به نام خدا

شرح پروژه و کد

پروژه در پایتون و به کمک کتابخانه پایتورچ پیاده سازی شده است.

در سلول اول کتاب خانه پایتورچ و زیر کتاب خانه های مورد نیاز به همراه تعدادی کتابخانه عمومی پایتون به پروژه افزوده شده است.

در سلول بعد حالت های مختلف اجرای پروژه قرار داده شده است: 8 حالت به قرار زیر:

- fomaml-5way-1shot (بخش دوم)
- maml-5way-1shot (بخش اول)
- fomaml-5way-5shot (بخش دوم)
- maml-5way-5shot (بخش اول)
- fomaml-5way-1shot_COSINE Scheduler (امتیازی)
- maml-5way-1shot_COSINE Scheduler (امتیازی)
- fomaml-5way-5shot_COSINE Scheduler (امتیازی)
- maml-5way-5shot_COSINE Scheduler (امتیازی)

با تنظیم پارامتر های سلول دوم هر یک از 8 حالت بالا را میتوان ایجاد کرد.

در سلول سوم هایپر پارامتر های متالرنینگ تعریف شده است. شامل:

meta_train_batch = 4

meta_test_batch = 20

meta_epochs = 20000

base_train_epochs = 5

base_test_epochs = 10

```
meta_train_rep_freq = 50
meta_test_freq = 200
meta_save_freq = 1000
alpha = 0.01 # base model learning rate
beta = 0.01 # meta model learning rate
```

سپس دو تسک ست تعریف کردیم، یکی برای ترین و دیگری برای تست. هر تسک ست شامل 4 تسک برای 1shot و دو تسک برای ترین به صورت 5shot میشود. برای تسک ست تست 20 تسک را در نظر گرفتیم (حل تمرین مربوطه در گروه 5 تسک را کافی دانستند و مقدار بیشتر را اختیاری در نظر گرفتند)

سلول بعد برای مشخص کردن دستگاهی که شبکه ها قرار است بر روی آن ترین شوند است. ما ترجیه استفاده را به GPU Cuda دادیم و در صورت موجود نبودن به روی cpuاجرا میشود.

در دو سلول بعدی دو تابع کمکی تعریف کرده ایم، تابع اول برای محاسبه دقت و f1 score است و تابع بعدی برای بخش امتیازی (تعیین نرخ یادگیری در هر اپیاک به کمک زمانبندی کسینوسی که استاد در کلاس اموزش دادند.)

سلول بعد معماری شبکه عصبی کانولوشن را تعیین میکند، این معماری مطابق با مقاله مرجع قرار گرفته شده ایجاد گردیده است. شامل 4 بلوک کانولوشن و یک لایه پرسپترون میشود. هر بلوک شامل یک لایه کانولوشن به دنبال آن Batch Norm و سپس Relu و در اخر مکسپول.

دو سلول بعد دو تابع اصلی این پیاده سازی را شامل میشوند، سلول اول شامل تابع اموزش مدل داخلی (حلقه داخلی) و سلول دوم شامل تابع آموزش متا مدل است. (شرح کامل دو تابع در ادامه امده است)

در سلول بعد حلقه اصلی برنامه قرار دارد، این حلقه به تعداد متا ایپاک ها که در این مسئله 20000 تا فرض شده است میچرخد.
در هر گردش حلقه، ابتدا تعدادی تسک از تسک ست آموزش انتخاب می شود (4 تا برای 1 شات و 2 تا برای 5 شات) و سپس به ازای هر تسک مدلی اموزش داده میشود. معماری این مدل ها با معماری که بالاتر توضیح داده شد یکسان است. برای اموزش هر مدل تابع اموزش مدل داخلی train_base_model صدا زده میشود. این تابع تسک، معماری مدل، ایپاک حلقه داخلی، دستگاه، الفا و مرتبه مشتق را گرفته و متناسبا مدلی را اموزش میدهد. برای اموزش مدل ابتدا داده ی ساپورت و کوئری تسک ورودی استخراج شده و سپس مدل به مدت ایپاک حلقه داخلی روی داده ساپورت ترین میشود. (5 ایپاک طبق داده های مسئله)
سپس مدل بر روی داده های کوئری پیش بینی انجام داده و مقدار لاس و گرادیانش را بر روی کوئری ذخیره می کنیم.

