英特尔中国
金融行业
AI实战手册

intel ai

Contents
目 录

注：* 部分为 2020 年版本更新内容

版本说明：

《2020 英特尔中国金融行业 AI 实战手册》除对 2019 年版本的内容进行详细修订外，还在内容上增补了以下内容：

- 英特尔® 傲腾™ 持久内存结合创新算法与数据库产品，面向金融数据提供高可用、低 TCO 的 AI 落地解决方案；
- 英特尔® 软件防护扩展技术为金融行业带来基于硬件环境的安全机制，使用户能够借助联邦学习方法，探索多源数据在 AI 中的应用；
- 英特尔多种先进软硬件产品为基于深度学习的知识图谱提供整体性支撑，为金融行业用户提供挖掘深度高价值信息的能力；
- 英特尔端到端的统一的大数据 AI 平台，助力金融行业大数据平台与 AI 能力建设。

主编：俞巍 陈治文

作者（排名不分先后，按姓氏首字母排序）：

胡英 乐鹏飞 李志强 吴国安 夏磊 袁超 臧战 赵玉萍 Parviz Peiravi

曹津 龚毅敏 鲁懿 陆礼明 沈飞廉 孙宇 王东方 魏剑 伊红卫 朱乐骏

此外，本手册的编撰工作也得到了各合作伙伴以及诸多英特尔同事们给予的大力支持帮助，在此表示感谢。

V.2020.002

趋势篇

人工智能持续驱动金融行业创新

经过多年演进，人工智能（Artificial Intelligence，AI）正进入一个发展新阶段。越来越多的企业在选择这一给人类经济与社会生活带来颠覆性影响的技术，来开启数字化转型的新篇章。尤其当我们把目光投向金融行业时更不难发现，在过去的十余年中，堪为这一行业风向标的巨头们正将更多资金投入大数据、机器人和云计算服务领域，这些举措也受到投资者的热捧。来自新浪财经的报道这样写道[1]：

> 时间倒回到 2000 年。华尔街投行高盛在纽约总部的美国股票交易柜台雇用了 600 名交易员，根据投资银行大客户的订单买卖股票。如今，他们只剩下了 3 个股票交易员。
>
> 根据美国财经媒体 Business Insider 的统计，高盛目前 33000 名全职员工中，超过 9000 名员工是程序员和工程师。近几年在公开场合，高盛 CEO 反复强调，高盛的定位已经今时不同往日，高盛是一家技术公司。
>
> ……
>
> 华尔街上另一家巨头摩根大通也采取了类似的做法。摩根大通很早就设立了技术中心，聘用约 4 万名技术人员专门研究大数据、机器人和云基础设施，技术预算达 96 亿美元，占其总收入 9%。去年，该公司还宣布使用全球首创的机器人来进行他们的全球股票算法交易。在此之前，摩根大通在欧洲研发的人工智能项目 LOXM 早就已经在交易中尝到了甜头。

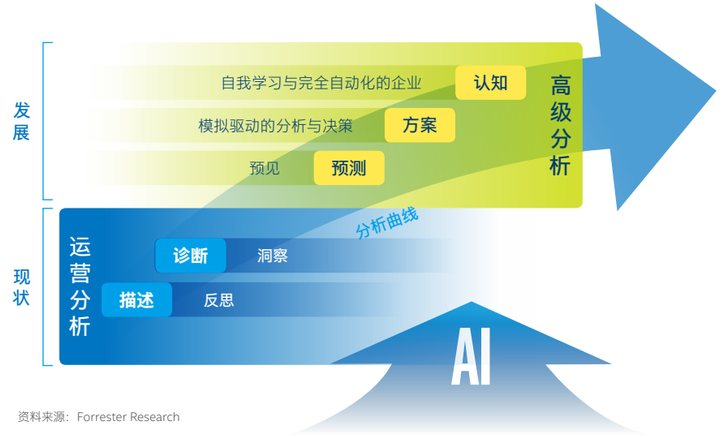
那么，AI 何以成为金融行业新的宠儿？究其原因，我们不难发现，其实是因为 AI 与金融行业在诸多特性上天然契合。首先，AI 与金融都构建在海量的数据之上，这为 AI 的模型训练与预测推理提供了丰沃土壤；其次，AI 可以大幅降低传统金融行业用于客户关系维护的成本；同时，AI 也能帮助金融企业展开更多的精准狙击，进一步提升业务质量；而更为重要的是，拥有众多创新基因的金融业务与 AI 结合，可为行业发展带来更大空间，为业务创新带来更多可能。

以银行为例，通过与 AI 融合，即可以依托庞大的业务数据，以智能的方式驱动数据分析与预测方法创新，进而获得新的洞察，拥抱更为灵敏、高效的商业模式，并规避诸如贷款逾期、违规欺诈等风险，在未来竞争中占得先机。

然而，从数据分析到 AI 应用并不能一蹴而就。如下图所示，企业在数据分析技术的演进上，一般会经历在规模和成熟度上递增的五个阶段。这五个阶段分别是：描述性、诊断性、预测性、方案制定和认知性。成熟的 AI 能力，适用于以上数据分析的各个阶段，是把数据分析推向更高成熟度和更大规模的重要动力。

近年来，AI 的算力、算法以及数据能力都获得了快速突破，包括：

- **算力提升：** 在摩尔定律的推动下，芯片制程技术和架构创新正使计算力不断取得突破，满足了机器智能对于计算密集



资料来源：Forrester Research

图 1-1-1　企业数据分析技术的演进

[1] 相关媒体报道请参见 http://finance.sina.com.cn/world/2018-05-22/doc-ihawmaua4464623.shtml

型算法的需求。例如，通过人工神经网络进行深度学习的概念已经存在了至少 20 年，但直到最近几年，当计算技术提升到能够满足 AI 所需的高精度、高速度的运算能力时，将这些计算密集型算法付诸实践应用才成为可能。

- **创新推动**：AI 发展仅靠计算力和数据来推动，还是远远不够的。驱动其向应用迈进的关键动力，毫无疑问是创新，正是它推动 AI 跨越了从研究到主流使用的临界点。实践已经证明，每一次 AI 算法的创新，都预示着更多的应用可能性，并吸引更多的创新者加入到应用开发行列中来。比如，20 世纪 90 年代的神经网络创新，驱动了对 AI 的重新阐述和研究，并在 2009 年和 2012 年，分别在语音识别和图像识别等领域获得了突破性进展，这也成为当前 AI 创新发展的催化剂。

同时，数据洪流的到来，也驱动金融行业 AI 迎来高速发展浪潮。一方面，物联网（IoT）的广泛应用、智能互联设备的快速普及，带来了越来越多的结构化和非结构化数据。有研究表明，到 2020 年，物联网中智能、互联设备产生的数据可达 40 ZB（1ZB=10 万亿亿字节）[2]。如此海量的数据，无疑为金融领域 AI 算法的训练，以及发掘全新洞察奠定了基石。另一方面，面向不同类别金融数据的处理和运用，也出现了更多的方法论和硬件产品。例如，面对金融数据普遍的高维特性，企业已经可以选择更有针对性的高维算法、高性能实时特征数据库，以及具有更好落盘性能和更高容量的存储设备来构建 AI 应用，降低 TCO 并提升处理效能。

值得一提的是，虽然高质高量的数据能给 AI 核心竞争力带来巨大提升已成为共识，但在金融行业，为了规避数据泄露风险，企业往往会构建一系列严密的防护措施，由此也引发了严重的数据孤岛问题。为了让多源数据的交互、传输和聚合更具安全性，联邦学习等新的联合学习方法，正帮助用户在确保数据安全的前提下，建立起更多安全可信的多源数据协同训练方案，确保为 AI 应用提供更丰富的数据源，以提升其精度。

现在，基于可信执行环境（Trusted Execution Environment，TEE），典型如英特尔® 软件防护扩展（Intel® Software Guard Extensions，英特尔® SGX）技术构建的联邦学习方案，已在保险定价、信贷风控、销量预测等多个金融领域落地实例化。以保险定价为例，方案可以在保护各合作方用户隐私数据不出本地的前提下，安全合规接入多方数据。进而，一方面帮助用户以多源多维度的用户行为数据，来提供个性化定价策略；另一方面，通过多源的安全大数据，有效识别恶意骗保行为。一些数据表明，基于联邦学习方法建立的保险数据模型具有更丰富的风险特征体系，可使行业定价准确率获得大幅提升。

通过以上算力、算法和数据处理技术进步， AI 正成为金融企业开展高质量数据分析和业务预测的重要手段。得益于 AI 技术的不断发展，在金融行业的前端、中台和后端，都已经有了相对成熟的应用方案。

在前端，感知类技术（计算机视觉、语音识别、自然语言处理等）不断走向成熟，有代表性的应用已有客服聊天机器人、自动身份识别等。

客服聊天机器人能够遵循与客户交互的标准路径，借助机器学习算法，观察对话并理解客户的意图，在遇到困难时将问题发送给人工处理，并对人工答复加以学习，从而持续提升客户服务质量、降低服务成本。而自动身份识别则是通过语音识别、面部识别等方式，分析用户声音、眼部、面部特征，对用户进行身份验证。此类 AI 验证方法比原来的安全问题或密码验证效率更高，而且无需用户默记密码，可让客户体验大为改善。

在中台，AI 可以提高基于信息的分析决策效率，帮助用户更加快速地抓住商机。既有的商业智能（Business Intelligence，BI）和传统的数据分析方法，往往停留在趋势分析、原因挖掘、数据挖掘与预测层面。而 AI 的引入，既延伸了分析的广度，也提高了分析的深度。AI 可以通过不断学习和完善，提高建议的相关性和特异性，实现“个性化分析”，为风险管理、营销、服务等提供基于智能化的分析和决策。

例如，AI 可以基于社交网络的信用评分，优化现有分数或为无信用记录的人员进行评分；也可以通过自然语言分析（Natural Language Processing，NLP）方法生成分析报告，还可以分析与评估财务数据。同时，AI 还可以开展动态欺诈检测，从实时复杂交易中发现和规避风险；更可以通过客户行为研究，提供个性化的财务健康建议。此外，金融企业还可以通过 AI，根据客户和产品 DNA，实现个性化营销，提供独一无二的个性化服务。

[2] 相关媒体报道请参阅：https://cloud.tencent.com/info/5a0335164080e583fd6a2d4a735e7def.html

通过引入联邦学习等新的联合学习方法，AI 可以横向聚合更多的训练数据来提升模型精度，从而带动跨地域、跨领域的企业级数据合作，以“合作共赢”的方式使各参与方都能从不断提升的 AI 能力中获益。

在后端，比如在行业的合规以及 IT、财务等支持职能中，存在大量高度重复性的工作。AI 的重要应用之一，正是要逐步接手这些重复性的工作。所以，AI 在后端支持流程中，也存在大量应用机会和潜力。

总之，围绕着前、中、后三端的 AI 能力，金融行业中的 AI 应用场景众多，目前主要聚焦在风险 & 合规、客户体验、营销决策和智慧运营四个方向。

图 1-1-2　金融行业人工智能应用场景

如下图所示，风险与合规主要包括了应对欺诈风险、信用风险、宏观风险、反洗钱以及合规政策文件分析等，AI 技术应用主要涉及机器学习、基于深度学习的人脸识别和语音识别以及知识图谱等。客户体验主要包括智能投顾、智能理赔、智能客服、身份识别、农牲识别等，AI 技术应用主要涉及基于深度学习的人脸识别和语音识别、行为模式识别、NLP 以及机器人技术等。营销决策主要包括客户画像、精准营销、智能推荐、征信评分、资金流向监控以及量化分析等，AI 技术应用主要涉及深度学习、机器学习等。智慧运营主要包括网点智能布局、单据识别、智能运维等，AI 技术主要涉及基于深度学习的图像识别、知识图谱等。

在下一章“实战篇”中，我们将围绕金融反欺诈、风险预测、客户营销、智能推荐、智能核保等多个场景，结合与中国银联、中国人寿上海数据中心、万事达卡、第四范式、中国平安以及合合信息等合作伙伴的经典案例，来详细阐述实战中的 AI 应用部署情况，揭示其所涉及的技术细节，尤其是英特尔相关技术与产品在这些真实场景中的应用和优化方案，以及硬件、软件配置的最佳实践推荐。

实战篇

基于大数据 +AI 的高效实时金融反欺诈解决方案

英特尔与金融用户合作探索利用 RNN 模型学习用户行为特征

反欺诈模型演进

在与金融客户的合作实践中，我们发现现有金融行业反欺诈应用模型设计往往存在以下问题：

- 学习用户行为的算法缺乏足够的应用实践；
- 传统深度学习方法对数据量的要求大，但金融企业无法针对算法给出每个用户行为模式的历史交易数据；
- 数据非平衡性（Imbalance ratio）状况严重，即绝大多数训练数据都源自正常交易行为，正常/非正常数据比例大概是 10 万 ~100 万比 1。

传统上，金融企业与机构往往采用基于规则的方式来构建其风控反欺诈模型，其特征就是不断建立、更新基于用户行为特征的规则库。当交易发生时，系统会通过规则引擎来监测该笔交易潜在的风险。

例如一个常常出国的商务人士，在交易规则库中，他出现大笔海外交易行为是正常的，对于一个很少离开居住地使用信用卡的老年人，他的正常交易行为可能是小额、本地和多笔的特性，当其出现大额交易情况时，这一异常交易特征就会被规则引擎所匹配，并引起警觉。而当该账户屡次出现异地不正常大额交易记录，这个账户可能就会被风控系统列入监控范围，并实施后续的核查工作。

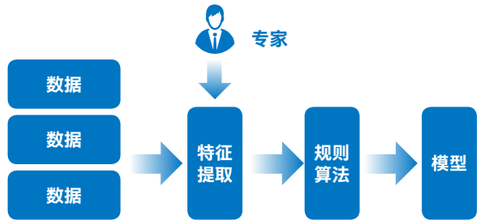


图 2-1-1 传统反欺诈模型方案

基于规则的风控系统固然有效，但其作为一种反向系统，需要规则库不断通过已有业务案例进行总结。这意味着用户每隔一段时间就要耗费大量资源来总结业务，更新规则。而随着业务场景的增多，交易规则复杂度也不断提升，使风控系统的资源消耗和监控时延持续增加。为此，金融机构开始尝试利用 AI 能力，构建更为高效的金融反欺诈模型，这一 AI 能力建设依托于机器学习、深度学习等多种方法。

与基于规则的方法相比， AI反欺诈方案具有更高的客观性及准确性，引人注目之点就是能够实施 “对规则的自我学习”。通俗来讲，基于规则的方法是预先告诉系统，A方法、B方法是错的，错了就告警。而机器学习、深度学习方法则是将大量历史数据作为学习样本，并通过大量的计算单元进行训练，从而得到一个评估模型。当新的交易进入这一模型时，系统能自我判别交易的合法性。

图 2-1-2 当前反欺诈模型中常见算法

如图 2-1-2 所示，现在，机器学习中一些优秀的分类算法，例如逻辑回归（Logistic Regression，LR）、随机森林（Random Forest，RF）以及梯度提升决策树（Gradient Boosting Decision Tree，GBDT）等分类算法都已被反欺诈模型广泛地采用。

基于 RNN 的深度学习方法

深度学习方法是反欺诈 AI 应用中常用的方案，递归神经网络（Recurrent Neural Networks，RNN）是金融反欺诈模型中常见的深度学习模型之一。典型的 RNN 结构如下图 2-1-3 所示，RNN 会对每一个时刻的输入，结合当前模型的状态，给出一个输出，从图中可以看出，RNN 的主体结构 A 的输入除了来自输入层的 X，还有一个循环，来提供当前时刻的状态，同时 A 的状态也会从当前步传递到下一步。

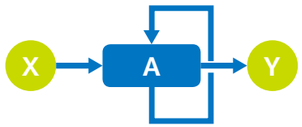


图 2-1-3 典型的 RNN 结构

在实践中，RNN还有两种重要的变种，即长短期记忆（Long Short-Term Memory，LSTM）和门控循环单元（Gated Recurrent Unit，GRU），LSTM可以通过3个特别的 “门” 结构设计，来避免经典RNN网络结构中的长期依赖问题，进而大幅度提升记忆时长。LSTM 中的 “门” 是Sigmod神经网络和位乘法的结合体，以Sigmod为激活函数的全连接神经网络层会输出一个0到1之间的值，描述当前可通过该结构的输入信息量，当Sigmod输出为1时，全部信息都可通过；反之当Sigmod输出为0时，任何信息都无法通过。而GRU是LSTM的改进版，其将LSTM的3个 “门” 结构合并为2个，在计算当前新信息的方法时和LSTM有所不同。

基于 RNN 的深度学习方法在基于序列的分析工作中，一直有着良好的表现。借助序列标记技术，RNN 深度学习方法可用于分析不同账户的实时行为状态。

但单一的深度学习方法在对大量交易数据进行欺诈侦测建模时，效果却并不理想。究其原因，是因为 RNN 等神经网络虽然能学习到交易序列间的特征关联，但对单笔交易内的特征学习能力不足，达不到预期的目标。

例如在反欺诈侦测中，“午夜发生的大额场外交易” 这样的特征组合可能是非常可疑的交易行为，而单独的 “场外”、“大额” 和 “午夜” 却都是常见特征。这一类特征组合，通过单一的深度学习方法很可能无法获得良好的训练结果。

模型实现及优化

RNN 深度学习方法可以使用 Keras、TensorFlow 等深度学习框架予以实现。Keras 的后端可以是多种框架，例如 TensorFlow、Theano、CNTK 等等。以下算法模型的开发，选择 TensorFlow 作为 Keras 的后端为例展开描述。可通过修改 $HOME/.keras/keras.json 内相关部分来进行设置：

```
"backend": "tensorflow"
```

■ 面向英特尔® 架构优化的 TensorFlow的安装

TensorFlow 是目前深度学习领域的主流框架之一，可以帮助深度学习开发者更高效地利用计算资源，构建深度学习模型。为充分利用英特尔® 架构和实现更高性能，目前 TensorFlow 框架已使用面向深度神经网络的英特尔® 数学核心函数库（Intel® Math Kernel Library for Deep Neural Networks，英特尔® MKL-DNN）进行了全面优化。

Anaconda 是一个开源的 Python 发行版本，其可以帮助用户使用英特尔® MKL-DNN 构建 TensorFlow（从 TensorFlow v1.9 开始支持该功能），以便为英特尔® 架构处理器提供更高性能。如果用户目前使用 Conda 软件包管理器管理环境和软件包，只需在虚拟环境中安装 Anaconda.org 中的 TensorFlow 软件包。

在Anaconda中安装TensorFlow命令如下：

```
conda install tensorflow
```

如果用户的 Anaconda 通道并非默认的最高优先级通道，也可使用如下命令获取面向英特尔® 框架优化的 TensorFlow。

```
conda install -c anaconda tensorflow
```

除上述安装方法外，面向英特尔® 架构优化的 TensorFlow 还可作为 wheel 和 docker 映像，或者 Conda 软件包分发。

另外，用户也可以通过 PIP，在现有 Python 环境中安装面向英特尔® 架构优化的 TensorFlow，命令如下：

```
pip install intel-tensorflow
```

或者用户也可在现有 Python 环境中安装 wheel，并建议使用英特尔® Python 分发包。在 Python2.7 和 Python3.6 中安装 1.13.1 版本的命令分别为：

```
# Python 2.7
pip install https://storage.googleapis.com/intel-optimized-tensorflow/tensorflow-1.13.1-cp27-cp27mu-linux_x86_64.whl
# Python 3.6
pip install https://storage.googleapis.com/intel-optimized-tensorflow/tensorflow-1.13.1-cp36-cp36m-linux_x86_64.whl
```

■ 面向英特尔® 架构的TensorFlow优化方法

面向英特尔® 架构平台对 TensorFlow 开展一系列优化，可以有效地提升模型工作效率。这些优化方法包括：调整处理器核心数量，引入非统一内存访问架构（Non- Uniform Memory Access Architecture，NUMA）技术以及英特尔® MKL-DNN。优化步骤如下：

■ 环境变量设置

首先，需要对环境变量进行设置，命令包括：清空系统的缓存（cache），将处理器设置为性能优先的模式，即运行在最高频率，打开处理器的睿频加速。设置命令如下所示：

```
echo 1 > /proc/sys/vm/compact_memory
echo 3 > /proc/sys/vm/drop_caches
echo 100 > /sys/devices/system/cpu/intel_pstate/min_perf_pct
echo 0 > /sys/devices/system/cpu/intel_pstate/no_turbo
echo 0 > /proc/sys/kernel/numa_balancing
cpupower frequency-set -g performance
export KMP_BLOCKTIME=1
export KMP_AFFINITY=granularity=fine,verbose,compact,1,0
export OMP_NUM_THREADS=20
```

- KMP_BLOCKTIME 设置为 1，是设置某个线程在执行完当前任务并进入休眠之前需要等待的时间，通常设置为 1 毫秒；
- KMP_AFFINITY 设置为 Compact，是表示在该模式下，线程绑定按计算核心的计算要求优先，先绑定同一个核心，再依次绑定同一个处理器上的下一个核心。此种绑定适用于线程之间具有数据交换或有公共数据的计算情况，优势在于可以充分利用多级缓存的特点；
- OMP_NUM_THREADS 设置为 20 是将并行执行线程的数量设定为一定的物理核心数。

■ 测试代码中添加线程控制

添加线程控制的代码如下：

```
config = tf.ConfigProto()
config.allow_soft_placement = True
config.intra_op_parallelism_threads = FLAGS.num_intra_threads
config.inter_op_parallelism_threads = FLAGS.num_inter_threads
```

如上述代码所示，在进行 tf.ConfigProto() 初始化时，我们也可以通过设置 intra_op_parallelism_threads 参数和 inter_op_parallelism_threads 参数，来控制每个操作符 op 并行计算的线程个数。二者的区别在于：

- intra_op_parallelism_threads 控制运算符 op 内部的并行，当运算符 op 为单一运算符，并且内部可以实现并行时，如矩阵乘法，reduce_sum 之类的操作，可以通过设置 intra_op_parallelism_threads 参数来并行，intra 代表内部。
- inter_op_parallelism_threads 控制多个运算符 op 之间的并行计算，当有多个运算符 op，并且它们相互独立，运算符和运算符之间没有直接的路径 Path 相连。TensorFlow 会尝试并行地对其进行计算，使用由 inter_op_parallelism_threads 参数来控制数量的一个线程池。

通常而言，intra_op_parallelism_threads 设置为单个处理器的物理核心数量，而 inter_op_parallelism_threads 则设置为 1 或者 2。

■ 利用 NUMA 特征来控制处理器计算资源的使用

数据中心使用的服务器，通常都是配置两颗或以上处理器，多数都采用 NUMA 技术使众多服务器像单一系统那样运转。处理器访问它自己的本地存储器的速度比非本地存储器快一些。为了在这样的系统中获得更好的计算性能，需要通过一些特定指令来加以控制。numactl 就是用于控制进程与共享存储的一种技术机制，它是 Linux 系统中广泛使用的计算资源控制方法。具体使用方法如下所示：

