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** |
| layers.nn主要作为网络搭建的工具箱，内部函数主要包括\_elementwise\_op和\_logical\_op，是对矩阵元素的操作;随机操作包括random和sample系列;数据处理包括图片序列化、onehot、pad和crop等操作;网络结构包括fc、embedding、conv、norm、pool等操作;张量操作分为LoD特有操作和普通张量操作 | 1. layers.nn |
| 实现了各种学习率的衰减方法; 实现了精确率accuray以及AUC (Area Under the Curve)的计算; 实现了常见损失函数，包括交叉熵、均方误差等等; 实现了RNN及其变种(GRU,LSTM)的基本cell，更高级的序列封装，解码器封装，动态的解码器封装，以及beam search方法封装 | 1. learning\_rate\_scheduler.py 2. metric\_op.py 3. loss.py 4. rnn.py |
| python/paddle/fluid/layers/io.py  这个模块主要定义了创建Variable以及输入输出相关函数 | 6.paddle.fluid.layers.data (deprecated)  改为使用 paddle.fluid.data  创建一个全局变量(Variable)， 这个Variable在整个计算图的任何位置都能获取到，这个Variable是一个placeholder, 在计算的时候才传入数据。  (1) 编译时不检查tensor的shape和数据类型，runtime的时候才检查  (2)创建tesnor的时候shape可以不确定, 用None或者-1，当计算的时候才确定shape  (3) 本API创建的变量默认 stop\_gradient 属性为true, 反向传播的时候梯度不会传递经过这个变量  (4)可以指定lod\_level并创建LoDTensor |
| 7. fluid.layers.read\_file  fluid.layers.py\_reader  read\_file 用于从文件中读取数据，py\_reader用于创建一个reader Variable  例如：  import paddle.fluid as fluid  reader = fluid.layers.py\_reader(capacity=64,  shapes=[(-1, 1, 28, 28), (-1, 1)], dtypes=['float32', 'int64'])  image, label = fluid.layers.read\_file(reader) |
| 8. fluid.layers.create\_py\_reader\_by\_data  用于直接从数据创建一个 reader Variable，例如  image=fluid.data(name='image',shape=[None,1,28,28],dtype='float32')  label = fluid.data(name='label', shape=[None, 1], dtype='int64')  reader= fluid.layers.create\_py\_reader\_by\_data(capacity=64,  feed\_list=[image, label]) |
| 9. fluid.layers.load  将文件中的数据加载到一个tensor中，例如：  tmp\_tensor = fluid.layers.create\_tensor(dtype='float32')  fluid.layers.load(tmp\_tensor, "./tmp\_tensor.bin") |
| paddle.fluid.executor / paddle.fluid.param\_attr / paddle.fluid.unique\_name / paddle.fluid.io 均为 paddle 执行引擎相关接口代码，包括程序执行、参数变量描述定义、参数名生成、程序IO的相关内容。 | 1. paddle.fluid.executor   该模块包含 Paddle 程序执行类 `Executor`，封装了程序执行环境、CPU/GPU 设备、变量表、模型程序内容。`Executor.run` 方法为执行 Paddle 程序的入口，内部调用 `Executor.\_run\_impl`，根据执行模式进一步调用内部的不同实现。  `Executor.run` 方法为执行 Paddle 程序的入口，实际执行的是 `Executor.\_run\_impl`。该方法执行一段程序时会首先查找是否存在优化后的程序的缓存，否则会调用 `Executor.\_prune\_program`，根据`fetch\_list`参数确定程序实际需要执行的部分并去除多余的部分以免去不必要的计算。之后根据程序是否是编译后的程序使用不同的内部方法执行程序。  未编译的程序代表训练的场景，由 `Executor.\_run\_program` 调用 `default\_executor` 的 `run` 方法，该方法在 `paddle.fluid.core` 模块中实现。编译后的程序如果为 `\_is\_inference` 则直接调用编译后程序的 `run` 方法执行计算；否则进入 `Executor.\_run\_parallel` 。 |
| 1. paddle.fluid.param\_attr   该模块主要实现神经网络中的参数描述，包括 name, initializer, learning rate, regularizer, trainable, gradient clip, 及 model average ，由 `ParamAttr` 类存储。 |
