Практическая работа по дисциплине "Нейроэволюционные вычисления"

Выполнил: Малкин Артем Юрьевич

Студент гр. 8ВМ22

Проверил: Григорьев Дмитрий Сергеевич

Старший преподаватель ОИТ ИШИТР ТПУ

19 Мая 2023

Номенклатура

NEAT	NeuroEvolution of Augmenting Topology
IoU	Intersection over Union
IN	Innovation Number

Цель работы

Реализовать алгоритм NEAT для решения задачи определения людей, больных диабетом, по данным анализов.

1 Ход работы

1.1 Используемые программные модули

В рамках практической работы был разработан программный код с применением модуля neat-python для реализации нейроэволюционного алгоритма NEAT. Для отображения схемы получившейся нейронной сети используется модуль networkx (создание объектов для дальнейшего их отображения) и matplotlib.

Модуль pandas используется для загрузки данных о людях, больных диабетом. Датасет представлен на сайте Kaggle[1]. Программный код был разработан на языке Python.

1.2 Описание алгоритма NEAT

Алгоритм NEAT предназначен для поиска оптимальной структуры нейронной сети, которая разрешит поставленную задачу. К примеру, задача определения людей, больных диабетом или раком, на основании анализов. Поиск оптимальной структуры нейронной сети предполагает поиск:

- Оптимального количества нейронов в скрытом слое сети, определение их связей.
- Оптимальных весовых коэффициентов для связей между нейронами.

Алгоритм NEAT оперирует инновационными числами (Innovation Numbers - IN), который содержит информацию о связях между нейронами, их весовом коэффициенте и включена ли эта связь. На рисунке 1 (взят из оригинальной статьи создателя алгоритма Kenneth O. Stanley [2]) сверху показан пример мутации нейронной сети: добавлена связь от нейрона 3 к нейрону 5. Поскольку такой связи ранее не было, этой информации о связи присваивается очередное IN. Если в ходе мутации создалась бы связь, которая уже участвовала в нейроэволюционном алгоритме, только что созданная связь получит то же IN.

На том же рисунке снизу при создании нового узла (Node) определились связи от нейрона 3 к нейрону 6 и от 6 к 4. Поскольку ранее была связь от 3 и 5 (IN = 7), а сейчас его нет, то данный представитель популяции лишается в своём фенотипе информации о этой связи.

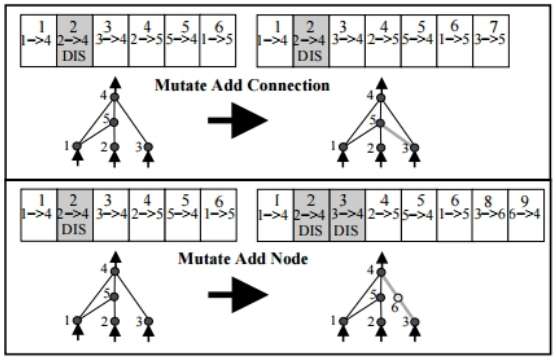

Figure 1: Процесс мутации нейронной сети

До того, как начать мутации, необходимо создать популяцию особей, а также произвести отбор особей. Оценка того, насколько особь приспособлена к решению поставленной задачи, происходит по заданной пользователем функции (fitness function). Те особи, что более близки к требуемым значениям (этап селекции), имеют больше шансов на скрещивание.

Скрещивание происходит по трём действиям:

1. Сопоставление генов. Гены - связи между узлами (нейронами) в сети. Совпадающие гены - те, которые имеют одинаковые IN, указывающие на происхождение генов. Совпадающие гены наследуются случайным образом от одного из родителей потомству.

2. Несовпадающие и избыточные гены. Несовпадающие - те, у которых различные IN и они находятся в одинаковом положении в геноме. Избыточные гены - гены с инновационным числом, превышающим число генов другого родителя. Оба типа генов наследуются от более приспособленного родителя без изменений. Если IN у генов совпадает, но различны состояния активности (enable/disable), то используется элемент случайности, согласно которой с некоторой долей вероятности потомок получит активированный или неактивированный ген. Это состояние может измениться в процессе мутации.