بعد از بدست اوردن گرادیان و لاس مدل های مختلف (که متناظر با تسک های مختلف ترین هستند) این لاس و گرادیان ها با هم ترکیب شده و میانگین گرفته میشود. از گرادیان حاصل برای اپدیت پارامتر های متا مدل استفاده میشود (در تابع train_meta_model)
در این تابع ما یکی یکی پارامتر های متا مدل را با گرادیان هم ارزش آپدیت می کنیم.
در انتها مدل جدید به جای متا مدل فعلی نشسته و به گردش بعدی حلقه میرویم. برای توضیحات بیشتر، خط به خط کد نوشته شده کامنت گذاری شده است.

```python
# %%
import torch
import torchvision
from torchvision import transforms
from torch import nn, optim
import numpy as np
import random
from PIL import Image
import matplotlib.pyplot as plt
from sklearn.metrics import accuracy_score, f1_score

import os, copy, time, math
import MiniImageNetTaskSet as ts

# %%
# fomaml-5way-5shot.ipynb
nway = 5
support_samples = 5
query_samples = 10
dv_mode = 'fo' # fo: first order (FOMAML) - so: second order (MAML)
use_lr_scheduler = False

#-- Running on 1650

# %%
meta_train_batch = 4
meta_test_batch = 20

meta_epochs = 20000
base_train_epochs = 5
base_test_epochs = 10

meta_train_rep_freq = 50
meta_test_freq = 200
meta_save_freq = 1000

alpha = 0.01 # base model learning rate
beta = 0.01 # meta model learning rate

ts_train = ts.MiniImageNetTaskSet(mode='train', nway=nway, support_samples=support_samples,
query_samples=query_samples)
#ts_val = ts.MiniImageNetTaskSet(mode='val', nway=nway, support_samples=support_samples,
query_samples=query_samples)
ts_test = ts.MiniImageNetTaskSet(mode='test', nway=nway, support_samples=support_samples,
query_samples=query_samples)

# %%
if torch.cuda.is_available():
    device = torch.device('cuda')
    print('OK CUDA was avaialbe.')
else:
    device = torch.device('cpu')
print(device)

# %%
def acc_f1(y, y_hat):
    p = y_hat.max(1, keepdim=True)[1]
    acc, f1 = accuracy_score(y.cpu(), p.cpu()), f1_score(y.cpu(), p.cpu(), average='macro')
    return acc, f1

# %%
def cosine_decay(epoch, beta):
    a = 0.5 * beta * (1 + math.cos(epoch * math.pi / meta_epochs))
    return a

# %%
meta_model = nn.Sequential(
    nn.Conv2d(3, 32, (3,3), (1,1)),
    nn.BatchNorm2d(32),
```

```
    nn.ReLU(),
    nn.MaxPool2d((2,2)),

    nn.Conv2d(32, 32, (3,3), (1,1)),
    nn.BatchNorm2d(32),
    nn.ReLU(),
    nn.MaxPool2d((2,2)),

    nn.Conv2d(32, 32, (3,3), (1,1)),
    nn.BatchNorm2d(32),
    nn.ReLU(),
    nn.MaxPool2d((2,2)),

    nn.Conv2d(32, 32, (3,3), (1,1)),
    nn.BatchNorm2d(32),
    nn.ReLU(),
    nn.MaxPool2d((2,2)),

    nn.Flatten(),
    nn.Linear(288, nway)
)

print(meta_model)

# %%
def train_base_model(base_model, episode, base_epochs, device, alpha, dv_mode):
    base_model_grads = None # variable to store base model grads after backward propagating Query data
loss

    D_sup, D_que = episode # extracting Support and Query data from episode
    X_sup, y_sup = D_sup # extracting Support inputs (X) and labels (y)
    X_que, y_que = D_que # extracting Query inputs (X) and labels (y)
    # assigning Support and Query tensors to device
    X_sup, y_sup = X_sup.to(device), y_sup.to(device)
    X_que, y_que = X_que.to(device), y_que.to(device)

    base_criterion = nn.CrossEntropyLoss() # base model loss function
    base_optimizer = optim.SGD(base_model.parameters(), lr=alpha) # base model optimizer

    # training base model on Support data
    base_model.train() # telling pytorch we want to train this base model
    for i in range(base_epochs):
        base_optimizer.zero_grad() # telling pytorch to clear calculated grads from previous computations
        y_sup_hat = base_model(X_sup) # feed forward of all Support inputs for current episode
        base_loss = base_criterion(y_sup_hat, y_sup) # computing the loss of current task Support

        # backward propagate derivatives for base model
        base_loss.backward()
        # update base model parameters
        base_optimizer.step()

    # testing trained base model on Query data
    else:
        base_model.train() # telling pytroch we want to train this base model
        base_optimizer.zero_grad() # telling pytorch to clear calculated grads from previous computations
        y_que_hat = base_model(X_que) # feed forward of all Query inputs for current episode
        base_que_loss = base_criterion(y_que_hat, y_que) # calculating query loss for base model

        if dv_mode == 'fo': # first order grad calc
            base_que_loss.backward() # backward propagating the query loss to calculate parameters
gradients w.r.t. query loss
        elif dv_mode == 'so': # second order grad calculation
            base_que_loss.backward(retain_graph=True)
            base_que_loss.backward()



        # storing base model grads w.r.t. Query data loss
        base_model_grads = [param.grad for param in list(base_model.parameters())]
```

```
        # storing base model loss over Query data
        base_model_loss = base_que_loss.item()

        # calculating base model's accuracy and F1 score over Query data
        base_model_acc, base_model_f1 = acc_f1(y_que, y_que_hat)

        # removing the base model from gpu to free vram
        del base_que_loss
        del base_loss
        del base_criterion
        del base_optimizer
        del base_model
        torch.cuda.empty_cache()

    return base_model_loss, base_model_grads, base_model_acc, base_model_f1


# %%
def train_meta_model(meta_model, beta, meta_batch, dv_mode, device, ls_base_models_grads,
ls_base_models_losses, ls_base_models_acc, ls_base_models_f1):
    # training meta model over a batch of episodes
    with torch.no_grad():
        _ = meta_model.to(device) # assigning meta model to device

        # a list to store meta model parameters grads
        meta_model_grads = [torch.zeros_like(param) for param in ls_base_models_grads[0]]

        # creating meta model parameters grads
        for base_model_grads in ls_base_models_grads: # for each base model
            for grad_index in range(len(base_model_grads)): # for each base model's parameter's grad
                base_model_grad = base_model_grads[grad_index] # get base model grad
                # create meta model parameter grad
                meta_model_grads[grad_index] += (1/meta_batch) * base_model_grad



        # updating meta model parameter
        for param, grad in zip(meta_model.parameters(), meta_model_grads):
            new_param = param - beta * grad
            param.copy_(new_param)

