neuron . . . 13
  - 4.2.1 Funkcja aktywacji . . . 14
- 4.3 Model wielowarstwowy . . . 15
  - 4.3.1 Proces uczenia . . . 16

5 Opis architektury aplikacji **17**
- 5.1 Schemat warstwy . . . 17
- 5.2 Tworzenie architektury sieci . . . 18
- 5.3 Schemat modelu . . . 18

6 Przeprowadzone badania **20**
- 6.1 Różne współczynniki uczenia . . . 21
- 6.2 Różna liczba cech . . . 23
- 6.3 Różne architektury . . . 26
- 6.4 Różna liczba sech - tanh . . . 28
- 6.5 Różne architektury - tanh . . . 30

7 Podsumowanie **32**
- 7.1 Dalsze możliwości rozwoju . . . 32
- 7.2 Co mogłem zrobić lepiej . . . 32

Rozdział 1

Wstęp

1.1 Cel projektu

1.2 Motywacja

Rozdział 2

Problem medyczny

Wybrany przeze mnie problem medyczny dotyczy klasyfikacji stanów ostrego brzucha. Za ten stan odpowiedzialne mogą być różne choroby, które zawsze wymagają interwejcji lekarza.

2.1 Opis chorób

Do klasyfikacji jest 8 chorób, zatem sieć neuronowa będzie miała za zadanie przypisać 1 z 8 klas. Są to:

1. Ostre zapalenie wyrostka robaczkowego,
2. Zapalenie uchyłków jelit,
3. Niedrożność mechaniczna jelit,
4. Perforowany wrzód trawienny,
5. Zapalenie woreczka żółciowego,
6. Ostre zapalenie trzustki,
7. Niecharakterystyczny ból brzucha,
8. Inne przyczyny ostrego bólu brzucha.

Histogram pokazuje, że rozkład klas jest nierównomierny. Na 476 obiektów aż 157 to „Niecharakterystyczny ból brzucha" i 141 ma etykietę „Ostre zapalenie wyrostka robaczkowego". Czyli do 2 klas należy ponad 60% obiektów. Może to mieć negatywny wpływ na jakość klasyfikacji.

2.2 Opis cech

Dane do tego problemu zawierają 31 cech. Są to odpowiedzi z wywiadu medycznego i wyniki przeprowadzonych badań. Możliwe wartości parametrów przedstawione są poniżej. Jak widać wszystkie liczby są naturalne mniejsze niż 11, także normalizacja czy skalowanie danych nie jest konieczne.

Tablica 2.1: Wszystkie cechy z odpowiedziami

L.p.	**Pytanie**	**Możliwe odpowiedzi**
	Ogólne	
1	Płeć	1) męska 2) żeńska

Kontynuacja na następnej stronie

Tablica 2.1 – *Wszystkie cechy z odpowiedziami - c.d.*

L.p.	Pytanie	Możliwe odpowiedzi
2	Wiek	1) poniżej 20 lat 2) 20 - 30 lat 3) 21 - 30 lat 4) 31 - 40 lat 5) 41 - 50 lat 6) powyżej 50 lat
	Ból	
3	Lokalizacja bólu na początku zachorowania	1) prawa górna ćwiartka 2) lewa górna ćwiartka 3) górna połowa 4) prawa połowa 5) lewa połowa 6) centralny kwadrat 7) cały brzuch 8) prawa dolna ćwiartka 9) lewa dolna ćwiartka 10) dolna połowa
4	Lokalizacja bólu obecnie	0) brak bólu 1) prawa górna ćwiartka 2) lewa górna ćwiartka 3) górna połowa 4) prawa połowa 5) lewa połowa 6) centralny kwadrat 7) cały brzuch 8) prawa dolna ćwiartka 9) lewa dolna ćwiartka 10) dolna połowa
5	Intensywność bólu	0) łagodny/brak 1) umiarkowany 2) silny
6	Czynniki nasilające ból	0) brak czynników 1) oddychanie 2) kaszel 3) ruchy ciała
7	Czynniki przynoszące ulgę	0) brak czynników 1) wymioty 2) pozycja ciała
8	Progresja bólu	1) ustepujący 2) bez zmian 3) nasilający się
9	Czas trwania bólu	1) mniej niż 12 godzin 2) 12 - 24 godzin 3) 24 - 48 godzin 4) powyżej 48 godzin
10	Charakter bólu na początku zachorowania	1) przerywany 2) stały 3) kolkowy

Kontynuacja na następnej stronie

Tablica 2.1 – *Wszystkie cechy z odpowiedziami - c.d.*

L.p.	Pytanie	Możliwe odpowiedzi
11	Charakter bólu obecnie	0) brak bólu 1) przerywany 2) stały 3) kolkowy
	Inne objawy	
12	Nudności i wymioty	0) brak 1) nudności bez wymiotów 2) nudności z wymiotami
13	Apetyt	1) zmniejszony 2) normalny 3) zwiększony
14	Wypróżnienia	1) biegunki 2) prawidłowe 3) zaparcia
15	Oddawanie moczu	1) normalne 2) dysuria
	Historia	
16	Poprzednie niestrawności	0) nie 1) tak
17	Żółtaczka w przeszłości	0) nie 1) tak
18	Poprzednie operacje brzuszne	0) nie 1) tak
19	Leki	0) nie 1) tak
	Ogólne badanie	
20	Stan psychiczny	1) pobudzony/cierpiący 2) prawidłowy 3) apatyczny
21	Skóra	1) blada 2) prawidłowa 3) zaczerwieniona (twarz)
22	Temperatura (pacha)	1) poniżej 36.5 stC 2) 36.5 - 37 stC 3) 37 - 37.5 stC 4) 37.5 - 38 stC 5) 38 - 39 stC 6) powyżej 39 stC
23	Tętno	1) poniżej 60 /min 2) 60 - 70 /min 3) 70 - 80 /min 4) 80 - 90 /min 5) 90 - 100 /min 6) 100 - 110 /min 7) 110 - 120 /min 8) 120 - 130 /min 9) powyżej 130 /min
	Oglądanie brzucha	
24	Ruchy oddechowe powłok brzusznych	1) normalne 2) zniesione

Kontynuacja na następnej stronie

Tablica 2.1 – *Wszystkie cechy z odpowiedziami - c.d.*

L.p.	Pytanie	Możliwe odpowiedzi
25	Wzdęcia	0) nie 1) tak
Badania palpacyjne		
26	Umiejscowienie bolesności uciskowej	0) brak bólu 1) prawa górna ćwiartka 2) lewa górna ćwiartka 3) górna połowa 4) prawa połowa 5) lewa połowa 6) centralny kwadrat 7) cały brzuch 8) prawa dolna ćwiartka 9) lewa dolna ćwiartka 10) dolna połowa
27	Objaw Blumberga	0) negatywny 1) pozytywny
28	Obrona mięśniowa	0) nie 1) tak
29	Wzmożone napięcie powłok brzusznych	0) nie 1) tak
30	Opory patologiczne	0) nie 1) tak
31	Objaw Murphy'ego	0) negatywny 1) pozytywny

2.3 Selekcja cech

Selekcja cech jest procesem wymaganym, gdy dane nie są dobrej jakości w wielu algorytmach uczenia maszynowego. Polega ona na wyborze podzbioru najlepszych cech według ustalonego kryterium. Analitycy danych przeprowadzają selekcję z następujących powodów:

- uproszczenie modelu, w celu ułatwienia interpretacji przez badaczy,
- skrócenie czasu treningu,
- zmniejszenie wymiarowości modelu,
- zwiększenie generalizacji poprzez uniknięcie zjawiska przeuczenia.

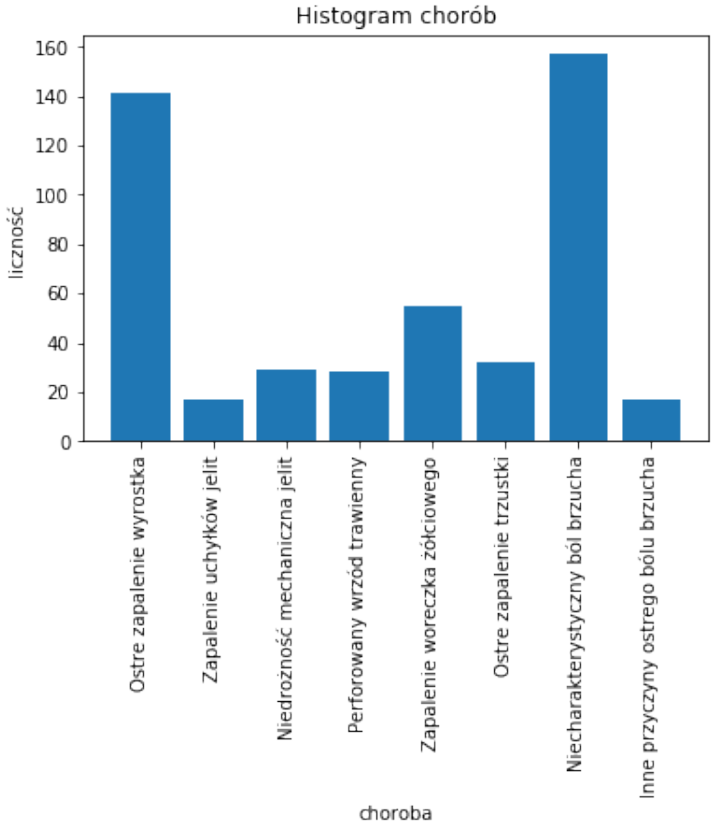
2.3.1 Test chi2

Metoda, którą wybrałem to test chi2. Jest to jedna z technik nieparametrycznych. Nadaje się bardzo dobrze do oceny istotności statystycznej cechy. Test ten polega na obliczeniu podanego poniżej wyrażenia dla każdej z cech i wybraniu takich, dla których wartość jest największa.

$$\chi^2 = \sum_{i=1}^{n} \frac{(O_i - E_i)^2}{E_i}$$

Gdzie:

- O_i - wartość mierzona,
- E_i - wartość oczekiwana,



Rysunek 2.1: Histogram występowania chorób

- n - liczba obiektów.

