Vilniaus Universitetas

Žaidimas „Kryžiukai-nuliukai“

Papildoma užduotis

Darbą atliko:

Dovydas Martinkus

Duomenų Mokslas 4 kursas 1gr.

Vilnius, 2022

**Turinys**

[1 Tikslas ir uždaviniai 3](#_Toc115375381)

[2 Duomenys **Klaida! Žymelė neapibrėžta.**](#_Toc115375382)

[3 Užduoties ataskaita 4](#_Toc115375383)

[4 Išvados 9](#_Toc115375384)

# Tikslas ir uždaviniai

Tikslas: sukurti programą, įgyvendinančią žaidimą „Kryžiukai-nuliukai“, taikant daugiasluoksnį tiesioginio sklidimo neuroninį tinklą.

Žaidimo taisyklės. Dviem žaidėjams skirtas žaidimas. Žaidimo laukas: 9 langelių (3×3) lentelė. Vienas žaidėjas lentelės langelius žymi X (kryžiukais), antrasis – 0 (nuliukais). Žaidėjai eina paeiliui. Laimi tas, kuris pirmas užpildo pilną eilutę, stulpelį ar įstrižainę savo ženklais.

Uždaviniai:

Sukurti programą, įgyvendinančią žaidimą „Kryžiukai-nuliukai“.

Nustatyti daugiasluoksnio neuroninio tinklo architektūrą, tinklo parametrus.

Paruošti duomenis tinklo mokymui.

Sukurti neuroninio tinklo apmokymo strategiją.

# Užduoties ataskaita

Užduotis atlikta naudojant programavimo kalbą „Python“. Naudota „TensorFlow“ biblioteka, patogumo dėlei pasitelkiant „Keras“ API.

Norint sudaryti mokymo duomenis pasirinkta generuoti atsitiktines žaidimo partijas. Mokymo duomenis sudaro atsitiktinai pasirinktos sugeneruotų žaidimo partijų stadijos (nuo 1 iki 9 ėjimo), tuo tarpu duomenų žymas – galutinis generuotos partijos rezultatas („X“ pergalė, „0“ pergalė arba lygiosios). Tiek duomenims, tiek duomenų žymoms naudotas užkodavimas pseudokintamaisiais (one-hot encoding):

**import** tensorflow **as** tf

**from** tensorflow.keras **import** layers

**import** matplotlib.pyplot **as** plt

**import** numpy **as** np

*# sudaro vieną partiją atsitiktinai prisirdamas ėjimus*

**def** one\_random\_game():

board **=** np**.**zeros((9,3,3))

coin\_toss **=** np**.**random**.**choice([**-**1,1])

current\_player **=** coin\_toss

**for** turn **in** range(0,9):

**if** turn **>** 0:

board[turn,:,:] **=** board[turn**-**1,:,:]

current\_player **=** current\_player **\*** **-**1

**while** **True**:

choice **=** np**.**random**.**randint(0, 9)

**if** board[turn,choice **//** 3, choice **%** 3] **==** 0:

board[turn,choice **//** 3, choice **%** 3] **=** current\_player

**break**

**return** board

*# tikrina ar yra laimėtojas*

**def** check\_winner(boards):

**for** i **in** boards:

results **=** np**.**concatenate((

np**.**array([np**.**sum(np**.**diagonal(i))]),

np**.**array([np**.**sum(np**.**diagonal(np**.**fliplr(i)))]),

np**.**sum(i,1),

np**.**sum(i,0)))

**if** np**.**any(np**.**isin(results,3)):

**return** 1

**elif** np**.**any(np**.**isin(results,**-**3)):

**return** **-**1

**return** 0

*# simuliuoja n atsitiktinių partijų*

**def** simulate\_games(n):

boards **=** []

winners **=** []

**for** i **in** range(0,n):

board **=** one\_random\_game()

boards**.**append(board)

winners**.**append(check\_winner(board))

**return** boards, winners

*# paruošiami mokymo duomenys*

**from** sklearn.model\_selection **import** train\_test\_split

**from** sklearn.preprocessing **import** OneHotEncoder, LabelEncoder

onehotencoder\_X **=** OneHotEncoder()

onehotencoder\_y **=** OneHotEncoder()

onehotencoder\_y**.**fit(np**.**array([**-**1,0,1])**.**reshape(**-**1,1))