    # calculate meta training loss, acc, f1 by averaging base models' losses, acc, f1
    meta_loss = sum(ls_base_models_losses) / meta_batch
    meta_acc = sum(ls_base_models_acc) / meta_batch
    meta_f1 = sum(ls_base_models_f1) / meta_batch

    return meta_model, meta_loss, meta_acc, meta_f1


# %%
ls_meta_train_metrics = []
ls_meta_test_metrics = []

time_begin = time.time()

for meta_i in range(1, meta_epochs+1):
    # sample meta_batch tasks
    train_episodes = ts_train.sample_episodes(meta_train_batch)

    ls_base_models_grads = []
    ls_base_models_losses = []
    ls_base_models_acc = []
    ls_base_models_f1 = []

    for episode, ep_i in zip(train_episodes, range(len(train_episodes))):
        base_model = copy.deepcopy(meta_model) # copy parameters of meta model to base model
        _ = base_model.to(device) # assign base model to device

        # training base model on Support and Calculate it's Gradients w.r.t. Query data
```

```
        base_model_loss, base_model_grads, base_model_acc, base_model_f1 = train_base_model(base_model,
                                                                                             episode,
                                                                                             base_train_epo
chs,
                                                                                             device,
                                                                                             alpha,
                                                                                             dv_mode)

        ls_base_models_losses.append(base_model_loss)
        ls_base_models_grads.append(base_model_grads)
        ls_base_models_acc.append(base_model_acc)
        ls_base_models_f1.append(base_model_f1)


    # training meta model w.r.t. base models' gradients
    lr = beta if not use_lr_scheduler else cosine_decay(meta_i, beta)
    meta_model, meta_train_loss, meta_train_acc, meta_train_f1 = train_meta_model(meta_model,
                                                                                  lr,
                                                                                  meta_train_batch,
                                                                                  dv_mode,
                                                                                  device,
                                                                                  ls_base_models_grads,
                                                                                  ls_base_models_losses,
                                                                                  ls_base_models_acc,
                                                                                  ls_base_models_f1)

    ls_meta_train_metrics.append((meta_train_loss, meta_train_acc, meta_train_f1))

    # storing model checkpoint
    if meta_i%meta_save_freq == 0:
        torch.save(meta_model.state_dict(), os.path.join('models',
f'model_{meta_i}_{nway}way_{support_samples}shot_{dv_mode}{"_COS" if use_lr_scheduler else ""}.pt'))

    # reporting back to user
    if meta_i%meta_train_rep_freq == 0:
        print(f'- Meta Epoch: {meta_i}, meta loss: {meta_train_loss}, meta acc: {round(meta_train_acc,
3)}, meta f1: {round(meta_train_f1, 3)}, took: {round(time.time()-time_begin, 1)} secs')
        time_begin = time.time()

    # meta testing
    if meta_i%meta_test_freq == 0:
        time_begin = time.time()
        test_episodes = ts_test.sample_episodes(meta_test_batch)
        ls_test_models_losses = []
        ls_test_models_acc = []
        ls_test_models_f1 = []

        for episode, ep_i in zip(test_episodes, range(len(test_episodes))):
            test_model = copy.deepcopy(meta_model) # copy parameters of meta model to val model
            _ = test_model.to(device) # assign base model to device

            # training val model on Support and Calculate it's loss, f1 and acc w.r.t. Query data
            test_model_loss, _, test_model_acc, test_model_f1 = train_base_model(test_model, episode,
base_test_epochs, device, alpha, dv_mode)

            ls_test_models_losses.append(test_model_loss)
            ls_test_models_acc.append(test_model_acc)
            ls_test_models_f1.append(test_model_f1)

        # calculate meta validation loss, acc, f1 by averaging validation models' losses, acc, f1
        meta_test_loss = sum(ls_test_models_losses) / meta_test_batch
        meta_test_acc = sum(ls_test_models_acc) / meta_test_batch
        meta_test_f1 = sum(ls_test_models_f1) / meta_test_batch
        ls_meta_test_metrics.append((meta_test_loss, meta_test_acc, meta_test_f1))

        print(f'[META TEST]- Meta Epoch: {meta_i}, meta loss: {meta_test_loss}, meta acc:
{round(meta_test_acc, 3)}, meta f1: {round(meta_test_f1, 3)}, took: {round(time.time()-time_begin, 1)}
secs')
        time_begin = time.time()
```

```
# %%
ls_meta_train_losses = [item[0] for item in ls_meta_train_metrics]
plt.plot(ls_meta_train_losses)
plt.xlabel('Meta Training Epoch')
plt.ylabel('Meta Training Loss')
plt.title('Meta Training Loss Plot')

# %%
ls_meta_train_acc = [item[1] for item in ls_meta_train_metrics]
plt.plot(ls_meta_train_acc)
plt.xlabel('Meta Training Epoch')
plt.ylabel('Meta Training Accuracy')
plt.title('Meta Training Accuracy Plot')

# %%
ls_meta_train_f1 = [item[2] for item in ls_meta_train_metrics]
plt.plot(ls_meta_train_f1)
plt.xlabel('Meta Training Epoch')
plt.ylabel('Meta Training F1-Score')
plt.title('Meta Training F1-Score Plot')

# %%
ls_meta_test_losses = [item[0] for item in ls_meta_test_metrics]
plt.plot(ls_meta_test_losses)
plt.xlabel('Meta Testing Epoch')
plt.ylabel('Meta Testing Loss')
plt.title('Meta Testing Loss Plot')

# %%
ls_meta_test_acc = [item[1] for item in ls_meta_test_metrics]
plt.plot(ls_meta_test_acc)
plt.xlabel('Meta Testing Epoch')
plt.ylabel('Meta Testing Accuracy')
plt.title('Meta Testing Accuracy Plot')

# %%
ls_meta_test_f1 = [item[2] for item in ls_meta_test_metrics]
plt.plot(ls_meta_test_f1)
plt.xlabel('Meta Testing Epoch')
plt.ylabel('Meta Testing F1-Score')
plt.title('Meta Testing F1-Score Plot')

# %%
import pandas as pd

# %%
df_test = pd.DataFrame(ls_meta_test_metrics, columns=['loss', 'acc', 'f1'])
df_train = pd.DataFrame(ls_meta_train_metrics, columns=['loss', 'acc', 'f1'])

df_test.to_csv('test_metrics.csv', index=False)
df_train.to_csv('train_metrics.csv', index=False)

# %%
```

