Andrei Mihnea-Stefan

Vision transformer(ViT)-combinatie Google si Facebook

Introducere

Am vazut un resnet ce foloseste layere convolutionale (documentatia este in README din folderol *resnet_andrei*). Acestea sunt mult mai usor de inteles decat tranformerele. In ceea ce voi prezenta aici, eu nu inteleg deocamdata nici 10% probabil.

Ideea este ca transformerele au un mecanizm de atentie. Formula matematica este destul de urata:

$$\text{Attention}(Q, K, V) = \text{Softmax}(QK^\top/\sqrt{d})V,$$

Unde, Q, K, V sunt matrici obtinute prin transformari lineare ale datelor de intrare X: $Q = XW_Q, K = XW_K, V = XW_V$. Ideea a fost introdusa initital in probabil cel mai faimos articol de cercetare in AI in ultimii ani. Pagina de internet cu implementarile in pytorch ale ideilor din acest articol este: https://nlp.seas.harvard.edu/2018/04/03/attention.html .

Google si Facebook au inceput sa aplice aceste idei de transformere la clasificare de imagini. In particular, le-au aplicat si la recunoasterea faciala. Anul trecut cele 2 companii au publicat articole in care au aratat pentru prima data ca transformerele au acuratete mai mare si sunt si mai rapide la antrenare decat resnet (AI-ul cel mai bun la recunoasterea facial pana atunci).

- Articolul Facebook: https://arxiv.org/pdf/2012.12877.pdf
- Articolul Google: https://arxiv.org/pdf/2010.11929.pdf

Este foarte folositor ca cele 2 companii au si codul gratis pe github. Pe noi ne intereseaza mai ales:

- Arhitectura neuronala Google este intretinuta de o persoana fizica pe github si poate fi gasita la: https://github.com/rwightman/pytorch-image-models/blob/master/timm/models/vision_transformer.py
- Arhitectura neuronala Facebook este intretinuta de echipa de cercetare de la Facebook, ciar pe github: https://github.com/facebookresearch/deit/blob/main/models_v2.py

Se pare ca cea mai buna acuratete obtinuta de Google este de 88,55%, iar de facebook este de 84,2%, ambele obtinute desigur pe acelasi set de imagini, numite ImageNet. De asemenea echipa de la Google are unul din pionerii aplicarii transformerelor in calsificarea de imagini, in general: Alexey Dosovitskiy. Din cate vom vedea, arhitectura neuronala a lui Google nu este neaparat mai complexa decat a Facebook-ului, dar implementarea facuta de Google este mai generala. Pratic, felul in care codul este scris de cei de la Google este mai general decat cel de la Facebook, dupa cum vom vedea

Abordarea mea

Dupa cum o sa vedem, clasa principala care defineste creierul are parametrii foarte asemanatori cu cei ai modelului DeiT (pentru detalii va rog sa cititi README-ul din folderul *DeiT_Facebook*). Deocamdata inca testez sa vad care model este mai bun: DeiT sau ViT (si micile variatiuni la care m-am

gandit). Pentru a face o comparatie, am pastrat aceleasi valori pentru parametrii clasei principale, acelasi optimizator, etc:

```
#The below number is 160 because the faces extracted by MTCNN are 160x160.
img_size=160
#The number of names that the model will see (population of romania over 14).
num_classes=data_sizes["train"]
#Variables inside the network. Please check examples that start at line 271 here:
#https://github.com/facebookresearch/deit/blob/main/models_v2.py
embed_dim=192
depth=12
num_heads=3
mlp_ratio=4

#Smallest model from FB examples is below.
model_fit=vit_models(num_classes=num_classes,img_size=img_size,patch_size=batches,
                     embed_dim=embed_dim,depth=depth,num_heads=num_heads,mlp_ratio=mlp_ratio,qkv_bias=True,
                    norm_layer=partial(nn.LayerNorm,eps=10**(-6)),
                    block_layers=Layer_scale_init_Block).to(device)

num_epochs=25
lr=3*10**(-3)
weight_decay=0.01
criterion=nn.CrossEntropyLoss()
#Facebook does AdamW. Please see paper.
#optimizer=optim.SGD(model_fit.parameters(),lr=10**(-2),momentum=0.9)
optimizer=torch.optim.AdamW(model_fit.parameters(),lr=lr, betas=(0.9, 0.999), eps=1e-08, weight_decay=weight_decay)
#lr_decay=lr_scheduler.StepLR(optimizer,step_size=7,gamma=0.1)
lr_decay=torch.optim.lr_scheduler.CosineAnnealingLR(optimizer=optimizer, T_max=num_epochs)
```

