Intro of Transformer and PanGu

June 2, Yaohui Ding

About “Transformer”

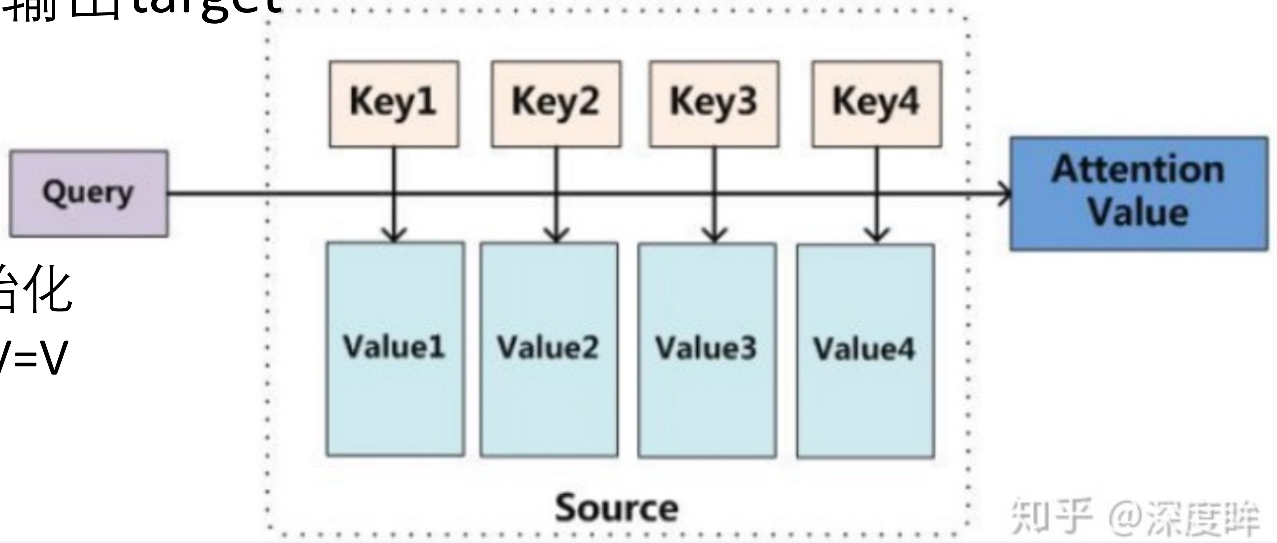
- Transformer in NLP
  - “Attention is all you need”, by google 2017. BERT is based on, no convolutional computing, take instead of CNN, RNN or LSTM
  - Generative Pre-trained Transformer, GPT-1 by OpenAI 2018
  - GPT-3, by OpenAI 2020 (open-sourced model is from NVIDIA MegetronLM and Microsoft DeepSpeed)
  - PanGu-Alpha, by Huawei, April 2021

- Major Advantages of Transformer (compared to CNN, RNN, LSTM)
  - attention-based: 自动聚焦到某一部分，而不需要逐行扫描，其本质是对输入进行自适应的加权，分配大的权重到认为重要的部分
  - better context info., “词与词之间相关性不取决于距离，而取决于相关性”
  - better parallel computing (compared to RNN): RNN needs computing hidden state flow 1-by-1
  - 重复结构，适合叠加构成大模型