در ادامه نمودار های 8 حالت مختلف آمده است، داده های ترین 20000 نقطه با رنگ آبی و داده های 100 بار تست کردن (هر 200 بار ترین یک تست) با رنگ نارنجی رسم شده است.

همانگونه که از نمودار ها مشخص است، نوسان نمودار آبی (20000 اپیاک متا ترین) بسیار بیشتر ازنمودار نارنجی (متا تست) است. از نظر بنده دلیل این اتفاق میتواند تلاش مدل برای فرار از مینیمم های محلی باشد. مدل به دنبال پارامترهایی است که بتواند برای رنج وسیعی از تسک ها خوب عمل کند، پس طبیعی است که وزن های مختلفی را مورد بررسی قرار دهد. از طرفی دلیل نوسانات کمتر روی تسک های تست، میتواند به دلیل پیدا شدن وزن های مناسب توسط بخش ترین باشد.
جداول زیر مقایسه نتایج مدل های مختلف روی تسک های تست و ترین است. اعداد به درصد هستند.

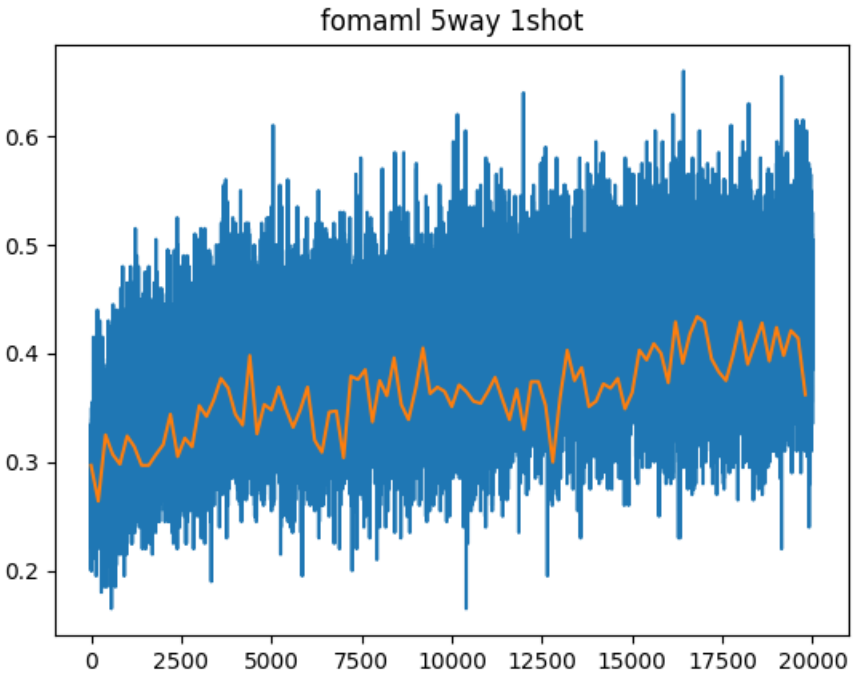
	5 way 1 shot			5 way 5 shot	
	Accuracy	F1-Score		Accuracy	F1-Score
FOMAML	36 ± 3	35 ± 3		57 ± 4	56 ± 4
FOMAML + Scheduler	37 ± 4	36 ± 4		58 ± 4	57 ± 4
MAML	35 ± 4	34 ± 4		56 ± 3	55 ± 3
MAML + Sheduler	36 ± 4	35 ± 4		57 ± 4	56 ± 4

