第七章 HyenaDNA模型训练

作业布置 & 参考资料

- 作业：
  - 1. 实践作业：使用Hydra配置文件自定义Hyena-DNA模型结构（如调整d_model为256、n_layer为4），并训练处理长序列DNA数据。
  - 2. 思考题：解释为何Hyena-DNA在长序列处理中参数量更低、速度更快，其核心创新点是什么？
- 参考资料：
  - 1. HyenaDNA原始论文：https://arxiv.org/abs/2306.15794
  - 2. HyenaDNA 项目常见问题解决方案（CSDN博客）

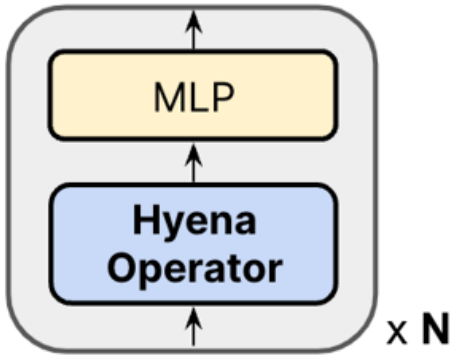
Hyena-dna

Sample from Human Genome

A C G T A C G T

Next token prediction

A C G T A C G T



A C G T A C G ?

(Single Nucleotide Tokens)

Downstream tasks

Regulatory element classification (short)

Chromatin profile, species classification (long)

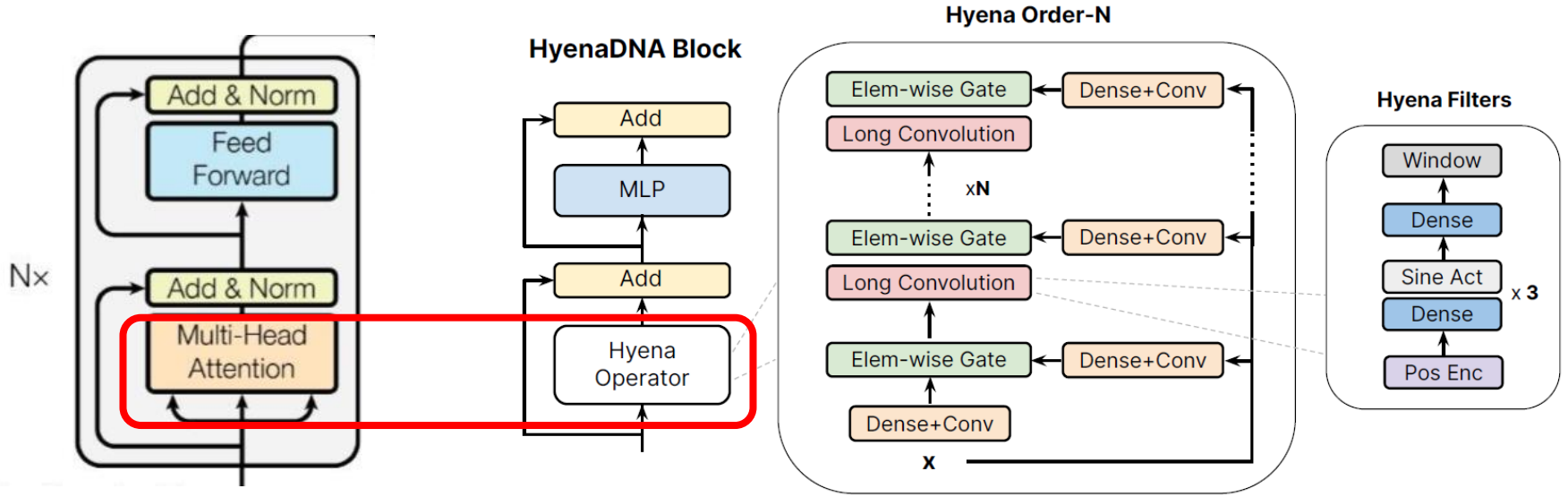
Explore in-context learning

Hyena-dna是decoder-only，用于下一个词元预测

单碱基分辨率、长序列

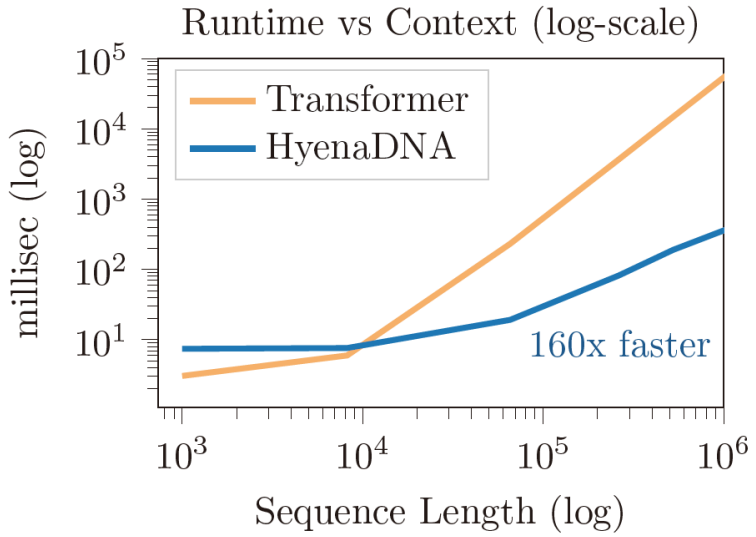
Hyena-dna

Transformer block



本质是Hyena Operator 替代Multi-Head Attention

好处就是在长序列上，处理速度快很多，且参数量大大降低



Next-token prediction

什么是 Next-token prediction?

Next-token prediction，顾名思义，就是在已知的文本序列基础上预测下一个最可能出现的词语或符号。这一任务是生成式语言模型的核心，驱动了从文本生成到代码补全等广泛的应用。

具体来说，Next-token prediction 的基本工作原理如下：

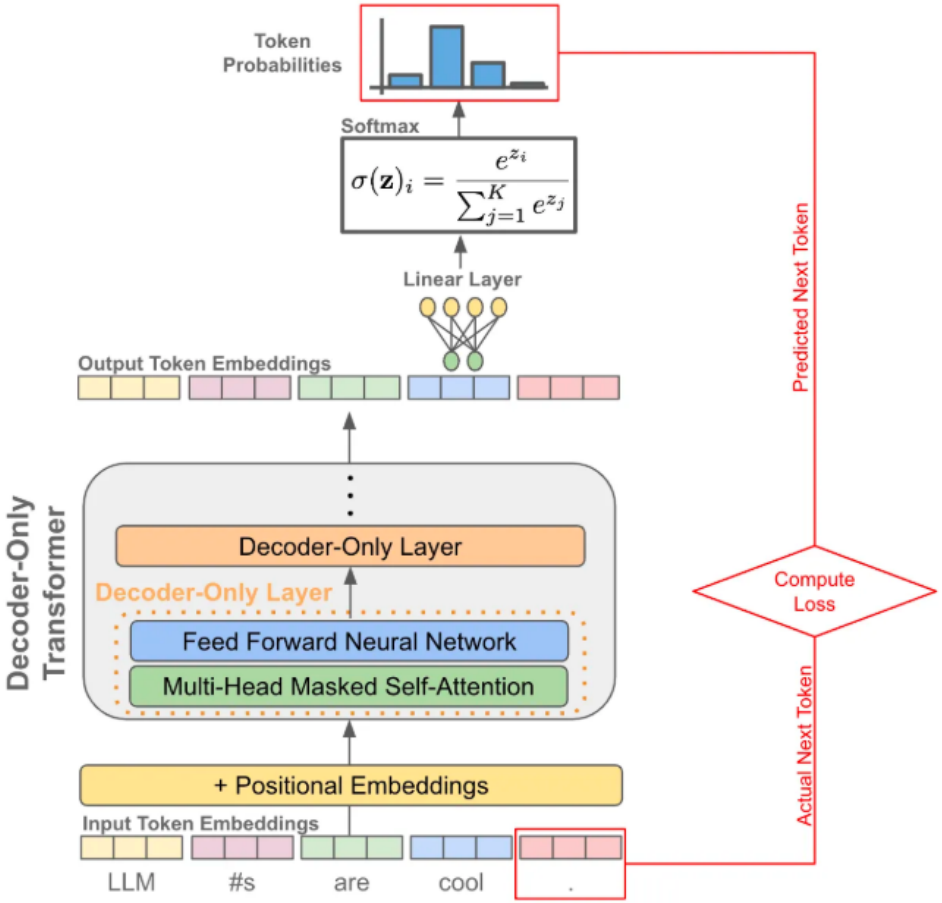
1. **输入序列：** 给定一段文本，模型会将其转化为数值表示（嵌入向量）。
2. **模型计算：** 基于输入，模型会通过多层计算，输出一个概率分布，表示每个可能词语出现的概率。
3. **输出选择：** 根据概率分布，模型选择最高概率的词语作为输出。

这一过程中，模型依赖于大量训练数据和复杂的数学结构，能够捕捉语义和上下文关系，从而生成合理、连贯的文本。

Next-token prediction 的独特性

与传统人工智能任务相比，Next-token prediction 展现了以下独特特性：

1. **生成能力：** 不仅能理解已有数据，还能基于上下文生成新的、语义连贯的内容。
2. **任务通用性：** 可用于翻译、写作、对话等多种任务，展现出极大的灵活性。
3. **数据驱动：** 完全依赖于训练数据，模型的能力直接反映了数据的质量与规模。

