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 رگرسیون هستند به این معنا که ستون مربوط به لیبل ها فقط مقداری عددی هستند. این داده ها دارای ۱۱۲۸ سطر و ۲۰۳تا 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 در فیچر ها ما مقادیرش بین ۰ و ۱ هستند نیازی به این کار نبود ولی داده های مربوط به لیبل ها را اسکیل کردیم. حال کل ثبت های داده را به دو بخش آموزشی و آزمون تقسیم میکنیم.

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.

```
from sklearn.model_selection import train_test_split
X_train, X_test, Y_train, Y_test = train_test_split(Xdata, Ydata, test_size=0.25, random_state=1)
```

```
from sklearn import preprocessing
scaler_labels = preprocessing.MinMaxScaler()
Y_train = scaler_labels.fit_transform(Y_train.values.reshape(-1, 1))
Y_test = scaler_labels.transform(Y_test.values.reshape(-1, 1))
```

```
X_train=np.array(X_train)
X_test=np.array(X_test)
```

```
# transform to torch tensor

tensor_x = torch.tensor(X_train, dtype=torch.float).to(device)
tensor_x2 = torch.tensor(X_test, dtype=torch.float).to(device)

tensor_y = torch.tensor(Y_train, dtype=torch.float).to(device)
tensor_y2 = torch.tensor(Y_test, dtype=torch.float).to(device)
```

متریک استفاده شده:MSEloss

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

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.

```python
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.

```python
    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.

```python
    # 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.

```python
    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)

            # zero the parameter gradients
            optimizer.zero_grad()

            # forward + backward + optimize
            outputs = net(inputs)
            loss = criterion(outputs, labels)
            loss.backward()
            optimizer.step()

```

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

```python
def test_score(config, net, device="cpu"):
    trainset, testset = load_data()

    testloader = torch.utils.data.DataLoader(
        testset, batch_size=int(config.get("batch_size",32)), shuffle=False, num_workers=2)

    ## Regression
    criterion = nn.MSELoss(reduction='sum')

    # Test loss
    test_loss = 0.0
    net.eval() # Prepare model for evaluation
    for i, data in enumerate(testloader):
        with torch.no_grad():
            inputs, labels = data
            inputs, labels = inputs.to(device), labels.to(device)

            outputs = net(inputs)

            # Inverse transform of the labels' scaler
            outputs = torch.tensor(scaler_labels.inverse_transform(outputs.detach().cpu())).to(device)
            labels = torch.tensor(scaler_labels.inverse_transform(labels.cpu())).to(device)

            loss = criterion(outputs, labels)
            test_loss += loss.cpu().numpy()

    return test_loss / len(testset)
```

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 اکسترمم 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 optuna با بیشترین میزان دقت مورد استفاده قرار دادیم.

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

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

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.

```python
    scheduler = ASHAScheduler(
        metric ="loss",
        mode="min",
        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.

```
2022-07-26 12:05:44,819 INFO tune.py:748 -- Total run time: 28.26 seconds (27.89 seconds for the tuning loop).
Best trial config: {'act1 ': 'relu', 'act2': 'tanh', 'act3': 'selu', 'lr': 0.001, 'batch_size': 16, 'hidden_dim1': 90.0, 'hidden_dim2': 190.0, 'hidden_dim3': 150.0}
Best trial final validation score: 0.4359037326724615
Best trial test set score: 0.45816472594560614
```

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..

```
Finished Training
0.3979180043482341
```

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.