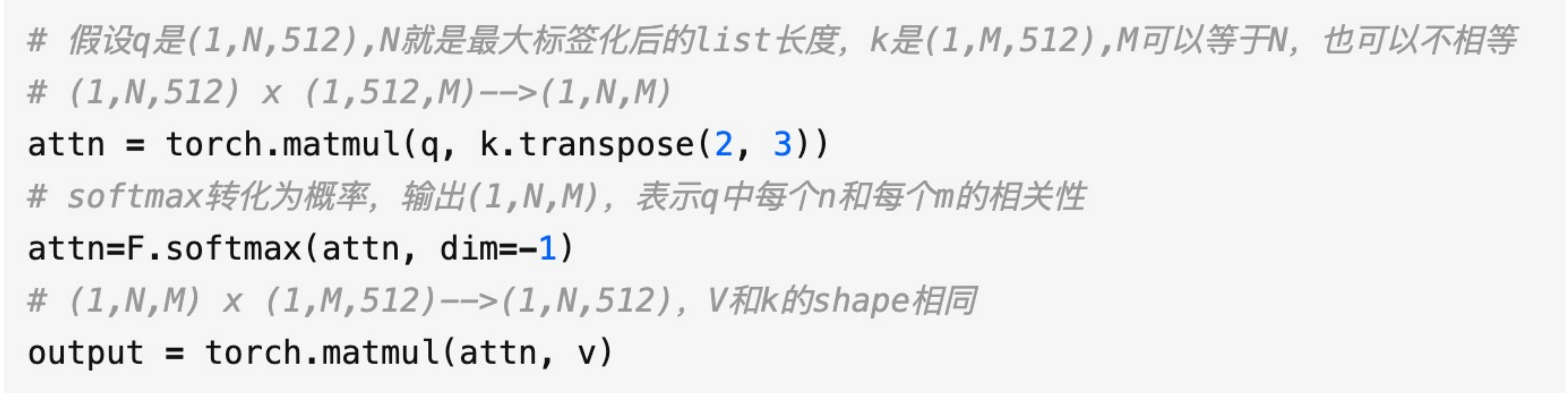
About Transformer

- Encoder-Decoder (PanGu has 36 encoders and 15 decoders)
  - encoder 把输入source 编码成统一的表示向量， embedding
  - decoder 把表示向量再解码成目标输出target

- QKV (attention核心)
  - WQ, WK, WV是3个自学习矩阵， 需要初始化通过网络自学习， X*WQ=Q, X*WK=K, X*WV=V
  - Q和K计算相关性， 转化为概率q
  - V和K同shape， V和q点乘， 即为attention



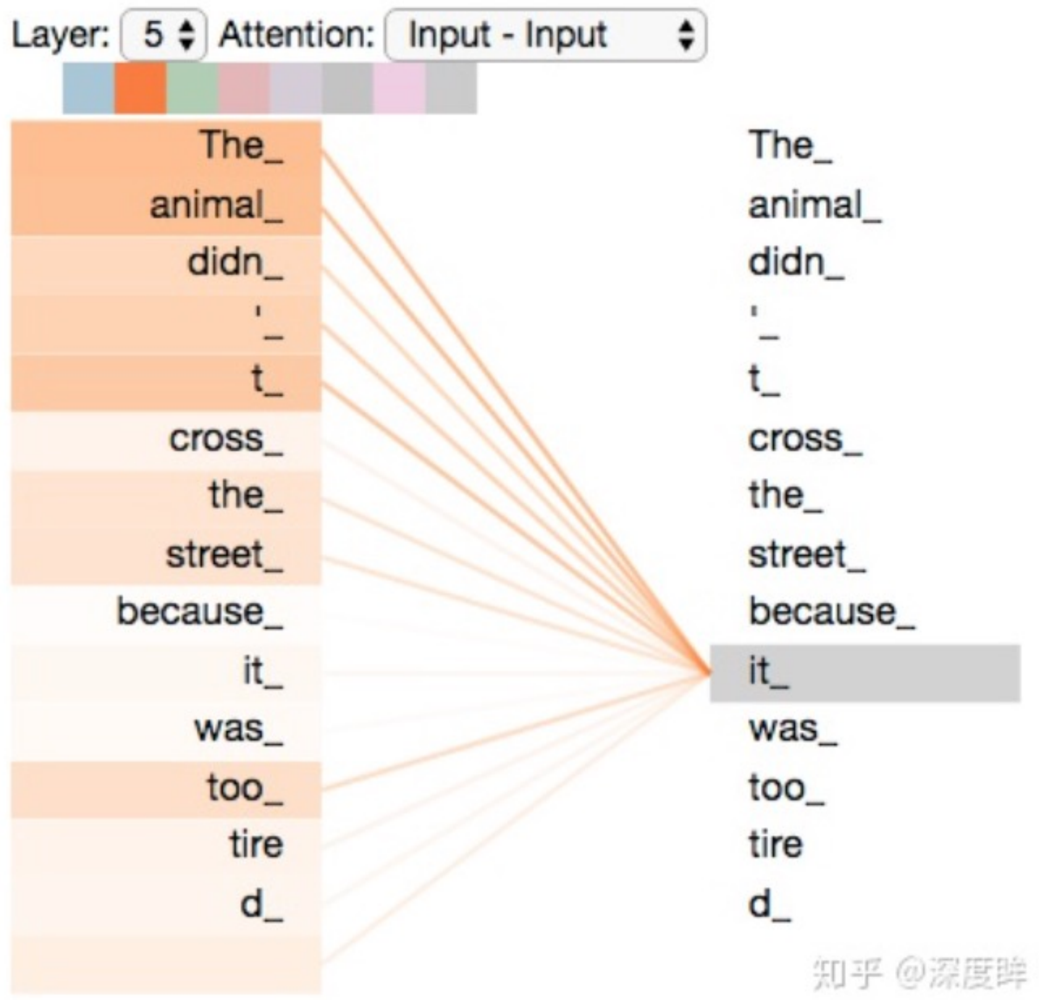
通过训练收敛后的QKV， 可以让每个单词的编码， 实际上都embed了上下文中其余单词的信息



```
# 假设q是(1,N,512),N就是最大标签化后的list长度，k是(1,M,512),M可以等于N，也可以不相等
# (1,N,512) x (1,512,M)-->(1,N,M)
attn = torch.matmul(q, k.transpose(2, 3))
# softmax转化为概率，输出(1,N,M)，表示q中每个n和每个m的相关性
attn=F.softmax(attn, dim=-1)
# (1,N,M) x (1,M,512)-->(1,N,512)，V和k的shape相同
output = torch.matmul(attn, v)
```

https://colab.research.google.com/github/tensorflow/tensor2tensor/blob/master/tensor2tensor/notebooks/hello_t2t.ipynb#scrollTo=OJKU36QAfqOC