3. Весовые коэффициенты связей. После определения структурных генов устанавливаются веса связей между генами потомства. Одним из методов является усреднение весов связей совпадающих генов обоих родителей.

На рисунке 2 продемонстрирован пример операции кроссовера (скрещивания) особей для создания потомства.

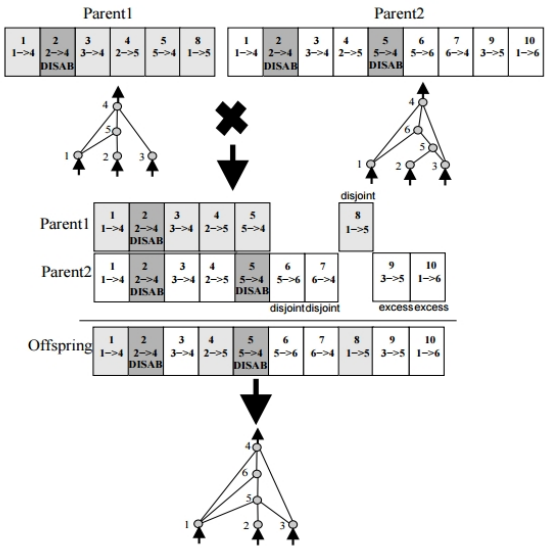

Figure 2: Операция кроссовера особей

Вместо того, что огульно тратить пули, пытаясь случайно попасть в оптимальную структуру сети, можно использовать хитрый алгоритм NEAT. Пули не работают, Жон (рисунок 3).

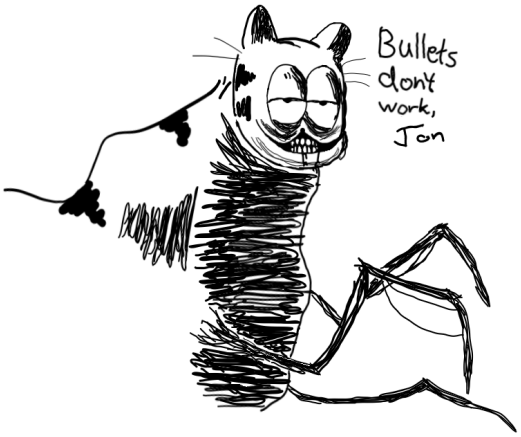

Figure 3: Bullets don't work, Jon

Вот так.