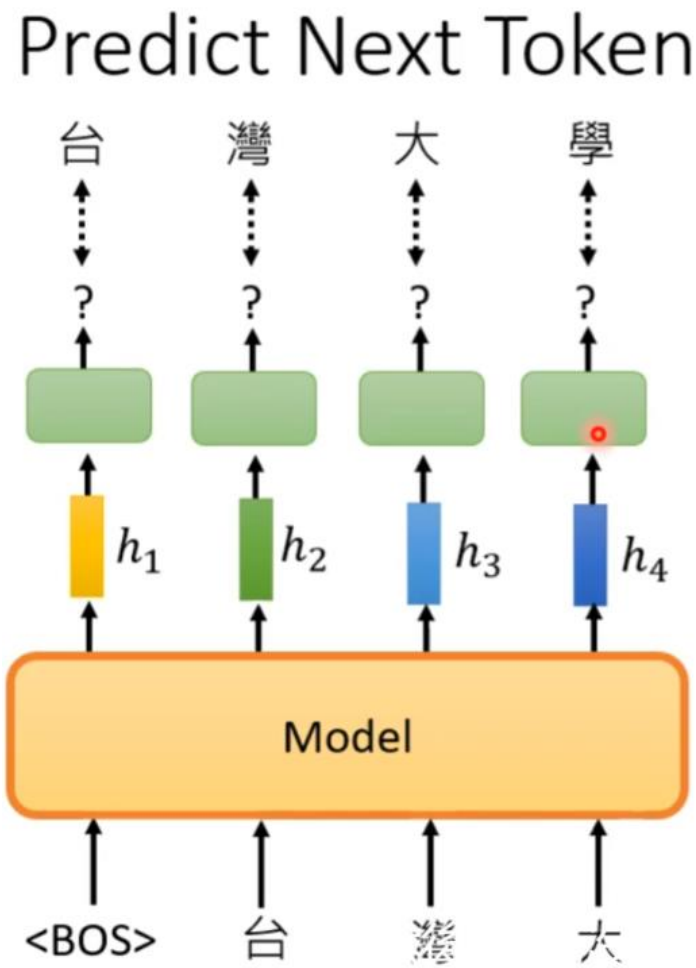


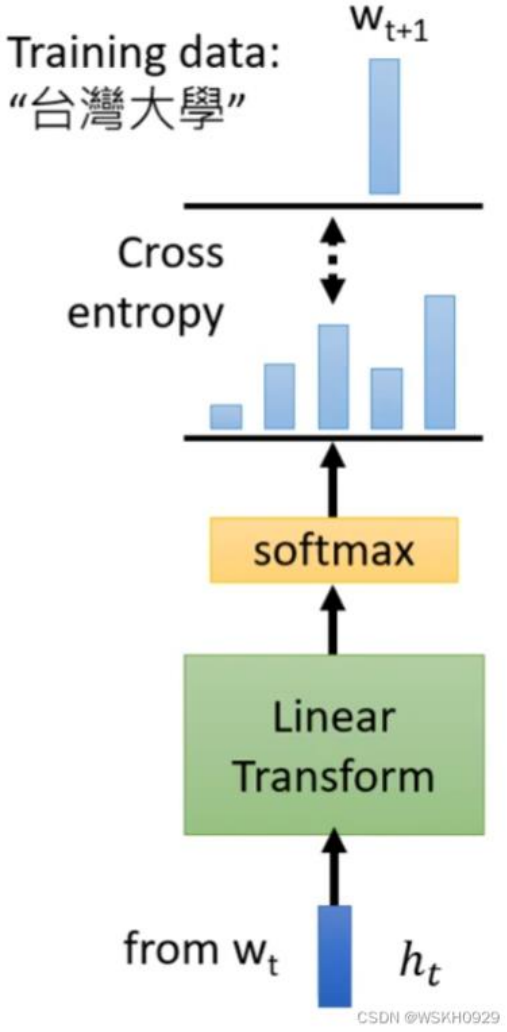
[从GPT到o3：Next-token Prediction 的核心奥秘（上）_人工智能_JustYan_InfoQ写作社区]

[从原理到代码理解语言模型训练和推理 - 知乎]

Next-token prediction

每个位置的 logits 代表的是下一个 token 的预测结果，而不是当前 token 本身





input_ids = [A, C, G, T]
# 假设对应的 token ID 是 [7, 8, 9, 10]

输入 token（input_ids）	输出 logits（logits[i]）预测的是	应该学到的 target
`A` (7)	模型预测下一个 token → `C` (8)	`target[0] = 8`
`C` (8)	模型预测下一个 token → `G` (9)	`target[1] = 9`
`G` (9)	模型预测下一个 token → `T` (10)	`target[2] = 10`

[【深度学习】李宏毅2021/2022春深度学习课程笔记 - Self-supervised Learning（自监督式学习）_李宏毅机器学习2022self-supervised learning笔记-CSDN博客]

Hyena-dna https://github.com/HazyResearch/hyena-dna

Pretraining on Human Reference Genome

The file structure should look like

```
data
|-- hg38/
    |-- hg38.ml.fa
    |-- human-sequences.bed
```

- Download fasta (.fa format) file (of the entire human genome) into hyena-dna/data/hg38. ~24 chromosomes in the whole genome (merged into 1 file), each chromosome is a continuous sequence, basically. Then download the .bed file with sequence intervals (contains chromosome name, start, end, split, which then allow you to retrieve from the fasta file)

```
mkdir -p data/hg38/
curl https://storage.googleapis.com/basenji_barnyard2/hg38.ml.fa.gz > data/hg38/hg38.ml.fa.gz
curl https://storage.googleapis.com/basenji_barnyard2/sequences_human.bed > data/hg38/human-sequences.b
```

```
python -m train wandb=null experiment=hg38/hg38_hyena model.d_model=128 model.n_layer=2 dataset.batch_s
```

Let's describe a little about this command.

- `experiment=hg38/hg38_hyena` passes the config for this experiment using a Hyena(DNA) model
- `model.d_model=128`, and `model.n_layer=2` select the model width and depth, key hyparams
- `dataset.max_length=1024` sets the max sequence length sampled from the dataset, the model layer max length is set from this too, or...
- `model.layer.l_max` # you can set the max model length manually
- `model.d_inner` # likewise, the reverse bottleneck with can be set manually too (default is 4x d_model)

Lots of other commands you can pass and customize, feel free to check out the `experiment=hg38/hg38_hyena` for details.

Hydra的参数传递

python -m train wandb=null
experiment=hg38/hg38_hyena
model.d_model=128 model.n_layer=2
dataset.batch_size=256
train.global_batch_size=256
dataset.max_length=1024 optimizer.lr=6e-4
trainer.devices=1

```
# @package _global_
defaults:
  - /pipeline: hg38
  - override /scheduler: cosine_warmup_timm

model:
  _name_: lm
  d_model: 32
  n_layer: 2
  d_inner: ${eval:4 * ${.d_model}}
  vocab_size: 11
  resid_dropout: 0.0
  embed_dropout: 0.1
  attn_layer_idx: [0, 1]
  attn_cfg:
    num_heads: 1
    use_flash_attn: True  # figure out how to use
    fused_bias_fc: True
    dropout: 0.1
  fused_mlp: False # figure out how to use fused MLP, maybe onl
  fused_dropout_add_ln: True
  residual_in_fp32: True
  pad_vocab_size_multiple: 8

trainer:
  accelerator: gpu
  devices: 1
  num_nodes: 1
  accumulate_grad_batches: ${div_up:${train.global_batch_size}, ${eval:${trainer.devic
  max_epochs: 100
  precision: 16  # bf16 only a100
  gradient_clip_val: 1.0
  # strategy: null

dataset:
  # batch_size: 32  # Per GPU
  batch_size: 16
  max_length: 700_000 # 262144

scheduler:
  t_in_epochs: False
  t_initial: ${eval:${div_up:${dataset.__train_len}, ${train.global_batch_size}} * ${t
  warmup_lr_init: 1e-6
  warmup_t: ${eval:${div_up:${dataset.__train_len}, ${train.global_batch_size}} * ${tr
  lr_min: ${eval:0.1 * ${optimizer.lr}}

optimizer:
  lr: 6e-4
  weight_decay: 0.1

train:
  gpu_mem: ${eval:"round(float(__import__('subprocess').check_output('nvidia-smi -i 0
  seed: 2222
  global_batch_size: 16
```

