گزارش پروژه

سمیرا مرادزاده و سنا آریا
دانشکده علوم ریاضی
دکتر بیژن احمدی
دکتر زهرا طاهری

چکیده

هدف اصلی در این پروژه پیدا کردن مقادیر بهینه برای هایپر پارامترهای یک مدل شبکه عصبی میباشد. منظور از مقدار بهینه در واقع مقادیری است که به ازاء آن ها مدل ما بهترین نتیجه را بدهد.

مقدمه

داده هایی که در این پروژه استفاده کردیم داده های کلسیفیکیشن هستند به این معنا که ستون مربوط به لیبل ها فقط صفر و یک هستند. این داده ها دارای ۲۰۳۹ سطر و ۲۰۳ تا ستون هستند، با توجه به نوع دو تا از ستون ها که جنس آن ها استرینگ میباشد آن دو ستون را حذف میکنیم و در نهایت کار را با ۲۰۱ ستون آغاز میکنیم. بعد از اینکه مطمئن شدیم دیتا های مورد استفاده مقدار خالی ندارد، فیچر ها را از لیبل ها جدا کردیم. حواسمان هست که دیتا ها را به صورت استاندارد اسکیل کنیم.

اما از آنجایی که داده های مورد استفاده ما مقادیرش بین ۰ و ۱ هستند نیازی به این کار نبود حال کل ثبت های داده را به دو بخش آموزشی و آزمون تقسیم میکنیم. اما دقت کنید با توجه به نا متقارن بودن تعداد ۰ و ۱ ها در ستون مربوط به لیبل ها مجبور بودیم تا به صورت معنی داری این تقسیم بندی را انجام دهیم.

و در آخر در بخش مقدمه و آماده سازی دیتا ها کافیست با دستور زیر آنها را به فرمتی برای ورود به تورچ آماده کردیم.

```
# transform to torch tensor

tensor_x = torch.tensor(train_data, dtype=torch.float).to(device)
tensor_x2 = torch.tensor(test_data, dtype=torch.float).to(device)

tensor_y = torch.tensor(train_labels, dtype=torch.float).to(device)
tensor_y2 = torch.tensor(test_labels, dtype=torch.float).to(device)
```

```
data.isnull().sum().sum()
```

```
0
```

```
Xdata = data.iloc[:, 0:200]
Ydata = data.iloc[:, 200:201]
```

```
from sklearn.model_selection import train_test_split
train_data, test_data, train_labels, test_labels = train_test_split(Xdata, Ydata, stratify = Ydata , test_size = 0.25)
```

متریک استفاده شده:

ما در این پروژه از متریک Roc-auc استفاده کردیم به این دلیل که برای این داده های از جنس رده بندی این متریک مرسوم تر بوده و همچنین نتیجه بهتری را در نهایت برایمان داشته است.

```
from sklearn.metrics import roc_auc_score

def compute_score(model, data_loader, device="cpu"):
    model.eval()
    metric = roc_auc_score
    with torch.no_grad():
        prediction_all= torch.empty(0, device=device)
        labels_all= torch.empty(0, device=device)
        for i, (feats, labels) in enumerate(data_loader):
            feats=feats.to(device)
            labels=labels.to(device)
            prediction = model(feats).to(device)
            prediction = torch.sigmoid(prediction).to(device)
            prediction_all = torch.cat((prediction_all, prediction), 0)
            labels_all = torch.cat((labels_all, labels), 0)
        try:
            t = metric(labels_all.int().cpu(), prediction_all.cpu()).item()
        except ValueError:
            t = 0
    return t
```

مدل اولیه:

مدلی که در ابتدا تعریف کردیم یک شبکه عصبی با چهار لایه و سه لایه پنهان میباشد و در این قسمت بعد لایه و اکتیویشن فانکشن را به عنوان هایپر پارامتر در نظر گرفتیم. اما ممکن است این بهترین مدل نباشد زیرا ما میتوانیم تعداد لایه ها را نیز به عنوان هایپر پارامتر در نظر بگیریم بنابراین ممکن است تا آخر این گزارش بتوان نتیجه گرفت که تعداد بیشتر یا کمتر این لایه ها برای این داده ها و این پروژه بهتر خواهد بود.

```
class Net(nn.Module):
    def __init__(self, config):
        super().__init__()

        self.config = config
        self.hidden_dim1 = int(self.config.get("hidden_dim1", 100))
        self.hidden_dim2 = int(self.config.get("hidden_dim2", 100))
        self.hidden_dim3 = int(self.config.get("hidden_dim3", 100))

        self.act1 = self.config.get("act1", "relu")
        self.act2 = self.config.get("act2", "relu")
        self.act3 = self.config.get("act3", "relu")

        self.linear1 = nn.Linear(200, self.hidden_dim1)
        self.linear2 = nn.Linear(self.hidden_dim1, self.hidden_dim2)
        self.linear3 = nn.Linear(self.hidden_dim2, self.hidden_dim3)
        self.linear4 = nn.Linear(self.hidden_dim3, 1)

    @staticmethod
    def activation_func(act_str):
        if act_str=="tanh":
            return eval("torch."+act_str)
        elif act_str=="selu" or act_str=="relu":
            return eval("torch.nn.functional."+act_str)

    def forward(self, x):
        output = self.linear1(x)
        output = self.activation_func(self.act1)(output)
        output = self.linear2(output)
        output = self.activation_func(self.act2)(output)
        output = self.linear3(output)
        output = self.activation_func(self.act3)(output)
        output = self.linear4(output)
        return output
```

تابع trainable :

در این قسمت مقدار چک پوینت را تعریف میکنیم و کار این تابع این است که مقدار کانفیگ های مورد بررسی و مقدار اسکور را سیو میکند و در هر بار اجرا گرفتن اسکور را با مقدار قبلی مقایسه میکند و اگر بهتر باشد مقدار جدید را نگه میدارد.

```
if checkpoint_dir:
    model_state, optimizer_state = torch.load(
        os.path.join(checkpoint_dir, "checkpoint"))
    net.load_state_dict(model_state)
    optimizer.load_state_dict(optimizer_state)
```

و در ادامه از مجموعه آموزشی مقداری را به عنوان مجموعه ولیدیشن جدا میکنیم.

```
# Split the dataset into training and validation sets
train_size = int(len(trainset) * 0.66)
train_subset, val_subset = random_split(trainset, [train_size, len(trainset) - train_size])
```

و در نهایت حلقه هایی را تعریف میکنیم تا به مقدار تعداد اپک ها اسکور را روی مجموعه آموزشی و ولیدیشن یا اعتبار سنجی محاسبه کند.

```
for epoch in range(epochs):  # loop over the dataset multiple times
    epoch_train_loss = 0.0
    # epoch_steps = 0
    net.train() # Prepare model for training
    for i, data in enumerate(trainloader):
        # get the inputs; data is a list of [inputs, labels]
        inputs, labels = data
        inputs, labels = inputs.to(device), labels.to(device)
```

حال وقت آن است که مقدار اسکور را روی مجموعه تست تعریف کنیم و به عنوان خروجی بخواهیم آن را چاپ کند.

```
def test_score(config, net, device="cpu"):
    trainset, testset = load_data()
    best_trial=config
    testloader = torch.utils.data.DataLoader(
            testset,
            batch_size=int(best_trial["batch_size"]),
            shuffle=False,
            num_workers=2)
    test_score = compute_score(net, testloader, device)
    print("Best trial test set score: {}".format(test_score))
```

تابع اصلی:

این تابعی است که اصلی ترین کار را انجام میدهد ابتدا مجموعه ای از هایپر پارامتر ها را با نام کانفیگ تعریف میکنیم. و مقادیری را به آن میدهیم تا با پای تورچ بهترین آن ها را بیابد.