2 Программный код

В рамках данной работы был разработан программный код с применением модуля neat-python для реализации нейроэволюционного алгоритма NEAT. Программный код был разработан на языке Python.

```
import networkx as nx
import matplotlib.pyplot as plt
import matplotlib.patches as mpatches
import pandas as pd
import neat
import my_visualize as visualize
import numpy as np
import pickle

dataset = pd.read_csv('diabetes.csv', header=None, sep=',', engine='python')
dataset = dataset.drop([0])
dataset = dataset.astype(float)
inputs = dataset.iloc[:, 0:8].values
outputs = dataset.iloc[:, 8].values
inputs = np.array(inputs)
outputs = np.array(outputs)
config_path = 'config-diabetis'
config = neat.Config(neat.DefaultGenome, neat.DefaultReproduction,
                     neat.DefaultSpeciesSet, neat.
```

```
                                DefaultStagnation,
                            config_path)

def eval_genomes(genomes,config):
    '''
    evaluate genomes
    '''
    for genome_id, genome in genomes:
        genome.fitness = len(inputs)
        net = neat.nn.FeedForwardNetwork.create(genome,
            config)
        for xi, xo in zip(inputs, outputs):
            output = net.activate(xi)
            #use threshold
            output = 1 if output[0] > 0.5 else 0
            genome.fitness -= (output - xo) ** 2

def plot_the_graph(winner):
    '''
    plot the graph of the winner
    '''
    list_of_nodes = []
    for node in winner.connections.values():
        temp = []
        for neuron in node.key:
            temp.append(neuron)
        temp.append(node.weight)
        list_of_nodes.append(temp)

    unique_nodes = []
    for node in list_of_nodes:
        if node[0] not in unique_nodes:
            unique_nodes.append(node[0])
        if node[1] not in unique_nodes:
            unique_nodes.append(node[1])

    G = nx.DiGraph()
    i_neg = 0
    i_pos = 0
    for node in unique_nodes:
        if node > 0:
            G.add_node(node, pos=(1, i_pos ))
            i_pos += 2
        elif node < 0:
            G.add_node(node, pos=(0, i_neg))
            i_neg += 2
        else:
            G.add_node(node, pos=(2, 5))

    for node in list_of_nodes:
```

```
68          G.add_edge(node[0], node[1], weight=node[2])
69
70      # positions of the nodes
71      pos = nx.get_node_attributes(G, 'pos')
72
73      # color the arrows
74      edge_colors = ['r' if G[u][v]['weight'] < 0 else 'g'
            for u,v in G.edges()]
75      # green for input nodes, yellow for hidden nodes, red
            for output nodes
76      node_colors = ['gray' if node < 0 else 'y' if node > 0
             else 'pink' for node in G.nodes()]
77
78      # add legend to plot
79      red_patch = mpatches.Patch(color='red', label='
            Negative weights')
80      green_patch = mpatches.Patch(color='green', label='
            Positive weights')
81      gray_patch = mpatches.Patch(color='gray', label='Input
             nodes')
82      yellow_patch = mpatches.Patch(color='yellow', label='
            Hidden nodes')
83      pink_patch = mpatches.Patch(color='pink', label='
            Output nodes')
84
85      plt.legend(handles=[red_patch, green_patch, gray_patch
            , yellow_patch, pink_patch])
86
87      # weights of the edges
88      weights = nx.get_edge_attributes(G, 'weight')
89      # draw the graph
90      nx.draw(G, pos, with_labels=True, node_color=
            node_colors, edge_color=edge_colors, node_size
            =500, arrows=True)
91      # draw the weights
92      #nx.draw_networkx_edge_labels(G, pos, edge_labels=
            weights)
93      plt.show()
94
95  def calculate_metrics(winner_net, inputs, outputs):
96      '''
97      calculate metrics for winner net
98      '''
99      TP = 0
100     TN = 0
101     FP = 0
102     FN = 0
103     for xi, xo in zip(inputs, outputs):
104         output = winner_net.activate(xi)
105         output = 1 if output[0] > 0.5 else 0
```

```
106             #print("input {!r}, expected output {!r}, got {!r
                    }".format(xi, xo, output))
107             if output == 1 and xo == 1:
108                 TP += 1
109             elif output == 0 and xo == 0:
110                 TN += 1
111             elif output == 1 and xo == 0:
112                 FP += 1
113             elif output == 0 and xo == 1:
114                 FN += 1
115 
116         accuracy = (TP + TN) / (TP + TN + FP + FN)
117         precision = TP / (TP + FP)
118         IoU = TP / (TP + FP + FN)
119 
120         print("TP: ", TP)
121         print("TN: ", TN)
122         print("FP: ", FP)
123         print("FN: ", FN)
124         print("accuracy: ", accuracy)
125         print("precision: ", precision)
126         print("IoU: ", IoU)
127 
128     def launch(generations = 300, reporter = False, checkpoint
            = False):
129         '''
130         launch NEAT algorithm
131         '''
132         p = neat.Population(config)
133         # Add a stdout reporter to show progress in the
                terminal.
134         if (reporter):
135             p.add_reporter(neat.StdOutReporter(True))
136         stats = neat.StatisticsReporter()
137         p.add_reporter(stats)
138         if (checkpoint):
139             p.add_reporter(neat.Checkpointer(5))
140         winner = p.run(eval_genomes, 300)
141         # Display the winning genome.
142         print('Best genome:\n{!s}'.format(winner))
143         winner_net = neat.nn.FeedForwardNetwork.create(winner,
                config)
144         calculate_metrics(winner_net, inputs, outputs)
145         #visualize.draw_net(config, winner, True, node_names=
                node_names)
146         visualize.plot_stats(stats, ylog=False, view=True)
147         return winner
148 
149     def save_model(model, path):
150         '''
```

```
151     save model using pickle
152     '''
153     with open(path, 'wb') as output:
154         pickle.dump(model, output, pickle.HIGHEST_PROTOCOL
                )
155
156 def load_model(path):
157     '''
158     load model using pickle
159     '''
160     with open(path, 'rb') as input:
161         model = pickle.load(input)
162     return model
```