Google provides a visualization tool for this attention scheme



About Transformer

1 - encoder vs. decoder

- input Q, K, V
- self-attention vs. multi-head attention
- feed forward, residual, layernorm …

2 - input dim = max_length(input seq)

- e.g. "I am a student", input dim = (max, 512) rather than (4, 512)

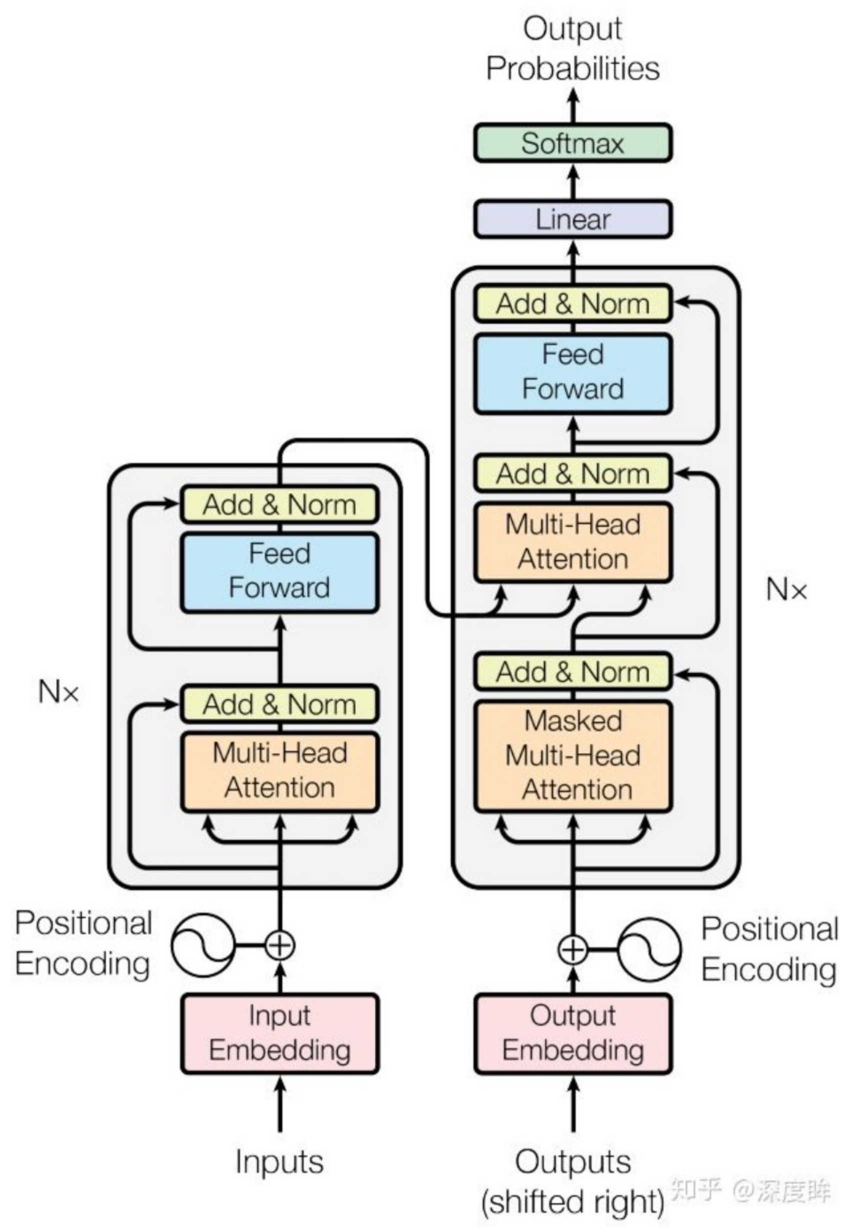
3 - positional encoding

- position of each word in a sequence

4 - mask

- convert batch input into step input

The attention block is built in torch.nn



A Simplified Transformer

About “Transformer”

- Transformer in Computer Vision
  - “An Image is Worth 16*16 Words”, by Google, Oct 2020, aka. **ViT**

    - “use transformer on sequences of image patches”,

    - this is a classification task, so only use transformer-encoder (no decoder)