图 2-1-4 NUMA 特征来控制处理器计算资源的使用

使用 Python 命令执行 numactl 命令如下：

```
numactl -C 0-19,40-59 -m 0 python3 test.py
```

命令中表示 test.py 在执行的时候只使用了处理器 #CPU0 中的 0-19 和 40-59 核，所使用的内存也只使用了处理器 #CPU0 对应的近端内存。

■ Keras/TensorFlow 实现多层 LSTM/GRU

在 keras 中实现多层的 LSTM/GRU 的代码如下:

```
classifier.add(GRU(number_of_cells=128,
    input_shape=(timesteps, data_dim),
    recurrent_dropout=0.3,
    activation='sigmoid',
    recurrent_activation='hard_sigmoid',
    return_sequences=True))
classifier.add(GRU(number_of_cells=64,
    recurrent_dropout=0.3,
    activation='sigmoid',
    recurrent_activation='hard_sigmoid'))
```

*更多面向英特尔® 架构优化的TensorFlow的技术细节，请参阅本手册技术篇相关介绍。

“三明治” 多层反欺诈侦测模型

■ 模型简介

“GBDT→GRU→RF”三层架构如图2-1-5所示，首先，这一框架将针对单一深度学习方法（例如RNN）在单笔交易内特征学习能力上的不足，通过英特尔提供的Analytics Zoo工具，在框架的前端引入GBDT模型进行特征优化，并将优化后的特征与人工特征相结合，作为GRU网络的输入，以此来学习序列间的特征，并且将单笔交易内的特征时序化。这一过程可以对数据实施有效的过滤，从而为后续的GRU模型提供真正有用的数据。

在中间层，框架并没有直接使用 GRU 网络的输出作为直接的欺诈侦测判别，而是将其作为序列间特征学习的一环，将学习得到的序列间特征与原先的交易内特征相结合，形成最终交易特征向量，

最后在此基础之上，为进一步地将时序特征进行融合学习。在框架的最后，这一架构还叠加了一个顶层的 RF 模型，作为最终的欺诈判别分类器。

■ 软件栈

“三明治” 多层反欺诈侦测模型软件栈如图2-1-6所示，底层是由英特尔® 至强® 金牌处理器（6000系列）为基础构建的硬件基础设施。其上，是RedHat Linux操作系统（Centos7.4 Kernel 3.10.0-957.12.1.el7.x86_64），并由虚拟化软件创建虚拟机。虚拟机上部署了英特尔® MKL-DNN或英特尔® MKL，并安装有面向英特尔® MKL-DNN的优化的TensorFlow1.10以及英特尔® Python分发包。在顶层部署了欺诈检测应用。

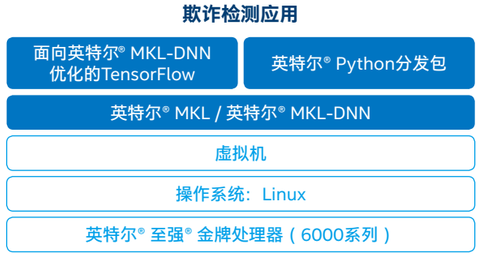


图 2-1-6 “三明治” 多层反欺诈侦测模型软件栈

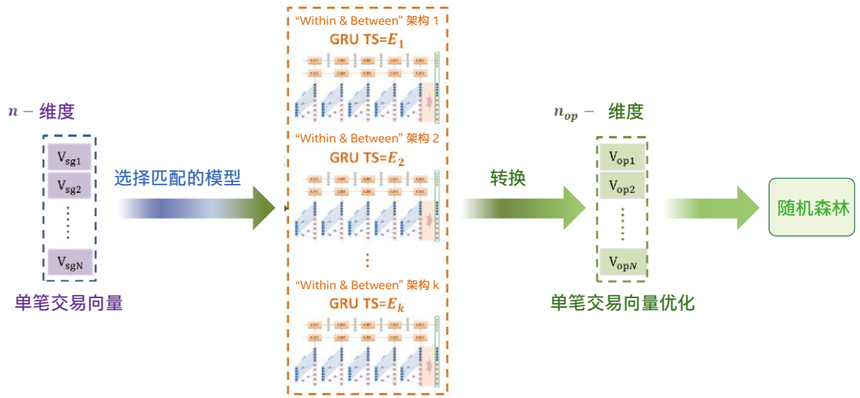


图 2-1-5 GBDT→GRU→RF “三明治” 结构反欺诈模型 [3]

[3] 三明治结构反欺诈模型技术详情请参阅：Transaction Fraud Detection Using GRU-centered Sandwich-structured Model

■ 英特尔® Python分发包的安装

可通过执行以下不同命令，在python3/python2环境下安装英特尔® Python分发包核心包:

```
conda create -n idp intelpython3_core python=3
conda create -n idp intelpython2_core python=2
```

或安装英特尔® Python分发包完整包:

```
conda create -n idp intelpython3_full python=3
conda create -n idp intelpython2_full python=2
```

激活应用环境:

在 Linux/MacOS 下:

```
source activate idp
```

在 Windows 下:

```
activate idp
```

使用 Conda 命令安装额外的软件包，例如 sympy 等:

```
conda install sympy
```

以上过程，详情请参阅:
https://software.intel.com/zh-cn/articles/using-intel-distribution-for-python-with-anaconda

*更多英特尔® Python分发包的技术细节，请参阅本手册技术篇相关介绍。

■ 针对数据非平衡性的处理办法

数据非平衡性是反欺诈应用中经常碰到的问题。在一些场景中，欺诈样本和正常样本的比例高达 10 万 -100 万比 1。基于这一数据不平衡状况，在采样和训练时，可以基于以下的原则:

1. 由于正常样本的数据量非常大，正常的采样就可以满足训练对于正常样本的需要;

2. 在训练的时候，我们把欺诈样本数据的权重提高。

```
class_weights= dict()
class_weights[0] = 1  #正常样本权重
class_weights[1] = 20  #欺诈样本权重
classifier.fit(X_train,
    y_train,
    batch_size=BS,
    epochs=runEpoch,
    class_weight=class_weights)
```

3. 不同于图像和语音等数据容易重新标注的情形，欺诈数据很难被生成出来并被标注。所以我们还是采用随机打乱次序并多次训练欺诈样本，但却仅需单次使用正常样本，以便进一步解决非平衡的问题。

■ 提升算法准确性的方法

■ 不同方法组合的次序和准确性有相关性:

特征内提取特性的算法与特征间提取特性的算法进行结合时，不同的次序会导致不同的准确性，我们的测试表明，在 2 个特征内提取特性的方法之间，加入特征间提取特性的方法所获得的准确性是最高的。

■ 三明治的结构使用旁路加强特征重用:

和 Densenet 算法一样，“三明治” 结构也构建了不同方法之间的连接关系，这种使用旁路的方法，使其可以通过加强特征重用，来提升整体的训练效果。

软件配置建议

对训练数据流程化建模和多层反欺诈模型构建的验证工作，可以参考以下基于英特尔® 架构平台的软件配置。

名称	规格
操作系统	Centos7.4
Linux 内核	3.10.0-957.12.1.el7.x86_64
工作负载	GRU
编译器	GCC5.4
库	英特尔® MKL-DNN最新版本
框架	面向英特尔® 架构优化的 TensorFlow 发布版
Hadoop发行版	Cloudera CDH-5.9.0. 或更高版本
Spark 版本	Apache Spark-2.1.0. 或更高版本
其他软件配置	Anylitics Zoo 英特尔® Python 分发包

中国银联应用案例

背景

伴随金融业务的高速扩张，其风险指数也在不断上升。尤其在银行卡、信用卡等领域，欺诈损失率正随着欺诈损失金额的增长而逐年上升。因此，反欺诈正成为金融行业实施风控的重要方向。

在风险形式上，传统风险与新型风险也正相互交织。除了层出不穷的传统金融欺诈手段，例如信用欺诈、盗刷欺诈、恶意套现以及保险业骗保等，伴随互联网时代出现的个人信息泄露、钓鱼网站、欺诈黑产化等问题，也带来更高频化、精准化的的金融欺诈犯罪。

为应对这一问题，各类金融机构也制定了众多缜密的反欺诈手段来予以反制。随着信息化技术的进步，尤其是 AI 技术的不断突破，越来越多的 AI 能力正与金融风控系统相结合，构成更有效的反欺诈模型。

作为一家提供专业银行与支付服务，发卡量和交易量市场份额世界第一的金融机构，中国银联（以下简称“银联”）正不遗余力地引入 AI 技术能力，开展高效金融反欺诈模型的构建。在本案例中，中国银联与英特尔一起，共同开展基于深度学习的反欺诈技术研究。

通过结合在基于规则、机器学习等反欺诈模型中汲取的经验，中国银联基于“三明治”结构的多层模型，以及基于英特尔® 架构的多方位优化，构建高效的欺诈侦测方案。目前，该方案已在伪卡/套现欺诈侦测等场景中进行了实测，并获得良好效果。

解决方案

中国银联采用 GBDT→GRU→RF “三明治”结构，构建了高效的反欺诈模型。首先，银联基于 Analytics Zoo 以及 Spark pipeline 对数据进行流程化建模。通常地，AI 训练模型需要针对用户的每条交易和行为进行分析，即通过算法学习到每个持卡人的消费行为模式，去分析是否异常，并在发现异常交易行为时启动拦截动作，但这需要系统引入海量的交易数据。同时，训练模型要学习到用户的历史交易行为，每个人至少需要数百笔非正常交易数据供模型学习所用。

为此，银联基于 Hadoop 构建了海量数据存储平台，并引入 Analytics Zoo 等来对训练数据进行流程化建模。针对学习历史交易数据不足的问题，利用建模过程，平台可从少量的原始字段中衍生出了几百个特征因子，归纳成当笔 / 上笔交易、长短时统计以及可信特征变量等 6 大维度，并通过这些特征工程来帮助模型进行更好的学习。而后，银联基于 “GBDT→GRU→RF” 三层架构模型，在上百个节点组成的训练集群上开展其反欺诈侦测模型的构建，并取得了良好的效果。

通过多方位的测评，全新的多层反欺诈模型无论是在召回率，还是在准确率方面都达到预期效果，与传统机器学习方法，或单一的 RNN 方法相比，F1 值（F1 Score，一种准确率和召回率的加权平均值，用于衡量侦测模型的性能表现）有了质的提升，超过了业务部署的临界点。

如图 2-1-7 所示，左图中，与其他机器学习、深度学习模型，或者多层模型相比，GBDT→GRU→RF “三明治” 结构反欺诈模型有着最优的精度 - 召回（Precision–Recall，PR）曲线（最上部曲线为 GBDT→GRU→RF “三明治” 结构反欺诈模型测试值）。在右图中，可以看出，随着数据非平衡率（Imbalance ratio）的增加，GBDT→GRU→RF “三明治” 结构反欺诈模型的 F1 值下降最为缓慢。

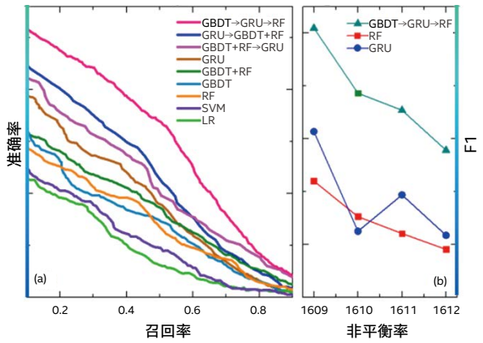


图 2-1-7 GBDT→GRU→RF “三明治” 结构反欺诈模型评估效果

在完成流程化建模和多层反欺诈侦测模型构建后，银联将其进行了封装和整合，并以 API 接口的方式提供端到端的智能分析解决方案，从而更好地为业务人员提供服务。如图 2-1-8 所示，用户通过 API 接口等方式提供入参，即可获得经过智能模型运算分析后的结果指标。以三明治结构的欺诈侦测模型为例，其

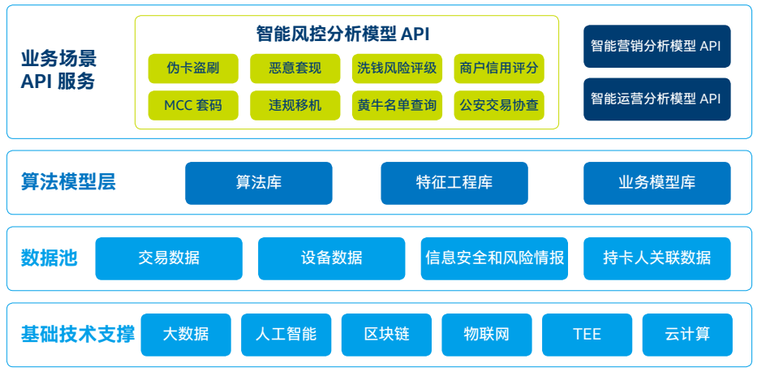


图 2-1-8 中国银联电子支付研究院智能分析服务平台架构图

可以为伪卡、套现等欺诈侦测场景提供底层模型支撑，而业务人员则不需要深入研究这些复杂的模型，只需要根据数据规范调用上层 API 即可。

在银联的训练集群中，全部采用了基于英特尔® 至强® 处理器的平台。该平台不仅在内核、高速缓存等方面表现优异，而且能以多项硬件增强技术助力提升框架性能。除了基本的计算能力支撑，平台还为银联AI反欺诈模型的构建提供了以下能力：

- 支持高度不规则的计算，如树构建、熵计算、树遍历、缩减等；
- 支持常规计算，如 GRU、非线性激活、批处理规范化等。

同时，英特尔® 至强® 处理器/英特尔® 至强® 可扩展处理器所集成的英特尔® 高级矢量扩展512（英特尔® AVX-512）技术，为银联 “三明治” 结构反欺诈模型提供了出色的并行计算能力。

小结

针对基于机器学习的方法对序列化交易特征学习能力的不足，以及单一深度学习模型对单笔交易内特征学习能力的限制，银联联合英特尔提出多层机器学习 + 深度学习模型，以技术创新大幅提升反欺诈模型的性能。

在这一创新过程中，英特尔不仅为这一新型的反欺诈模型提供了高性能处理器产品作为算力引擎，还提供了多样化、可扩展、全方位的技术支撑，为三明治结构欺诈侦测模型中每一个层面所用的方法，都提供了有针对性的优化手段和工具，从而帮助整个反欺诈模型进一步提升了效率。

基于英特尔® 至强® 处理器/英特尔® 至强® 可扩展处理器的硬件平台为中国银联反欺诈模型成功构建、应用提供的强劲算力，以及英特尔提供的多项优化措施，用户在未来也可选择性能更强、在AI领域有着更多优化方法的第二代英特尔® 至强® 可扩展处理器等更新硬件产品，来构建其性能更优的解决方案。

基于 AI 的高效信贷逾期风险预测解决方案

英特尔与金融用户协作利用 AI 模型开展信贷逾期风险预测

信贷逾期风险挑战

信贷是金融机构最重要的资产业务之一，随着各类商业银行信贷业务规模的不断扩张，逐渐增加的不良贷款不仅正逐渐侵蚀着银行的利润，而且还会占用宝贵的信贷额度，影响银行的放贷能力，使优质的项目无法获得信贷支持。

更为严重的是，当不良贷款超过一定限度，就会极大地影响业务经营与运转，为银行带来风险。 此外，不良贷款的大量发生还会诱发社会道德风险，如果处理不良贷款的力度过大又可能会引起企业连锁倒闭破产，增加财政风险和社会危机的几率。

来自中国银行保险监督管理委员会银（下简称“银保监会”）的数据显示，截止 2018 年四季度末，中国商业银行不良贷款余额 2.03 万亿元人民币，不良贷款率 1.83%[4]。因此，对信贷业务实施高效的贷前贷后风险管控，就成为银行构建风控系统的重要内容。

目前，商业银行针对信贷逾期风险预测主要有两类应用场景，一类是在贷款前就进行的贷前风险评估，其主要关注预测结果的时效性和可解释性；另一类是针对贷款发放后的贷后风险预测，其主要关注预测结果的准确率和可解释性。

对于贷前风险预测，商业银行在发放贷款前，主要是通过对企业所在行业发展特点以及企业实际经营、资产负债、信用状况等进行多方位的人工调研，以此评估贷款发放的风险等级。但这类方式通常效率低，需要耗费大量的人工时间，且伴有明显的主观判断的问题。

针对贷后风险预测，商业银行通常会通过人工方式，定期或不定期地根据借款企业所属行业及经营特点，进行现场或非现场检查，通过与贷款人沟通，对借款企业的财务信息、经营状况的分析以及贷款资金的流向监测，来掌握可能造成违约风险和信用风险的因素，防止违约贷款。

囿于人力与成本问题，这种人工方式只能每月或每季度进行一次，遇到问题也只能依赖经验层层上报，等待风险管理部门决策后采取行动。因而面临以下几个主要问题：

- 人工投入大，预测时间长。每个月银行工作人员都要逐月预测当月到期以及后面三个月到半年即将到期的贷款逾期风险，层层上报并等待决策处理，整个过程需要将近一个月；占用了贷款管理中风险控制的宝贵处理时间。
- 人工预测质量良莠不齐。有的工作人员很有经验，预测准确；有的则经验不足，无法将风险消弭在襁褓之中；
- 多种因素影响预测。在人工预测的过程中，会受到市场环境的多变性、经济活动的周期性以及银行、企业两侧信息的不对称性等因素的干扰，进而影响风险预测的时效性和准确性。

同时，人工预测除了准确性无法得到保障之外，这一工作也缺乏完整的知识体系，无法逐步通过经验的积累来提升预测的效率和准确率，因而就无法形成良性闭环。

随着信贷业务的不断扩张，商业银行传统基于人工的风险预测方式也承受着越来越大的挑战。

以上的问题，以及来自银保监会对于每月逾期情况的监察，无疑给银行带来了极大的成本和管理压力。因此，银行希望将自己丰沛的业务数据资源利用起来，通过 AI 构建更有效的信贷逾期风险预测系统。而要构建完整的信贷逾期风险预测 AI 架构，实现高准确率、低延时以及可解释的贷款逾期预测方案，就需要针对业务数据和环境数据进行分析和预测。

前者，也就是业务数据，是金融机构对企业用户的金融资产状况、未来流水、资金用途等数据的记录。目前，业界通常采用机器学习或者深度学习的方法来构建预测模型；而后者，即环境数据，对其则可以采用 NLP 的方法进行研究和预测。

在英特尔与金融用户的合作探索中，双方合作构建了基于 LSTM 和传统机器学习的混合模型，来应对用户在准确性和可解释性两方面的需求。同时，也针对环境数据的 NLP 模型的构建进行了探索。

信贷逾期风险预测模型的架构设计

■ 基于机器学习方法

基于树的机器学习方法是在信贷逾期风险预测模型上常用的技术，其预测结果通常具备较好的可解释性。其中 XGBoost 是一种重要的机器学习模型，是 boosting 的集成学习，由大量

[4] 数据源引自银保监会官方网站：
http://www.cbrc.gov.cn/chinese/newShouDoc/CDF5FDDEDAE14EFEB351CD93140E6554.html

分类回归树（Classification And Regression Tree，CART）集合而成的强分类器。

CART 回归树是在二叉树上不断根据特征进行分叉，例如当前树节点 J 是基于 a 个特征值进行分叉，则特征值小于 b 的样本划分为左子树，大于 y 的样本划分为右子树：

$$J_1(a,b) = \{X|X^{(a)} \le b\}\ (\text{左子树})$$
$$J_2(a,b) = \{X|X^{(a)} > b\}\ (\text{右子树})$$

CART 回归树实质就是在该特征维度上对样本空间进行划分，典型 CART 回归树产生的目标函数为：

$$\sum_{x_i \in J_m} (y_i - f(x_i))^2$$

回到 XGBoost，其核心思想，就是通过不断进行特征分裂来生成新的分叉树，每添加一个树，其实就是学习一个新函数来拟合上次预测的残差。因此，XGBoost 目标函数可以定义为：

$$y_i = \sum_j q_j x_{ij}$$

当有 k 个样本，其第 n 轮的模型预测结果为：

$$y_i^{(n)} = \sum_{k=1}^{n} f_k(x_i) = y_i^{(n-1)} + f_n(x_i)$$

与 GBDT 等机器学习方法相比，XGBoost 有着如下的优势：

- XGBoost 支持并行计算，可以充分利用处理器的多线程能力，尤其当其工作在英特尔® 架构平台上时，可以有效利用英特尔®AXV-512 等最新指令集带来的强大并行计算能力；
- XGBoost 在其代价函数中引入了正则化项，可以有效地控制模型的复杂度，防止模型过拟合；
- XGBoost 支持列抽样（column subsampling）方式，不仅能够防止过拟合，还能降低计算复杂度；
- GBDT 在进行优化时只用到一阶导数信息，而 XGBoost 则对代价函数进行了二阶泰勒展开，同时用到了一阶和二阶导数，有着更好的预测效果。

因此，XGBoost 机器学习模型已经被广泛地运用到信贷逾期风险预测的解决方案中，使用 XGBoost 进行贷款风险的经典预测步骤如图 2-2-1 所示，分为数据导入、数据清洗与准备、模型建立、模型评估以及模型效果对比等几个主要步骤：

图 2-2-1　使用 XGboost 进行贷款风险预测

■ 基于RNN/LSTM的深度学习方法

深度学习方法也是信贷逾期风险预测中日益广泛应用的方案，其中 RNN 是一种经典的深度学习模型。在典型的 RNN 结构中，每一个输入结合当前模型的状态都会得到输出，RNN 的主体结构 A 的输入除了来自输入层的 X，还有一个循环来提供当前时刻的状态。同时 A 的状态也会从当前一步传递到下一步。

LSTM 是 RNN 重要的衍伸模型，其可以通过特别的 “门” 结构设计来避免经典 RNN 结构中的长期依赖问题，使其大幅度提升记忆时长。基于 LSTM 的深度学习方法很适合运用在基于序列的分析工作中，也就是说，银行可以利用过去一段时间内，围绕贷款发放后的一系列特征，例如企业经营情况、账户流水等，来预测未来一段时间内贷款可能面临的逾期风险。

但纯粹的深度学习方法存在着过程缺乏可解释性的缺陷，而银行等金融机构往往需要对推理得到的结果进行解释，也就是需要理解模型依据哪些信息和条件获得了特定的预测输出结果。这样的解释能够为金融客户在改善业务流程、改进客户体验等方面提供指导。深度学习方法对于用户往往呈现的是黑盒状态，这使得可解释的深度学习方法成为今后深度学习方法实施优化的重要方向。

■ 机器学习与深度学习集成方案

为提升可解释性和准确性，需考虑采用其他方法。模型融合就是一种非常有效的技术，可以在大部分的机器学习任务中提高回归或者分类的准确性，并可以直接使用不同模型的结果文件进行融合，也可以使用一个模型的预测结果作为另一个模型的特征进行训练，然后得到新的预测结果。不同类型的模型学习训练的原理不同，所学到的知识也不一样，对其进行融合，可以提升训练的效果。例如，通过将树模型 XGBoost 和 LSTM 深度学习模型融合的方式，能使预测能力得到进一步增强，同时又使模型具备可解释性。模型融合的整体结构如下图：