hg38/hg38_hyena.yaml

75行配置参数

train.py

```
> class SequenceLightningModule(pl.LightningModule):…


  ### pytorch-lightning utils and entrypoint ###

> def create_trainer(config, **kwargs):…


> def train(config):…




  @hydra.main(config_path="configs", config_name="config.yaml")
> def main(config: OmegaConf):…


  if __name__ == "__main__":
      main()
```

```
@hydra.main(config_path="configs", config_name="config.yaml")
def main(config: OmegaConf):

    # Process config:
    # - register evaluation resolver
    # - filter out keys used only for interpolation
    # - optional hooks, including disabling python warnings or
    config = utils.train.process_config(config)

    # Pretty print config using Rich library
    utils.train.print_config(config, resolve=True)

    train(config)


if __name__ == "__main__":
    main()
```


```
train_cfg = {'train': {'seed': 42, 'interval': 'step', 'monitor': 'test/loss', 'mode': 'min', 'ema': 0.0, 'test': False, 'debug': False, 'ignore_warnin
gs': False, 'state': {'mode': None, 'n_context': 0, 'n_context_eval': 0}, 'ckpt': None, 'disable_dataset': False, 'validate_at_start': False, 'pretrain
ed_model_path': None, 'pretrained_model_strict_load': True, 'pretrained_model_state_hook': {'_name_': None}, 'post_init_hook': {'_name_': None}, 'layer
_decay': {'_name_': None, 'decay': 0.7}, 'gpu_mem': 41, 'global_batch_size': 32}, 'tolerance': {'logdir': './resume', 'id': None}, 'wandb': None, 'trai
ner': {'_target_': 'pytorch_lightning.Trainer', 'devices': 1, 'accelerator': 'gpu', 'accumulate_grad_batches': 1, 'max_epochs': 50, 'gradient_clip_val'
: 1.0, 'log_every_n_steps': 10, 'limit_train_batches': 1.0, 'limit_val_batches': 1.0, 'num_nodes': 1, 'precision': 16}, 'loader': {'batch_size': 50, 'n
um_workers': 4, 'pin_memory': True, 'drop_last': True}, 'dataset': {'_name_': 'methyl', 'meta_file': '/home/yhsun/Project/MethMarkGPT/Ratio_base/wgbs_p
retrain_test/data_info.tsv', 'fasta_file': None, 'dataset_name': 'methyl', 'tokenizer_name': 'char', 'cache_dir': None, 'max_length': 8192, 'add_eos':
True, 'batch_size': 32, 'batch_size_eval': 64, 'num_workers': 12, 'shuffle': True, 'pin_memory': True, 'fasta_plus_file': '/home/yhsun/Project/MethMark
GPT/Ratio_base/scrpit/plus_genome.fa', 'fasta_minus_file': '/home/yhsun/Project/MethMarkGPT/Ratio_base/scrpit/minus_genome.fa', 'max_length_val': 8192,
 'max_length_test': 8192, 'pad_max_length': None, 'rc_aug': True, 'use_fixed_len_val': False, 'replace_N_token': False, 'pad_interval': False}, 'optimi
zer': {'_name_': 'adamw', 'lr': 0.0006, 'weight_decay': 0.1, 'betas': [0.9, 0.999]}, 'scheduler': {'_name_': 'cosine_warmup_timm', 't_in_epochs': False
, 't_initial': 190750, 'lr_min': 5.99999999999999995e-05, 'warmup_lr_init': 1e-06, 'warmup_t': 1907.5}, 'callbacks': {'learning_rate_monitor': {'logging
_interval': 'step'}, 'timer': {'step': True, 'inter_step': False, 'epoch': True, 'val': True}, 'params': {'total': True, 'trainable': True, 'fixed': Tr
ue}, 'model_checkpoint': {'monitor': 'test/loss', 'mode': 'min', 'save_top_k': 1, 'save_last': True, 'dirpath': 'checkpoints/', 'filename': 'test/loss'
, 'auto_insert_metric_name': False, 'verbose': True}}, 'task': {'_name_': 'lm', 'loss': 'cross_entropy', 'torchmetrics': ['perplexity', 'num_tokens'],
'metrics': ['accuracy', 'accuracy@3']}, 'encoder': None, 'decoder': None, 'model': {'_name_': 'lm', 'd_model': 128, 'n_layer': 8, 'd_inner': 512, 'voca
b_size': 15, 'resid_dropout': 0.0, 'embed_dropout': 0.1, 'fused_mlp': False, 'fused_dropout_add_ln': False, 'checkpoint_mixer': False, 'checkpoint_mlp'
: False, 'residual_in_fp32': True, 'pad_vocab_size_multiple': 8, 'layer': {'_name_': 'hyena', 'emb_dim': 5, 'filter_order': 64, 'short_filter_order': 3
, 'l_max': 8194, 'modulate': True, 'w': 10, 'lr': 0.0006, 'wd': 0.0, 'lr_pos_emb': 0.0}}}
```


传入config，train函数会完成处理

如何实现给hg38/hg38_hyena.yaml
完成如此多的参数初始化？

Config 处理，将yaml文件类的配置信息转为字典结构

train.py

python hydra_test.py experiment=hg38/hg38_hyena

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

@hydra.main(config_path="configs", config_name="config.yaml", version_base="1.1")
def main(config: OmegaConf):
    print(OmegaConf.to_yaml(config))
    OmegaConf.save(config, "hydra_hg38_config.yaml")


if __name__ == "__main__":
    main()
```

```
%run hydra_test.py experiment=hg38/hg38_hyena
✓ 0.5s
```

```
train:
  seed: 2222
  interval: step
  monitor: test/loss
  mode: min
  ema: 0.0
```

hg38/hg38_hyena.yaml

```yaml
dataset:
  # batch_size: 32  # Per GPU
  batch_size: 256
  max_length: 1024 # 262144, 524288
  # optional, default is max_length
  max_length_val: ${dataset.max_length}
  max_length_test: ${dataset.max_length}
  tokenizer_name: char
  pad_max_length: null  # needed for bpe tok
  add_eos: true
  rc_aug: false
  num_workers: 12
  use_fixed_len_val: false  # placing a fixe
  replace_N_token: false  # replace N (uncer
  pad_interval: false  # handle uncertain to
```

hydra_hg38_config.yaml

```yaml
dataset:
  _name_: hg38
  bed_file: null
  fasta_file: null
  dataset_name: hg38
  tokenizer_name: char
  cache_dir: null
  max_length: 1024
  add_eos: true
  batch_size: 256
  batch_size_eval: ${eval:${.batch_size} * 2}
  num_workers: 12
  shuffle: true
  pin_memory: true
  __train_len: ${div_up:1_000_000_000, ${.max_length}}
  __l_max: ${.max_length}
  max_length_val: ${dataset.max_length}
  max_length_test: ${dataset.max_length}
  pad_max_length: null
  rc_aug: false
  use_fixed_len_val: false
  replace_N_token: false
  pad_interval: false
```

新增的参数怎么来的？

会生成一个hydra_hg38_config.yaml，中间包含151行配置参数

从75到151行的参数变化，是config间的调用关系

```
(hyena-dna)  7月-01~2025 | 12:00:54 | yhsun@localhost | ~/Project/MethMarkGPT/Ratio_base/hyena-dna
$ python hydra_test.py experiment=hg38/hg38_hyena --info defaults

Defaults List
*************
| Config path                    | Package             | _self_ | Parent                    |
-------------------------------------------------------------------------------------------
| hydra/output/default           | hydra               | False  | hydra/config              |
| hydra/launcher/basic           | hydra.launcher      | False  | hydra/config              |
| hydra/sweeper/basic            | hydra.sweeper       | False  | hydra/config              |
| hydra/help/default             | hydra.help          | False  | hydra/config              |
| hydra/hydra_help/default       | hydra.hydra_help    | False  | hydra/config              |
| hydra/hydra_logging/default    | hydra.hydra_logging | False  | hydra/config              |
| hydra/job_logging/default      | hydra.job_logging   | False  | hydra/config              |
| hydra/env/default              | hydra.env           | False  | hydra/config              |
| hydra/config                   | hydra               | True   | <root>                    |
| config.yaml                    |                     | True   | <root>                    |
| trainer/default                | trainer             | False  | pipeline/hg38             |
| loader/default                 | loader              | False  | pipeline/hg38             |
| dataset/hg38                   | dataset             | False  | pipeline/hg38             |
| optimizer/adamw                | optimizer           | False  | pipeline/hg38             |
| scheduler/cosine_warmup_timm   |                     | False  | pipeline/hg38             |
| callbacks/base                 | callbacks           | False  | pipeline/hg38             |
| callbacks/checkpoint           | callbacks           | False  | pipeline/hg38             |
| pipeline/hg38                  |                     | True   | experiment/hg38/hg38_hyena |
| experiment/hg38/hg38_hyena     |                     | True   | config.yaml               |
-------------------------------------------------------------------------------------------
```

```
config.yaml
  └── experiment/hg38/hg38_hyena.yaml
        └── pipeline/hg38.yaml
              ├── trainer/default.yaml
              ├── loader/default.yaml
              ├── dataset/hg38.yaml
              ├── optimizer/adamw.yaml
              ├── scheduler/cosine_warmup_timm.yaml
              └── callbacks/
                    ├── base.yaml
                    └── checkpoint.yaml
```

> experiment > hg38 > ! hg38_hyena.yaml

```yaml
# @package _global_
defaults:
  - /pipeline: hg38
  - override /scheduler: cosine_warmup_timm
```

> pipeline > ! hg38.yaml

```yaml
# @package _global_
defaults:
  - /trainer: default
  - /loader: default
  - /dataset: hg38
  - /optimizer: adamw
  - /scheduler: cosine_warmup
  - /callbacks: [base, checkpoint]

train:
  monitor: test/loss
  mode: min

task:
  _name_: lm
  loss: cross_entropy
  torchmetrics: ['perplexity', 'num_tokens']

encoder: null
decoder: null
```

dataset/hg38.yaml

configs > dataset > ! hg38.yaml

```yaml
_name_: hg38
bed_file: null
fasta_file: null
dataset_name: hg38
tokenizer_name: null
cache_dir: null
max_length: 1024
add_eos: True
batch_size: 8  # per GPU
batch_size_eval: ${eval:${.batch_size} * 2}
num_workers: 4  # For preprocessing only
shuffle: True
pin_memory: True
__train_len: ${div_up:1_000_000_000, ${.max_length}}
__l_max: ${.max_length}
```

callbacks/base.yaml

> callbacks > ! base.yaml

```yaml
learning_rate_monitor:
  # _target_: pytorch_lightning.callbac
  logging_interval: ${train.interval}

timer:
  # _target_: callbacks.timer.Timer
  step: True
  inter_step: False
  epoch: True
  val: True

params:
  # _target_: callbacks.params.ParamsLo
  total: True
  trainable: True
  fixed: True
```

callbacks/checkpoint.yaml

> callbacks > ! checkpoint.yaml

```yaml
model_checkpoint:
  # _target_: pytorch_lightning.call
  monitor: ${train.monitor} # name o
  mode: ${train.mode} # can be "max"
  save_top_k: 1 # save k best models
  save_last: True # additionaly alwa
  dirpath: "checkpoints/"
  # this saves an annoying "epoch=12
  # seems like you can override the
  # filename: "{epoch:02d}",
  filename: ${train.monitor}
  auto_insert_metric_name: False
  verbose: True
```

train.py

```python
def train(config):
    if config.train.seed is not None:
        pl.seed_everything(config.train.seed, workers=True)
    trainer = create_trainer(config)
    model = SequenceLightningModule(config)

    # Load pretrained_model if specified
    if config.train.get("pretrained_model_path", None) is not None:
        # PTL style.  Note, method returns a new model object, and need to pass config.
        model = SequenceLightningModule.load_from_checkpoint(
            config.train.pretrained_model_path,
            config=config,
            strict=config.train.pretrained_model_strict_load,
        )

    # Run initial validation epoch (useful for debugging, finetuning)
    if config.train.validate_at_start:
        print("Running validation before training")
        trainer.validate(model)

    if config.train.ckpt is not None:
        trainer.fit(model, ckpt_path=config.train.ckpt)
    else:
        trainer.fit(model)
    if config.train.test:
        trainer.test(model)
```

定义trainer

定义model

模型训练
调用pytorch-lightning框架

Trainer