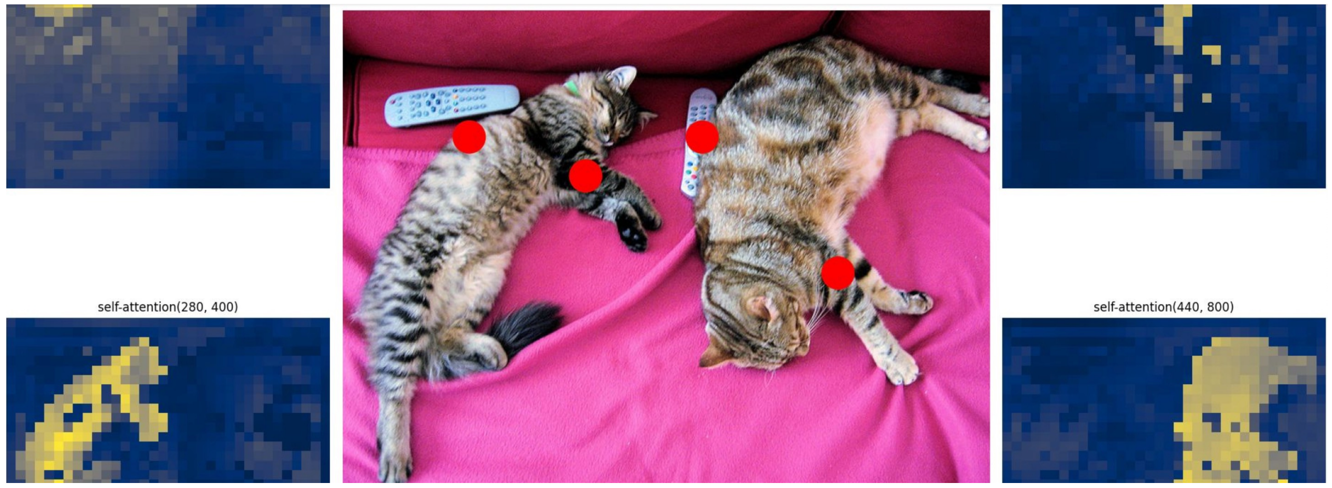
    - “CNN is not necessary”, however, “using CNN feature map is better…”

    https://github.com/lucidrains/vit-pytorch/blob/main/vit_pytorch/vit.py

  - “End-to-End Object Detection with Transformers”, by Facebook, Mar 2020, aka **DETR**
    - 1st transformer implementation on object detection task
    - novel CV architecture (no anchor generation, etc.), concise implementation
    - seems like not that revolutionary, use CNN feature map, performance (training time and accuracy) is similar to FastRCNN

Some Comments

- Even more popular in 2021
  - Better explainability, “attention is weight-based”
  - Keep exactly same encoder-decoder details, so CVers can borrow more from the NLP



- In CV, hard to say “CNN is dead”,
  - e.g., feature maps are still from CNN backbones
  - needs more works to prove it is better
    - if training size is small, …
    - computing resource

PanGu 介绍

- 华为HDC大会发布两大盘古模型 1）encoder-decoder 中文理解模型， **2）PanGu-Alpha, a pre-trained decoder 已开源**

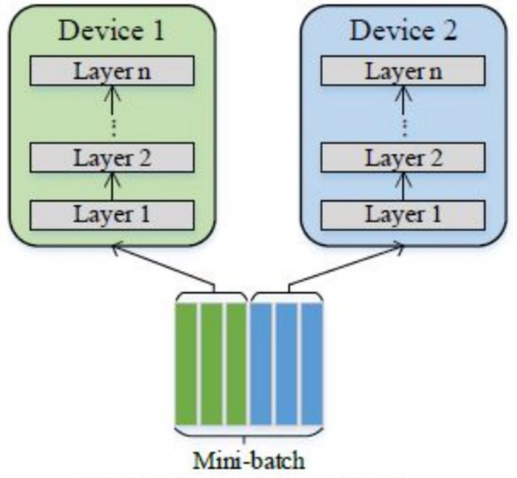
千亿级模型	参数量	样本	训练框架	简述
PanGu-Alpha	200B（2000亿）	80T (中文), 1T	MindSpore + CANN + 2048晟腾DSC芯片	5D混合并行
GPT-3	175B	45T (非中文), 570G	Pytorch + 标准CPUGPU	
Baidu 文心 ERNIE	100B		PaddlePaddle + 标准CPUGPU	4D混合并行

- 上述NLP大模型的目标一致，就是更强的泛化能力
  - 更少的domain data，不要在domain data上overfitting
  - 不需要finetune就能有很好效果
    - zero/one/few shot型任务上优势显著
- 基于MindSpore framework的5D混合并行

