ДОДАТОК А

Псевдокоди алгоритмів

А.1 Псевдокод алгоритму DQN

Input Replay memory size M, batch size d, number of episodes E, and number of time steps T
Inititalize Main network weights θ
Inititalize Target network weights θ^-
Inititalize Replay memory
for $e = 1, \ldots, E$ **do**
  Inititalize state s_1, and action a_1
  for $t = 1, \ldots, T$ **do**
    Take action $a_t = \text{argmax}_a\, Q^\pi(s_t, a; \theta)$ with probability $1 - \epsilon$ or a random action with probability ϵ
    Get reward r_t and observe next state s_{t+1}
    if Replay capacity M is full **then**
      Delete the oldest tuple in memory
    end if
    Store the tuple (s_t, a_t, r_t, s_{t+1}) to replay memory
    Sample random d tuples from replay memory
    $$y_t = \begin{cases} r_t, & \text{if } t = T. \\ r_t + \gamma \max_a Q^\pi(s_{t+1}, a_{t+1}; \theta_t^-), & \text{otherwise.} \end{cases}$$
    Perform policy gradient using y_t for updating θ
    Update target network every N step, $\theta^- = \theta$
  end for
end for

А.2 Псевдокод алгоритму «Random Forest»

Precondition: A training set $S := (x_1, y_1), \ldots, (x_n, y_n)$, features F, and number of trees in forest B.

function RANDOMFOREST(S, F)
  $H \leftarrow \emptyset$
  for $i \in 1, \ldots, B$ **do**
    $S^{(i)} \leftarrow$ A bootstrap sample from S
    $h_i \leftarrow$ RANDOMIZEDTREELEARN($S^{(i)}, F$)
    $H \leftarrow H \cup \{h_i\}$
  end for
  return H
end function
function RANDOMIZEDTREELEARN(S, F)
  At each node:
    $f \leftarrow$ very small subset of F
    Split on best feature in f
  return The learned tree
end function

А.3 Псевдокод алгоритму SVR

Data : Dataset with p^* variables and binary outcome.

Output: Ranked list of variables according to their relevance.

Find the optimal values for the tuning parameters of the SVM model;

Train the SVM model;

$p \leftarrow p^*$;

while $p \geq 2$ **do**

  $SVM_p \leftarrow$ SVM with the optimized tuning parameters for the p variables and observations in **Data**;

  $w_p \leftarrow$ calculate weight vector of the SVM_p $(w_{p1}, \ldots, w_{pp})$;

  $rank.criteria \leftarrow (w_{p1}^2, \ldots, w_{pp}^2)$;

  $min.rank.criteria \leftarrow$ variable with lowest value in $rank.criteria$ vector;

  Remove $min.rank.criteria$ from **Data**;

  $Rank_p \leftarrow min.rank.criteria$;

  p ← p - 1 ;

end

$Rank_1 \leftarrow$ variable in **Data** $\notin (Rank_2, \ldots, Rank_{p^*})$;

return $(Rank_1, \ldots, Rank_{p^*})$

А.4 Псевдокод алгоритму SAEs

Load database: z^{GT} (train partition);
Define network architecture: $N_{\mathrm{in}}^{\mathrm{SAE}} = N_{\mathrm{out}}^{\mathrm{SAE}} = D$, N_h^{SAE}, N_d^{SAE}, σ_j^{SAE};
Define hyperparameters: l_r^{SAE}, λ_r^{SAE};
Initialize $w_{i,j}^{\mathrm{SAE}}$, b_j^{SAE};
for each epoch **do**
    Initialize loss function: $C = 0$;
    for each train snapshot **do**
        Encoder: $\boldsymbol{x}_n^{\mathrm{SAE}} = q_\phi(\boldsymbol{z}_n^{\mathrm{GT}})$;
        Decoder: $\boldsymbol{z}_n^{\mathrm{SAE}} = p_\theta(\boldsymbol{x}_n^{\mathrm{SAE}})$;
        Loss function: $C \leftarrow C + \mathcal{L}_n^{\mathrm{rec}} + \lambda_r^{\mathrm{SAE}} \mathcal{L}_n^{\mathrm{reg}}$;
    end for
    MSE loss function: $\mathcal{L}^{\mathrm{SAE}} \leftarrow \frac{C}{N_{\mathrm{train}}}$
    Backward propagation;
    Optimizer step;
end for