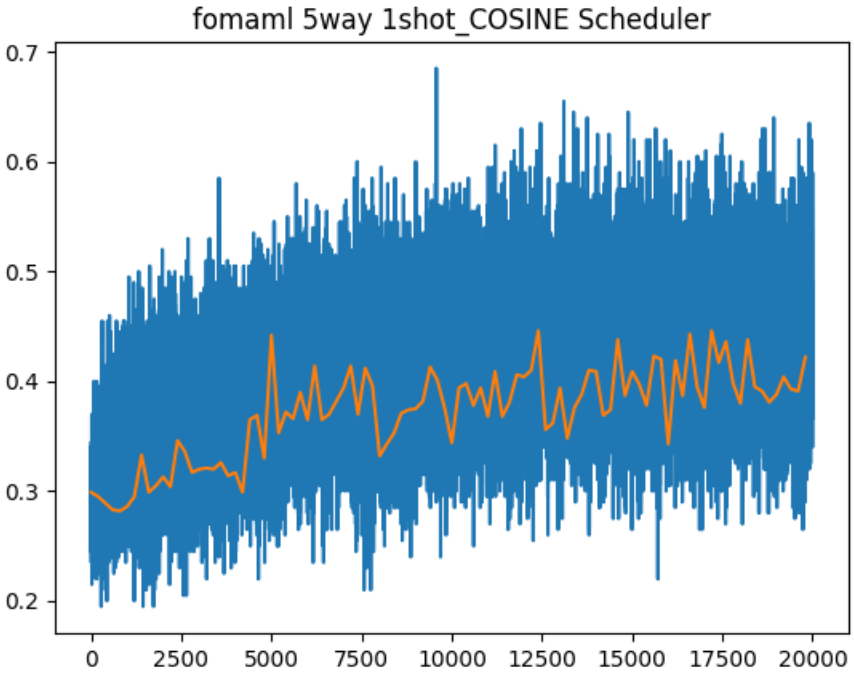
نتایج بر روی تسک های تست

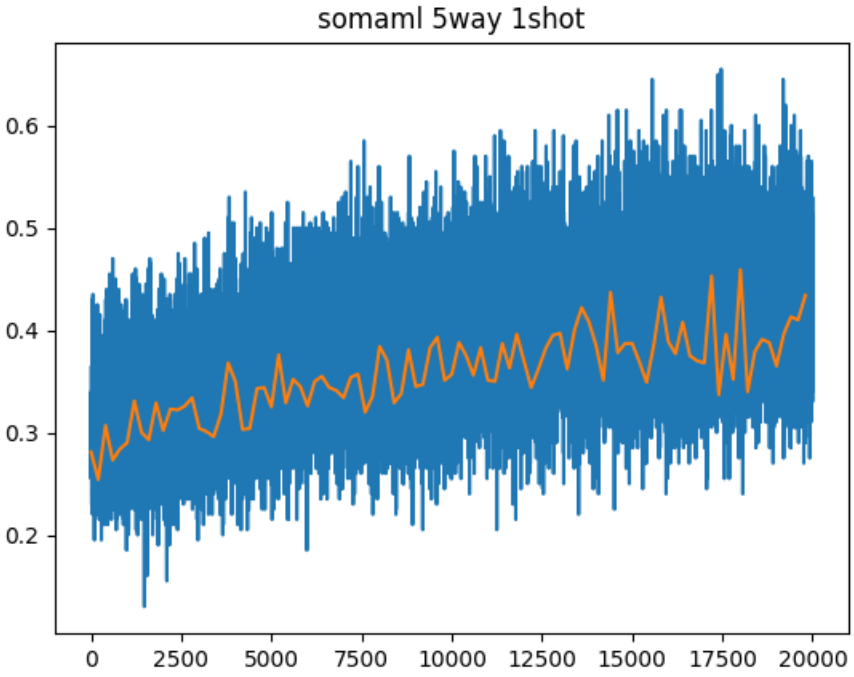
	5 way 1 shot			5 way 5 shot	
	Accuracy	F1-Score		Accuracy	F1-Score
FOMAML	39 ± 6	37 ± 6		62 ± 7	61 ± 7
FOMAML + Scheduler	41 ± 6	39 ± 6		63 ± 7	62 ± 7
MAML	38 ± 6	36 ± 7		60 ± 7	60 ± 7
MAML + Sheduler	40 ± 6	36 ± 6		61 ± 7	61 ± 7