```python
def create_trainer(config, **kwargs):
    callbacks: List[pl.Callback] = []
    logger = None

    # WandB Logging
    if config.get("wandb") is not None:
        # Pass in wandb.init(config=) argument to get the nice 'x.y.0.z' hparams logged
        # Can pass in config_exclude_keys='wandb' to remove certain groups
        import wandb

        logger = CustomWandbLogger(
            config=utils.to_dict(config, recursive=True),
            settings=wandb.Settings(start_method="fork"),
            **config.wandb,
        )

    # Lightning callbacks
    if "callbacks" in config:
        for _name_, callback in config.callbacks.items():
            if config.get("wandb") is None and _name_ in ["learning_rate_monitor"]:
                continue
            log.info(f"Instantiating callback <{registry.callbacks[_name_]}>")
            callback._name_ = _name_
            callbacks.append(utils.instantiate(registry.callbacks, callback))

    # Add ProgressiveResizing callback
    if config.callbacks.get("progressive_resizing", None) is not None:
        num_stages = len(config.callbacks.progressive_resizing.stage_params)
        print(f"Progressive Resizing: {num_stages} stages")
        for i, e in enumerate(config.callbacks.progressive_resizing.stage_params):
            # Stage params are resolution and epochs, pretty print
            print(f"\tStage {i}: {e['resolution']} @ {e['epochs']} epochs")

    # Configure ddp automatically
    n_devices = config.trainer.get('devices', 1)
    if isinstance(n_devices, Sequence):  # trainer.devices could be [1, 3] for example
        n_devices = len(n_devices)
    if n_devices > 1 and config.trainer.get('strategy', None) is None:
        config.trainer.strategy = dict(
            _target_='pytorch_lightning.strategies.DDPStrategy',
            find_unused_parameters=False,
            gradient_as_bucket_view=True,  # https://pytorch-lightning.readthedocs.io/en/stable/ad
        )

    # Init lightning trainer
    log.info(f"Instantiating trainer <{config.trainer._target_}>")
    # special processing for seqlen warmup scheduler (reload)
    if config.callbacks.get("seqlen_warmup_reload", None) is not None:
        # we need to instantiate manually instead of with hydra, since it expects a dict instead o
        # so we convert everything to dicts (from hydra configs)
        trainer_config_dict = dict(config.trainer)
        epochs_cume = 0  # track cumulative epochs
        accumulate_grad_schedule = {}  # contains the accumulate_grad_batches schedule to init the

        for stage in config.callbacks.seqlen_warmup_reload.stage_params:
            batch_size = stage['batch_size']  # curr batch size at this stage
            grad_accum_factor = config.train.global_batch_size // batch_size  # grad accum factor
            accumulate_grad_schedule[epochs_cume] = grad_accum_factor  # set the grad accum factor
            epochs_cume += stage['epochs']  # increment epochs_cume for next stage
        trainer_config_dict['accumulate_grad_batches'] = accumulate_grad_schedule  # set the accum
        trainer_config_dict.pop('_target_')  # only hydra uses this to instantiate
        # Set DDPStrategy to work with pl.Trainer
        config.trainer.pop('strategy')
        trainer_config_dict['strategy'] = DDPStrategy(find_unused_parameters=False, gradient_as_b
        trainer = pl.Trainer(**trainer_config_dict, callbacks=callbacks, logger=logger)
    else:
        trainer = hydra.utils.instantiate(config.trainer, callbacks=callbacks, logger=logger)

    return trainer
```

```python
# Lightning callbacks
if "callbacks" in config:
    for _name_, callback in config.callbacks.items():
        if config.get("wandb") is None and _name_ in ["learning_rate_monitor"]:
            continue
        log.info(f"Instantiating callback <{registry.callbacks[_name_]}>")
        callback._name_ = _name_
        callbacks.append(utils.instantiate(registry.callbacks, callback))
```

```python
else:
    trainer = hydra.utils.instantiate(config.trainer, callbacks=callbacks, logger=logger)

return trainer
```

Trainer的组成部分：

1. config.trainer里的参数配置
2. callbacks里的参数配置
3. logger的设置

```python
callbacks = {
    "timer": "src.callbacks.timer.Timer",
    "params": "src.callbacks.params.ParamsLog",
    "learning_rate_monitor": "pytorch_lightning.callbacks.LearningRateMonitor",
    "model_checkpoint": "pytorch_lightning.callbacks.ModelCheckpoint",
    "early_stopping": "pytorch_lightning.callbacks.EarlyStopping",
    "swa": "pytorch_lightning.callbacks.StochasticWeightAveraging",
    "rich_model_summary": "pytorch_lightning.callbacks.RichModelSummary",
    "rich_progress_bar": "pytorch_lightning.callbacks.RichProgressBar",
    "progressive_resizing": "src.callbacks.progressive_resizing.ProgressiveResizing",
    "seqlen_warmup": "src.callbacks.seqlen_warmup.SeqlenWarmup",
    "seqlen_warmup_reload": "src.callbacks.seqlen_warmup_reload.SeqlenWarmupReload",
    "gpu_affinity": "src.callbacks.gpu_affinity.GpuAffinity"
}
```

Trainer

```python
# Lightning callbacks
if "callbacks" in config:
    for _name_, callback in config.callbacks.items():
        if config.get("wandb") is None and _name_ in ["learning_rate_monitor"]:
            continue
        log.info(f"Instantiating callback <{registry.callbacks[_name_]}>")
        callback._name_ = _name
        callbacks.append(utils.instantiate(registry.callbacks, callback))
```

```python
registry = {
    "model": "models.MyModel",
    "optimizer": torch.optim.Adam
}
config = {
    "_name_": "model",
    "hidden_size": 128,
    "dropout": 0.1
}
```

```
model = instantiate(registry, config)
```

↓ 等价于

```
model = models.MyModel(hidden_size=128, dropout=0.1)
```

```python
def instantiate(registry, config, *args, partial=False, wrap=None, **kwargs):
    """
    registry: Dictionary mapping names to functions or target paths (e.g. {'model': 'models.Seque
    config: Dictionary with a '_name_' key indicating which element of the registry to grab, and
    wrap: wrap the target class (e.g. ema optimizer or tasks.wrap)
    *args, **kwargs: additional arguments to override the config to pass into the target construc
    """

    # Case 1: no config
    if config is None:
        return None
    # Case 2a: string means _name_ was overloaded
    if isinstance(config, str):
        _name_ = None
        _target_ = registry[config]
        config = {}
    # Case 2b: grab the desired callable from name
    else:
        _name_ = config.pop("_name_")
        _target_ = registry[_name_]

    # Retrieve the right constructor automatically based on type
    if isinstance(_target_, str):
        fn = hydra.utils.get_method(path=_target_)
    elif isinstance(_target_, Callable):
        fn = _target_
    else:
        raise NotImplementedError("instantiate target must be string or callable")

    # Instantiate object
    if wrap is not None:
        fn = wrap(fn)
    obj = functools.partial(fn, *args, **config, **kwargs)

    # Restore _name_
    if _name_ is not None:
        config["_name_"] = _name_

    if partial:
        return obj
    else:
        return obj()
```

```python
    return None
# Case 2a: string means _name_ was ove
if isinstance(config, str):
    _name_ = None
    _target_ = registry[config]
    config = {}
# Case 2b: grab the desired callable from name
else:
    _name_ = config.pop("_name_")
    _target_ = registry[_name_]
```

✅ `config`：**就是传参用的字典**

- 它包含了构造目标（类或函数）所需的参数；
- 其中 `_name_` 指定要构造哪一个目标（在 `registry` 里查找用）；
- 其他键值对就是这个类/函数的初始化参数。

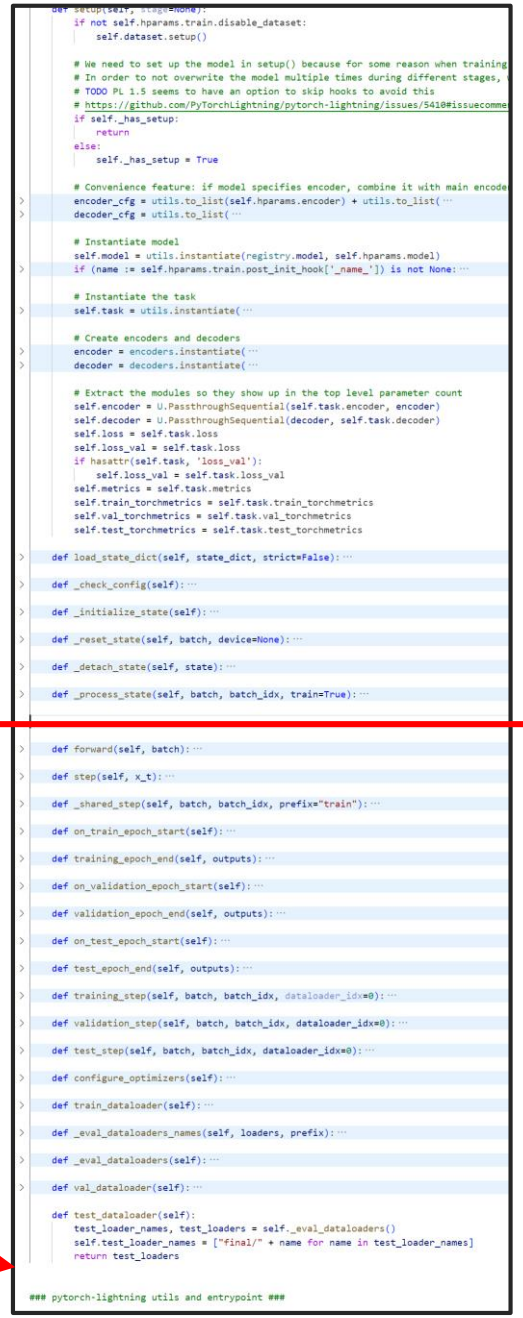
✅ `_target_`：**就是你要构造的类或函数本身**

- 可以是类名（例如 `MyModel`），也可以是字符串路径（例如 `'models.MyModel'`）；
- 由 `registry[_name_]` 决定；
- 会传入 config 的参数调用：`MyModel(**config)`。