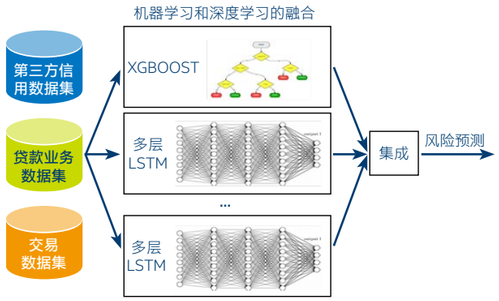


图 2-2-2　模型融合的整体结构

信贷逾期风险预测模型的算法实现

信贷逾期风险预测模型训练数据

信贷逾期风险预测模型训练数据一般包括了几年内客户在银行所做的贷款交易数据，以及当月客户本身的经营状况的量化数据。此外，人工评判的业务逻辑也会被作为高级的特征，加入到数据集中进行训练。

■ 信贷逾期风险预测模型软件平台

分布式机器学习社区（Distributed Machine Learning Community, DMLC）现在已经发布了基于英特尔® Python 分发包和面向英特尔® 架构优化的 TensorFlow 深度学习框架构建的 XGBoost 开源包。XGBoost 开源包提供一个 wrapper 类，允许模型可以与 Scikit-Learn 框架中的其他分类器或回归器协同使用。

XGBoost 可以通过英特尔® Python 分发包加速训练和推理过程。英特尔® Python 分发包内置了英特尔针对数据分析和机器学习的加速库（Intel® Data Analytics Acceleration Library, Intel® DAAL），该加速库可以加速机器学习过程，并且充分利用英特尔 ® 架构的硬件资源。

■ 面向英特尔® 架构优化的 TensorFlow

作为领先的深度学习框架，TensorFlow 已经广泛应用到不同行业的 AI 应用中。如何让 TensorFlow 框架在基于英特尔 ® 架构的平台上发挥最佳性能，就是面向英特尔® 架构优化的 TensorFlow 进行优化的方向。其优化主要从三个层面进行：

1）英特尔® MKL-DNN 的集成；

2）计算图的优化；

3）Kernel 的优化。

通过以上三个层面的优化，可以确保最常用的运算能在优化过的 MKL-DNN 基元上进行，并且可以通过算子融合的方式来优化计算图，另外还可以优化多个线程库，使它们能共存，而不是互相争夺处理器资源。通过这些软件层面的优化，在不改变神经网络模型的情况下训练和推理的整体性能获得了显著提升，具体请参考下文中的 “XGBoost 模型及训练” 介绍。

***更多英特尔® MKL-DNN的技术细节，请参阅本手册技术篇相关介绍。**

■ 数据的预处理和特征工程

数据的预处理和特征工程会根据具体的任务和数据而做不同的处理，例如进行 One-hot 编码以及数据标准化。One-hot 编码可以将类别型的变量转换成数值变量；而数据的标准化主要是为了加速模型的训练和收敛速度。

```
test_df = pd.concat([test_df,pd.get_dummies(test_df['Feature 1'], prefix='F1')],axis=1)
test_df = pd.concat([test_df,pd.get_dummies(test_df['Feature 2'], prefix='F2')],axis=1)
... ...
test_df = pd.concat([test_df,pd.get_dummies(test_df['Feature N'], prefix='FN')],axis=1)
test_df.drop(['Feature 1'], axis=1, inplace=True)
test_df.drop(['Feature 2'], axis=1, inplace=True)
... ...
test_df.drop(['Feature N'], axis=1, inplace=True)
```

类别型变量转换成数值型变量之后，会将原有的类别型变量删除，并对整个数据集进行标准化处理：

```
std_scale = preprocessing.StandardScaler()
for column in need_to_scale:
test_df[column] = std_scale.fit_transform(test_df[column].reshape(-1, 1))
print("{} mean = {}, var = {} ".format(column, std_scale.mean_, std_scale.var_))
```

■ XGBoost 模型及训练

■ XGBoost 模型实现

XGBoost 模型可以直接采用 XGBoost 开源包进行部署，或通过对原生 XGBoost 的包装类来实现，这种方式可以采用 SKlearn 风格的编程方式，更为简单易用。

```
# fit model on training data
eval_set = [(X_test, y_test)]
# hyper parameters
h_param = {'learning_rate': 0.1,
        'n_estimators': 2000,
        'max_depth': 2,
        'min_child_weight': 10,
        'gamma': 0.0,
        'subsample': 0.8,
        'colsample_bytree': 0.8,
        'objective': 'binary:logistic',
        'eval_metric': 'auc',
        'scale_pos_weight': imbalance_weight,
        'reg_lambda': 0.7,
        'reg_alpha': 1.25,
        'cv': 10}
model1 = XGBClassifier(learning_rate=h_param['learning_rate'],
                n_estimators=h_param['n_estimators'],
                max_depth=h_param['max_depth'],
                min_child_weight=h_param['min_child_weight'],
                gamma=h_param['gamma'],
                subsample=h_param['subsample'],
                colsample_bytree=h_param['colsample_bytree'],
                objective=h_param['objective'],
                eval_metric=h_param['eval_metric'],
                scale_pos_weight=h_param['scale_pos_weight'],
                reg_lambda=h_param['reg_lambda'],
```

```
                reg_alpha=h_param['reg_alpha'],
                seed=randint(1, 65535),
                cv=h_param['cv'])
xgb_param = model1.get_xgb_params()
print("------Parameters------")
print(xgb_param)
 # Fit the algorithm on the data
model1.fit(X_train, y_train, eval_metric='auc')
```

■ XGBoost 参数调优

XGBoost 的超参非常多，可以把所有的参数分为以下三类：

1) 通用参数：宏观函数控制，这部分的参数基本不需要调整；

2) Booster 参数：控制每一步的 Booster(tree/regression) 的相关参数，需要仔细调整，会影响最终的性能；

3) 学习目标参数：控制训练目标的表现，一般随任务而确定，且一般不需要调整。

所以需要调整的参数主要与 Booster 相关，参见下表：

参数	说明
max_depth	树的最大深度。树越深通常模型越复杂，更容易过拟合
learning_rate	学习率或收缩因子
n_estimators	弱分类器数目
gamma	节点分裂所需的最小损失函数下降值
min_child_weight	叶子结点需要的最小样本权重（hessian）和
subsample	构造每棵树的所用样本比例
colsample_bytree	构造每棵树的所用特征比例
colsample_bylevel	树在每层每个分裂的所用特征比例
reg_alpha	L1/L0正则的惩罚系数
reg_lambda	L2正则的惩罚系数

可以通过网格搜索的方式来逐步调整以上表格列出的 Booster 参数，网格搜索由于需要通过交叉验证的方式来选取最优参数，当多个参数同时优化时会非常耗时，所以需要逐个或者对相关参数逐组来优化。在调整中，可以通过 XGBoost 自带的 cv 函数来调整树的数目，用 XGBRegressor 或者 XGBClassifier（XGBoost 的 Sklearn 包，回归和分类问题的调整策略一致）和 GridSearchCV 调整其他参数。

下面，首先定义一个函数用于调整最优树的数目：

```
def modelFit(alg, X_train, y_train, folds, useTrain=True, early_stopping_rounds=50):
    if useTrain:
        dtrain = xgb.DMatrix(data=X_train, label=y_train)
        params = alg.get_xgb_params()
        cvResult = xgb.cv(params=params, dtrain=dtrain, num_boost_round=params["n_estimators"], folds=folds,
                early_stopping_rounds=early_stopping_rounds, metrics="rmse")
        alg.set_params(n_estimators=cvResult.shape[0])
        print("n_estimators is: {}".format(cvResult.shape[0]))
    alg.fit(X_train, y_train)
    prediction = alg.predict(X_train)
    error = mean_squared_error(y_train, prediction)
    print("mean squared error is {}".format(error))
    fealmp = pd.Series(alg.get_booster().get_fscore()).sort_values(ascending=True)
    fealmp.plot(kind="barh", title="Feature Importance")
```

第二步：定义一个基本的 XGBRegressor（可以将需要调整的参数都列出，方便后面调整），通过调用上面定义的函数得到树的数目：

```
xgbRegressor = XGBRegressor(
    n_estimators=2000,
    learning_rate=0.1,
    max_depth=6,
    min_child_weight=1,
    gamma=0,
    subsample=1,
    colsample_bytree=1,
    colsample_bylevel=1,
    reg_alpha=0,
    reg_lambda=1,
    objective='reg:linear',
    seed=123,
    n_jobs=-1)
Kfold = KFold(n_splits=5, random_state=0, shuffle=True)
modelFit(xgbRegressor, X_train, y_train, kfold)
```

得到：

```
n_estimators is: {523}
```

第三步：根据各个参数的意义分组调整，开始可以将参数调整的步长放大进行粗调，等结果出来后可以减小步长进行细调：

```
param_grid1 = {"max_depth":range(3,10,2), "min_child_weight":range(1,10,2)}
xgbRegressor1 = XGBRegressor(
    n_estimators=523,
    learning_rate=0.1,
    max_depth=6,
    min_child_weight=1,
    gamma=0,
    subsample=1,
    colsample_bytree=1,
    colsample_bylevel=1,
    reg_alpha=0,
    reg_lambda=1,
    objective='reg:linear',
    seed=123,
    n_jobs=-1)
gridcv1 = GridSearchCV(xgbRegressor1, param_grid=param_grid1, scoring="neg_mean_squared_error", cv=5, n_jobs=-1)
gridcv1.fit(X_train, y_train)
gridcv1.best_params_, gridcv1.best_score_
```

得到：

```
({'max_depth':9,'min_child_weight':5})
```

```
param_grid2 = {"max_depth":[8,9,10], "min_child_weight":[4,5,6]}
xgbRegressor2 = XGBRegressor(
    n_estimators=523,
    learning_rate=0.1,
    max_depth=6,
    min_child_weight=1,
    gamma=0,
    subsample=1,
    colsample_bytree=1,
    colsample_bylevel=1,
    reg_alpha=0,
    reg_lambda=1,
    objective='reg:linear',
    seed=123,
    n_jobs=-1)
gridcv2 = GridSearchCV(xgbRegressor2, param_grid=param_grid2, scoring="neg_mean_squared_error", cv=5, n_jobs=-1)
gridcv2.fit(X_train, y_train)
gridcv2.best_params_, gridcv2.best_score_
```

得到：

```
({'max_depth':9,'min_child_weight':6})
```

逐个调整需要优化的参数，得到最终的最优参数：gamma

```
param_grid4 = {"gamma":[x/10 for x in range(0,6)]}
xgbRegressor4 = XGBRegressor(
    n_estimators=473,
    learning_rate=0.1,
    max_depth=9,
    min_child_weight=6,
    gamma=0,
    subsample=1,
    colsample_bytree=1,
    colsample_bylevel=1,
    reg_alpha=0,
    reg_lambda=1,
    objective='reg:linear',
    seed=123,
    n_jobs=-1)
gridcv4 = GridSearchCV(xgbRegressor4, param_grid=param_grid4, scoring="neg_mean_squared_error", cv=5, n_jobs=-1)
gridcv4.fit(X_train, y_train)
gridcv4.best_params_, gridcv4.best_score_
```

得到：

```
Out[20]:({'gamma':0.1})
```

subsample	构造每棵树的所用样本比例
colsample_bytree	构造每棵树的所用特征比例
colsample_bylevel	树在每层每个分裂的所用特征比例

```
param_grid5 = {"subsample":[x/100 for x in range(70,90,5)], "colsample_bytree":[x/100 for x in range(70,90,5)],
              "colsample_bylevel":[x/100 for x in range(70,90,5)]}
xgbRegressor5 = XGBRegressor(
    n_estimators=473,
    learning_rate=0.1,
    max_depth=9,
    min_child_weight=6,
    gamma=0.1,
    subsample=1,
    colsample_bytree=1,
    colsample_bylevel=1,
    reg_alpha=0,
    reg_lambda=1,
    objective='reg:linear',
    seed=123,
    n_jobs=-1)
gridcv5 = GridSearchCV(xgbRegressor5, param_grid=param_grid5, scoring="neg_mean_squared_error", cv=5, n_jobs=-1)
gridcv5.fit(X_train, y_train)
gridcv5.best_params_, gridcv5.best_score_
```

得到：

```
Out[20]:({'colsample_bylevel':0.75,'colsample_bytree':0.85,'subsample':0.7})
```

以下两个参数可以一起调整，

reg_alpha	L1/L0正则的惩罚系数
reg_lambda	L2 正则的惩罚系数

```
param_grid7 = {"reg_alpha":[1e-3, 1e-2, 0.05, 1e-1, 0, 1], "reg_lambda":[1e-3, 1e-2, 0.05, 1e-1, 0, 1]}
xgbRegressor7 = XGBRegressor(
    n_estimators=429,
    learning_rate=0.1,
    max_depth=9,
    min_child_weight=6,
    gamma=0.1,
    subsample=0.7,
    colsample_bytree=0.75,
    colsample_bylevel=0.95,
    reg_alpha=0,
    reg_lambda=1,
    objective='reg:linear',
    seed=123,
    n_jobs=-1)
gridcv7 = GridSearchCV(xgbRegressor7, param_grid=param_grid7, scoring="neg_mean_squared_error", cv=5, n_jobs=-1)
gridcv7.fit(X_train, y_train)
gridcv7.best_params_, gridcv7.best_score_
```

得到：

```
Out[28]:({'reg_alpha':1,'reg_lambda':0.1})
```

最后再用优化的全部参数来调整一下树的数目，

```
xgbRegressor9 = XGBRegressor(
    n_estimators=2000,
    learning_rate=0.05,
    max_depth=9,
    min_child_weight=6,
    gamma=0,
    subsample=0.7,
    colsample_bytree=0.75,
    colsample_bylevel=0.95,
    reg_alpha=1,
    reg_lambda=0.1,
    objective='reg:linear',
    seed=123,
    n_jobs=-1)
modelFit(xgbRegressor9, X_train, y_train, kfold)
```

■ 模型训练和推理

模型的超参全部确定后，就可以通过指定 XGBooster 的参数生成 Booster 来训练模型，通过不同任务的相应评估标准来评估模型的训练效果。

```
model = XGBClassifier()
eval_set = [(X_test, y_test)]
model.fit(X_train, y_train)
# make predictions for test data
y_predictions = model.predict(X_test)
# evaluate predictions
accuracy = metrics.accuracy_score(y_test, y_predictions)
```

```
print("Accuracy: %.2f%%" % (accuracy * 100.0))
acc = metrics.accuracy_score(y_test,y_predictions)
recall = metrics.recall_score(y_test,y_predictions)
f1 = metrics.f1_score(y_test,y_predictions)
precision = metrics.precision_score(y_test,y_predictions)
print("ACC: %.4f" % (acc * 100.0))
print("Precesion: %.4f" % (precision * 100.0))
print("Recall: %.4f" % (recall * 100.0))
print("F1-score: %.4f" % (f1 * 100.0))
metrics.confusion_matrix(y_test,y_predictions)
```

■ 使用面向英特尔® 架构优化的TensorFlow实现LSTM

实现方法

面向英特尔® 架构优化的TensorFlow集成了英特尔® MKL-DNN, 能够在英特尔® 架构的硬件上充分利用硬件资源，通过矢量化、并行化，以及利用英特尔® 深度学习加速技术（VNNI指令集）等多种优化手段，实现对LSTM等网络模型的加速。

■ 模型架构的定义

可以采用 2 层 LSTM 网络来构建网络结构，其中一层的 LSTM 网络是由一个基本的 LSTM 层加一层 Dropout 组成，LSTM 的输出后接了三层的全连接网络。

```
def make_cell(lstm_size):
    lstm = tf.nn.rnn_cell.BasicLSTMCell(lstm_size, state_is_tuple=True)
    drop = tf.contrib.rnn.DropoutWrapper(lstm, output_keep_prob=keep_prob_)
    return drop
def LSTM(x, weights, biases):
multi_layer_cell = tf.nn.rnn_cell.MultiRNNCell([make_cell(num_hidden) for _ in range(2)],
                                               state_is_tuple=True)
    outputs, state = tf.nn.dynamic_rnn(multi_layer_cell, x, dtype = tf.float32)
    outputs = tf.transpose(outputs, [1, 0, 2])
    # We only need the last output tensor to pass into a classifier
fc_32 = tf.layers.dense(outputs[-1], 32, activation=tf.nn.relu, name='FC_32')
fc_16 = tf.layers.dense(fc_32, 8, activation=tf.nn.relu, name='FC_16')
logit = tf.layers.dense(fc_16, num_classes, name='logit')
return logit
```

定义损失函数和优化器，最小化损失函数：

```
logits = LSTM(X, weights, biases)
weighted_logits = tf.multiply(logits, class_weight)
prediction = tf.nn.softmax(weighted_logits)
# Define loss and optimizer
loss_op = tf.reduce_mean(tf.nn.softmax_cross_entropy_with_logits(
    logits=weighted_logits, labels=Y))
 optimizer = tf.train.GradientDescentOptimizer(learning_rate=learning_rate)
train_op = optimizer.minimize(loss_op)
```

■ 模型训练

根据训练服务器的配置，设置训练的系统配置参数：

```
# Initialize the variables (i.e. assign their default value)
init = tf.global_variables_initializer()
conf = tf.ConfigProto(intra_op_parallelism_threads=56,
                      inter_op_parallelism_threads=2,
                      allow_soft_placement=True)
```

模型训练和验证（可以通过 Tensorboard 来监测多个需要观察的参数），同时将训练好的模型保存下来：

```
# Start training
with tf.Session(config=conf) as sess:
    saver = tf.train.Saver()
    # Create a summary to monitor cost tensor
    tf.summary.scalar("loss_op", loss_op)
    tf.summary.scalar("accuracy", accuracy)
     merged_summary_op = tf.summary.merge_all()
     summary_writer = tf.summary.FileWriter('./tf_logs_train', sess.graph)
     test_writer = tf.summary.FileWriter("./tf_logs_test",sess.graph)
     # Run the initializer
    sess.run(init)
    i = 1
    for epoch in range(training_epochs):
        for b in range(total_batches):
            offset = (b * batch_size) % (lab_tr.shape[0] - batch_size)
            batch_x = X_tr[offset:(offset + batch_size), :]
            batch_y = lab_tr[offset:(offset + batch_size), :]
            _, c, summary = sess.run([train_op, loss_op, merged_summary_op],feed_dict={X: batch_x, Y: batch_y, keep_prob_: 0.9})
            summary_writer.add_summary(summary, i)
            i=i+1
    print("Optimization Finished!")
    saver.save(sess,"checkpoints/loan-lstm.ckpt")
#用验证集验证模型效果：
    validation_batches = (X_vld.shape[0]//batch_size)
    i = 0
    precision_score_arr = []
    recall_score_arr = []
    f1_score_arr = []
    # Calculate accuracy for validation set
    for b in range(validation_batches):
        offset = (b * batch_size) % (lab_vld.shape[0] - batch_size)
        batch_x = X_vld[offset:(offset + batch_size), :]
        batch_y = lab_vld[offset:(offset + batch_size), :]
        # Calculate batch loss and accuracy
        pred, pred_2, Y_1, loss, acc, summary_val = sess.run([prediction, pred_val, Y_, loss_op, accuracy, merged_summary_op], feed_dict={X: batch_x, Y: batch_y, keep_prob_ : 1.0})
        test_writer.add_summary(summary_val, i)
        i=i+1
        if b % display_step == 0 or b == 1:
            print("Step " + str(b) + ", Minibatch Loss= " + \
                "{:.4f}".format(loss) + ", Validation Accuracy= " + \
                "{:.3f}".format(acc))
        precision_val = precision_score(Y_1, pred_2)
        recall_val = recall_score(Y_1, pred_2)
        f1_val = f1_score(Y_1, pred_2)
        precision_score_arr.append(precision_val)
        recall_score_arr.append(recall_val)
        f1_score_arr.append(f1_val)
        print("Precision: {}".format(precision_val))
        print ("Recall: {}".format(recall_val))
        print ("f1_score: {}".format(f1_val))
        print ("confusion_matrix")
        print (confusion_matrix(Y_1, pred_2))
        fpr, tpr, tresholds = roc_curve(Y_1, pred_2)
    print("Validation precision mean = {}".format(sum(precision_score_arr)/len(precision_score_arr)))
    print("Validation recall mean = {}".format(sum(recall_score_arr)/len(recall_score_arr)))
    print("Validation f1 mean = {}".format(sum(f1_score_arr)/len(f1_score_arr)))
```

■ 模型推理

模型的推理阶段通过测试集数据给出模型的最终推断效果，需要首先载入之前保存的模型：

```
with tf.Session(config=conf) as sess:
    precision_score_arr = []
    recall_score_arr = []
    f1_score_arr = []
    # Restore
    saver.restore(sess, tf.train.latest_checkpoint('checkpoints'))
     batch_size = 100
    test_batches = (test_x.shape[0]//batch_size)
    i = 0
    print("test_batches = {}".format(test_batches))
    # Calculate accuracy for validation set
    for b in range(test_batches):
        offset = (b * batch_size) % (test_y.shape[0] - batch_size)
        batch_x = test_x[offset:(offset + batch_size), :]
        batch_y = test_y[offset:(offset + batch_size), :]
        # Calculate batch loss and accuracy
        pred, pred_2, Y_1, acc, summary_val = sess.run
        ([prediction, pred_val, Y_, accuracy, merged_summary
        _op], feed_dict={X: batch_x, Y: batch_y, keep_prob_ : 1.0})
        print("Step " + str(b) + \
              ", Test Accuracy= " + \
              "{:.3f}".format(acc))
        print("Y_1.shape = {}".format(Y_1.shape))
        print("pred_2.shape = {}".format(pred_2.shape))
        print("pred.shape = {}".format(pred.shape))
        print("Y_1: {}".format(Y_1))
        print("pred_2: {}".format(pred_2))
        print("pred: {}".format(pred))
        precision_val = precision_score(Y_1, pred_2)
        recall_val = recall_score(Y_1, pred_2)
        f1_val = f1_score(Y_1, pred_2)
        precision_score_arr.append(precision_val)
        recall_score_arr.append(recall_val)
        f1_score_arr.append(f1_val)
        print("Precision: {}".format(precision_val))
        print ("Recall: {}".format(recall_val))
        print ("f1_score: {}".format(f1_val))
        print ("confusion_matrix")
        print (confusion_matrix(Y_1, pred_2))
        fpr, tpr, tresholds = roc_curve(Y_1, pred_2)
    print("Test precision mean = {}".format(sum(precision
    _score_arr)/len(precision_score_arr)))
    print("Test recall mean = {}".format(sum(recall_score
    _arr)/len(recall_score_arr)))
    print("Test f1 mean = {}".format(sum(f1_score_arr)
    /len(f1_score_arr)))
```

