Paddle模块介绍和分析

**倪子烜：**

1.代码总结与梳理：

要求：文字分条目列出，描述不少于200字，模块不少于7个

| 代码整体认识描述 | 代码模块或主要组成总结 |
| --- | --- |
| fluid.layers.nn函数库主要功能就是定义了神经网络中的大量计算操作(op)，其中总共有147个函数。 | 1.fc：构建全连接层 |
| 2.embedding:取出自己的embedding数据中的对应索引数据 |
| 3.linear\_chain\_crf：实现了线性链条件随机场（linear chain CRF）的前向—反向算法，主要用于NLP。 |
| 4.crf\_decoding：该层读取由 [linear\_chain\_crf](https://www.paddlepaddle.org.cn/documentation/docs/zh/develop/api_cn/layers_cn/linear_chain_crf_cn.html#cn-api-fluid-layers-linear-chain-crf) 学习的 emission feature weights（发射状态特征的权重）和 transition feature weights (转移特征的权重) 进行解码。 本层实现了 Viterbi 算法，可以动态地寻找隐藏状态最可能的序列，该序列也被称为 Viterbi 路径（Viterbi path），从而得到观察标签 (tags) 序列。 |
| 5.cos\_sim：实现了余弦相似度算子（Cosine Similarity Operator） |
| 6.dropout:实现dropout操作。 |
| 7.chunk\_eval:计算语块识别（chunk detection）的准确率、召回率和F1值，常用于命名实体识别（NER，语块识别的一种）等序列标注任务中。 |
|  | 8.softmax:实现softmax操作。 |
|  | 9.conv2d：～ |
|  | 10.conv3d：～ |
|  | 11.pool2d:~ |
|  | 12.pool3d:~ |
|  | 13.adaptive\_pool2d:~ |
|  | 14.adaptive\_pool3d:~ |
|  | 15.batch\_norm:~ |
|  | 16.inplace\_abn：一种activation batch normalization。<<https://arxiv.org/abs/1712.02616>> |
|  | 17.instance\_norm：实现实例正则化 |
|  | 18.data\_norm：数据正则化，可用作conv2d和fully\_connected操作的正则化函数 |
|  | 19.layer\_norm:层归一化层（Layer Normalization Layer），其可以应用于小批量输入数据。 |
|  | 20.group\_norm:实现group normalization。<https://arxiv.org/abs/1803.08494> |
|  | 21.spectral\_norm：用于计算fc、conv1d、conv2d、conv3d层的权重参数的谱正则值 |
|  | 22.conv2d\_transpose：二维转置卷积层（Convlution2D transpose layer） |
|  | 23.conv3d\_transpose：三维转置卷基层 |
|  | 24.reduce\_sum：对指定维度上的Tensor元素进行求和运算，并输出相应的计算结果。 |
|  | 25.reduce\_mean：对指定维度上的Tensor元素进行平均值算，并输出相应的计算结果。 |
|  | 26.reduce\_max：对指定维度上的Tensor元素求最大值运算，并输出相应的计算结果。 |
|  | 27.reduce\_min：对指定维度上的Tensor元素求最小值运算，并输出相应的计算结果。 |
|  | 28.reduce\_prod：对指定维度上的Tensor元素进行求乘积运算，并输出相应的计算结果。 |
|  | 29.reduce\_all：对指定维度上的Tensor元素进行与逻辑（&）计算，并输出相应的计算结果。 |
|  | 30.reduce\_any：对指定维度上的Tensor元素进行或逻辑（|）计算，并输出相应的计算结果。 |
|  | 31.split：将输入Tensor分割成多个子Tensor。 |
|  | 32.l2\_normalize：计算欧几里得距离之和对x进行归一化。 |
|  | 33.matmul：矩阵相乘 |
|  | 34.topk：查找输入Tensor的最后一维的前k个最大项，返回它们的值和索引。 |
|  | 35.ctc\_greedy\_decoder：贪婪策略解码序列 |
|  | 36.transpose：对输入的多维Tensor进行数据重排 |
|  | 37.im2sequence：使用 filter 扫描输入的Tensor并将输入Tensor转换成序列 |
|  | 38.row\_conv：行卷积（Row-convolution operator）或称之为超前卷积（lookahead convolution）<https://dyogatama.github.io/> |
|  | 39.multiplex：根据给定的index参数，该OP从每个输入Tensor中选择特定行构造输出Tensor。 |
|  | 40.smooth\_l1：计算变量 x 和 y 的smooth L1 loss |
|  | 41.one\_hot：将输入（input）中的每个id转换为一个one-hot向量 |
|  | 42.autoincreased\_step\_counter：创建一个自增变量，每个迭代累加一次，默认首次返回值为 1，默认累加步长为 1。 |
|  | 43.reshape：换形状 |
|  | 44.squeeze：会根据axes压缩输入Tensor的维度。如果指定了axes，则会删除axes中指定的维度，axes指定的维度要等于1。如果没有指定axes，那么所有等于1的维度都会被删除。 |
|  | 45.unsqueeze：向输入（input）的shape中一个或多个位置（axes）插入维度。 |
|  | 46.lod\_reset：根据给定的参数 y 或 target\_lod ，重设输入 x (LoDTensor) 的 LoD 信息。 |
|  | 47.lod\_append：给x的LoD添加level |
|  | 48.lrn：实现了局部响应正则化层（Local Response Normalization Layer），用于对局部输入区域正则化，执行一种侧向抑制（lateral inhibition）。 |
|  | 49.pad：在Tensor上填充一个由 pad\_value 给出的常数值，填充宽度由 paddings 指定 |
|  | 50.pad\_constant\_like：使用 pad\_value 填充 y ，填充到每个维度值的数量由x和y的形状而指定 |
|  | 51.label\_smooth：实现了标签平滑的功能。 |
|  | 52.roi\_pool：实现了roi池化操作，对非均匀大小的输入执行最大池化，以获得固定大小的特征映射 |
|  | 53.roi\_align：实现RoIAlign操作。 |
|  | 54:dice\_loss：用来比较预测结果跟标签之间的相似度，通常用于二值图像分割，即标签为二值，也可以做多标签的分割。 |
|  | 55:image\_resize：用于调整一个batch中图片的大小 |
|  | 56:resize\_bilinear：该OP应用双向性插值法调整输入图片的大小，输出形状按优先级由actual\_shape、out\_shape和scale指定。 |
|  | 57.resize\_trilinear：该层对输入进行放缩，基于给定的由 actual\_shape , out\_shape , scale 确定的输出shape，进行三线插值。 |
|  | 58.resize\_nearest：该OP对输入图片进行大小调整，在高度方向宽度方向进行最邻近插值（nearest neighbor interpolation）操作。 |
|  | 59.image\_resize\_short：该OP用于调整一批图片的大小。输入图像的短边将被调整为给定的out\_short\_len 。输入图像的长边按比例调整大小，最终图像的长宽比保持不变。 |
|  | 60.gather：根据索引 index 获取输入（input）的最外层维度的条目，并将它们拼接在一起。 |
|  | 61.gather\_nd：该OP是gather的高维推广，并且支持多轴同时索引。 |
|  |  |
|  | 62.scatter：该OP根据index中的索引值将updates数据更新到input中。 |
|  | 63.scatter\_nd\_add：该OP通过对Variable中的单个值或切片应用稀疏加法，从而得到输出的Variable。 |
|  | 64.scatter\_nd：该OP根据 index ，将 updates 添加到一个新的张量中，从而得到输出的Variable。 |
|  | 65.random\_crop：该操作对batch中每个实例进行随机裁剪，即每个实例的裁剪位置不同，裁剪位置由均匀分布随机数生成器决定。所有裁剪后的实例都具有相同的维度，由 shape 参数决定。 |
|  | 66.log：实现log操作 |
|  | 67.relu：实现relu操作 |
|  | 68.selu：实现SeLU激活函数 |
|  | 69.mean\_iou：该OP计算均值IOU |
|  | 70.crop：该OP根据偏移量(offsets)和形状(shape)，裁剪输入张量。 |
|  | 71.crop\_tensor：根据偏移量（offsets）和形状（shape），裁剪输入（x）Tensor。 |
