从零开始，如何阅读一篇人工智能论文，及构建论文与代码的实现

本次Chat的第一部分：

首先讲解如何从零基础开始阅读一篇机器学习方向的论文，以及对待论文中的数学问题。随后，从一篇经典论文入手，讲解如何快速梳理和理解一个深度学习框架及模型。

最近人工智能和机器学习方向的论文非常多，那么一个有工程背景、学术经验较少或者有一定经验的工程师，如何阅读一篇人工智能相关的论文呢？

在刚开始的学术探索中，我倾向于全文精读，尤其是深度学习领域的经典论文，但发现这种方式花费时间太多，以至于挤压了我的真正目的——工程实现和工程结合。并且，因为想抓住的东西太多，反而没有抓住一篇文章的核心，导致很容易忘记，比如昨天读的文章就像喝了水一样忘掉了。

我将和大家从两个方面探讨。

一、从零开始，阅读一篇论文的层次

这里的从零开始，指的是我们要从零了解这篇文章做了什么事情、使用了什么方法、得到什么结果，这样的方法和结果对我有没有什么借鉴。

而不是说，接触到一个全新的领域，从读论文开始入手。对于没有过接触的陌生领域。我的方法是，先看中文综述，中文博士论文，而后是英文综述。通过中文综述，可以首先了解这个领域的基本名词、实验常用方法。否则直接从论文入手的话，作者站的高度和我们的水平不一致，很容易想当然的理解或者根本看不下去。因此，在阅读这篇文章之前，对于这篇文章中涉及到的基础知识，对应中文基础都理解透彻。

这时，回归到从零开始理解这篇文章的状态。

对一篇文章的阅读往往有3个递增的层次：

层次1. 读懂这篇文章的概要信息（5-10分钟）

- 认真读懂标题、摘要、简介（title, abstract, and introduction）。
- 只读各个部分和子部分（section and sub-section）的标题，跳过具体内容。
- 读懂结论和讨论（作者通常会在这里论述本研究的不足和缺失，为未来的研究提供建议，指明方向）。
- 浏览参考文献，记下哪些文献是你已经读过的。

因此，在第一层次过后，应该能回答出以下5个问题：

1. 文章分类：关于实现方法的文章？对于已有系统的分析文章？对于研究理论的描述文章？
2. 内容：有没有对应的相关paper？这篇文章是基于什么样的基础理论？（theoretical bases）
3. 文章的假设（assumptions）是真的正确么？
4. 贡献：这篇文章是在效果上（state of art）有了明显进步？还是方法上有了创新？还是完善了基础理论？
5. 清晰度：是一篇描述清晰的文章么？

第一个层次完成你就可以觉得是否要深入第二个层次，它足够做你的某天想用到时的知识储备，而不是现在立刻入手。

层次2. 抓住文章的内容，忽略文章细节（1个小时）

第二个层次需要认真读，抓住重点：

1. 对图、表的含义以及他们支持的结论弄懂。
2. 记下参考文献中你认为重要的未读文献，它能让你对这篇文章的背景有深刻理解。

完成第2个层次，要达到知道文章用了哪些证据，如何证明了一个什么样的结论。

尤其在这个层次中，如果遇到读不懂（原因有很多：公式太多、对术语不理解、对实验手段不熟悉、参考文献的文献过多）。说明我们还没有和作者在一个基础上，建议先从几篇重要的参考文献入手，补充背景知识。