贷款逾期风险混合预测模型

■ 模型简介

基于LSTM和传统机器学习的贷款逾期风险混合预测模型，融合了机器学习和深度学习两类方法的优点，既通过深度学习保障了预测的准确性，又通过机器学习的方法来提供了预测的可解释性；同时，这一混合模式还可以使用面向英特尔® 架构优化的TensorFlow和英特尔® Python分发包等先进工具和产品来实施优化。因此可以为商业银行等金融机构提供高效的预测服务。

这一模型的基本结构与工作流程如图 2-2-3 所示，首先，是特征分析和数据预处理。在这一步骤，来自金融机构本地大数据平台的数据，或者第三方提供的数据（例如征信机构的数据）会在系统中进行处理，这包括了对缺失数据的处理、对数据范围的处理、对数据不平衡性的处理以及对数据重要特征的分析。同时，随着数据集容量的增加和复杂化，该模型还可以使用不同的预处理工具包和新模型来应对各种类型的数据输入。

第二步，是利用深度学习模型（LSTM）和传统机器学习模型（XGBoost/RF）分别对样本数据进行训练和推理，并得到各自相关的结果；而后，混合模型会将分别对结果进行加权处理，更新权值并做出预测。

方案的最后一步，是将本轮的预测结果重新导入模型头部，根据预测效果更新特征值和权值，并进行下一轮的预测。

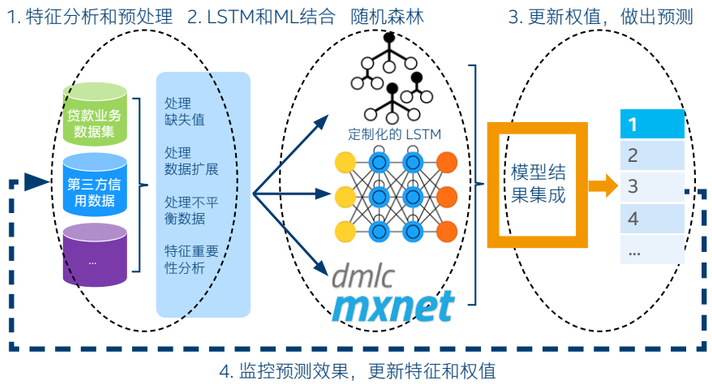


图 2-2-3 基于 LSTM 和传统机器学习的混合模型

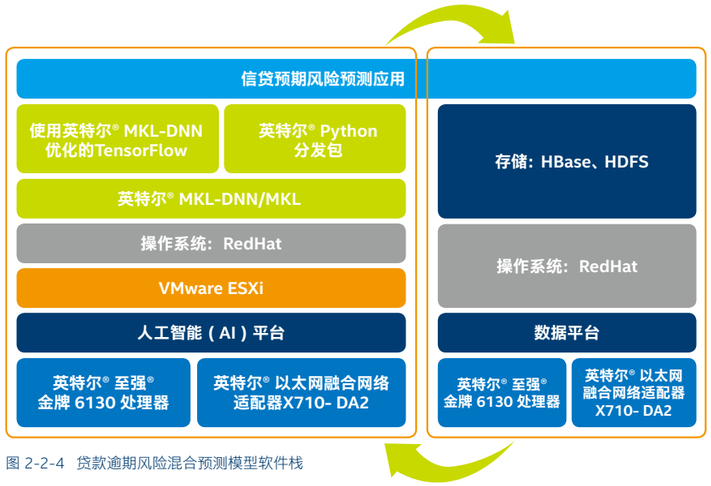


图 2-2-4　贷款逾期风险混合预测模型软件栈

■ 软件栈

贷款逾期风险混合预测模型软件栈如图 2-2-4 所示，在左侧，其底层是由英特尔® 至强® 金牌 6130 处理器和英特尔® 以太网融合网络适配器 X710-DA2 构建的硬件基础设施；其上是 AI 能力层，由 VMware ESXi 提供虚拟化环境，安装有 RedHat Linux 操作系统（CentOS Linux release 7.4.1708），并部署了英特尔® MKL-DNN 或英特尔® MKL、面向英特尔 ® 架构优化的 TensorFlow1.10 以及英特尔® Python 分发包。在右侧，底层是由英特尔® 至强 ® 金牌 6130 处理器和英特尔 ® 以太网融合网络适配器 X710-DA2 构建的硬件基础设施；其上是数据层，安装有 RedHat Linux 操作系统（CentOS Linux release 7.4.1708），并部署有 Apache HBase 分布式数据库以及 Hadoop 分布式文件系统（Hadoop Distributed File System，HDFS），提供分布式数据存储读写能力。在 AI 能力层和数据层之上，部署了贷款逾期风险混合预测应用。

■ 系统架构

在实际的系统建设中，如图 2-2-5 所示，贷款逾期风险混合预测方案可由如下方式构建。整个系统从左至右分为外部数据处理子系统、在线系统以及离线系统。首先，对于外部数据，系统通过统一的数据接口汇入数据规划与监控平台，而后由一个服务接口将部分数据送至离线系统。

其次，在离线系统中，来自外部数据子系统和在线系统的部分数据将被汇入一个数据集市（Data Mart），而后这些数据在得到清洗之后，进入离线的模型训练和算法部署流程，经训练后的模型算法将被导入在线子系统的预测系统中。

而在在线子系统中，其前置系统首先会将部分数据推给数据集市用于模型训练，其它数据则通过数据推送系统，进入由 Storm 集群构建的分布式实时计算系统进行预测调度等步骤。之后，数据就会进入预测系统进行推理预测，得到的结果将被送到测试平台进行验证，最后的结果经由名单管理模块以及风险标签生成模块，再返回离线子系统的算法部署和模型训练模块，进行算法迭代。

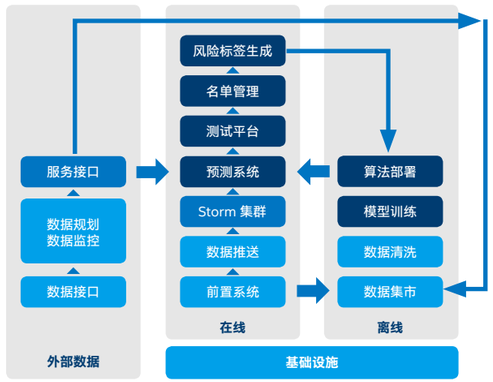


图 2-2-5　贷款逾期风险混合预测方案系统架构

软硬件配置建议

以上信贷逾期风险预测模型的构建，可以参考以下基于英特尔® 架构平台的环境进行配置：

硬件配置

名称	规格
处理器	双路英特尔® 至强® 金牌 6230 处理器或更高
基础频率	2.10GHz
核心/线程	16/32
HT	On
Turbo	On
内存	192G (16G DDR4 2666MHz x12)
硬盘	英特尔® DC S3320数据中心级固态盘 480GB
BIOS	SE5C620.86B.00.01.0013.030920180427

软件配置

名称	规格
操作系统	CentOS Linux release 7.4.1708 (Core)
Linux内核	3.10.0-957.12.1.el7.x86_64
工作负载	LSTM/XGBoost
编译器	GCC 5.4
库	英特尔® MKL最新版本
框架	面向英特尔® 架构优化的TensorFlow发布版
其他软件配置待	英特尔® Python分发包

某大型商业银行应用案例

成效

来自该用户的实际部署验证表明，最终的混合方案可以有效地提升预测的准确率，并大幅降低预测时延。数据显示，与人工预测方案相比，LSTM 方法的准确性提升 1 倍，而混合模型方案的预测准确率能够提升 2 倍以上，同时预测时延则缩短到了 2 天（预测效率提升 10 倍以上）。另外，在在线预测方案（可放贷风险预测）中，每笔预测时间均小于 1 秒，显著提升了客户满意度。

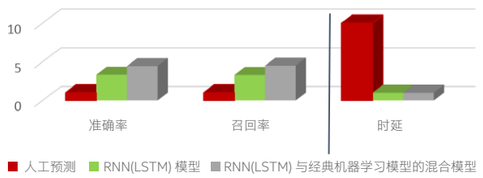


图 2-2-6　不同预测方案结果比对

该商业银行用户基于混合模型构建的训练和推理集群，全部基于英特尔® 架构平台。英特尔® 架构处理器平台不仅在内核、高速缓存等方面表现优异，还能以大量的硬件增强技术助力提升框架性能。例如，英特尔® 至强® 处理器和英特尔® 至强® 可扩展处理器所具备的英特尔® AVX-512，可以为 XGBoost 模型提供出色的并行计算能力。

小结

利用多样化的 AI 方法为金融用户提供适合的解决方案，是英特尔结合自身技术能力，面向行业推出量身定制解决方案的有益尝试。基于 LSTM 和传统机器学习的混合预测模型，不仅在预测准确性上获得了令人满意的效果，也充分满足了用户对于预测过程可解释性的要求。

英特尔不仅为这一新型的混合预测提供了高性能处理器产品，还提供了面向英特尔® 架构优化的 TensorFlow 和英特尔® Python 分发包等多样化的软件优化能力，从而有力地提升了它的工作效率。目前，该混合预测模型已在某商业银行用户处得到了实践部署，为用户带去了高效、准确的信贷逾期风险预测能力。未来，双方还计划进一步探索利用 NLP 模型面向大环境数据展开分析和预测，并使预测的效果更为全面和准确。

虽然在已有的案例方案中，采用了基于英特尔® 至强® 处理器 / 英特尔® 至强® 可扩展处理器的服务器。在未来，用户还可选择性能更强，且在 AI 领域有着更多优化措施的第二代英特尔® 至强® 可扩展处理器，来构建其解决方案。

基于 AI 的金融行业精准营销策略

基于 AI 的金融行业精准营销策略探索

背景介绍

一直以来，金融行业都是积极利用AI能力加速业务发展、提升营销效率的典范。这源于：首先，金融行业企业往往具备完备的信息化系统，并重视业务数据的采集和积累，由此积累了海量的数据，为AI应用提供了坚实的基础；其次，银行、保险、证券等金融类业务都是基于数据展开，大量繁琐的数据处理工作，亟需AI来助力提升效率；另外，深度学习的快速发展，使AI与金融行业的融合有了更多的应用场景。这其中，基于AI的金融行业精准营销策略正受到越来越多的关注。

金融行业较高的信息化水平和数据优势，推动业内企业加速进行各类推荐系统的构建，以“千人千面”、“全用户画像”等方式，推动精准营销和个性化营销等重要应用的实施。利用海量结构化/非结构化数据，金融企业正构建一系列营销决策模型，对终端用户的行为喜好、使用体验以及购买意图等做出深入分析，进而推测市场前景，为相关金融产品或商业交易提供个性化建议，为金融企业营销创新提供新鲜动力。

为迎接这一趋势，多家金融行业企业在与英特尔的合作探索过程中，通过英特尔开源的“大数据分析 +AI”平台 Analytics Zoo，已经利用神经协同过滤（Neural Collaborative Filtering，NCF）模型、宽深（Wide and Deep，WAD）等深度学习模型，构建了高效的业务推荐系统。

推荐系统

■ 常见的推荐系统

推荐系统（Recommender System，RS）是一种信息过滤工具，能引导企业以个性化的方式从大量可能的选项中发现消费者的偏好，从而改善客户消费体验、提升企业的营销效果，并在目标营销产品 / 计划的准确性方面发挥重要作用。例如，如果商家提供优惠给购买潜力最高的消费者，那么这一措施无疑是更有效的。

现在，推荐系统已经成为许多行业拓展销售和服务的关键工具。例如，有 80% 的用户在 Netflix 上通过推荐来选择接下来要观看的电影[5]；而 YouTube 上的这一数字为 60%[6]；另还有数据表明基于深度学习的推荐系统在推荐质量方面正获得越来越多的认可[7]。

推荐模型一般可分为三类，即协同过滤、基于内容和混合系统。基于协同过滤的推荐算法基于用户大概率会选择与曾经购买过的商品类似的产品，其通过学习用户与商品的历史交互，利用显式的（例如用户先前的评级）或隐式的反馈（例如用户购买后的评价）来提出建议。这一方式不需要进行特征值筛选，比较适合作为最初的模型。

而基于内容的推荐算法，其原理是假定用户通常会喜欢某项内容与所关注过的产品相似的产品，比如你买了某理财产品 A，基于内容的推荐算法发现理财产品 B，与你之前购买的理财产品 A 有类似的收益率或年限，就会向你推荐。这一方法可以避免推荐系统的冷启动问题，而其不足在于可能会重复推荐，同时这一算法也依赖大量特征值分析。

目前，深度学习正越来越多地被用于构建高效率的推荐模型。传统的机器学习算法在以前的解决方案中起着至关重要的作用，但随着模型和特征工程的日趋复杂，近年来，人们也提出了许多基于深度学习的神经推荐模型，以进一步提高营销活动的有效性。

■ 推荐系统构建过程

如图 2-3-1 所示，推荐系统的构建过程可由以下几个主要步骤组成：数据清洗、特征工程、建模、评估调优。

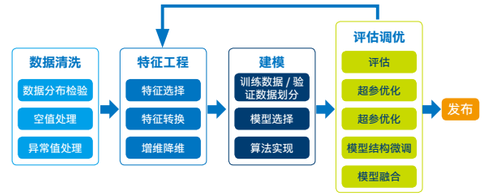


图 2-3-1　推荐系统构建过程

- 一般来说，原始数据往往都会包含各种脏数据，其会很大程度上影响模型训练和预测的准确度。数据清洗过程就是通过对数据进行重新审查和校验，来保证数据一致性。数据清洗主要包括了数据分布检验、异常值处理和空值处理等功能；

[5] Carlos A Gomez-Uribe and Neil Hunt. 2016. Netflix 推荐系统：算法、商业价值和创新。管理信息系统的 ACM 事务 (TMIS) 6, 4 (2016), 13.
[6] James Davidson, Benjamin Liebald, Junning Liu, Palash Nandy, Taylor Van Vleet, Ullas Gargi, Sujoy Gupta, Yu He, Mike Lambert, Blake Livingston, and Dasarathi Sampath. 2010. The YouTube Video Recommendation System. 第四届 ACM 推荐系统会议录第 29 页至第 29 页 (RecSys '10).
[7] Shuai Zhang, Lina Yao, and Aixin Sun. Deep learning-based Recommender System: A Survey and New Perspectives. arXiv preprint arXiv:1707.07435, 2017.

- 特征工程过程是从数据中抽取出对结果预测有用的信息，并进行维度转换，最终形成特征向量，其包括了特征选择，特征转换和增维 / 降维等主要功能；
- 建模过程包括了模型的选择、模型训练和算法实现；
- 评估与调优包括了超参优化、模型结构的调优，以及交叉验证和模型融合等工作。调优之后，还需要根据调优结果进行判断，回到初始的数据清洗和特征工程部分。

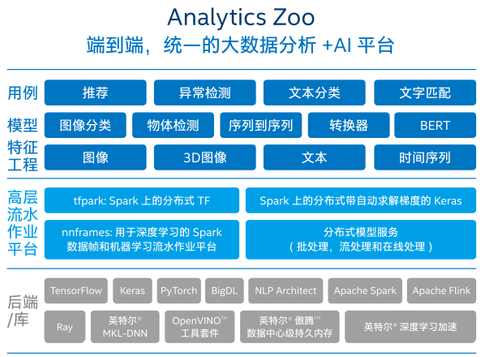
Analytics Zoo

Analytics Zoo 是由英特尔开源的、统一的 “大数据分析 +AI” 平台，它可以无缝地将 TensorFlow、Keras、PyTorch 、BigDL、Ray、Spark 以及 Flink 等软件与框架集成到一个统一的体系，并扩展到大型 Apache Hadoop/Spark 集群，用于深度学习所需的分布式训练或预测。

Analytic Zoo 可在基于英特尔® 至强® 可扩展处理器的集群上运行，来满足企业深度学习的需求。它允许用户直接在既有的大数据基础设施，例如 Apache Hadoop/Spark 上开发和运行深度学习应用程序。通过使用 Plain Old Java Object (POJO) 、本地 Java API 或 Scala/Python 模型加载 API，Analytic Zoo 可以无缝集成到 Web 服务 (如 Spark Streaming，Kafka 等) 中。

通过 Analytics Zoo，用户可以进行以下工作：

- 使用 Spark 进行数据处理和分析；
- 使用TensorFlow、Keras 或 PyTorch 进行深度学习模型开发；
- 在 Spark 和 BigDL 上进行分布式训练 / 推理；



https://github.com/intel-analytics/analytics-zoo

图 2-3-2　Analytics Zoo 提供丰富的端到端分析能力和 AI 支持

同时，Analytics Zoo 还提供了丰富的端到端分析能力和 AI 支持，包括：

- 易于使用的高级分析流水线 API (例如传输学习支持、自动编程操作、Spark DataFrame、MLPipelines、在线模型服务 API 等) ；
- 用于图像、文本、3D 图像等的常见特征工程操作；
- 大量内置深度学习模型 (例如对象检测、图像分类、文本分类、推荐、异常检测、文本匹配、序列到序列等) ；
- 丰富的参考用例 (例如异常检测、情绪分析、欺诈检测、图像相似性等) 。

利用 Analytics Zoo，企业用户可以获得以下优势：

- 分析存储在同一大数据集群上的大量数据 (HDFS、Apache HBase 和 Apache Hive 等) ，而不是移动或复制数据；
- 将深度学习功能添加到现有的分析应用程序和机器学习流水线中，而不是重建它们；
- 利用现有的大数据集群和基础设施 (资源分配，负载管理和企业级的监控) ；
- 在训练阶段进行交叉验证时，深度学习算法会产生指数性增长的隐藏嵌入特征，并自动执行内部特征选择和优化，从而显著减少特征工程工作量；在构建模型时，算法只关注一些预先定义的滑动特征和自定义重叠特征，删除大部分 LongTime Variable (LTV) 预计算工作，能够节省大量时间和资源；
- 传统的机器学习 (ML) 方法严重依赖于人机学习专家来优化模型，而 Analytics Zoo 提供了更多选项来找到一个更佳的、更稳健的执行配置，大幅提升了自动模型优化的能力；
- 由于 Analytic Zoo 可以作为英特尔® 至强® 处理器上的标准 Spark 程序运行，因此部署或操作成本为零。

* 更多 Analytic Zoo 技术细节，请参阅本手册技术篇相关介绍。

几种典型的 AI 推荐深度学习模型

■ 神经协同过滤 (NCF) 模型

NCF[8] 模型是目前常见的基于深度学习的推荐算法之一 [9]。如前所述，协同过滤算法依赖于显性反馈与隐形反馈。但在实践中，显性反馈往往并不明显，更多得到的是隐形反馈。对于隐性反馈数据，可以使用矩阵分解 (MF) 来抽象为推荐问题。但传统的 MF 模型作为潜在因素 (latent factor) 的线性模型，虽然可以反应用户与商品之间的互动关系，但不能真实地反应用户是否喜欢该商品。

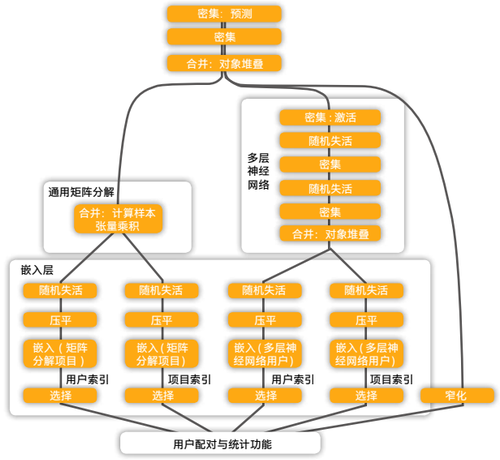


图 2-3-3　神经协同过滤 (NCF) 模型的示例

NCF 利用引入深度神经网络来解决这一问题，它可以使用深度神经网络 (DNN) 从数据中学习交互函数，从而消除 MF 模型的限制。如图 2-3-3 所示，其利用 Embedding Layer 将输入层的稀疏表示映射为一个新的潜在向量，然后分别将用户输入和商品输入送入多层神经网络结构。在左侧，模型使用了一个通用矩阵分解 (GMF) 结构用于处理线性交互；在右侧，则使用多层神经网络 (MLP) 进行处理非线性交互。最后使两者相互融合，从而获得更好的推荐效果。现在，通过 Analytics Zoo，用户就可以轻松构建 NCF 模型。

■ 宽深模型

宽深学习模型是 2016 年提出的一个 DNN-Linear 混合模型，其分为宽分量模型和深分量模型两个部分，如图 2-3-4 中模式右侧部分所示，宽分量模型是一个单层感知器，是一个广义线性模型。与传统推荐系统通过基于离散特征的线性算法来进行推荐的方式相比，宽分量模型通过获得用户的历史行为数据，例如点击哪些页面，购买过哪些商品，然后通过编码构成离散特征并进行计算。

宽分量模型推荐方式对于大规模的稀疏数据有很好的效果，而且对模型具有很强的解释性。以逻辑回归 (LR) 为例，每个离散特征都可以对应模型中的一个权重值，特征的权重值与特征对结果的影响息息相关。但宽分量模型的特征衍生需要大量人为干涉和专家经验的介入，且预测效果较差。

深分量模型是与神经协同过滤模型类似的多层感知器，它是通过深度学习获得一系列向量，并用这些向量作为特征的一部分参与到训练中。通过深分量模型产生的特征有以下优点：一方面其可以弥补人为提取特征造成的维度限制，提高预测准确性；另一方面其特征是由深度学习框架自动生成，无需人力干预，可提高训练效率。但由于深分量模型产生的向量是隐性特征，这使得预测过程往往缺乏明确的可解释性。

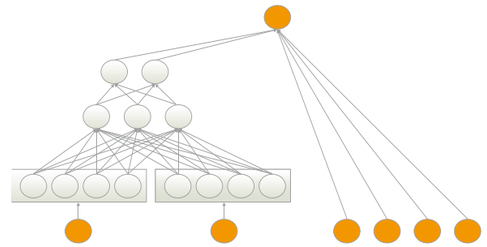


图 2-3-4　宽深模型图

因此，WAD 模型通过将宽分量模型和深分量模型进行结合，以求获得更具效率的推荐系统。WAD 模型使用了 SparseTensor，以及为稀疏数据计算明确设计的一些层，例如 SparseLinear, SparseJoinTable 等。

Analytics Zoo 对 WAD 模型提供了良好的支持，具备 DataFrame 和弹性分布式数据集 (Resilient Distributed Datasets，RDD) 两种接口，用于数据准备和训练，并为用户的不同场景提供了应用灵活性。另外，Analytics Zoo 中的 WAD 模型还允许 Spark 1.5 兼容到最新版本。

基于 Analytics Zoo 的模型实现

■ 神经协调过滤模型系统实现

以下部分将阐述如何在基于 Spark 的 Analytics Zoo 和 BigDL 上构建一个基于显式反馈的 NCF。

系统环境如下：

- Python 2.7/3.5/3.6
- JDK 8
- Spark 1.6.0/2.1.1/2.1.2/2.2.0 (与编译 Analytics Zoo 的 Spark 版本要保持一致)
- Analytics Zoo 0.5.0
- Jupyter Notebook 4.1

[8] 关于 NCF 技术描述，请参阅：https://www.comp.nus.edu.sg/~xiangnan/papers/ncf.pdf
[9] Xiangnan He, Lizi Liao, Hanwang Zhang, Liqiang Nie, Xia Hu, and Tat-Seng Chua. 2017. Neural Collaborative Filtering. In Proceedings of the 26th International Conference on World Wide Web. International World Wide Web Conferences Steering Committee, 173–182.

