第五章 基于Hydra实现模型的高级管理

作业布置 & 参考资料

- 作业：
  - 1. 实践作业：使用Hydra配置一个简单的分类模型（如Iris数据集），并通过命令行调整优化器类型（SGD/Adam）。
  - 2. 思考题：解释Hydra的hydra.run.dir参数在实验管理中的作用，并说明如何通过它实现结果的可追溯性？
- 参考资料：
  - 1. Hydra官方文档
  - 2. 配置可视化工具：hydra-config-viewer

为什么需要Hydra这样的东西？

做对比、消融实验时，参数过多造成的混乱

比如Evo中有替代Hyena算子为多头注意力的实验
如果10层逐一替代，需要至少写10个模型结构，然后运行10次

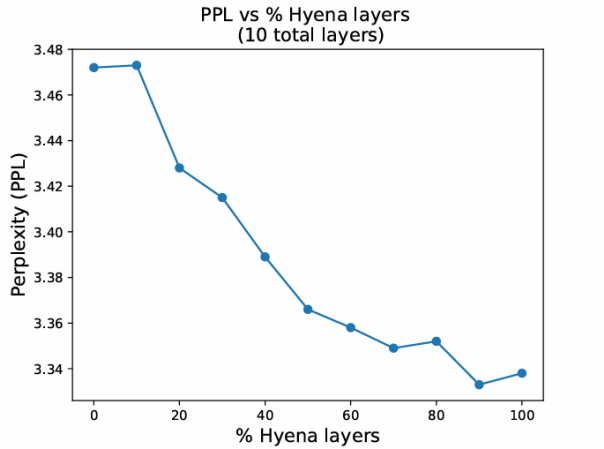


Fig. S7 | Perplexity comparison by amount of hyena layer percentage. We progressively replace attention for hyena layers and observe improved perplexity, with an optimal hybrid ratio of 90% hyena and 10% attention. All models use 10 layers and a width of 768. Hyperparameters and FLOPs are controlled.

修改10次实验模型结构→运行10次

改10个config文件→运行10次

```python
# 定义一个12层的混合架构：第3,6,9层用MHA，其余用SSM
for layer_idx in range(12):
    mixer_cls = create_mixer_cls(
        layer=ssm_config,                          # 默认SSM配置
        attn_layer_idx=[3, 6, 9],                  # 指定MHA层
        attn_cfg={"num_heads": 8, "causal": True},
        layer_idx=layer_idx,
        process_group=parallel_group
    )
    # 实际使用时实例化
    mixer = mixer_cls(hidden_dim=768)
    # 该mixer可能是SSM或MHA实例，但对外接口一致
```

```python
def create_mixer_cls(
    layer=None,
    process_group=None,
    attn_layer_idx=None,
    attn_cfg=None,
    layer_idx=None,
    sequence_parallel=True,
    device=None,
    dtype=None,
):
    factory_kwargs = {"device": device, "dtype": dtype}
    parallel_kwargs = (
        {"process_group": process_group, "sequence_parallel": sequence_parallel}
        if process_group is not None
        else {}
    )
    if attn_layer_idx is not None and layer_idx in attn_layer_idx:
        causal = True if attn_cfg is None else attn_cfg.pop("causal", True)
        fused_bias_fc = (
            False if attn_cfg is None else attn_cfg.get("fused_bias_fc", False)
        )
        if not fused_bias_fc:
            assert process_group is None, "TensorParallel MHA requires fused_bias_fc"
        mha_cls = MHA if process_group is None else ParallelMHA
        # ParallelMHA doesn't take 'fused_bias_fc', it is assumed that we fuse matmul + bias
        if process_group is not None:
            attn_cfg = copy.deepcopy(attn_cfg)  # Don't modify the original cfg
            attn_cfg.pop("fused_bias_fc", None)
        mixer_cls = partial(
            mha_cls,
            causal=causal,
            layer_idx=layer_idx,
            **(attn_cfg if attn_cfg is not None else {}),
            **parallel_kwargs,
            **factory_kwargs,
        )
    else:
        fused_bias_fc = False if layer is None else layer.get("fused_bias_fc", False)
        if process_group is not None:
            assert fused_bias_fc, "TensorParallel SSM requires fused_bias_fc"
        mixer_cls = instantiate(
            registry.layer,
            layer,
            partial=True,
            layer_idx=layer_idx,
            **factory_kwargs,
            **parallel_kwargs,
        )
        # mixer_cls = partial(ssm_cls, layer_idx=layer_idx,
        #                     **(ssm_cfg if ssm_cfg is not None else {}),
        #                     **parallel_kwargs, **factory_kwargs)
    return mixer_cls
```