**def** create\_data(n):

boards, winners **=** simulate\_games(n)

X **=** []

**for** board **in** boards:

X**.**append(board[np**.**random**.**randint(0, 9)]**.**flatten())

*# one-hot formatas rezultatui*

y **=** onehotencoder\_y**.**transform(np**.**array(winners)**.**reshape(**-**1,1))**.**toarray()

X\_train, X\_test, y\_train, y\_test **=** train\_test\_split(X,y,train\_size**=**0.8,test\_size**=**0.2)

*# one-hot formatas lentos būsenai*

X\_train **=** onehotencoder\_X**.**fit\_transform(X\_train)**.**toarray()

X\_test **=** onehotencoder\_X**.**transform(X\_test)**.**toarray()

X\_train **=** np**.**delete(X\_train, [0,3,6,9,12,15,18,21,24], axis**=**1)

X\_test **=** np**.**delete(X\_test, [0,3,6,9,12,15,18,21,24], axis**=**1)

**return** X\_train, X\_test, y\_train, y\_test

X\_train, X\_test, y\_train, y\_test **=** create\_data(10000)

Sudarytas tiesioginio sklidimo neuroninį tinklas naudoja du pilnai sujungtus sluoksnius su ReLu aktyvacijos funkcijomis (kuriuos sudaro atitinkamai 128 ir 64 sluoksniai). Klasės prognozei gauti naudojamas tris neuronus turintis sluoksnis su softmax aktyvacijos funkcija. Kaip matome 3 pav. po 15 mokymo epochų modelis pasiekia 70% tikslumą mokymo aibėje (tiesa, dėl mokymo duomenų specifikos tai apskritai nėra geras modelio „įgūdžių“ įvertinimas).

inputs **=** tf**.**keras**.**Input(shape**=**(18,))

x **=** layers**.**Dense(units**=**128, kernel\_initializer**=**'uniform', activation**=**'relu')(inputs)

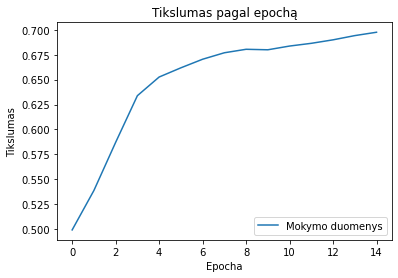
x **=** layers**.**Dense(units**=**64, kernel\_initializer**=**'uniform', activation**=**'relu')(x)

output **=** layers**.**Dense(units**=**3, activation**=**'softmax')(x)

model **=** tf**.**keras**.**Model(inputs, output)

model**.**compile(optimizer**=**'adam', loss**=**'categorical\_crossentropy', metrics**=**['accuracy'])

history **=** model**.**fit(X\_train,y\_train, batch\_size**=**10, epochs**=**15)



1 pav. Mokymo tikslumas pagal epochą

Apmokytas modelis panaudotas implementuoti „kryžiukų-nuliukų“ žaidimą. Modelio ėjimo metu patikrinami visi galimi ėjimai ir pasirenkamas tas, kuriam apmokytas modelis grąžina didžiausia savo laimėjimo tikimybę:

**def** next\_moves(current\_board):

n\_possible\_moves **=** np**.**where(current\_board**==**0)[0]**.**shape[0]

possible\_moves **=** np**.**repeat(np**.**reshape(current\_board,(1,3,3)),n\_possible\_moves,axis**=**0)

model\_format **=** np**.**zeros((n\_possible\_moves,9))

**for** i **in** range(len(possible\_moves)):

possible\_moves[i,np**.**where(current\_board**==**0)[0][i],np**.**where(current\_board**==**0)[1][i]] **=** **-**1

model\_format[i,:] **=** possible\_moves[i,:,:]**.**flatten()

**return** possible\_moves, model\_format

*# atspausdina tuo metu esančią lentą su 'X' ir 'O'*

**def** print\_board(board):

board\_print **=** np**.**empty\_like(board,str)

board\_print[board **==** 1] **=** 'X'

board\_print[board **==** **-**1] **=** 'O'

print(board\_print)