|  | 72.affine\_grid：用于生成仿射变换前后的feature maps的坐标映射关系。在视觉应用中，根据该OP得到的映射关系，将输入feature map的像素点变换到对应的坐标，就得到了经过仿射变换的feature map。 |
|  | 73.pad2d：该OP依照 paddings 和 mode 属性对input进行2维 pad 。 |
|  | 74.elu：实现ELU激活层 |
|  | 75.relu6：实现relu6激活操作 |
|  | 76.pow：实现平方操作 |
|  | 77.stanh：STanh 激活算子 |
|  | 78.hard\_sigmoid：sigmoid的分段线性逼近激活函数，速度比sigmoid快。 <https://arxiv.org/abs/1603.00391>。 |
|  | 79.swish：逐元素计算 Swish 激活函数 |
|  | 80.prelu：实现prelu操作 |
|  | 81.brelu：实现BRELU激活操作 |
|  | 82.leaky\_relu：实现leaky\_relu操作 |
|  | 83.soft\_relu：～ |
|  | 84.flatten：输入的多维Tensor展平成2-D Tensor矩阵 |
|  | 85.stack：沿axis轴对输入x进行堆叠操作。 |
|  | 86.filter\_by\_instag：此函数通过instag来过滤ins batch，大量属于同样的tags的样本，我们可以指定我们想要的一些tags，属于这些tags的样本将会被保留在输出中，其余的将会移除。 |
|  | 87.unstack：将单个dim为 D 的Tensor沿 axis 轴unpack为 num 个dim为 (D-1) 的Tensor |
|  | 88.expand：该OP会根据参数 expand\_times 对输入 x 的各维度进行复制。通过参数 expand\_times 来为 x 的每个维度设置复制次数。 |
|  | 89.expand\_as：通过 target\_tensor的维度来为 x 的每个维度设置广播的次数，使得x 的维度与target\_tensor的维度相同。 |
|  | 90.uniform\_random\_batch\_size\_like：此操作初始化一个变量，该变量的随机值是从[min, max]范围内的均匀分布中采样的。用于获取输入维度值的input\_dim\_idx将用于调整输出维度的大小。 |
|  | 91.gaussian\_random：实现高斯随机 |
|  | 92.sampling\_id：从输入的多项分布中进行采样。 |
|  | 93.gaussian\_random\_batch\_size\_like：生成数据符合高斯随机分布的 Tensor。 |
|  | 94.sum：求和 |
|  | 95.slice：沿多个轴生成 input 的切片。 |
|  | 96.strided\_slice：与numpy的slice类似 |
|  | 97.shape：获得输入Tensor的shape。 |
|  | 98.rank：用于计算输入Tensor的维度（秩）。 |
|  | 99.size：用于计算输入tensor的尺寸 |
|  | 100.scale：缩放算子 |
|  | 101.elementwise\_add：是逐元素相加算子 |
|  | 102.elementwise\_div：相除 |
|  | 103.elementwise\_sub：相减 |
|  | 104.elementwise\_sub：相乘 |
|  | 105.elementwise\_max：逐元素取最大 |
|  | 106.elementwise\_max：取最小 |
|  | 107.elementwise\_pow：开方 |
|  | 108.elementwise\_mod:取模 |
|  | 109.elementwise\_floordiv：整除 |
|  | 110.logical\_and：逻辑与运算 |
|  | 111.logical\_or：逻辑或 |
|  | 112.logical\_xor：逻辑异或 |
|  | 113.logical\_not：逻辑非运算 |
|  | 114.clip：该OP对输入Tensor每个元素的数值进行裁剪，使得输出Tensor元素的数值被限制在区间[min, max]内。 |
|  | 115.clip\_by\_norm：此算子将输入 X 的L2范数限制在 max\_norm 内。如果 X 的L2范数小于或等于 max\_norm ，则输出（Out）将与 X 相同。如果X的L2范数大于 max\_norm ，则 X 将被线性缩放，使得输出（Out）的L2范数等于 max\_norm |
|  | 116.mean:所有元素的均值 |
|  | 117.merge\_selected\_rows：累加合并 [SelectedRows](https://github.com/PaddlePaddle/Paddle/blob/develop/paddle/fluid/framework/selected_rows.h) ( x ) 中的重复行，并对行值由小到大重新排序 |
|  | 118.mul：矩阵乘法 |
|  | 119.maxout：实现maxout操作。<http://www.jmlr.org/proceedings/papers/v28/goodfellow13.pdf> |
|  | 120.space\_to\_depth：对成块的空间数据进行重组，输出一个输入张量的拷贝，其高度和宽度维度上的值移至通道维度。 |
|  | 121. affine\_channel：对输入的每个 channel 应用单独的仿射变换。用于将空间批量归一化替换为其等价的固定变换。 |
|  | 122.similarity\_focus：实现SimilarityFocus(相似度聚焦)运算。<https://www.aclweb.org/anthology/N16-1108/> |
|  | 123.hash:实现哈希算子。 |
|  | 124.grid\_sampler：一种差值方法 |
|  | 125.log\_loss：对输入的预测结果和目标标签进行计算，返回负对数损失值。 |
|  | 126:add\_position\_encoding：将输入inpu中每个位置（序列中的位置）的特征与对应的位置编码加权求和。《Attention is all you need》 |
|  | 127.bilinear\_tensor\_product:对两个输入执行双线性张量积。 |
|  | 128.get\_tensor\_from\_selected\_rows：从SelectedRows类型的输入中获取向量数据，以LoDTensor的形式输出。 |
|  | 129.shuffle\_channel:将输入x的通道混洗重排。 它将每个组中的输入通道分成 group 个子组，并通过逐一从每个子组中选择元素来获得新的顺序。 |
|  | 130.temporal\_shift:用于对输入X做时序通道T上的位移操作，为TSM(Temporal Shift Module)中使用的操作 |
|  | 131.py\_func:自己应用选择的函数定义操作 |
|  | 132.psroi\_pool：执行PSROIPooling运算，是位置敏感的感兴趣区域池化方法（Position sensitive region of interest pooling，也称为PSROIPooling）。 |
|  | 133.prroi\_pool：精确区域池化方法（Precise region of interest pooling，也称为PRROIPooling） |
|  | 134.pixel\_shuffle：将一个形为[N, C, H, W]的Tensor重新排列成形为 [N, C/r\*\*2, H\*r, W\*r] 的Tensor。<https://arxiv.org/abs/1609.05158v2> |
|  | 135.fsp\_matrix：用于计算两个4-D Tensor特征图的求解过程（FSP）矩阵。 |
|  | 136.continuous\_value\_model：在CTR项目中，用于去除或处理 input 中的展示和点击值。只能用于cpu |
|  | 137.where：计算输入元素中为True的元素在输入中的坐标（index） |
|  | 138.sign：对输入x中每个元素进行正负判断，并且输出正负判断值：1代表正，-1代表负，0代表零。 |
|  | 139.unique：为x返回一个unique张量和一个指向该unique张量的索引。 |
|  | 140.unique\_with\_counts：对输入Tensor元素进行去重，获取去重后结果Tensor，同时获取去重后结果在原始输入中的计数Tensor以及在原始输入中的索引Tensor。 |
|  | 141.deformable\_conv：对输入4-D Tensor计算2-D可变形卷积。<https://arxiv.org/abs/1811.11168v2> |
|  | 142.unfold：im2col函数一样 |
|  | 143.deformable\_roi\_pooling:对输入进行了可形变的感兴趣区域(ROI)池化操作。 |
|  | 144.shard\_index:该函数对输入的索引根据分片（shard）的偏移量重新计算。 |
|  | 145.hard\_swish:实现了hard\_swish激活函数。 |
|  | 146.gather\_tree:从最后一个时间步回溯产生完整的id序列。 |
|  | 147.uniform\_random:使用从范围[min，max)内均匀分布采样的随机值初始化一个Tensor。 |