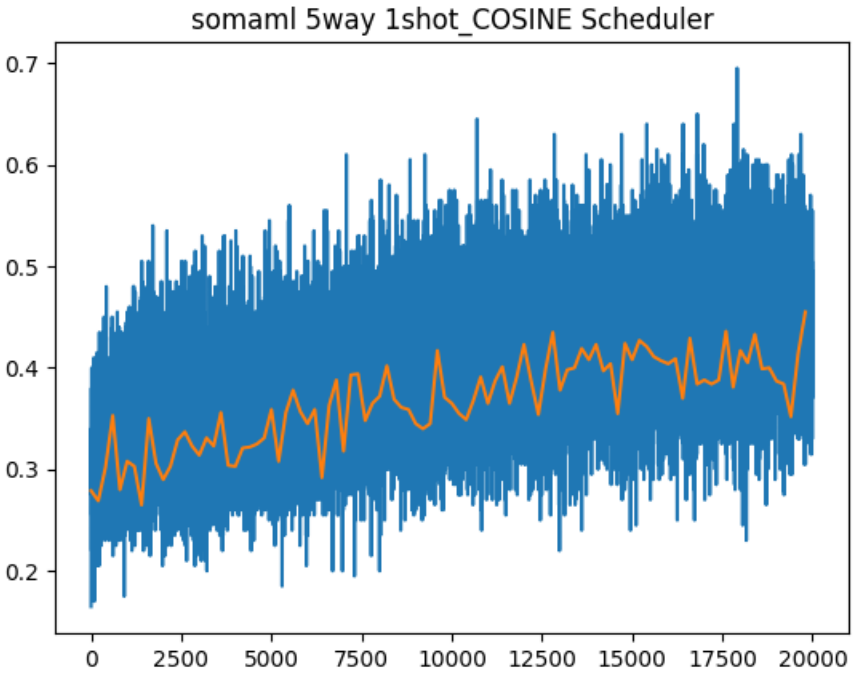
نتایج بر روی تسک های ترین

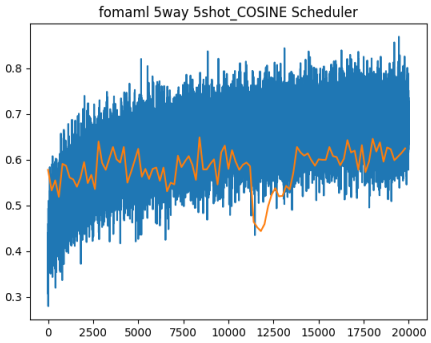
برای دیدن نتایج کامل (شامل میانگین، انحراف معیار، مینیمم، مکسیمم) به فایل analysis.xlsx بروید.
مدل هایی که به صورت First order MAML ترین شدند دقت بالاتری نسبت به مدل های عادی MAML دارند و از طرفی مدل های 5 شات نسبت به 1 شات دقت بالاتری نشان داده اند.
توجه داشته باشید، که دقت ها و F1 score ها به صورت میانگین هستند.

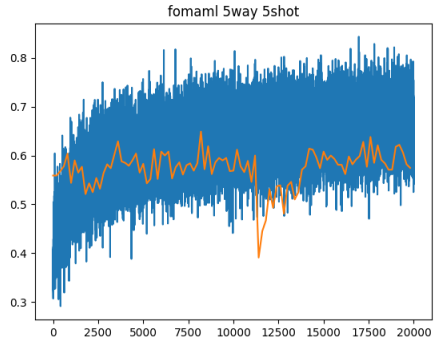

fomaml 5way 1shot
0.6
0.5
0.4
0.3
0.2
0
2500
5000
7500
10000
12500
15000
17500
20000


fomaml 5way 1shot_COSINE Scheduler
0.7
0.6
0.5
0.4
0.3
0.2
0
2500
5000
7500
10000
12500
15000
17500
20000

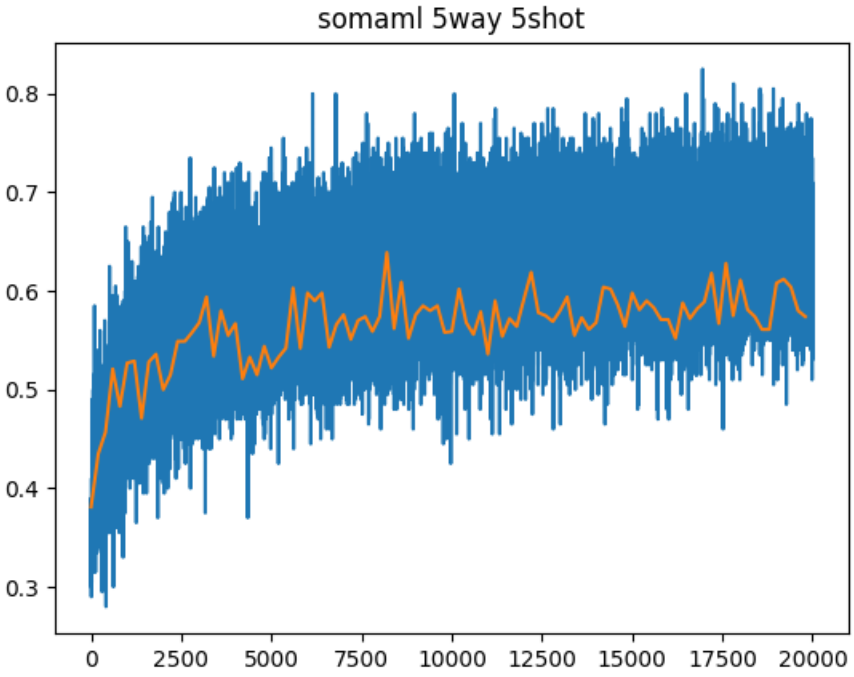

somaml 5way 1shot
0.6
0.5
0.4
0.3
0.2
0
2500
5000
7500
10000
12500
15000
17500
20000

