Vilniaus Universitetas

Žaidimas „Kryžiukai-nuliukai“

Papildoma užduotis

Darbą atliko:

Dovydas Martinkus

Duomenų Mokslas 4 kursas 1gr.

Vilnius, 2022

Turinys

1 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.

2 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 arba kažkurio iš žaidėjų pergalės), 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 j,i in enumerate(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, j
        elif np.any(np.isin(results,-3)):
            return -1, j

    return 0, 8
```

```
# simuliuoja n atsitiktinių partijų
def simulate_games(n):
    boards = []
    winners = []
    for i in range(0,n):
        board = one_random_game()
        winner, n_boards = check_winner(board)

        boards.append(board[0:n_boards+1,:,:])
        winners.append(winner)
    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, board.shape[0])].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)
```

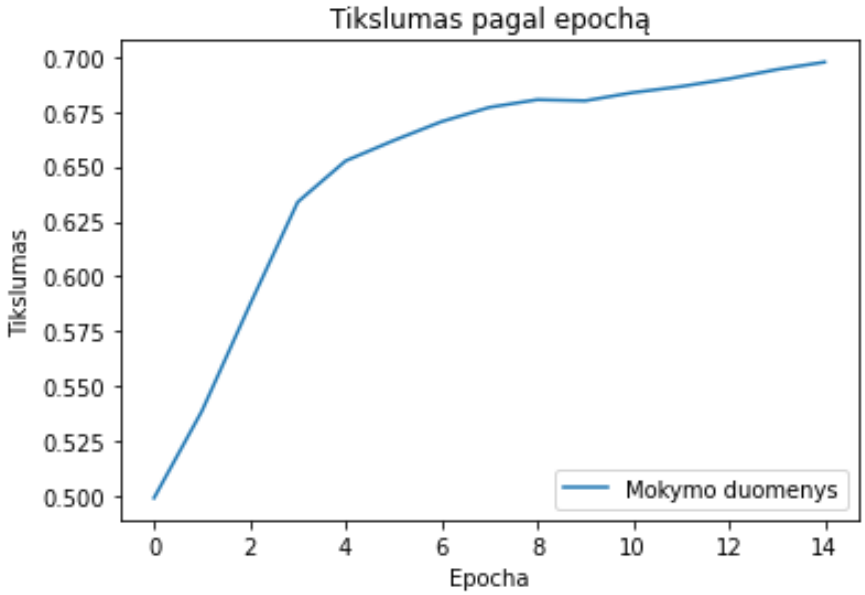
Sudarytas tiesioginio sklidimo neuroninį tinklas naudoja du pilnai sujungtus sluoksnius su ReLu aktyvacijos funkcijomis (kuriuos sudaro atitinkamai 128 ir 64 sluoksniai). Klasės prognozei gauti naudojamas tris neuronus turintis sluoksnis su softmax aktyvacijos funkcija. Kaip matome 1 pav., po 15 mokymo epochų modelis pasiekia 70 % tikslumą mokymo aibėje (tiesa, dėl mokymo duomenų specifikos tai apskritai nėra geras modelio „įgūdžių“ įvertinimas, nes priklausomai nuo jam pateikto pavyzdžio, modeliui gali tekti spėti žaidimo rezultatą tik iš pradinių žaidėjų ėjimų).

```
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%
```

3 Išvados

Naudojant atsitiktinai sugeneruotų partijų modelio mokymo strategiją sudarytas tiesioginio sklidimo dirbtinis neuroninis tinklas. Modeliui pateikti atsitiktinai sugeneruotų „kryžiukų-nuliukų“ partijų atsitiktiniai etapai, kartu su žaidėjo, kuris galiausiai laimėjo partiją, žyma.

Neuroniniame tinkle panaudoti du pilnai sujungti sluoksniai su ReLu aktyvacijos funkcijomis (kuriuos sudarė atitinkamai 128 ir 64 neuronai). 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.