

论文复现 – 经验分享

论文复现大致包含以下几步：

- 1. 验证源代码的有效性
- 2. 项目层面的代码转换
- 3. 框架层面的代码迁移
- 4. 模型训练
- 5. 部署推理
- 6. 文档编写

1. 验证原代码的有效性

- 首先需要确认官方的代码是可以复现论文中的结果的，否则后续的复现就意义不大了
- 具体做法就是根据官方源码中的指示下载数据，训练模型，然后将模型的评估结果与论文中的结果进行比对

2. 项目层面的代码转换

- 该步骤包含以下几步：1. 通过PaConvert转换项目 2. 手工修正无法自动转换的代码 3. 前反向对齐

1. 通过PaConvert转换项目

- 使用PaConvert可以将torch项目转化为paddle项目，包含以下两个步骤
 - 拉取PaConvert源码：
 - git clone <https://github.com/PaddlePaddle/PaConvert.git>
 - 调用PaConvert：
 - python PaConvert/paconvert/main.py --in_dir ./parseq （源码所在文件夹）
 - 整个项目的转换结果会存储在./paddle_project 文件夹中

2. 手工修正无法自动转换的代码：

- PaConvert只能进行粗糙地代码转换，因此仍然存在一些无法自动转换的代码，需要我们手工进行修正
- 无法自动转换的代码会以两种形式出现：
 - 显式无法转换的代码：
 - 这类代码会在相应行出现一个">>>"前缀，这类代码可在对齐前修改，如下图所示的torch.nn.MultiheadAttention和paddle.nn.MultiHeadAttention

```
>>> self.self_attn = torch.nn.MultiheadAttention(d_model, nhead,
>>> dropout=dropout, batch_first=True)
>>> self.cross_attn = torch.nn.MultiheadAttention(d_model, nhead,
>>> dropout=dropout, batch_first=True)
```

不支持直接转换的API会带有一个">>>"的前缀

- 转换成功，但存在Bug的代码：
 - 这类代码需要具体跑通网络时才能发现，可以在对齐的过程中修改，如paddle.nn.embedding.weight._padding_idx 对应 torch.nn.embdding.weight.padding_idx，两者在padding_idx这个变量上相差了一个前缀的"_"，自动转换会忽略该前缀
- 代码的修正有以下两种方式：
 - 涉及API差异的代码：
 - 这类代码通常可以通过手动查询[Torch2Paddle](#)进行替换，此时需要注意查看Paddle和Torch的官方文档，根据参数的含义以及返回结果进行转换。下面记录两个涉及API差异进行转换的代码：
 - MultiheadAttention
 - paddle的默认为batch_first模式，因此不像torch一样具有该参数
 - paddle默认不返回attention_weight，需要指定need_weights=True
 - 推理的参数没有key_padding_mask（巨坑，但可以间接实现该功能，参考[Issue](#)）
 - 参数初始化的操作：
 - 可以从[PaddleSeg/Param_init.py](#)中复制
 - 涉及外部开源库的代码：
 - 如果涉及到一些因外部开源库而产生的无法转换代码，则需要手动增加代码，以替换开源库，如timm的VisionTransformer，可以通过从PaddleClas的backbone中复制相应的代码进行替换，替换后记得对照两者的文档检查参数和具体计算过程是否一致。

3. 前反向对齐：

- 前反向对齐的在代码运行层面的验证代码转换的正确性，前向指的是网络前向推理的过程，反向指的是梯度反向传播更新参数的过程。前反向对齐可以利用PaDiff工具进行，但该工具还在早期开发阶段，可能容易出现bug，所以建议采用Paddiff初始化+常规对齐的方式进行对齐。
- 利用对齐工具[PaDiff](#)对齐