Wartości testu dla wszystkich cech mają następujące wartości:

Tablica 2.2: Wartości chi2 dla wszystkich cech

L.p.	**Cecha**	**Wartość chi2**
1	Charakter bólu obecnie	127.811
2	Czynniki przynoszące ulgę	87.453
3	Nudności i wymioty	84.633
4	Czas trwania bólu	84.273
5	Umiejscowienie bolesności uciskowej	77.456
6	Lokalizacja bólu obecnie	70.865
7	Czynniki nasilające ból	59.357
8	Tętno	58.152
9	Apetyt	54.489
10	Wypróżnienia	42.184
11	Charakter bólu na początku zachorowania	32.127
12	Lokalizacja bólu na początku zachorowania	31.430
13	Ruchy oddechowe powłok brzusznych	31.192
14	Progresja bólu	30.502
15	Objaw Blumberga	21.387
16	Wiek	21.228
17	Skóra	20.202
18	Intensywność bólu	18.438
19	Temperatura (pacha)	17.708
20	Stan psychiczny	15.930
21	Leki	15.554
22	Objaw Murphy'ego	13.666
23	Obrona mięśniowa	13.062
24	Oddawanie moczu	12.322
25	Wzmożone napięcie powłok brzusznych	11.406
26	Wzdęcia	8.771
27	Opory patologiczne	8.504
28	Poprzednie operacje brzuszne	7.007
29	Płeć	6.195
30	Poprzednie niestrawności	4.470
31	Żółtaczka w przeszłości	0.590

Najlepszymi cechami są te, które mają wysoką wartość chi2. Zatem ograniczając liczbę cech, do klasyfikacji brane będą te z góry tabeli. Cechy o niskiej wartości, jak na przykład „Żółtaczka w przeszłości", nie polepszą klasyfikacji, a mogą ją nawet pogorszyć.

Rozdział 3

Techologie

3.1 Python

Rysunek 3.1: Logo języka Python

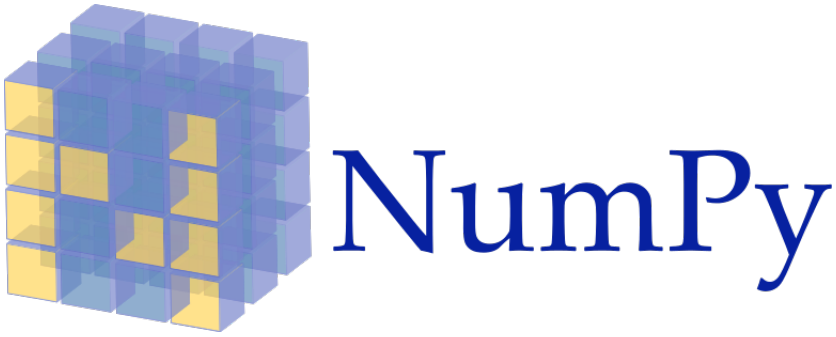
Python to otwarto-źródłowy, wysoko poziomowy język programowania ogólnego przeznaczenia. Stworzony został 26 lat temu przez holenderskiego programistę Guido van Rossuma. Najpolularniejszy interpreter Pythona napisany jest w języku C. Jednak w odróżnieniu od C, C++ i Javy, Python jest interpretowalny i nie używa się w nim nawiasów klamrowych do oddzielenia bloków kodu. Jest przez to bardziej czytelny i nie odstrasza ludzi aspirujących do bycia programistami. Zamias klamr stosuje się wcięcia w kodzie, które powinny wynosić 4 spacje na każdy poziom. Od wersji 3.5 w Pythonie można jawnie stosować typowanie, czyli na przykład twórca funkcji może umieścić informację w kodzie, jakiego typu powinny być argumenty i jaki typ funkcja zwraca. Dzięki temu czas potrzebny na zrozumienie cudzego kodu staje się krótszy.

Python ma wiele zastosowań:

- nauka programowania,
- web development,
- aplikacje konsolowe,
- aplikacje okienkowe,
- gry komputerowe,
- naukowe,
- analiza danych.

Od kilku lat Python zyskuje duże zainteresowanie naukowców z różnych dziedzin nauki z racji swojej prostoty i wszechstronności. Powstało również wiele gotowych modułów do zastosowania w uczeniu maszynowym, ale nie będę używał ich w tym projekcie.

3.2 NumPy



Rysunek 3.2: Logo NumPy

NumPy to otwarto-źródłowa biblioteka do Pythona służąca do obliczeń naukowych. Umożliwia przechowywanie danych w wielowymiarowych tablicach i macierzach (tensorach) oraz wykonywanie skomplikowanych funkcji na nich. Napisana została w większości w języku C, co sprawia, że kod wykonywany jest szybciej niż w samym Pythonie. Tablice z NumPy są wykorzystywane w wielu bibliotekach, jako podstawowa struktura danych. W tym projekcie używam jej do przechowywania wag w każdej warstwie sieci.

3.3 matplotlib

Rysunek 3.3: Logo matplotlib

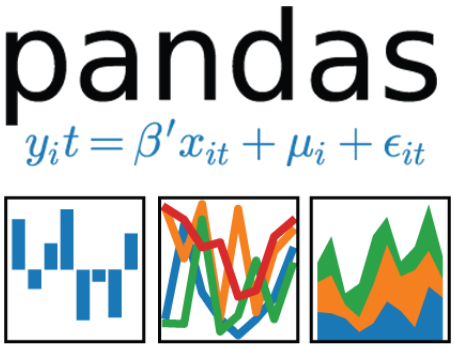
Matplotlib to najpopularniejsza biblioteka do tworzenia wykresów w Pythonie. Wraz z biblioteką NumPy bardzo często wykorzystywana jest do analizy i wizualizacji danych. Jest bardzo prosta w obsłudze. W kilka linii jesteśmy w stanie stworzyć prosty wykres i wyeksportować go do pliku graficznego. Wspiera takie typy wykresów jak:

- liniowy,
- histogram,
- punktowy,
- 3D,
- biegunowy.

Pozwala również wyświetlać obrazy w oknach z poziomu skryptu w Pythonie.

3.4 pandas

Pandas to biblioteka napisana w Pythonie służąca do manipulacji i analizy danych. Oferuje struktury danych, które ułatwiają operowanie na plikach csv, json i xlsx. Umożliwia operacje podobne do znanych z języka SQL. Są to: grupowanie danych, sortowanie po indeksie lub po innej kolumnie, łączenie tabel i usuwanie duplikatów.



Rysunek 3.4: Logo pandas

3.5 Git

Rysunek 3.5: Logo Gita

Git to rozproszony system kontroli wersji, czyli narzędzie do śledzenia zmian w plikach źródłowych. Jest to oprogramowanie używane głównie do zarządzania kodem, ale może być używane również do trzymania historii innych plików. Git ma na celu szybkość, spójność danych i wspieranie pracy rozproszonej wśród zespołów. Nie wymaga ciągłego dostępu do Internetu. Jest wykorzystywany w prawie wszystkich nowoczesnych projektach.

Git został napisany przez Linusa Torvaldsa w 2005 roku, jako narzędzie do tworzenia jądra Linuksa, gdyż żaden inny system kontroli wersji nie spełniał jego wymagań.

Główną strukturą w Gicie jest repozytorium. Każde repozytorium przypisane jest do jednego projektu. Posiada ono historię w formie grafu skierowanego, który jest drzewem. Git umożliwia poruszanie się po tym drzewie pozwalając przeglądać repozytorium w danym stanie.

Praca z Gitem rozpoczyna się od sklonowania istniejącego repozytorium lub stworzenia nowego, pustego. Użytkownik po zmianie jakiegoś pliku śledzonego przez Gita może zrobić „commit", czyli zapisać obecny stan projektu. Każdy „commit" ma przypisaną wiadomość, w której twórca „commita" informuje, co zmienił. Po „scommitowaniu" można zsynchronizować stan repozytorium z głównym serwerem. Dopiero wtedy inni użytkownicy mogą zobaczyć, jakie zaszły zmiany i pobrać do siebie najnowszą wersję.

Git to potężne narzędzie, każdy programista powinien potrafić z niego korzystać. Łatwo jest poznać podstawy Gita i nie wymaga dużo czasu opanowanie ich. Zaawansowana znajomość Gita pozwala na robienie niesamowitych rzeczy w repozytorium.

W projekcie inżynierskim korzystam z Gita do zapisywania postępów w tworzeniu aplikacji. Kod jest przechowywany na serwerze firmy GitHub.

Rysunek 3.6: Logo Dockera

3.6 Docker

Docker to otwarto-źródłowe narzędzie służące do konteneryzacji aplikacji. Zapewnia dodatkową warstę abstrakcji nad systemem operacyjnym. Działa zarówno na Linuksie, jak i na Windowsie. Pierwsze wersje Dockera od 2013 roku napisane były w Pythonie, a kolejne w języku Go.

W wielu aplikacjach Docker używany jest w celu ułatwienia wdrożenie aplikacji na serwery produkcyjne. Jest przydatny również w czasie wytwarzania dla deweloperów, gdyż idealnie nadaje się na środowisko testowe.

Docker udostępnia na swojej stronie internetowej wiele predefiniowanych obrazów z zainstalowanymi aplikacjami, które są gotowe do użycia. Zalogowani użytkownicy mogą również publikować swoje własne obrazy zbudowane przez nich. Pozwala to dzielić się swoją pracą z całą społecznością.