默认MHA模块，即transformer里的多头注意力

改为自定义的计算层

为什么需要Hydra这样的东西？

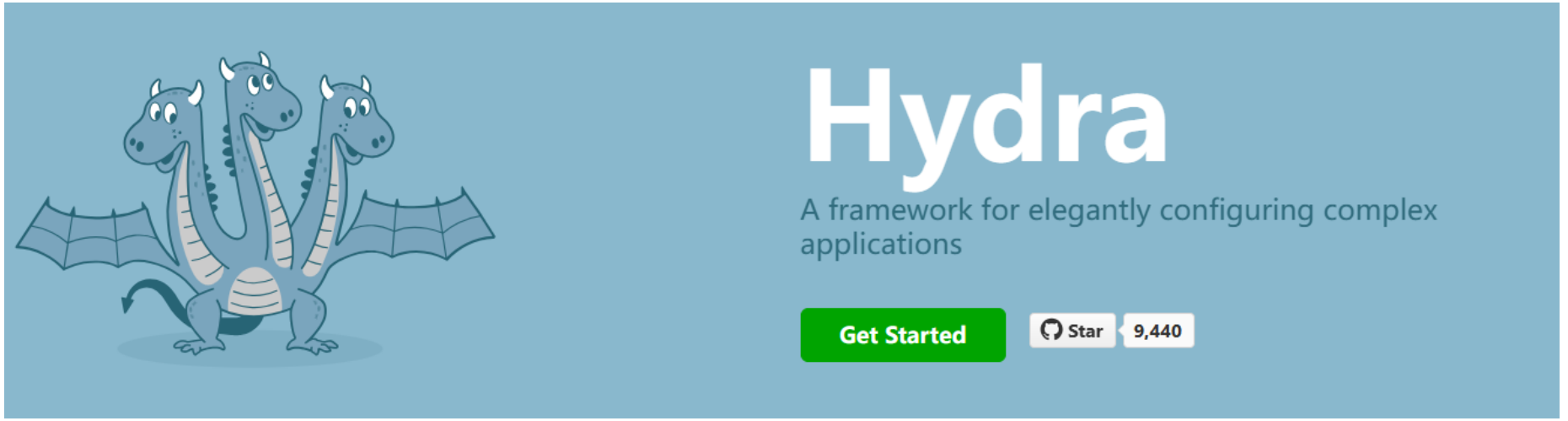
OmegaConf

Overview

OmegaConf is a YAML based hierarchical configuration system, with support for merging configurations from multiple sources (files, CLI argument, environment variables) providing a consistent API regardless of how the configuration was created. OmegaConf also offers runtime type safety via Structured Configs.