2.关键方法、算法、数据摘取与深入分析（Python或C++）：

要求：方法/公式/数据 + 文字，深入解读代码方法不少于7个

| 代码/算法/数据片段 | **输入输出及功能描述分析** |
| --- | --- |
| paddle.fluid.layers.conv2d(input, num\_filters, filter\_size, stride=1, padding=0, dilation=1, groups=None, param\_attr=None, bias\_attr=None, use\_cudnn=True, act=None, name=None, data\_format="NCHW") | 1.该OP是二维卷积层（convolution2D layer），根据输入、滤波器、步长（stride）、填充（padding）、膨胀比例（dilations）一组参数计算输出特征层大小。输入和输出是NCHW或NHWC格式，其中N是批尺寸，C是通道数，H是特征高度，W是特征宽度。滤波器是MCHW格式，M是输出图像通道数，C是输入图像通道数，H是滤波器高度，W是滤波器宽度。如果组数(groups)大于1，C等于输入图像通道数除以组数的结果。如果bias\_attr不为False，卷积计算会添加偏置项。如果指定了激活函数类型，相应的激活函数会作用在最终结果上。 |
| paddle.fluid.layers.pool2d(input, pool\_size=-1, pool\_type='max', pool\_stride=1, pool\_padding=0, global\_pooling=False, use\_cudnn=True, ceil\_mode=False, name=None, exclusive=True, data\_format="NCHW") | 2.该OP使用上述输入参数的池化配置，为二维空间池化操作，根据 input ，池化核大小 pool\_size ，池化类型 pool\_type ，步长 pool\_stride ，填充 pool\_padding 参数得到输出。输入 input 和输出（out）采用NCHW或NHWC格式，N为批大小，C是通道数，H是特征高度，W是特征参数 pool\_size 和 pool\_stride 含有两个整型元素，分别表示高度和宽度维度上的参数。输入 input 和输出（out）的形状可能不同。 |
| paddle.fluid.layers.cross\_entropy(input, label, soft\_label=False, ignore\_index=-100) | 3.该OP计算输入input和标签label间的交叉熵，可用于计算硬标签或软标签的交叉熵。 |
| paddle.fluid.layers.dropout(x, dropout\_prob, is\_test=False, seed=None, name=None, dropout\_implementation='downgrade\_in\_infer') | 4.丢弃或者保持x的每个元素独立。Dropout是一种正则化手段，通过在训练过程中阻止神经元节点间的相关性来减少过拟合。根据给定的丢弃概率，dropout操作符按丢弃概率随机将一些神经元输出设置为0，其他的仍保持不变。dropout op可以从Program中删除，提高执行效率。 |

3.实验结果呈现与说明（如调试模型的某个中间环节）：

要求：结果截图 + 文字，能证明以上解读的实验不少于3项

| 实验结果截图 | **描述说明** |
| --- | --- |
|  | 1.fluid.layers.nn函数主要是一些功能块。依照不同的功能实现不同的操作。其中以动态图为主。 |
|  | 2.通过实现全连接层来实践整个计算和训练的过程。 |
|  | 3.实现了一个完整的minst训练和测试过程。 |

**李俊成：**

1. 代码总结与梳理：

要求：文字分条目列出，描述不少于200字，模块不少于7个

|  |  |
| --- | --- |
| 代码整体认识描述 | **代码模块或主要组成总结** |
| 1. param\_attr.py模块用于保存参数的属性值，在组建网络的时候对网络参数的属性值进行初始化，包括参数的名称、初始化的方法、学习速率、正则项、是否进行梯度更新和是否进行梯度截断。param\_attr.py模块由类ParamAttr和类WeightNormParamAttr构成。其中类WeightNormParamAttr继承于ParamAttr，在ParamAttr的基础上加了规范化。  2. executor.py模块定义了支持单GPU、多GPU以及CPU运行的静态图的执行器。  3. io.py模块定义了一系列关于输入输出的接口函数，包括将输入数据打包成batch\_size大小的reader装饰器、缓存数据读取器、数据加载器、模型参数加载和保存等IO接口。  4. unique\_name.py模块用于对参数的命名空间进行管理。具体地，generate函数用于产生以前缀key开头的唯一名称。guard函数用于更改命名空间，和with语句一同使用。switch语句用于切换当前上下文环境中的命名空间。 | 1. 类ParamAttr |
| 1. 类WeightNormParamAtt |
| 1. 类Executor |
| 1. global\_scope函数 |
| 1. scope\_guard函数 |
| 1. load函数 |
| 1. save函数 |
| 1. get\_program\_parameter函数 |
| 1. get\_program\_persistable\_vars函数 |
| 1. generate函数 |
| 1. guard函数 |
| 1. switch函数 |
|  |