ДОДАТОК Б

Код програми

Б.1 Реалізація DQN алгоритму

```
mport random
import torch
import torch.nn as nn
import nnarch
from collections import namedtuple
import copy
import math

device = "cuda" if torch.cuda.is_available() else "cpu"
Transition = namedtuple('Transition',
                        ('state', 'action', 'next_state', 'reward'))



class DqnAgent:
  def     init     (
        self,
        mode: str,
        replay,
        target_update: int,
        gamma: float,
        use_sgd: bool,
        eps_start: float,
        eps_end: float,
        eps_decay: int,
        input_dim: int,
```

```
        output_dim: int,
        batch_size: int,
        network_file: str = ''
    ):
        self.mode = mode
        self.replay = replay
        self.target_update = target_update
        self.gamma = gamma
        self.use_sgd = use_sgd
        self.eps_start = eps_start
        self.eps_end = eps_end
        self.eps_decay = eps_decay
        self.n_actions = output_dim
        self.batch_size = batch_size

        self.network_file = network_file
        self.policy_net = nnarch.DqnNetwork(input_dim, output_dim).to(device)
        self.target_net = nnarch.DqnNetwork(input_dim, output_dim).to(device)
        self.policy_net_copy = nnarch.DqnNetwork(input_dim, output_dim).to(device)
        if network_file:
        self.policy_net.load_state_dict(torch.load(network_file,
      map_location=torch.device(device)))
        self.target_net.load_state_dict(self.policy_net.state_dict())
        self.target_net.eval()

        self.learn_steps = 0
        self.z = None
        self.fixed_gamma = copy.deepcopy(gamma)
```

```
self.update_gamma = False
self.q_value_batch_avg = 0
def select_action(self, state, steps_done, invalid_action):
    original_state = state
    state = torch.from_numpy(state)
    if self.mode == 'train':
            sample = random.random()
            eps_threshold = self.eps_end + (self.eps_start - self.eps_end) *
    math.exp(-1. * steps_done / self.eps_decay)
    if sample > eps_threshold:
    with torch.no_grad():
            _,      sorted_indices      =      torch.sort(self.policy_net(state),
    descending=True)
            if invalid_action:
                    return sorted_indices[1]
            else:
                    return sorted_indices[0]
     else:
             decrease_state = [(original_state[0] + original_state[4]) / 2,
                               (original_state[1] + original_state[5]) / 2,
                               (original_state[2] + original_state[6]) / 2,
                               (original_state[3] + original_state[7]) / 2]
    congest_phase = [i for i, s in enumerate(decrease_state) if abs(s-1) <
1e-2]
     if len(congest_phase) > 0 and invalid_action is False:
             return random.choice(congest_phase)
     else:
             return random.randrange(self.n_actions)
     else:
             with torch.no_grad():
```

```
            _, sorted_indices = torch.sort(self.policy_net(state),
        descending=True)
         if invalid_action:
            return sorted_indices[1]
        else:
            return sorted_indices[0]
    def learn(self):
         if self.mode == 'train':
            if self.replay.steps_done <= 10000:
                return
            loss_fn = nn.MSELoss()
            if self.use_sgd:
                optimizer = torch.optim.SGD(self.policy_net.parameters(),
            lr=0.00025)
            else:
            optimizer = torch.optim.RMSprop(self.policy_net.parameters(),
