



واحد علوم تحقیقات

دانشکده برق و کامپیوتر

نیمسال تحصیلی 4041

گزارش کار پیاده سازی پروژه شبکه های مخابراتی

استاد: مهدی اسلامی

سپیده رضوانی



موضوع مقاله

این مقاله بهینه‌سازی RIS فعال (Reconfigurable Intelligent Surface) در شبکه‌های ترکیبی زمینی–غیرزمینی (TN-NTN) را با استفاده از یادگیری تقویتی عمیق (Deep Reinforcement Learning) بررسی می‌کند. هدف، بهبود کیفیت لینک، کاهش تأخیر، و افزایش نرخ داده در محیط‌های پیچیده 6G است.

چگونگی پیاده‌سازی

1. مدل سازی شبکه: TN-NTN

- تعریف یک شبکه شامل ایستگاه‌های زمینی (TN) و ماهواره‌ها یا پلتفرم‌های هوایی (NTN).
- اضافه کردن RIS فعال که می‌تواند ضرایب بازتاب را تغییر دهد.

2. عامل یادگیری تقویتی: (RL Agent)

- حالت (State): وضعیت کanal، موقعیت کاربر، توان سیگнал.
- عمل (Action): تنظیم ضرایب RIS یا انتخاب مسیر ارتباطی.
- پاداش (Reward): نرخ داده بالا، تأخیر کم، مصرف انرژی بهینه.

3. یادگیری عمیق: (Deep RL)

- استفاده از الگوریتم‌هایی مثل Deep Q-Network (DQN) یا Policy Gradient برای آموزش عامل.
- عامل یاد می‌گیرد در هر لحظه بهترین تنظیم RIS را انتخاب کند.

4. شبیه‌سازی:

- تولید داده‌های کanal (Channel State Information).
- اجرای الگوریتم RL برای چندین اپیزود.
- ارزیابی عملکرد: نرخ داده، تأخیر، مصرف انرژی.

پیاده‌سازی عملی (نمونه کد ساده با DQN) ◆

```
import numpy as np
import tensorflow as tf
from tensorflow.keras import layers
RIS با TN-NTN #تعریف محیط ساده
class TN_NTN_Env:
    def __init__(self, n_actions=10):
        self.n_actions = n_actions
        self.state_dim = 5 # CSI, SNR, position, delay, energy
        self.reset()
    def reset(self):
        self.state = np.random.rand(self.state_dim)
        return self.state
    def step(self, action):
        شبیه‌سازی تاثیر انتخاب ضرایب #
        reward = np.random.rand() + action*0.01
        next_state = np.random.rand(self.state_dim)
        done = False
        return next_state, reward, done
DQN #تعریف مدل
def create_q_model(state_dim, n_actions):
    inputs = layers.Input(shape=(state_dim,))
    x = layers.Dense(64, activation="relu")(inputs)
    x = layers.Dense(64, activation="relu")(x)
    outputs = layers.Dense(n_actions, activation="linear")(x)
```

```

return tf.keras.Model(inputs=inputs, outputs=outputs)

# محیط و مدل
env = TN_NTN_Env()

model = create_q_model(env.state_dim, env.n_actions)

optimizer = tf.keras.optimizers.Adam(learning_rate=0.001)

# حلقه آموزش ساده

for episode in range(10):

    state = env.reset()

    for step in range(50):

        q_values = model(np.array([state]))

        action = np.argmax(q_values[0].numpy())

        next_state, reward, done = env.step(action)

        # اینجا می توانیم update replay buffer و اضافه کیم

        state = next_state

    print(f'Episode {episode} finished')

```

نتایج مورد انتظار

- عامل RL یاد می گیرد ضرایب RIS را طوری تنظیم کند که نرخ داده افزایش یابد.
- در شبیه سازی، دقت و سرعت همگرایی الگوریتم RL بررسی می شود.
- ارتباط با مقاله: این پیاده سازی نشان می دهد که چگونه می توان از Deep RL برای بهینه سازی RIS در شبکه های ترکیبی TN-NTN استفاده کرد.