| 1. paddle.fluid.unique\_name   该模块用于给神经网络定义中的不同层、参数等生成唯一的 name，添加 `\_0` `\_1` 的后缀，避免出现重名的参数或层定义。 |
| 1. paddle.fluid.io   该模块实现保存及读取神经网络定义及参数变量。主要入口为 `save\_vars` 、 `save\_params`、`load\_vars`、`load\_params`、`save\_inference\_model` 等。在内部区分本地和分布式的各种情况，调用相应的实现。 |
| Framework：主要完成对代码中paddle框架整体情况的监测，例如：监测paddle版本、是否使用动态图、cpu和cuda使用情况、变量赋值和复制等基础操作。Operator：所有操作都由Operator表示，并且Operator被视为Block指令中的构建。  Core：确定环境是否匹配。  \_\_init\_\_：对实例的属性进行初使化。  bootstrap：从环境变量中读取gflag。  Parameter：参数是从变量派生的。 参数是一个可持久变量，在每次迭代后都会由优化程序进行更新。神经网络的训练本质上是其参数的更新。  Program：创建python程序。  IrGraph：创建一个变量和操作可视化的，基于c++的图。 | 1. require\_version(min\_version, max\_version=None)   Operator类：\_has\_kernel()、\_to\_readable\_code()等。  avx\_supported()  \_\_bootstrap\_\_()  Parameter类:\_\_str\_\_()、to\_string()等。  Program类: global\_seed()、\_op\_role()等。  IrGraph类: clone()、is\_test()等。 |
|  | 1. fluid.compiler, 根据构建策略转化fluid.Program或fluid.Graph, 如操作融合显存优化等   fluid.initializer, 可训练参数的初始化操作, 如卷积, 反卷积的初始化  fluid.regularizer, 模型正则化, 即模型的结构奉献  fluid.optimizer, 参数优化算法, SGD, 自适应梯度, 自适应学习率算法 |

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.  **def** fc(input,  size,  num\_flatten\_dims=1,  param\_attr=**None**,  bias\_attr=**None**,  act=**None**,  name=**None**):  helper = LayerHelper(**"fc"**, \*\*locals()) check\_type(input, **'input'**, (list, tuple, Variable), **'fc'**) **if** isinstance(input, (list, tuple)):  **for** i, input\_x **in** enumerate(input):  check\_type(input\_x, **'input['** + str(i) + **']'**, Variable, **'fc'**) dtype = helper.input\_dtype() check\_dtype(dtype, **'input'**, [**'float16'**, **'float32'**, **'float64'**], **'fc'**) mul\_results = [] **for** input\_var, param\_attr **in** helper.iter\_inputs\_and\_params():  input\_shape = input\_var.shape  param\_shape = [  reduce(**lambda** a, b: a \* b, input\_shape[num\_flatten\_dims:], 1)  ] + [size]   w = helper.create\_parameter(  attr=param\_attr, shape=param\_shape, dtype=dtype, is\_bias=**False**)  tmp = helper.create\_variable\_for\_type\_inference(dtype)  helper.append\_op(  type=**"mul"**,  inputs={**"X"**: input\_var,  **"Y"**: w},  outputs={**"Out"**: tmp},  attrs={**"x\_num\_col\_dims"**: num\_flatten\_dims,  **"y\_num\_col\_dims"**: 1})  mul\_results.append(tmp)  **if** len(mul\_results) == 1:  pre\_bias = mul\_results[0] **else**:  pre\_bias = helper.create\_variable\_for\_type\_inference(dtype)  helper.append\_op(  type=**"sum"**,  inputs={**"X"**: mul\_results},  outputs={**"Out"**: pre\_bias},  attrs={**"use\_mkldnn"**: **False**}) *# add bias* pre\_activation = helper.append\_bias\_op(pre\_bias, dim\_start=num\_flatten\_dims) *# add activation* **return** helper.append\_activation(pre\_activation) | fc是layers.nn中最基础的操作，但是他的用法和pytorch和tensorflow还有些区别，首先是它支持list的Variable，对他们进行操作时对list中每个元素fc后再求和；然后是fc支持paddle特有的LoD tensor，他是对LoD张量中numpy的数据进行FC再分割；最后num\_flatten\_dimsz支持输入矩阵是高维的时候，按照指定的维度进行二维的变换  综上FC的输入包括input张量、size表示输出维度、num\_flatten\_dims表示分割的维度、param\_attr和bias\_attr表示可以传递预先训练的W和b全权值和偏置,act为定义激活函数,name是命名  内部实现主要是通过调用LayerHelper这个封装类进行实现，检查类型、当有多个输入的时候迭代、每个输入进行操作的创建包括W的乘法和b的加法 |
| 2.  **1.paddle.fluid.layers.exponential\_decay**  (learning\_rate,decay\_steps,decay\_rate,staircase=False)  learning\_rate (Variable|float) - 初始学习率，类型可以为学习率变量(Variable)或float型常量。  decay\_steps (int) - 学习率衰减步长，见以上衰减运算。  decay\_rate (float) - 学习率衰减率。见以上衰减运算。  staircase (bool) - 若为True，按离散区间衰减学习率，即每 decay\_steps 步学习率衰减 decay\_rate 。若为False，则按以上衰减运算持续衰减。默认False。  **2.paddle.fluid.layers.cross\_entropy**  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。  硬标签交叉熵算法：若soft\_label = False, label[i1,i2,...,ik] 表示每个样本的硬标签值:  output[i1,i2,...,ik]=−log(input[i1,i2,...,ik,j]),label[i1,i2,...,ik]=j,j!=ignore\_index  软标签交叉熵算法：若soft\_label = True, label[i1,i2,...,ik,j] 表明每个样本对应类别j的软标签值:  output[i1,i2,...,ik]=−∑jlabel[i1,i2,...,ik,j]∗log(input[i1,i2,...,ik,j])  input (Variable) – 维度为 [N1,N2,...,Nk,D] 的多维Tensor，其中最后一维D是类别数目。数据类型为float32或float64。  label (Variable)–输入input对应的标签值。若soft\_label=False，要求label维度为 [N1,N2,...,Nk] 或 [N1,N2,...,Nk,1] ，数据类型为int64，且值必须大于等于0且小于D；若soft\_label=True，要求label的维度、数据类型与input相同，且每个样本各软标签的总和为1。  soft\_label (bool) – 指明label是否为软标签。默认为False，表示label为硬标签；若soft\_label=True则表示软标签。  ignore\_index (int) – 指定一个忽略的标签值，此标签值不参与计算，负值表示无需忽略任何标签值。仅在soft\_label=False时有效。 默认值为-100。  返回： 表示交叉熵结果的Tensor，数据类型与input相同。若soft\_label=False，则返回值维度与label维度相同；若soft\_label=True，则返回值维度为 [N1,N2,...,Nk,1] 。  返回类型：Variable  **3.paddle.fluid.layers.lstm(input,init\_h,init\_c,max\_len,hidden\_size,num\_layers, dropout\_prob=0.0,is\_bidirec=False, is\_test=False, name=None, default\_initializer=None, seed=-1)**  参数：  input ( Variable ) - LSTM的输入张量，维度为 [batch\_size,seq\_len,input\_dim] 的 3-D Tensor，其中 seq\_len 为序列的长度， input\_dim 为序列词嵌入的维度。数据类型为 float32 或者 float64。  init\_h ( Variable ) – LSTM的初始 hidden 状态，维度为 [num\_layers,batch\_size,hidden\_size] 的 3-D Tensor，其中 num\_layers 是LSTM的总层数，hidden\_size 是隐层维度。 如果is\_bidirec = True， 维度应该为 [num\_layers∗2,batch\_size,hidden\_size] 。数据类型为 float32 或者 float64。  init\_c ( Variable ) - LSTM的初始 cell 状态。维度为 [num\_layers,batch\_size,hidden\_size] 的 3-D Tensor，其中 num\_layers 是LSTM的总层数，hidden\_size 是隐层维度。 