            lr=0.00025)
            transitions = self.replay.sample(self.batch_size)
            batch = Transition(*zip(*transitions))
            state_batch = torch.cat(batch.state)
            action_batch = torch.cat(batch.action).view(self.batch_size, 1)
            next_state_batch = torch.cat(batch.next_state)
            reward_batch = torch.cat(batch.reward).view(self.batch_size,
        1)
            state_action_values = self.policy_net(state_batch).gather(1,
        action_batch)
            with torch.no_grad():
            argmax_action =
            self.policy_net(next_state_batch).max(1)[1].view(self.batch_siz
        e, 1)
```

```
                q_max = self.target_net(next_state_batch).gather(1,
            argmax_action)
                expected_state_action_values = reward_batch + self.gamma *
            q_max
            # for plot
            self.q_value_batch_avg =
            torch.mean(state_action_values).item()

            loss = loss_fn(state_action_values, expected_state_action_values)
        optimizer.zero_grad()

            loss.backward()

            self.cal_z(state_batch, action_batch, q_max)

            for param in self.policy_net.parameters():
            param.grad.data.clamp_(-1, 1)
            optimizer.step()
            self.learn_steps += 1
            self.update_gamma = False

        def cal_z(self, state_batch, action_batch, q_max):
        self.policy_net_copy.load_state_dict(self.policy_net.state_dict())
        z_optimizer = torch.optim.SGD(self.policy_net_copy.parameters(),
    lr=0.0001)
        state_action_copy_values = self.policy_net_copy(state_batch).gather(1,
        action_batch)
        z_optimizer.zero_grad()
        f_gamma_grad = torch.mean(0.00025 * q_max * state_action_copy_values)
        f_gamma_grad.backward()
```

```
 self.z = {'l1.weight': self.policy_net_copy.l1.weight.grad,
                'l1.bias': self.policy_net_copy.l1.bias.grad,
                'l2.weight': self.policy_net_copy.l2.weight.grad,
                'l2.bias': self.policy_net_copy.l2.bias.grad}

 def learn_gamma(self):
        loss_fn = nn.MSELoss()
        optimizer = torch.optim.SGD(self.policy_net.parameters(),
lr=0.00025)

 transitions = self.replay.sample(self.batch_size)
 batch = Transition(*zip(*transitions))
 state_batch = torch.cat(batch.state)
 action_batch = torch.cat(batch.action).view(self.batch_size, 1)
 next_state_batch = torch.cat(batch.next_state)
 reward_batch = torch.cat(batch.reward).view(self.batch_size, 1)
 state_action_values = self.policy_net(state_batch).gather(1, action_batch)
 with torch.no_grad():
        argmax_action =
self.policy_net(next_state_batch).max(1)[1].view(self.batch_size, 1)
        q_max = self.target_net(next_state_batch).gather(1, argmax_action)
        expected_state_action_values = reward_batch + self.fixed_gamma *
q_max

 loss = loss_fn(state_action_values, expected_state_action_values)
 optimizer.zero_grad()
 loss.backward()

 l1_weight = self.policy_net.l1.weight.grad * self.z['l1.weight']
 l1_bias = self.policy_net.l1.bias.grad * self.z['l1.bias']
```

```
        l2_weight = self.policy_net.l2.weight.grad * self.z['l2.weight']
        l2_bias = self.policy_net.l2.bias.grad * self.z['l2.bias']

        gamma_grad = -0.99 * torch.mean(torch.cat((l1_weight.view(-1),
l1_bias.view(-
        1), l2_weight.view(-1), l2_bias.view(-1))))
        self.gamma += gamma_grad
        self.update_gamma = True
```