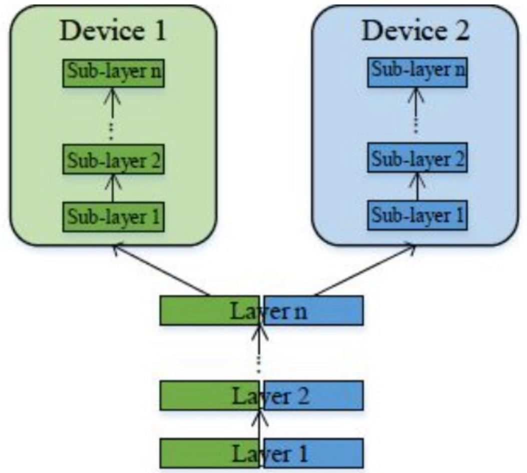
PanGu的5D混合并行

1. 数据并行
2. 模型并行（基于算子级）
3. 流水并行（Pipeline）
4. 重计算
5. 优化器并行

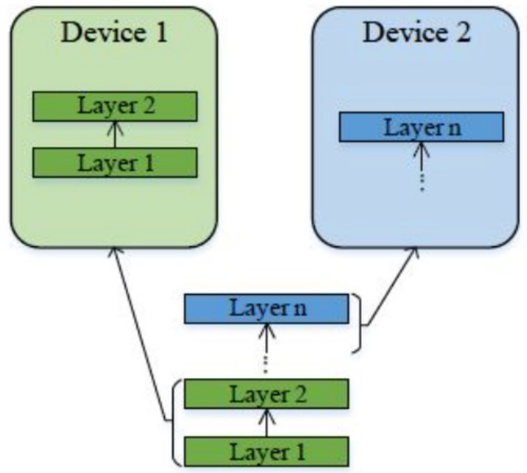
more, 图算融合， 根据通信比的拓扑设计, API设计等



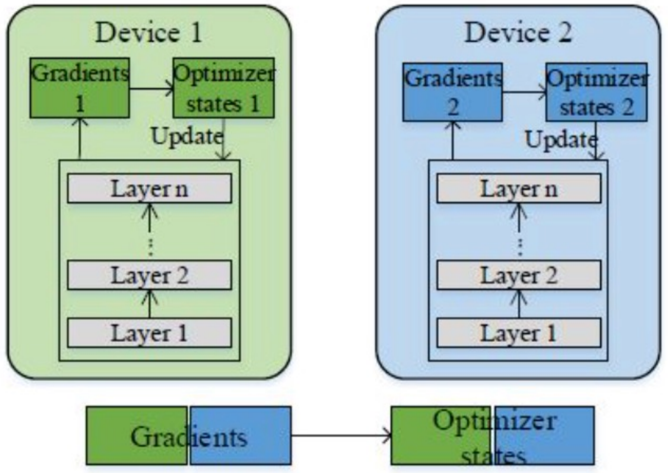
(a) Data is partitioned in data parallelism



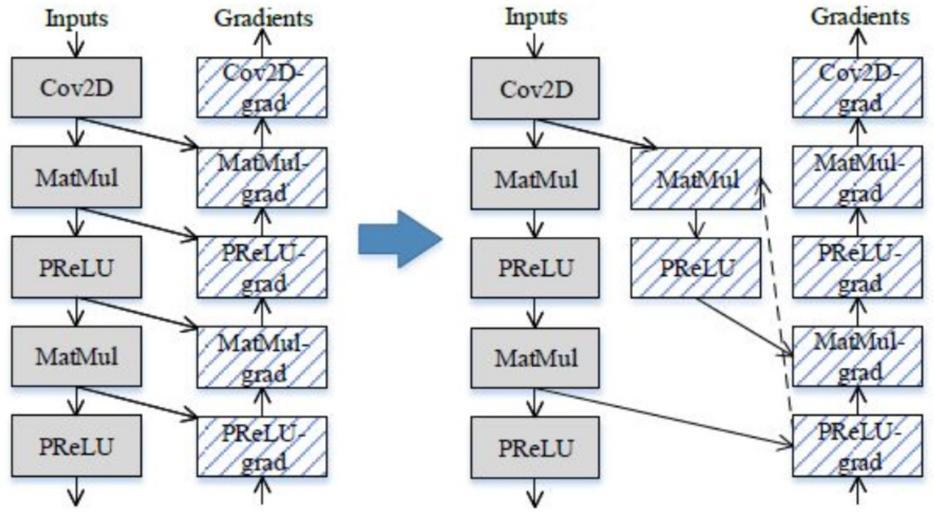
(b) Each layer is partitioned in op-level model parallelism



(c) Different layers are placed on different devices in pipeline model parallelism



(d) Optimizer states and gradients are partitioned along data parallelism in optimizer model parallelism