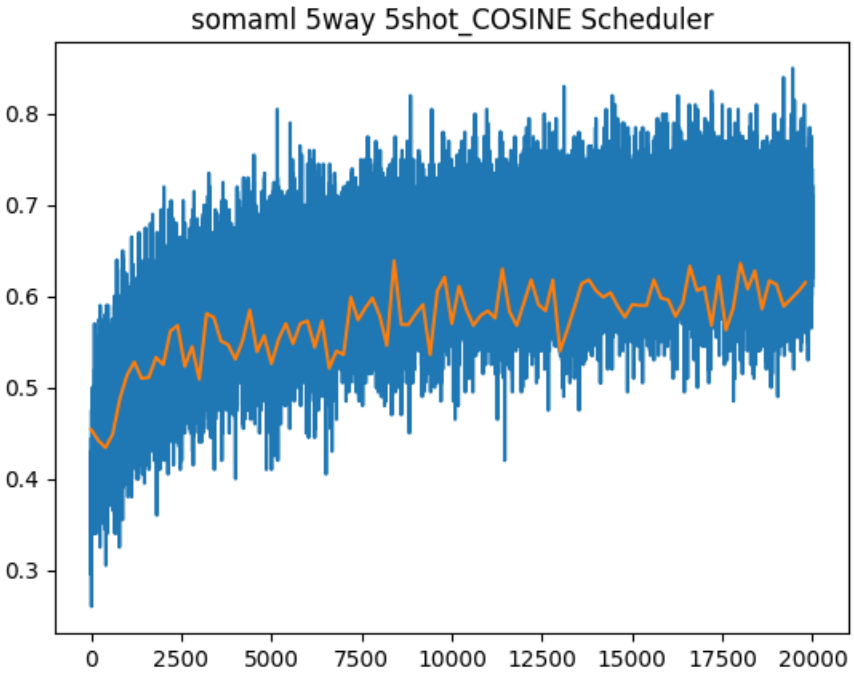

somaml 5way 1shot_COSINE Scheduler
0.7
0.6
0.5
0.4
0.3
0.2
0
2500
5000
7500
10000
12500
15000
17500
20000





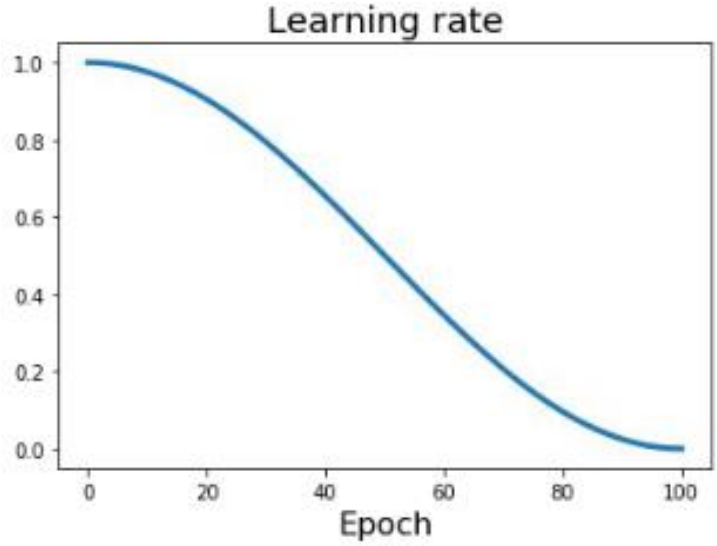
این دوتا نمودار به خاطر یک باگ در کد در زمان ترین، یک شیفت در نتایج تست خورده است، اون تکه که افت شدید کرده در واقع در ابتدای نمودار بوده است. بسیار علاقه مند بودم که این مشکل رو درست کنم اما Google Colab دیگه بهم GPU نداد. داده های تست دقیق و درست هستند به همین دلیل از نظر میانگین و ماکسیمم و مینیمم تفاوتی ندارند (نتایج جداول چند صفحه قبل صحیح است) اما چون ترتیب قرار گیری این نتایج اشتباه شده است، نمودار ها دچار این آشفتگی شده اند.





به صورت 5 شات چون دیتای بیشتری برای ترین در اختیار مدل بوده است، مدل توانسته به دقت بالاتری برسد. چه در حالت ترین و چه در حالت تست.

استفاده از نرخ یادگیری داینامیک و متغیر نیز به مدل برای یادگیری بهتر کمک می کند. برای متغیر کردن نرخ یادگیری، بنده از یادگیری کسینوسی که در کلاس توسط استاد تدریس شد استفاده کردم. در این نوع یادگیری ما نرخ یادگیری را به عنوان حاصل ضرب یک نرخ یادگیری پایه در یک عبارت کسینوسی در نظر میگیریم، فرمول محاسبه نرخ یادگیری در این حالت به شرح زیر میباشد.



Cosine: $\alpha_t = \frac{1}{2}\alpha_0 \left(1 + \cos(t\pi/T)\right)$

که در آن آلفا صفر نرخ یادگیری پایه، تی بزرگ تعداد ماکسیمم ایپاک ها و تی کوچک ایپاک فعلی را نشان میدهد. نرخ یادگیری متغیر با کوچکتر کردن تدریجی گام های بروزرسانی پارامتر های مدل، باعث جلوگیری از تغییر ناگهانی مدل و رفتار واریانسی آن میشود. به این صورت که در ابتدا به مدل آزادی عمل بیشتری در یادگیری میدهد و به تدریج این ازادی کاهش یافته و مدل را به ثبات میرساند. از دید فضای حل مسئله، در ابتدا ما فضای بیشتری را مورد کاوش قرار میدهیم و با گذشت زمان فضا مورد جست و جو کاهش بیشتری پیدا میکند و در نهایت به اطراف نقطه ای محدود می شود.