- PaDiff主要分为三步：实例化模型，复制参数，自动计算Diff
- 主要可以参考[使用PaDiff工具对齐ViTPose流程示例](#)进行对齐。但我只进行到复制参数那步，自动计算Diff的过程出bug了，所以在利用PaDiff的参数复制功能统一两个网络的权重后，我选择利用常规的对齐方法进行对齐。
- 注意事项：
 - 安装：PaDiff需要通过从github上拉取最新代码安装，不要通过pip安装，因为官方文档对应的是最新版的develop分支
 - 注意对齐前需要打开eval()模式，否则会对齐失败
 - 实例化模型时需要修正网络中存在的代码转换失败的Bug
 - 复制参数时，若网络中存在如MultiheadAttention这样的网络，需要利用auto_layer_map
 - 如果在自动计算Diff时出现了bug，建议直接开始用常规的前向对齐方法来做吧。
- 常规对齐方法：先用Padiff加载权重的方式统一两者的权重（即将torch的权重转化为paddle，然后让paddle载入该权重，或者用padiff复制随机初始化的权重），然后输入同一个随机图片，对比中间结果的输出。
 - 参考资料：[权重转换脚本参考](#)，[前向对齐](#)，[反向对齐](#)
 - 前向对齐：
 - 前向对齐需要用到reprod_log这个包，需要pip安装一下
 - 主要参考的是 [MobileNetv3对齐示例](#)中的4.1节
 - 注意事项：（血的教训） fake_input一定要设计为batch_size>1的，因为有些cases是只有在多batch时才会出现（比如padding操作），这个时候可能就会发现还未对齐的操作。
 - 对齐结果：

```
[2023/08/15 16:58:28] root INFO: logits:
[2023/08/15 16:58:28] root INFO:   mean diff: check passed: True, value: 6.7658784246305e-06
[2023/08/15 16:58:28] root INFO: diff check passed
```

前向对齐结果，diff_threshold==1e-5

- 代码：

</> Python | 收起 ^

```
1 @hydra.main(config_path='../parseq/configs', config_name='main', version_base='1.2')
2 def main(config: DictConfig):
3     with open_dict(config):
4
5         # 实例化两个模型
6         torch_model = PARSeq_torch(...)
7         paddle_model = PARSeq_paddle(...)
8
9         # 前向对齐需要开启eval()模式，以避免由dropout等层的影响
10        torch_model = torch_model.eval()
11        paddle_model.eval()
12
13        # 构造伪输入
14        inp = paddle.rand((1, 3, 32, 128)).numpy().astype("float32")
15        inp = ({'images': torch.as_tensor(inp) }, {'images': paddle.to_tensor(inp)})
16
17        # 用padiff初始化两个模型
18        module = create_model(torch_model)
19        module.auto_layer_map("base")
20        layer = create_model(paddle_model)
21        layer.auto_layer_map("raw")
22        assign_weight(module, layer)
23
24        # 前向推理
25        torch_out = module(inp[0]['images'])
26        paddle_out = layer(inp[1]['images'])
27
28        reprod_logger = ReprodLogger()
29        if not os.path.exists("./result"):
30            os.mkdir("./result")
31        # save the paddle output
32        reprod_logger.add("logits", paddle_out.cpu().detach().numpy())
33        reprod_logger.save("./result/forward_paddle.npy")
34        # save the torch output
35        reprod_logger.add("logits", torch_out.cpu().detach().numpy())
36        reprod_logger.save("./result/forward_torch.npy")
```

```
37
38     # load data
39     diff_helper = ReprodDiffHelper()
40     torch_info = diff_helper.load_info("./result/forward_torch.npy")
41     paddle_info = diff_helper.load_info("./result/forward_paddle.npy")
42     # compare result and produce log
43     diff_helper.compare_info(torch_info, paddle_info)
44     diff_helper.report(path="./result/log/forward_diff.log", diff_threshold=1e-5)
```

- 反面对齐：
 - 主要参考的是 [05_test_backward.py](#) 这份代码
 - 相比于前面对齐，增加了scheduler和optimizer，用同一份输入反复训练该网络，并反向传播
 - 原理：若反向的过程是对齐的，则每次网络参数更新后，下次的网络输出的loss应该是相近的
 - 坑：除了BN和dropout外，模型内可能还有一些额外的随机值（比如parsq里面的tgt_permut），需要注意去固定这类值，否则会导致两个网络的输出无法对齐
 - 对齐结果：

```
[2023/08/15 19:59:34] root INFO: loss_0:
[2023/08/15 19:59:34] root INFO:   mean diff: check passed: True, value: 9.5367431640625e-07
[2023/08/15 19:59:34] root INFO: lr_0:
[2023/08/15 19:59:34] root INFO:   mean diff: check passed: True, value: 0.0
[2023/08/15 19:59:34] root INFO: loss_1:
[2023/08/15 19:59:34] root INFO:   mean diff: check passed: True, value: 2.384185791015625e-06
[2023/08/15 19:59:34] root INFO: lr_1:
[2023/08/15 19:59:34] root INFO:   mean diff: check passed: True, value: 0.0
[2023/08/15 19:59:34] root INFO: loss_2:
[2023/08/15 19:59:34] root INFO:   mean diff: check passed: True, value: 3.814697265625e-06
[2023/08/15 19:59:34] root INFO: lr_2:
[2023/08/15 19:59:34] root INFO:   mean diff: check passed: True, value: 0.0
[2023/08/15 19:59:34] root INFO: diff check passed
```