Praca z Dockerem polega na uruchomieniu kontenera z wybranego obrazu. Obraz dockerowy to tak jakby zapisany stan maszyny wirtualnej. Kontener jest konkretną uruchomioną instancją obrazu.

Poza oficjalnymi obrazami, istnieje również możliwość tworzenia własnych obrazów do poszczególnych aplikacji. Polega to na stworzeniu pliku domyślnie o nazwie Dockerfile, gdzie podany jest obraz bazowy oraz lista komend do wykonania. Po zbudowaniu obrazu jedną komendą, można uruchomiać kontenery.

W swoim projekcie inżynierskim korzystałem z Dockera, badania przeprowadzałem na serwerze, gdzie nie ma zainstalowanych wymaganych przeze mnie zależności. Dlatego to było jedynym rozwiązaniem.

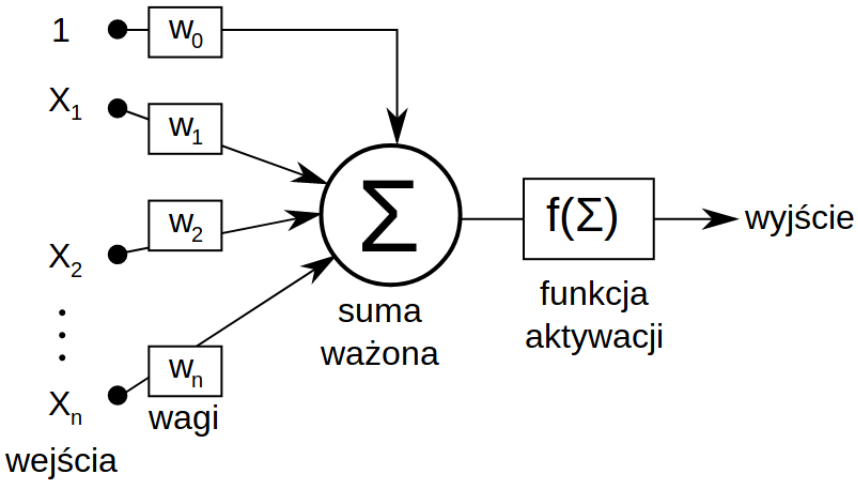
Rozdział 4

Sieć neuronowa

4.1 Wprowadzenie

Sztuczna sieć neuronowa to pewna struktura matematyczna, która może być zaimplementowana programowo lub sprzętowo. Początkowo taki twórcy takich modeli inspirowali się zwierzęcym mózgiem, w którym połączone ze sobą neurony tworzą sieć. Taka sieć przetwarza sygnały wejściowe wykonując na nich pewne operacje. Sieci wykorzystywane są często do rozwiązywania problemów klasyfikacji, z racji na ich zdolność uczenia. Np. potrafią przetwarzać zdjęcia i opisywać, co się na nich znajduje. Przed skorzystaniem z sieci należy ją nauczyć, co sprowadza się do przekazywania na wejście sieci danych uczących razem z poprawną klasą, do której dane obiekty należą.

4.2 Neuron



Rysunek 4.1: Schemat neuronu

$$s = \sum_{i=0}^{n} w^{(i)} \cdot x^{(i)} = w^T \cdot x$$

Neuron stanowi podstawowy budulec sztucznej sieci neuronowej. Składa się z ustalonej liczby wejść, wraz z odpowiadającymi im wagami. Ponadto neuron zawiera nieliniową funkcję aktywacji oraz jedno wyjście. Jego zadanie to obliczenie iloczynu skalarnego wektora wejść z wektorem wag. Dodatkowo możemy przyjąć, że bias jest dodatkowym wejściem neuronu o wartości 1. Następnie obliczona suma ważona poddawana jest

funkcji aktywacji i przekazywana na wyjście neuronu. W procesie uczenia wagi w neuronie zmieniają się tak, by wyliczona wartość funkcji błędu była jak najmniejsza.

4.2.1 Funkcja aktywacji

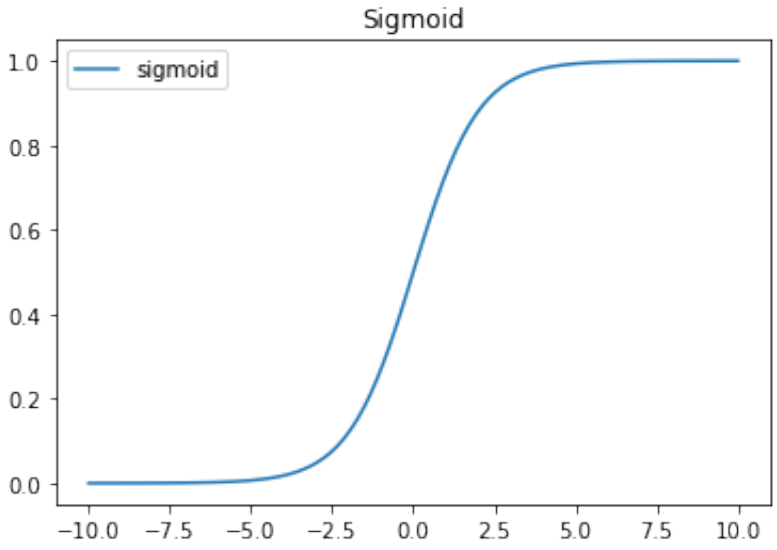
Funkcja aktywacji to funkcja, która wykorzystywana jest w sztucznych sieciach neuronowych, a dokładniej w samym neuronie do zmiany wartości wyjścia. Ma to na celu sprawienie, że sieć jest w stanie lepiej się uczyć nawet przy małej liczby neuronów. W uczeniu maszynowym znanych jest wiele rodzajów takich funkcji. W tej pracy opiszę tylko dwie, które użyłem do budowy sieci neuronowej.

$$y = f(s) = f(w^T \cdot x)$$

Sigmoid

Pierwszą opisywaną funkcją jest sigmoid, zwana też „sigmoidalną funkcją unipolarną". Bardzo dobrze nadaje się jako funkcja aktywacji, gdyż jej dziedzina to cały zbiór liczb rzeczywistych. Ma tę cechę, że zbiór warości mieści się w zakresie (0, 1). Jest to również wada, że szybko się „nasyca". Kolejnym minusem tej funkcji jest to, że wartości nie zcentralizowane wokół zera. Ponadto wykorzystuje funkcję ekpotencjalną, która jest kosztowna obliczeniowo. Jej wzór to:

$$\sigma(x) = \frac{1}{1 + e^{-x}}$$

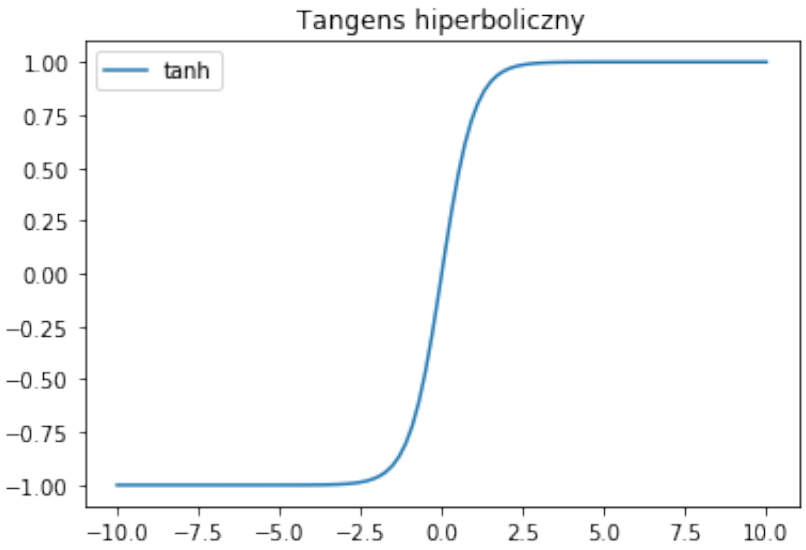


Rysunek 4.2: Funkcja aktywacji - sigmoid

Tangens hiperboliczny

Kolejna funkcja, która wykorzystywana jest w sieciach neuronowych to tangens hiperboliczny (tanh). W tym przypadku również dziedziną jest zbiór liczb rzeczywistych. Spłaszcza wyjście w zakresie (-1, 1). Podobnie jak sigmoid szybko się „nasyca". W przeciwieństwie do poprzedniej funkcji jest scentralizowany wokół zera. Tanh również korzysta z funkcji ekpotencjalnej. Jej wzór to:

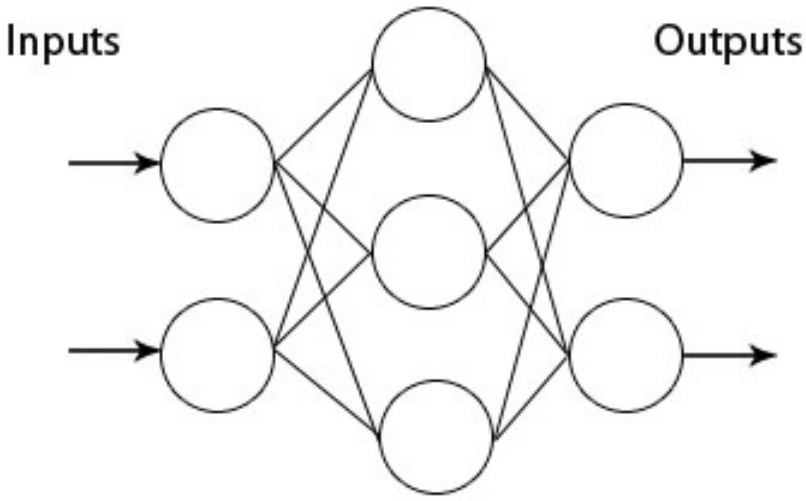
$$\text{tgh}(x) = \frac{2}{1 + e^{-2x}} - 1$$



Rysunek 4.3: Funkcja aktywacji - tanh

4.3 Model wielowarstwowy

Kiedy pojedyncze neurony mają te same sygnały wejściowe, tworzą wówczas tak zwaną warstwę w sieci neuronowej. Dlatego sieć neuronowa ma budowę warstwową. Zawsze występuje warstwa wejściowa i wyjściowa. Ponadto mogą występować warstwy ukryte. Ich ilość zależy od rozwiązywanego problemu.