层次3. 深入细节理解文章（5-6小时）

如果这个文章是你想应用到目前工程中的，则需要第3个层次。目标是能够在相同的假设条件下，重现（re-implement）论文。

同时，要注重论文在GitHub上的对应代码，跳到程序中能加速理解。

比较你重现的结果和原论文，就能真正理解一篇文章的创新点，以及它的隐含前提或假设。并且你能从重现过程中得到一些你未来工作的方向。

做这三个层次的好处就是，能够让你对读一篇文章的时间有合理的估计，甚至可以根据时间和你的工作需要调整掌握一篇文章的深度。

二、如何阅读包含很多数学内容的机器学习论文

这个在很多AI论文中很普遍，所以一般来讲，在第一个层次中，不需要理解一个公式的所有步骤。尽量的跳过公式，读文字描述，读实验结果，读结论。

随着你平时工作中数学的积累，在第二个层次中，你也许能直接通过看公式来真正理解作者的目的和步骤。

如果非要进入第三个层次，可能需要跟着文章做一些推导。但是实际上，如果有现成的代码实现，可以让你从工程的角度更好的理解数学过程。

最后，建议大家用这种方式，尝试把这128篇论文中自己感兴趣的领域根据自己的需要，调整阅读层次地读完。（笔者刚刚读完啦，欢迎一起来交流哦）

128篇论文，21大领域的机器学习论文

下面结合TensorFlow的架构和体系设计论文《TensorFlow:
Large-Scale Machine Learning on Heterogeneous Distributed Systems》来讲解以下两点：

TensorFlow的编程模型和基本概念

结合不到20行代码讲解静态图模型。

TensorFlow的运行方式分如下4步：

1. 加载数据及定义超参数；
2. 构建网络；
3. 训练模型；

4. 评估模型和进行预测。

下面我们以一个神经网络为例，讲解TensorFlow的运行方式。在这个例子中，我们构造一个满足一元二次函数y = ax2+b的原始数据，然后构建一个最简单的神经网络，仅包含一个输入层、一个隐藏层和一个输出层。通过TensorFlow将隐藏层和输出层的weights和biases的值学习出来，看看随着训练次数的增加，损失值是不是不断在减小。

生成及加载数据

首先来生成输入数据。我们假设最后要学习的方程为y = x2 − 0.5，我们来构造满足这个方程的一堆x和y，同时加入一些不满足方程的噪声点。

```
import tensorflow as tf
import numpy as np
# 编造满足一元二次方程的函数
x_data = np.linspace(-1,1,300)[:, np.newaxis] # 为了使点更密一些，我们构建了300个点，分布在-1到1区间，直接采用np生成等差数列的方法，并将结果为300个点的一维数组，转换为300×1的二维数组
noise = np.random.normal(0, 0.05, x_data.shape) # 加入一些噪声点，使它与x_data的维度一致，并且拟合为均值为0、方差为0.05的正态分布
y_data = np.square(x_data) - 0.5 + noise # y = x^2 - 0.5 + 噪声
```

接下来定义x和y的占位符来作为将要输入神经网络的变量：

```
xs = tf.placeholder(tf.float32, [None, 1])
ys = tf.placeholder(tf.float32, [None, 1])
```

构建网络模型

这里我们需要构建一个隐藏层和一个输出层。作为神经网络中的层，输入参数应该有4个变量：输入数据、输入数据的维度、输出数据的维度和激活函数。每一层经过向量化（y = weights×x + biases）的处理，并且经过激活函数的非线性化处理后，最终得到输出数据。

下面来定义隐藏层和输出层，示例代码如下：

```
def add_layer(inputs, in_size, out_size, activation_function=None):
  # 构建权重：in_size×out_size大小的矩阵
  weights = tf.Variable(tf.random_normal([in_size, out_size]))
  # 构建偏置：1×out_size的矩阵
  biases = tf.Variable(tf.zeros([1, out_size]) + 0.1)
  # 矩阵相乘
  Wx_plus_b = tf.matmul(inputs, weights) + biases
  if activation_function is None:
     outputs = Wx_plus_b
  else:
     outputs = activation_function(Wx_plus_b)
return outputs # 得到输出数据
# 构建隐藏层，假设隐藏层有10个神经元
h1 = add_layer(xs, 1, 20, activation_function=tf.nn.relu)
# 构建输出层，假设输出层和输入层一样，有1个神经元
prediction = add_layer(h1, 20, 1, activation_function=None)
```

接下来需要构建损失函数：计算输出层的预测值和真实值间的误差，对二者差的平方求和再取平均，得到损失函数。运用梯度下降法，以0.1的效率最小化损失：

```
# 计算预测值和真实值间的误差
loss = tf.reduce_mean(tf.reduce_sum(tf.square(ys - prediction),
```

```
                        reduction_indices=[1]))
train_step = tf.train.GradientDescentOptimizer(0.1).minimize(loss)
```

