Akademia Górniczo-Hutnicza Im. Stanisława Staszica w Krakowie

Wydział Zarządzania

Projekt Sztuczne Sieci Neuronowe

Przewidywanie czy dana osoba dostanie udaru mózgu

Autorzy : Mateusz Mulka, Tomasz Solarski

Kierunek Studiów : Informatyka i Ekonometria rok 2

Spis treści

Cel projektu i problemu

Udar mózgu jest zespołem objawów klinicznych i jest on jedną z głównych przyczyn niepełnosprawności wśród osób dorosłych i ludzi starszych, a więc znacznej części społeczeństwa. Oprócz tego on powodować liczne społeczne i ekonomiczne problemy, a nawet śmierć. Z tego powodu bardzo ważna jest możliwość jego zapobiegania, a więc też wykrywania potencjalnego zagrożenia wystąpienia udaru. Można tego dokonać między innymi analizując parametry krwi oraz stan serca.

Za nasz główny cel w tym projekcie obraliśmy stworzenie sieci neuronowej, która z jak największą skutecznością na podstawie dostępnych danych będzie w stanie określić czy dana osoba jest narażona na wystąpienie udaru mózgu. W określeniu stopnia odniesionego przez nas sukcesu pomoże nam porównanie wyniku dokładności naszej sieci neuronowej (accuracy) do wyników jakie odniosły inne osoby w przyszłości. W tym celu będziemy posiłkować się literaturą opisaną poniżej. W praktyce tego typu sieci neuronowe przy dostatecznie dużym poziomie accuracy dla nowych danych mogą w istotnym stopniu ułatwić pracę lekarzy i pomóc im w ratowaniu życia i zdrowia pacjentów.

Opis danych

W projekcie będziemy pracowali na danych pochodzących ze strony Kaggle.com. Zbiór danych znajduje się pod tym linkiem: https://www.kaggle.com/datasets/fedesoriano/stroke-prediction-dataset. Dane składają się z 5110 wierszy, a każdy wiersz reprezentuje 12 wartości dla badanej osoby. Te wartości to:

- **id** – identyfikator
- **gender** – płeć
- **age** – wiek
- **hypertension** - czy osoba posiada nadciśnienie
- **heart_disease** - czy osoba posiada jakąś chorobę serca
- **ever_married** – czy osoba była kiedyś w związku małżenskim
- **work_type** – typ wykonywanej pracy
- **Residence_type** – czy osoba mieszka na wsi czy w mieście
- **avg_glucose_level** – poziom cukru we krwi
- **bmi** - wskaźnik masy ciała (BMI)
- **smoking_status** – status osoby palącej
- **stroke** - czy osoba dostała udaru mózgu czy nie

Zmienne kategoryczne to: gender, ever_married, work_type, Residence_type, smoking_status.

Zmienne binarne to: hypertension, heart_disease, stroke.

Zmienne ciągłe to: age, avg_glucose_level, bmi.

Przegląd literatury

1) W badaniu przeprowadzonym przez Soumyabrata Dev, Hawei Wang, Chidozie Shamrock Nwosu, Nishtha Jain, Bhradwaj Veeravalli, Deepu John w przeprowadzonym przez siebie badaniu pracowali na danych prosto ze szpitali. Na ich podstawie udało im się pokazać korelację między udarem, a takimi czynnikami jak między innymi płeć, wiek, bmi, status palacza, średni poziom glukozy czy przeszłe choroby serca. Opisywane dane były więc niezmiernie podobne do tych, na których my pracowaliśmy.W swojej pracy skupili się oni głównie na wskazaniu istotności poszczególnych czynników na wystąpienie udaru jednak przedstawili też wyniki accuracy. Uzyskana przez nich dokładność wynosiła 78% przy loss rate równym 19%. Jest to wynik trochę lepszy od naszego, prawdopodobnie stoi za tym stwierdzenie korelacji poszczególnych zmiennych na wynik końcowy i na tej podstawie edycja danych które dostaje model.

2) Thia Tazin, Md Nur Alam, Nahian Nakiba Dola i Mohammad Sajibul Bari z North South University w Bashundara w Bangladeszu oraz Sami Bourouis i Mohammad Monirujjaman Khan z Taif University w Taif w Arabii Saudyjskiej za pomocą kilku metod uczenia algorytmu podjęli się opracowania modelu przewidującego udar mózgu. Jako algorytmy wybralki Random Forest, Logistic Regression, Decision Tree oraz Votin Classifier. Dzięki złożoności wybranych przez nich algorytmów oraz bazowania na F1-score, który omija problem nierównego rozłożenia danych udało im się uzyskać między 73% , a 95% zależnie od obranego algorytmu.

3) W badaniu przeprowadzonym przez Koreanski Instytut Badawczy Elektroniki i Telekomunikacji (ETRI) oraz Uniwersytet Youngsan w Korei przez Jeahak Yu, Sejin Park, Soon-Hyun Kwon, Chee Meng Benjamin Ho, Cheol-Sig Pyo oraz Hansung Lee, analizującym podobny problem jednak z wykorzystaniem sygnałów elektromiograficznych w czasie rzeczywistym użyto dwóch oddzielnych kategorii AI. Pierwszą z nich była metoda lasu losowego, czyli jedna z metod uczenia maszynowego, dzięki której udało się im uzyskać dokładność na poziomie 90.38%. Druga metoda wykorzystywała metodę uczenia głębokiego o nazwie długotrwała pamięć krótkotrwała (LSTM) dzięki której udało im się uzyskać aż 98.958% skuteczności. Są to wyniki zdecydowanie lepsze od naszego, prawdopodobnie może wynikać to ze sposobu doboru metod ale przede wszystkim z dostępu do dużej ilości danych. Nasz wynik finalny, z powodów opisanych w sprawozdaniu, bazuje na próbce jedynie 418 rekordów co zdecydowanie wpływa na dokładność całego modelu.