1. 关键方法、算法、数据摘取与深入分析（Python或C++）：

要求：方法/公式/数据 + 文字，深入解读代码方法不少于7个

|  |  |
| --- | --- |
| **代码/算法/数据片段** | **输入输出及功能描述分析** |
| 1. param\_attr.ParamAttr.\_\_init\_\_ | ParamAttr类保存了参数的属性值，在构建网络的时候作为一个参数传入，对所构建网络的属性值进行初始化。其中ParamAttr类最重要的方法便是它的初始化方法。\_\_init\_\_方法的输入是参数的名称、初始化的方法、学习速率、正则项、是否进行梯度更新和是否对模型参数进行平均优化。该方法首先调用check\_type方法，对传入的参数类型进行检测，若类型符合要求，则将输入参数值作为类的实例的属性进行保存。当创建网络的时候，将该实例作为参数传入。例如，w\_param\_attrs = fluid.ParamAttr(name="fc\_weight", learning\_rate=0.5, regularizer=fluid.regularizer.L2Decay(1.0), trainable=True)  x = fluid.data(name='X', shape=[None, 1], dtype='float32')  y\_predict = fluid.layers.fc(input=x, size=10, param\_attr=w\_param\_attrs)  先初始化了一个ParamAttr实例，接着，在调用fluid.layers.fc创建全连接层的时候，将ParamAttr实例作为param\_attr参数传入，确定了该层网络的参数名称、学习速率、正则项等。 |
| 2. executor.Executor.run | 执行指定的Program或者CompiledProgram。需要注意的是，执行器会执行Program或CompiledProgram中的所有算子，而不会根据fetch\_list对Program或CompiledProgram中的算子进行裁剪。同时，需要传入运行该模型用到的scope，如果没有指定scope，执行器将使用全局scope，即fluid.global\_scope()。run函数的输入参数包括：  program (Program|CompiledProgram) – 该参数为被执行的Program或CompiledProgram，如果未提供该参数，即该参数为None，在该接口内，main\_program将被设置为fluid.default\_main\_program()。默认为：None。  feed (list|dict) – 该参数表示模型的输入变量。如果是单卡训练，feed 为 dict 类型，如果是多卡训练，参数 feed 可以是 dict 或者 list 类型变量，如果该参数类型为 dict ，feed中的数据将会被分割(split)并分送给多个设备（CPU/GPU），即输入数据被均匀分配到不同设备上；如果该参数类型为 list ，则列表中的各个元素都会直接分别被拷贝到各设备中。默认为：None。  fetch\_list (list) – 该参数表示模型运行之后需要返回的变量。默认为：None。  feed\_var\_name (str) – 该参数表示数据输入算子(feed operator)的输入变量名称。默认为："feed"。  fetch\_var\_name (str) – 该参数表示结果获取算子(fetch operator)的输出变量名称。默认为："fetch"。  scope (Scope) – 该参数表示执行当前program所使用的作用域，用户可以为不同的program指定不同的作用域。默认值：fluid.global\_scope()。  return\_numpy (bool) – 该参数表示是否将返回返回的计算结果（fetch list中指定的变量）转化为numpy；如果为False，则每个变量返回的类型为LoDTensor，否则返回变量的类型为numpy.ndarray。默认为：True。  use\_program\_cache (bool) – 该参数表示是否对输入的Program进行缓存。如果该参数为True，在以下情况时，模型运行速度可能会更快：输入的program为 fluid.Program ，并且模型运行过程中，调用该接口的参数（program、 feed变量名和fetch\_list变量）名始终不变。默认为：False。  Executor.run函数的返回值为fetch\_list中指定的变量值，fetch\_list类型为list。 |
| 3 .executor.Executor.train\_from\_dataset | 在创建完执行器后，train\_from\_dataset函数从预定义的数据集中训练。 数据集在Paddle的高性能IO模块paddle.fluid.dataset中定义。 给定Program（或CompiledProgram），train\_from\_dataset将使用paddle.fluid.dataset中的所有数据样本。输入scope可由用户给出, 默认情况下使用的scope是global\_scope()。训练中的线程数是thread个， 默认值为0，表示使用paddle.fluid.dataset中用户配置的线程数。 可以设置debug，以便执行器显示所有算子的运行时间和当前训练任务的吞吐量。当用户设置fetch\_list和fetch\_info时 （两者长度需要一致）时，会打印出fetch\_list中所有变量的值，打印该值的间隔为print\_period。  train\_from\_dataset的线程数可以与dataset的线程数不同，在本接口内会自动调整，用户可以灵活配置dataset的preload线程数、shuffle线程数、数据queue的数目，以及train\_from\_dataset的线程数。  train\_from\_dataset可以非常容易扩展到大规模分布式在线和离线训练。例如可以与Paddle Fleet配合使用，完成千亿或万亿级别大规模稀疏参数的CTR训练，并且性能出色。  train\_from\_dataset函数的输入参数为：  program (Program|CompiledProgram) – 需要执行的program,如果没有给定那么默认使用default\_main\_program (类型是Program)  dataset (paddle.fluid.Dataset) – 在此函数外创建的数据集，用户应当在调用函数前提供完整定义的Dataset。默认为None  scope (Scope) – 执行这个program的域，用户可以指定不同的域。默认为全局域  thread (int) – 用户想要在这个函数中运行的线程数量。默认为0，表示使用传入的dataset的线程数  debug (bool) – 是否开启debug模式，默认为False  fetch\_list (Variable List) – 返回变量列表，每个变量都会在训练过程中被打印出来，默认为None  fetch\_info (String List) – 每个变量的打印信息，默认为None  print\_period (int) – 每两次打印之间间隔的mini-batches的数量，默认为100  Executor. train\_from\_dataset函数没有返回值。 |
| 4. executor.Executor.infer\_from\_dataset | Executor.infer\_from\_dataset函数从预定义的数据集中做预测。 数据集在paddle.fluid.dataset中定义。infer\_from\_dataset的参数与train\_from\_dataset相同，两者的区别是infer\_from\_dataset>没有发送梯度和更新参数。infer\_from\_dataset可以非常容易地用于大规模分布式多线程中的离线评估。infer\_from\_dataset函数的输入参数为：  program (Program|CompiledProgram) – 需要执行的program,如果没有给定那么默认使用default\_main\_program (类型是Program)  dataset (paddle.fluid.Dataset) – 在此函数外创建的数据集，用户应当在调用函数前提供完整定义Dataset。默认为None  scope (Scope) – 执行这个program的域，用户可以指定不同的域。默认为全局域  thread (int) – 用户想要在这个函数中运行的线程数量。线程的实际数量为min(Dataset.thread\_num, thread)，如果thread > 0，默认为0  debug (bool) – 是否开启debug模式，默认为False  fetch\_list (Variable List) – 返回变量列表，每个变量都会在训练过程中被打印出来，默认为None  fetch\_info (String List) – 每个变量的打印信息，默认为None  print\_period (int) – 每两次打印之间间隔的mini-batches的数量，默认为100。  函数的返回值为空。 |
| 5. io.load\_inference\_model | load\_inference\_model从指定文件路径中加载预测模型(Inference Model)，即调用该接口可获得模型结构（Inference Program）和模型参数。若只想加载预训练后的模型参数，请使用 load\_params 接口。输入的参数为：  dirname (str) – 待加载模型的存储路径。  executor (Executor) – 运行 Inference Model 的 executor ，详见 执行引擎 。  model\_filename (str，可选) – 存储Inference Program结构的文件名称。如果设置为None，则使用 \_\_model\_\_ 作为默认的文件名。默认值为None。  params\_filename (str，可选) – 存储所有模型参数的文件名称。当且仅当所有模型参数被保存在一个单独的二进制文件中，它才需要被指定。如果模型参数是存储在各自分离的文件中，设置它的值为None。默认值为None。  pserver\_endpoints (list，可选) – 只有在分布式预测时才需要用到。当训练过程中使用分布式查找表(distributed lookup table)时, 预测时需要指定pserver\_endpoints的值。它是 pserver endpoints 的列表，默认值为None。  load\_inference\_model返回值为包含三个元素的列表，三个元素分别是：  program （Program）– Program （详见 基础概念 ）类的实例。此处它被用于预测，因此可被称为Inference Program。  feed\_target\_names （list）– 字符串列表，包含着Inference Program预测时所需提供数据的所有变量名称（即所有输入变量的名称）。  fetch\_targets （list）– Variable （详见 基础概念 ）类型列表，包含着模型的所有输出变量。通过这些输出变量即可得到模型的预测结果。  该方法首先判定dirname参数指向的路径是否存在，若文件路径不存在，则抛出异常。接着判断model\_filename和param\_filename是否存在，若不存在，则使用默认值来确定最终文件路径。接着加载program, feed\_target\_names, fetch\_targets。 |
| 6. unique\_name.generate | generate方法的输入参数为key，即产生名称的指定前缀。返回值为包含前缀key的唯一命名。generate方法通过UniqueNameGenerator类的实例来完成唯一命名的产生。在generate方法中实例化了一个UniqueNameGenerator类的实例，接着调用该类的\_\_call\_\_函数来产生唯一的命名。 |
| 7. unique\_name.guard | guard方法用于更改命名空间，与with语句一起使用。使用后，在with语句的上下文中使用新的命名空间，调用generate接口时相同前缀的名称将从0开始重新编号。接收的参数为new\_generator，参数类型为字符串，即新命名空间的名称。该函数无返回值。guard方法通常与with上下文控制一同使用，通过with fluid.unique\_name.guard('A'):语句来创建新的命名空间。 |

1. 实验结果呈现与说明（如调试模型的某个中间环节）：

要求：结果截图 + 文字，能证明以上解读的实验不少于3项

|  |  |
| --- | --- |
| **实验结果截图** | **描述说明** |
| 1. fluid.Executor.run | 首先创建了执行器exe，接着定义了数据、网络、优化器和损失函数。接着，我们调用run函数，完成了静态图上的计算，通过fetch\_list得到了loss值。 |
| 2.fluid.Executor.infer\_from\_dataset | 首先创建了执行器exe，执行器通过调用infer\_from\_dataset方法，从预定义的数据集中做预测。 数据集在paddle.fluid.dataset中定义。 |
| 3.fluid.unique\_name.guard | guard方法在with语句的上下文中使用新的命名空间，从结果可以看到，name\_2和name\_1位于两个不同的命名空间。 |