反面对齐结果，differ_threshold==1e-5

- 代码：

</> Python | 收起 ^

```
1 def train_one_epoch_paddle(batch, tgt_perms, model, optimizer, lr_scheduler, max_iter, reprod_logger):
2     for idx in range(max_iter):
3         loss = model.model.training_step(batch, tgt_perms)
4         reprod_logger.add("loss_{}".format(idx), loss.cpu().detach().numpy())
5         reprod_logger.add("lr_{}".format(idx), np.array(lr_scheduler.get_lr()))
6         optimizer.clear_grad()
7         loss.backward()
8         optimizer.step()
9     reprod_logger.save("./result/losses_paddle.npy")
10
11 def train_one_epoch_torch(batch, tgt_perms, model, optimizer, lr_scheduler, max_iter, reprod_logger):
12     for idx in range(max_iter):
13         loss = model.model.training_step(batch, tgt_perms)
14         reprod_logger.add("loss_{}".format(idx), loss.cpu().detach().numpy())
15         reprod_logger.add("lr_{}".format(idx), np.array(lr_scheduler.get_last_lr()))
16         optimizer.zero_grad()
17         loss.backward()
18         optimizer.step()
19     reprod_logger.save("./result/losses_ref.npy")
20
21 @hydra.main(config_path='../parseq/configs', config_name='main', version_base='1.2')
22 def main(config: DictConfig):
23     with open_dict(config):
24
25         config.model.dropout = 0
26
27         # set determinnistic flag
28         torch.backends.cudnn.deterministic = True
29         torch.backends.cudnn.benchmark = False
30         FLAGS_cudnn_deterministic = True
31
32         # 实例化两个模型
33         torch_model = PARSeq_torch(...)
34         paddle_model = PARSeq_paddle(...)
```

```
35
36     # 前向对齐需要开启eval()模式，以避免由dropout等层的影响
37     torch_model.eval()
38     paddle_model.eval()
39
40     # 构造伪输入和伪标签
41     inp = paddle.rand((1, 3, 32, 128)).numpy().astype("float32")
42     label_str = ["okidoki",]
43     inp = ({'images': torch.as_tensor(inp), 'labels':label_str}, {'images': paddle.to_tensor(inp),
'labels':label_str})
44
45     # 固定模型中除了bn和dropout外的随机数tgt_permut，将其以额外参数的形式传进网络，网络中也需要相应地进行修改
46     tgt_permut = np.array([[0, 1, 2, 3, 4, 5, 6, 7, 8],]*6)
47
48     # 用padi f f初始化两个模型
49     module = create_model(torch_model)
50     module.auto_layer_map("base")
51     layer = create_model(paddle_model)
52     layer.auto_layer_map("raw")
53     assign_weight(module, layer)
54
55     # 定义两个优化器
56     max_iter = 3
57     lr = 1e-3
58     momentum = 0.9
59     lr_gamma = 0.1
60
61     # init optimizer
62     opt_paddle = paddle.optimizer.Momentum(learning_rate=lr, momentum=momentum,
parameters=layer.model.parameters())
63     opt_torch = torch.optim.SGD(module.model.parameters(), lr=lr, momentum=momentum)
64     lr_scheduler_paddle = paddle.optimizer.lr.StepDecay(lr, step_size=max_iter // 3, gamma=lr_gamma)
65     lr_scheduler_torch = lr_scheduler.StepLR(opt_torch, step_size=max_iter // 3, gamma=lr_gamma)
66
67     reprod_logger = ReprodLogger()
68
69     # 反向Loss计算
70     # torch_loss = module.model.training_step((inp[0]['images'], inp[0]['labels']), tgt_perms)
71     # paddle_loss = layer.model.training_step((inp[1]['images'], inp[1]['labels']), tgt_perms)
72
73     train_one_epoch_paddle((inp[1]['images'], inp[1]['labels']), tgt_permut, layer, opt_paddle,
lr_scheduler_paddle, max_iter, reprod_logger)
74     train_one_epoch_torch((inp[0]['images'], inp[0]['labels']), tgt_permut, module, opt_torch,
lr_scheduler_torch, max_iter, reprod_logger)
75
76     # load data
77     diff_helper = ReprodDiffHelper()
78     torch_info = diff_helper.load_info("./result/losses_ref.npy")
79     paddle_info = diff_helper.load_info("./result/losses_paddle.npy")
80
81     # compare result and produce log
82     diff_helper.compare_info(torch_info, paddle_info)
83     diff_helper.report(path="./result/log/backward_diff.log", diff_threshold=1e-5)
```