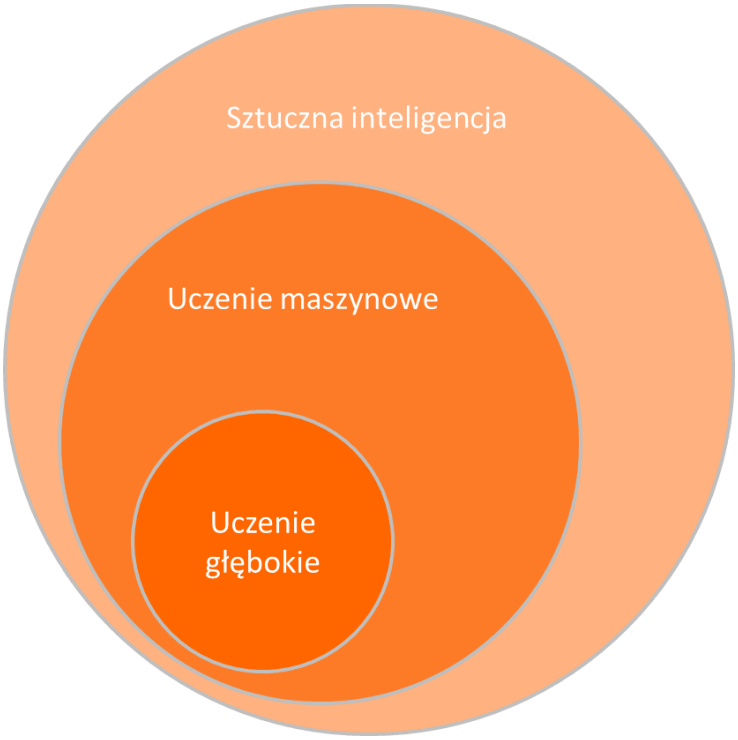
Wstęp teoretyczny

Historia sieci neuronowych

Początki sieci neuronowych sięgają lat 40. XX wieku, kiedy to opracowano model neuronu, który potrafił rozpoznawać jedynie dwie kategorie obiektów, na podstawie wag zadanych przez operatora. Dopiero pod koniec lat 50. dalszy rozwój tej dziedziny zaowocował zbudowaniem pierwszej sieci neuronowej zwanej perceptronem, której zadaniem było rozpoznawanie znaków. Po pierwszej fali zainteresowania nastąpił okres stagnacji i krytycyzmu, który trwał do czasu wprowadzenia algorytmu wstecznej propagacji błędów w 1986r. Przełom nastąpił w 2006r. gdy zostały opracowane sieci głębokie z efektywnymi metodami ich uczenia. Bardzo przyczynił się do tego wzrost mocy obliczeniowej procesorów.[1]

Deep learning – uczenie głębokie

Uczenie głębokie jest uczeniem maszynowym, które opiera się na sztucznych sieciach neuronowych. W uczeniu głębokim proces uczenia nie wymaga kontroli człowieka, zachodzi w sposób nienadzorowany. Zadaniem oprogramowania jest naśladowanie i symulowanie zachowania ludzkiego umysłu.[2] Deep Learning polega na tworzeniu sieci neuronowych, które zamiast organizować dane i wykonywać szereg zdefiniowanych równań, ustalają podstawowe parametry danych, rozpoznają wzorce za pomocą wielu warstw przetwarzania.[3]

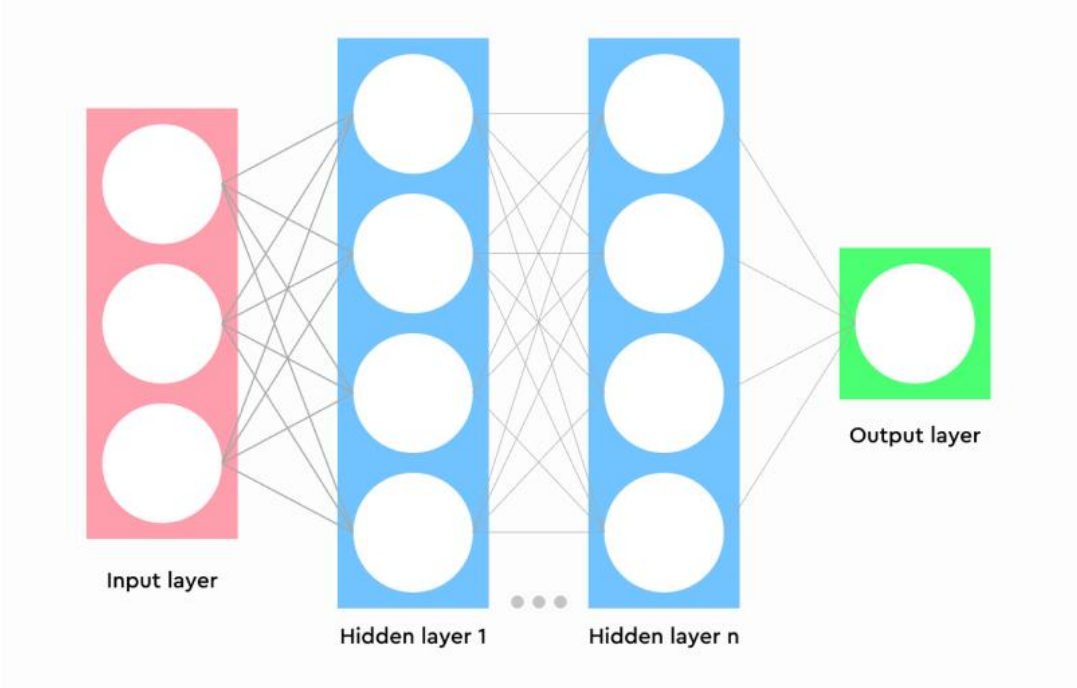


Rysunek 1: Schemat podziału sztucznej inteligencji

[1] https://bulldogjob.pl/readme/czym-jest-deep-learning-i-sieci-neuronowe
[2] https://thestory.is/pl/journal/co-to-jest-deep-learning/
[3] https://bulldogjob.pl/readme/czym-jest-deep-learning-i-sieci-neuronowe