OmegaConf 是一个灵活的 Python 配置管理库，由 Facebook AI（Meta）开发，主要用于处理嵌套结构的配置文件（如 YAML、JSON 或 Python 字典），并支持动态修改和合并配置。

Hydra 是基于 OmegaConf 的高级配置框架，由 Meta 开发，主要用于管理复杂的实验配置，支持动态配置组合、命令行覆盖和多实验并行运行。



OmegaConf

test_config.yaml

```yaml
model:
  name: "resnet50"
  params:
    lr: 0.001
    batch_size: 32
data:
  train_path: "/data/train"
  test_path: "/data/test"
```

```python
from omegaconf import OmegaConf

### load yaml
config = OmegaConf.load("test_config.yaml")
```

0.0s

```python
print(f"Config info: {config}")
print(f"Model info: {config['model']}")
print(f"Model name: {config['model']['name']}")
```

0.0s

```
Config info: {'model': {'name': 'resnet50', 'params': {'lr': 0.001, 'batch_size': 32}}, 'dat
Model info: {'name': 'resnet50', 'params': {'lr': 0.001, 'batch_size': 32}}
Model name: resnet50
```

OmegaConf

访问配置项

```
config['model']['name']
```
✓ 0.0s

```
'resnet50'
```

```
config.model.name
```
✓ 0.0s

```
'resnet50'
```

##修改配置文件

```
config.model.params.batch_size
```
✓ 0.0s

```
32
```

```
config.model.params.batch_size = 128
config.model.params.batch_size
```
✓ 0.0s

```
128
```

##嵌套配置　类似于字典嵌套字典

```yaml
model:
  name: "resnet50"
  params:
    lr: 0.001
    batch_size: 32
data:
  train_path: "/data/train"
  test_path: "/data/test"
```

继承配置　　类似于字典合并

base_config.yaml

```yaml
defaults:
  - _self_

training:
  batch_size: 64
  learning_rate: 0.001
  optimizer: "adam"
  epochs: 100

model:
  name: "resnet18"
  pretrained: True

data:
  root_dir: "/datasets/imagenet"
  num_workers: 8
```

custom_config.yaml

```yaml
_base_: base_config.yaml

training:
  batch_size: 256  # 覆盖基础配置的 batch_size
  learning_rate: 0.01  # 覆盖学习率
  epochs: 50  # 减少训练轮次

model:
  name: "efficientnet_b0"  # 更换模型架构

data:
  root_dir: "/datasets/custom_imagenet"  # 使用自定义数据集路径
```

```
base_cfg = OmegaConf.load("base_config.yaml")
custom_cfg = OmegaConf.load("custom_config.yaml")
```
✓ 0.0s

```
merged_cfg = OmegaConf.merge(base_cfg, custom_cfg)
```
✓ 0.0s

```
custom_cfg.training.optimizer
```
⊗ 0.4s

```
---------------------------------------------------------------
ConfigAttributeError                      Traceback (most
Cell In[29], line 1
----> 1 custom_cfg.training.optimizer
```

```
merged_cfg.training.optimizer
```
✓ 0.0s

```
'adam'
```

custom的config仅携带custom的属性值

merge后的config携带有custom和base的属性值

OmegaConf化简pl iris代码

iris.yaml

```yaml
model:
  n_feature: 4
  n_hidden: 16
  n_output: 3
  lr: 0.01

trainer:
  default_root_dir: outputs/
  accelerator: gpu
  devices: [0]
  max_epochs: 5
  check_val_every_n_epoch: 1
  logger: true
  callbacks:
    model_checkpoint:
      dirpath: outputs/
      save_top_k: 1
      save_last: true
      monitor: valid_accuracy
      save_on_train_epoch_end: true
      mode: max
      filename: best_model_{epoch:02d}-{valid_accuracy:.2f}

data:
  path: "iris/iris.csv"
  batch_size: 16
```