Rysunek 4.4: Schemat neuronu

W warstwie wejściowej znajduje się tyle neuronów, ile jest badanych cech. W moim przypadku będzie mniej niż 31. Neurony w tej warstwie nie mają wag, lecz przekazują dalej dokładnie to, co otrzymały.

Liczba neuronów w warstwie wyjściowej również nie jest przypadkowa. Warstwa ta składa się z takiej samej liczby neuronów, co liczba klas w zadanym problemie. Problem medyczny, na którym pracuję dotyczy klasyfikacji ośmio klasowej, dlatego będzie osiem neuronów wyjściowych. Każdy z nich będzie zwracał wartość

przynależności do danej klasy.

4.3.1 Proces uczenia

Uczenie sieci neuronowej, w którym znamy poprawne klasy dla danych uczących nazywamy uczeniem nadzorowanym lub uczeniem z nauczycielem. Polega na porównaniu wyjścia sieci z wartością oczekiwaną i dostrajaniu wag w neuronach tak, by minimalizować pewną ustaloną funkcję błędu.

Korekcja wag odbywa się w procesie zwanym propagacją wsteczną. Propagacja wsteczna korzysta z gradientowych metod optymalizacji, które są wydajne obliczeniowo i bardzo skuteczne do uczenia sieci. Propagacja wsteczna wymaga również funkcję błędu. W tym celu stosuje się tak zwany „błąd średnio-kwadratowy", który jest obliczany następująco:

$$Q(w_k) = \sum_{k=1}^{N} (d_k - y_k)^2$$

gdzie:

- Q - błąd średniokwadratowy,
- w_k - wektor wag w k-tym neuronie,
- N - liczba danych uczących,
- d_k - poprawna wartość,
- y_k - aktualna wartość,

Algorytm jest wieloetapowy. Pierwszym krokiem jest dostarczenie obiektu na wejście sieci i sprawdzenie jakie otrzymamy wyjście. Następnie rekurencyjnie zmniejszana jest wartość funkcji błędu od końca sieci zmieniając wagi w kolejnych warstwach.

Tutaj korzysta się ze współczynnika uczenia. Na początku powinien mieć ustaloną wartość, a następnie wraz z kolejnymi epokami uczenia może się zmniejszać.

Można to porównać do stania nad doliną górską, gdzie chcemy dość do najniższego punktu. Najpierw należy ustalić kierunek marszu, by później kroczyć w tym kierunku. Na początku kroki mogą być ogromne, ale w miarę schodzenia, powinniśmy je zmniejszać, by nie przekroczyć minimum.

Rozdział 5

Opis architektury aplikacji

5.1 Schemat warstwy

Listing 5.1: Schemat klasy Layer

```
class Layer:
    def __init__(self, shape, activation='sigmoid'):
        ...

    def feedforward(self, x: np.ndarray) -> np.ndarray:
        ...

    def calc_delta(self, d: np.ndarray = None):
        ...

    def calc_gradient(self):
        ...

    def update_weights(self, learning_rate=.2):
        ...
```

Powyższy fragment kodu przedstawia schemat klasy Layer. Jest to implementacja jednej warstwy w sieci neuronowej. Przypomina schemat struktury danych zwanej listą dwukierunkową, gdyż zawiera referencje do poprzedniej i następnej warstwy. Klasa zawiera w sobie tablicę, która jest składa się z wag połączeń do poprzedniej warstwy.

Przy tworzeniu instancji należy podać krotkę liczba oznaczającą kształt warstwy. Dodatkowo można przekazać nazwę funkcji aktywacji, którą domyślnie jest to sigmoid.

Zadanie funkcji „feedforward" to przyjęcie tablicy liczb z poprzedniej warstwy, obliczenie iloczynu skalarnego z aktualnymi wagami i poddanie wyjścia funkcją aktywacji. Następnie funkcja powinna rekurencyjnie wywołać samą siebie na następnej warstwie jeśli nie jest ostatnia w sieci. W przeciwnym przypadku zwraca wyliczone wyjście całej sieci.

Funkcja „calc_delta" wywoływana jest rekurencyjne, ale w przeciwnym kierunku. Oblicza ona różnicę pomiędzy spodziewanym wyjściem warstwy a aktualnym. Pozwoli to później skorygować wagi każdej warstwy.

Następna funkcja to „calc_gradient". Jest równnież wywoływana rekurencyjnie zaczynając od końca sieci. Oblicza wartość gradientu na podstawie wyjścia warstwy oraz wartości delty.

Ostatnią funkcją jest „update_weights". Jak jej nazwa wskazuje, to właśnie ona zmienia wartości wag w warstwach odejmując iloczyn obliczonego gradientu ze współczynnikiem uczenia. W miarę uczenia współczynnik uczenia może się zmieniać, dlatego uznałem, że to dobre miejsce na dostarczenie tego współczynnika warstwie sieci neuronowej.

5.2 Tworzenie architektury sieci

Listing 5.2: Schemat tworzenia sieci

```
def input_data(shape: Tuple[Optional[int], int]) -> Layer:
    ...


def fully_connected(incoming: Layer, n_units: int,
                    activation='sigmoid') -> Layer:
    ...


net = input_data(shape=(None, 30))
net = fully_connected(net, 24, activation='sigmoid')
net = fully_connected(net, 16, activation='sigmoid')
net = fully_connected(net, 12, activation='sigmoid')
net = fully_connected(net, 8, activation='sigmoid')
```

Funkcja „input_data" tworzy pierwszą warstwę sieci neuronowej. Jako argument przyjmuje krotkę z wymiarem danych, które będą przekazywane do sieci. Ustawia w instacji warstwy, flagę informującą, że jest to warstwa wejściowa i zwraca ją.

Druga ważna funkcja to „fully_ connected". Pierwszym argumentem, który należy jej podać to istniejąca już sieć neuronowa składająca się z połączonych rekurencyjnie warstw. „n_units" to parametr, który jest liczbą neuronów w nowo utworzonej warstwie. Na koniec można podać, jaką funkcję aktywacji ma mieć ta warstwa. Gdy nie zostanie podana, to domyślnie będzie to „sigmoid". Ta funkcja ustawia odpowiednie flagi w instancji warstwy i przypisuje wskaźniki do następników i poprzedników, jak w liście dwu kierunkowej.

Kilka kolejnych linii kodu to proces tworzenia całej sieci neuronowej. Ta sieć przyjmuje 30 cech i może przypisać każdy obiekt do 8 klas. Pomiędzy wejściem a wyjściem znajdują się 3 warstwy ukryte o rozmiarach 24, 16 i 12 neuronów odpowiednio. Taka sieć jest gotowa przyjęcia danych i do nauki klasyfikacji.

5.3 Schemat modelu

Listing 5.3: Schemat klasy Model

```
class Model:
    def __init__(self, network: Layer):
        ...

    def fit(self, X_inputs: np.ndarray, Y_targets: np.ndarray,
            validation_set: Tuple[np.ndarray, np.ndarray] = None,
            learning_rate=None, n_epoch=10,
            shuffle=False, train_file='train.json'):
        ...

    def predict(self, x: np.ndarray) -> np.ndarray:
        ...

    def predict_label(self, x: np.ndarray) -> int:
        ...

    def load(self, model_file: str):
        ...
```

```
    def save(self, model_file: str):
        ...
```

Klasa Model jest nakładką na sieć neuronową. To z niej użytkownik aplikacji bezpośrednio korzysta. Model w inicjalizerze przyjmuje jeden argument - zaprojektowaną sieć neuronową.

Funcja „fit" odpowiada za uczenienie sieci. Przyjmuje wiele argumentów w celu sprarametryzowania procesu.

- X_inputs - tablica ze wszystkimi danymi uczącymi bez przypisanych im klas,
- Y_targets - tablica zawierająca numery klas dla obiektów uczących,
- validation_set - krotka zawierająca dane testowe,
- learning_rate - krotka z 2 liczbami: współczynnikiem uczenia na początku i na końcu,
- n_epoch - liczba określająca liczbę epoch,
- shuffle - flaga informująca o tym, czy dane uczące mają być pomieszane,
- train_file - nazwa pliku w którym zapisywany jest postęp uczenia.

Metoda „fit" zbiera wszystkie możliwe dane z procesu uczenia, loguje je na konsolę i zapisuje do pliku. Dzięki temu możliwe jest przeprowadzenie badań i porównanie efektywności uczenia dla różnych wartości parametrów.

Funkcje „predict" i „predict_label" pozwalają skasyfikować wiele obiektów jednocześnie. Przyjmują tylko tablicę z wartościami cech. Jeśli potrzebne jest znać wsparcia wszystkich dla klas, to należy skorzystać z „predict", a jeśli wystarczy informacja o najbardziej prawdopodobnej klasie, to można skorzystać z „predict_label".

Ostatnie ważne funkcje to „save" i „load". Obie potrzebują na wejściu nazwę pliku. Funkcja „save" zapisuje aktualne wagi do tego pliku w formie JSONa, a „load" wczytuje je ustawia wagi we wszyskich warstwach. Ważne jest, aby pamiętać, że architektura sieci musi się zgadzać.