Sieci neuronowe

Sieci neuronowe w swojej architekturze przypominają ludzki mózg i składają się ze sztucznych neuronów. Najczęściej spotykanym rodzajem sieci jest sieć wielowarstwowa, w której neurony grupowane są w warstwy. Jej cechą charakterystyczną jest posiadanie przynajmniej jednej warstwy ukrytej, pośredniczącej w komunikacji między danymi na wejściu i wyjściu. W ramach jednej warstwy neurony nie komunikują się między sobą, natomiast pomiędzy warstwami obowiązuje zasada „każdy z każdym". Neuron jest prostym elementem obliczeniowym, do którego doprowadzane są sygnały z wejść sieci lub neuronów poprzedniej warstwy. Każdy sygnał, czyli zmienna wejściowa mnożony jest przez odpowiadającą mu wagę, a te wagi zmieniane są w trakcie procesu uczenia. Następnie zważona suma cech wejściowych przekazywana jest jako argument do funkcji aktywacji. Funkcja aktywacji to funkcja, według, której obliczana jest wartość wyjścia neuronów sieci neuronowej. Przykładami funkcji aktywacji są: funkcja liniowa, funkcja sigmoidalna, funkcja skoku jednostkowego, funkcja Gaussa. Wybór tej funkcji zależy od rodzaju problemu jaki zadajemy sieci do rozwiązania.[4]



Rysunek 2: Budowa sieci neuronowej
Źródło: https://thestory.is/pl/journal/co-to-jest-deep-learning/

[4] https://pl.wikipedia.org/wiki/Funkcja_aktywacji

Część praktyczna – tworzenie modelu i analiza

Opis modelu

1) **Tworzenie modelu –** tworzymy model deklaracją sieci - sekwencyjną, pozwala nam to na tworzenie modelu warstwa po warstwie. Jest odpowiedni dla zwykłego stosu warstw, w którym każda warstwa ma dokładnie jeden tensor wejściowy i jeden tensor wyjściowy. W tym przypadku do modelu dodajemy 5 warstw, pierwsza z nich to warstwa wejściowa, kolejne 3 to warstwy ukryte z funkcją aktywacji relu, a piąta z nich to warstwa wyjściowa, gdzie funkcją aktywacji jest sigmoid, ponieważ daje się ja w ostatnie wartswie dla problemów binarnych.
2) **Kompilacja modelu** – ma za zadanie skonfigurować model dla treningu.

- Dobieramy **optymalizator** sieci - **ADAM** - Adaptive Moment Estimation, który jest najczęściej stosowany i najbardziej efektywny dla większości zastosowań. ADAM jest to kombinacja dwóch metod gradientu prostego (Momentum oraz Root Mean Square Propagation), czyli algorytmu, który ma na celu znalezienie minimum lokalnego zadanej funkcji celu.
- Dobieramy funkcję straty (funkcję błędu) - w ramach optymalizacji należy wielokrotnie estymować bląd dla aktualnego stanu modelu. Używa się jej do oszacowania straty modelu, tak żeby można było zaktualizować wagi w celu zmniejszenia straty podczas następnej oceny. Nasz model dotyczyczy binarnej klasyfikacji więc wybraliśmy Binary Cross-Entropy Loss. Cross-entropy to jest typowa funkcja do problemów binarnych. Zwykle stosuje się ją zawsze na początku i zmienia jeśli się ma do tego jakiś powód.

3) **Trenowanie modelu –** trenujemy model, ustawiamy parametr epoch(ilość przejść jaką przechodzimy przez nasz trening set)

Przygotowanie danych

Unikalne wartości dla danych kategorycznych:

	gender
0	Male
1	Female
2	Other

	ever_married
0	Yes
1	No

	Residence_type
0	Urban
1	Rural

	work_type
0	Private
1	Self-employed
2	Govt_job
3	children
4	Never_worked

	smoking_status
0	formerly smoked
1	never smoked
2	smokes
3	Unknown

Sprawdzenie czy występują braki w danych.

```
gender                 0
age                    0
hypertension           0
heart_disease          0
ever_married           0
work_type              0
Residence_type         0
avg_glucose_level      0
bmi                  201
smoking_status         0
stroke                 0
dtype: int64
```

Wiersze z brakami w danych zostały usunięte z naszego zbioru danych, ponieważ nie było ich znacząco dużo.

One-hot encoding kategorycznych zmiennych

```
temp = pd.get_dummies(df.loc[:,['gender',
'work_type', 'Residence_type',
'smoking_status']])
```
✓ 0.6s Python

```
df.drop(columns=['gender', 'work_type',
'Residence_type', 'smoking_status'],
inplace=True)
```
✓ 0.4s Python

```
df = pd.concat([df, temp], axis=1)
```
✓ 0.4s Python

Zamieniamy wartości zmiennej 'ever_married' na wartości 0-1

```
df.ever_married = df.ever_married.map(dict
(Yes=1, No=0))
```
✓ 0.4s Python

Normalizujemy dane ilościowe

```
cols_to_norm = ['age', 'avg_glucose_level',
'bmi']
df2 = df.copy()
df[cols_to_norm] = df[cols_to_norm].apply
(lambda x: (x - x.min()) / (x.max() - x.min())
)
```
✓ 0.6s Python

Ostatecznie nasz zestaw danych zawiera następujące kolumny:

```
['age',
 'hypertension',
 'heart_disease',
 'ever_married',
 'avg_glucose_level',
 'bmi',
 'stroke',
 'gender_Female',
 'gender_Male',
 'gender_Other',
 'work_type_Govt_job',
 'work_type_Never_worked',
 'work_type_Private',
 'work_type_Self-employed',
 'work_type_children',
 'Residence_type_Rural',
 'Residence_type_Urban',
 'smoking_status_Unknown',
 'smoking_status_formerly smoked',
 'smoking_status_never smoked',
 'smoking_status_smokes']
```