3. 框架层面的代码迁移

- 在项目层面转换完代码并进行前反向对齐后，我们还需要将项目中的组网，前处理，后处理，优化器等代码分别放置到Paddle的官方框架中
- 以OCR为例， [官方新增模型要求](#)新增数据加载和预处理，组网，后处理，损失函数，指标评估，优化器相关的代码。因此需要手动将转换产生的项目中的代码逐块放入其中。
- **注意事项：**
 - 代码迁移完成后，可能还会发现一些在步骤2中未发现的一些实现层面的差异，需要再次进行代码转换
 - 预处理和后处理也很关键，若不清楚，需要先把Torch中相关的代码Debug一遍，了解清楚相关的细节

4. 模型训练

- 代码迁移完毕后，即可利用框架中的指令对新模型进行训练，并进行评估
- 将评估结果与1中复现的Torch模型的评估结果对比，询问相关RD现有的Diff是否满足复现的标准。

与Torch模型在Real(COCO-Text, RCTW17, Uber-Text, ArT, LSVT, MLT19, ReCTS, TextOCR and OpenVINO)数据集下训练，在IIIT5k, SVT, IC13,IC15,SVTP,CUTE数据集下测试结果对比:

1	Method	Train data	IIIT5k	SVT	IC13		IC15		SVTP	CUTE	Mean
2			3,000	647	857	1015	1811	2077	645	288	
3	PARSeq_N (Torch)	R	97.60	97.53	97.20	97.34	88.57	87.24	93.02	96.88	93.57
4	PARSeq_N (PPOCR)	R	97.50	96.91	97.08	97.14	88.63	87.53	92.87	96.53	93.52
5	PARSeq_A (Torch)	R	98.93	97.84	97.90	98.03	89.90	88.64	94.57	97.92	94.74
6	PARSeq_A (PPOCR)	R	98.23	97.68	97.78	97.83	90.28	89.17	94.88	96.53	94.65

训练结果对比

- 注意事项：
 - 记得将训练集的dataloader的shuffle设为True
 - 优化器，数据增强等操作对训练结果影响较大，需要认真对齐

5. 部署推理：

- 基于训练好的模型，还需要进行部署推理的测试，包括：
 - 静态图模型的导出：该步骤可能出现Bug，需要及时修正
 - 部署推理的测试：需要在测试样例中测试导出的模型是否正确

静态图模型导出成功：
[2023/09/06 19:26:40] ppocr INFO: load pretrain successful from ./output/rec/parseq/best_accuracy
[2023/09/06 19:26:45] ppocr INFO: inference model is saved to ./inference/rec_parseq/inference ...

Python部署推理结果验证通过：

[2023/09/06 19:37:44] ppocr INFO: Predicts of ./doc/imgs_words/en/word_1.png:('JOINT', 26.278064727783203)
[2023/09/06 19:37:44] ppocr INFO: Predicts of ./doc/imgs_words/en/word_2.png:('yourself', 24.12157440185547)
[2023/09/06 19:37:44] ppocr INFO: Predicts of ./doc/imgs_words/en/word_3.png:('154', 18.765256881713867)
[2023/09/06 19:37:44] ppocr INFO: Predicts of ./doc/imgs_words/en/word_4.png:('197', 25.423019409179688)
[2023/09/06 19:37:44] ppocr INFO: Predicts of ./doc/imgs_words/en/word_5.png:('727', 21.823705673217773)

静态图导出提示信息以及样例测试结果

6. 文档编写

- 打包压缩训练好的模型和相应的yml文件，上传至云中
- 编写中英文的使用文档（./doc/doc_cn和./doc/doc_en），并填入模型权重的下载地址
- 在相应的overview登记新模型（文档链接，模型精度等）