(e) Some activation memories are abandoned in the forward phase to reduce the peak memory consumption

PanGu的5D混合并行

1. 数据并行： 基于MindSpore的mini-batch, 设备间梯度同步， 再更新

2. 模型并行：基于算子的切分

- 矩阵乘算子MatMul(x, w)为例， x是训练数据， w是模型参数， 并行策略((4, 1), (1, 1))表示将x按行切4份， 保持w不切， 如果一共有4台设备， 那么每台设备拥有一份x的切片， 和完整的w
- **其它算子如**ReLu, Softmax, dropout都有被重建
- Layer由算子构成， 算子的切分使得layer更好的切分， 同时要**结合流水并行**

友商做法：

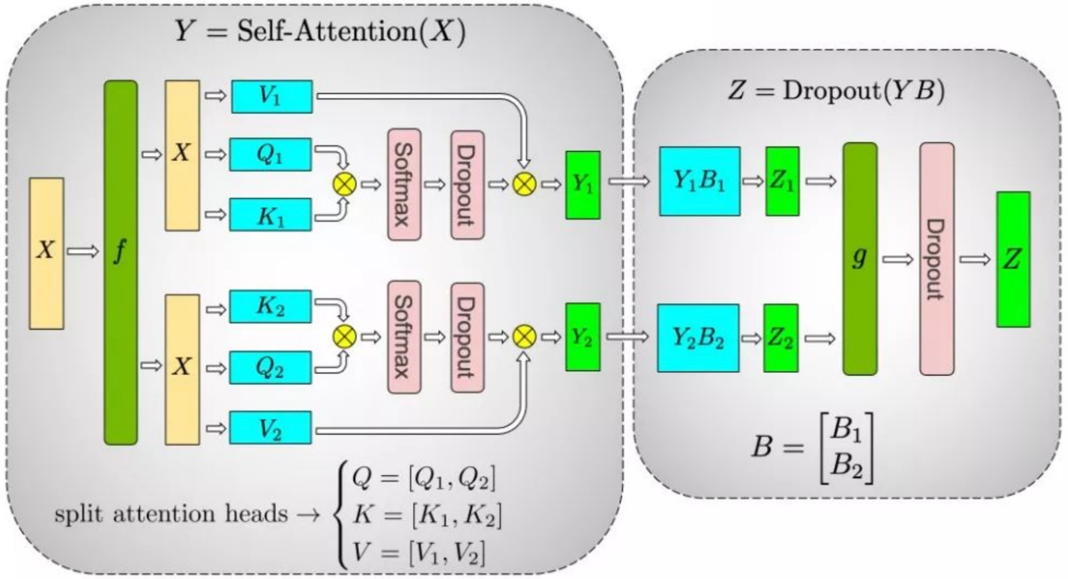
1）pytorch DDP自带的模型并行其实是pipeline并行， 没有切分算子 (下页)

2）NVIDIA的MegaTron的模型并行， 修改了mesh-tensorflow的做法， 就是将layer里的variable以列或行为单位， 把大tensor切分成小tensor （如右）