Przystępujemy do tworzenia modelu

Tworzenie modelu

Wybór odpowiedniej ilości neuronów oraz warstw sieci. Problemem tym zajęliśmy się metodą prób i błędów. Z uwagi na 20 różnych zmiennych objaśniających stwierdziliśmy, że najlepszą startową wartością liczby neuronów będzie właśnie liczba 20, dodatkowo skoro nasza zmienne objaśniana jest pojedyncza oraz jest ona binarna do warstwy wychodzącej przypisaliśmy jeden neuron z funkcją aktywacji sigmoid, która według zgłębionej przez nas literatury jest najlepsza dla zagadnień binarnych gdyż zwraca wartości z zakresu od 0 do 1.

```
[79] model = Sequential()
     model.add(Dense(20, input_dim=20, activation='sigmoid'))
     model.add(Dense(20, activation='sigmoid'))
     model.add(Dense(1, activation='sigmoid'))

[80] model.compile(optimizer='adam', loss='binary_crossentropy', metrics=['accuracy'
```

Szybko zauważyliśmy jednak problem, że już właściwie od pierwszej obserwacji poziom accuracy modelu na zbiorze treningowym był bardzo wysoki i utrzymywał się na stałym poziomie. Model od drugiej epoki przestał się uczyć.

```
- loss: 0.4195 - accuracy: 0.8571 -
- loss: 0.1960 - accuracy: 0.9583 -
- loss: 0.1769 - accuracy: 0.9583 -
- loss: 0.1732 - accuracy: 0.9583 -
- loss: 0.1723 - accuracy: 0.9583 -
- loss: 0.1714 - accuracy: 0.9583 -
- loss: 0.1706 - accuracy: 0.9583 -
- loss: 0.1697 - accuracy: 0.9583 -
```

Poza zbiorami treningowymi i walidacyjny sprawdziliśmy jeszcze zbiór testowy. Co prawda accuracy wyszło tu bardzo duże jednak czuliśmy, że coś może być nie tak.

```
model.evaluate(x_test,y_test)

31/31 [==============================] - 0s 2ms/step - loss: 0.1442 - accuracy: 0.9572 -
[0.14421053230762482, 0.9572301506996155, 0.0, 0.0]
```

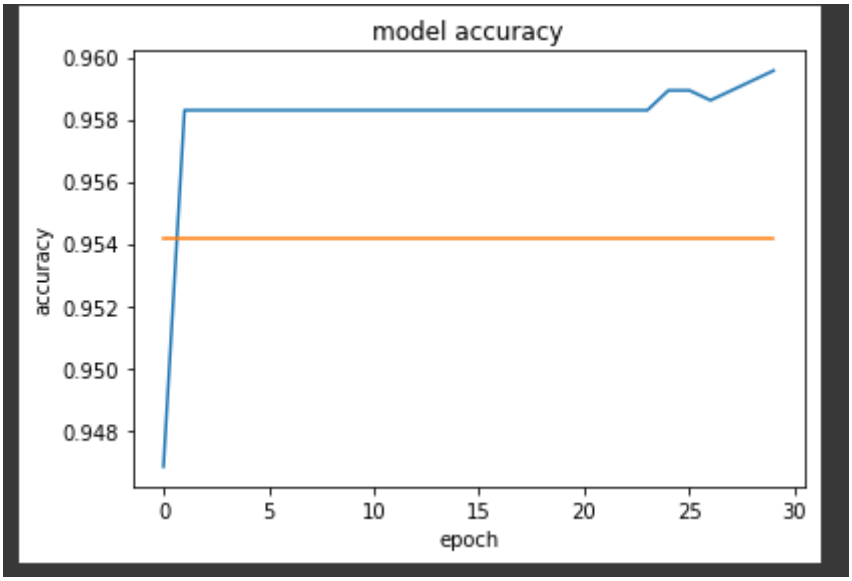
```
from sklearn.metrics import accuracy_score
y_pred=model.predict(x_test)
y_pred=(y_pred>0.5).astype(int)

from sklearn.metrics import  confusion_matrix
print(confusion_matrix(y_pred,y_test))
print("Classification on Depp Learning Accuracy score",accuracy_score(y_test,y_pred))
Deep_Learning_Score=accuracy_score(y_test,y_pred)
```

```
[[940  42]
 [  0   0]]
Classification on Depp Learning Accuracy score 0.9572301425661914
```

W celu poprawy tego problemu postanowiliśmy zmienić funkcję aktywacji na relu na wszystkich warstwach poza warstwą wyjściową oraz zmniejszyć ilość epok do 30 gdyż liczba 100 okazała się zbędna. Zmiany się pojawiły jednak były one na tyle znikome, ze postanowiliśmy dodać kolejne warstwy ukryte, a samą ilość neuronów na poszczególnych warstwach zmniejszyć. W ten sposób chcieliśmy uzyskać sieć, która będzie w mogła w bardziej szczegółowy sposób zagłębić się w problem.

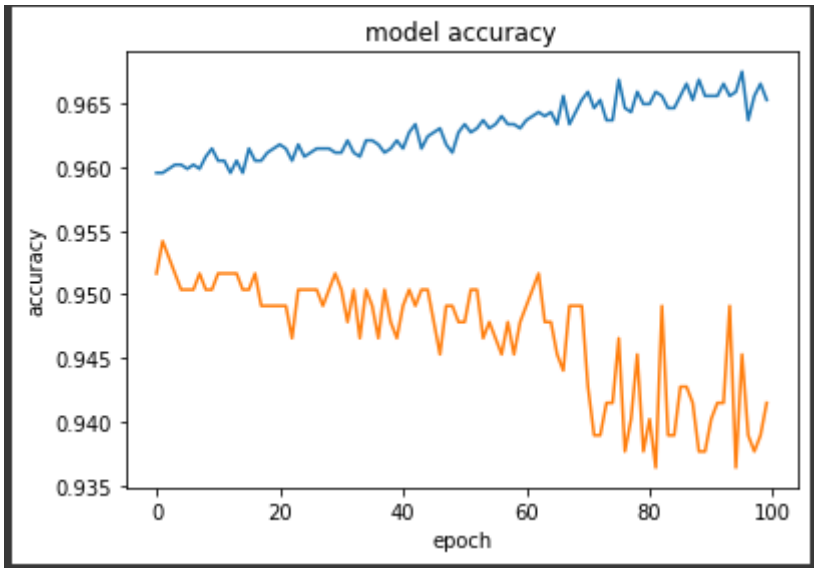
Na wykresie niebieska linia oznacza accuracy dla zestawu treningowego, pomaranczowa dla walidacyjnego



Wartości w dalszym stopniu nie wyglądały na zadowalające, rozbieżność między accuracy była bardzo mała a model praktycznie się nie uczył pomiędzy epokami, z uwagi na jednak minimalne zmiany postanowiliśmy wrócić do 100 epok.