定义配置文件

这里要注意，这些参数都是传给class的值，所以就需要保证配置文件中key要跟class的属性值对上，这样才能实现映射

```python
df_train, df_valid, df_test = generate_dataset(config.data.path)
batch_size = config.data.batch_size

train_ds = Iris_Dataset(df_train)
valid_ds = Iris_Dataset(df_valid)
train_dl = torch.utils.data.DataLoader(train_ds, shuffle = True, batch_size = batch_size, num_workers = 4)
valid_dl = torch.utils.data.DataLoader(valid_ds, shuffle = True, batch_size = batch_size, num_workers = 4)
```

```python
model_save_dir = "./pl_train_output"
n_feature = 4
n_hidden = 16
n_output = 3
max_epoch = 10
batch_size = 8
learning_rate = 0.01
monitor = "valid_accuracy"
check_val_every_n_epoch = 1

### 参数初始化
model = Lit_iris_classifier(n_feature = n_feature, n_hidden = n_hidden, n_output = n_output,lr = learning_rate)

### OmegaConf初始化
model = Lit_iris_classifier(**config.model)
```

```python
### 参数初始化
trainer = pl.Trainer(
        default_root_dir=model_save_dir,
        accelerator="gpu" if torch.cuda.is_available() else 'auto',
        devices = [0],
        max_epochs = max_epoch,
        callbacks=[
            ModelCheckpoint(…
        ],
        logger=True,
        check_val_every_n_epoch = check_val_every_n_epoch
    )
```

```python
### OmegaConf初始化
config.trainer.accelerator = "gpu" if torch.cuda.is_available() else 'auto'
checkpoint_callback = ModelCheckpoint(**config.trainer.callbacks.model_checkpoint)
trainer = pl.Trainer(
    **{k: v for k, v in config.trainer.items() if k != "callbacks"},
    callbacks=[checkpoint_callback]
)
```

hydra

某种意义上说hydra是OmegaConf的plus版

OmegaConf 管理配置内容，hydra 管理配置生命周期和运行流程

hydra可以直接访问到base_config的配置，而无需像OmegaConf一样需要merge(实现了真正的继承)

hydra_custom_config.yaml

```yaml
defaults:
  - base_config.yaml  # 自动继承基础配置
  - _self_            # 允许当前文件覆盖

training:
  batch_size: 256
  learning_rate: 0.01
  epochs: 50

model:
  name: "efficientnet_b0"

data:
  root_dir: "/datasets/custom_imagenet"
```

```python
### hydra_test.py
import hydra
from omegaconf import DictConfig

@hydra.main(config_path=".", config_name="hydra_custom_config", version_base="1.1")
def train(cfg: DictConfig):
    print(cfg.training.optimizer)

train()
```

Generate Code Markdown

```
%run hydra_test.py
```

✓ 0.4s

```
adam
c:\Users\YoeHuuui\.conda\envs\hyena-dna\lib\site-packages\hydra\_internal\hydra.py:119: Us
See https://hydra.cc/docs/1.2/upgrades/1.1_to_1.2/changes_to_job_working_dir/ for more inf
  ret = run_job(
```

hydra

在 Hydra 配置中，_target_ 表示你想实例化的 Python 类或函数的路径，Hydra 会根据这个路径自动调用构造函数，并传入配置中的其他参数

hydra_torch.yaml

```yaml
model:
  _target_: torch.nn.Linear
  in_features: 128
  out_features: 10
```

```python
from hydra.utils import instantiate
from omegaconf import OmegaConf

cfg = OmegaConf.load("./hydra_torch.yaml")
model = instantiate(cfg.model)
model
```

✓ 0.0s

```
Linear(in_features=128, out_features=10, bias=True)
```

嵌套调用也不成问题

```yaml
trainer:
  _target_: pytorch_lightning.Trainer
  check_val_every_n_epoch: 1
  logger: true
  callbacks:
    - _target_: pytorch_lightning.callbacks.ModelCheckpoint
      save_top_k: 1
      save_last: true
      save_on_train_epoch_end: true
      mode: max
      filename: "best_model_{epoch:02d}-{valid_accuracy:.2f}"
```

```python
cfg = OmegaConf.load("./hydra_trainer.yaml")
trainer = instantiate(cfg.trainer)
trainer.check_val_every_n_epoch
```