Rozdział 6

Przeprowadzone badania

Do badań nad wpływem różnych parametrów na jakość klasyfikacji stworzyłem 120 różnych modeli sieci neuronowej. Modele te uczyły się na serwerze, który posiadał 30 CPU o nieznanych parametrach, z których wykorzystałem tylko 24 CPU. Ponadto serwer miał do dyspozycji 100 GB pamięci RAM. Czas uczenia wyniósł około 68 minut.

Dodatkowo przeprowadziłem te same badania na innym serwerze posiadającym tylko 1 CPU i 4 GB pamięci RAM. Tutaj modelu uczyły się przez ponad 44 godziny.

Tablica 6.1: Parametry uczenia sieci

L.p.	**Parametr**	**Wartości**
1	architektura	1) 24, 16, 12, 8 2) 16, 12, 8 3) 16, 8 4) 8
2	funkcja aktywacji	1) sigmoid 2) tanh
3	liczba cech	1) 30 2) 20 3) 10
4	współczynnik uczenia	1) od 0.2 do 0.2 2) od 0.2 do 0.01 3) od 0.2 do 0.001 4) od 0.1 do 0.1 5) od 0.1 do 0.01

Parametr „architektura” informuje ile jest warstw ukrytych i ile zawierają one neuronów. Każda sieć na końcu ma 8 neuronów, ponieważ jest 8 możliwych klas.

Do badań użyłem różnej liczby cech. Są to 30, 20 lub 10 cech o najwyższej wartości chi2.

„Współczynnik uczenia” to dwie liczby, wartość na początku uczenia i na końcu. Jeśli jest powtórzona liczba, znaczy to, że był stały. W przeciwnym wypadku zmieniał się liniowo z każdą kolejną epoką.

Tablica 6.2: Testowane modele

L.p.	Nazwa	Architektura	Funkcja aktywacji	Liczba cech	Współczynnik uczenia
1	M_1	24_16_12_8	sigmoid	30	0.2_0.2
2	M_2	24_16_12_8	sigmoid	30	0.2_0.01
3	M_3	16_8	sigmoid	10	0.2_0.001
4	M_4	16_8	sigmoid	20	0.2_0.001
5	M_5	16_8	sigmoid	30	0.2_0.001
6	M_6	8	sigmoid	30	0.1_0.01
7	M_7	16_8	sigmoid	30	0.1_0.01
8	M_8	16_12_8	sigmoid	30	0.1_0.01
9	M_9	24_16_12_8	sigmoid	30	0.1_0.01
10	M_{10}	24_16_12_8	tanh	10	0.1_0.01
11	M_{11}	24_16_12_8	tanh	20	0.1_0.01
12	M_{12}	24_16_12_8	tanh	30	0.1_0.01

6.1 Różne współczynniki uczenia

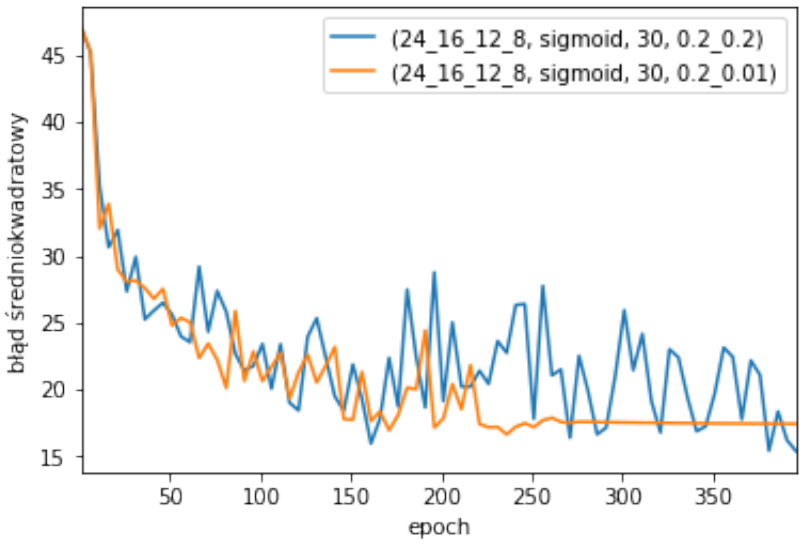
Poniżej znajduje się tabela z przebiegu uczenia dwóch modeli o architekturze 24_16_12_8, funkcji aktywacji sigmoid, uczonym na 30 cechach i różniącym się współczynnikiem uczenia.

Tablica 6.3: Porównianie modeli M_1 i M_2

	M_1		M_2	
Epoka	**Dokładność**	**Błąd**	**Dokładność**	**Błąd**
1	0.277	47.031	0.277	47.031
10	0.513	37.573	0.563	33.302
20	0.588	32.163	0.622	29.612
30	0.622	27.482	0.597	30.062
40	0.647	26.037	0.647	27.523
50	0.655	27.505	0.655	27.221
60	0.655	31.413	0.664	25.058
70	0.655	29.571	0.723	22.468
80	0.697	22.540	0.689	22.567
90	0.714	22.712	0.731	22.202
100	0.639	33.643	0.706	23.672
110	0.672	30.809	0.756	21.424
120	0.765	20.840	0.723	25.234
130	0.773	19.664	0.714	23.904
140	0.748	21.401	0.731	21.937
150	0.739	22.939	0.739	20.739
160	0.798	18.715	0.824	17.646
170	0.748	20.608	0.790	17.594
180	0.807	19.054	0.773	20.165
190	0.790	20.590	0.824	16.881
200	0.849	16.287	0.798	19.377
210	0.798	16.966	0.815	18.842
220	0.782	21.849	0.815	17.510
230	0.765	20.966	0.824	17.073
240	0.756	22.650	0.832	17.168
250	0.849	16.440	0.815	17.868
260	0.807	17.105	0.815	18.350
270	0.824	16.844	0.824	17.483
280	0.832	16.893	0.832	17.576
290	0.815	17.927	0.832	17.544
300	0.773	21.567	0.832	17.523
310	0.655	35.103	0.832	17.505
320	0.798	17.729	0.832	17.489
330	0.857	13.654	0.832	17.475
340	0.790	20.248	0.832	17.462
350	0.790	19.990	0.832	17.450
360	0.782	19.722	0.832	17.441
370	0.739	24.702	0.832	17.433
380	0.824	16.167	0.832	17.427
390	0.815	18.179	0.832	17.423
400	0.807	16.703	0.832	17.420

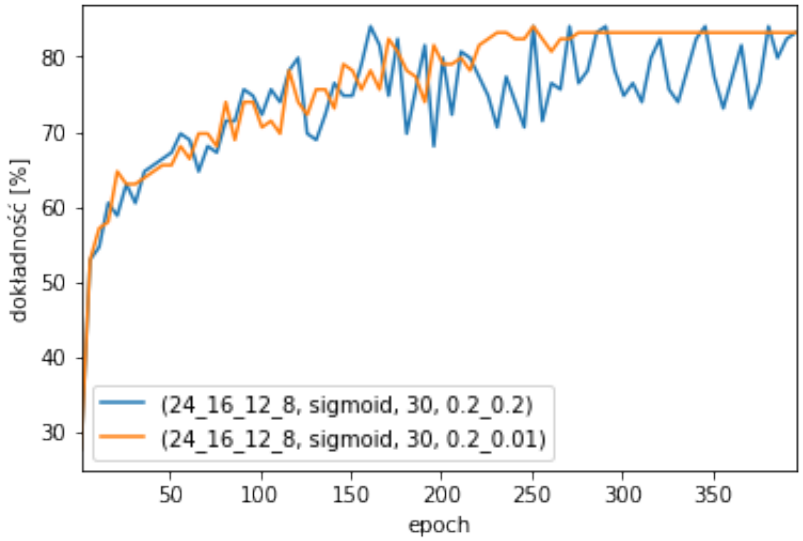
Wykres wartości błędu przedstawia błąd średniokwadratowy z procesu uczenia co 5 epokę. Trend funkcji jest mniej więcej podobny, ale widać, że dla przypadku ze stałym współczynnikiem uczenia wartość błędu spada mniej gwałtownie. Gdy zmniejsza się współczynnik uczenia trening przebiega bardzo agresywnie, czasem daje chwilowo lepsze wyniki, ale przez większość epok błąd jet większy.

Dla wykresu porównianiа dokładności można wysnuć podobne wnioski, że lepiej jest zmniejszać współczynnik uczenia w trakcie nauki. Dzieje się tak dlatego, że po wielu epokach, gdy współczynnik uczenia dalej jest relatywnie duży gradient pomija minimum i zaczyna rosnąć. Oba modele osiągnęły wysoką dokładność



Rysunek 6.1: Porównanie wartości błędu modeli M_1 i M_2

ponad 80%.