如果is\_bidirec = True， 维度应该为 [num\_layers∗2,batch\_size,hidden\_size] 。数据类型为 float32 或者 float64。  max\_len (int) – LSTM的最大长度。输入张量的第一个 input\_dim 不能大于 max\_len。  hidden\_size (int) - LSTM hidden 状态的维度。  num\_layers (int) – LSTM的总层数。例如，该参数设置为2，则会堆叠两个LSTM，其第一个LSTM的输出会作为第二个LSTM的输入。  dropout\_prob (float，可选) – dropout比例，dropout 只在 rnn 层之间工作，而不是在时间步骤之间。dropout 不作用于最后的 rnn 层的 rnn 输出中。默认值为 0.0。  is\_bidirec (bool，可选) – 是否是双向的LSTM。默认值为 False。  is\_test (bool，可选) – 是否在测试阶段。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。  default\_initializer (Initializer，可选) – 用于初始化权重的初始化器，如果为None，将进行默认初始化。默认值为 None。  seed (int，可选) – LSTM中dropout的seed，如果是-1，dropout将使用随机seed。默认值为 1。  返回： 经过lstm运算输出的三个Tensor的tuple，包括  rnn\_out：LSTM hidden的输出结果的Tensor，数据类型与input一致，维度为 [batch\_size,seq\_len,hidden\_size] 。如果 is\_bidirec 设置为True，则维度为 [batch\_size,seq\_len,hidden\_size∗2]  last\_h：LSTM最后一步的hidden状态的Tensor，数据类型与input一致，维度为 [num\_layers,batch\_size,hidden\_size] 。如果 is\_bidirec 设置为True，则维度为 [num\_layers∗2,batch\_size,hidden\_size]  last\_c：LSTM最后一步的cell状态的Tensor，数据类型与input一致，维度为 [num\_layers,batch\_size,hidden\_size] 。如果 is\_bidirec 设置为True，则维度为 [num\_layers∗2,batch\_size,hidden\_size] |  |
| 1. fluid.layers.sequence\_concat   fluid.layers.sequence\_first\_step  fluid.layers.sequence\_slice | fluid.layers. sequence\_concat将输入的LodTensor级联起来。  输入：一个包含多个LodTensor的list  输出：级联后的LodTensor  LodTensor用于储存变长序列，在此假设我们储存的是包含不同单词数的句子  例如：input=[x1,x2]  x1.lod = [[0, 3, 5]]#偏移表示  # 这表示index=0,1,2的单词为一个句子，index=3,4的单词为一个句子  x1.data = [[1], [2], [3], [4], [5]]  x1.shape = [5, 1]  # x1.shape = [5, 1]表示x1中共有5个单词，每个单词的embedding vector的维度为1  x2.lod = [[0, 2, 4]] #偏移表示  x2.data = [[6], [7], [8], [9]]  x2.shape = [4, 1]  并且要满足 len(x1.lod[0]) == len(x2.lod[0])  则级联后的输出为：  out.lod = [[0, 3+2, 5+4]]  out.data = [[1], [2], [3], [6], [7], [4], [5], [8], [9]]  out.shape = [9, 1]  fluid.layers.sequence\_first\_step 获取输入的LodTensor中每个sequence的第一个/最后一个 time-step  输入：LoDTensor , 其lod\_level ≤ 2. dtype为 float32  输出：一个Tensor，包含输入的LodTensor的每个sequence的第一个embedding vector。(1)当输入LoDTensor的lod\_level==1时，输出为一个普通tensor  (2)当输入的LoDTensor的lod\_level==2时，输出为一个lod\_level=1的LoDTensor。  Case1：输入为1-level LoDTensor:  input.lod = [[0, 2, 5, 7]]  input.data = [[1.], [3.], # index=0,1  [2.], [4.], [6.], # index=2,3,4  [5.], [1.]] # index= 5,6  input.shape = [7, 1]  output 是一个普通tensor  out.shape = [3, 1]  out.shape[0] == len(x.lod[-1]) -1 == 3  out.data = [[1.], [2.], [5.]],  where 1.=first(1., 3.), 2.=first(2., 4., 6.), 5.=first(5., 1.)  