Б.2 Архітектура мережі DQN

```
import torch
import torch.nn as nn
import torch.nn.functional as F
import numpy as np

device = "cuda" if torch.cuda.is_available() else "cpu"

def normalize_output(outputs):
    with torch.no_grad():
        for m, output in enumerate(outputs):
            max_t = torch.max(torch.abs(output))
            if abs(max_t-0) < 1e-2:
                continue
            for n, t in enumerate(output):
                outputs[m][n] /= max_t
    return outputs

class DqnNetwork(nn.Module):
    def __init__(self, inputs, outputs):
```

```
super(DqnNetwork, self).__init__()
self.l1 = nn.Linear(inputs, 512)
self.l2 = nn.Linear(512, outputs)
c = np.sqrt(1 / inputs)
nn.init.uniform_(self.l1.weight, -c, c)
nn.init.uniform_(self.l1.bias, -c, c)
nn.init.uniform_(self.l2.weight, -c, c)
nn.init.uniform_(self.l2.bias, -c, c)
def forward(self, x):
        x = x.to(device)
        return x
```

Б.3 Прогнозування транспортного потоку за допомогою мереж LSTM, SAE та GRU

```
import math
import warnings
import numpy as np
import pandas as pd
import tensorflow as tf
from data.data import process_data
from keras.models import load_model
from keras.utils.vis_utils import plot_model
import sklearn.metrics as metrics
import matplotlib as mpl
import matplotlib.pyplot as plt
warnings.filterwarnings("ignore")

tf.compat.v1.disable_v2_behavior()
```

```
def MAPE(y_true, y_pred):
    y = [x for x in y_true if x > 0]
    y_pred = [y_pred[i] for i in range(len(y_true)) if y_true[i] > 0]
    num = len(y_pred)
    sums = 0
    for i in range(num):
        tmp = abs(y[i] - y_pred[i]) / y[i]
        sums += tmp
    mape = sums * (100 / num)
    return mape

def eva_regress(y_true, y_pred):
    mape = MAPE(y_true, y_pred)
    vs = metrics.explained_variance_score(y_true, y_pred)
    mae = metrics.mean_absolute_error(y_true, y_pred)
    mse = metrics.mean_squared_error(y_true, y_pred)
    r2 = metrics.r2_score(y_true, y_pred)
    print('explained_variance_score:%f' % vs)
    print('MAPE:%f%%' % mape)
    print('MAE:%f' % mae)
    print('MSE:%f' % mse)
    print('RMSE:%f' % math.sqrt(mse))
    print('R2:%f' % r2)

def plot_results(y_true, y_preds, names):
    d = '2022-3-4 00:00'
    x = pd.date_range(d, periods=288, freq='5min')

    fig = plt.figure()
    ax = fig.add_subplot(111)
```

```
        ax.plot(x, y_true, label='Дані з доріг')
        for name, y_pred in zip(names, y_preds):
            ax.plot(x, y_pred, label=name)
        plt.legend()
        plt.grid(True)
        plt.xlabel('Час дня')
        plt.ylabel('Потік')

        date_format = mpl.dates.DateFormatter("%H:%M")
        ax.xaxis.set_major_formatter(date_format)
        fig.autofmt_xdate()

        plt.show()

def main():
        lstm = load_model('model/lstm.h5')
        gru = load_model('model/gru.h5')
        saes = load_model('model/saes.h5')
        models = [lstm, gru, saes]
        names = ['LSTM', 'GRU', 'SAEs']

        lag = 12
        file1 = 'data/train.csv'
        file2 = 'data/test.csv'
        _, _, X_test, y_test, scaler = process_data(file1, file2, lag)
        y_test = scaler.inverse_transform(y_test.reshape(-1, 1)).reshape(1, -
1)[0]
        y_preds = []
        for name, model in zip(names, models):
            if name == 'SAEs':
```

```
                X_test = np.reshape(X_test, (X_test.shape[0],
        X_test.shape[1]))
    else:
            X_test = np.reshape(X_test, (X_test.shape[0], X_test.shape[1],
    1))
    file = 'images/' + name + '.png'
    plot_model(model, to_file=file, show_shapes=True)
    predicted = model.predict(X_test)
    predicted = scaler.inverse_transform(predicted.reshape(-1,
1)).reshape(1, -1)[0]
    y_preds.append(predicted[:288])
    print(name)
    eva_regress(y_test, predicted)

    plot_results(y_test[: 288], y_preds, names)

if __name__ == '__main__':
        main()
```

ДОДАТОК В

Апробація результатів наукових досліджень

УДК 681.5:656

ВИЗНАЧЕННЯ ЗАДАЧ ДЛЯ РЕАЛІЗАЦІЇ АВТОМАТИЗОВАНОЇ СИСТЕМИ КОНТРОЛЮ РУХУ ТРАНСПОРТУ

Маслов О.А.
Науковий керівник – к.т.н., доц. Демська Н.П.
Харківський національний університет радіоелектроніки, каф. КІТАМ,
м. Харків, Україна
тел. +380967936373, e-mail: oleksandr.maslov1@nure.ua

This work deals with the development of an automated traffic control system. The importance of such systems for improving traffic efficiency and reducing accidents is discussed. The structure and functions of the system, including sensors, controllers, and software, are described. The problems that must be solved in order to create an effective traffic management system are given.

Транспорт забезпечує швидке та зручне пересування людей та вантажів, однак, зростаюча кількість транспортних засобів призводить до загострення проблем безпеки на дорогах та погіршення екологічної ситуації. Розроблення автоматизованої системи контролю руху транспорту є одним з важливих кроків у розв'язанні цих проблем.

Основними завданнями системи контролю руху транспорту є забезпечення безпеки руху на дорозі (для цього використовуються різні технології, такі як системи відеоспостереження, датчики руху та інші.) та покращення ефективності дорожнього руху.

Розвиток інтелектуальних транспортних систем є однією з перспективних напрямків в розвитку транспортної інфраструктури. Інтелектуальні транспортні системи дозволяють забезпечити автоматичне керування транспортом та забезпечити високу точність контролю руху транспорту.