■ 下载或安装 Analytics Zoo

通过 pip 安装 analytics-zoo 或者下载预编译包（prebuilt package），请参考：https://analytics-zoo.github.io/master/#PythonUserGuide/install/（也可以在 https://analytics-zoo.github.io 主页的左侧索引里，选择 User Guide —>Python —> Install）

通过PIP安装后运行

用户可以轻松地使用以下命令来运行示例：

```
export SPARK_DRIVER_MEMORY=22g
jupyter notebook --notebook-dir=./ --ip=* --no-browser
```

请参阅如下地址了解更多PIP安装后的运行指南：
https://analytics-zoo.github.io/master/#PythonUserGuide/run/#run-after-pip-install

通过预编译包安装后运行

对本地模式（master=local[*]）或集群模式下的 spark 运行以下命令：

```
export SPARK_HOME=the root directory of Spark
export ANALYTICS_ZOO_HOME=the folder where you extract the downloaded Analytics Zoo zip package
  ${ANALYTICS_ZOO_HOME}/bin/jupyter-with-zoo.sh \
  --master ${MASTER} \
  --driver-cores 4 \
  --driver-memory 22g \
  --total-executor-cores 4 \
  --executor-cores 4 \
  --executor-memory 22g
```

请参阅如下地址了解更多非PIP安装后的运行指南：
https://analytics-zoo.github.io/master/#PythonUserGuide/run/#run-without-pip-install

■ 基于显式反馈的 NCF 实现

下文将描述如何建立一个神经网络推荐系统和一个基于显式反馈的 NCF。可以使用推荐系统的 API 在 Analytics Zoo 中构建模型，并使用相应的优化器来训练模型。

系统（推荐系统：原则、方法和评价[10]）通常通过系统界面提示用户为项目提供评分，以构建和改进模型。推荐的准确性取决于用户提供的评级数量。

NCF 利用多层感知器学习用户 - 项目的交互功能，同时 NCF 可以在其框架下表达和推广矩阵分解。includeMF（布尔型）是为用户提供的，用于构建一个具有或不具有矩阵分解的 NCF。

本案例中使用的数据集是 movieens-1M[11]，其包含了 6,000 个用户对 4,000 部电影的 100 万个评分，有五个等级。我们将尝试将每对（用户、电影）划分为 5 个类，并使用平均绝对误差评估算法的效果。

参考文献：

- 一种电影推荐的 Keras 实现方法，详见：https://github.com/ririw/ririw.github.io/blob/master/assets/Recommending movies.ipynb 以及 http://blog.richardweiss.org/2016/09/25/movie-embeddings.html
- NCF 相关论文，详见：https://www.comp.nus.edu.sg/~xiangnan/papers/ncf.pdf

导入必要的库：

```
In [1]:
from zoo.pipeline.api.keras.layers import *
from zoo.models.recommendation import UserItemFeature
from zoo.models.recommendation import NeuralCF
from zoo.common.nncontext import init_nncontext
import matplotlib
from sklearn import metrics
from operator import itemgetter
from bigdl.dataset import movielens
from bigdl.util.common import *
matplotlib.use('agg')
import matplotlib.pyplot as plt
%pylab inline
Populating the interactive namespace from numpy and matplotlib
```

初始化NN上下文，可以得到一个用于优化BigDL性能配置的 SparkContext：

```
In [2]:
sc = init_nncontext("NCF Example")
```

■ 数据准备：下载并读入1M大小的 movielens 数据：

```
In [3]:
movielens_data = movielens.get_id_ratings("/tmp/movielens/")
```

数据中每条记录的格式为（userid、movieid、rating_score）。用户 ID 的范围在 1 到 6,040 之间，电影 ID 的范围在 1 到 3,952 之间，评级以五星级为单位（仅限全星级），记录用户数和电影数，供以后使用。

```
In [4]:
min_user_id = np.min(movielens_data[:,0])
max_user_id = np.max(movielens_data[:,0])
min_movie_id = np.min(movielens_data[:,1])
max_movie_id = np.max(movielens_data[:,1])
rating_labels= np.unique(movielens_data[:,2])
print(movielens_data.shape)
print(min_user_id, max_user_id, min_movie_id, max_movie_id, rating_labels)
(1000209, 3)
(1, 6040, 1, 3952, array([1, 2, 3, 4, 5]))
```

[10] 详情请参阅：http://www.sciencedirect.com/science/article/pii/S1110866515000341
[11] 数据集详见 https://grouplens.org/datasets/movielens/1m/

将原始数据转换为 RDD[12] 格式的样本。在本例中，直接使用了 BigDL 的优化器来训练模型，它要求以 RDD 格式提供数据。以下示例中是一个 BigDL 数据结构，它可以分别使用 2 个 numpy 数组、feature 和 label 构建。此处的 API 接口是 Sample.from_ndarray(feature, label），用于将标签从 1 开始转换为零。

```
In [5]:
def build_sample(user_id, item_id, rating):
    sample = Sample.from_ndarray(np.array([user_id, item_id]), np.array([rating]))
    return UserItemFeature(user_id, item_id, sample)
pairFeatureRdds = sc.parallelize(movielens_data)\
    .map(lambda x: build_sample(x[0], x[1], x[2]-1))
pairFeatureRdds.take(3)
Out[5]:
[<zoo.models.recommendation.recommender.UserItemFeature at 0x11473ffd0>,
<zoo.models.recommendation.recommender.UserItemFeature at 0x124eb9110>,
<zoo.models.recommendation.recommender.UserItemFeature at 0x11473fed0>]
```

将数据随机分为序列（80%）和验证（20%）。

```
In [6]:
trainPairFeatureRdds, valPairFeatureRdds = pairFeatureRdds.randomSplit([0.8, 0.2], seed= 1)

valPairFeatureRdds.cache()
train_rdd= trainPairFeatureRdds.map(lambda pair_feature: pair_feature.sample)
val_rdd= valPairFeatureRdds.map(lambda pair_feature: pair_feature.sample)
val_rdd.persist()
Out[6]:
PythonRDD[3] at RDD at PythonRDD.scala:48
In [7]:
print(train_rdd.count())
train_rdd.take(3)
799923
Out[7]:
[Sample: features: [JTensor: storage: [ 1. 661.], shape: [2], float], labels: [JTensor: storage: [2.], shape: [1], float],
 Sample: features: [JTensor: storage: [ 1. 914.], shape: [2], float], labels: [JTensor: storage: [2.], shape: [1], float],
 Sample: features: [JTensor: storage: [1.000e+00 3.408e+03], shape: [2], float], labels: [JTensor: storage: [3.], shape: [1], float]]
```

■ 构建模型

在 Analytics Zoo 中，可以轻松地调用 Neuracf API 来构建 NCF 模型——只需要根据数据指定用户计数、项目计数和类编号，然后根据需要添加隐藏层，还可以选择在网络中包含矩阵分解。该模型中可以输入 BigDL 的优化器，或 Analytics Zoo 中的 NNClassifier。在以下案例中，演示了如何使用 BigDL 的优化器。

```
In [8]:
ncf = NeuralCF(user_count=max_user_id,
               item_count=max_movie_id,
               class_num=5,
               hidden_layers=[20, 10],
               include_mf = False)
creating: createZooKerasInput
creating: createZooKerasFlatten
creating: createZooKerasSelect
creating: createZooKerasEmbedding
creating: createZooKerasEmbedding
creating: createZooKerasFlatten
creating: createZooKerasFlatten
creating: createZooKerasMerge
creating: createZooKerasDense
creating: createZooKerasDense
creating: createZooKerasDense
creating: createZooKerasModel
creating: createZooNeuralCF
```

■ 编译模型

根据特定的优化器、损失和评估指标编译模型，优化器会在训练集上尽量减少神经网络相对于其权重 / 偏差的损失。要在 BigDL 中创建优化器，至少要指定以下参数，包括 model（神经网络模型）、criteria（丢失函数）、traing_dd（训练数据集）以及 batch size。

有关创建高效优化器的详细信息，请参阅以下编程指南和优化器手册。
编程指南：https://bigdl-project.github.io/master/#Programming Guide/optimization/
优化器：https://bigdl-project.github.io/master/#APIGuide/Optimizers/Optimizer/

```
In [9]:
ncf.compile(optimizer= "adam",
            loss= "sparse_categorical_crossentropy",
            metrics=['accuracy'])
creating: createAdam
creating: createZooKerasSparseCategoricalCrossEntropy
creating: createZooKerasSparseCategoricalAccuracy
```

■ 收集日志

可以通过 tensorboard 来查看摘要：

```
In [10]:
tmp_log_dir = create_tmp_path()
ncf.set_tensorboard(tmp_log_dir, "training_ncf")
```

■ 训练模型

```
In [11]:
ncf.fit(train_rdd,
        nb_epoch= 10,
        batch_size= 8000,
        validation_data=val_rdd)
```

■ 预测

Analytics Zoo 模型使用 model.predict（val-rdd）API 对给定的数据进行推理。返回 RDD 结果，通过 Predict_class 类返回预测标签：

[12] RDD 样本具体描述请参阅：https://bigdl-project.github.io/master/#APIGuide/Data/#sample

```
In [12]:
results = ncf.predict(val_rdd)
results.take(5)

results_class = ncf.predict_class(val_rdd)
results_class.take(5)
Out[12]:
[5, 5, 4, 4, 4]
```

在 Analytics Zoo 中，提供了 3 个独特的 API 来预测用户项目对，并为用户或给定的候选项目提出推荐：

```
In [13]:
userItemPairPrediction = ncf.predict_user_item_pair(valPairFeatureRdds)
for result in userItemPairPrediction.take(5): print(result)
UserItemPrediction [user_id: 1, item_id: 1193, prediction: 5, probability: 0.476881682873]
UserItemPrediction [user_id: 1, item_id: 2804, prediction: 5, probability: 0.451132953167]
UserItemPrediction [user_id: 1, item_id: 594, prediction: 4, probability: 0.481520324945]
UserItemPrediction [user_id: 1, item_id: 2398, prediction: 4, probability: 0.415099412203]
UserItemPrediction [user_id: 1, item_id: 1097, prediction: 4, probability: 0.453616738319]
```

为每个用户推荐 3 个项目，在 RDD 特性中给出候选项：

```
In [14]:
userRecs = ncf.recommend_for_user(valPairFeatureRdds, 3)
for result in userRecs.take(5): print(result)
UserItemPrediction [user_id: 4904, item_id: 2019, prediction: 5, probability: 0.9045779109]
UserItemPrediction [user_id: 4904, item_id: 318, prediction: 5, probability: 0.902075052261]
UserItemPrediction [user_id: 4904, item_id: 912, prediction: 5, probability: 0.866227447987]
UserItemPrediction [user_id: 3456, item_id: 1356, prediction: 5, probability: 0.832679390907]
UserItemPrediction [user_id: 3456, item_id: 1374, prediction: 5, probability: 0.799858570099]
```

为每个项目推荐 3 个用户，在 RDD 特性中给出候选项：

```
In [15]:
itemRecs = ncf.recommend_for_item(valPairFeatureRdds, 3)
for result in itemRecs.take(5): print(result)
UserItemPrediction [user_id: 195, item_id: 3456, prediction: 5, probability: 0.525387585163]
UserItemPrediction [user_id: 1926, item_id: 3456, prediction: 5, probability: 0.483191937208]
UserItemPrediction [user_id: 4298, item_id: 3456, prediction: 5, probability: 0.468448847532]
UserItemPrediction [user_id: 1271, item_id: 1080, prediction: 5, probability: 0.747303187847]
UserItemPrediction [user_id: 2447, item_id: 1080, prediction: 5, probability: 0.743132531643]
```

■ 评估

绘制训练和验证损失曲线：

```
In [16]:
#retrieve train and validation summary object and read the loss data into ndarray's.
train_loss = np.array(ncf.get_train_summary("Loss"))
val_loss = np.array(ncf.get_validation_summary("Loss"))
#plot the train and validation curves
# each event data is a tuple in form of (iteration_count, value, timestamp)
plt.figure(figsize = (12,6))
plt.plot(train_loss[:,0],train_loss[:,1],label='train loss')
plt.plot(val_loss[:,0],val_loss[:,1],label='val loss',color='green')
plt.scatter(val_loss[:,0],val_loss[:,1],color='green')
plt.legend();
plt.xlim(0,train_loss.shape[0]+10)
plt.grid(True)
plt.title("loss")
Out[16]:
Text(0.5,1,'loss')
```

绘制准确率：

```
In [17]:
plt.figure(figsize = (12,6))
top1 = np.array(ncf.get_validation_summary("Top1Accuracy"))
plt.plot(top1[:,0],top1[:,1],label='top1')
plt.title("top1 accuracy")
plt.grid(True)
plt.legend();
```

宽深网络实施案例

在下文中，将使用 Analytics Zoo 的推荐 API 建立一个宽度线性模型和一个深度神经网络，即宽深网络，并使用 BigDL 优化器来训练网络。宽深模型结合了记忆强度和广义化，可被用于一般大尺度回归和分类问题，其中包括突发输入特征（例如，带有大量可能特征值的类别特征）。

系统环境如下：

- Python 2.7/3.5/3.6
- DK 8
- Spark 1.6.0/2.1.1/2.1.2/2.2.0（需要与用来编译 Analytics Zoo 的 Spark 版本保持一致）
- Analytics Zoo 0.5.0
- Jupyter Notebook 4.1

下载或安装 Analytics Zoo 请参见第 38 页的运行指南。

■ 初始化

导入所需库

```
In [1]:
from zoo.models.recommendation import *
from zoo.models.recommendation.utils import *
from zoo.common.nncontext import init_nncontext
import os
import sys
import datetime as dt
import matplotlib
matplotlib.use('agg')
import matplotlib.pyplot as plt
%pylab inline
Populating the interactive namespace from numpy and matplotlib
```

初始化 NN 上下文，可以得到一个用于优化 BigDL 性能配置的 SparkContext：

```
In [2]:
sc = init_nncontext("WideAndDeep Example")
```

■ 数据准备

下载并读入 movielens 1M 评级数据，并了解维度：

```
In [3]:
from bigdl.dataset import movielens
movielens_data = movielens.get_id_ratings("/tmp/movielens/")
min_user_id = np.min(movielens_data[:,0])
max_user_id = np.max(movielens_data[:,0])
min_movie_id = np.min(movielens_data[:,1])
max_movie_id = np.max(movielens_data[:,1])
rating_labels= np.unique(movielens_data[:,2])
print(movielens_data.shape)
print(min_user_id, max_user_id, min_movie_id, max_movie_id, rating_labels)
(1000209, 3)
(1, 6040, 1, 3952, array([1, 2, 3, 4, 5]))
```

将评级数据转换为数据帧，将用户和项目数据读取为数据帧。将标签从 1 开始转换为零：

```
In [4]:
sqlContext = SQLContext(sc)
from pyspark.sql.types import *
from pyspark.sql import Row
Rating = Row("userId", "itemId", "label")
User = Row("userId", "gender", "age" ,"occupation")
Item = Row("itemId", "title" ,"genres")

ratings = sc.parallelize(movielens_data)\
    .map(lambda l: (int(l[0]), int(l[1]), int(l[2])-1))\
    .map(lambda r: Rating(*r))
ratingDF = sqlContext.createDataFrame(ratings)

users= sc.textFile("/tmp/movielens/ml-1m/users.dat")\
    .map(lambda l: l.split("::")[0:4])\
    .map(lambda l: (int(l[0]), l[1], int(l[2]), int(l[3])))\
    .map(lambda r: User(*r))
userDF = sqlContext.createDataFrame(users)

items = sc.textFile("/tmp/movielens/ml-1m/movies.dat")\
    .map(lambda l: l.split("::")[0:3])\
    .map(lambda l: (int(l[0]), l[1], l[2].split('|')[0]))\
    .map(lambda r: Item(*r))
itemDF = sqlContext.createDataFrame(items)
```

链接并转换数据。例如，性别将被用作分类特征，职业和性别将被用作交叉特征：

```
In [5]:
from pyspark.sql.functions import col, udf

gender_udf = udf(lambda gender: categorical_from_vocab_list(gender, ["F", "M"], start=1))
bucket_cross_udf = udf(lambda feature1, feature2: hash_bucket(str(feature1) + "_" + str(feature2), bucket_size=100))
genres_list = ["Crime", "Romance", "Thriller", "Adventure", "Drama", "Children's",
               "War", "Documentary", "Fantasy", "Mystery", "Musical", "Animation", "Film-Noir", "Horror",
               "Western", "Comedy", "Action", "Sci-Fi"]
genres_udf = udf(lambda genres: categorical_from_vocab_list(genres, genres_list, start=1))

allDF = ratingDF.join(userDF, ["userId"]).join(itemDF, ["itemId"]) \
        .withColumn("gender", gender_udf(col("gender")).cast("int")) \
        .withColumn("age-gender", bucket_cross_udf(col("age"), col("gender")).cast("int")) \
        .withColumn("genres", genres_udf(col("genres")).cast("int"))
allDF.show(5)
+-------+------+-----+------+---+----------+--------------+------+----------+
|itemId|userId|label|gender|age|occupation|         title|genres|age-gender|
+-------+------+-----+------+---+----------+--------------+------+----------+
|   26|  3391|    3|     2| 18|         4|Othello (1995)|     5|        24|
|   26|  1447|    4|     2| 18|         4|Othello (1995)|     5|        24|
|   26|  5107|    3|     1| 45|         0|Othello (1995)|     5|         5|
|   26|  2878|    3|     1| 50|        20|Othello (1995)|     5|        47|
|   26|  1527|    1|     2| 18|        10|Othello (1995)|     5|        24|
+-------+------+-----+------+---+----------+--------------+------+----------+
only showing top 5 rows
```

宽深模型共享的特殊数据特征信息及其特征生成。在这里，我们将职业性别作为广泛的基础部分，将年龄和性别作为广泛的交叉部分，将流派和性别作为指标，将用户 ID 和项目 ID 用于嵌入。

```
In [6]:
bucket_size = 100
column_info = ColumnFeatureInfo(
        wide_base_cols=["occupation", "gender"],
        wide_base_dims=[21, 3],
        wide_cross_cols=["age-gender"],
        wide_cross_dims=[bucket_size],
        indicator_cols=["genres", "gender"],
        indicator_dims=[19, 3],
        embed_cols=["userId", "itemId"],
        embed_in_dims=[max_user_id, max_movie_id],
        embed_out_dims=[64, 64],
        continuous_cols=["age"])
```

将数据转换为样本的 RDD，可以直接使用 BigDL 的优化器来训练模型，它要求以 RDD（sample）格式提供数据。一个示例是一个 BigDL 数据结构，其可以分别使用 2 个 numpy 数组、feature 和 label 构建。API 接口是 sample.from ndarray（feature，label）。宽深模型需要两个输入张量，一个是宽模型的稀疏张量，另一个是深模型的密集张量。

```
In [7]:
rdds = allDF.rdd\
.map(lambda row: to_user_item_feature(row, column_info))\
.repartition(4)
trainPairFeatureRdds, valPairFeatureRdds = rdds.randomSplit([0.8, 0.2], seed= 1)
valPairFeatureRdds.persist()
train_data= trainPairFeatureRdds.map(lambda pair_feature: pair_feature.sample)
test_data= valPairFeatureRdds.map(lambda pair_feature: pair_feature.sample)
```

■ 创建宽深模型

在 AnalyticsZoo 中，通过调用宽深 API 可以轻松构建宽深模型，仅需要根据数据指定模型类型、类编号以及特性的列信息，还可以更改网络中的其他默认参数，如隐藏层。该模型可以输入 BigDL 的优化器，或 AnalyticsZoo 的 NNClassifier。以下示例演示了如何使用 BigDL 的优化器：

```
In [8]:
wide_n_deep = WideAndDeep(5, column_info, "wide_n_deep")
creating: createZooKerasInput
creating: createZooKerasInput
creating: createZooKerasInput
creating: createZooKerasInput
creating: createZooKerasSparseDense
creating: createZooKerasFlatten
creating: createZooKerasSelect
creating: createZooKerasEmbedding
creating: createZooKerasFlatten
creating: createZooKerasFlatten
creating: createZooKerasSelect
creating: createZooKerasEmbedding
creating: createZooKerasFlatten
```

```
creating: createZooKerasMerge
creating: createZooKerasDense
creating: createZooKerasDense
creating: createZooKerasDense
creating: createZooKerasDense
creating: createZooKerasMerge
creating: createZooKerasActivation
creating: createZooKerasModel
creating: createZooWideAndDeep
```

■ **创建并优化训练模型：**

```
In [9]:
wide_n_deep.compile(optimizer = "adam",
                    loss= "sparse_categorical_crossentropy",
                    metrics=['accuracy'])
creating: createAdam
creating: createZooKerasSparseCategoricalCrossEntropy
creating: createZooKerasSparseCategoricalAccuracy
In [10]:
tmp_log_dir = create_tmp_path()
wide_n_deep.set_tensorboard(tmp_log_dir, "training_wideanddeep")
```

训练网络，直到完成，并得到一个训练完成的模型：

```
In [11]:
%%time
# Boot training process
wide_n_deep.fit(train_data,
                batch_size = 8000,
                nb_epoch = 10,
                validation_data = test_data)
print("Optimization Done.")
Optimization Done.
CPU times: user 54.3 ms, sys: 19.7 ms, total: 74 ms
Wall time: 2min 30s
```

■ **预测和推荐**

Analytics Zoo 模型使用 model.predict（val-rdd）API 来基于给定数据进行推理。返回 RDD 结果。Predict_class 类返回预测标签。

```
In [12]:
results = wide_n_deep.predict(test_data)
results.take(5)

results_class = wide_n_deep.predict_class(test_data)
results_class.take(5)
Out[12]:
[4, 2, 4, 5, 2]
```

在 Analytics Zoo 中，提供了 3 个独特的 API 来预测用户项目对，并为用户或给定的候选项目提出推荐：

```
In [13]:
userItemPairPrediction = wide_n_deep.predict_user_item_pair(valPairFeatureRdds)
for result in userItemPairPrediction.take(5): print(result)
UserItemPrediction [user_id: 5305, item_id: 26, prediction: 4, probability: 0.447520256042]
UserItemPrediction [user_id: 1150, item_id: 26, prediction: 2, probability: 0.42147180438]
UserItemPrediction [user_id: 4294, item_id: 26, prediction: 4, probability: 0.338612318039]
UserItemPrediction [user_id: 5948, item_id: 26, prediction: 5, probability: 0.385789096355]
UserItemPrediction [user_id: 3825, item_id: 26, prediction: 2, probability: 0.292931675911]
```