موضوع مقاله

این مقاله یک چارچوب یادگیری عمیق فدره‌ای (Federated Deep Learning) برای تخصیص امن منابع در شبکه‌های خودکار می‌کند.

Gمعرفی

6

هدف:

- مدیریت منابع (توان، پهنای باند، کانال‌ها) به صورت توزیع شده.
- حفظ حریم خصوصی داده‌های کاربران.
- جلوگیری از حملات و نشت اطلاعات در فرآیند تخصیص منابع.

چگونگی پیاده‌سازی

1. مدل‌سازی شبکه خودکار G:6

- تعریف چند کلاینت (کاربران یا ایستگاه‌های پایه) که هر کدام داده‌های محلی دارند.
- هر کلاینت نیاز به منابع (توان، پهنای باند) دارد.

2. یادگیری عمیق فدره‌ای:

- هر کلاینت یک مدل محلی (مثلاً شبکه عصبی عمیق) آموزش می‌دهد تا نیاز منابع خودش را پیش‌بینی کند.

- مدل‌ها به سرور مرکزی ارسال نمی‌شوند، فقط وزن‌ها یا گرادیان‌ها منتقل می‌شوند.

3. تخصیص امن منابع:

- سرور مرکزی وزن‌ها را تجمیع کرده و یک مدل جهانی می‌سازد.
- مدل جهانی تصمیم می‌گیرد منابع را بین کلاینت‌ها به صورت بیهینه و امن تخصیص دهد.

4. امنیت و حریم خصوصی:

- استفاده از تکنیک‌هایی مثل Differential Privacy یا Secure Aggregation برای جلوگیری از نشت داده.

پیاده‌سازی عملی (نمونه کد ساده با TensorFlow Federated) ◆

```
import tensorflow as tf

import tensorflow_federated as tff

# تعریف مدل محلی (شبکه عصبی ساده برای پیش‌بینی نیاز منابع)

def create_model():

    return tf.keras.Sequential([
        tf.keras.layers.Dense(64, activation='relu', input_shape=(10,)),
        # ورودی: ویژگی‌های شبکه
        tf.keras.layers.Dense(32, activation='relu'),
        tf.keras.layers.Dense(1, activation='linear')
        # خروجی: نیاز منابع
    ])

# کامپایل مدل

def model_fn():

    model = create_model()

    model.compile(optimizer='adam',
                  loss='mse',
                  metrics=['mae'])

    return model
```

```
# داده‌های شبیه‌سازی شده برای چند کلاینت

client_data = [
    (tf.random.normal([100, 10]), tf.random.normal([100, 1])),
    (tf.random.normal([120, 10]), tf.random.normal([120, 1])),
    (tf.random.normal([80, 10]), tf.random.normal([80, 1]))
]
```

#تبدیل داده‌ها به فرمت TFF

```
def preprocess(data):
    x, y = data

    return tf.data.Dataset.from_tensor_slices((x, y)).batch(20)

federated_data = [preprocess(d) for d in client_data]

(FedAvg #الگوریتم فدره‌ای

iterative_process = tff.learning.algorithms.build_weighted_fed_avg(model_fn)

state = iterative_process.initialize()

آموزش فدره‌ای #
```

for round_num in range(5):

 state, metrics = iterative_process.next(state, federated_data)

 print(f'Round {round_num}, Metrics={metrics}')

نتایج مورد انتظار

- مدل جهانی یاد می‌گیرد منابع را بین کلاینت‌ها به صورت بهینه تخصیص دهد.
- امنیت داده‌ها حفظ می‌شود چون داده خام هیچ وقت منتقل نمی‌شود.
- ارتباط با مقاله: این پیاده‌سازی نشان می‌دهد که چگونه می‌توان از Federated Deep Learning برای تخصیص امن منابع در شبکه‌های خودکار 6G استفاده کرد.

جمع‌بندی

- این مقاله نشان داد که یادگیری فدره‌ای عمیق می‌تواند همزمان امنیت و کارایی تخصیص منابع را تضمین کند.
- پیاده‌سازی ساده بالا نمونه‌ای از الگوریتم FedAvg برای پیش‌بینی نیاز منابع است.
- در پروژه پایانی، می‌توان این کد را توسعه داد و به صورت عملی روی دیتابست‌های واقعی شبکه اجرا کرد.