**丁亚东：**

1. 代码总结与梳理：

要求：文字分条目列出，描述不少于200字，模块不少于7个

|  |  |
| --- | --- |
| **代码整体认识描述** | **代码模块或主要组成总结** |
| 包含四个文件   1. loss.py,主要各种loss函数的实现 2. learning\_rate\_scheduler.py 包含各种学习率decay方法的实现   3.metric\_op.py 计算accuracy的实现和Area Under the Curve(AUC)的实现  4.rnn.py rnn网络里面各种网络层的实现 | 1. LearningRateDecay()类所有schedule的基类 |
| 1. PiecewiseDecay()类实现阶梯下降学习率衰减算法 |
| 1. CosineDecay()类实现cosine退火下降算法 |
| 1. center\_loss类，实现   loss的计算，包含各种类型的检查等等 |
| 1. cross\_entropy 类实现交叉熵loss的计算 |
| 1. accuracy方法，计算预测的topk结果和label之间的准确度。 |
| 1. auc 方法，计算Area Under the Curve(AUC)，即ROC曲线和坐标轴的面积 |
| 1. RNNCell 所有rnn变体及rnn cell的基类 |
| 1. Decoder 所有解码器实例的基类 |

1. 关键方法、算法、数据摘取与深入分析（Python或C++）：

要求：方法/公式/数据 + 文字，深入解读代码方法不少于7个

|  |  |
| --- | --- |
| **代码/算法/数据片段** | **输入输出及功能描述分析** |
| 1. class NaturalExpDecay(LearningRateDecay): | 输入: 1.learning\_rate  2. decay\_steps  3. decay\_rate  输出: 无  描述: 实现的是自然对数衰减scheduler。 |
| 2. class ExponentialDecay(LearningRateDecay) | 输入: 1.learning\_rate  2. decay\_steps  3. decay\_rate  输出: 无  描述: 实现的是指数衰减scheduler。 |
| 3. def center\_loss | 输入:1. input   1. label 2. num\_classes   输出:loss  描述:计算input跟label之间的center\_loss |
| 4.def bpr\_loss | 输入:1. Input   1. label 2. name   输出: bpr\_loss  描述:计算input跟lable之间的bpr\_loss |
| 5.def cross\_entroy | 输入:1. Input   1. label   输出: cross\_entroy loss  描述: 计算input和lable之间的交叉熵 |
| 1. def auc | 输入:1.input   1. label 2. curve 曲线的种类   输出:auc  描述:计算ROC曲线或者PR曲线与坐标轴的面积，用于衡量分类器的性能 |
| 7.def accuracy | 输入:1.input  2. label  3. k  输出: accuracy  描述:预测的结果前k位里面有没有label的值，有的话就算预测正确，计算预测的准确率。 |

1. 实验结果呈现与说明（如调试模型的某个中间环节）：

要求：结果截图 + 文字，能证明以上解读的实验不少于3项

|  |  |
| --- | --- |
| **实验结果截图** | **描述说明** |
| 1. | CTC loss的计算结果，输入为产生的随机数 |
| 2. | NaturalExpDecay的实验结果，显示的是学习率的变化 |
| 3. | Piecewise学习率调整的结果 |

**戴光裕：**

1. 代码总结与梳理：

要求：文字分条目列出，描述不少于200字，模块不少于7个

|  |  |
| --- | --- |
| **代码整体认识描述** | **代码模块或主要组成总结** |
| Core.py读取avx或no\_avx的core部分并设置paddle的lib路径。Framework.py对paddle中program这一神经网络配置形式的各个组成部分block，以及组成block的variable，operator等的类进行了定义。 | 1. avx\_supported（），根据是否支持AVX导入avx\_core或noavx\_core. |
| 1. set\_paddle\_lib\_path（）设置paddle的site-packages的路径。 |
| 1. variable类，定义了变量的属性 |
| 1. operator类，定义了运算符的属性 |
| 1. block类，定义了paddle中由多个变量及运算符组成的块的属性。 |
| 1. program类，定义了由多个块组成的program的属性。 |
| 1. parameter类，定义了参数这一随训练迭代的特殊变量的属性。 |

1. 关键方法、算法、数据摘取与深入分析（Python或C++）：

要求：方法/公式/数据 + 文字，深入解读代码方法不少于7个

|  |  |
| --- | --- |
| **代码/算法/数据片段** | **输入输出及功能描述分析** |
| 1. avx\_supported（） | 输入：无，输出：bool，指代当前系统是否支持AVX core，根据其结果选择读取avx\_core或noaxv\_core |
| 2. cuda\_places() cpu\_places()  cuda\_pinned\_places() | 输入：device\_id(cuda\_place) /device\_count （cpu\_place或cuda\_pinned\_place），设定创建CUDAPlace, CPUPlace, CUDAPinnedPlace对象的具体GPU名称(cuda\_place) 或数量（cpu\_place或cuda\_pinned\_place） |
| 3. Class Variable (object) | Paddle的变量类，定义了变量的一系列属性，比如所属的块，类型，名称，数据类型等。 |
| 4.Class Operator (object) | Paddle的操作符类，定义了操作符的一系列属性，比如所属的块，desc描述，类型，输入输出，属性等。 |
| 5.Class Block (object) | Paddle的块类，定义了块的一系列属性，比如所属的程序(program)，包含的变量，包含的操作列表等。 |
| 6.Class program (object) | Paddle的程序类，定义了程序的一系列属性，fluid中的程序替代了传统框架中模型的概念，用程序包含块（block）的形式描述模型 |
| 7.Class parameter | Paddle的参数类，作为变量的子类增加定义了参数的一系列属性，比如是否可训练，优化器属性等。 |
| 8. Class NameScope name\_scope() | 为operators生成不同的命名空间，用于参数调试和可视化。 |
| 9.default\_start\_program()  default\_main\_program() | 生成默认的初始程序和主程序 |
| 10.program\_guard() | 用with语句重写初始程序及主程序，层函数中with的block中的变量和操作符将被添加进主程序 |

1. 实验结果呈现与说明（如调试模型的某个中间环节）：

要求：结果截图 + 文字，能证明以上解读的实验不少于3项

|  |  |
| --- | --- |
| **实验结果截图** | **描述说明** |
| 1. | 按默认主程序的格式创建一个主程序 |
| 2. | 使用with fluid.program\_guard()在主程序中添加变量 |
| 3. | 创建包含一个relu层及x，y数据的起始程序和主程序。输出包含relu层和x，y参数的起始程序和主程序内容。 |

**沈锴：**

1. 代码总结与梳理：

要求：文字分条目列出，描述不少于200字，模块不少于7个

|  |  |
| --- | --- |
| **代码整体认识描述** | **代码模块或主要组成总结** |
| 1. io.py模块定义了一系列关于输入输出的接口。data会在全局block中创建变量，read\_file会从reader中读取数据，从reader生成双缓冲队列，从硬盘中读取数据，创建读取数据的reader。  2. sequence\_lod.py模块定义了一系列sequence计算操作，比如卷积计算函数，池化函数，softmax函数，concat操作，改变维度的reshape，expand操作等等。这一系列函数主要是针对tensor的计算操作。  3. ops.py定义了一系列激活函数，比如thresholded\_relu等等。  4. tensor.py定义了基本数据类型tensor的接口函数。比如create\_tensor, diag等一系列创建tensor的函数，cast, sum等一系列tensor的操作函数。 | 1.load函数 |
| 2．sequence\_conv函数 |
| 3.sequence\_softmax |
| 4.create\_tensor |
| 5.cast |
| 6.concat |
| 7.py\_reader |
| 8.softshrink |
| 9.hard\_shrink |
| 10.read\_file |
| 11.gelu |