为每个用户推荐 3 个项目，在 RDD 特性中给出候选项：

```
In [14]:
userRecs = wide_n_deep.recommend_for_user(valPairFeatureRdds, 3)
for result in userRecs.take(5): print(result)
UserItemPrediction [user_id: 4904, item_id: 1221, prediction: 5, probability: 0.901316523552]
UserItemPrediction [user_id: 4904, item_id: 593, prediction: 5, probability: 0.890776693821]
UserItemPrediction [user_id: 4904, item_id: 913, prediction: 5, probability: 0.888917982578]
UserItemPrediction [user_id: 1084, item_id: 50, prediction: 5, probability: 0.632001161575]
UserItemPrediction [user_id: 1084, item_id: 912, prediction: 5, probability: 0.584099054337]
```

为每个项目推荐 3 个用户，在 RDD 特性中给出候选项：

```
In [15]:
itemRecs = wide_n_deep.recommend_for_item(valPairFeatureRdds, 3)
for result in itemRecs.take(5): print(result)
UserItemPrediction [user_id: 1835, item_id: 1084, prediction: 5, probability: 0.745298802853]
UserItemPrediction [user_id: 3864, item_id: 1084, prediction: 5, probability: 0.744241654873
UserItemPrediction [user_id: 5582, item_id: 1084, prediction: 5, probability: 0.739497065544
UserItemPrediction [user_id: 4511, item_id: 3764, prediction: 4, probability: 0.44239372015]
UserItemPrediction [user_id: 116, item_id: 3764, prediction: 4, probability: 0.365347951651]
```

■ **绘制收敛曲线：**

```
In [16]:
#retrieve train and validation summary object and read the loss data into ndarray's.
train_loss = np.array(wide_n_deep.get_train_summary("Loss"))
val_loss = np.array(wide_n_deep.get_validation_summary("Loss"))
#plot the train and validation curves
# each event data is a tuple in form of (iteration_count, value, timestamp)
plt.figure(figsize = (12,6))
plt.plot(train_loss[:,0],train_loss[:,1],label='train loss')
plt.plot(val_loss[:,0],val_loss[:,1],label='val loss',color='green')
plt.scatter(val_loss[:,0],val_loss[:,1],color='green')
plt.legend();
plt.xlim(0,train_loss.shape[0]+10)
plt.grid(True)
plt.title("loss")
Out[16]:
Text(0.5,1,'loss')
```

绘制精度：

```
In [17]:
plt.figure(figsize = (12,6))
top1 = np.array(wide_n_deep.get_validation_summary
("Top1Accuracy"))
plt.plot(top1[:,0],top1[:,1],label='top1')
plt.title("top1 accuracy")
plt.grid(True)
plt.legend();
plt.xlim(0,train_loss.shape[0]+10)
Out[17]:
(0, 1010)

In [18]:
valPairFeatureRdds.unpersist()
Out[18]:
PythonRDD[82] at RDD at PythonRDD.scala:48
In [19]:
sc.stop()
```

软硬件配置建议

以上基于AI的精准营销策略模型的构建，可以参考以下基于英特尔® 架构的平台，环境配置如下：

硬件配置

名称	规格
处理器	双路英特尔® 至强® 处理器E5-2650 v4或更高
基础频率	2.20GHz
基础频率	12/24
HT	BIOS 默认设置 (enabled 或 disabled 皆可)
Turbo	BIOS 默认设置 (enabled 或 disabled 皆可)
内存	384G (32G DDR4 2666MHz x12)
硬盘	21TB
BIOS	出厂设置，或后续升级的任何版本
其他硬件配置	10GbE 网络带宽

软件配置

名称	规格
操作系统	Ubuntu 14.04 LTS *最新支持的操作系统版本，请参考 https://analytics-zoo.github.io/
Linux内核	3.14
工作负载	Analytics Zoo based NCF, WAD, ALS model training & model inference.
编译器	gcc 4.8
库	Analytics Zoo- bigdl_0.6.0-spark_2.2.0（已含英特尔® MKL） • Spark MLlib 2.2.0
框架	Analytics Zoo，BigDL
其他软件配置	• Hadoop发行版本：Cloudera Distributed Hadoop（CDH）5.12.1 • Spark版本：2.2 • Java 平台，标准版开发工具包（JDK）1.8

应用案例

中国人寿上海数据中心实现寿险业务再发现

背景

作为保费收入超过 4,000 亿元人民币的超大型保险企业旗下重要一员，中国人寿上海数据中心正力图建设先进 AI 能力，助力业务人员高效地向不同客户推荐个性化的险种，从而解决因业务规模和险种规模不断扩大带来的问题。

此前，营销人员只能通过个人从业经验和目前公司的主推险种，来给客户推荐，而很少考虑到客户自身的需求。这样就带来两个问题，首先是客户的需求并没有得到真正满足；其次是可能会导致撤单或退保行为，为公司营收造成损失。

营销人员在进行险种推荐时，缺乏良好的方法论是造成这一问题的主要原因。尤其是对于没有经验的年轻营销员来说，更容易产生误导式的推销。因此，中国人寿上海数据中心计划以数据为支撑，构建基于 AI 的推荐模型，支持营销人员通过更有效率地进行险种推荐，提升客户满意度。

■ **方案与成效**

中国人寿上海数据中心业务推荐系统平台架构如图 2-3-5 所示。该平台基于 Analytics Zoo 搭建，其中大数据平台采用了 CDH 5.10 版本，通过 Sqoop 将业务系统中的数据导入 HDFS，数据清洗和部分预处理使用 Hive/Impala 进行，也可以使用 Python、Scala 进行数据预处理，然后把处理好的数据存入 IMPALA 或者 HIVE。然后，使用 Spark On Hive 以结构化形式读取数据，调用 BigDL 进行模型训练。

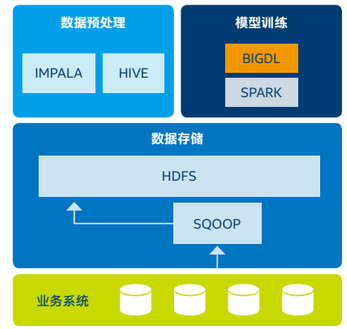


图 2-3-5 中国人寿上海数据中心业务推荐系统平台架构

基于深度学习的中国人寿上海数据中心业务推荐系统主要采用了 NCF 模型，如前文 NCF 模型相关介绍所述，模型分为左右两个部分，左侧的是通用矩阵分解，对输入向量进行乘法运算；右侧的模型是多层神经网络，将输入的 user 和 item 的特征拼接在一起，进行多层的变换，而后在上层将两个模型的结果整合，并通过 sigmoid 方法将这些特征转化成为最终的用户倾向值。在本案例中，NCF 参数设置为：

- Embedding 初始化为均值是 0, 方差为 0.01 的正态分布；
- Batch Size 设为 2800；
- 调优方法为 Adam；

模型的输出是每个用户对每个险种的评分，通过对这些评分进行逆序排序，给用户推荐得分较高的前几个险种。

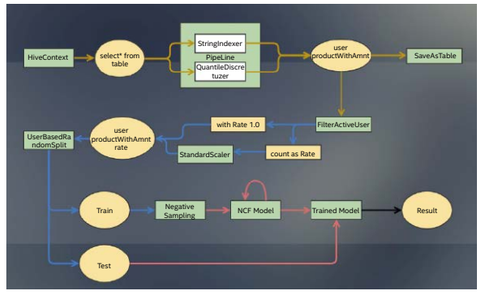


图 2-3-6　中国人寿上海数据中心推荐模型基本处理流程

如图 2-3-6 所示，推荐模型中的基本处理流程分为以下步骤：

1. 数据的预处理过程：使用 HiveContext，直接以 SQL 的方式从 Hive 中读取数据；
2. 数据读取后，将存储为 Spark 中的 DataFrame 对象；为了使数据适用于神经网络，通过 Spark 中的 PipeLine 接口，使用 String indexer 将数据映射为离散数据；
3. 取得 (user, productWithAmnt) 这样的数据组后，进行去重操作，并为每条数据加上用户购买偏好评价，例如将购买过的评价 rate 设为 1；
4. 进行训练集与测试集的划分，并通过 Spark API 在训练数据中加入负面数据；
5. 最后，训练集将会用于数据的训练，测试集用于对模型训练结果进行验证。

这些步骤通过将平台构建在 Analytics Zoo 上，借助其具备的大量高级分析流水线 API 和特性，对 Spark DataFrame、MLPipelines 等提供有力支持，有效提升整个流程的工作效率。

中国人寿上海数据中心通过两个主要的指标对推荐系统的效果进行了评估，两个指标分别是命中率 (Hit Rate) 和归一化折扣累积增益 (Normalized Distributed Cumulative Gain, NDCG)。在本案例中，如图 2-3-7 所示，中国人寿上海数据中心推荐系统的 Hit Rate 为 99.8%，NDCG 达到了 0.66，这一结果超过了预期的数值，因此可以认为，该推荐系统具有良好的效果。

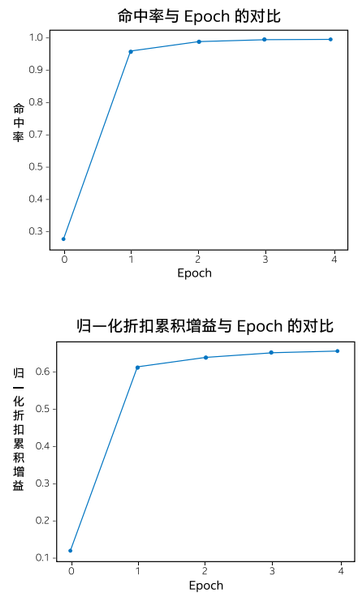


图 2-3-7　中国人寿上海数据中心推荐系统效果的评估结果

万事达卡推荐服务优化

■ 背景与挑战

作为全球领先的支付解决方案提供商，万事达卡 (MasterCard) 拥有 26 亿张信用卡，年交易量达 560 亿笔，并正通过将 AI 集成到其平台来更好地为客户服务。但在这一过程中，万事达卡也遇到了如下挑战：

- 部署时间长，大量的深度学习模块均需要在万事达卡既有系统上重建；
- 与万事达卡其他企业信息化模块兼容性差，例如无法利用现有的 ETL、数据仓库和其他分析相关的数据技术与工具集；
- 数据需要在不同模块之间频繁复制，I/O 性能成为瓶颈。

为应对这些挑战，万事达卡与英特尔开展合作，引入 Analytics Zoo “大数据分析 +AI” 平台，构建基于深度学习的推荐算法。基于最新的研究和行业实践，方案选择了 NCF 和宽深 WAD 模型作为推荐的两个候选模型，来自 Analytics Zoo 的 Keras 风格 API 也被用于基于 Python 和 Scala 构建深度学习模型。

在模型构建完成后，万事达卡利用 Analytic Zoo 的服务 API，已经将深度学习和模型服务流程嵌入到基于 Apache NiFi 构建的企业数据流水线中。

为了验证基于 Analytics Zoo 构建的深度学习推荐算法，万事达卡对 Spark 机器学习和 Analytics Zoo 的 BigDL 模型进行了基准测试，前者选择 Spark MLlib 方法的交替最小二乘法 (Alternating Least Squares，ALS) 模型。深度学习模型与 ALS 模型方法比较框图如图 2-3-8 所示。

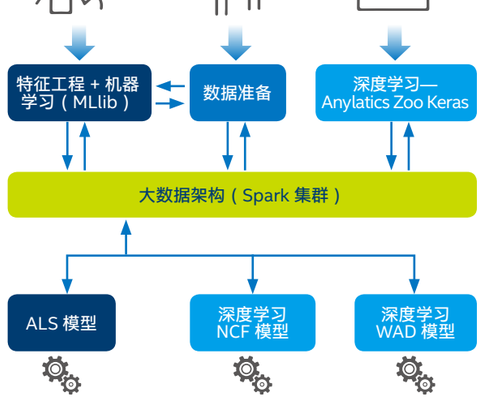


图 2-3-8 将深度学习模型与 ALS 模型进行比较

■ 方案配置与成效

对比方案中，采用过去三年万事达卡从特定渠道收集的数据集，包括了：

- 不同的合格消费者：675,000
- 用于基准的目标商家 (优惠或广告系列)：2,000
- 已知交易：14 亿 (原始数据 53 GB)
- 消费时间：12 -24 个月作为训练，1-2 个月作为验证

万事达卡推荐系统的深度学习模型与 ALS 模型的对比效果主要基于以下四个指标：

1. ROC 曲线下面积 (ROC AUG)
2. 精确度与召回率曲线下面积 (PR AUC)
3. 精准度与召回率
4. 每位客户的前 20% 精准度

从验证结果来看，深度学习模型比 ALS 模型有显著的改进，如下表所示：

	NCF 模型	WAD 模型
对比ALS，召回率改进	29% ⬆	26% ⬆
对比ALS，精准度提升	18% ⬆	21% ⬆
对比ALS，前20%精准度增长	14% ⬆	16% ⬆

表 1　深度学习模型相比 ALS 模型的改进结果

小结

金融作为一个注重数据和流程的传统行业，在多年的运作中积累了大量数据，而通过AI应用，可以从中发掘更多的价值，辅助开展各类业务，并为终端用户提供更多的个性化服务，提升用户体验。

利用Analytics Zoo提供的端到端AI与大数据分析能力，以及其中大量的模型和API，金融企业得以快速地利用自己的数据资源，在其既有大数据平台，例如Hadoop、Spark上构建基于NCF、WAD等深度学习模型的推荐系统，而无须从头建设，可大幅减少金融企业建设业务推荐系统的成本与时间。

在中国人寿上海数据中心、万事达卡等案例中，解决方案都采用了英特尔® 至强® 处理器/英特尔® 至强® 可扩展处理器为基础的硬件平台。在未来，用户还可以选择性能更强、在AI领域有着更多优化方法的第二代英特尔® 至强® 可扩展处理器等更新一代硬件产品，来构建性能更出色、AI训练/推理能力更强劲的解决方案。

加速 AI 影像分析能力
推动 AI 赋能保险行业

用 AI 加速保险行业影像分析

保险行业中的影像分析

保险行业中的各个险种都对影像分析有着巨大需求。例如，车险的投保和出险，需要被投保人在投保系统中上传身份证、行驶证、车辆合格证等证照，再由后台工作人员进行审核。常用的各类证件、签章多达数十个，全部采用人工审核不仅费时费力，也很容易出现错误。又如，日益受到关注的健康险，也需要相关核保人员判读被保险人的 X 光、CT 等影像，进而对被投保人的近期和远期健康状况做出准确评估。

增强 AI 能力，提升用户体验

目前， 包括人脸检测识别、图像分割等一系列基于影像分析的 AI 应用，正在保险行业中得到越来越广泛的应用。将 AI 影像分析应用嵌入到保险业务经营、风险管理、智能客服以及内部控制的全流程，能够有效捕捉风险、优化业务流程，实现保险行业的 AI 赋能。例如在上述的车险、健康险处理中，通过 AI 影像分析，结合 NLP 技术，可以快速筛选出必要的理赔材料，自动提取审核信息，然后通过核赔规则以及风控模型给出理赔金，自动、高效完成理赔。

针对该领域的AI应用需求，英特尔在人脸检测、比对、识别、活检等各个模块上都有相应的算法和模型可供参考。例如，由英特尔推出的OpenVino™ 工具套件已经提供了几十个预训练好的AI模型，让用户无需从零开始，即可立即构建诸如人脸检测识别等AI应用。

* 更多 OpenVino ™工具套件的技术细节，请参阅本手册技术篇相关介绍。

再以人脸活体检测为例，FeatherNet 是英特尔与华中科技大学合作，针对人脸识别反欺诈应用研发的一个轻量级卷积神经网络（Convolutional Neural Networks，CNN）。与传统 CNN 相比，它主要有两个特点：首先是以流模块（Streaming Model）替代了全局平均池化（Global Average Pooling，GAP），GAP 虽然在许多深度神经网络中可用于降维和防止过拟合，但由于其缺乏区域权重区分的能力，因此在人脸识别场景中，反而容易降低准确率。而 FeatherNet 中加入了含有 DWConv 层的流模块来替代 GAP，在准确率上获得了大幅提升。其次，FeatherNet 针对多模态数据的融合，构建了一种新的融合分类器，能够把从多模态数据中学习到的模型进行组合和级联，用来帮助模型提升准确率 [13]。

基于 ResNet 的深度学习方法

■ ResNet 简介

深度神经网络是目前 AI 影像分析中应用最广泛的网络模型之一，在经典的深度神经网络中，网络层数越多，能够提取到的不同层次的特征越丰富。同时，更深的网络能够使得提取到的特征更抽象，更富有语义信息。

但随着深度不断增加，退化（Degradation）问题也随之产生，即准确率会先上升直至饱和，而继续增加深度，却导致准确率逐渐下降。残差网络（Residual Net，ResNet）可以有效地解决这一问题。如图 2-4-1 所示，在 ResNet 中可以构成多个残差块结构，其输入与期望输出相等，构成一种恒等映射的关系。

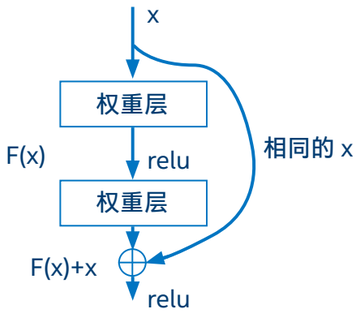


图 2-4-1　ResNet 残差块构造

这样的结构，可以让深度神经网络在不断增加深度的同时保持准确率。现在，ResNet 已经被广泛地使用在图像识别等 AI 应用场景。

■ 模型实现

面向英特尔® 架构优化的Caffe，为RESNET50网络提供了优化版本的caffe prototxt文件，位于：

1. CAFFE_HOME/models/intel_optimized_models/resnet50_v1/resnet50_fp32_acc.prototxt

使用 dummy 数据的 prototxt 文件位于：

1. CAFFE_HOME/models/intel_optimized_models/resnet50_v1/resnet50_fp32_perf.prototxt.

[13] 有关 FeatherNet 技术与性能描述请详见 https://arxiv.org/pdf/1904.09290.pdf

■ 在英特尔® 至强® 处理器平台上优化代码运行效率

■ 面向英特尔® 架构优化的Caffe的安装

基于面向英特尔® 架构优化的Caffe 1.1.6的安装方法如下：

```
git clone -b 1.1.6 --recursive https://github.com/intel/caffe.git intelcaffe
cd intelcaffe
./scripts/prepare_env.sh
mkdir build
cd build
cmake ..
build -j
```

把面向英特尔® 架构优化的Caffe的Python目录添加到Pythonpath 这个环境变量：

```
export PYTHONPATH=/... /intelcaffe/python:$PYTHONPATH
```

■ 面向英特尔® 架构优化的 Caffe 的内存优化[14]

默认情况下，面向英特尔® 架构优化的Caffe为每个层都分配了单独的输出缓冲区。由于输出缓冲区使用不同的内存地址，而不是在本地内存缓存中，在层转发中的许多内存查找都会导致潜在的缓存未命中。因此，循环缓冲区共享机制，即跨层复用预先分配的内存缓冲区的方法，在面向英特尔® 架构优化的Caffe中可被用于降低缓存丢失率。在编译阶段，通过图遍历来标识输出缓冲区的最大尺寸。在执行阶段，一旦一个层完成执行，该层的内存缓冲区将被释放并放回循环队列以供重用。

同时，在多实例执行时，也可以利用权重共享技术，来为系统提供的更好的性能表现。权重共享的机制是通过在同一个NUMA 节点内的多个进程之间，共享权重缓冲区来提高处理器三级缓存（L3 Cache）和内存之间的缓存命中率。

■ 利用 NUMA 特征来控制处理器计算资源的使用

在数据中心，通常会引入NUMA技术使众多服务器像单一系统那样运转。由于处理器访问它自己的本地存储器的速度比非本地存储器快一些。为了在这样的系统中获得更好的计算性能，需要通过一些特定指令来加以控制。numactl就是用于控制进程与共享存储的一种技术机制，它是Linux系统中广泛使用的计算资源控制方法。面向英特尔® 架构优化的Caffe在运行推理时，也可以使用numactl的命令来提高计算的效率，提升吞吐量。

具体使用方法如下所示：

```
numactl -c 0-19,40-59 caffe time --forward_only --phase TEST -model MODEL_NAME
```

在执行时，只使用了处理器 #CPU0 中的 0-19 和 40-59 核，以及与处理器 #CPU0 对应的近端内存。那么相应的，还可以在处理器 #CPU1 上面运行类似的命令：

```
numactl -c 20-39,60-79 caffe time --forward_only --phase TEST -model MODEL_NAME
```

此外，很多代码囿于并行度难以大幅提高，如果把这些任务在更少的处理器核心上面运行，效率会更高。所以，如果用 numactl 的方式绑定处理器核心来运行更多的实例，往往可以获得更高的吞吐量。尽管延时可能会有所上升，但通常还是会在应用可以接受的范围内。

基于 3D V-Net 分割网络的深度学习方法

V-Net 采用端到端训练的完全卷积网络来处理 3D 影像数据，完成影像分割工作。网络模型不再对数据进行切片，而是使用 3D 卷积网络层，直接处理三维数据。此外，它还可利用基于相似系数定制的目标函数指导网络训练，来优化训练、提升速度。

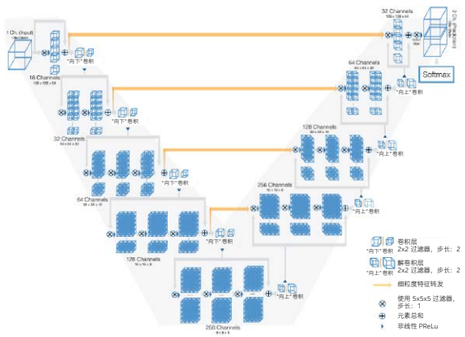


图 2-4-2　V-Net 卷积神经网络略图

将数据规模限定为 128 × 128 × 64 大小的三维数据，设 Batch Size=1，估算卷积网络的内存负载。在计算参数所占内存时需要注意考虑 bias, 故数据所需内存负载为：313.7864M × 4Bytes=1255.1456MB ≈ 1.23GB（注意这里只是前向计算过程，后向计算过程大概需要两倍于此的内存）

- 权值所需内存负载为：77.44695M × 4Bytes=309.7878MB ≈ 0.303GB；
- 整个网络模型所需的内存负载大小约为：1.23+0.303 ≈ 1.53GB；
- 如果增加一倍的内存临时储存，就需要：3.06GB。