Case 2:  输入是一个 2-level LoDTensor  input.lod = [[0, 2, 2, 5], [0, 1, 3, 4, 4, 7]]  input.data = [[1.], #index=0  [3.], [2.], #index=1,2  [4.], #index=3  # empty sequence  [6.], [5.], [1.]] # index=4,5,6    input.shape = [7, 1]  会对最后一个lod\_level [0, 1, 3, 4, 4, 7]. 进行提取first time-step的操作，其中对于empty sequence会padding, pad\_value = 0.0  Output 是一个lod\_level=1的LoDTensor  out.shape= [5, 1]  out.lod = [[0, 2, 2, 5]]  out.shape[0] == len(x.lod[-1]) -1== 5  out.data = [[1.], [3.], [4.], [0.0], [6.]]  where 1.=first(1.), 3.=first(3., 2.), 4.=first(4.), 0.0 = pad\_value, 6.=first(6., 5., 1.) |
| fluid.layers.sequence\_slice 将LodTensor切片  输入：LodTensor， offset(切片起始位置), length(切片长度)  输出：切片后的LodTensor  例如：  input.data = [[a1, a2], [b1, b2], [c1, c2],  [d1, d2], [e1, e2]],  input.lod = [[0,3, 5]],  input.dims = (5, 2), # 有5个单词，每个单词的embedding vector维度==2  offset.data = [[0], [1]]  length.data = [[2], [1]],  out.data = [[a1, a2], [b1, b2],  [e1, e2]],  out.lod = [[0,2, 3]],  out.dims = (3, 2).  # 3==2+1 == length.data[0][0]+ length.data[1][0] |
| 1. if not isinstance(min\_version, str):  raise TypeError(  "The type of 'min\_version' in require\_version must be str, but received %s."  % (type(min\_version))) | 函数require\_version(min\_version,max\_version=None)  中的语句，用来判断第一个参数的类型是否为string。 |
| 2. def child(self, prefix):  if prefix not in self.\_children:  new\_child = NameScope(prefix, self)  self.\_children[prefix] = [new\_child]  else:  new\_child = NameScope(prefix + "\_%d" % len(self.\_children[prefix]),  self)  self.\_children[prefix].append(new\_child)  return new\_child | Namescope类中，当一前缀不在字典中时，创建该前缀的字典对。在时，通过len（）函数判断该操作为此前缀的第几个操作。 |
| 3. def is\_compiled\_with\_cuda():  return core.is\_compiled\_with\_cuda() | 判断是否可以使用gpu运行paddle代码。 |
| 4. dtype = np.dtype(np\_dtype)  if dtype == np.float32:  return core.VarDesc.VarType.FP32  elif dtype == np.float64:  return core.VarDesc.VarType.FP64  elif dtype == np.float16:  return core.VarDesc.VarType.FP16  elif dtype == np.int32:  return core.VarDesc.VarType.INT32  elif dtype == np.int16:  return core.VarDesc.VarType.INT16  elif dtype == np.int64:  return core.VarDesc.VarType.INT64  elif dtype == np.bool:  return core.VarDesc.VarType.BOOL  elif dtype == np.uint16:  return core.VarDesc.VarType.INT16  elif dtype == np.uint8:  return core.VarDesc.VarType.UINT8  elif dtype == np.int8:  return core.VarDesc.VarType.INT8  else:  raise ValueError("Not supported numpy dtype %s" % dtype) | 将numpy中的数据类型转换为paddle中的数据类型，属于convert\_np\_dtype\_to\_dtype\_(np\_dtype)  函数。 |