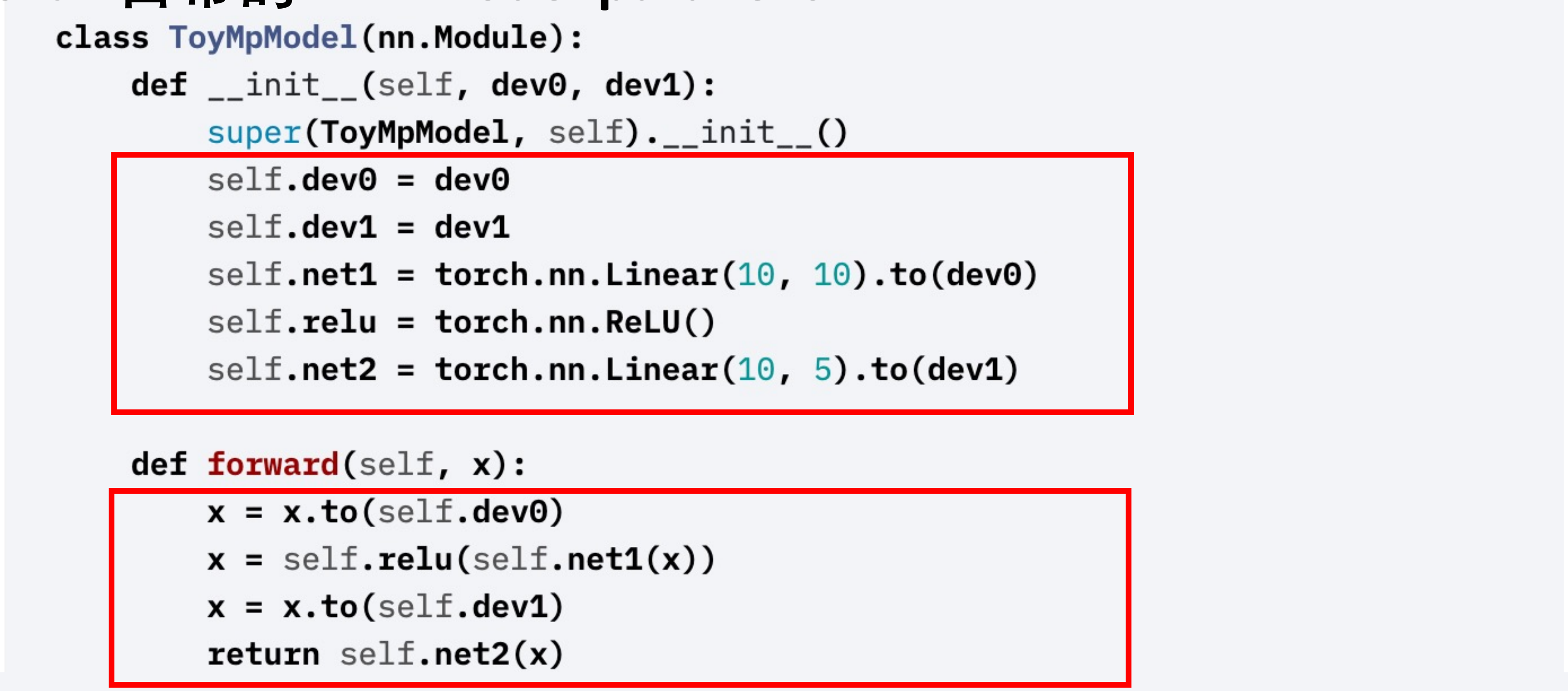
3）Google 的Gpipe， 和我们做法很类似， 但只支持TPU上， 归类为流水并行