```python
# Retrieve the right constructor automatically based on type
if isinstance(_target_, str):
    fn = hydra.utils.get_method(path=_target_)
elif isinstance(_target_, Callable):
    fn = _target_
else:
    raise NotImplementedError("instantiate target must be string or callable")
```

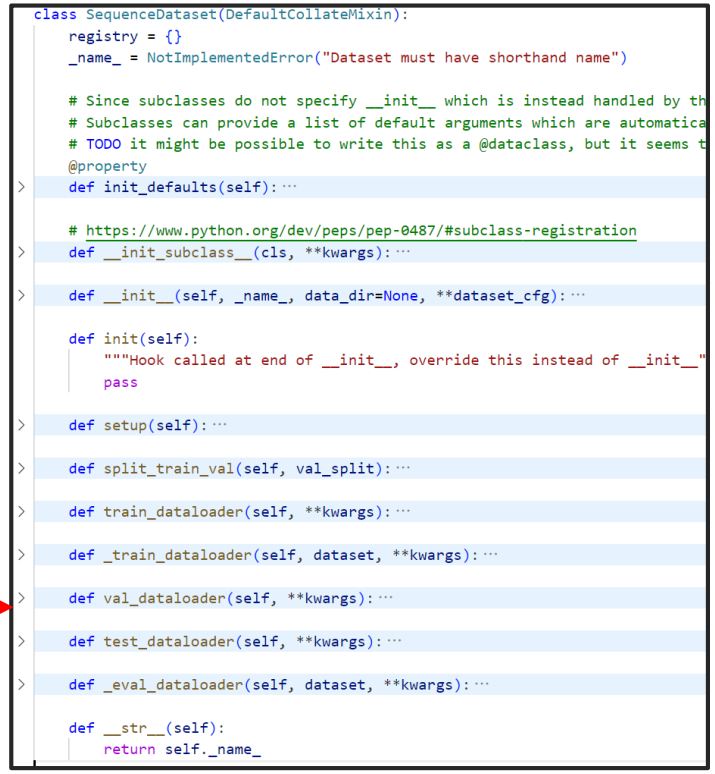
Model

```python
class SequenceLightningModule(pl.LightningModule):
    def __init__(self, config):
        # Disable profiling executor. This reduces memor
        try:
            torch._C._jit_set_profiling_executor(False)
            torch._C._jit_set_profiling_mode(False)
        except AttributeError:
            pass

        super().__init__()
        # Passing in config expands it one level, so ca
        self.save_hyperparameters(config, logger=False)

        # Dataset arguments
        self.dataset = SequenceDataset.registry[self.hpa
            **self.hparams.dataset
        )

        # Check hparams
        self._check_config()

        # PL has some bugs, so add hooks and make sure t
        self._has_setup = False

        self.setup()  ## Added by KS
```



SequenceDataset的基类定义

定义数据加载



pl.LightningModule基类的继承，其中内容与之前介绍的一致

- Model的定义
- Epoch的训练
- Epoch的评估指标

Model

```
self.dataset = SequenceDataset.registry[self.hparams.dataset._name_](
    **self.hparams.dataset
)
```

根据dataset._name_确定所使用的类

```
class HG38(SequenceDataset) ···
class GenomicBenchmark(HG38): ···
class NucleotideTransformer(HG38): ···
class ChromatinProfile(HG38): ···
class Species(HG38): ···
class ICLGenomics(HG38): ···
class HG38Fixed(HG38): ···
```

```
class HG38(SequenceDataset):
    """
    Base class, other dataloaders can inherit from this class.

    You must implement the following functions:
        - __init__
        - setup

    You can then use (already have access to) the following functions:
        - train_dataloader
        - val_dataloader
        - test_dataloader

    """
    ###### very important to set this! ######
    _name_ = "hg38"  # this name is how the dataset config finds the
    ########################################
```

configs > dataset > ! hg38.yaml

```
1   _name_: hg38
2   bed_file: null
3   fasta_file: null
4   dataset_name: hg38
5   tokenizer_name: null
6   cache_dir: null
7   max_length: 1024
8   add_eos: True
9   batch_size: 8  # per GPU
10  batch_size_eval: ${eval:${.batch_size} * 2}
11  num_workers: 4  # For preprocessing only
12  shuffle: True
13  pin_memory: True
14  __train_len: ${div_up:1_000_000_000, ${.max_length}}
15  __l_max: ${.max_length}
```

```
def init_datasets(self):
    """Init the datasets (separate from the tokenizer)"""

    # delete old datasets to free memory
    if hasattr(self, 'dataset_train'):
        self.dataset_train.fasta.seqs.close()
        del self.dataset_train.fasta.seqs

    # delete old datasets to free memory
    if hasattr(self, 'dataset_test'):
        self.dataset_test.fasta.seqs.close()
        del self.dataset_test.fasta.seqs

    # Create all splits: torch datasets
    self.dataset_train, self.dataset_val, self.dataset_test = [
        HG38Dataset(split=split,
                    bed_file=self.bed_file,
                    fasta_file=self.fasta_file,
                    max_length=max_len,
```

类中对数据载入进行定义

Model

```python
# Instantiate model
self.model = utils.instantiate(registry.model, self.hparams.model)
```

```python
model = {
    # Backbones from this repo
    "model": "src.models.sequence.SequenceModel",
    "lm": "src.models.sequence.long_conv_lm.ConvLMHeadModel",
    "lm_simple": "src.models.sequence.simple_lm.SimpleLMHeadModel",
    "vit_b_16": "src.models.baselines.vit_all.vit_base_patch16_224",
    "dna_embedding": "src.models.sequence.dna_embedding.DNAEmbeddingModel",
    "bpnet": "src.models.sequence.hyena_bpnet.HyenaBPNet"
}

layer = {
    "id": "src.models.sequence.base.SequenceIdentity",
    "ff": "src.models.sequence.ff.FF",
    "mha": "src.models.sequence.mha.MultiheadAttention",
    "s4d": "src.models.sequence.ssm.s4d.S4D",
    "s4_simple": "src.models.sequence.ssm.s4_simple.SimpleS4Wrapper",
    "long-conv": "src.models.sequence.long_conv.LongConv",
    "h3": "src.models.sequence.h3.H3",
    "h3-conv": "src.models.sequence.h3_conv.H3Conv",
    "hyena": "src.models.sequence.hyena.HyenaOperator",
    "hyena-filter": "src.models.sequence.hyena.HyenaFilter",
    "vit": "src.models.sequence.mha.VitAttention",
}
```

```python
def create_mixer_cls(…

def create_mlp_cls(…

def create_block(…

# https://github.com/huggingface/transformers/blob/c28d0
def _init_weights(…

class LMBackbone(nn.Module):…

class ConvLMHeadModel(nn.Module, GenerationMixin):…

class DNAEmbeddingModel(nn.Module, GenerationMixin):…

def load_backbone(model, state_dict, freeze_backbone=Fal

def shard_state_dict_tp(state_dict, world_size, rank, pa
```

Model

```python
class ConvLMHeadModel(nn.Module, GenerationMixin):
    def __init__(
        self,
        d_model: int,
        n_layer: int,
        d_inner: int,
        vocab_size: int,
        process_group=None,
        layer=None,
        attn_layer_idx=None,
        attn_cfg=None,
        max_position_embeddings=0,
        resid_dropout: float = 0.0,
        embed_dropout: float = 0.1,
        dropout_cls=nn.Dropout,
        layer_norm_epsilon: float = 1e-5,
        initializer_cfg=None,
        fused_mlp=False,
        fused_dropout_add_ln=False,
        residual_in_fp32=False,
        pad_vocab_size_multiple: int = 1,
        sequence_parallel=True,
        checkpoint_mlp=False,
        checkpoint_mixer=False,
        device=None,
        dtype=None,
        **kwargs,
    ) -> None:
        factory_kwargs = {"device": device, "dtype": d
        super().__init__()
        self.process_group = process_group
        if vocab_size % pad_vocab_size_multiple != 0:
        self.backbone = LMBackbone(···
        if process_group is None:···
        else:···
        # Initialize weights and apply final processing
        self.apply(···
        self.tie_weights()

    def tie_weights(self):
        self.lm_head.weight = self.backbone.embeddings
        if self.process_group is not None:
            sync_shared_params(self, self.process_grou

    def forward(···
```

```python
class LMBackbone(nn.Module):
    def __init__(
        self,
        d_model: int,
        n_layer: int,
        d_inner: int,
        vocab_size: int,
        process_group=None,
        layer=None,
        attn_layer_idx=None,
        attn_cfg=None,
```

```python
        if process_group is None:
            self.embeddings = GPT2Embeddings(···
        else:
            self.embeddings = ParallelGPT2Embeddings(···
```

```python
        self.layers = nn.ModuleList(
            [
                create_block(···
                for i in range(n_layer)
            ]
        )

        self.drop_f = nn.Dropout(resid_dropout)
        self.ln_f = nn.LayerNorm(d_model, eps=layer_norm_epsilon, **

        if process_group is not None:···

        self.apply(
            partial(···
        )
        self.tie_weights()

    def tie_weights(self):
        if self.process_group is not None:
            sync_shared_params(self, self.process_group)

    def forward(self, input_ids, position_ids=None, inference_params
```

模型结构最基本的定义

Model

```python
def create_block(
    d_model,
    d_inner=None,
    process_group=None,
    layer=None,
    attn_layer_idx=None,
    attn_cfg=None
```

```python
    factory_kwargs = {"device": device, "dtype": dtype}
    mixer_cls = create_mixer_cls(···
    mlp_cls = create_mlp_cls(···
    norm_cls = partial(nn.LayerNorm, eps=layer_norm_epsilon,
    block = Block(
        d_model,
        mixer_cls,
        mlp_cls,
        norm_cls=norm_cls,
        prenorm=True,
        resid_dropout1=resid_dropout1,
        resid_dropout2=resid_dropout2,
        fused_dropout_add_ln=fused_dropout_add_ln,
        residual_in_fp32=residual_in_fp32,
        sequence_parallel=sequence_parallel and process_grou
        mark_shared_params=process_group is not None,
    )
```