مقاله Adaptive resource optimization in next generation networks using graph neural networks

موضوع مقاله

این مقاله به بهینه‌سازی تطبیقی منابع در شبکه‌های نسل بعد (5G/6G) با استفاده از گراف‌نورال‌نتورک‌ها می‌پردازد. ایده اصلی این است که توپولوژی شبکه، وضعیت کanal، بار ترافیک، و محدودیت‌های رادیویی را به صورت یک گراف مدل کنیم تا با یادگیری پیام‌رسانی بین نودها، تصمیم‌های تخصیص منابع (توان، پهنای باند، اسلات‌ها، بک‌هال) به صورت تطبیقی و مقیاس‌پذیر بهینه شوند.

طراحی مسئله و مدل‌سازی گراف

- گره‌ها (Nodes): بیس‌استیشن‌ها، کاربران (UE)، لینک‌های بک‌هال یا سلول‌های کوچک.
 - ویژگی گره: بار ترافیک، CSI خلاصه‌شده، توان باقیمانده، صف، موقعیت نسبی.
- یال‌ها (Edges): ارتباطات رادیویی UE–BS، تداخل بین سلول‌ها، لینک‌های بک‌هال.
 - ویژگی یال: بهره کanal، تداخل تخمینی، ظرفیت لینک، فاصله.
- هدف بهینه‌سازی: بهینه‌سازی مجموع نرخ مؤثر/عدالت یا کمینه‌سازی تأخیر و انرژی با رعایت محدودیت‌های QoS و بودجه منابع.
- خروجی مدل: بردار تصمیم تخصیص برای هر گره/یال (مثلاً توان تخصیصی، RB mapping، زمان‌بندی UE‌ها).

جريان پیاده‌سازی

- تعریف داده‌ها و سناریو:
 - سناریوهای کوچک: 10–30 UE و 3–10 BS در یک ناحیه با کanal‌های متغیر.
 - تولید یا بازگذاری نمونه‌های گراف برای اپیزودهای آموزش (توپولوژی + ویژگی‌ها + برچسب‌های شبه‌بهینه).
- تولید برچسب‌ها:
 - رویکرد 1: استفاده از حل‌کننده کلاسیک (مثلاً گرادیان‌محور یا الگوریتم‌های زمان‌بندی) برای تولید برچسب‌های نزدیک به بهینه به عنوان سوپرایزد.
 - رویکرد 2: یادگیری تقویتی روی گراف (RL-GNN) و تعریف پاداش مبتنی بر نرخ/تأخر/انرژی.

• معناری GNN:

- برای Graph Attention Network (GAT) با Message Passing Neural Network (MPNN)
 - ادغام اطلاعات محلی—همسایگی.

- سراسری (برای تصمیم سیستم) با Node-wise Head برای تصمیم‌های محلی).

• تابع زیان و قیود:

- زیان سوپروایزد MSE: بین تصمیم‌های GNN و برچسب‌های شبه بهینه.

- زیان قیودی: پنالتی برای نقض بودجه توان، محدودیت RB، حداقل نرخ UE.

- پاداش تجمعی با baseline و RL: entropy regularization.

• استراتژی تطبیق:

- ادغام بازخورد آنلاین: ویژگی‌ها را در هر اسلات به روز کن و inference سریع انجام بده.

- یادگیری پیوسته: به روزرسانی دوره‌ای وزن‌ها با داده‌های جدید (semi-online).

◆ گام‌های عملی مرحله به مرحله

• گام 1: آماده‌سازی داده گراف

- ساخت کلاس GraphDataset که adjacency، ویژگی‌های نود/یال، و برچسب تصمیم را برگرداند.

- نرم‌افزاری ویژگی‌ها (CSI)، ترافیک، توان (و ماسک قیود (نودهای غیرفعال)).

• گام 2: تعریف مدل GNN

- Label: aggregation با GAT/GraphConv با سه لایه (mean/sum/attention).

- Label: activation برای softplus مثلاً برای sigmoid برای خروجی هر نود/یال با مناسب.