训练模型

我们让TensorFlow训练1000次，每50次输出训练的损失值：

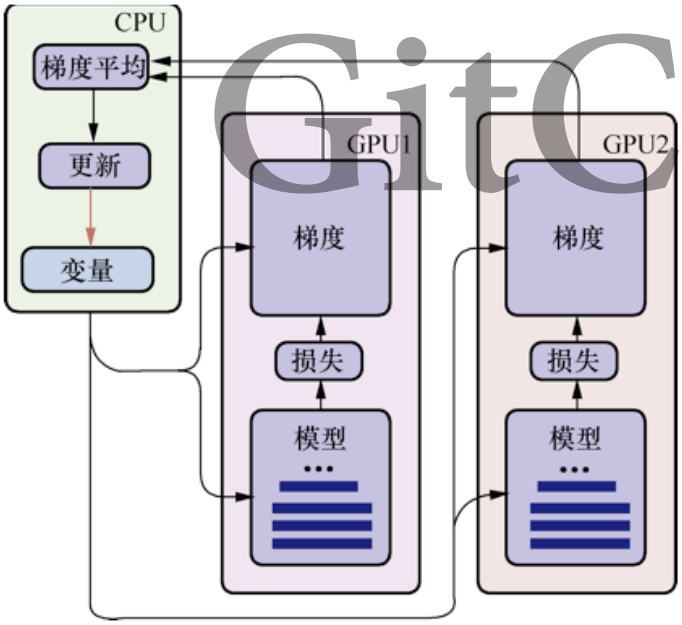
```
init = tf.global_variables_initializer() # 初始化所有变量
sess = tf.Session()
sess.run(init)

for i in range(1000): # 训练1000次
  sess.run(train_step, feed_dict={xs: x_data, ys: y_data})
  if i % 50 == 0: # 每50次打印出一次损失值
    print(sess.run(loss, feed_dict={xs: x_data, ys: y_data}))
```

TensorFlow的基本实现

包括：设备、分布式运行机制、跨设备间通信、梯度计算。

TensorFlow的分布式有两种模式，数据并行和模型并行，我们最常用的就是数据并行。数据并行的原理很简单，如图所示。其中CPU主要负责梯度平均和参数更新，而GPU1和GPU2主要负责训练模型副本（model replica）。这里称作“模型副本”是因为它们都是基于训练样例的子集训练得到的，模型之间具有一定的独立性。



具体的训练步骤如下。

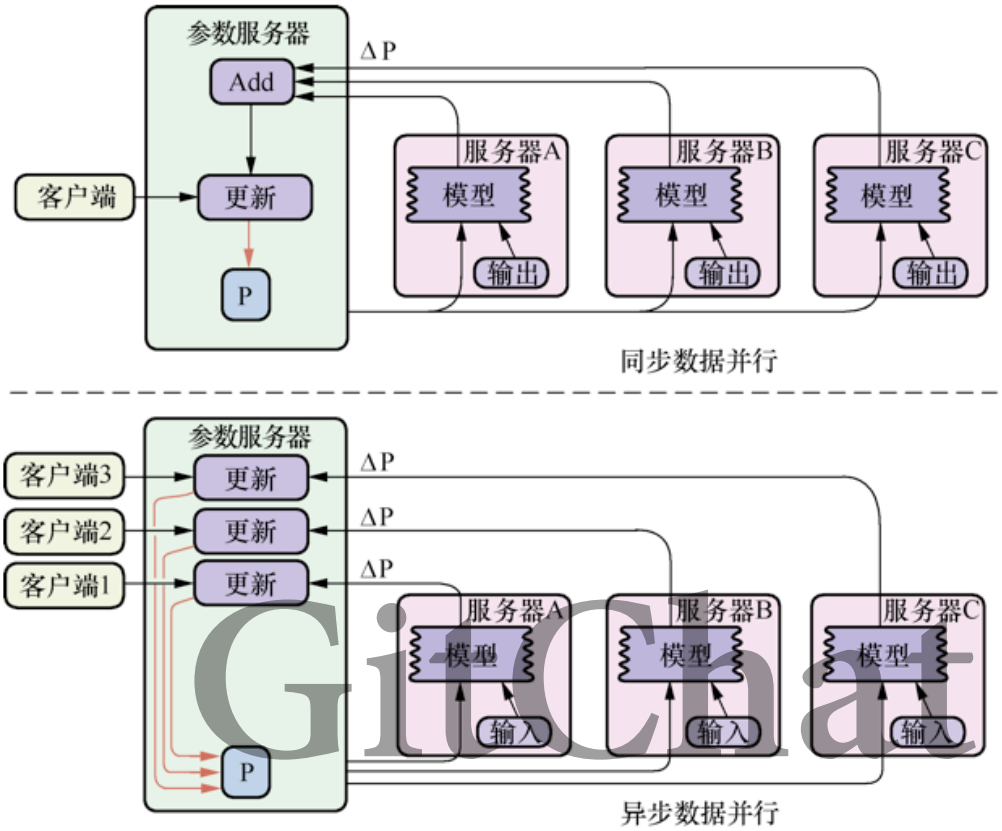
1. 在GPU1和GPU2上分别定义模型网络结构。
2. 对于单个GPU，分别从数据管道读取不同的数据块，然后进行前向传播，计算出损失，再计算当前变量的梯度。
3. 把所有GPU输出的梯度数据转移到CPU上，先进行梯度求平均操作，然后进行模型变量的更新。
4. 重复第1步至第3步，直到模型变量收敛为止。