Następnie dodaliśmy kolejną warstwę, tym razem z jeszcze mniejszą ilością neuronów.

```
model = Sequential()
model.add(Dense(20, input_dim=20, activation='relu'))
model.add(Dense(16, activation='relu'))
model.add(Dense(16, activation='relu'))
model.add(Dense(8, activation='relu'))
model.add(Dense(1, activation='sigmoid'))
```

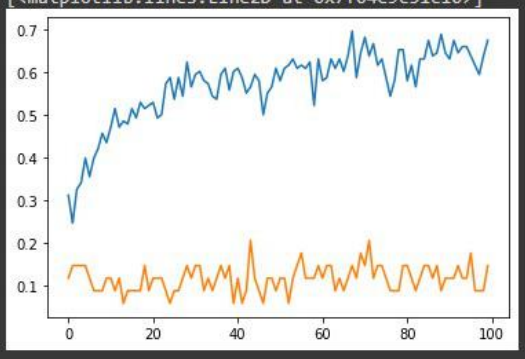


(w kodzie wykresy nie są w takiej samej kolejności co sprawozdanie, wynika to z tego, że na początku odczytywaliśmy czyste wartości, a dopiero później zdecydowaliśmy się na narysowanie wykresu)

Otrzymane wyniki były na pewno bardziej odbiegające od siebie jednak zdziwił nas trend spadkowy accuracy . Dowiedzieliśmy się o jeszcze kilku metrykach dzięki którym można sprawdzać poprawność modelu.

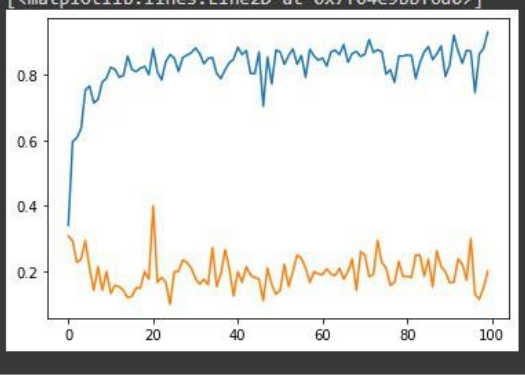
```
plt.plot(history.history['recall_3'])
plt.plot(history.history['val_recall_3'])
```

```
[<matplotlib.lines.Line2D at 0x7f04e9c51c10>]
```




```
[ ] plt.plot(history.history['precision_3'])
    plt.plot(history.history['val_precision_3'])
```

```
[<matplotlib.lines.Line2D at 0x7f04e9bbf6d0>]
```



Po przygotowaniu wykresów recall oraz precision zauważyliśmy bardzo duże różnice między tymi wartościami. W tym momencie zaczęliśmy się zastanawiać nad tym, że to może z samymi danymi jest coś nie w porządku.

```
[ ] df["stroke"].value_counts()

    0    4700
    1     209
    Name: stroke, dtype: int64
```

Niestety ale niezbalansowane dane okazały się prawdziwe, przy ich doborze nie wiedzieliśmy, że może być to problemem jednak w trakcie tworzenia projektu dzięki zdobytej wiedzy byliśmy w stanie stwierdzić, że nasz model najprawdopodobniej przy około 4,3% szansie na obecność choroby stwierdził, że bardziej opłacalne będzie oznaczanie każdego lub przynajmniej znacznej większości przypadków jako osoby zdrowe, zamiast badać faktyczne zależności zachodzące w tym modelu. I faktycznie 100 punktów procentowych – 4,3 punktów procentowych = 95,7 p.p. co jest równe uzyskiwanemu przez nas accuracy. Z tego powodu zdecydowaliśmy się stworzyć nowy model, tym razem dla mniejszej ilości obserwacji : 209 przypadków osób które dostały udar i 209 przypadków osób które udaru nie dostały. Wiersze podajemy w losowej kolejności.

Tworzenie modelu dla zmniejszonej i zbalansowanej ramki danych

Przygotowujemy ramkę danych

```
temp1 = df[df.stroke == 1].sample(209)
temp2 = df[df.stroke == 0].sample(209)
df418 = pd.concat([temp1,temp2],axis=0)
df418 = df418.sample(frac=1)
df418.info()
```

Dobieranie najbardziej odpowiedniej ilości warstw oraz neuronów

```
model418 = Sequential()
model418.add(Dense(20, input_dim=20,
activation='relu'))
model418.add(Dense(16, activation='relu'))
model418.add(Dense(16, activation='relu'))
model418.add(Dense(8, activation='relu'))
model418.add(Dense(1, activation='sigmoid'))
```