1. 关键方法、算法、数据摘取与深入分析（Python或C++）：

要求：方法/公式/数据 + 文字，深入解读代码方法不少于7个

|  |  |
| --- | --- |
| **代码/算法/数据片段** | **输入输出及功能描述分析** |
| 1. sequence\_conv | 该OP在给定的卷积参数下（如卷积核数目、卷积核大小等），对输入的变长序列（sequence）LoDTensor进行卷积操作。默认情况下，该OP会自适应地在每个输入序列的两端等长地填充全0数据，以确保卷积后的序列输出长度和输入长度一致。支持通过配置 padding\_start 参数来指定序列填充的行为。  输入：input (Variable) - 维度为 （M,K) 的二维LoDTensor，仅支持lod\_level为1。其中M是mini-batch的总时间步数，K是输入的 hidden\_size 特征维度。数据类型为float32或float64。  num\_filters (int) - 滤波器的数量。  filter\_size (int) - 滤波器的高度（H）；不支持指定滤波器宽度（W），宽度固定取值为输入的 hidden\_size 。默认值为3。  filter\_stride (int) - 滤波器每次移动的步长。目前只支持取值为1，默认为1。  padding (bool) - 此参数不起任何作用，将在未来的版本中被移除。 无论 padding 取值为False或者True，默认地，该函数会自适应地在每个输入序列的两端等长地填充全0数据，以确保卷积后的输出序列长度和输入长度一致。默认填充是考虑到输入的序列长度可能会小于卷积核大小，这会导致无正确计算卷积输出。填充为0的数据在训练过程中不会被更新。默认为True。  padding\_start (int) - 表示对输入序列填充时的起始位置，可以为负值。负值表示在每个序列的首端填充 |padding\_start| 个时间步（time\_step）的全0数据；正值表示对每个序列跳过前 padding\_start 个时间步的数据。同时在末端填充 filter\_size+padding\_start−1 个时间步的全0数据，以保证卷积输出序列长度和输入长度一致。如果 padding\_start 为None，则在每个序列的两端填充 filter\_size2 个时间步的全0数据；如果 padding\_start 设置为0，则只在序列的末端填充 filter\_size−1 个时间步的全0数据。默认为None。  bias\_attr (ParamAttr) - 指定偏置参数属性的对象。默认值为None，表示使用默认的偏置参数属性。具体用法请参见 ParamAttr 。  param\_attr (ParamAttr) - 指定权重参数属性的对象。默认值为None，表示使用默认的权重参数属性。具体用法请参见 ParamAttr 。  act (str) – 应用于输出上的激活函数，如tanh、softmax、sigmoid，relu等，支持列表请参考 激活函数 ，默认值为None。  name (str，可选) – 具体用法请参见 Name ，一般无需设置，默认值为None。  输出：和输入序列等长的LoDTensor，数据类型和输入一致，为float32或float64 |
| 2. py\_reader | 创建一个在Python端提供数据的reader  该OP返回一个Reader Variable。该Reader提供了 decorate\_paddle\_reader() 和 decorate\_tensor\_provider() 来设置Python generator作为数据源，将数据源中的数据feed到Reader Variable。在c++端调用 Executor::Run() 时，来自generator的数据将被自动读取。与 DataFeeder.feed() 不同，数据读取进程和 Executor::Run() 进程可以使用 py\_reader 并行运行。在每次数据传递开始时调用reader的 start() ，在传递结束和抛出 fluid.core.EOFException 异常后执行 reset() 。  输入：  capacity (int) – py\_reader 维护的缓冲区的容量数据个数。  shapes (list|tuple) – 一个列表或元组，shapes[i]是代表第i个数据shape，因此shape[i]也是元组或列表。  dtypes (list|tuple) – 一个string的列表或元组。为 shapes 对应元素的数据类型，支持bool，float16，float32，float64，int8，int16，int32，int64，uint8。  lod\_levels (list|tuple) – lod\_level的整型列表或元组  name (str，可选) – 具体用法请参见 Name ，一般无需设置，默认值为None。  use\_double\_buffer (bool) – 是否使用双缓冲区，双缓冲区是为了预读下一个batch的数据、异步CPU -> GPU拷贝。默认值为True。  输出：  reader，从reader中可以获取feed的数据，其dtype和feed的数据dtype相同。 |
| 3. load | 该OP操作将从磁盘文件中加载LoDTensor/SelectedRows变量。  输入：out (Variable) - 需要加载的LoDTensor或SelectedRows。  file\_path (str) - 从“file\_path”中加载的变量Variable  load\_as\_fp16 (BOOLEAN) - 如果为真，张量首先进行加载然后类型转换成float16。如果为假，张量将直接加载，不需要进行数据类型转换。默认为false。  输出：None |
| 4. softshrink | Softshrink激活函数    输入：  x (Variable0 - 张量（Tensor）  alpha (float) - 上面公式中alpha的值  输出：  张量(Tensor) |
| 5. create\_tensor | Tensor用于在框架中承载数据，使用 create\_tensor 可以创建一个指定数据类型的Lod-Tensor变量  输入：  dtype (str|numpy.dtype) - 创建的Tensor的数据类型，支持数据类型为bool, float16， float32， float64， int8， int16， int32， int64。  name (str， 可选) - 具体用法请参见 Name ，一般无需设置，默认值为None。  persistable (bool，可选) - 用于设置创建的Tensor的persistable属性，若不设置则默认设置为False。  输出：  创建的Tensor，数据类型为dtype。 |
| 6.cast | Fluid 使用 cast 将数据转换为指定类型。  输入：  x (Variable) - 输入的多维Tensor或LoDTensor，支持的数据类型为：bool、float16、float32、float64、uint8、int32、int64。  dtype (str|np.dtype|core.VarDesc.VarType) - 输出Tensor的数据类型。支持的数据类型为：bool、float16、float32、float64、int8、int32、int64、uint8。  输出：  Tensor或LoDTensor，维度与 x 相同，数据类型为 dtype |
| 7. sequence\_softmax | 该OP根据LoD信息将输入的第0维度进行划分，在划分的每一个区间内部进行运算。  对第i个区间内的元素的计算公式如下：    输入Tensor的维度可为 [N，1] 或者 [N] ，推荐使用 [N] 。  输入：  input (Variable) - 维度为 [N,1] 或者 [N] 的LoDTensor，推荐使用 [N] 。支持的数据类型：float32，float64。  use\_cudnn (bool，可选) - 是否用cudnn核，仅当安装cudnn版本的paddle库且使用gpu训练或推理的时候生效。支持的数据类型：bool型。默认值为False。  name (str，可选) – 具体用法请参见 Name ，一般无需设置，默认值为None。  输出：  根据区间计算softmax之后的LoDTensor，其维度与input的维度一致，数据类型与input的数据类型一致。 |

1. 实验结果呈现与说明（如调试模型的某个中间环节）：

要求：结果截图 + 文字，能证明以上解读的实验不少于3项

|  |  |
| --- | --- |
| **实验结果截图** | **描述说明** |
| 1. sequence\_softmax | 先创建了一个节点x，然后用softmax计算，可以看到，softmax节点被成功创建 |
| 2. create\_tensor | 创建了一个数据类型为32位float浮点的节点tensor  可以看到该节点被成功打印出来 |
| 3. softshrink | 创建了一个名为input，大小为32x784的数据节点data，然后经过softshrink激活函数，得到节点result，可以看到被成功创建 |