数据并行的目的主要是提高SGD的效率。例如，假如每次SGD的mini-batch大小是1000个样本，那么如果切成10份，每份100个，然后将模型复制10份，就可以在10个模型上同时计算。

但是，因为10个模型的计算速度可能是不一致的，有的快有的慢，那么在CPU更新变量的时候，是应该等待这一mini-batch全部计算完成，然后求和取平均来更新呢，还是让一部分先计算完的就先更新，后计算完的将前面的覆盖呢？

这就引出了同步更新和异步更新的问题。

分布式随机梯度下降法是指，模型参数可以分布式地存储在不同的参数服务器上，工作节点可以并行地训练数据并且能够和参数服务器通信获取模型参数。更新参数也分为同步和异步两种方式，即为异步随机梯度下降法（Async-SGD）和同步随机梯度下降法（Sync-SGD）。如图：



同步随机梯度下降法（也称同步更新、同步训练）的含义是在进行训练时，每个节点上的工作任务需要读入共享参数，执行并行的梯度计算，同步需要等待所有工作节点把局部的梯度算好，然后将所有共享参数进行合并、累加，再一次性更新到模型的参数；下一个批次中，所有工作节点拿到模型更新后的参数再进行训练。

这种方案的优势是，每个训练批次都考虑了所有工作节点的训练情况，损失下降比较稳定；劣势是，性能瓶颈在于最慢的工作节点上。在异构设备中，工作节点性能常常不同，这个劣势非常明显。

异步随机梯度下降法（也称异步更新、异步训练）的含义是每个工作节点上的任务独立计算局部梯度，并异步更新到模型的参数中，不需要执行协调和等待操作。

这种方案的优势优势是，性能不存在瓶颈；劣势是，每个工作节点计算的梯度值发送回参数服务器会有参数更新的冲突，一定程度上会影响算法的收敛速度，在损失下降过程中抖动较大。

同步更新和异步更新如何选择？有没有优化方式呢？

同步更新和异步更新的实现区别主要在于更新参数服务器的参数的策略。在数据量小，各个节点的计算能力比较均衡的情况下，推荐使用同步模式；在数据量很大，各个机器的计算性能参差不齐的情况下，推荐使用异步模式。具体使用哪一种还可以看实验结果，一般数据量足够大的情况下异步更新效果会更好。