Singura diferenta dintre DeiT si ViT este un parametru numit *block_layers*. Astea sunt niste sub-retele neuronale pe care oamenii de la Google si de La Facebook le-au definit. Eu nu am facut decat sa le pun in acelasi cod ca sa puteti sa alegeti pe care doriti din ele:

- *Block* este definita de Facebook in mare pare, la care am adaugat 2 *LayerScale* dupa ideile de la Google (pentru ca acuratetea Google pare putin peste cea de la Facebook): *self.ls1* si *self.ls2* din codul urmator

```
#Original Block class from FB to which I try to add the extra layers that Google has
#Can be passed to block_layers parameter in main class.
class Block(nn.Module):
    # token from https://github.com/rwightman/pytorch-image-models/blob/master/timm/models/vision_transformer.py
    def __init__(self, dim, num_heads, mlp_ratio=4., qkv_bias=False, qk_scale=None, drop=0., attn_drop=0.,
                 drop_path=0., act_layer=nn.GELU, norm_layer=nn.LayerNorm,Attention_block = Attention,Mlp_block=Mlp
                 ,init_values=1e-4):
        super().__init__()
        self.norm1 = norm_layer(dim)
        self.attn = Attention_block(
            dim, num_heads=num_heads, qkv_bias=qkv_bias, qk_scale=qk_scale, attn_drop=attn_drop, proj_drop=drop)
        # NOTE: drop path for stochastic depth, we shall see if this is better than dropout here
        #Below, I add the LayerScale that Google has.
        self.ls1=LayerScale(dim,init_values=init_values) if init_values else nn.Identity()
        self.drop_path1 = DropPath(drop_path) if drop_path > 0. else nn.Identity()

        self.norm2 = norm_layer(dim)
        mlp_hidden_dim = int(dim * mlp_ratio)
        self.mlp = Mlp_block(in_features=dim, hidden_features=mlp_hidden_dim, act_layer=act_layer, drop=drop)
        #Below I add another LayerScale that Google has
        self.ls2=LayerScale(dim,init_values=init_values) if init_values else nn.Identity()
        self.drop_path2 = DropPath(drop_path) if drop_path > 0. else nn.Identity()


    def forward(self, x):
        x = x + self.drop_path1(self.ls1(self.attn(self.norm1(x))))
        x = x + self.drop_path2(self.ls2(self.mlp(self.norm2(x))))
        return x
```

- *Layer_scale_init_Block* care este copiata exact de la Facebook:

```python
#Take class from below from FB
#Can be passed to block_layers parameter in main class.
class Layer_scale_init_Block(nn.Module):
    # taken from https://github.com/rwightman/pytorch-image-models/blob/master/timm/models/vision_transformer.py
    # with slight modifications
    def __init__(self, dim, num_heads, mlp_ratio=4., qkv_bias=False, qk_scale=None, drop=0., attn_drop=0.,
                 drop_path=0., act_layer=nn.GELU, norm_layer=nn.LayerNorm,Attention_block = Attention,Mlp_block=Mlp
                 ,init_values=1e-4):
        super().__init__()
        self.norm1 = norm_layer(dim)
        self.attn = Attention_block(
            dim, num_heads=num_heads, qkv_bias=qkv_bias, qk_scale=qk_scale, attn_drop=attn_drop, proj_drop=drop)
        # NOTE: drop path for stochastic depth, we shall see if this is better than dropout here
        self.drop_path = DropPath(drop_path) if drop_path > 0. else nn.Identity()
        self.norm2 = norm_layer(dim)
        mlp_hidden_dim = int(dim * mlp_ratio)
        self.mlp = Mlp_block(in_features=dim, hidden_features=mlp_hidden_dim, act_layer=act_layer, drop=drop)
        self.gamma_1 = nn.Parameter(init_values * torch.ones((dim)),requires_grad=True)
        self.gamma_2 = nn.Parameter(init_values * torch.ones((dim)),requires_grad=True)

    def forward(self, x):
        x = x + self.drop_path(self.gamma_1 * self.attn(self.norm1(x)))
        x = x + self.drop_path(self.gamma_2 * self.mlp(self.norm2(x)))
        return x
```