الگوریتم جستجو:

این الگوریتم در واقع به دنبال بهترین فضای جستجو هست و در واقع فضایی که احتمال وجود نقطه اکترمم در آن بیشتر باشد را می یابد.

که البته ما از الگوریتم های مختلف استفاده کردیم و نتایج را با هم مقایسه کردیم و در نهایت به الگوریتم با بیشترین میزان دقت مورد استفاده قرار دادیم.

```
# Optuna search algorithm
from ray.tune.suggest.optuna import OptunaSearch
from ray.tune.suggest import ConcurrencyLimiter
search_alg = OptunaSearch(
    metric="score", #or accuracy, etc.
    mode="max", #or max
    #seed = 42,
    # points_to_evaluate=[
    # {'lr': 0.0005, 'hidden_size': 150.0, 'readout1_out': 200.0, 'readout2_out': 180.0}
    # ],
    )
```

اسکجولر ها:

در واقع این قسمت کمک میکند تا از انجام محاسبات اضافی و صرفه جویی کند.

در این قسمت هم اسکجولر های مختلفی میشد که استفاده کرد و ما تغییر آن ها متوجه شدیم که بهترین نتیجه را اسکجولر ASHA دارد.

```
search_alg = ConcurrencyLimiter(search_alg, max_concurrent=10)

scheduler = ASHAScheduler(
    metric ="score",
    mode="max",
    max_t=max_num_epochs,
    reduction_factor=2,
    grace_period=4,
    brackets=5
    )
```

نتیجه نهایی:

اسکور نهایی و بهترین هایپر پارامتر ها را مشاهده میکنیم.

```
2022-07-25 15:19:52,720 INFO tune.py:748 -- Total run time: 185.04 seconds (184.74 seconds for the tuning loop).
Best trial config: {'act1 ': 'tanh', 'act2': 'relu', 'act3': 'tanh', 'lr': 0.0006000000000000001, 'batch_size': 8, 'hidden_dim1': 90.0, 'hidden_dim2': 100.0, 'hidden_dim3': 80.0}
Best trial final validation score: 0.7902226157348847
Best trial test set score: 0.8109829059829061
```

تابع نهایی:

برای نتیجه گیری دقیق تر لازم است تا مدل شبکه عصبی با کانفیگ هایی که در نتیجه قبل مشاهده میکنیم تعریف کنیم و دقت را روی مجموعه تست مشاهده کنیم.

با کانفیگ های قبلی مدل را تعریف کردیم اما نتیجه ای با دقت نزدیک به ۵۰ درصد را مشاهده کردیم که خب دقت خیلی خوبی به نظر نمیرسد.

```
Finished Training
Best trial test set score: 0.5055662393162393
```

برای رفع احتمالی مشکل تصمیم گرفتیم تا تغییری در تعداد اپک ها بدیم و مقدار آن را از ۱۰ به ۵۰ تغییر دادیم، بعد از آن به ۳۰ اما مشاهده کردیم این بهترین نتیجه ای بود که روی این داده ها به دست آوردیم.

اما احتمالا بتوان با تغییر مقدار هایپر پارامتر ها و یا حتی تبدیل تعداد لایه ها به هایپر پارامتر و اپتیمایز کردن آن، نتیجه بهتری گرفت اما به زمان بیشتری برای آموزش این بخش نیاز داشتیم. امیدواریم بتوانیم در آینده نتیجه را بهبود ببخشیم.

توجه:

ممکن است نتایجی که در نوت بوک قرار داده شده بر روی این صفحه، به دست آمده است با نتایجی که عکس آن را در گزارش مشاهده میکنید متفاوت باشد که دلیل آن رندم بودن وزن های اولیه شبکه عصبی در هر دور از اجرا کردن کد باشد.

و ما در پروژه عکس بهترین نتیجه را قرار دادیم.