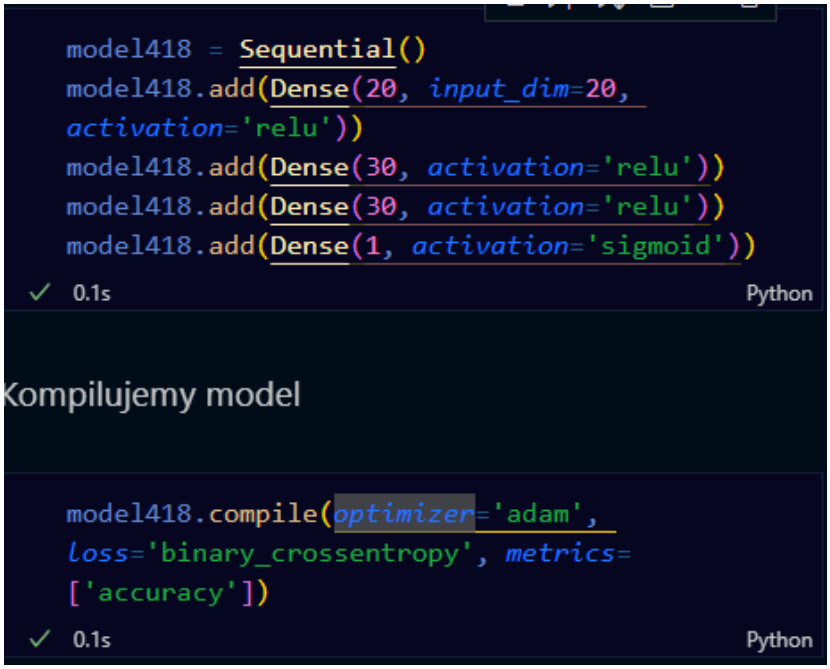
✓ 0.2s Python

Kompilujemy model

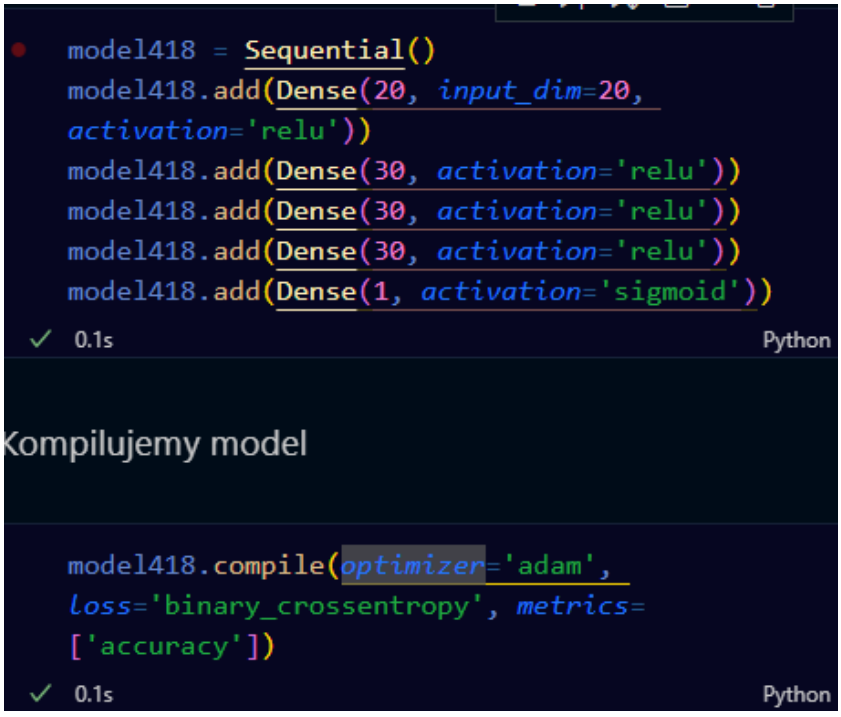
```
model418.compile(optimizer='adam',
loss='binary_crossentropy', metrics=['accuracy']
)
```