[14] 详细技术描述请参阅：https://arxiv.org/abs/1805.08691

Layer(batch-size=1)		Data(M)	Weight(M)
Input		1.048576	0
Conv1	Conv3D11	16.77722	0.002016
	Conv3D12	16.77722	0.032016
	Conv3D13	16.77722	0.032016
Pool1	Conv3D	4.194304	0.004128
Conv2	Conv3D21	4.194304	0.128032
	Conv3D22	4.194304	0.128032
	Conv3D23	4.194304	0.128032
Pool2	Conv3D	1.048576	0.016448
Conv3	Conv3D31	1.048576	0.512064
	Conv3D32	1.048576	0.512064
	Conv3D33	1.048576	0.512064
Pool3	Conv3D	0.262144	0.065664
Conv4	Conv3D41	0.262144	2.048128
	Conv3D42	0.262144	2.048128
	Conv3D43	0.262144	2.048128
Pool4	Conv3D	0.065536	0.2624
Bottom	Conv3D51	0.065536	8.192256
	Conv3D52	0.065536	8.192256
	Conv3D53	0.065536	8.192256
Deconv4	Deconv3D41	0.524288	0.524544
	rConv3D41	0.524288	8.192256
	rConv3D42	0.524288	8.192256
	rConv3D43	0.524288	8.192256
	rConv3D44	0.524288	8.192256
Deconv3	Deconv3D31	2.097152	0.262272
	rConv3D31	2.097152	2.048128
	rConv3D32	2.097152	2.048128
	rConv3D33	2.097152	2.048128
	rConv3D34	2.097152	2.048128
Deconv2	Deconv3D21	11.01005	0.0656
	rConv3D21	11.01005	0.512064
	rConv3D22	11.01005	0.512064
	rConv3D23	11.01005	0.512064
	rConv3D24	11.01005	0.512064
Deconv1	Deconv3D11	33.55443	0.016416
	rConv3D11	33.55443	0.128032
	rConv3D12	33.55443	0.128032
	rConv3D13	33.55443	0.128032
	rConv3D14	33.55443	0.128032
Softmax	Conv3D	2.097152	0.000066
	Output	2.097152	0
总计		313.7864	77.44695

英特尔® 架构带来的性能提升

■ 英特尔® 至强® 可扩展处理器的 AI 增强特性

具有创新微架构的新一代英特尔® 至强® 可扩展处理器具有更多的内核、更高并发度的线程和更充沛的高速缓存。同时，它集成的大量硬件增强技术，特别是英特尔® AVX- 512 等技术，能够为 AI 推理过程提供强劲的并行计算能力，让用户获得更好的深度学习效果。

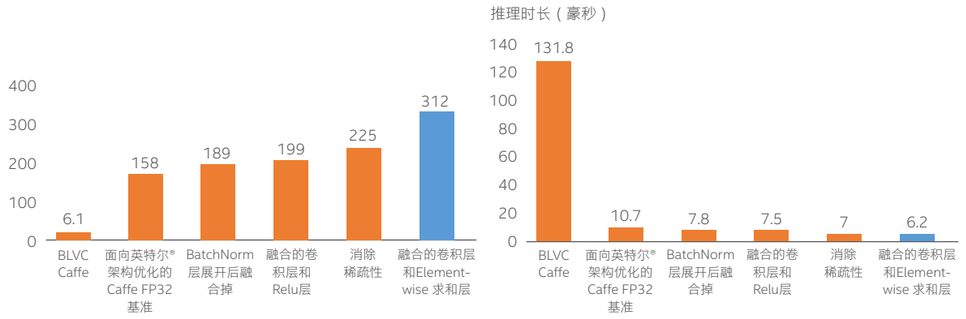


图 2-4-3　面向英特尔® 架构优化的 Caffe 在英特尔® 至强® 可扩展处理器上加入优化方案后，在推理吞吐量和推理时长性能上与 BLVC Caffe 对比

*更多面向英特尔® 架构优化的Caffe的技术细节，请参阅本手册技术篇相关介绍。

同时，英特尔® 架构处理器针对众多流行 AI 框架，诸如 BVLC Caffe、TensorFlow、Apache MXNet 等，进行了大量的优化工作。以面向英特尔® 架构优化的 Caffe 为例，其相较于 BVLC Caffe，让英特尔® 至强® 可扩展处理器的优势得到进一步释放[15]，实现了 1+1>2 的效果。

■ 面向英特尔® 架构优化的 Caffe 方法与代码

面向英特尔® 架构优化的 Caffe 通过在层内部调用英特尔® MKL-DNN 的 API 来调用优化的指令集，大幅提升程序的指令并行化效果。而英特尔® MKL-DNN 会自动调用英特尔® 至强® 可扩展处理器内置的英特尔® AVX-512 指令集以及第二代英特尔® 至强® 可扩展处理器内置的深度学习加速技术（VNNI 指令集）。

* 更多第二代英特尔® 至强® 可扩展处理器以及 VNNI 指令集的技术细节，请参阅本手册技术篇相关介绍。

层融合

层融合（Layer Fusion）技术，例如 BN+Scale，Conv+Sum，Conv+Relu，BN InPlace 以及 Sparse Fusion 等，可用来提升深度学习的性能。层融合技术与面向英特尔® 架构优化的 Caffe 框架融合，使 ResNet 等卷积神经网络在英特尔® 至强® 可扩展处理器平台上进行 2D 图像推理时，性能可媲美甚至超越现有平台。同时，它们还对可从 VNNI 指令集获得优化支持的 INT8 精度推理提供良好的支持，且框架提供的 calibration 等工具可以帮助用户将神经网络无缝切换到 INT8, 进而实现更大幅度的性能提升。

一项数据表明，与使用 BVLC Caffe 相比，面向英特尔® 架构优化的 Caffe 运行在英特尔® 至强® 可扩展处理器上的同时加入层融合技术，并使用 ResNet50 卷积神经网络，在同等测评环境中执行 AI 推理，如图 2-4-3 所示，单位时间推理性能可提升达前者的 51 倍之多，推理时长则缩短至前者的 4.7%[16]。

[15] 针对英特尔® 架构优化的 Caffe 官方网站：https://github.com/intel/caffe

[16] 该数据援引自《Highly Efficient 8-bit Low Precision Inference of Convolutional Neural Networks with IntelCaffe》一文：https://arxiv.org/pdf/1805.08691.pdf，测试配置如下：卷积模型：ResNet50，硬件：AWS single-socket c5.18xlarge。

软硬件配置建议

以上基于AI的影像分析模型的构建，可以参考以下基于英特尔®架构的平台，环境配置如下：

硬件配置

名称	规格
节点数	1
Socket	2
处理器	英特尔® 至强® 金牌6148 处理器或更高
基础频率	2.40Ghz
核心/线程	20/40
HT	On
Turbo	On
内存	192G (16G DDR4 2666MHz x12)
BIOS	1.46

软件配置

名称	规格
操作系统	Ubuntu 16.04
Linux内核	3.10.0-693.21.1.el7.x86_64
工作负载	Resnet50/VNet
基础频率	Gcc 4.8.5
深度学习加速库	英特尔® MKL-DNN 最新版本
深度学习框架	面向英特尔® 架构优化的Caffe发布版

中国平安应用案例

背景

健康险是商业保险的重要险种之一。随着人们对大病医疗重视程度的日益提高，该险种的保费规模、产品种类以及投保范围等方面也在逐步拓展。来自中国银保监会的统计数据显示，截止到 2018 年底，健康险业务原保险保费收入达 5448.13 亿元，同比增长 24.12%，占全部原保险保费收入的 14.33%[17]。

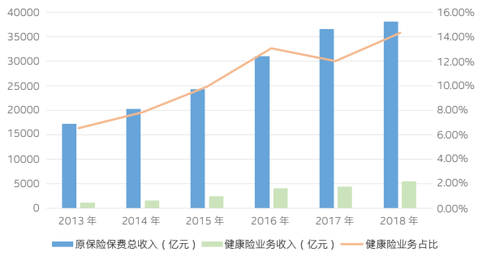


图 2-4-4　近年来国内健康险业务占比趋势

与更为成熟的保险市场相比，这一数字仍有巨大的提升空间。以美国为例，其商业健康险在保费总额中的占比为 40% 左右[18]。由此也可以预见健康险在中国巨大的市场潜力。但健康险市场在快速发展的过程中，也正受到一些短板的制约。

与其他险种相比，健康险的标的物是被保险人的健康。保险公司需要对被保险人的疾病状况和意外伤害进行准确、明晰的评估，以降低健康险经营风险、控制赔付率。然而，这一工作的技术难度、管理难度远比其他险种来得复杂，需要相关工作人员具备极专业的病理知识和实践经验。

医学影像不仅是医疗机构最常用的诊疗依据，也是保险机构判断被保险人健康状况的重要依据。虽然医学影像设备已在各级医院中得到广泛的使用，但医学影像的精准判读却面临挑战。读片医生不仅需要有临床医学、医学影像学等方面的专业知识，也必须熟练掌握放射学、CT、核磁共振、超声学等相关技能，同时，还需要具备运用各种影像诊断技术进行疾病判断的能力。因此，通常只有经验丰富的影像科医生才具备准确判读的能力。

现在，利用先进的 AI 技术来协助进行医学影像判读，不仅可对图像实施有效的分割和定位，而且可通过对图像的深层次分析，察觉肉眼难以发现的细微病理特征，从而提升各类恶性疾病的早期发现概率。因此，基于 AI 的 2D/3D 医学影像训练与推理，在有效帮助各大医疗机构提升诊疗效率的同时，也可为保险机构进行精准的健康评估提供有效手段。

解决方案

在机器学习或深度学习的概念中，由训练得到的 AI 模型被应用于新的数据，这一过程被称为推理。在医学影像判读中，利用训练得到的模型，被用于推理判断病理特征。因此，推理效率的高低直接关系到医学影像判读的效率。

本案例中，平安正利用前文介绍的在2D图像分类、检测及定位上有着非常优异特性的ResNET和前沿的3D图像分割模型V-Net分割网络，以及面向英特尔® 架构优化的Caffe等深度学习框架，对2D/3D医学影像进行AI推理。

如图2-4-5所示，平安医学影像应用的推理过程主要分为四个阶段：在前处理阶段，系统会进行医学影像切片、ROI区域选取、数据增强、归一化输入准备等操作；在推理引擎中，系统会利用ResNet网络和3D V-Net分割网络，运行在面向英特尔®架构优化的Caffe框架上，对输入的医学影像进行推理操作，并得到处理结果；在后处理阶段，系统会根据需求，进行多种形态学处理、执行预测结果合并等操作；在最后的应用层处理阶段，系统能以XML等方式，对外输出和显示结果。

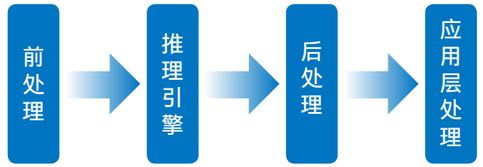


图 2-4-5　平安医学影像 AI 推理流程

利用 ResNet 网络和 3D V-Net 分割网络，平安 AI 团队开展了高效的 AI 推理工作，并取得可喜成果。如图 2-4-6 所示，在使用这些领先技术对老年黄斑变性等眼科疾病的光学相干断层扫描（Optical Coherence tomography，OCT）影像进行病灶分割后，通过结果可以清晰地发现，病患的视网膜内积液（黄线圈定部分）、视网膜下积液（红线圈定部分）以及视网膜下高反射物质（紫线圈定部分）均能被智能应用清晰地标注出来。过去需要经验丰富的医生费时费力去完成的工作，现在通过 AI 应用，仅需数秒就可以完成。

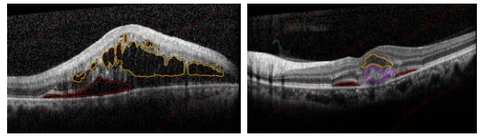


图 2-4-6　基于智能应用实现的 OCT 病灶分割结果

通过实验室以及临床的反复训练和推理，平安智能医学影像分析已在多个应用场景中获得骄人战绩。例如在肺结节检测上，肺结节的早期确认是降低肺癌死亡率的良好手段，它可以通过低剂量 CT 检查来筛查高风险人群，从而及早获知风险，但由此带来的海量 CT 影像也大大迟滞了筛查的效率，而利用 AI 来做肺结节检测，可以大大提升筛查的效率。在 2018 年初的肺结节分析（LUng Nodule Analysis，LUNA）评测中，平安不仅凭借 “平安肺结节智能读片技术” 荣获全球第一，更分别以 95.1% 和 96.8% 的精度，刷新了 “肺结节检测” 和 “假阳性筛查” 的世界纪录[19]。另一方面，高效的医学影像分析也可以准确地分析出所核实的保险是否是骗保的行为，从而大大提升了平安保险业务的反欺诈能力。

小结

基于 AI 的影像分析能够有效助力金融机构提高业务办理效率、防范欺诈风险并改善用户体验。通过 Caffe、TensorFlow 等深度学习框架，此类应用已经在保险行业的智能核保流程中，针对病理影像判读，票据处理等场景获得了广泛的使用。

将英特尔® 至强® 可扩展处理器与针对英特尔® 架构优化的深度学习框架引入这些智能应用中，不仅可以有效提升智能应用的推理效率，而且能够以更高的性价比增强应用的落地能力和可部署性，加速AI在保险行业的应用。

[17] 该数据援引自中国银保监会官网：http://bxjg.circ.gov.cn/web/site0/tab5257/info4132154.htm
[18] 该数据援引自媒体报道：http://www.chinairn.com/news/20150211/140425992.shtml
[19] 数据援引自 Luna 官网：https://luna16.grand-challenge.org/Results/

维护数据安全，打破数据孤岛 为AI应用提供更丰富数据源

借助联邦学习方法，探索多源数据在 AI 中的应用

AI 与联邦学习

■ 多样性数据集对于 AI 发展的重要意义

得益于算法、算力和数据的不断发展，人工智能（Artificial Intelligence，AI）技术也在近十年间获得巨大突破，并逐渐落地于金融、医疗、制造等行业。这其中，训练数据集的规模和质量，正深刻影响着 AI 性能的优劣。如图 2-5-1 所示，研究数据表明，用户训练数据集规模越大，所获得的训练效果也更佳[20]。

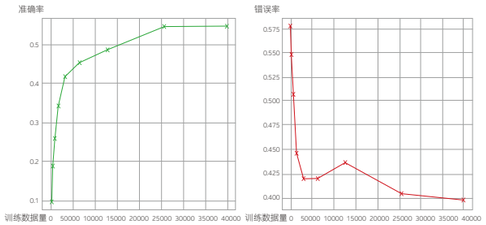


图 2-5-1 更大训练数据集带来更好训练效果

同时，一些研究也表明，更大的训练数据集，也能有效解决金融行业 AI 训练中常见的数据不平衡等问题[21]。因此可见，通过结合日趋成熟的算法、以及日渐丰沛的算力，寻求更大规模与更高质量的数据集，将成为左右 AI 效能的重要因素。

但在实际中，AI 训练所需的海量数据集往往分布在不同企业、不同部门所属的数据源中，并出于数据安全性的考虑而彼此割裂。这种数据孤岛现象，显然会带来 AI 训练在金融行业中的训练效果不佳的问题。传统上，多个金融企业或部门想要共同训练模型，需要利用分布式系统这类模式，将数据整合到其中一方，但这种简单的数据整合，既无法保证数据交互的安全性，也极大增加了数据隐私泄露的风险。

数据安全和隐私日益受到人们的关注，同时，法律法规在此方面的保护也越来越细致和成熟。例如，在 2019 年 5 月由国家互联网信息办公室会同相关部门研究起草的《数据安全管理办法（征求意见稿）》中，就对数据处理使用和数据安全监督管理提出了明确的意见要求。

现在，对多源海量高质量训练数据的渴求，和对数据安全的担忧这一矛盾，无疑已成为阻碍 AI 技术在各行各业特别是金融行业的发展与应用落地的巨大挑战。为有效应对这一挑战，2016 年，来自 Google AI 的研究人员提出了针对数据孤岛问题的用于训练深度学习网络的联邦学习（Federated Learning）方法，来满足 AI 训练可以在保证隐私和信息安全的情况下进行跨用户，跨部门、跨企业的数据使用。

■ 主流联邦学习方法

目前主流的联邦学习的基本流程如图 2-5-2 所示，以 A、B 两个金融企业对一个风控模型实施联合训练的场景为例，企业 A、B 的业务系统各自拥有大量的用户信用卡刷卡记录数据，出于数据安全考虑，这些高度敏感数据存在于各自的数据中心中，并通过防火墙实施了高等级隔离，任何直接的数据访问都会被拒绝。

通过纵向联邦学习的方式来训练这两组数据源，首先要进行数据对齐。由于不同数据源的数据样本并非雷同，因此，如图 2-5-2 中左半部分所示，联邦学习系统需要通过加密的数据实体对齐技术，在 A、B 不公开各自数据的前提下确认双方的共有数据样本（即共同用户），以便联合这些数据的特征进行建模。

通过横向联邦学习的方式来训练这两组数据源，也首先要进行特征对齐。由于不同数据源的特征维度并非雷同，因此，还是如图 2-5-2 中左半部分所示，联邦学习系统需要通过加密的数据实体对齐技术，在 A、B 不公开各自数据的前提下确认双方的共有特征维度（即共同特征），以便联合所有数据的共同特征进行建模。

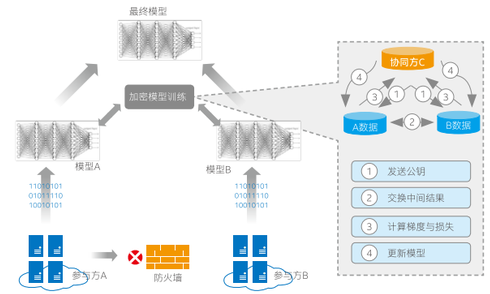


图 2-5-2 联邦学习基本架构

[20] 数据与图表援引自 De Berker, A., Predicting the Performance of Deep Learning Models, https://medium.com/@archydeberker/predicting-the-performance-of-deep-learning-models-9cb50cf0b62a
[21] 结论援引自 Juba, B. and H. S. Le, Precision-Recall Versus Accuracy and the Role of Large Data Sets, Association for the Advancement of Artificial Intelligence, 2018.

在确定共有数据样本及共同特征后，A、B 双方就可利用这些样本进行模型训练。为保证训练过程中的数据保密性，如图 2-5-2 中右半部分所示，需要借助协同方 C 进行加密训练。加密训练过程分为以下四步：

第一步：协同方 C 把公钥分发给 A 和 B，用以对训练过程中需要交换的数据进行加密；

第二步：A 和 B 之间互相以加密的形式交互用于计算梯度的中间结果；

第三步：A 和 B 分别基于加密的梯度值进行计算，并将结果汇总给协同方 C。协同方 C 通过汇总结果计算总梯度值并进行解密；

第四步：协同方 C 将解密后的梯度分别回传给 A 和 B，A 和 B 再根据梯度更新各自模型的参数。

上述训练迭代步骤将一直持续至损失函数收敛，训练过程完成并得到最终的模型。可以看出，与一般的分布式机器学习 / 深度学习方法相比，联邦学习方法具有以下特征：

- 数据不脱离本地：参与者利用自身拥有的数据训练全局模型；
- 每个参与方都参与学习过程，模型损失可控；
- 训练过程中兼顾隐私和安全，参与各方能够在不披露底层数据及其加密形态的前提下共建模型。

同时，联邦学习的另一重要特点，是其具有良好的效果激励机制，即，建立模型以后，模型的效果会在实际应用中表现出来，并记录在永久数据记录机制（如区块链）上。提供数据多的机构所获得的模型效果会更好，模型效果取决于数据提供方对自己和他人的贡献。这些模型的效果在联邦机制上会分发给各数据源，并继续激励更多数据源加入。

借由以上特点，联邦学习为各行业 AI 应用提供了跨企业、跨部门的数据使用方式和模型构建方法，实现各数据源的私密数据不出本地，只通过加密机制下的参数交换，在不违反数据隐私法规的情况下建立学习模型优化机制。

Google 的联邦学习源码可参考：https://www.tensorflow.org/federated/

正如前面例子的描述，联邦学习不只有一种方式。根据数据集的不同，联邦学习可以分为横向联邦学习（Horizontal Federated Learning）、纵向联邦学习（Vertical Federated Learning）以及联邦迁移学习（Federated Transfer Learning）三种主要方式。其中，横向联邦学习是在不同数据集的用户特征重叠较多，而用户重叠较少的情况下，将数据集按用户维度切分，并取出双方用户特征相同而用户不完全相同的那部分数据进行训练。例如，在同一家商业银行中，来自不同分行的用户的同一用户数据，就可以按照横向联邦学习方式进行训练。

纵向联邦学习，是在不同数据集的用户重叠较多而用户特征重叠较少的情况下，将数据集按照特征维度切分，并取出双方用户相同、而用户特征不完全相同的那部分数据进行训练。典型场景例如同一金融集团旗下，同一批用户在保险业务和信用卡业务中的数据，就可以按照纵向联邦学习方式进行训练。而联邦迁移学习是在用户和用户特征重叠较少的情况下，不对数据进行切分，而利用迁移学习的方法来完成数据联合训练。

■ 联邦学习方案在金融行业的应用

联邦学习从诞生伊始，就获得用户的巨大关注，并在行业用户的大力推动和不断实践下，衍生出大量行业解决方案。金融行业虽然一贯重视信息技术的发展，并在长期的经营中积累了丰厚的业务数据，但金融行业集团化、规模化经营的方式，以及金融数据的高度敏感性，也造成部门与部门之间，分公司与分公司之间，乃至金融企业与外部企业之间的天然数据孤岛。

在金融企业纷纷将智能风控、精准营销、反欺诈等 AI 应用作为其业务转型的重要引擎，联邦学习也日渐成为金融行业有效维护数据安全、打破数据孤岛，为 AI 应用提供更丰富数据源的有力抓手。

由上节的联邦学习流程我们可以得知，在金融企业利用联邦学习方法，聚合多源数据实施 AI 模型训练的过程中， AI 模型、数据或过程参数需要通过网络在各个数据节点中进行传输和交互。众所周知，数据的暴露面越大，其所面临的安全风险也越高。因此，无论是各节点中的硬件设施、操作系统等，还是路由器、网关等网络设备受到 “污染”，都有可能带来数据泄露和被篡改的安全风险。

例如，黑客可能通过在网络转发设备上安装嗅探器（Sniffer）来截取数据报文，也可能利用冷启动（Cold Boot）攻击方式来读取服务器重启后的数据残留，或者直接通过内存总线窥探、内存篡改等方法攻击内存中的数据。形形色色的攻击方法令系统防不胜防。而要构建自下而上、涵盖软、硬件和操作系统的安全防护机制，不仅会带来巨大的资源消耗，增加用户的总拥有成本（Total Cost of Ownership，TCO），实际防护效果也未必尽如人意。

因此，企业构建联邦学习系统的最核心的环节，是为用户打造高效可信的数据共享方式。目前，基于硬件可信执行环境（Trusted Execution Environment，TEE）技术的可信计算解决方案越来越受到金融行业的青睐。其核心概念为，以第三方硬件为载体，为不同的数据源提供安全可信的环境。如图 2-5-3 所示，是采用可信执行环境（TEE）之后建议采用的联邦学习架构。如图所示，来自 A、B 不同数据源的数据，可以在上方由硬件创建的 TEE 环境中进行共享和模型训练。