Для визначення місцезнаходження транспортних засобів можна використовувати глобальну позиційну систему (GPS). Однак, ця система має недостатню точність в зонах з високою забудованістю та обмежена пропускна здатність в мережі. Тому, для більш точного визначення місцезнаходження транспортних засобів можна використовувати комбінований підхід, який включає в себе використання GPS та інших сучасних технологій.

Однією з таких технологій є технологія Radio Frequency Identification (RFID), яка дозволяє безконтактно ідентифікувати транспортні засоби. Для цього на транспортний засіб встановлюється спеціальний пристрій, що містить RFID-мітку, яка зчитується з використанням радіохвиль. Це дозволяє точно визначити місцезнаходження транспортного засобу та забезпечити його контроль на всіх етапах руху.

Крім того, для автоматизації процесу контролю руху транспорту можна використовувати системи відеоспостереження. Вони дозволяють в режимі реального часу відстежувати рух транспорту, фіксувати порушення ПДР та здійснювати дистанційний контроль за рухом транспорту.

Для розроблення автоматизованої системи контролю руху транспорту потрібно вирішити кілька проблем.

Перша проблема – це вибір необхідного обладнання для встановлення на транспортні засоби. Для цього потрібно провести дослідження різних типів обладнання і обрати найбільш ефективний варіант.

Друга проблема – це збір та обробка даних, що надходять від обладнання. Для цього потрібно розробити програмне забезпечення, яке б збирало та аналізувало інформацію про рух транспорту.

Третя проблема – це взаємодія з іншими системами контролю руху транспорту. Для того, щоб система працювала ефективно, вона повинна бути інтегрована з іншими системами контролю руху, наприклад, з системою моніторингу дорожньої ситуації.

Четверта проблема – це забезпечення захисту даних, які надходять від обладнання та зберігаються в системі. Для цього потрібно розробити систему безпеки, яка захистить дані від несанкціонованого доступу.

Остання проблема – це визначення вартості розробки та впровадження системи. Для цього потрібно провести економічний аналіз і визначити, чи буде система рентабельною.

Успішне вирішення цих проблем дозволить створити ефективну та безпечну систему контролю руху транспорту, яка зможе покращити безпеку дорожнього руху та зменшити кількість аварій на дорогах.

Для реалізації даного проекту необхідно врахувати ряд факторів, таких як вибір технічних засобів, розробка програмного забезпечення, налагодження взаємодії між системою та транспортними засобами, забезпечення захисту даних тощо.

Отже, розробка автоматизованої системи контролю руху транспорту є важливим кроком у поліпшенні безпеки на дорозі та забезпеченні більш ефективного управління транспортним потоком.

Список використаних джерел:

1. "Intelligent Transportation Systems." Federal Highway Administration, United States Department of Transportation, www.fhwa.dot.gov/its/.

2. Li, Qingquan, et al. "Intelligent Transportation Systems: A Comprehensive Review." IEEE Transactions on Intelligent Transportation Systems, vol. 14, no. 4, 2013, pp. 1784-1796.

3. "Smart Transport for Cities: The Future of Transportation." European Commission, 2018, ec.europa.eu/jrc/en/publication/eur-scientific-and-technical-research-reports/smart-transport-cities-future-transportation.

ДОДАТОК Г

Демонстраційний матеріал

Формат	Позиція.	ПОЗНАЧЕННЯ	НАЙМЕНУВАННЯ	Кількість	Примітки
			Текстові документи		
А4	1	ГЮІК 464430.043 ПЗ			
			Пояснювальна записка	75	с.
			Інші документи		
А4	2		Псевдокоди алгоритмів	3	с.
А4	3		Код програми	11	с.
А4	4		Апробація результатів наукових досліджень	3	с.
.ppt	5		Демонстраційний графічний матеріал		

Змін.	Арк	Номер докум.	Підпис	Дата	ГЮІК 464430.043 ВД			
Розроб.		Маслов О. А.			Розроблення інтелектуальної системи керування потоком автотранспорту в динамічних умовах Відомість кваліфікаційної роботи	Літера	Аркуш	Аркуші
Перевір.		Демська Н. П.				Н	1	1
						Кафедра КІТАР ХНУРЕ		
Н.контр.		Стародубцев М. Г.						
Затв.		Невлюдов І. Ш.						