```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
```

```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
```

默认MHA模块，即transformer里的多头注意力

改为自定义的计算层

Model

```
# Instantiate the task
self.task = utils.instantiate(
    tasks.registry, self.hparams.task, dataset=self.dataset, model=self.model
)
```

```
class BaseTask:
    """ Abstract class that takes care of:
    - loss function
    - arbitrary metrics
    - forward pass
    - (optional) encoder module that interfaces with dataset (inputs) and model
    - (optional) decoder module that interfaces with dataset (targets) and model
    """
    encoder = None
    decoder = None

    def __init__(self, dataset=None, model=None, loss=None, loss_val=None, metri
        """ This class is allowed to grab attributes directly off a constructed
        self.dataset = dataset
        self.model = model
        if metrics is None: metrics = []
        self.metric_names = to_list(metrics)

        if torchmetrics is None: torchmetrics = []
        self.torchmetric_names = to_list(torchmetrics)
        self._tracked_torchmetrics = {}
```

BaseTask为基类，定义损失函数、评估指标和前向传播

```
> class BaseTask: ···

> class Scalar(nn.Module): ···

> class LMTask(BaseTask): ···

> class MultiClass(BaseTask): ···

> class MaskedMultiClass(MultiClass): ···

> class HG38Task(LMTask): ···

> class AdaptiveLMTask(BaseTask): ···

registry = {
    'base': BaseTask,
    'multiclass': MultiClass,
    'lm': LMTask,
    'hg38': HG38Task,
    "masked_multiclass": MaskedMultiClass,
}
```

不同任务类型，对应不同类的使用

基于config文件自己实现类的初始化

```python
class LMTask(BaseTask):
    def forward(self, batch, encoder, model, decoder, _state):
        """Passes a batch through the encoder, backbone, and decoder"""
        # z holds arguments such as sequence length
        x, y, *z = batch # z holds extra dataloader info such as resolu

        print(f"Input shape: {x.shape}")
        print(f"Target shape: {y.shape}")

        if len(z) == 0:
            z = {}
        else:
            assert len(z) == 1 and isinstance(z[0], dict), "Dataloader
            z = z[0]
        x, w = encoder(x, **z) # w can model-specific constructions such

        print(f"Encoder structure: {encoder}")

        x, state = model(x, **w, state=_state)

        print(f"Model structure: {model}")

        self._state = state

        print(f"Decoder structure: {decoder}")

        x, w = decoder(x, state=state, **z)

        x = x.logits
        x = rearrange(x, '... C -> (...) C')
        y = rearrange(y, '... -> (...)')

        print(f"rearrange shape: {x.shape}")
        print(f"rearrange shape: {y.shape}")
        #print(f"w  shape: {w.shape}")

        return x, y, w
```

```
    (drop_f): Dropout(p=0.0, inplace=False)
    (ln_f): LayerNorm((128,), eps=1e-05, elementwise_affine=True)
  )
  (lm_head): Linear(in_features=128, out_features=16, bias=False)
)
(encoder): Identity()
(decoder): Identity()
(train_torchmetrics): MetricCollection(
  (num_tokens): NumTokens()
  (perplexity): Perplexity(
    (loss_fn): CrossEntropyLoss()
  ),
  prefix=train/
)
(val_torchmetrics): MetricCollection(
  (num_tokens): NumTokens()
  (perplexity): Perplexity(
    (loss_fn): CrossEntropyLoss()
```

```
Input shape x: torch.Size([128, 1023])
Encoder structure: Identity()
Decoder structure: Identity()
Output shape x: torch.Size([128, 1023, 16])
Rearrange shape x: torch.Size([130944, 16])
Rearrange shape y: torch.Size([130944])
Epoch 0:   1%|
Input shape x: torch.Size([128, 1023])
Encoder structure: Identity()
Decoder structure: Identity()
Output shape x: torch.Size([128, 1023, 16])
Rearrange shape x: torch.Size([130944, 16])
Rearrange shape y: torch.Size([130944])
Epoch 0:   1%|
```

Input [batchsize, seqlen-1]

Output [batchsize, seqlen-1, vacab_size]

Rearrange [batchsize* (seqlen – 1), vacab_size]

Tensor	Shape	含义
Input IDs	`(B, L)`	B 是 batch size，L 是序列长度
Embedding	`(B, L, H)`	每个 token 映射到维度为 H 的向量
Logits	`(B, L, V)`	V 是词表大小，每个 token 的预测概率
Flatten logits	`(B×L, V)`	用于 `cross_entropy`
Target token	`(B, L)` → `(B×L,)`	目标 token ID

Task

loss, accuracy定义

```python
def cross_entropy(logits, y, ignore_index=-100):
    logits = logits.view(-1, logits.shape[-1])
    y = y.view(-1)
    return F.cross_entropy(logits, y, ignore_index=ignore_index)

def accuracy(logits, y):
    logits = logits.view(-1, logits.shape[-1])
    preds = torch.argmax(logits, dim=-1)
    if y.numel() > logits.shape[0]:
        # Mixup leads to this case: use argmax class
        y = y.argmax(dim=-1)
    y = y.view(-1)
    return torch.eq(preds, y).float().mean()
```

Training_step里定义了_shared_step，这个是用来作为辅助函数，计算一堆指标等东西，这里是定义loss

```python
def training_step(self, batch, batch_idx, dataloader_idx=0):
    loss = self._shared_step(batch, batch_idx, prefix="train")

    # Log the loss explicitly so it shows up in WandB
    # Note that this currently runs into a bug in the progress bar with ddp (a
    # https://github.com/PyTorchLightning/pytorch-lightning/pull/9142
    # We additionally log the epochs under 'trainer' to get a consistent prefi
    loss_epoch = {"trainer/loss": loss, "trainer/epoch": self.current_epoch}
    self.log_dict(
        loss_epoch,
        on_step=True,
        on_epoch=False,
        prog_bar=False,
        add_dataloader_idx=False,
        sync_dist=True,
    )
```

```python
def _shared_step(self, batch, batch_idx, prefix="train"):

    self._process_state(batch, batch_idx, train=(prefix == "train"))
    x, y, w = self.forward(batch)

    # Loss
    if prefix == 'train':
        loss = self.loss(x, y, **w)
    else:
        loss = self.loss_val(x, y, **w)

    # Metrics
    metrics = self.metrics(x, y, **w)
    metrics["loss"] = loss
    metrics = {f"{prefix}/{k}": v for k, v in metrics.items()}

    # Calculate torchmetrics
    torchmetrics = getattr(self, f'{prefix}_torchmetrics')
    torchmetrics(x, y, loss=loss)

    log_on_step = 'eval' in self.hparams and self.hparams.eval.get('log_on

    self.log_dict(
        metrics,
        on_step=log_on_step,
        on_epoch=True,
        prog_bar=True,
        add_dataloader_idx=False,
        sync_dist=True,
    )

    # log the whole dict, otherwise lightning takes the mean to reduce it
    # https://pytorch-lightning.readthedocs.io/en/stable/visualize/logging
    self.log_dict(
        torchmetrics,
        on_step=log_on_step,
        on_epoch=True,
        prog_bar=True,
        add_dataloader_idx=False,
        sync_dist=True,
    )
    return loss
```

x: [batchsize* (seqlen – 1), vacab_size

这里的loss和准确度计算是整个batch里的所有碱基

Dataset

```python
class HG38(SequenceDataset):
    """
    Base class, other dataloaders can inherit from this class.

    You must implement the following functions:
        - __init__
        - setup

    You can then use (already have access to) the following functions:
        - train_dataloader
        - val_dataloader
        - test_dataloader

    """
    ###### very important to set this! ######
    _name_ = "hg38"  # this name is how the dataset config finds the right
    ########################################

    def __init__(self, bed_file, fasta_file, tokenizer_name=None, dataset_

    def setup(self, stage=None):...

    def init_datasets(self):...

    def train_dataloader(self, *args: Any, **kwargs: Any) -> DataLoader:

    def val_dataloader(self, *args: Any, **kwargs: Any) -> Union[DataLoad

    def test_dataloader(self, *args: Any, **kwargs: Any) -> Union[DataLoa
```

```python
# Create all splits: torch datasets
self.dataset_train, self.dataset_val, self.dataset_test = [
    HG38Dataset(split=split,
                bed_file=self.bed_file,
                fasta_file=self.fasta_file,
                max_length=max_len,
                tokenizer=self.tokenizer,  # pass the tokenize wrappe
                tokenizer_name=self.tokenizer_name,
                add_eos=self.add_eos,
                return_seq_indices=False,
                shift_augs=None,
                rc_aug=self.rc_aug,
                return_augs=False,
                replace_N_token=self.replace_N_token,
                pad_interval=self.pad_interval)
    for split, max_len in zip(['train', 'valid', 'test'], [self.max_l
]
```

```python
class HG38Dataset(torch.utils.data.Dataset):
    '''
    Loop thru bed file, retrieve (chr, start, end), query fasta
    '''

    def __init__(...

    def __len__(self):...

    def replace_value(self, x, old_value, new_value):...

    def __getitem__(self, idx):
        """Returns a sequence of specified len"""
        # sample a random row from df
        row = self.df.iloc[idx]
        # row = (chr, start, end, split)
        chr_name, start, end = (row[0], row[1], row[2])

        seq = self.fasta(chr_name, start, end, max_length=self.m

        if self.tokenizer_name == 'char':

            seq = self.tokenizer(seq,
                add_special_tokens=True if self.add_eos else Fal
                padding="max_length",
                max_length=self.max_length,
                truncation=True,
```

```python
data = seq[:-1].clone()  # remove eos
target = seq[1:].clone()  # offset by 1, includes eos
```