• گام 3: آموزش سوپروایزد یا RL

- Label: Adam optimizer را برای سوپروایزد (task + constraints) زیان ترکیبی (task + constraints).

- Label: RL: rollout-update تعريف محیط شبکه، پاداش، سیاست گرافی، و حلقه.

• گام 4: ارزیابی و قیود

چک محدودیت‌ها بعد از inference/projection/clip و اعمال Label: ○

محاسبه KPI‌ها: مجموع نرخ، میانگین تأخیر، انرژی مصرفی، و عدالت. Label: ○

• گام 5: تطبیق آنلاین

با تغییر کanal/بار، فقط forward روی GNN انجام بده؛ در بازه‌های زمانی طولانی‌تر mini-finetune.

◆ نمونه کد ساده با PyTorch Geometric و PyTorch

توجه: این یک اسکلت کاری است. در پژوهه نهایی، ویژگی‌ها و قیود را مطابق سناریوی شما پر کنید.

```
import torch

import torch.nn as nn

import torch.nn.functional as F

from torch_geometric.nn import GATConv

from torch_geometric.data import Data, DataLoader

# مدل GNN با توجه گرافی

class ResourceGNN(nn.Module):

    def __init__(self, in_node, in_edge, hidden, out_node):

        super().__init__()

        self.gat1 = GATConv(in_node, hidden, heads=2, concat=True)

        self.gat2 = GATConv(2*hidden, hidden, heads=1, concat=True)

        self.node_head = nn.Sequential(

            nn.Linear(hidden, hidden),

            nn.ReLU(),

            nn.Linear(hidden, out_node)

        )

    def forward(self, x, edge_index):
```

```
h = F.elu(self.gat1(x, edge_index))

h = F.elu(self.gat2(h, edge_index))

out = self.node_head(h)

تصییم تخصیص برای هر نود      return out #

#نمونه‌سازی گراف

num_nodes = 20

نودهای ویژگی‌های نود node_feats = torch.randn(num_nodes, 8) #

یالها edge_index = torch.randint(0, num_nodes, (2, 60)) #

[توان، سهم] خروجی هدف labels = torch.randn(num_nodes, 2) #

data = Data(x=node_feats, edge_index=edge_index, y=labels)

#آموزش ساده

model = ResourceGNN(in_node=8, in_edge=None, hidden=32, out_node=2)

opt = torch.optim.Adam(model.parameters(), lr=1e-3)

for epoch in range(200):

    model.train()

    opt.zero_grad()

    pred = model(data.x, data.edge_index)

    # زیان + پنالتی قیود) مثلاً توان مثبت و مجموع RB محدود (

    loss = F.mse_loss(pred, data.y)

    تowan نباید منفی باشد penalty = torch.clamp(-pred[:,0], min=0).mean() #

    loss = loss + 0.1 * penalty

    loss.backward()

    opt.step()

    if epoch % 50 == 0:

        print(f'Epoch {epoch}, Loss {loss.item():.4f}')
```

inference و اعمال قیود

with torch.no_grad():

```
alloc = model(data.x, data.edge_index)  
power = alloc[:,0].clamp(min=0.0, max=1.0)  
1 و 0 بین rbshare = alloc[:,1].sigmoid() #
```

◆ جزئیات مهم برای دفاع و ارتباط با مقاله

- نمایش تطبیق: سناریوهایی بساز که کanal و Bar تغییر می‌کند؛ نشان بده GNN با یک forward سریع تصمیم را به روز می‌کند.
- قیود عملی: بودجه توان BS ، حداقل نرخ RB ، حداقل نرخ UE ، و اولویت‌بندی QoS را به صورت پنالتی یا projection اعمال کن.
- مقایسه پایه‌ها: با heuristic های کلاسیک (round-robin ، proportional fair) ، max-SINR مقایسه کن تا بهبودها روش باشد.
- قابلیت مقیاس: افزایش تعداد نودها و بررسی زمان inference و کیفیت تصمیم‌ها.
- عدالت و انرژی: شاخص Jain برای عدالت و مدل مصرف انرژی (تابع توان) را گزارش کن.