Listing 1: Программный код реализации алгоритма NEAT с применением модуля neat-python

3 Результаты работы

В датасете содержится 784 записей о людях, которые могут быть больны диабетом. В результате работы нейроэволюционного алгоритма NEAT в количестве 300 поколений была получена модель, значение фитнес-функции которой составляет 541 (то есть из 784 данных записей она правильно предскажет наличие или отсутствие болезни в 541 случае).

Был получен словарь (Python Dictionary), содержащий информацию о связях между нейронами входного, скрытого, выходного слоёв. Информация (рисунок 4) включает в себя сведения о:

- Скрытых и выходном нейронах: смещение, функция активации, способ агрегации.
- Связях между нейронами: входной и выходной нейроны, весовой коэффициент, активирована ли связь.

Была проведена оценка качества предсказания метриками Accuracy и Precision, IoU. Были определены следующие значения TP, TN, FP, FN:

- TP: 83 (столько человек действительно болеют диабетом, модель корректно их определила как больных людей).
- TN: 458 (человек не болеет, система полагает так же).
- FP: 42 (человек не болеет диабетом, но модель считает иначе).
- FN: 185 (человек болеют диабетом, но система не опредилила их).

Таким образом, значение метрики Accuracy составляет 0.7044, Precision - 0.664, IoU - 0.2677. Эти метрики говорят о не лучшем определении

```
Nodes:
    0 DefaultNodeGene(key=0, bias=0.1921233737777397, response=1.0, activation=sigmoid, aggregation=sum)
    25 DefaultNodeGene(key=25, bias=0.906180209238768 3, response=1.0, activation=sigmoid, aggregation=sum)
    26 DefaultNodeGene(key=26, bias=2.54161651184519, response=1.0, activation=sigmoid, aggregation=sum)
    27 DefaultNodeGene(key=27, bias=2.465819716184214, response=1.0, activation=sigmoid, aggregation=sum)
    28 DefaultNodeGene(key=28, bias=-0.0925126271451665, response=1.0, activation=sigmoid, aggregation=sum)
    242 DefaultNodeGene(key=242, bias=-2.7453973382974795, response=1.0, activation=sigmoid, aggregation=sum)
    321 DefaultNodeGene(key=321, bias=1.4265064754196617, response=1.0, activation=sigmoid, aggregation=sum)
Connections:
    DefaultConnectionGene(key=(-8, 27), weight=0.6369901485449989, enabled=True)
    DefaultConnectionGene(key=(-8, 28), weight=2.7494594241769366, enabled=False)
    DefaultConnectionGene(key=(-8, 321), weight=-0.576259583293118, enabled=True)
    DefaultConnectionGene(key=(-7, 25), weight=2.8066810805228637, enabled=True)
    DefaultConnectionGene(key=(-7, 26), weight=1.153590880605467, enabled=True)
    DefaultConnectionGene(key=(-7, 27), weight=1.1786734332462694, enabled=False)
    DefaultConnectionGene(key=(-7, 28), weight=3.4706160114056224, enabled=True)
    DefaultConnectionGene(key=(-6, 26), weight=0.5138074342322662, enabled=True)
    DefaultConnectionGene(key=(-5, 26), weight=0.853516244837919, enabled=False)
    DefaultConnectionGene(key=(-5, 27), weight=-1.560979814133694, enabled=True)
```

Figure 4: Выходные данные о лучшей модели (в отчёте приведён неполный список Connections)

людей, больных диабетом, как людей, которые болеют диабетом. Но модель показывает хорошие результаты, когда нужно подтвердить, что человек не болен диабетом.

На рисунке 5 представлен график значений фитнес-функции у лучшей модели и в среднем по поколению (с учётом стандартного отклонения).