✓ 0.0s

```
GPU available: False, used: False
TPU available: False, using: 0 TPU cores
HPU available: False, using: 0 HPUs

1
```

hydra化简pl iris代码

🔄 与 Config Groups 联合使用

你可以将多个 `_target_` 放入 `configs/` 目录下作为可选模块：

```
configs/
├── model/
│   ├── resnet.yaml
│   └── transformer.yaml
├── optimizer/
│   ├── adam.yaml
│   └── sgd.yaml
```

yaml

```yaml
# config.yaml
defaults:
  - model: resnet
  - optimizer: adam
```

运行：

bash

```bash
python train.py model=transformer optimizer=sgd
```

```
$ tree -h configs/
configs/
├── [4.0K]  callbacks
│   ├── [ 322]  base.yaml
│   ├── [1.1K]  checkpoint.yaml
│   ├── [  89]  gpu_affinity.yaml
│   ├── [ 203]  rich.yaml
│   └── [ 667]  wandb.yaml
├── [3.1K]  config.yaml
├── [4.0K]  dataset
│   ├── [ 407]  chromatin_profile.yaml
│   ├── [1.7K]  genomic_benchmark.yaml
│   ├── [ 289]  hg38_fixed_test.yaml
│   ├── [ 380]  hg38_methyl.yaml
│   ├── [ 349]  hg38.yaml
│   ├── [ 515]  icl_hg38.yaml
│   ├── [2.4K]  nucleotide_transformer.yaml
│   └── [ 568]  species.yaml
├── [4.0K]  evals
│   ├── [2.2K]  hg38_decoder.yaml
│   ├── [ 636]  hg38.yaml
│   ├── [ 471]  hyena_dna_512ksl.yaml
│   ├── [ 963]  icl_genomics.yaml
│   ├── [1.0K]  instruction_tuned_genomics.yaml
│   └── [1.2K]  soft_prompting_genomics.yaml
├── [4.0K]  experiment
│   ├── [ 189]  base.yaml
│   └── [4.0K]  hg38
│       ├── [2.6K]  chromatin_profile.yaml
│       ├── [3.7K]  genomic_benchmark_attention.yaml
│       ├── [3.8K]  genomic_benchmark_load_finetuned_model.yaml
│       ├── [3.5K]  genomic_benchmark_scratch.yaml
│       ├── [3.5K]  genomic_benchmark.yaml
│       ├── [2.3K]  hg38_attention.yaml
│       ├── [2.1K]  hg38_fixed_test_attention.yaml
│       ├── [1.9K]  hg38_fixed_test.yaml
│       ├── [2.4K]  hg38_hyena_icl.yaml
│       ├── [3.0K]  hg38_hyena_seqlen_warmup_reload.yaml
│       ├── [2.2K]  hg38_hyena.yaml
│       ├── [2.4K]  hg38_methyl.yaml
│       ├── [1.5K]  hg38.yaml
│       ├── [2.5K]  nucleotide_transformer.yaml
│       ├── [2.5K]  species_attention.yaml
│       ├── [4.0K]  species_seqlen_warmup_reload.yaml
│       └── [2.7K]  species.yaml
├── [4.0K]  loader
│   └── [ 125]  default.yaml
├── [4.0K]  model
│   ├── [4.0K]  layer
│   │   ├── [ 105]  ff.yaml
│   │   ├── [ 111]  h3-conv.yaml
│   │   ├── [ 335]  hyena_dna.yaml
│   │   ├── [ 351]  hyena-filter.yaml
│   │   ├── [ 275]  hyena.yaml
│   │   ├── [  11]  id.yaml
│   │   ├── [ 125]  mha_dna.yaml
│   │   └── [ 123]  mha.yaml
│   └── [ 164]  transformer.yaml
├── [4.0K]  optimizer
│   ├── [ 132]  adamw.yaml
│   ├── [ 184]  adam.yaml
│   └── [ 137]  sgd.yaml
├── [4.0K]  pipeline
│   ├── [ 387]  chromatin_profile.yaml
│   ├── [ 388]  genomic_benchmark.yaml
│   ├── [ 336]  hg38.yaml
│   ├── [ 446]  nucleotide_transformer.yaml
│   └── [ 378]  species.yaml
├── [4.0K]  scheduler
│   ├── [ 205]  constant_warmup.yaml
│   ├── [ 124]  constant.yaml
│   ├── [ 234]  cosine_warmup_timm.yaml
│   ├── [ 191]  cosine_warmup.yaml
│   ├── [ 248]  cosine.yaml
│   ├── [ 191]  linear_warmup.yaml
│   ├── [ 165]  multistep.yaml
│   ├── [ 346]  plateau.yaml
│   └── [ 145]  step.yaml
├── [4.0K]  task
│   ├── [  84]  lm.yaml
│   ├── [  86]  masked_multiclass_classification.yaml
│   ├── [ 115]  multiclass_classification.yaml
│   ├── [ 122]  multilabel_classification.yaml
│   └── [  88]  regression.yaml
└── [4.0K]  trainer
    ├── [ 328]  debug.yaml
    ├── [ 442]  default.yaml
    ├── [1.1K]  full.yaml
    └── [ 643]  lm.yaml
```