✓ 0.1s Python

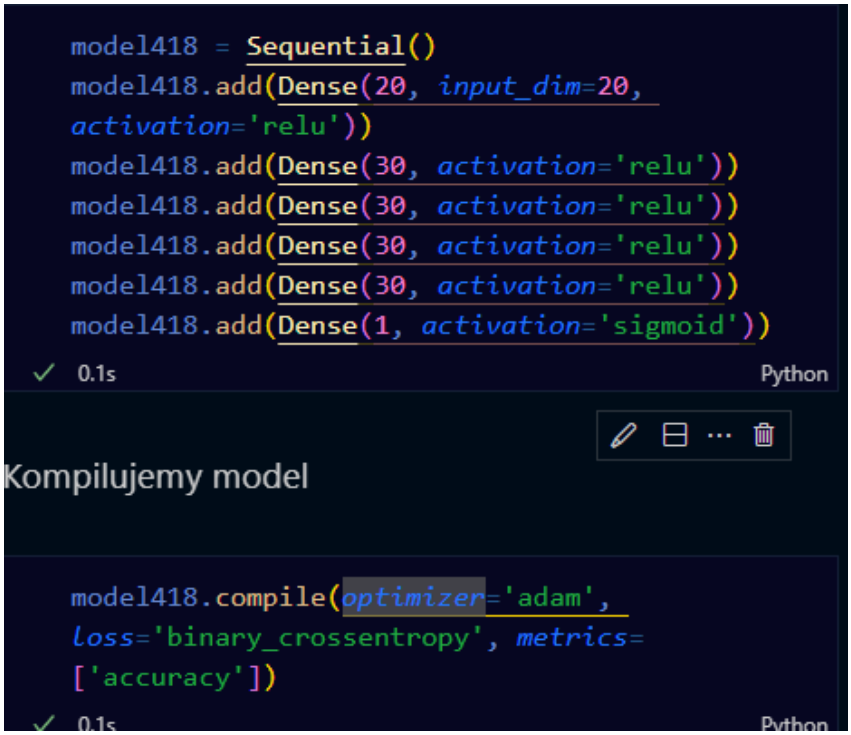
```
Train: 0.907, Test: 0.726
```



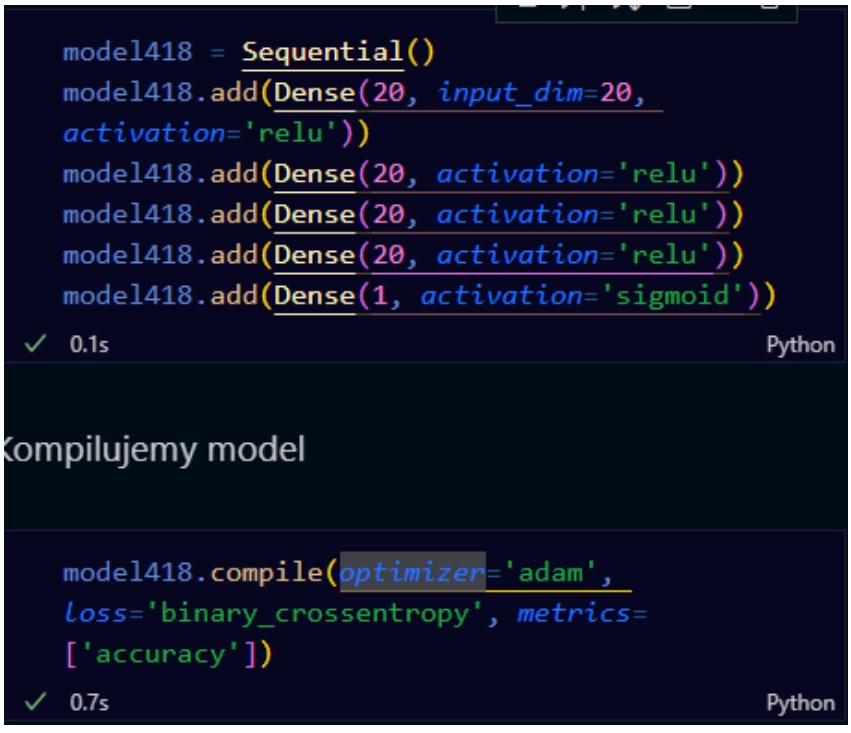
Train: 0.904, Test: 0.714



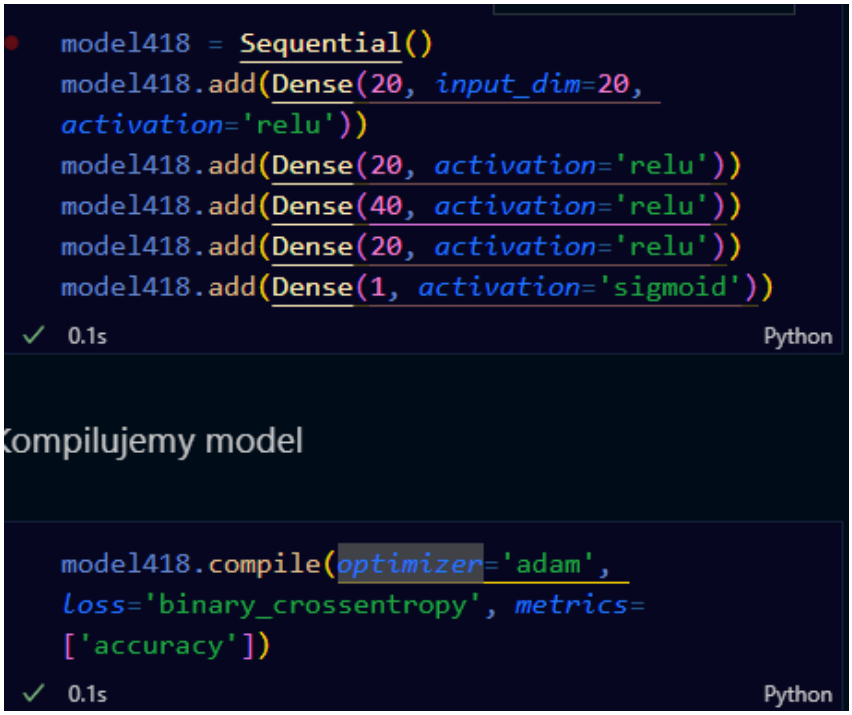
Train: 0.931, Test: 0.702



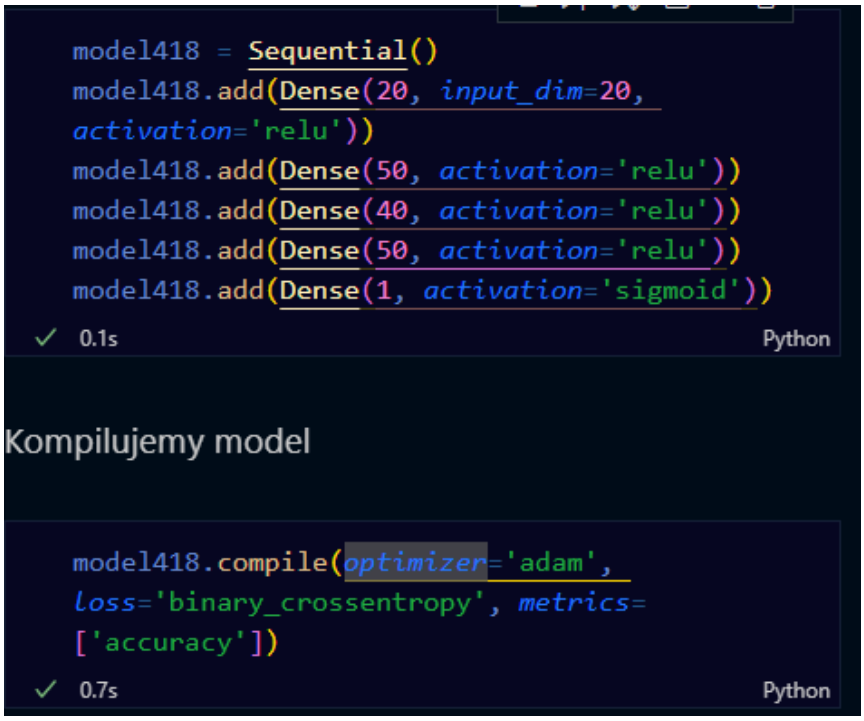
Train: 0.964, Test: 0.655



Train: 0.913, Test: 0.655




```
Train: 0.961, Test: 0.690
```




```
Train: 0.940, Test: 0.655
```