```python
return data, target
```

- **核心逻辑**：构造自回归训练对。
  - `input_ids[:-1]` 是模型的输入（去掉最后一个 token）
  - `input_ids[1:]` 是目标 label（去掉第一个 token）
- 举例：
  - 原序列 token: `[1, 2, 3, 4]`
  - 输入：`[1, 2, 3]`，目标：`[2, 3, 4]`
- 这是 **经典的语言建模目标：预测下一个 token。**

序列分词

构建自回归训练对

Dataset

{'[CLS]': 0, '[SEP]': 1, '[BOS]': 2, '[MASK]': 3, '[PAD]': 4, '[RESERVED]': 5, '[UNK]': 6, 'A': 7, 'C': 8, 'G': 9, 'T': 10, "N’ : 11}

AATAATAAAGTGCCTT **add_eos: true**

Raw input:
tensor([7, 7, 10, 7, 7, 10, 7, 7, 7, 9, 10, 9, 8, 8, 10, 1])

Autoregressive input:
tensor([7, 7, 10, 7, 7, 10, 7, 7, 7, 9, 10, 9, 8, 8, 10]),
tensor([7, 10, 7, 7, 10, 7, 7, 7, 9, 10, 9, 8, 8, 10, 1])

AATAATAAAGTGCCTT **add_eos: false**

Raw input:
tensor([7, 7, 10, 7, 7, 10, 7, 7, 7, 9, 10, 9, 8, 8, 10, 10])

Autoregressive input:
tensor([7, 7, 10, 7, 7, 10, 7, 7, 7, 9, 10, 9, 8, 8, 10]),
tensor([7, 10, 7, 7, 10, 7, 7, 7, 9, 10, 9, 8, 8, 10, 10])

基于config文件自己实现模型初始化

Hyena-dna如果理解配置文件的逻辑，直接使用其集成好的代码也没问题

但是要自定义的时候，还是建议把**相应的组件给拆分出来**，或者说**基于配置文件生成出来**

举例：基于hyena-dna训练序列

```
train_cfg = {'train': {'seed': 42, 'interval': 'step', 'monitor': 'test/loss', 'mode': 'min', 'ema': 0.0, 'test': False, 'debug': False, 'ignore_warnings': False, 'state': {'mode': None, 'n_context': 0, 'n_context_eval': 0}, 'ckpt': None, 'disable_dataset': False, 'validate_at_start': False, 'pretrained_model_path': None, 'pretrained_model_strict_load': True, 'pretrained_model_state_hook': {'_name_': None}, 'post_init_hook': {'_name_': None}, 'layer_decay': {'_name_': None, 'decay': 0.7}, 'gpu_mem': 41, 'global_batch_size': 128}, 'tolerance': {'logdir': './resume', 'id': None}, 'wandb': None, 'trainer': {'_target_': 'pytorch_lightning.Trainer', 'devices': 1, 'accelerator': 'gpu', 'accumulate_grad_batches': 1, 'max_epochs': 50, 'gradient_clip_val': 1.0, 'log_every_n_steps': 10, 'limit_train_batches': 1.0, 'limit_val_batches': 1.0, 'num_nodes': 1, 'precision': 16}, 'loader': {'batch_size': 50, 'num_workers': 4, 'pin_memory': True, 'drop_last': True}, 'dataset': {'_name_': 'methyl', 'meta_file': '/home/yhsun/Project/MethMarkGPT/Ratio_base/wgbs_pretrain_test/data_info_2.tsv', 'fasta_file': None, 'dataset_name': 'methyl', 'tokenizer_name': 'char', 'cache_dir': None, 'max_length': 1024, 'add_eos': True, 'batch_size': 128, 'batch_size_eval': 128, 'num_workers': 24, 'shuffle': True, 'pin_memory': True, 'max_length_val': 1024, 'max_length_test': 1024, 'pad_max_length': None, 'rc_aug': True, 'use_fixed_len_val': False, 'replace_N_token': False, 'pad_interval': False}, 'optimizer': {'_name_': 'adamw', 'lr': 0.0006, 'weight_decay': 0.1, 'betas': [0.9, 0.999]}, 'scheduler': {'_name_': 'cosine_warmup_timm', 't_in_epochs': False, 't_initial': 190750, 'lr_min': 5.9999999999999995e-05, 'warmup_lr_init': 1e-06, 'warmup_t': 1907.5}, 'callbacks': {'learning_rate_monitor': {'logging_interval': 'step'}, 'timer': {'step': True, 'inter_step': False, 'epoch': True, 'val': True}, 'params': {'total': True, 'trainable': True, 'fixed': True}, 'model_checkpoint': {'monitor': 'test/loss', 'mode': 'min', 'save_top_k': 1, 'save_last': True, 'dirpath': 'checkpoints/', 'filename': 'test/loss', 'auto_insert_metric_name': False, 'verbose': True}}, 'task': {'_name_': 'lm', 'loss': 'cross_entropy', 'torchmetrics': ['perplexity', 'num_tokens'], 'metrics': ['accuracy', 'accuracy@3']}, 'encoder': None, 'decoder': None, 'model': {'_name_': 'lm', 'd_model': 128, 'n_layer': 8, 'd_inner': 512, 'vocab_size': 15, 'resid_dropout': 0.0, 'embed_dropout': 0.1, 'fused_mlp': False, 'fused_dropout_add_ln': False, 'checkpoint_mixer': False, 'checkpoint_mlp': False, 'residual_in_fp32': True, 'pad_vocab_size_multiple': 8, 'layer': {'_name_': 'hyena', 'emb_dim': 5, 'filter_order': 64, 'short_filter_order': 3, 'l_max': 8194, 'modulate': True, 'w': 10, 'lr': 0.0006, 'wd': 0.0, 'lr_pos_emb': 0.0}}}
```

```
config = OmegaConf.create(train_cfg)
config = utils.train.process_config(config)

#utils.train.print_config(config, resolve=True)

#trainer, model = train(config)

if config.train.seed is not None:
    pl.seed_everything(config.train.seed, workers=True)


trainer = create_trainer(config)

#dm = methyl(meta_file="/home/yhsun/Project/MethMarkGPT/Ratio_base/wgbs_pretrain_test/data_info.tsv")

model = SequenceLightningModule(config)
#dm = methyl(**config.dataset)

#trainer.fit(model, datamodule=dm)
trainer.fit(model)
```

Config处理

Trainer初始化

Model 初始化

基于config文件自己模型的初始化

```
Global seed set to 42
Using 16bit native Automatic Mixed Precision (AMP)
GPU available: True (cuda), used: True
TPU available: False, using: 0 TPU cores
IPU available: False, using: 0 IPUs
HPU available: False, using: 0 HPUs
`Trainer(limit_train_batches=1.0)` was configured so 100% of the batches per epoch will be used..
`Trainer(limit_val_batches=1.0)` was configured so 100% of the batches will be used..
<class 'genomics.methyl'>
Generate dataset
/home/yhsun/Project/MethMarkGPT/Ratio_base/wgbs_pretrain_test/data_info_2.tsv
methyl
**Using Char-level tokenizer**
OK1
{'[CLS]': 0, '[SEP]': 1, '[BOS]': 2, '[MASK]': 3, '[PAD]': 4, '[RESERVED]': 5, '[UNK]': 6, 'A': 7, 'C': 8, 'G': 9, 'T': 10, 'M': 11, 'X': 12, 'U': 13,
'N': 14}
OK2
['../scrpit/pancancer_train/GSM721194.train.plus.genome.parquet']
[MethylDataset] Indexing parquet files: 100%|| 1/1 [00:00<00:00, 22.67it/s]
[INFO] Warmup cache (max 10) ...
['../scrpit/pancancer_valid/GSM721194.valid.plus.genome.parquet']
[MethylDataset] Indexing parquet files: 100%|| 1/1 [00:00<00:00, 175.54it/s]
[INFO] Warmup cache (max 10) ...
['../scrpit/pancancer_test/GSM721194.test.plus.genome.parquet']
[MethylDataset] Indexing parquet files: 100%|| 1/1 [00:00<00:00, 184.33it/s]
[INFO] Warmup cache (max 10) ...
OK3
self.dataset.setup ok
Run BaseTask
{'perplexity': <class 'src.tasks.torchmetrics.Perplexity'>, 'num_tokens': <class 'src.tasks.torchmetrics.NumTokens'>}
LOCAL_RANK: 0 - CUDA_VISIBLE_DEVICES: [0]
Hyperparameter groups [{'weight_decay': 0.0, 'lr': 0.0006}]

  | Name               | Type              | Params
-------------------------------------------------------
0 | model              | ConvLMHeadModel   | 1.7 M
1 | encoder            | Identity          | 0
2 | decoder            | Identity          | 0
3 | train_torchmetrics | MetricCollection  | 0
4 | val_torchmetrics   | MetricCollection  | 0
5 | test_torchmetrics  | MetricCollection  | 0
-------------------------------------------------------
1.7 M     Trainable params
0         Non-trainable params
1.7 M     Total params
3.479     Total estimated model params size (MB)
Epoch 0: 100%|| 468/468 [03:45<00:00,  2.08it/s, loss=1.29, val/accuracy=0.404, val/accuracy@3=0.866, val/loss=1.280, val/num_tokens=5441337, val/perp
Epoch 0, global step 389: 'test/loss' reached 1.27863 (best 1.27863), saving model to 'checkpoints/test/loss-v1.ckpt' as top 1
Epoch 1: 100%|| 468/468 [03:53<00:00,  2.01it/s, loss=1.2, val/accuracy=0.444, val/accuracy@3=0.878, val/loss=1.190, val/num_tokens=1.06e+7, val/perpl
Epoch 1, global step 778: 'test/loss' reached 1.19610 (best 1.19610), saving model to 'checkpoints/test/loss-v1.ckpt' as top 1
Epoch 2: 100%|| 468/468 [03:46<00:00,  2.07it/s, loss=1.16, val/accuracy=0.460, val/accuracy@3=0.883, val/loss=1.160, val/num_tokens=1.58e+7, val/perp
```

分词器

加载数据，写入缓存

模型初始化与训练

模型推理

token预测，给input的token给与logits