از طرف دیگر با توجه به نتایج بدست آمده که در جدول قابل مشاهده است، مدل ها در حالت first order maml بهتر ترین شده اند و نتایج بهتری را ارائه داده اند که این امر در مقاله مرجع نیز مورد اشاره واقع شده بود.

برای اجرای پروژه فایل maml.ipynb را به کمک نرم افزاری مناسب مانند Google Colab باز کنید و سپس بعد از تنظیم پارامتر های مورد نیاز (ایجاد هرکدام از 8 حالت) در سلول دوم، کل سلول ها را با هم دیگر اجرا کنید. توجه داشته باشید در هنگام اجرا در کنار فایل نوت بوک، فولدری به نام models موجود باشد تا کد نوشته شده بتواند به ازای هر 1000 ایپاک یک مدل ذخیره کند که در صورت از دست رفتن اجرا، بتوانید از نزدیک ترین اجرا مجدد از سر بگیرید.

همچنین فایل MiniImageNetTaskSet.py نیز باید در کنار فایل نوت بوک موجود باشد. این فایل شامل کلاس و توابع مورد نیاز برای پارس کردن تصاویر و ایجاد تسک ست مناسب است. تمام خطوط کد این فایل درون فایل توضیح داده شده است. برای راحتی شما این کد ها در ادامه نیز پیوست شده است.

```python
import torch
from torchvision import transforms
import random
from PIL import Image
import matplotlib.pyplot as plt

import os


class MiniImageNetTaskSet():
    '''A task set helper class

    '''
    def __init__(self, root='miniimagenet/', mode='train', nway=5,
support_samples=1, query_samples=10) -> None:
        self.root = root
        self.data_dir = os.path.join(root, 'data')
        self.mode = mode
        self.nway = nway
        self.sup_count = support_samples
        self.que_count = query_samples

        with open(os.path.join(root, 'splits', f'{mode}.txt')) as file:
            self.C = [line.strip().split(',') for line in file.readlines()]

        self.transform = transforms.Compose([
            #transforms.Resize((32,32)),
            transforms.ToTensor(),
            transforms.Normalize((0.485, 0.456, 0.406), (0.229, 0.224, 0.225)),
        ])



    def __len__(self):
        'Denotes the total number of classes'
        return len(self.C)


    def sample_episode(self):
        # sample nway class from all classes avaialabe to this task set
        C = random.sample(self.C, k=self.nway)

        ls_tensors_sup = []
        ls_tensors_que = []
        ls_labels_sup = []
```

```
        ls_labels_que = []

        # load all nway * (sup_count + que_count) samples
        for cc, ci in zip(C, range(len(C))):
            c = cc[0]
            # get all images of class c
            all_c_images = os.listdir(os.path.join(self.data_dir, c))
            # get a sup_count + que_count sample out of all
            selected_images = random.sample(all_c_images,
k=self.sup_count+self.que_count)
            # load images to torch tensors
            tensors = [self.transform(Image.open(os.path.join(self.data_dir, c,
image_name))).unsqueeze(0) for image_name in selected_images]
            ls_tensors_sup += tensors[:self.sup_count]
            ls_labels_sup += [ci] * self.sup_count
            ls_tensors_que += tensors[self.sup_count:]
            ls_labels_que += [ci] * self.que_count



        #
        X_sup = torch.cat(ls_tensors_sup, 0)
        y_sup = torch.tensor(ls_labels_sup)
        #
        X_que = torch.cat(ls_tensors_que, 0)
        y_que = torch.tensor(ls_labels_que)

        # shuffling Support and Query data
        shuff_indices_sup = random.sample(range(X_sup.shape[0]), X_sup.shape[0])
        X_sup = X_sup[shuff_indices_sup]
        y_sup = y_sup[shuff_indices_sup]

        shuff_indices_que = random.sample(range(X_que.shape[0]), X_que.shape[0])
        X_que = X_que[shuff_indices_que]
        y_que = y_que[shuff_indices_que]

        D_sup = (X_sup, y_sup) # nway * sup_count
        D_que = (X_que, y_que) # nway * que_count

        return (D_sup, D_que) # a sampled task


    def sample_episodes(self, batch):
        return [self.sample_episode() for _ in range(batch)]

    def visualize_episode(self, task):
```

```
        D_sup, D_que = task
        X_sup, y_sup = D_sup
        X_que, y_que = D_que

        X = torch.cat([X_sup, X_que], 0)

        noi = 4
        num_of_images = noi * noi
        for index in range(1, num_of_images+1):
            img = X[index-1]

            _ = plt.subplot(noi,noi, index)
            _ = plt.axis('off')
            _ = plt.imshow(img.moveaxis(0, -1))
```

پیاده سازی این فایل دقیقا مشابه با مقاله مرجع که هم راستا با تدریس استاد در کلاس بوده است انجام شده است.

فایل کمکی run-settings.ipynb برای راحتی ایجاد هر یک از 8 حالت گفته شده ساخته شده است. با اجرای این کد و کپی کردن هر یک از 8 سلول موجود در این فایل در سلول دوم فایل maml.ipynb میتوانید یکی از 8 حالت را ترین کنید.