下面展示将如何创建一个TensorFlow服务器集群，以及如何在该集群中分布式计算一个静态图。

TensorFlow分布式集群的所有节点执行的代码都是相同的。分布式任务代码具有固定的结构：

```
# 第1步：命令行参数解析，获取集群的信息ps_hosts和worker_hosts，
以及当前节点的角色信息job_name和task_index。例如：
tf.app.flags.DEFINE_string("ps_hosts", "", "Comma-separated list of hostname:port
pairs")
tf.app.flags.DEFINE_string("worker_hosts", "", "Comma-separated list of
hostname:port
pairs")
tf.app.flags.DEFINE_string("job_name", "", "One of 'ps', 'worker'")
tf.app.flags.DEFINE_integer("task_index", 0, "Index of task within the job")
FLAGS = tf.app.flags.FLAGS
ps_hosts = FLAGS.ps_hosts.split(",")
worker_hosts = FLAGS.worker_hosts(",")

# 第2步：创建当前任务节点的服务器
cluster = tf.train.ClusterSpec({"ps": ps_hosts, "worker": worker_hosts})
server = tf.train.Server(cluster, job_name=FLAGS.job_name,
task_index=FLAGS.task_index)

# 第3步：如果当前节点是参数服务器，则调用server.join()无休止等待；如果是工作节点，则执行第4步
if FLAGS.job_name == "ps":
  server.join()

# 第4步：构建要训练的模型，构建计算图
elif FLAGS.job_name == "worker":
# build tensorflow graph model

# 第5步：创建tf.train.Supervisor来管理模型的训练过程
# 创建一个supervisor来监督训练过程
sv = tf.train.Supervisor(is_chief=(FLAGS.task_index == 0), logdir="/tmp/train_logs")
# supervisor负责会话初始化和从检查点恢复模型
sess = sv.prepare_or_wait_for_session(server.target)
# 开始循环，直到supervisor停止
while not sv.should_stop()
    # 训练模型
```


采取上面的代码框架，对MNIST数据集进行分布式训练，代码见：

https://github.com/tensorflow/tensorflow/blob/master/tensorflow/tools/dist_test/python/mnist_replica.py#L1

本次Chat的第二部分： 讲解如何将你手上的需求转化为论文的描述并实现出来。

以推荐系统为例：参考文章：

https://www.textkernel.com/building-large-knowledge-graph-recruitment-domain/

我们以招聘的招聘推荐系统中知识库的构建为例，讲解在哪里以及如何引入NLP和知识图谱的方法。

（1）为什么建设知识库？如下就是一个基于知识库的搜索：

也就是我们希望把对应的职位描述，结构化为知识图谱：

我们知道知识图谱包括实体和实体关系，那么结合招聘的场景，实体库中就应该包含：职位库、职业库、简历库、实体词库。而实体关系可能有归属关系、层级关系、关联关系。

我们来对职位描述做结构化的抽取，来设计实体关系的标签体系，如下：

	标签细分	标签名称			
招聘要求	基本要求	性别	年龄	身高	工作年限
		学历			
	外貌要求	五官外貌	形象气质	身体健康	
	性格品质	性格	品格		
	软技能	技能技巧	沟通能力		
	专业技能	技能要求	毕业专业	证书	IT类
		外语能力	驾照	软件技能	技术类
	兴趣	兴趣标签	喜欢热爱		
	工作经验	工作经历	经验标签	应届生	
	必备能力				
	加分要求	自备车标签			

具体如何来抽取呢？

1. 寻找定位词和标点符号，切分成短句。

   职位 内容：(营业员/学徒): 负责 吧台，跟随师傅调制 饮料，切配果盘，小吃; 待遇：正式员工底薪3000-3500元/月+奖金+五险一金，公司包吃住 工作地点：公司根据员工住所安排最近的上班地点

2. 从短句中基于特征字/词定位核心内容

   (营业员/学徒)
   吧台，跟随师傅调制饮料， 切配 果盘，小吃
   正式员工底薪 3000-3500 元/月+奖金+ 五 险 一 金
   公司根据员工住所安排 最近 的上班 地点