Rysunek 6.2: Porównanie dokładności modeli M_1 i M_2

6.2 Różna liczba cech

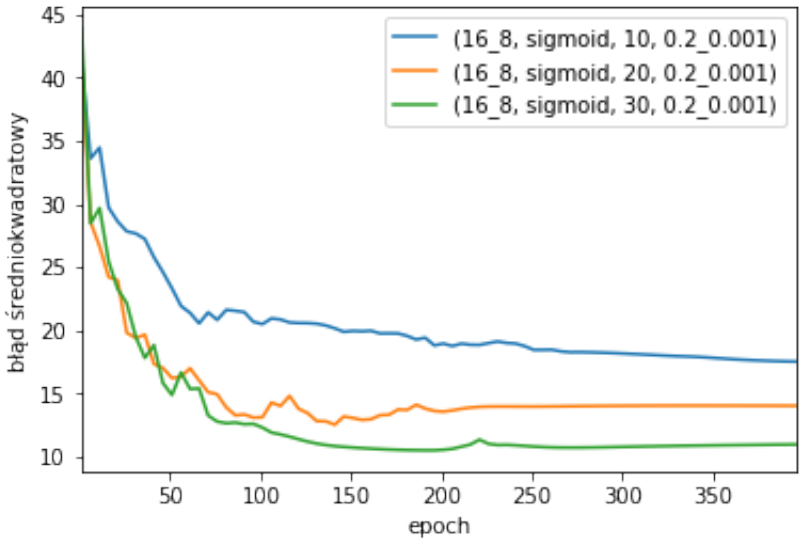
Drugie porównanie to modele o architekturze 16_8, funkcji aktywacji sigmoid. Trenowane były z malejącym współczynnikiem uczenia. Parametr, który je różni, to liczba cech. M_3 klasyfikuje korzystając z 10 cech o najwyższym współczynniku chi2. M_4 dostaje 20 cech, a M_5 uczy się na prawie wszystkich cechach, czyli 30 najlepszych.

Tablica 6.4: Porównianie modeli M_3, M_4 i M_5

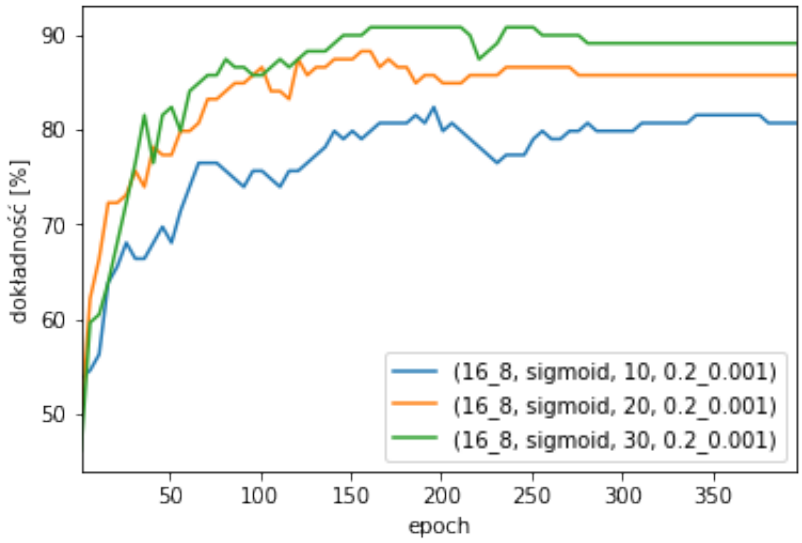
	M_3		M_4		M_5	
Epoka	**Dokładność**	**Błąd**	**Dokładność**	**Błąd**	**Dokładność**	**Błąd**
1	0.538	41.299	0.496	41.714	0.462	43.907
10	0.571	34.109	0.613	26.876	0.613	28.462
20	0.647	29.445	0.714	24.232	0.655	23.769
30	0.664	27.188	0.723	20.451	0.765	20.308
40	0.672	26.045	0.782	17.584	0.765	18.426
50	0.681	23.634	0.782	16.687	0.824	15.077
60	0.739	21.233	0.790	16.759	0.849	14.902
70	0.773	20.575	0.832	15.302	0.849	13.756
80	0.765	22.901	0.849	14.367	0.874	12.665
90	0.739	21.739	0.849	13.351	0.866	12.680
100	0.765	21.240	0.857	13.708	0.857	12.414
110	0.739	20.959	0.832	14.686	0.874	11.797
120	0.756	20.595	0.866	13.803	0.874	11.426
130	0.773	20.562	0.874	12.516	0.882	11.068
140	0.798	20.210	0.874	12.519	0.891	10.854
150	0.798	19.903	0.866	13.173	0.899	10.735
160	0.798	19.924	0.882	12.943	0.908	10.647
170	0.807	19.784	0.866	13.364	0.908	10.576
180	0.807	19.581	0.849	13.898	0.908	10.529
190	0.815	19.234	0.849	14.285	0.908	10.507
200	0.807	18.782	0.849	13.564	0.908	10.536
210	0.798	18.772	0.849	13.768	0.908	10.761
220	0.782	18.848	0.857	13.938	0.874	11.259
230	0.765	19.123	0.857	13.972	0.882	10.923
240	0.773	18.967	0.866	13.971	0.908	10.914
250	0.790	18.496	0.866	13.969	0.908	10.813
260	0.790	18.480	0.866	13.976	0.899	10.745
270	0.798	18.272	0.866	13.991	0.899	10.722
280	0.807	18.269	0.857	14.007	0.891	10.729
290	0.798	18.229	0.857	14.019	0.891	10.752
300	0.798	18.170	0.857	14.027	0.891	10.780
310	0.807	18.101	0.857	14.034	0.891	10.808
320	0.807	18.028	0.857	14.040	0.891	10.834
330	0.807	17.961	0.857	14.043	0.891	10.858
340	0.815	17.909	0.857	14.043	0.891	10.880
350	0.815	17.824	0.857	14.042	0.891	10.899
360	0.815	17.725	0.857	14.040	0.891	10.917
370	0.815	17.647	0.857	14.038	0.891	10.933
380	0.807	17.589	0.857	14.035	0.891	10.948
390	0.807	17.545	0.857	14.032	0.891	10.962
400	0.807	17.523	0.857	14.032	0.891	10.972

Z wykresu porównania wartości błędu uczenia tych modeli można wywnioskować, że liczba cech ma wpływ na jakość klasyfikacji. Im więcej cech, tym model popełnia mniejszy błąd. Od dwusetnej epoki błąd maleje bardzo wolno, dlatego można by przerwać wcześniej uczenie bez straty na jakości klasyfikacji.

Wykres dokładności od epoki potwierdza, że dla tego zbioru danych warto użyć większej liczby cech, kosztem większego i bardziej skomplikowanego modelu. Tak jak na poprzednim wykresie, tutaj też po dwusetnej nie widać żadnej poprawy, a nawet obniżenie jakości predykcji może wskazywać na przeuczenie klasyfikatora. To znaczy, że za bardzo dostosował się do danych uczących tracąc zdolność do uogólniania.



Rysunek 6.3: Porównanie wartości błędu modeli M_3, M_4 i M_5



Rysunek 6.4: Porównanie dokładności modeli M_3, M_4 i M_5

6.3 Różne architektury

Kolejnej analizie poddano sieci neuronowe mające funkcję aktywacji sigmoid oraz 30 neuronów w warstwie wejściowej. Współczynnik uczenia malał od 1.0 do 0.01 we wszystkich modelach tak samo. Kolejne klasyfikatory miały następujące architektury 8, 16_8, 16_12_8 i 24_16_12_8.

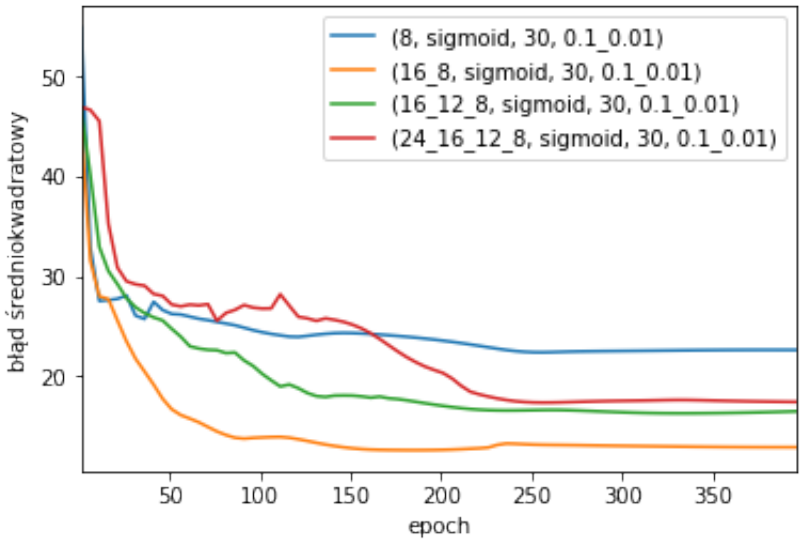
Wydawać by się mogło, że im więcej warstw ukrytych, tym model dokonuje klasyfikacji z wyższą dokładnością.