**常嘉乐：**

1. 代码总结与梳理：

要求：文字分条目列出，描述不少于200字，模块不少于7个

|  |  |
| --- | --- |
| **代码整体认识描述** | **代码模块或主要组成总结** |
| 1. fluid.regularizer：   有两种正则化方法，L1DecayRegularizer （L1Decay）；L2DecayRegularizer （L2Decay）。  该类生成的实例对象，需要在ParamAttr或者optimizer中进行设置；在ParamAttr中设置时，只对该网络层中的参数生效；在optimizer中设置时，会对所有的参数生效；如果同时设置，在ParamAttr中设置的优先级会高于在optimizer中的设置。 | L1Decay实现L1权重衰减正则化，用于模型训练，使得权重矩阵稀疏。 |
| L2Decay实现L2权重衰减正则化，用于模型训练，有助于防止模型对训练数据过拟合。 |
| 1. fluid.optimizer：   神经网络最终是一个最优化问题，在经过前向计算和反向传播后，Optimizer（一个类）使用反向传播梯度，优化神经网络中的参数。 | **SGD/SGDOptimizer：**是实现随机梯度下降的一个Optimizer的子类，当需要训练大量样本的时候，往往选择SGD来使损失函数更快的收敛。 |
| **Momentum/MomentumOptimizer：**Momentum 优化器在SGD基础上引入动量，减少了随机梯度下降过程中存在的噪声问题。 |
| **Adagrad/AdagradOptimizer：**Adagrad 优化器可以针对不同参数样本数不平均的问题，自适应地为各个参数分配不同的学习率。 |
| **RMSPropOptimizer：**[RMSProp优化器](http://www.cs.toronto.edu/~tijmen/csc321/slides/lecture_slides_lec6.pdf)，是一种自适应调整学习率的方法，主要解决使用Adagrad后，模型训练中后期学习率急剧下降的问题。 |
| 其他优化器不经常使用，此处略过。 |
| 1. fluid. initializer：   fluid通过设置ParamAttr的initializer属性为单个parameter设置初始化方式。  以下为fluid支持的初始化方式： | **BilinearInitializer：**线性初始化方法。用该方法初始化的反卷积操作可当做线性插值操作使用。 |
| **ConstantInitializer：**常数初始化方式，将parameter初始化为指定的数值。 |
| **NormalInitializer：**随机高斯分布初始化方法。 |
| **TruncatedNormalInitializer：**随机截断高斯分布初始化方法。 |
| **UniformInitializer：**随机均匀分布初始化方式。 |
| **MSRAInitializer：**是一个鲁棒性特别强的初始化方法，适应非线性激活函数。 |
| **XavierInitializer：**实现了Xavier权重初始化方法，该函数用于保持所有层的梯度尺度几乎一致。 |
| 1. fluid. compiler：   CompiledProgram根据*build\_strategy*的配置将输入的Program或Graph进行转换和优化，例如：计算图中算子融合、计算图执行过程中开启内存、显存优化等。  我们还可以使用with\_data\_parallel将程序转化为数据并行程序，使其能够运行在多个设备上。该接口⽤于将输⼊的Program或Graph进⾏转换，以便通过数据并⾏模式运⾏该模型。如果在构建CompiledProgram和调⽤with\_data\_parallel时都指定了build\_strategy，那么，在CompiledProgram中的build\_strategy将会被复写，因此，如果是数据并行训练，建议在调用with\_data\_parallel接口时设置build\_strategy。  **注：**该API仅仅支持静态图模式 | **CompiledProgram：**编译成一个用来执行的Graph。首先使用layers(网络层)创建程序，其次可以使用CompiledProgram来在运行之前优化程序（可选），最后定义的程序或CompiledProgram由Executor运行。 |
| **BuildStrategy：**其使用户更精准地控制ParallelExecutor中SSA图的建造方法。可通过设置ParallelExecutor中的BuildStrategy成员来实现此功能。 |
| **ExecutionStrategy：**其允许用户更加精准地控制program在ParallelExecutor中的运行方式。可以通过在ParallelExecutor中设置本成员来实现。  （用于选择执行图的方式，例如使用多少线程，每次清理临时变量之前进行的迭代次数）。 |
| **ParallelExecutor：**其专门设计用来实现数据并行计算，着力于向不同结点(node)分配数据，并行地在不同结点中对数据进行操作。如果在GPU上使用该类运行程序，node则用来指代GPU，ParallelExecutor也将自动获取在当前机器上可用的GPU资源。如果在CPU上进行操作，node则指代CPU，同时你也可以通过添加环境变量CPU\_NUM来设置CPU设备的个数。但是如果没有设置该环境变量，该类会调用multiprocessing.cpu\_count来获取当前系统中CPU的个数。 |

1. 关键方法、算法、数据摘取与深入分析（Python或C++）：

要求：方法/公式/数据 + 文字，深入解读代码方法不少于7个

|  |  |
| --- | --- |
| **代码/算法/数据片段** | **输入输出及功能描述分析** |
| L1权重衰减正则化的计算公式如下： | L1正则化对梯度的影响不再是线性地缩放每个参数;而是添加了一项与sign(参数)同号的常数，由于超参数是一个大于0的数，因此L1正则化相对于未正则化的模型而言，每步更新后权重向量向0靠拢。 |
|  | 参数program\_or\_graph的意思是：若其为Program，那么它将首先被降成一个graph，以便进一步优化。如果它是一个graph（以前可能优化过），它将直接用于进一步的优化。注：只有使用with\_data\_parallel选项编译时才支持graph。 |
|  | 如果设置了share\_vars\_from，当前的CompiledProgram将与share\_vars\_from指定的CompiledProgram共享参数值。需要设置该参数的情况：**模型训练过程中需要进行模型测试，并且训练和测试都是采用数据并行模式**，那么测试对应的CompiledProgram在调用with\_data\_parallel时，需要将share\_vars\_from设置为训练对应的CompiledProgram。由于CompiledProgram只有在第一次执行时才会将变量分发到其他设备上，因此share\_vars\_from指定的CompiledProgram必须在当前CompiledProgram之前运行。默认为：None |
|  | int型成员，它表明了清空执行时产生的临时变量需要的程序执行迭代次数。因为临时变量的形状可能在两次重复过程中保持一致，所以它会使整体执行过程更快，默认值为1。  （如果在调用.run方法时获取结果数据，ParallelExecutor 会在当前程序重复执行尾部清空临时变量）  （在一些NLP模型里，该成员会致使GPU内存不足。此时，应减少num\_iteration\_per\_drop\_scope的值） |
|  | **num\_trainers (int) ：**如果该值大于1， NCCL将会通过多层级node的方式来初始化。每个node应有相同的GPU数目。 随之会启用分布式训练。该参数默认为1。  **trainer\_id (int) ：**必须与 num\_trainers 参数同时使用。trainer\_id 是当前所在node的“rank”（层级），从0开始计数，该参数默认为0。  使用fetch\_list执行一个 ParallelExecutor 对象。  如果是单卡训练，feed为dict类型，如果是多卡训练，参数feed可以是dict或者list类型变量，如果该参数是dict类型，feed中的数据将会被分割(split)并分送给多个设备（CPU/GPU）。反之，如果它是list，则列表中的各个元素都会直接分别被拷贝到各设备中。如果样本数少于设备数，程序会报错，因此运行模型时，应额外注意数据集的**最后一个batch的样本数**是否少于当前可用的CPU核数或GPU卡数，如果少于，建议丢弃该batch。  如果可用的CPU核数或GPU卡数大于1，则为每个变量最后获取的结果都是list类型，且这个list中的每个元素都是各CPU核或GPU卡上的变量，fetch出来的结果为不同设备上的相同变量值（fetch\_list中的变量）在第0维拼接在一起。 |
|  | 该两类生成的实例对象，需要设置在[ParamAttr](https://www.paddlepaddle.org.cn/documentation/docs/zh/develop/api_cn/fluid_cn/ParamAttr_cn.html#cn-api-fluid-paramattr)或者optimizer中，在ParamAttr中设置时，只对该网络层中的参数生效；在optimizer中设置时，会对所有的参数生效；如果同时设置，在ParamAttr中设置的优先级会高于在optimizer中的设置。 |
|  | 加入“动量”的优化器，参数更新的方向更稳定，比如[Momentum](https://www.paddlepaddle.org.cn/documentation/docs/zh/api_cn/optimizer_cn/MomentumOptimizer_cn.html#cn-api-fluid-optimizer-momentumoptimizer)优化器。每个批次的数据含有抽样误差，这样会导致梯度更新的方向波动较大。如果我们引入物理动量的概念，给梯度下降的过程加入一定的“惯性”累积，就可以减少更新路径上的震荡，即每次更新的梯度由“历史多次梯度的累积方向”和“当次梯度”加权相加得到。历史多次梯度的累积方向往往是从全局视角更正确的方向，这与“惯性”的物理概念很像，也是为何其起名为“Momentum”的原因。  根据不同参数距离最优解的远近，动态调整学习率，比如[AdaGrad](https://www.paddlepaddle.org.cn/documentation/docs/zh/api_cn/optimizer_cn/AdagradOptimizer_cn.html#cn-api-fluid-optimizer-adagradoptimizer)优化器。通过调整学习率的实验可以发现：当某个参数的现值距离最优解较远时（表现为梯度的绝对值较大），我们期望参数更新的步长大一些，以便更快收敛到最优解。当某个参数的现值距离最优解较近时（表现为梯度的绝对值较小），我们期望参数的更新步长小一些，以便更精细的逼近最优解。参数更新的步长应该随着优化过程逐渐减少，减少的程度与当前梯度的大小有关。根据这个思想编写的优化算法称为“AdaGrad”，Ada是Adaptive的缩写，表示“适应环境而变化”的意思。  因为上述两个优化思路是正交的，所以可以将两个思路结合起来，这就是当前广泛应用的[Adam](https://www.paddlepaddle.org.cn/documentation/docs/zh/api_cn/optimizer_cn/AdamOptimizer_cn.html#cn-api-fluid-optimizer-adamoptimizer)算法。 |