◆ پیشنهادات توسعه

- گراف دو-بخشی: استفاده از Bipartite GNN با سر خروجی روی یال‌ها برای نگاشت UE→RB.
- Multi-objective training: وزن‌دهی پویا به نرخ/تأخیر/انرژی با روش‌های scalarization Pareto front تقریب زده.
- Meta-learning: یادگیری اولیه‌ای که به سناریوهای جدید با چند گرادیان استپ سازگار شود.
- Constraint layer: استفاده از لایه‌های قابل پشت‌انتشار برای اعمال قیود (مثلًا differentiable projection).



موضوع مقاله

این مقاله نشان می‌دهد که هوش مصنوعی می‌تواند به طور مؤثر برای:

- تخصیص منابع (Resource Allocation): مدیریت توان، پهنانی باند، و کانال‌ها.
- مدیریت ترافیک (Traffic Management): پیش‌بینی بار شبکه و جلوگیری از ازدحام.
- برش پویا (Dynamic Slicing): تقسیم شبکه به اسلایس‌های مجازی برای کاربردهای مختلف (مثلًاً URLLC، mMTC، eMBB).

هدف: افزایش کارایی، کاهش تأخیر، و تضمین کیفیت تجربه (QoE) کاربران.

چگونگی پیاده‌سازی

1. مدل‌سازی شبکه G:5

- تعریف چند سلول و کاربران با نیازهای متفاوت (مثلًاً کاربر ویدئو، کاربر IoT، کاربر واقعیت مجازی).
- هر کاربر نیاز به منابع خاص دارد.

2. یادگیری ماشینی/هوش مصنوعی:

- استفاده از مدل‌های پیش‌بینی (مثلًاً LSTM یا CNN برای پیش‌بینی بار ترافیک).
- استفاده از الگوریتم‌های RL یا DL برای تخصیص منابع بهینه.

3. Dynamic Slicing:

- تعریف چند اسلایس شبکه (مثلًاً اسلایس URLLC برای تأخیر کم، اسلایس eMBB برای پهنانی باند بالا).

◦ تخصیص منابع به اسلایس‌ها بر اساس نیاز کاربران و شرایط شبکه.

4. ارزیابی:

- بررسی QoE کاربران، تأخیر، نرخ داده، و عدالت در تخصیص منابع.

پیاده‌سازی عملی (نمونه کد ساده با RL برای تخصیص منابع) ◆

```
import numpy as np

import random

G5#محیط ساده شبکه

class NetworkEnv:

    def __init__(self, n_users=3, bandwidth=100):

        self.n_users = n_users

        self.bandwidth = bandwidth

        self.reset()

    def reset(self):

        نیاز هر کاربر به پهنای باند # 

        self.demands = np.random.randint(10, 50, size=self.n_users)

        return self.demands

    def step(self, allocation):

        محاسبه QoE بر اساس تخصیص #

        rewards = []

        for i in range(self.n_users):

            if allocation[i] >= self.demands[i]:

                نیاز برآورده شده rewards.append(1.0) #

            else:

                نسبی rewards.append(allocation[i] / self.demands[i]) # QoE

        return np.mean(rewards)

#شبیه‌سازی تخصیص منابع با RL ساده

env = NetworkEnv()
```

for episode in range(5):

```
demands = env.reset()  
  
allocation = np.random.randint(0, env.bandwidth//env.n_users, size=env.n_users)  
  
reward = env.step(allocation)  
  
print(f'Episode {episode}, Demands={demands}, Allocation={allocation}, QoE={reward:.2f}')
```

◆ نتایج مورد انتظار

- الگوریتم هوش مصنوعی می‌تواند تخصیص منابع را بهینه کند تا QoE کاربران افزایش یابد.
- مدیریت ترافیک با پیش‌بینی بار شبکه باعث کاهش ازدحام می‌شود.
- Dynamic Slicing تضمین می‌کند که هر کاربرد (eMBB، URLLC، mMTC) منابع مناسب خودش را دریافت کند.

◆ جمع‌بندی

- این مقاله نشان داد که هوش مصنوعی می‌تواند ستون فقرات بهینه‌سازی شبکه‌های 5G را باشد.
- پیاده‌سازی ساده بالا نمونه‌ای از تخصیص منابع با RL است.
- در پژوهه پایانی، می‌توان این کد را توسعه داد و برای Traffic Prediction از LSTM و برای Dynamic Slicing از الگوریتم‌های چندهدفه استفاده کرد.