hydra化简pl iris代码

```yaml
model:
  _target_: lit_iris_classifier.Lit_iris_classifier
  n_feature: 4
  n_hidden: 16
  n_output: 3
  lr: 0.01

trainer:
  _target_: pytorch_lightning.Trainer
  default_root_dir: outputs/
  accelerator: gpu
  devices: 1
  max_epochs: 5
  check_val_every_n_epoch: 1
  logger: true
  callbacks:
    - _target_: pytorch_lightning.callbacks.ModelCheckpoint
      dirpath: outputs/
      save_top_k: 1
      save_last: true
      monitor: valid_accuracy
      save_on_train_epoch_end: true
      mode: max
      filename: best_model_{epoch:02d}-{valid_accuracy:.2f}

data:
  path: "iris/iris.csv"
  batch_size: 16
```

```
model = instantiate(config.model)
model
```

✓ 0.0s

```
Lit_iris_classifier(
  (model): Iris_classifier(
    (hidden): Linear(in_features=4, out_features=16, bias=True)
    (relu): ReLU()
    (out): Linear(in_features=16, out_features=3, bias=True)
  )
  (criterion): CrossEntropyLoss()
)
```

```
config.trainer.accelerator = "gpu" if torch.cuda.is_available() else 'auto'
### 测试电脑没cuda
config.trainer.devices = 1
trainer = instantiate(config.trainer)
```

✓ 0.0s

```
GPU available: False, used: False
TPU available: False, using: 0 TPU cores
HPU available: False, using: 0 HPUs
```

总结

OmegaConf定义参数，进行简单的参数传入

Hydra可以进一步以_target_方式为类传递参数

	OmegaConf	Hydra
本质	轻量级配置管理库	基于 OmegaConf 的高级配置框架
开发方	Meta (Facebook)	Meta (Facebook)
依赖关系	独立库	依赖 OmegaConf 作为底层引擎

功能	OmegaConf	Hydra
加载 YAML 配置	✅ `OmegaConf.load("config.yaml")`	✅ `@hydra.main(...)` 自动加载
创建配置对象	✅ `OmegaConf.create({...})`	❌ 通常通过 YAML 文件实现
变量插值 `${var}`	✅ 支持	✅ 支持（同源）
动态实例化对象	✅ `instantiate(cfg.model)`	✅ 自动支持
组合多 YAML 配置	手动合并 `OmegaConf.merge()`	✅ 支持 Config Group & Override
配置 CLI 注入	❌ 需手动实现	✅ 支持 `python script.py param=value`
多实验多种配置管理	❌	✅ Hydra 核心能力
自动运行目录 `outputs/`	❌	✅ 内建
多个运行任务（sweeper）	❌	✅ 支持 GridSearch、Optuna