- *ParallelBlock* care este aproape copiata exact de la Google. Aceasta este o implementare generalizata a verisunii Facebook numita *Block_paralx2* deoarece *Block_paralx2* poate fi obtinuta din codul de mai jos pentru *num_parallel=2*:

```python
#Use the generalized implementation that Google has for FB's Block_paralx2 (num_parallel=2 here for FB's implementation).
#Can be passed to block_layers parameter in main class.
class ParallelBlock(nn.Module):
    def __init__(self,dim,num_heads,num_parallel=2,mlp_ratio=4,qkv_bias=False,qk_scale=None,init_values=10**(-4),
                 drop=0,attn_drop=0,drop_path=0,act_layer=nn.GELU,norm_layer=nn.LayerNorm,Attention_block=Attention,
                 Mlp_block=Mlp):
        super().__init__()
        self.num_parallel=num_parallel
        self.attns=nn.ModuleList()
        self.ffns=nn.ModuleList()
        for _ in range(num_parallel):
            self.attns.append(nn.Sequential(OrderedDict([
                ("norm",norm_layer(dim)),
                ("attn",Attention(dim,num_heads=num_heads,qkv_bias=qkv_bias,qk_scale=qk_scale,attn_drop=attn_drop,proj_drop=d
                ("ls",LayerScale(dim,init_values=init_values) if init_values else nn.Identity()),
                ("drop_path",DropPath(drop_path) if drop_path>0 else nn.Identity())
            ])))
            self.ffns.append(nn.Sequential(OrderedDict([
                ("norm",norm_layer(dim)),
                ("mlp",Mlp(dim,hidden_features=int(dim*mlp_ratio),act_layer=act_layer,drop=drop)),
                ("ls",LayerScale(dim,init_values=init_values) if init_values else nn.Identity()),
                ("drop_path", DropPath(drop_path) if drop_path>0 else nn.Identity())
            ])))
    def _forward_jit(self,x):
        x=x+torch.stack([attn(x) for attn in self.attns]).sum(dim=0)
        x=x+torch.stack([ffn(x) for ffn in self.ffns]).sum(dim=0)
        return x

    #@torch.jit.ignore
    def forward(self,x):
        return self._forward_jit(x)
```

- *ParallelLayer_scale_init_Block* este o implementare a mea in spiritul general introdus de Google de mai sus pentru clasa din Facebook numita *Layer_scale_init_Block_paralx2*. Practic *Layer_scale_init_Block_paralx2* este aproape un caz particular al codului de mai jos pentru *num_parallel=2*. Zic aproape pentru ca nu am reusit sa implementez in acest spirit general introdus de Google cei 4 parametrii pe care Facebook ii numeste *gamma*. Dar sper ca o data cu cresterea *num_parallel*, reteaua neuronala rezultata o sa depaseasca lipsa acestor 4 parametrii gamma. De asemenea, am introdus 2 *LayerScale*-uri pe care Facebook nu le avea (in spiritul *ParallelBlock-ului de mai sus*):

```python
#Can be passed to block_layers parameter in main class.
#Implementation that generalizes Layer_scale_init_Block following Google's idea for ParallelBlock.
class ParallelLayer_scale_init_Block(nn.Module):
    def __init__(self,dim,num_heads,mlp_ratio=4,qkv_bias=False,qk_scale=None,drop=0,attn_drop=0,
                 drop_path=0,act_layer=nn.GELU,norm_layer=nn.LayerNorm,Attention_block=Attention,
                  Mlp_block=Mlp,init_values=10**(-4),num_parallel=2):
        super().__init__()
        self.num_parallel=num_parallel
        self.attns=nn.ModuleList()
        self.ffns=nn.ModuleList()
        ######################3
        #do not know how to multiply by gammas since those are just compositions of functions basically.
        #hopefully it does not matter since gammas are parameters and maybe it will be better if num_parameters>2
        #could add LayerScale to both attention and forward feed.
        ############################
        mlp_hidden_dim = int(dim * mlp_ratio)
        for _ in range(num_parallel):
            self.attns.append(nn.Sequential(OrderedDict([
                ("norm",norm_layer(dim)),
                ("attn",Attention_block(dim,num_heads=num_heads,qkv_bias=qkv_bias,qk_scale=qk_scale,
                                        attn_drop=attn_drop,proj_drop=drop)),
                ("ls",LayerScale(dim,init_values=init_values) if init_values else nn.Identity()),
                ("drop_path",DropPath(drop_path) if drop_path>0 else nn.Identity())
            ])))
            self.ffns.append(nn.Sequential(OrderedDict([
                ("norm",norm_layer(dim)),
                ("mlp",Mlp_block(in_features=dim,hidden_features=mlp_hidden_dim,act_layer=act_layer,drop=drop)),
                ("ls",LayerScale(dim,init_values=init_values) if init_values else nn.Identity()),
                ("drop_path",DropPath(drop_path) if drop_path>0 else nn.Identity())
            ])))

    def forward(self,x):
        x=x+torch.stack([attn(x) for attn in self.attns]).sum(dim=0)
        x=x+torch.stack([ffn(x) for ffn in self.ffns]).sum(dim=0)
        return x
```