1. 实验结果呈现与说明（如调试模型的某个中间环节）：

要求：结果截图 + 文字，能证明以上解读的实验不少于3项

|  |  |
| --- | --- |
| **实验结果截图** | **描述说明** |
|  | 用户还可以使用 program\_guard 配合 with 语句，修改配置好的 default\_startup\_program 和 default\_main\_program。 |
|  | 如果feed参数为dict类型，那么传入 ParallelExecutor 的数据量必须大于可用的CPU核数或GPU卡数。否则，C++端将会抛出异常。应额外注意核对数据集的最后一个batch是否比可用的CPU核数或GPU卡数大。 |
|  | 使⽤相同shape的np.ndarray对paddle变量初始化。 |
|  | Adaptive Gradient 优化器(自适应梯度优化器，简称Adagrad)可以针对不同参数样本数不平均的问题，自适应地为各个参数分配不同的学习率。 |

4.疑惑问题点整理：

| 问题点： |
| --- |
| 1.fluid.initializer中有两个模块——force\_init\_on\_cpu（该接口获得一个是否强制在CPU上初始化变量的布尔型标志位）；init\_on\_cpu（该接口设置强制变量在CPU上初始化）。  不太清楚这两个函数设置的实际意义。 |
| 2.PaddlePaddle Fluid中有两种执行器。一种是Executor默认的单线程执行器，另一种是并行计算执行器——[数据并行执行引擎](https://www.paddlepaddle.org.cn/documentation/docs/zh/develop/api_guides/low_level/parallel_executor.html#api-guide-parallel-executor)。Executor和[数据并行执行引擎](https://www.paddlepaddle.org.cn/documentation/docs/zh/develop/api_guides/low_level/parallel_executor.html#api-guide-parallel-executor)的配置是不同的。不太清楚分开两个执行器的原因，合并成一个他不香吗？ |
| 3．现有的CompiledProgram仅仅支持静态图，不支持动态图的原因是否为——动态图在运行中计算资源以及操作都是在动态变化中，不好进行分布式训练呢？有没有好解决的方案？ |
| 4．如果初始化方法将权值参数全部初始化为0或者一个常数，可能会带来的问题是导致网络中所有的神经元学习到的是相同的特征，这个方法的意义何在？force\_cpu (bool)用于表示初始化的位置，为True时，会强制在CPU上初始化（即使executor设置的是CUDA），两种初始化有什么区别？ |
| 5. ops中只有几个函数的定义，找不到tanh等激活函数的定义，但却在\_\_all\_\_中，是否定义在其他位置 |
| 6. 感觉静态图调试存在困难，可有针对静态图的调试工具。 |
| 7. 感觉集成度有点高，嵌套的很复杂。 |
| 8. 不太理解OpProtoHolder类的功能和作用 |
| 9. Start\_up\_program和main\_program的区别与联系 |
| 10. dynamic\_lstmp相比较dynamic\_lstm的提升在哪里 |
| 11. nce loss在哪些情况下面适用 |
| 12. ParamAttr类用于保存网络的部分参数属性，有一些属性不包含在内，不能在初始化时通过网络的param\_attr参数传入，这里哪些参数可以由ParamAttr指定，哪些需要额外指定，设计的标准是什么？ |
| 13.fluid.unique\_name.generator方法的实现为什么需要先设计一个UniqueNameGenerator类，再实例化一个全局的实例，最终在该方法中调用这个实例的方法。这样的设计感觉过于冗杂。 |
| 14. 静态图调试困难，是否有针对静态图进行调试的工具？ |

5.可改进功能或可开展工作整理：

要求：文字分条目列出，可改进功能功能不少于2点

| **可改进的功能** | **拟开展工作** |
| --- | --- |
| CompiledProgram用于转换程序以进行各种优化的方法有：预先计算一些逻辑，以便每次运行更快；转换Program，使其可以在多个设备中运行；转换Program以进行优化预测或分布式训练（此部分尚未完成）。 | 了解转换Program分布式训练的难点 |
| 能否开发针对静态图的调试工具。 | 尝试总结静态图调试方法。 |
| 跟进最新算法的benchmark。 | 可以实现一些benchmark供开发者使用。 |
| 可以考虑在core.py中标注一下导入的avx\_core或noavx\_core的功能有哪些 | 对avx\_core和noavx\_core的动态库进行一些分析 |
| 对于动态图和静态图模式的不同功能可以在framework中进行更多的区分 | 从顶层设计上了解paddle的动态图和静态图模式的区别 |
| rnn.py类里面的层次结构不是很清晰 | 进一步梳理类之间的关系，了解同样实现类型的类在不同条件下的使用 |
| digraph mode下面有些类是不可以使用的 | 了解某些类比如dynamic\_lstm等等的兼容性 |
| fluid.unique\_name.generator | 对其实现方式进行修改 |
| 静态图调试 | 调研静态图调试相关的工具，尝试加入paddle中 |

6.可参考其他技术资料及文献：

要求：网页链接或文件整理，参考外部资料不少于3处

| 可参考地址 | 材料设计内容简述 |
| --- | --- |
| <https://www.paddlepaddle.org.cn/documentation/docs/zh/api_cn/index_cn.html> | 对paddle各个模块进行介绍 |
| <https://github.com/fengdu78/PaddlePaddle-Tutorial> | 对PaddlePaddle相关资源进行了整理与汇总。 |
| <https://www.jikexueyuan.com/course/padlle/> | Paddle学习 |
| <https://www.zhihu.com/question/54356960> | Seq2seq里面的beam search的具体过程。 |
| <https://baike.baidu.com/item/AUC/19282953> | AUC和ROC的具体含义及计算 |
| <https://zhuanlan.zhihu.com/p/31841042> | BPR loss的具体含义以及计算过程 |
| <https://www.zhihu.com/question/50043438> | Nce loss的直观解释 |
| <https://www.paddlepaddle.org.cn/documentation/docs/zh/api_guides/low_level/program.html> | 关于paddlepaddle的program概念及其下层的block，operator，variable的编程思想 |
| <https://www.jianshu.com/p/1707d2a75f75> | 有关program，name\_scope的一些使用方法 |
| <https://zhuanlan.zhihu.com/p/108402148> | Paddle编程方法的一些基础思想和例子 |
| <https://blog.csdn.net/xs11222211/article/details/82931120>  <https://blog.csdn.net/xs11222211/article/details/84262491>  https://zhuanlan.zhihu.com/p/129912419 | 简单的深度学习分布式训练理论知识和实战 |