Tablica 6.5: Porównianie modeli M_6, M_7, M_8 i M_9

	M_6		M_7		M_8		M_9	
Epoka	**Dokładność**	**Błąd**	**Dokładność**	**Błąd**	**Dokładność**	**Błąd**	**Dokładność**	**Błąd**
1	0.361	54.950	0.420	45.093	0.387	46.705	0.387	46.968
10	0.630	28.059	0.605	28.339	0.563	33.505	0.521	46.049
20	0.647	28.050	0.655	26.151	0.630	29.422	0.580	31.339
30	0.697	26.420	0.723	22.149	0.647	27.208	0.630	29.242
40	0.681	27.203	0.756	19.402	0.672	25.992	0.613	28.203
50	0.697	26.257	0.824	16.846	0.697	24.958	0.630	27.234
60	0.697	26.023	0.832	15.827	0.681	23.236	0.647	27.873
70	0.706	25.647	0.832	15.026	0.681	22.635	0.639	27.101
80	0.714	25.314	0.840	14.182	0.714	22.061	0.647	26.490
90	0.723	24.919	0.840	13.757	0.748	21.656	0.655	27.194
100	0.731	24.451	0.840	13.870	0.756	20.421	0.664	26.650
110	0.723	24.131	0.849	13.930	0.773	19.475	0.664	26.430
120	0.731	23.937	0.840	13.729	0.748	18.962	0.664	26.588
130	0.723	24.139	0.840	13.372	0.765	18.062	0.681	25.698
140	0.714	24.308	0.840	13.044	0.756	18.048	0.672	25.671
150	0.714	24.331	0.840	12.807	0.756	18.073	0.655	25.225
160	0.723	24.246	0.849	12.668	0.782	17.879	0.681	24.412
170	0.731	24.122	0.849	12.622	0.782	17.810	0.723	23.230
180	0.739	23.974	0.849	12.606	0.790	17.619	0.739	22.015
190	0.739	23.798	0.849	12.606	0.807	17.330	0.748	21.055
200	0.739	23.590	0.857	12.630	0.815	17.058	0.765	20.394
210	0.748	23.349	0.857	12.693	0.815	16.827	0.782	19.212
220	0.748	23.076	0.857	12.782	0.815	16.665	0.790	18.236
230	0.748	22.790	0.857	13.083	0.807	16.587	0.798	17.815
240	0.748	22.544	0.849	13.245	0.807	16.578	0.798	17.523
250	0.765	22.413	0.849	13.167	0.815	16.604	0.798	17.386
260	0.748	22.411	0.857	13.130	0.815	16.627	0.798	17.369
270	0.731	22.462	0.849	13.109	0.824	16.603	0.798	17.412
280	0.731	22.496	0.840	13.083	0.815	16.538	0.798	17.466
290	0.739	22.517	0.840	13.056	0.815	16.464	0.807	17.510
300	0.748	22.535	0.840	13.031	0.815	16.395	0.807	17.540
310	0.748	22.553	0.840	13.008	0.815	16.340	0.790	17.565
320	0.739	22.572	0.849	12.986	0.815	16.305	0.790	17.598
330	0.739	22.591	0.849	12.966	0.815	16.291	0.790	17.628
340	0.731	22.608	0.849	12.947	0.815	16.294	0.790	17.612
350	0.731	22.622	0.857	12.930	0.815	16.310	0.790	17.569
360	0.731	22.633	0.857	12.916	0.815	16.335	0.790	17.530
370	0.731	22.639	0.857	12.906	0.815	16.367	0.790	17.497
380	0.731	22.641	0.857	12.900	0.815	16.407	0.790	17.469
390	0.731	22.640	0.857	12.897	0.815	16.454	0.790	17.447
400	0.731	22.636	0.866	12.898	0.815	16.509	0.790	17.429

Wykres wartości błędu pokazuje, że bardziej skomplikowana architekrura wcale nie zapewnia lepszych wyników. Największy model M_9 przez prawie połowę okresu uczenia radził sobie najgorzej, a na koniec wyprzedził najmniejszy model M_6.

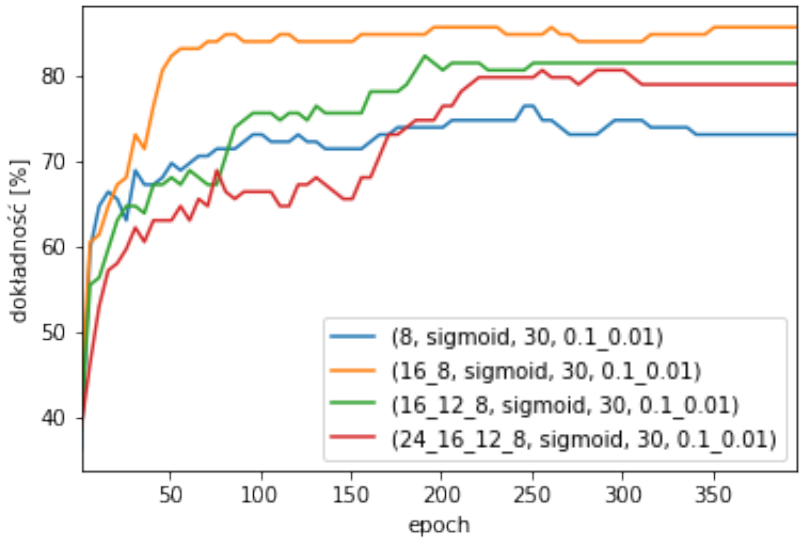
Najlepsze wyniki osiągnięto z jedną warstwą ukrytą w modelu M_7, potem z dwoma warstwami ukrytymi M_8.



Rysunek 6.5: Porównanie wartości błędu modeli M_6, M_7, M_8 i M_9

Zależność dokładności od epoki nie zmienia sytuacji. Model bez warstw ukrytych osiągnął ponad 73% dokładności, co było zaskoczeniem.

Ponownie można było przerwać uczenie w okolicy epoki numer 200, gdyż dalsze trenowanie nie poprawiało rezultatów.



Rysunek 6.6: Porównanie dokładności modeli M_6, M_7, M_8 i M_9

6.4 Różna liczba sech - tanh

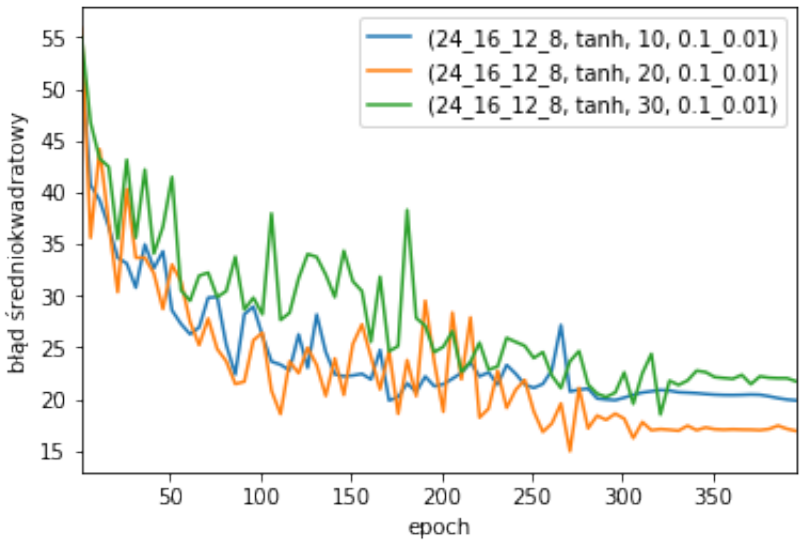
W drugiej części przeanalizowano modele z inną funkcją aktywacji - tanh.

Porównano sieci o architekturze 24_16_12_8 z malejącym współczynnikiem uczenia od 0.1 do 0.01. Klasyfikatory różniły się ponownie liczbą cech.

Tablica 6.6: Porównianie modeli M_{10}, M_{11} i M_{12}

	M_{10}		M_{11}		M_{12}	
Epoka	**Dokładność**	**Błąd**	**Dokładność**	**Błąd**	**Dokładność**	**Błąd**
1	0.353	52.640	0.277	55.873	0.118	55.034
10	0.538	39.613	0.487	39.530	0.345	52.538
20	0.571	33.717	0.580	33.959	0.555	36.910
30	0.605	33.379	0.597	32.794	0.664	32.991
40	0.664	33.391	0.672	28.346	0.580	35.834
50	0.613	35.795	0.630	30.813	0.630	33.222
60	0.689	30.368	0.655	27.470	0.655	30.992
70	0.647	30.664	0.756	24.646	0.605	34.815
80	0.723	25.034	0.664	27.952	0.622	35.889
90	0.613	34.623	0.756	20.232	0.622	32.761
100	0.689	25.934	0.748	24.249	0.672	27.784
110	0.790	20.253	0.748	20.231	0.639	35.127
120	0.773	24.424	0.756	22.806	0.647	31.005
130	0.706	26.605	0.765	20.004	0.538	35.759
140	0.739	24.553	0.697	25.398	0.664	31.898
150	0.790	20.773	0.697	27.043	0.714	25.219
160	0.639	29.552	0.714	24.707	0.714	25.421
170	0.790	20.973	0.731	22.933	0.706	27.352
180	0.807	20.067	0.723	24.532	0.714	25.905
190	0.782	21.582	0.765	23.374	0.647	37.363
200	0.756	24.949	0.773	20.865	0.739	23.828
210	0.773	20.389	0.790	17.957	0.756	24.335
220	0.798	22.621	0.739	23.427	0.739	23.932
230	0.773	24.285	0.773	21.307	0.765	23.108
240	0.790	22.683	0.798	20.925	0.731	25.638
250	0.765	22.440	0.773	21.445	0.756	22.974
260	0.798	23.804	0.790	17.580	0.723	24.399
270	0.782	21.929	0.832	16.134	0.790	22.530
280	0.798	21.549	0.832	16.583	0.790	21.926
290	0.798	20.037	0.815	17.821	0.773	21.182
300	0.807	20.206	0.815	17.117	0.782	22.278
310	0.798	20.617	0.815	16.688	0.765	22.705
320	0.807	20.875	0.824	17.641	0.765	25.037
330	0.815	20.677	0.824	17.042	0.815	19.822
340	0.807	20.595	0.824	17.087	0.798	22.790
350	0.807	20.455	0.824	17.412	0.824	20.356
360	0.807	20.403	0.824	17.062	0.807	21.937
370	0.807	20.450	0.832	17.048	0.824	18.904
380	0.807	20.319	0.832	17.427	0.815	21.773
390	0.807	19.992	0.832	17.430	0.815	21.767
400	0.815	19.859	0.824	16.917	0.815	21.706

Wykres wartości błędu pokazuje niespodziewanie, że ucząc model na 20 cechach otrzymano najmniejszy

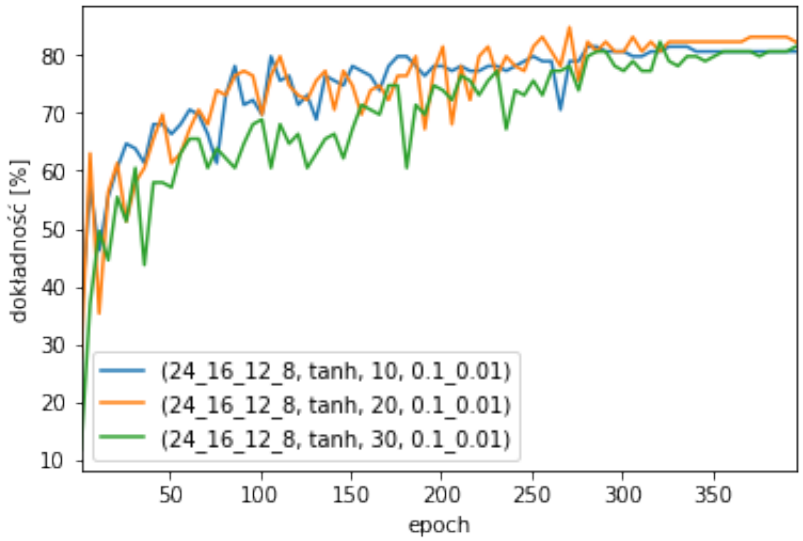


Rysunek 6.7: Porównanie wartości błędu modeli M_{10}, M_{11} i M_{12}

błąd. 10 cech daje również lepszy wynik niż 30, choć tu różnica jest mniejsza.