Acest *LayerScale* este copiat de la Google, care are un parametru de tipul *gamma* pe care codul general de mai sus pierduse din implementarea Facebook:

```python
#LayerScale from Google
class LayerScale(nn.Module):
    def __init__(self, dim, init_values=1e-5, inplace=False):
        super().__init__()
        self.inplace = inplace
        self.gamma = nn.Parameter(init_values * torch.ones(dim))

    def forward(self, x):
        return x.mul_(self.gamma) if self.inplace else x * self.gamma
```

Ultima clasa din cod este copiata de la Facebook, dar nu m-am jucat cu ea inca:

```python
#Copy class from below from FB
class MHLP_stem(nn.Module):
    """ MHLP_stem: https://arxiv.org/pdf/2203.09795.pdf
    taken from https://github.com/rwightman/pytorch-image-models/blob/master/timm/models/vision_transformer.py
    with slight modifications
    """
    def __init__(self, img_size=224,  patch_size=16, in_chans=3, embed_dim=768,norm_layer=nn.SyncBatchNorm):
        super().__init__()
        img_size = to_2tuple(img_size)
        patch_size = to_2tuple(patch_size)
        num_patches = (img_size[1] // patch_size[1]) * (img_size[0] // patch_size[0])
        self.img_size = img_size
        self.patch_size = patch_size
        self.num_patches = num_patches
        self.proj = torch.nn.Sequential(*[nn.Conv2d(in_chans, embed_dim//4, kernel_size=4, stride=4),
                                          norm_layer(embed_dim//4),
                                          nn.GELU(),
                                          nn.Conv2d(embed_dim//4, embed_dim//4, kernel_size=2, stride=2),
                                          norm_layer(embed_dim//4),
                                          nn.GELU(),
                                          nn.Conv2d(embed_dim//4, embed_dim, kernel_size=2, stride=2),
                                          norm_layer(embed_dim),
                                         ])


    def forward(self, x):
        B, C, H, W = x.shape
        x = self.proj(x).flatten(2).transpose(1, 2)
        return x
```

Concluzie abordarea mea

Deci, in conlcuzie, am rescris o clasa de la Facebook in cod mai general, am adaugat 2 *LayerScale*-uri, dar am pierdut un net de 2 variabile gamma. In schimb codul general se reproduce pe sine de *num_parallel* ori.

Gruparea imaginilor, antrenarea si evaluarea sunt exact inainte la *resnet* si *DeiT*.

Am mai adaugat la toate aceste retele neuronale 2 linii care salveaza modelul antrenat intr-un fisier. Acesta poate fi citit tot de pytorch si recunoasterea faciala o sa foloseasca acest model antrenat deja direct. Practic nu trebuie sa treci prin antrenare mereu. O data ce este facuta, puteti sa salvati modelul si sa il cititi din nou si sa faceti predictii, care se fac mult mai repede decat antrenarea. Predictiile practice sunt doar multiplicari de matrici, dar antrenarea presupune optimizare (incredibil de multe derivate partiale – unul din modelele facebook avand 86 de milioane de variabile de exemplu):

```
save_model_path=r"C:\Users\mihnea.andrei\Python scripts\ViT_Google_FB\saved_weights"
save_model_file="\ViT-ParallelLayer_scale_init_Block,embed_dim=%d,depth=%d,num_heads=%d,mlp_ratio=%d,train_loss=%.4f,train_ac
    (embed_dim,depth,num_heads,mlp_ratio,np.mean(train_losses),100*np.mean(train_accs))

torch.save(model_fit,save_model_path+save_model_file)
```

De asemenea, am adaugat niste grafice cu loss-ul la antrenare si acuratetea medie:

```
plt.figure(figsize=(10,5))
plt.title("ViT_GOOG-FB training losses")
plt.plot(train_losses,label="train losses")
plt.xlabel("iterations")
plt.ylabel("loss")
plt.legend()
plt.show()
```

```
train_avg_accs=100*np.cumsum(train_accs)/list(range(len(train_accs)))
plt.figure(figsize=(10,5))
plt.title("ViT_GOOG-FB cumulative training average")
plt.plot(train_avg_accs,label="train accuracies")
plt.xlabel("iterations")
plt.ylabel("accuracy")
plt.legend()
plt.show()
```

Acum nu fac decat sa incerc diversele optiuni mentionate mai sus pentru parametrul *block_layers* din clasa principala *vit_models*.