3. 核心词抽取

   营业员 学徒
   吧台 调制饮料 切配果盘 小吃
   底薪3000-3500元/月 奖金 五险一金
   最近的上班地点

那么在这个过程中，如何来寻找定位词呢？一般分为3步：

（1）定位词->种子词->定位词。例如：

（2）基于词性标注。例如：

对文本分词，进行词性标注， 查找里面的 动词、数词、量词 ，作为定位词

（3）基于语法。例如：

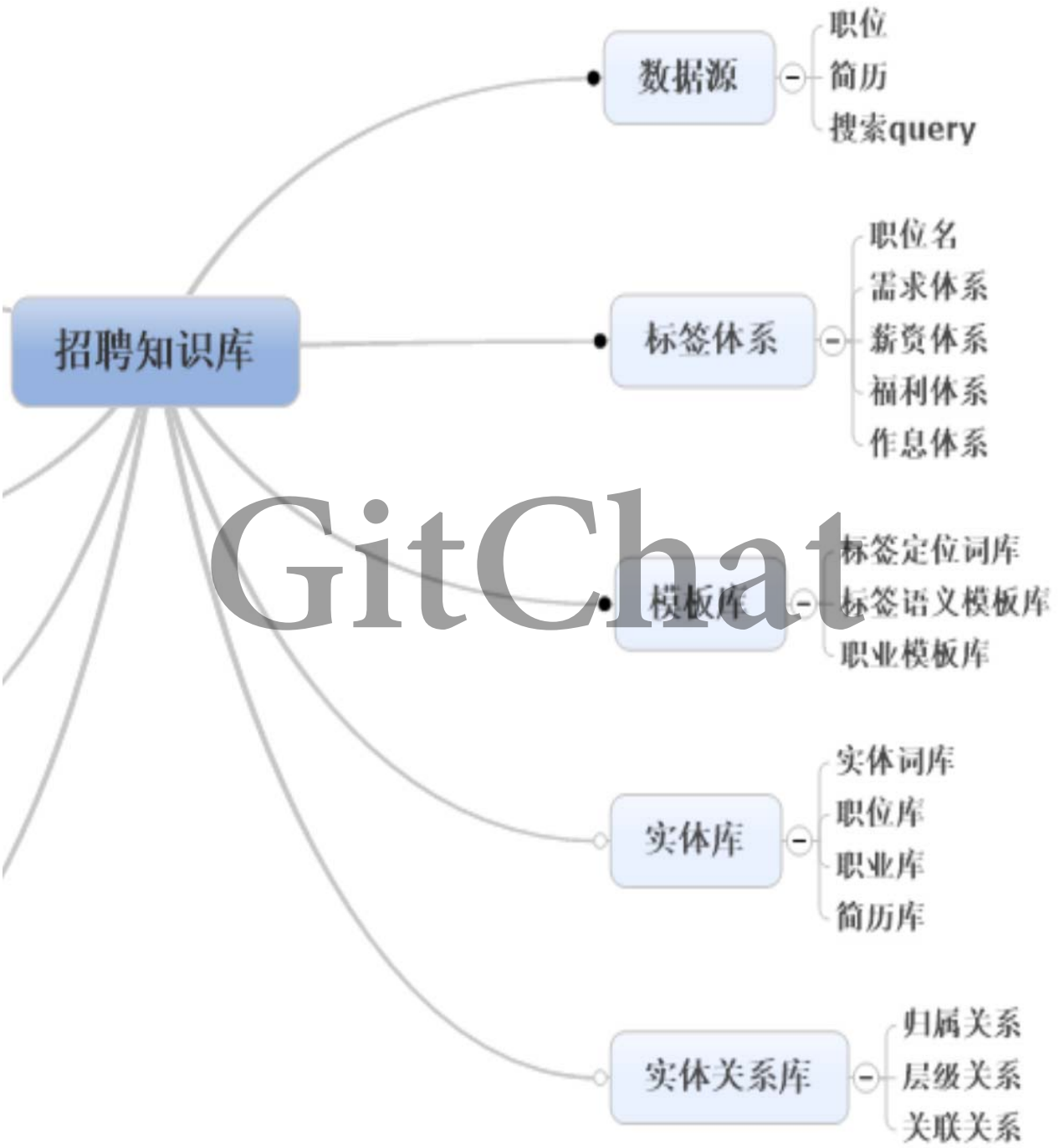
动词后面连续的名词、简称

词组联合共现频率高。动词+形容词，动词+副词的组合

关于词性标注，详见汉语词性标注集：

https://gist.github.com/luw2007/6016931

最终建立起一个招聘知识库：



最后，希望大家能够多读论文，并且总结和温习读过的文章，结合GitHub上的开源实现，在TensorFlow上多多练习。在一个领域多的论文积累量，能发现很多存在的问题和机会。