| 5. import paddle.fluid as fluid  with fluid.name\_scope("s1"):  a = fluid.data(name='data', shape=[None, 1], dtype='int32')  b = a + 1  with fluid.name\_scope("s2"):  c = b \* 1  with fluid.name\_scope("s3"):  d = c / 1  with fluid.name\_scope("s1"):  f = fluid.layers.pow(d, 2.0)  with fluid.name\_scope("s4"):  g = f - 1  for op in fluid.default\_main\_program().block(0).ops:  print(op.type)  print(op.desc.attr("op\_namescope")) | 尝试对操作进行分级命名 |
| 6. parameters = []  for each\_block in self.blocks:  parameters.extend(each\_block.all\_parameters())  return parameters | 获得程序中的所有参数。 |
| 7. if program is None:  program = default\_main\_program()  assert isinstance(name, str)  assert isinstance(program, Program)  return program.global\_block().var(name) | 从程序的全局块中按名称获取变量。 |

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. **import** paddle.fluid **as** fluid **import** numpy **as** np  a = fluid.create\_lod\_tensor(np.random.rand(15,6).astype(np.float32) ,  [[3,1,2] , [3,2,4,1,2,3]],  fluid.CPUPlace()) *# print(a.shape) # when input are multiple tensors* data\_1 = fluid.data(name=**"data\_1"**, shape=[15,6], dtype=**"float32"**) *# data\_2 = fluid.data(name="data\_2", shape=[-1, 36], dtype="float32")* fc = fluid.layers.fc(input=data\_1, size=20, act=**"tanh"**,num\_flatten\_dims=1)  cpu = fluid.core.CPUPlace() exe = fluid.Executor(cpu)  exe.run(fluid.default\_startup\_program())  *# x = np.random.randn(3,5,8).astype("float32") # y= np.random.randn(5,36).astype("float32")* outs = exe.run(  feed={**'data\_1'**:a},  fetch\_list=[fc],return\_numpy=**False**)  *#输出执行结果* print(np.array(outs[0])) *#输出 result 的 LoD* print(outs[0].lod()) |  |
| 2.  data\_1 = fluid.data(name=**"data\_1"**, shape=[15,6], dtype=**"float32"**) data\_2 = fluid.data(name=**"data\_2"**, shape=[15, 6], dtype=**"float32"**) fc = fluid.layers.fc(input=[data\_1,data\_2], size=4, act=**"tanh"**,num\_flatten\_dims=1)  cpu = fluid.core.CPUPlace() exe = fluid.Executor(cpu)  exe.run(fluid.default\_startup\_program())  a = np.random.randn(15,6).astype(**"float32"**) b= np.random.randn(15,6).astype(**"float32"**)  outs = exe.run(  feed={**'data\_1'**:a,**'data\_2'**:b},  fetch\_list=[fc],return\_numpy=**False**) print(outs[0]) |  |
| 3.  data\_1 = fluid.data(name=**"data\_1"**, shape=[3,8,6], dtype=**"float32"**)fc = fluid.layers.fc(input=[data\_1], size=4, act=**"tanh"**,num\_flatten\_dims=1)  cpu = fluid.core.CPUPlace() exe = fluid.Executor(cpu)  exe.run(fluid.default\_startup\_program())  a = np.random.randn(3,8,6).astype(**"float32"**) outs = exe.run(  feed={**'data\_1'**:a},  fetch\_list=[fc],return\_numpy=**False**) print(outs[0]) |  |