4）OpenAI， 在GPU上的Gpipe， 结合重计算

5）MS的Zero Infinity首先提出算子切片

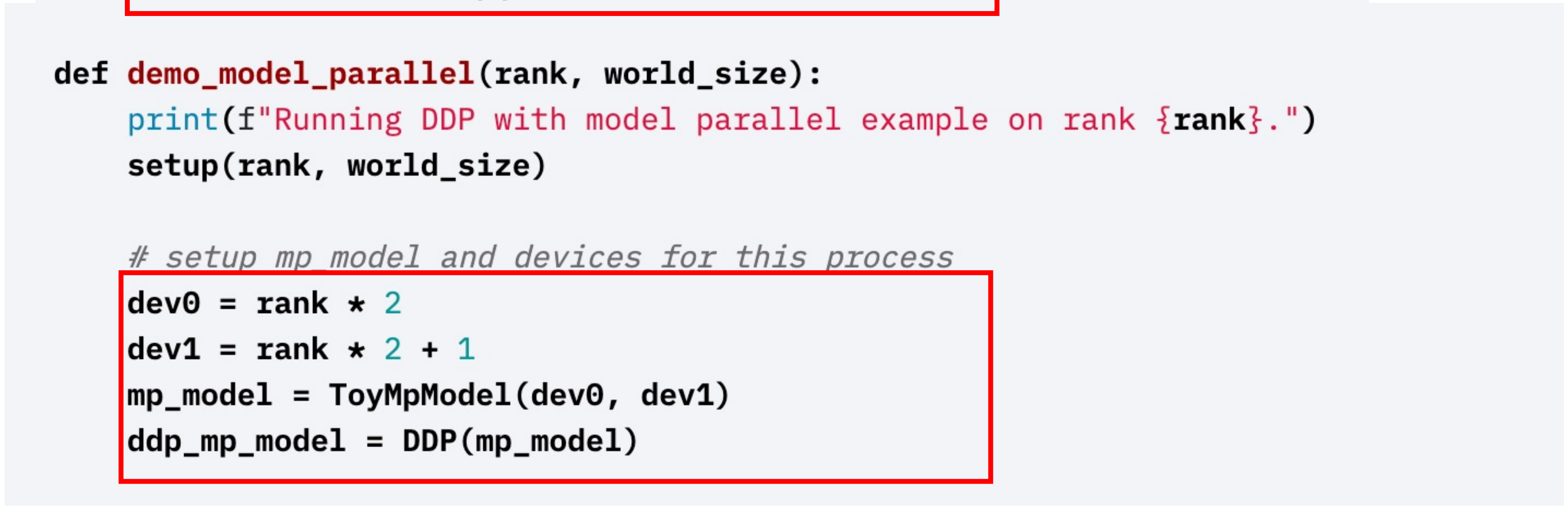


Pytorch 自带的DDP model parallelism



```python
class ToyMpModel(nn.Module):
    def __init__(self, dev0, dev1):
        super(ToyMpModel, self).__init__()
        self.dev0 = dev0
        self.dev1 = dev1
        self.net1 = torch.nn.Linear(10, 10).to(dev0)
        self.relu = torch.nn.ReLU()
        self.net2 = torch.nn.Linear(10, 5).to(dev1)

    def forward(self, x):
        x = x.to(self.dev0)
        x = self.relu(self.net1(x))
        x = x.to(self.dev1)
        return self.net2(x)
```



```python
def demo_model_parallel(rank, world_size):
    print(f"Running DDP with model parallel example on rank {rank}.")
    setup(rank, world_size)

    # setup mp model and devices for this process
    dev0 = rank * 2
    dev1 = rank * 2 + 1
    mp_model = ToyMpModel(dev0, dev1)
    ddp_mp_model = DDP(mp_model)
```

PanGu的5D混合并行

3. 流水并行：按layer分成不同的stage，以时间接力方式运行

- 和mini-batch类似的分成了micro-batch，但是不是空间上切分，而是时间上切分，这就是Gpipe
- 用Nvidia新论文举例，3072张A100，这个GPT有1T的参数，128层transformer layer，被分成了64个stage，每个 stage 跑在 6台 DGX-A100 上，其中 6 台机器之间进行数据并行，每台机器内部的 8 张卡之间做模型并行，整个集群的 3072 张 A100 按照机器拓扑被划分成了 [6 x 8 x 64] 的矩阵

- issue：分stage容易，但是正向计算后，还是需要整合全部计算量再进行activation才能反向计算
  - 提早开始反向计算，不等activation，而是重新计算一次正向计算，即重计算

4. 重计算 Rematerialization:

1 - gradient-checkpointing，就是根据checkpoint的tensor重算一遍前向中间值 (Gpipe)
2 - CPU offload，就是把activation的结果输出到CPU主存，从而节省GPU显存

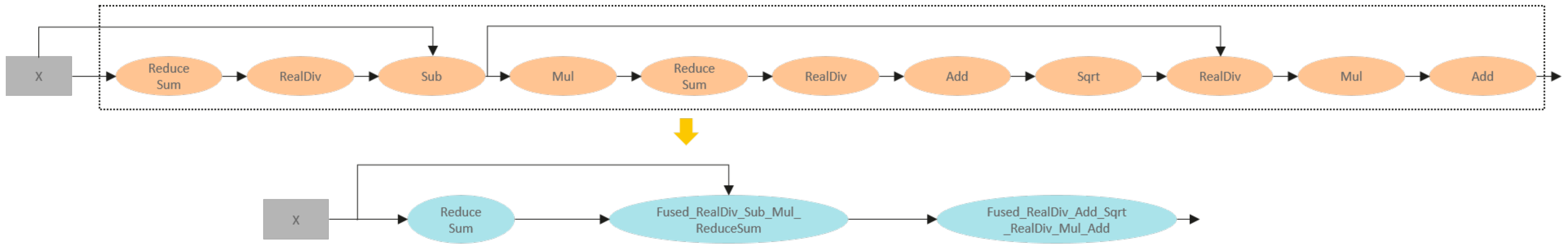
- 1是用计算开销换显存，2是用传输开销换显存
- 重计算的配置，是根据实验获得的，重计算引入的多余的计算量不得超过一层的计算量
  - PanGu中64 layer -> 16 stage，input tensor 按行分为16片，QKV 按列分为8片，so output tensor 128 片

PanGu的5D混合并行

5. 优化器并行: 就是把参与优化的参数和梯度切分到多台设备，比如adam里的momentum，切片，让每个卡只保留切片而不是全部的momentum

More

- 图算融合：对算子进行拆分和重组，
  - “用户使用角度的易用性算子” -> “硬件执行角度的高性能算子”
  - 例：将LayerNorm算子由11个小算子转成1个单算子和2个融合算子



- 硬件拓扑优化：
  - 对带宽要求低的，比如数据并行和优化器并行，允许放在不同机架的机器上
  - 通信量很大的，比如算子级并行，切片放在同一服务器的不同卡上

Comments

- 开源的这个小的模型，重在内容生成，更像GPT3；没开源的更大，除了生成还有理解能力

- 大模型主要考虑推高pretrain模型的上限，其实并没怎么考虑商用的问题
  - 端云，调用云侧提供的推理服务API
  - 将大模型小型化，类似华为的tiny bert提供tiny pangu，接受精度损失

- 个人观点：我们不需要追逐大模型的训练或finetune，而更应该关注，"如何在有限的算力（或预算）下实现中型模型的训练或微调"
  - 例如ZeRO-infinity之类，"如何以更少的机器，跑更大的模型"