Patrzymy na metrykę accuracy, która jest stosowana do problemów klasyfikacji. Wybieramy model z accuracy: **Train: 0.907, Test: 0.726.** Ważne jest aby wartość accuracy była wysoka, ale także aby różnica, pomiędzy accuracy z testowego, a treningowego zbioru była jak najmniejsza.

```
model418 = Sequential()
model418.add(Dense(20, input_dim=20,
activation='relu'))
model418.add(Dense(16, activation='relu'))
model418.add(Dense(16, activation='relu'))
model418.add(Dense(8, activation='relu'))
model418.add(Dense(1, activation='sigmoid'))
```

✓ 0.2s Python

Kompilujemy model

```
model418.compile(optimizer='adam',
loss='binary_crossentropy', metrics=['accuracy']
)
```

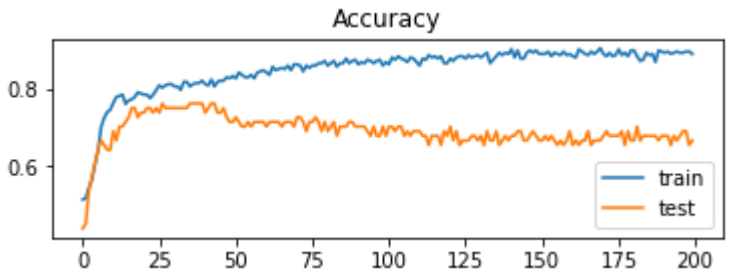
✓ 0.1s Python

Model trenujemy z liczbą epoch=200.

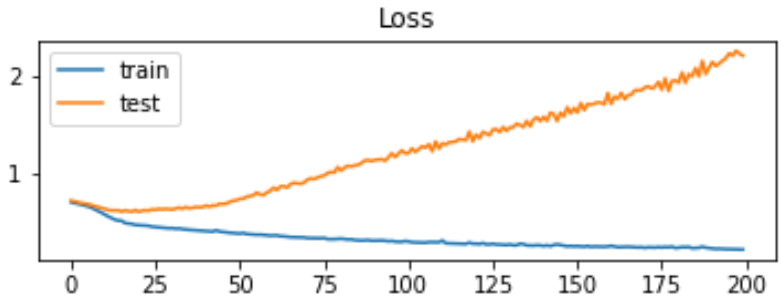
```
history = model418.fit(x418_train, y418_train,
validation_data=(x418_test, y418_test),
epochs=200, verbose=0)
```

✓ 9.8s Python

Na wykresie dla accuracy widzimy, że modelu nie da się już bardziej wytrenować, jest on wręcz przetrenowany. Zmienimy liczbę epoch i przedstawimy nowy wykres.



Na wykresie dla loss widzimy, że ma różną wydajność w przypadku danych testowych jak i treningowych. Obydwie funkcje zaczynają od siebie odchodzić oznacza to, że powinniśmy zakończyć trenowanie na wcześniejszej epoce. Występuje przetrenowanie modelu.

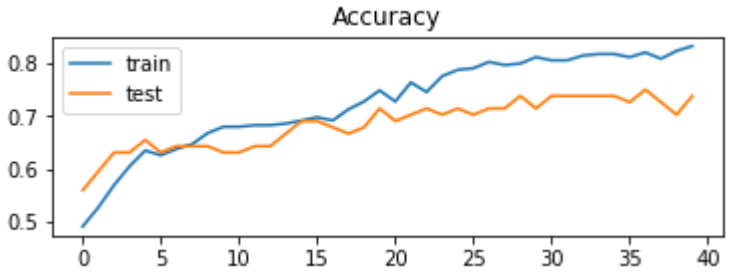


Wyciągając wnioski z powyższych rozważań zmniejszamy liczbę epoch do 40 i trenujemy model od nowa.

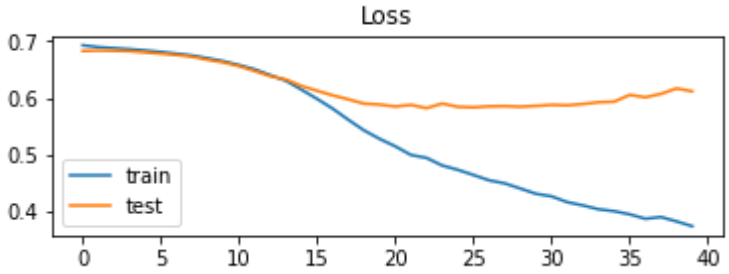
```
history = model418.fit(x418_train, y418_train,
validation_data=(x418_test, y418_test),
epochs=40, verbose=0)
```

✓ 2.7s Python

Wartość accuracy dla obu modeli rośnie do około 40 epoki. Później następuje przetrenowanie modelu.



W wykresu dla loss widzimy, że funkcje dla danych treningowych i testowych rozchodzą się w wartości epoch około 14. Straty maleją dla treningowego zbioru cały czas, a dla testowego zaczynają być stałe/wzrastać od 14 epoki.



Jak możemy zauważyć, to dla bardziej zbalansowanych danych to znaczy zmienna celu jest w równych proporcjach, accuracy już nie jest takie duże dla obu zbiorów danych (treningowego i testowego).

Bibliografia

https://machinelearningmastery.com/display-deep-learning-model-training-history-in-keras/

https://machinelearningmastery.com/choose-an-activation-function-for-deep-learning/

https://www.geeksforgeeks.org/intuition-of-adam-optimizer/

https://machinelearningmastery.com/how-to-choose-loss-functions-when-training-deep-learning-neural-networks/

https://thestory.is/pl/journal/co-to-jest-deep-learning/

https://www.tensorflow.org/api_docs/python/tf/keras/metrics/Precision

https://bulldogjob.pl/readme/czym-jest-deep-learning-i-sieci-neuronowe

https://pl.wikipedia.org/wiki/Funkcja_aktywacji

https://www.ncbi.nlm.nih.gov/pmc/articles/PMC8641997/
https://www.sciencedirect.com/science/article/pii/S2772442522000090

https://www.mdpi.com/2076-3417/10/19/6791