```
pretrained_model_path = config.train.pretrained_model_path
model = SequenceLightningModule.load_from_checkpoint(
            pretrained_model_path,
            config=config,
            strict=config.train.pretrained_model_strict_load,
        )

embedding_weights = model.model.backbone.embeddings.word_embeddings.weight

#print(embedding_weights)

model = model.to("cuda")
model.eval()


input = [7,  7, 10,  7,  7, 10,  7,  7,  7,  9, 10,  9,  8,  8, 10, 10]

input_x = input[:-1]
input_ids = torch.tensor([input_x], dtype=torch.long).to("cuda")

with torch.no_grad():
    output, state = model.model.forward(input_ids)
    print(output.logits)

logits = output.logits

print(f"Shape logits: {logits.shape}")

pred_ids = torch.argmax(logits, dim=-1)
print(f"Input ids: {input}")
print(f"True ids: {input[1:]}")
print(f"Pred ids: {pred_ids}")
```

导入预训练的权重文件，并初始化model

Forward获取logits

argmax将logits转为prediction ids

```
Shape logits: torch.Size([1, 15, 16])
Input ids: [7, 7, 10, 7, 7, 10, 7, 7, 7, 9, 10, 9, 8, 8, 10, 10]
True ids: [7, 10, 7, 7, 10, 7, 7, 7, 9, 10, 9, 8, 8, 10, 10]
Pred ids: tensor([[ 7,  7,  7,  7,  7,  7,  7,  7,  7,  7,  7,  7,  7,  7, 10]],
```

正确率46.67%

模型推理

获取序列的embedding信息，可用于可视化和下游分类

需要明白model的框架，在正确的层后面取前向传播即可

```
features = {}

def hook_fn(module, input, output):
    features["last_hidden_state"] = output.detach()


handle = model.model.backbone.ln_f.register_forward_hook(hook_fn)
output, _   = model.model.forward(input_ids)
last_hidden_state = features["last_hidden_state"]

print(last_hidden_state.shape)
print(last_hidden_state)

last_hidden_state = model.model.backbone(input_ids)
print(last_hidden_state.shape)
print(last_hidden_state)
```

```
torch.Size([1, 15, 128])
tensor([[[ 0.7827, -1.2525,  1.1667,  ..., -0.5038, -0.7896,  0.9977],
         [ 0.7948, -1.2367,  1.1676,  ..., -0.6339, -0.8245,  0.9781],
         [ 0.8619, -1.0982,  1.3225,  ..., -0.6707, -1.0373,  1.0955],
         ...,
         [ 0.5735, -2.1165,  0.4356,  ..., -0.4281, -0.4211,  1.1095],
         [ 0.4180, -2.3366,  0.1654,  ..., -0.3489, -0.1096,  1.3109],
         [ 0.5360, -0.7765,  1.2067,  ..., -0.9045, -1.2236,  0.9671]]],
       device='cuda:0')
torch.Size([1, 15, 128])
tensor([[[ 0.7827, -1.2525,  1.1667,  ..., -0.5038, -0.7896,  0.9977],
         [ 0.7948, -1.2367,  1.1676,  ..., -0.6339, -0.8245,  0.9781],
         [ 0.8619, -1.0982,  1.3225,  ..., -0.6707, -1.0373,  1.0955],
         ...,
         [ 0.5735, -2.1165,  0.4356,  ..., -0.4281, -0.4211,  1.1095],
         [ 0.4180, -2.3366,  0.1654,  ..., -0.3489, -0.1096,  1.3109],
         [ 0.5360, -0.7765,  1.2067,  ..., -0.9045, -1.2236,  0.9671]]],
       device='cuda:0', grad_fn=<NativeLayerNormBackward0>)
```

通过 forward hook 截取

PyTorch是一个流行的深度学习框架，以其动态图和易于使用的接口而受到广泛欢迎。在其设计中，hook机制是一个非常实用的功能，它允许开发者在不修改网络结构的前提下，介入到模型的前向传播和反向传播过程中。

Hook机制主要通过以下三种函数实现：

- register_forward_hook()：这个函数允许我们在某个模块的前向传播完成后注册一个回调函数。这个回调函数会接收到该模块的输入和输出，从而让我们有机会获取和分析中间层的输出特征。
- register_backward_hook()：与register_forward_hook()类似，这个函数允许我们在反向传播过程中注册一个回调函数。这个回调函数会在计算完模块的梯度后被调用，接收模块的输入梯度和输出梯度，这有助于我们理解和可视化梯度流动的过程。
- register_hook()：这是一个更底层的接口，可以直接在Tensor级别注册hook。当该Tensor的梯度被计算时，注册的回调函数会被调用。这为自定义梯度计算、监控特定变量的梯度行为以及进行更复杂的操作提供了灵活性。

相当于在前向传播过程中设置一个路障，等到了设定的地方，就输出

显式调用 backbone 的 forward

模型推理

获取序列的embedding信息，可用于可视化和下游分类

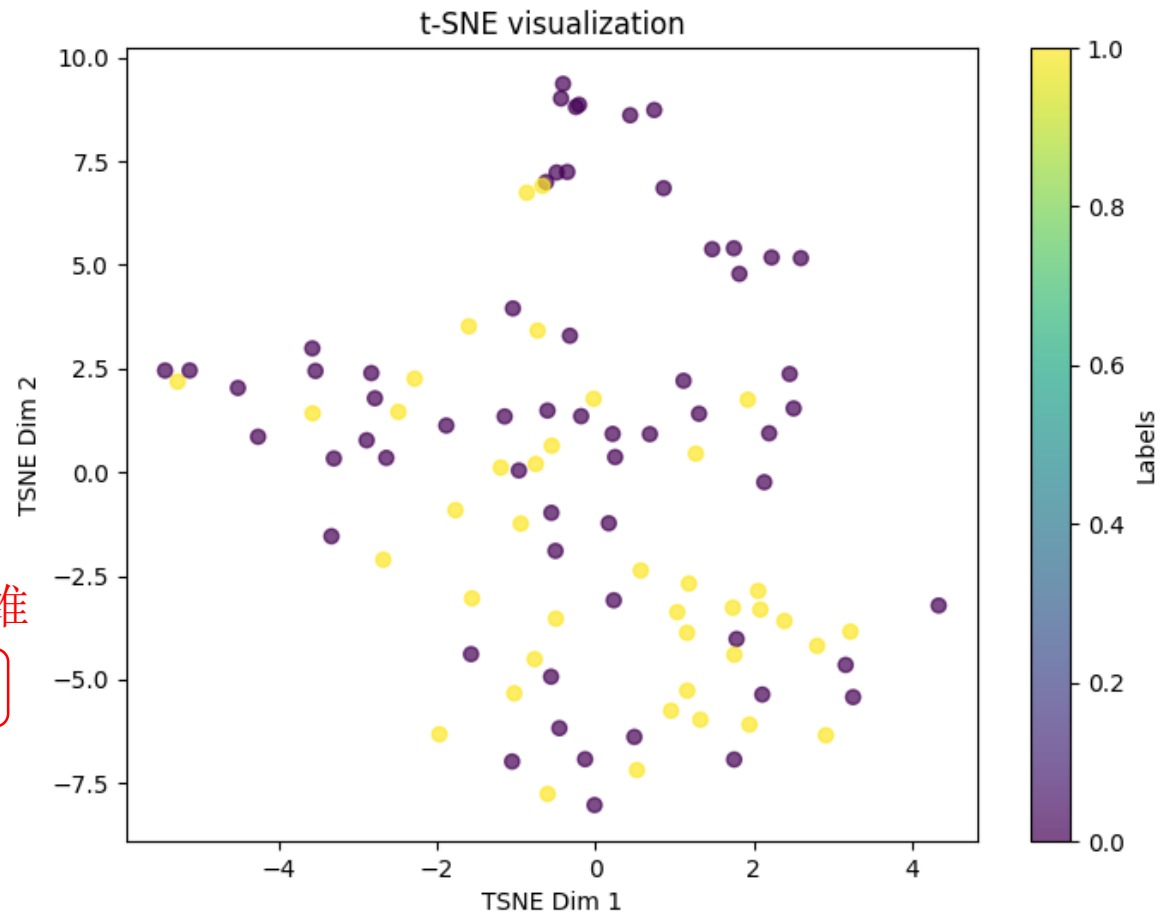
数据来源：nucleotide transformer enhancer dataset

https://huggingface.co/datasets/InstaDeepAI/nucleotide_transformer_downstream_tasks_revised/tree/main/enhancers

```python
from src.dataloaders.datasets.hg38_char_tokenizer import CharacterTokenizer
import pandas as pd

test_data = pd.read_parquet("test.parquet")
seq_data = test_data['sequence'][:100]
label = test_data['label'][:100]

tokenizer = CharacterTokenizer(
                characters=['A', 'C', 'G', 'T', 'N'],
                model_max_length=401,
                add_special_tokens=False,
                add_eos = False,)

input_ids = tokenizer(seq_data.tolist())["input_ids"]
input_ids = torch.tensor(input_ids, dtype=torch.long)
input_ids = input_ids.to("cuda")
last_hidden_state = model.model.backbone(input_ids)

embedding = last_hidden_state.mean(dim=1)
embedding = embedding.detach().cpu().numpy()

import matplotlib.pyplot as plt
from sklearn.manifold import TSNE

def plot_tsne(features, perplexity=30, n_iter=1000, random_state=42):
    ###
    tsne = TSNE(n_components=2, perplexity=perplexity, n_iter=n_iter, random_state=random_state)
    features_2d = tsne.fit_transform(features)
    plt.figure(figsize=(8, 6))
    scatter = plt.scatter(features_2d[:, 0], features_2d[:, 1], c=label, alpha=0.7)
    plt.colorbar(scatter, label='Labels')
    plt.title('t-SNE visualization')
    plt.xlabel('TSNE Dim 1')
    plt.ylabel('TSNE Dim 2')
    plt.show()

plot_tsne(embedding)
```

取隐空间

t-sne降维



前100个序列的t-sne降维可视化

总结

- Hyena-dna是decoder-only，用于下一个词元预测。下一个词元预测的时候，每个token的输出是下一个token的预测值
- Hyena-dna是基于hydra的高度封装结果，想要自定义，需要自己拆分调用逻辑和函数类初始化
- Hyena-dna计算的loss和accuracy是基于batch的所有token计算的，算的是batch-level