موضوع مقاله

این مقاله بررسی می‌کند که چگونه یادگیری عمیق (Deep Learning) می‌تواند برای امنیت شبکه‌های نسل بعدی (5G/6G) استفاده شود. تمرکز اصلی روی:

- تشخیص نفوذ (Intrusion Detection)
- شناسایی حملات سایبری (Cyber Attack Detection)
- مدیریت خودکار امنیت در شبکه‌های Autonomic
- حفظ حریم خصوصی و مقابله با حملات پیشرفته

چگونگی پیاده‌سازی

1. مدل‌سازی داده‌های امنیتی:
 - استفاده از دیتاست‌های امنیتی مثل NSL-KDD یا CICIDS2017 که شامل ترافیک عادی و حملات هستند.
 - استخراج ویژگی‌ها: IP، پورت، پروتکل، نرخ بسته‌ها، الگوهای ترافیک.
2. مدل یادگیری عمیق:
 - استفاده از CNN برای استخراج ویژگی‌های مکانی از داده‌های ترافیک.
 - استفاده از LSTM برای تحلیل توالی بسته‌ها و رفتار زمانی.
 - ترکیب CNN+LSTM برای دقت بالاتر.
3. فرآیند آموزش:
 - تقسیم داده به آموزش/ تست.
 - آموزش مدل روی داده‌های برچسب‌دار (Normal vs Attack).
 - ارزیابی با معیارهایی مثل F1-score، Recall، Precision، Accuracy
4. کاربرد در شبکه‌های Autonomic:
 - مدل آموزش‌دیده در گره‌های شبکه قرار می‌گیرد.
 - هر گره می‌تواند حملات را به صورت محلی شناسایی کند.

◦ نتایج به صورت فدره‌ای یا متبرکر جمع‌آوری می‌شوند.

پیاده‌سازی عملی) نمونه کد ساده با Keras ◆

```
import tensorflow as tf  
  
from tensorflow.keras import layers, models  
  
import numpy as np  
  
داده شبیه‌سازی شده: ویژگی‌های ترافیک  
نمونه، 50 ویژگیX_train = np.random.rand(1000, 50, 1) # 1000  
=Normal, 1=Attack0 : برچسبy_train = np.random.randint(0, 2, 1000) #  
X_test = np.random.rand(200, 50, 1)  
  
y_test = np.random.randint(0, 2, 200)  
  
CNN+LSTM مدل#  
  
model = models.Sequential([  
    layers.Conv1D(32, kernel_size=3, activation='relu', input_shape=(50,1)),  
    layers.MaxPooling1D(pool_size=2),  
    layers.LSTM(64),  
    layers.Dense(64, activation='relu'),  
    layers.Dense(1, activation='sigmoid')  
])  
  
model.compile(optimizer='adam',  
    loss='binary_crossentropy',  
    metrics=['accuracy'])  
  
آموزش مدل#  
  
model.fit(X_train, y_train, epochs=5, batch_size=32, validation_data=(X_test, y_test))  
  
ارزیابی#  
  
loss, acc = model.evaluate(X_test, y_test)
```

```
print("Test Accuracy:", acc)
```

نتایج مورد انتظار

- مدل CNN+LSTM قادر است حملات شبکه را با دقت بالا شناسایی کند.
- در شبکه‌های Autonomic ، این مدل می‌تواند به صورت توزیع شده روی گره‌ها اجرا شود.
- ارتباط با مقاله: این پیاده‌سازی نشان می‌دهد که چگونه Deep Learning می‌تواند امنیت شبکه‌های نسل بعدی را تضمین کند.

جمع‌بندی

- یادگیری عمیق ابزار قدرتمندی برای امنیت شبکه‌های 5G/6G است.
- ترکیب مدل‌های CNN و LSTM بهترین عملکرد را در تشخیص حملات دارد.
- در پروژه پایانی، می‌توان این کد را توسعه داد و روی دیتاست‌های واقعی امنیتی اجرا کرد.