*# skirta žaisti su apmokytu modeliu*

**def** play(model, verbose **=** **True**):

board **=** np**.**zeros((3,3))

coin\_toss **=** np**.**random**.**choice([**-**1,1])

current\_player **=** coin\_toss

**for** i **in** range(0,9):

current\_player **=** current\_player **\*** **-**1

**if** verbose:

print\_board(board)

winner **=** check\_winner(board**.**reshape(1,3,3))

**if** winner **==** 1:

print("\nJūs laimėjote")

**return** 1

**elif** winner **==** **-**1:

print("\nJūs pralaimėjote")

**return** **-**1

**if** current\_player **==** 1:

**while** **True**:

row **=** int(input('Pasirinkite eilutę\n'))

column **=** int(input('Pasirinkite stulpelį\n'))

**if** board[row,column] **!=** 0:

print("Negalima, pasirinkite iš naujo!\n")

**else**:

board[row,column] **=** 1

**break**

**else**:

possible\_moves, model\_format **=** next\_moves(board)

model\_format **=** onehotencoder\_X**.**transform(model\_format)**.**toarray()

model\_format **=** np**.**delete(model\_format, [0,3,6,9,12,15,18,21,24], axis**=**1)

board **=** possible\_moves[np**.**argmax(model**.**predict(model\_format)[:,0])]

**if** i **==** 8:

print("\nLygiosios")

**return** 0

Norint tiksliau patikrinti apmokyto modelio „įgūdžius“ pasirinkta paleisti jį žaisti prieš atsitiktinai langelius pasirenkantį (zero-skill) modelį. Gauti rezultatai parodė, kad apmokytas modelis laimėjo 93% partijų ir lygiosiomis sužaidė 1%.

*# skirta žaisti su apmokytu modeliu*

**def** play\_random(model):

board **=** np**.**zeros((3,3))

coin\_toss **=** np**.**random**.**choice([**-**1,1])

current\_player **=** coin\_toss

**for** i **in** range(0,9):

current\_player **=** current\_player **\*** **-**1

winner **=** check\_winner(board**.**reshape(1,3,3))

**if** winner **==** 1:

**return** 1

**elif** winner **==** **-**1:

**return** **-**1

**if** current\_player **==** 1:

**while** **True**:

choice **=** np**.**random**.**randint(0, 9)

**if** board[choice **//** 3, choice **%** 3] **==** 0:

board[choice **//** 3, choice **%** 3] **=** 1

**break**

**else**:

possible\_moves, model\_format **=** next\_moves(board)

model\_format **=** onehotencoder\_X**.**transform(model\_format)**.**toarray()

model\_format **=** np**.**delete(model\_format, [0,3,6,9,12,15,18,21,24], axis**=**1)

board **=** possible\_moves[np**.**argmax(model**.**predict(model\_format)[:,0])]

**if** i **==** 8:

**return** 0

**def** play\_n(n):

winners **=** []

**for** i **in** range(n):

winners**.**append(play\_random(model))

print("Modelis laimėjo " **+** str(np**.**sum(np**.**array(winners)**==-**1)**/**n**\***100) **+** "%")

print("Lygiosios " **+** str(np**.**sum(np**.**array(winners)**==**0)**/**n**\***100) **+** "%")

play\_n(100)

Modelis laimėjo 93.0%

Lygiosios 1.0%

# Išvados

Naudojant atsitiktinai sugeneruotų partijų modelio mokymo strategija sudarytas tiesioginio sklidimo dirbtinis neuroninis tinklas.

Neuroniniame tinkle panaudoti du pilnai sujungti sluoksnius su ReLu aktyvacijos funkcijomis (kuriuos sudarė atitinkamai 128 ir 64 sluoksniai). Klasei prognozuoti naudotas tris neuronus turintis sluoksnis su softmax aktyvacijos funkcija.

Gauti rezultatai parodė, kad apmokytas modelis laimėjo 93% partijų ir lygiosiomis sužaidė 1% žaisdamas prieš atsitiktinai langelius pasirenkantį modelį.