| 4. fluid.layers.sequence\_concat | Line5~line10构建了计算图  line12~line13和line19~line20分别创建了输入变量x\_lod\_tensor和y\_lod\_tensor  x\_lod\_tensor的lod为[[2,2]]，创建的时候是长度表示，即x的第一个句子包含两个单词，第二个句子也包含两个单词，在内部会转为偏移表示，正如输出的那样，x.lod=={{0,2,4}}，这表示x只有一个level  第一个句子包含index=0,1的单词，第二个句子包含index=2,3的单词  x的shape为[4,1]这表示x有四个单词，每个单词的embedding vector的维度为1  最后级联之后的输出为out，out.lod={{0,5,11}}  即，第一个句子包含index=0~4的单词，其中index=0,1的单词来自x，index=3,4的单词来自y；第二个句子包含index=5~10的单词 其中index=5,6的单词来自x, index=7~10的单词来自y |
| 5. fluid.layers.sequence\_first\_step | Line4~line7构建计算图  Line9~line14创建一个2-level的输入LodTensor  level0是[2,0,3]，表示包含3份文本，每份文本分别包含2,0,3个句子。level1是[1,2,1,0,3]  第一份文本包含的2个句子中，第一个句子有1个单词，第二个句子有2个单词；第二份文本包含0个句子，没有单词  第三份文本包含的3个句子中，第一个句子有1个单词，第二个句子有0个单词，第三句子有3个单词  最终的output相当于提取每个句子的第一个单词，对于包含0个单词的句子，则以0填充，而对于包含0个句子的文本，则无视。也就是说，根据level1=[1,2,1,0,3]所指定的句子来提取。最终输出结果的level0等于输入的level0 |
| 6. fluid.layers.sequence\_slice | Line4~line11构建计算图  Line13~line14定义输入LodTensor  x\_lod\_tensor.lod==[[3,2]]  表示第一个句子包含3个单词，第二个句子包含2个单词，x.dim==[5,2]表示x含有5个单词，每个单词的embedding vector的维度是2.  offset==[[0],[1]]  length==[[2],[1]]  表示对第一个句子，从第0个单词开始,取两个单词  对第二个句子，从第1个单词开始，取一个单词。(offset中的值是相对于当前这个句子的index，而不是全局的)  因此输出的lod为[0,2,3]，表示第一个句子有2个单词，第二个句子有1个单词 |
| 7. | vgg16 on Cifar10 |
| 1. | 检查所安装paddle是否符合要求，即如图，所安装paddle版本是否在0.1.0～10.0.0之间。 |
| 2. | 判断在当前终端是否可以在gpu上运行paddle。如图，为无法使用gpu。 |
| 3. | 为操作分级命名。 |

1. 疑惑问题点整理：

要求：文字分条目列出，问题点不少于3点

|  |
| --- |
| **问题点** |
| 1. 装饰器有些有有些没有templatedoc |
| 1. paddlepaddle 模型的训练状态和推理状态有什么区别, 即dropout, freeze running mean在哪里设置 |
| 1.通过2.5与3.3的实验发现在进行除法操作时，paddle会先行调用fill\_constant操作。 |
| 2.不太理解switch\_startup\_program(program)函数的功能。 |
| 3.paddle静态图与动态图使用情况，什么情况下使用静态图更好，什么情况应该使用动态图更好。 |

1. 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

|  |  |
| --- | --- |
| **可改进功能** | **拟开展工作** |
| 1. Executor接口过于繁琐 | 进一步包装executor接口 |
| 1. 添加其他损失函数的实现   RNN的二阶优化 |  |
| 1. 增加meta-learning API | 将工作Eign-Reptile用paddle实现。 |
| 1. 增加noisy labels API | 将工作SEE-Reptile用paddle实现。 |

1. 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

|  |  |
| --- | --- |
| **资料地址** | **资料涉及内容简述** |
| 1. https://zhuanlan.zhihu.com/p/133645486 | Tensorflow系统架构 |
| 1. https://www.paddlepaddle.org.cn/documentation/docs/zh/develop/api\_cn/layers\_cn.html |  |
| 1. https://www.paddlepaddle.org.cn/documentation/docs/zh/1.7/beginners\_guide/basic\_concept/lod\_tensor.html#cn-user-guide-lod-tensor 2. 介绍了LoD(Level-of-Detail) Tensor的表示形式，LoDTensor是Paddle的高级特性，是对Tensor的一种扩充。LoDTensor通过牺牲灵活性来提升训练的效率 |  |
| 1. PaddlePaddle深度学习实战 https://item.jd.com/29224619003.html | Python基础, paddlepaddle基础, VAE, Seq2Seq, GAN实践 |