Wykres dokładności nie potwierdza wyraźnie tego wniosku, ale i tutaj model M_{11} radzi sobie nieznacznie lepiej niż inne.

Wszystkie 3 modele przekroczyły nieznacznie próg 80%.

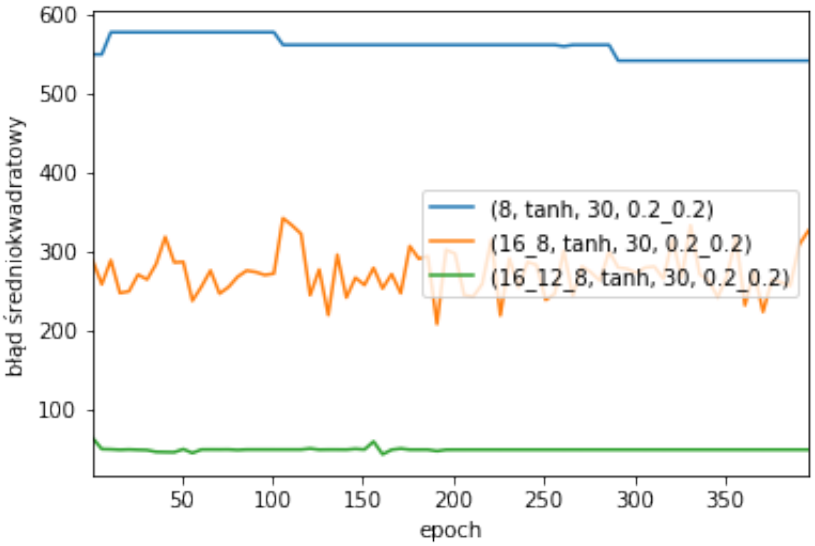


Rysunek 6.8: Porównanie dokładności modeli M_{10}, M_{11} i M_{12}

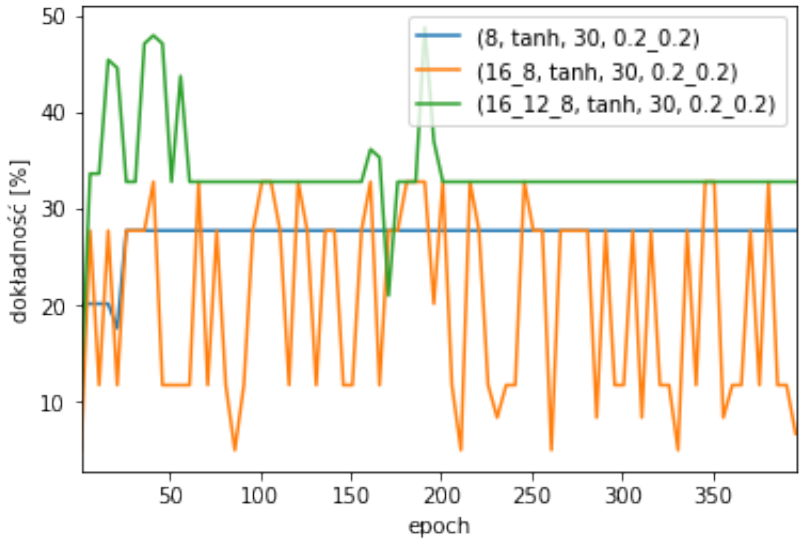
6.5 Różne architektury - tanh

Tablica 6.7: Porównianie modeli M_{13}, M_{14} i M_{15}

	M_{13}		M_{14}		M_{15}	
Epoka	**Dokładność**	**Błąd**	**Dokładność**	**Błąd**	**Dokładność**	**Błąd**
1	0.202	548.499	0.050	287.852	0.118	63.359
10	0.202	576.500	0.328	237.090	0.319	48.922
20	0.193	576.500	0.118	277.368	0.294	48.806
30	0.277	576.500	0.118	220.278	0.328	48.924
40	0.277	576.500	0.118	271.187	0.328	49.436
50	0.277	576.500	0.277	268.174	0.538	47.290
60	0.277	576.500	0.328	259.154	0.353	50.235
70	0.277	576.500	0.328	292.961	0.328	48.984
80	0.277	576.500	0.277	259.343	0.395	48.063
90	0.277	576.500	0.277	247.395	0.328	48.988
100	0.277	576.499	0.277	278.024	0.328	48.974
110	0.277	560.500	0.118	293.126	0.328	48.972
120	0.277	560.500	0.118	318.174	0.328	47.771
130	0.277	560.500	0.118	317.897	0.328	48.945
140	0.277	560.500	0.328	246.111	0.328	48.834
150	0.277	560.500	0.328	293.770	0.328	48.849
160	0.277	560.500	0.328	276.355	0.521	45.437
170	0.277	560.500	0.067	308.603	0.303	51.606
180	0.277	560.500	0.328	282.134	0.328	48.819
190	0.277	560.500	0.118	269.000	0.353	48.910
200	0.277	560.500	0.277	252.517	0.328	48.791
210	0.277	560.500	0.067	311.607	0.328	48.796
220	0.277	560.499	0.084	284.886	0.328	48.796
230	0.277	560.499	0.277	275.749	0.328	48.795
240	0.277	560.499	0.118	308.011	0.328	48.795
250	0.277	560.498	0.328	265.692	0.328	48.796
260	0.277	560.466	0.328	257.700	0.328	48.798
270	0.277	560.500	0.328	282.665	0.328	48.801
280	0.277	560.500	0.118	259.123	0.328	48.801
290	0.277	540.500	0.118	235.230	0.328	48.802
300	0.277	540.500	0.118	266.620	0.328	48.800
310	0.277	540.500	0.050	276.539	0.328	48.798
320	0.277	540.500	0.328	271.965	0.328	48.796
330	0.277	540.500	0.328	253.050	0.328	48.794
340	0.277	540.500	0.118	221.669	0.328	48.794
350	0.277	540.499	0.118	216.252	0.328	48.797
360	0.277	540.499	0.328	223.183	0.328	48.799
370	0.277	540.499	0.118	250.361	0.328	48.802
380	0.277	540.499	0.118	259.041	0.328	48.803
390	0.277	540.499	0.277	256.229	0.328	48.802
400	0.277	540.499	0.118	260.246	0.328	48.800



Rysunek 6.9: Porównanie wartości błędu modeli M_{13}, M_{14} i M_{15}



Rysunek 6.10: Porównanie dokładności modeli M_{13}, M_{14} i M_{15}

Rozdział 7

Podsumowanie

7.1 Dalsze możliwości rozwoju

7.2 Co mogłem zrobić lepiej

Tekst podsumowania

Spis rysunków

2.1 Histogram występowania chorób 7

3.1 Logo języka Python 9
3.2 Logo NumPy 10
3.3 Logo matplotlib 10
3.4 Logo pandas 11
3.5 Logo Gita 11
3.6 Logo Dockera 12

4.1 Schemat neuronu 13
4.2 Funkcja aktywacji - sigmoid 14
4.3 Funkcja aktywacji - tanh 15
4.4 Schemat neuronu 15

6.1 Porównanie wartości błędu modeli M_1 i M_2 23
6.2 Porównanie dokładności modeli M_1 i M_2 23
6.3 Porównanie wartości błędu modeli M_3, M_4 i M_5 25
6.4 Porównanie dokładności modeli M_3, M_4 i M_5 25
6.5 Porównanie wartości błędu modeli M_6, M_7, M_8 i M_9 27
6.6 Porównanie dokładności modeli M_6, M_7, M_8 i M_9 27
6.7 Porównanie wartości błędu modeli M_{10}, M_{11} i M_{12} 29
6.8 Porównanie dokładności modeli M_{10}, M_{11} i M_{12} 29
6.9 Porównanie wartości błędu modeli M_{13}, M_{14} i M_{15} 31
6.10 Porównanie dokładności modeli M_{13}, M_{14} i M_{15} 31

Spis tablic

2.1 Wszystkie cechy z odpowiedziami 3
2.2 Wartości chi2 dla wszystkich cech 8

6.1 Parametry uczenia sieci 20
6.2 Testowane modele 21
6.3 Porównianie modeli M_1 i M_2 22
6.4 Porównianie modeli M_3, M_4 i M_5 24
6.5 Porównianie modeli M_6, M_7, M_8 i M_9 26
6.6 Porównianie modeli M_{10}, M_{11} i M_{12} 28
6.7 Porównianie modeli M_{13}, M_{14} i M_{15} 30