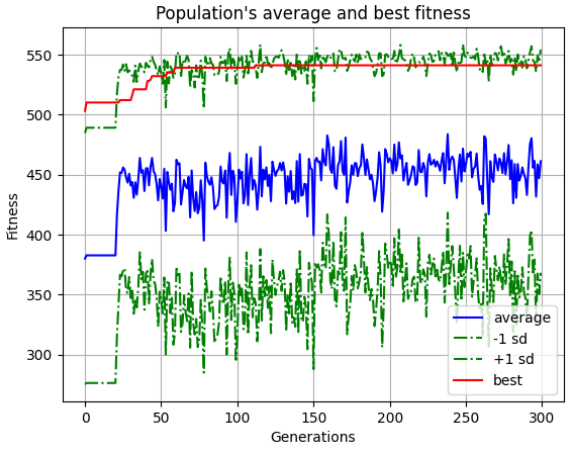

Figure 5: Среднее и лучшее значения фитнес-функции у популяции

На рисунке 6 представлена схема нейронной сети лучшей модели. На рисунке цифрами [-8:0] обозначены:

- 0: выходной сигнал (1 или 0 - болен или здоров)
- -1: Pregnancies - количество беременностей
- -2: Glucose - концентрация глюкозы в плазме через 2 часа после пероральной нагрузки

- -3: BloodPressure - диастолическое артериальное давление (мм рт. ст.)
- -4: SkinThickness - толщина кожной складки (мм)
- -5: Insulin - 2-часовой сывороточный инсулин (мкЕд/мл)
- -6: BMI - индекс массы тела (масса в кг / (рост в м)**2)
- -7: DiabetesPedigreeFunction - функция родословной диабета
- -8: Age - возраст (полных лет)

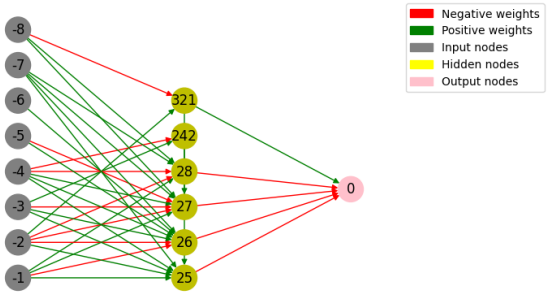

Figure 6: Структура лучшей модели

Для оценки промежуточного результата есть возможность изучать значения фитнес-функции есть встроенный в neat-python инструментарий (рисунок 7).

```
****** Running generation 0 ******

Population's average fitness: 383.04000 stdev: 105.62423
Best fitness: 500.00000 - size: (5, 36) - species 8 - id 8
Average adjusted fitness: 0.504
Mean genetic distance 3.574, standard deviation 0.491
Population of 100 members in 50 species:
   ID   age  size  fitness  adj fit  stag
  ====  ===  ====  =======  =======  ====
     1    0     2    308.0    0.186     0
     2    0     2    337.0    0.309     0
     3    0     2    496.0    0.983     0
     4    0     2    268.0    0.017     0
     5    0     2    286.0    0.093     0
     6    0     2    268.0    0.017     0
     7    0     2    499.0    0.996     0
     8    0     2    500.0    1.000     0
```

Figure 7: Оценка фитнес-функции у особей данного поколения

Заключение

В ходе выполнения практической работы было получение представление работы алгоритма NEAT в задачах поиска оптимальной структуры нейронной сети. Разработан программный код с применением модуля neat-python для реализации данного алгоритма на языке высокого уровня Python. Реализована функция отображения структуры нейронной сети, полученной в результате работы алгоритма NEAT. Получен опыт работы с набором макрорасширений системы компьютерной вёрстки TeX.

References

[1] "Pima Indians Diabetes Database," *Kaggle*, May 2023, https://www.kaggle.com/datasets/uciml/pima-indians-diabetes-database.

[2] Kenneth O. Stanley, R. M., "Evolving Neural Networks through Augmenting Topologies," *Evolutionary Computation 10 by Massachusetts Institue of Technology*, Vol. 2, February 2002, pp. 99–127.