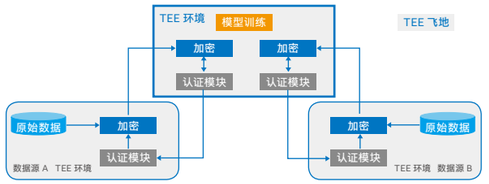


图 2-5-3 TEE 方案架构

作为 TEE 方案技术实现的典型代表，英特尔® 软件防护扩展（Intel® Software Guard Extensions，英特尔® SGX）技术通过在特定硬件（例如内存）中构造出一个可信的 “飞地”（Enclave），如图 2-5-4 所示，使数据和应用程序的安全边界仅限于飞地本身以及处理器内，同时，其运行过程也可不依赖于其他软、硬件设备。这意味着，数据的安全保护是独立于软件操作系统或硬件配置之外，即便在硬件驱动程序、虚拟机乃至操作系统均受到攻击破坏的情况下，也能有助于防止数据泄露。此外，作为硬件级的安全技术，SGX/TEE 的技术方案具有其他技术所不可比拟的高效率，其对应带来的经济性和实用性是企业构建联邦学习系统做方案选择时最看重的因素。

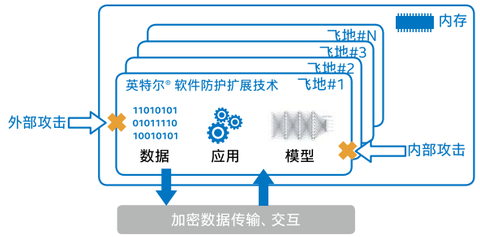


图 2-5-4 英特尔® SGX 技术以可信 “飞地” 来确保数据安全

现在，英特尔正率先与众多金融领域合作伙伴一起，在安全可信的环境中，利用多源数据协同实施 AI 训练。这一探索现在已在实践中取得了丰硕的实践成果，并在众多项目中赢得了用户的良好反馈。

英特尔® SGX 技术

■ 英特尔® SGX 技术简介

英特尔于 2013 年推出的英特尔® SGX 技术，是通过一组新的指令集扩展与访问控制机制，实现不同应用程序间的隔离运行，从而增强应用程序代码和数据的安全性，为它们提供更强的保护性来防止信息泄漏或被篡改。

传统上，数据的隐私保护和安全防护大都是工作在操作系统或软件层面，但是当操作系统或软件也受到了 “感染” 时，数据的安全性就变得岌岌可危。如图 2-5-5 所示，虽然应用程序可以通过安全扫描，防火墙等对来自外部黑客或应用程序的攻击进行防护，但是恶意软件、恶意代码如果利用操作系统漏洞，就可以绕过这些防护，直接攻击关键的隐私数据。

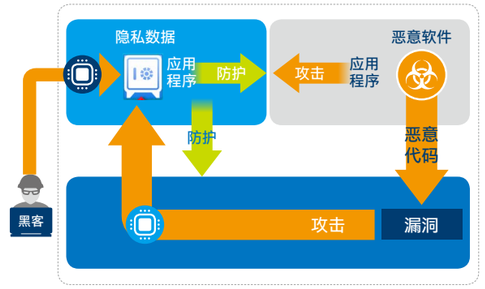


图 2-5-5 被实施内部攻击的应用程序

而英特尔® SGX 技术的特性，是允许开发人员可将敏感信息或应用程序置于飞地中。飞地是在特定硬件（例如内存）中划出的，具有更强安全保护的执行区域，其不依赖于固件和软件的安全状态，拥有基于硬件的机密性和完整性，因此可以帮助系统阻止来自更高权限进程的访问。

因此，英特尔® SGX 可以为用户提供以下主要特性：

a) 增强的保密性和完整性

如图 2-5-6 所示，飞地工作在隔离的硬件环境（支持 SGX 技术的英特尔架构处理器、内存）中，并通过密钥对应用系统和数据实施鉴权，即使在操作系统（OS）、BIOS 或虚拟机（VMM）等中存在高权限恶意软件或恶意代码，也很难对数据实施攻击；

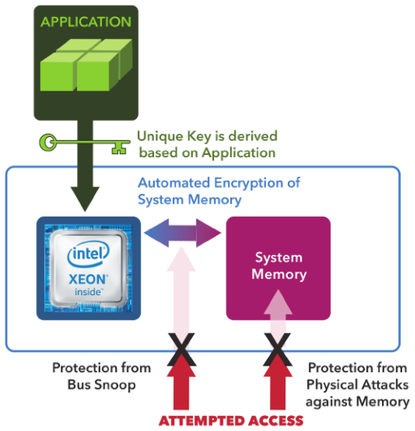


图 2-5-6　SGX 具有增强的保密性和完整性

b) 更小的安全攻击面

如图 2-5-7 所示，SGX 技术将应用程序与敏感数据限定运行在受保护的硬件飞地中，杜绝了传统上恶意程序可能从硬件、虚拟机和操作系统发起的攻击，更小的攻击面带来了更高的安全性；

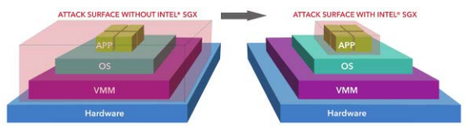


图 2-5-7　SGX 具有更小的安全攻击面

c) 远程鉴权和控制能力

用户可以通过执行远程鉴权，向各参与方证明运行环境实在合法的飞地里进行加载的；这样，可以更安全地将密钥、凭据和其他敏感数据提供给飞地；

d) 更低学习曲线

采用 SGX 技术的相关应用程序都可以基于英特尔® 处理器开发、集成和执行，开发人员无需熟悉额外的软硬件环境，学习曲线更低。

■ 英特尔® SGX 安装与配置

用户可以通过引入英特尔® SGX SDK 来创建基于 SGX 技术的解决方案，该 SDK 提供了以下内容：

- API
- 函数库
- 文档
- 样本源码
- 工具

您可以访问以下链接获得最新的英特尔® SGX SDK：

基于 Windows 系统的 SDK 下载地址

https://registrationcenter.intel.com/en/forms/?productid=2614

基于 Linux 系统的 SDK 下载地址

https://01.org/intel-software-guard-extensions/downloads

要使用英特尔® SGX 技术，用户需要首先确认当前使用的 CPU 支持该技术，并且在 BIOS 菜单中确认 SGX 已设为 “enabled” 状态。然后，用户可参阅 SDK 附带的安装指南来完成 SDK 安装过程。

在不支持英特尔® SGX 的硬件平台上，用户也可以安装 SDK 并使用模拟模式（simulation mode）来进行 SGX 应用程序的开发工作，应用程序在模拟模式下运行时的行为和支持 SGX 的真实硬件环境下基本一致，但在此模式下运行应用程序不会得到 SGX 提供的实际保护。

■ 英特尔® SGX鉴权方法

鉴权是英特尔® SGX 技术使用中，保护数据私密性和安全性的关键步骤，其可以为系统提供以下三种安全能力：

- 飞地中应用程序，或数据的身份认证；
- 飞地中未测量状态（例如执行模式）的详细信息；
- 飞地中是否存在应用程序，或数据被篡改的可能。

目前英特尔® SGX 提供了两类远端鉴权方法，英特尔® EPID（Intel® Enhanced Privacy ID）和英特尔® SGX DCAP（Intel® Software Guard Extensions Data Center Attestation Primitives）。下面分别对这两种方法做一个简单介绍。

基于英特尔® EPID 的远端鉴权

基于英特尔® EPID 的远端鉴权方法是通过英特尔® EPID 签名来对飞地中的应用程序或数据实施鉴权，其可以最大限度地降低使用英特尔® SGX 可信计算基础（TCB）的平台，在处理多个安全性版本时的复杂性。

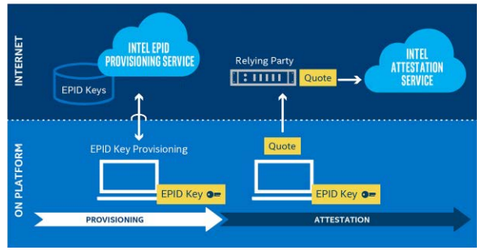


图 2-5-8　基于英特尔® EPID 的远端鉴权

如图 2-5-8 所示，这一鉴权方法是为设备提供了授权后的 EPID 密钥，如果设备使用的私钥被窃取或泄露，EPID 系统会自动识别出此情况，并吊销该设备以防止信息的泄露。

更多英特尔® EPID 安全性技术具体请参阅：

https://software.intel.com/zh-cn/articles/intel-enhanced-privacy-id-epid-security-technology

相关代码样本请参阅：

https://software.intel.com/zh-cn/articles/code-sample-intel-software-guard-extensions-remote-attestation-end-to-end-example

基于英特尔® SGX DCAP的远端鉴权

基于英特尔® SGX DCAP 的远端鉴权允许用户构建并部署自己的鉴权服务，这对于满足以下企业、数据中心和云服务提供商的要求：

- AI 服务所在的网络环境无法访问互联网服务；
- 用户具有严格的内部鉴权机制；
- 在特殊网络、架构中部署中部署的 AI 服务，例如对等网络。

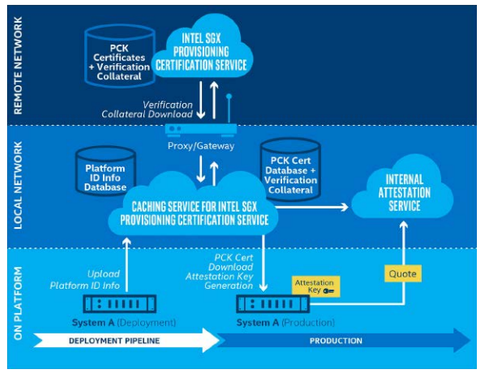


图 2-5-9 基于英特尔® SGX DCAP 的远端鉴权

如图 2-5-9 所示，与基于英特尔® EPID 的远端鉴权方法相比，DCAP 方法中一次完整的鉴权过程可在用户自身的网络环境中完成，仅在服务器部署阶段和 TCB Recovery 阶段才需要从外部获取鉴权信息。因此，在数据中心环境中，DCAP 方法具有更低的延时和更灵活的网络环境适应性。

用户可以参阅以下的技术文档获得更多细节：

- 数据中心鉴证定向指南：https://download.01.org/intel-sgx/dcap-1.1/linux/docs/Intel_SGX_DCAP_ECDSA_Orientation.pdf
- 基于 Linux 系统的安装程序和文档：https://01.org/intel-software-guard-extensions/downloads
- 支持英特尔® SGX DCAP 的第三方鉴证：https://software.intel.com/sites/default/files/managed/f1/b8/intel-sgx-support-for-third-party-attestation.pdf
- 英特尔® SGX DCAP 源码（含例子程序）：https://github.com/intel/SGXDataCenterAttestationPrimitives

■ 基于英特尔® SGX的应用开发和移植

在开发基于 SGX 的应用程序时，用户可以使用 SGX SDK 提供的可信运行库（Trusted Libraries）。可信运行库中包括了 C/C++ 的大部分 API，以及受保护的文件系统、常用密码学操作、多线程支持等功能。要查看 SGX SDK 提供的所有 API，可查阅 SGX 开发人员参考手册。

Windows 版 SGX 开发人员参考手册

https://software.intel.com/zh-cn/download/sgx-sdk-developer-reference-windows

Linux 版 SGX 开发人员参考手册

https://01.org/intel-softwareguard-extensions/documentation/intelr-software-guard-extensions-documentation

对于 AI 类型的应用程序开发，SGX SDK 专门提供了用于深度神经网络的英特尔® DNNL（Deep Neural Network Library），使得用户在 SGX 的可信执行环境中也能调用 DNNL 的标准 API。用户可将已有的、基于 DNNL API 开发的深度神经网络应用程序容易地移植到 SGX 飞地中运行。

SGX DNNL 开源代码：

https://github.com/intel/linux-sgx/tree/master/external/dnnl

在开发基于 SGX 的 AI 应用时，用户通常用以下方式保护数据的机密性：

- 负责计算的节点 1 在 SGX 飞地中产生一个安全的随机数，以此作为密钥。节点 1 运行的代码不能将密钥、用密钥解密出的数据和计算时的中间结果传递到飞地外；

- 节点 1 使用一种远端鉴权方式，产生报告证明自己的软硬件环境是安全的。然后，节点 1 向另一个提供数据或梯度的节点 2 发送数据请求；
- 节点 2 验证节点 1 的报告，确定节点 1 是安全的。同时，节点 2 用其他方法确定节点 1 有权访问该数据；
- 节点 2 用节点 1 提供的密钥加密数据并发送，供节点 1 解密后使用。

以联邦学习为例，使用以上方式进行设计，一个联邦学习应用可以在 SGX 中实现以下工作流程：

1. 应用程序创建一个 SGX 飞地的实例，在其中运行自身的可信部分（trusted code）。
2. 应用程序的可信部分随机产生一个非对称密钥对，将私钥保留在 SGX 飞地中。
3. 应用程序使用远端鉴权产生一个可验证的软硬件环境报告，其中包含 2 中产生的公钥，作为当前节点的身份。
4. 应用程序将远端鉴权的报告和公钥发送给网络中的其他节点。其他节点在验证当前节点的报告后，记录下当前节点的公钥作为可信的参与方之一。
5. 重复 1-4 的过程，直到所有节点都记录下其他节点的公钥。
6. 节点之间按照联邦学习算法完成加密的数据对齐和梯度交换步骤。每个节点使用其他节点的公钥，用于加密后续通信中的数据。

■ 英特尔® SGX与Graphene的集成

英特尔® SGX 技术为用户带来了基于硬件环境的安全机制，使其中基于关键数据的 AI 训练和推理变得更为安全可靠。但在实际部署中，用户可能面临一个新的问题，对于新的 AI 应用，用户可以在代码构建之初就可如上一节所述，与 SGX SDK 提供的可信运行库进行集成。但在有些 AI 应用开发过程中，用户会重用一些已有的开源框架或算法实现，例如基于 Python 和 TensorFlow 实现的神经网络模型，此时就需要对相关应用进行大量的移植工作。

图 2-5-10　应用程序与英特尔® SGX 技术的集成

如图 2-5-10 所示，用户首先需要基于英特尔® SGX 技术，对应用源代码（基于 C++、Python 等语言编写）进行修改移植，然后与英特尔® SGX SDK 提供的可信运行库进行编译集成形成应用程序，最后再去硬件可信环境中执行。这一过程不仅需要工程师拥有娴熟的英特尔® SGX 移植技巧和经验，同时还需要工程师能够对进行移植的应用程序（如开源框架、算法实现等）的源代码保持相当的熟悉度，而在一些金融企业中，许多应用的开源框架或算法实现都已经有了较长历史，因此源代码修改移植的工作量会非常巨大。

为减少上述应用程序移植的工作量，避免用 C/C++ 语言重新编写这些代码，用户可选择使用一些支持英特尔 ® SGX 的 Library OS 来运行这些已有的代码，例如 Graphene（https://github.com/oscarlab/graphene）、Occlum、Scone、Anjuna 等。需要注意的是，如果用户选用基于 Library OS 的方案，还需要正确配置 Library OS 中的安全选项，并评估载入的现有代码，否则不正确的软件行为会影响应用的性能、安全性或稳定性。

作为英特尔® SGX 重要的开源兼容性工具，Graphene 可以通过对动态加载库，动态链接、多进程抽象以及文件认证等的支持，使用户可在 Graphene SGX 环境中直接运行原始应用。以运行 TensorFlow 框架为例，如图 2-5-11 所示，Graphene SGX 环境可以基于硬件中的 TEE 环境构建，其上用户可通过创建一个在 Graphene SGX 环境中运行的 Python 运行环境，然后在这个环境中运行 TensorFlow 和用户需要的神经网络模型代码。同样这些应用也可以在 C++ 等环境中运行，而在深度学习方法中常用的 OpenVINO™ 工具套件、Analytic Zoo 平台等，也可在这一环境中便捷地运行。

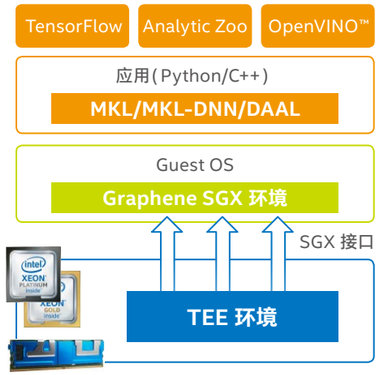


图 2-5-11　基于 Graphene SGX 环境运行应用程序

这一过程中，用户无需修改相应的模块代码，即可在 SGX 环境中执行所需的计算工作。而 Graphene 会在不受信任的主机接口上执行加密和语义检查，开发人员只需提供一个清单文件来配置应用程序环境和隔离策略，其余都可由 Graphene 来自动完成。

■ 支持英特尔® SGX 技术的处理器

值得一提的是，随着金融行业中，基于云服务（包括公有云、私有云和混合云）的 AI 应用变得愈来愈广泛，而既有云服务器硬件设备未必全部支持英特尔® SGX 技术，这给企业使用 SGX 技术带来了障碍。企业要么需要耐心等待下一个硬件更新周期，要么需要付出额外的采购成本。为解决这一难题，英特尔正逐步在其处理器中集成 SGX 技术，为用户提供了快速部署 SGX 技术的能力。

英特尔® 至强® E-2200 处理器是英特尔旨在帮助企业构建高效可靠的 IT 系统（包括云服务），推动业务持续性增长的处理器平台，其共有 12 种型号，SKU 指定为 E-22xxG，核心 / 线程数量从 4 /4 到 8/16 不等。就最高处理速度而言，两个高端 SKU 的最大睿频均为 5.0 GHz。同时，英特尔® 至强® E-2200 处理器集成了英特尔® SGX 技术来提供硬件增强型安全。

对于用户而言，第二代智能英特尔® 至强® E 处理器带来的优势包括以下 4 个主要方面：

- **性能**：根据英特尔的测试，第二代智能英特尔® 至强® E 处理器的性能是 2015 年推出的入门级服务器性能的 2 倍
- **可扩展性**：第二代智能英特尔® 至强® E 处理器的最大内存增加了一倍，达到 128GB，最大内存带宽增加了约 20%，达到 41.6 GB/ 秒。
- **可靠性**：服务器管理功能内置在采用了英特尔® 主动管理技术（英特尔® AMT）的第二代智能英特尔 ® 至强 ® E 处理器中，并且通过新的固件更新，提供了英特尔服务器平台服务和节点管理器。
- **安全性**：得益于英特尔® SGX，硬件增强型安全是新款处理器的重要组成部分。

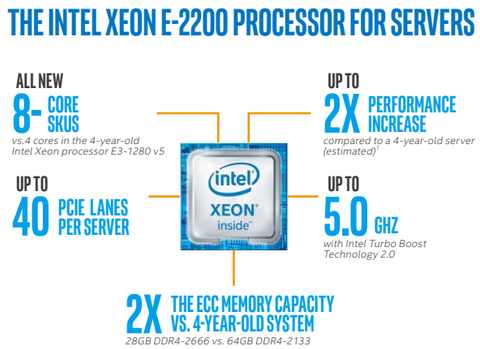


图 2-5-12　英特尔推出的英特尔® 至强® E-2200 处理器

除此之外，英特尔也正计划在更多处理器平台，例如至强系列处理器中加入英特尔 ® SGX 技术，帮助用户更便捷地部署基于硬件增强的安全机制，从而推动联邦学习等方法在 AI 领域的应用与发展。

基于英特尔® SGX 的解决方案

■ 1+N 式多源数据 AI 模型训练解决方案

借助英特尔 ® SGX 技术，金融企业可以根据自身的实际情况构建多样化的解决方案。下文将简单介绍一种基于英特尔® SGX 技术的 1+N 式的多源数据 AI 模型训练解决方案。

1+N 式解决方案架构如图 2-5-13 所示，其由位于中心的聚合服务器（Aggregator）“飞地”以及部署在各处的 N 个边缘“飞地”组成网络。聚合服务器和各个数据源中的“飞地”，均是由英特尔® SGX 技术提供的处理器指令，在内存中构造出的具有高等级安全访问权限的可信区域。

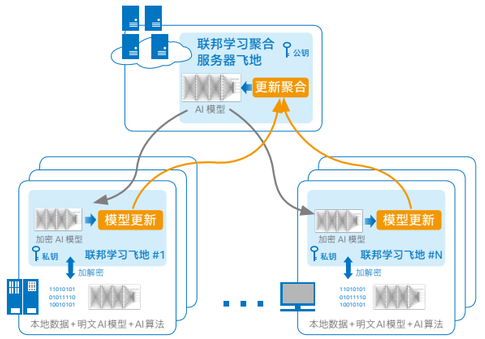


图 2-5-13　使用英特尔® SGX 技术的联邦学习方案

方案中，在加密通道中被传输的是待训练优化的 AI 模型以及相关的中间参数，而训练数据、明文 AI 模型以及 AI 算法则被留存在各个节点本地。在初始化过程中，“飞地”会自己产生公私密钥对，公钥注册到聚合服务器，私钥保存在各自的“飞地”里。当训练开始时，首先，聚合服务器会和目标“飞地”建立加密连接（通过公私密钥对的非对称算法提供的能力，来协商本次连接的对称加密密钥，防止中间人攻击）。连接建立后，聚合服务器会首先将待训练的 AI 模型加密推送到各个“飞地”中，然后各个“飞地”把模型解密传送到本地 AI 训练环境对本地数据实施训练。训练结束后，本地 AI 训练环境将训练得到的中间参数返回至本地的“飞地”。

解决方案还可针对业务需求，加入了一系列创新流程：例如，每个本地环境中的“飞地”都是联邦的可信代理，随着后期算法可以直接运行在“飞地”里，这个可信代理在本地环境里可以做的事情会越来越多。接下来，“飞地”会在加密连接里，把中间参数加密传回给聚合服务器“飞地”。聚合服务器“飞地”会将收到的中间参数进行快速聚合，并根据结果对 AI 模型进行优化调整，而后进行下一轮的迭代。

上述过程都是在“飞地”中实现的。也就是说，在方案的整个循环迭代过程中，AI 模型以及中间参数，都在加密通道以及“飞地”内进行传递和交互，并不与外界软、硬件接触，形成了安全可信的“内循环”。而基于英特尔® SGX 架构的处理器，可为“飞地”的构建、加密通道的铺设以及中间参数交互和聚合提供强大算力。

对于各节点对训练效果贡献度的评估，解决方案也给出了令人满意的实践方法。在方案中，当有 N 个数据源时，可以先对所有节点进行训练，得到全量的训练效果。尔后再对除了待评估节点以外的 N-1 个节点分别进行训练（例如，评估节点 #1 时，对节点 #2 至 #N 进行训练）。在得到不同训练效果的模型后，系统可以计算出每个数据节点在联邦学习中的“贡献系数”，进而对各个数据节点在 AI 联合训练中的贡献度给出精确评估，并据此进行方案调整。

■ 基于 Graphene 构建深度学习训练模型

通过 Graphene 工具，用户可以便捷地在支持英特尔® SGX 技术的硬件环境中迅速地搭建深度学习训练模型，下文将简述如何快速构建 Graphene SGX 环境，并在其上运行 TensorFlow 框架来进行模型训练。

以 Ubuntu 18.04 及 Python 3.6 环境为例，基本流程如下：

a) 安装英特尔® SGX SDK， SGX PSW 和 SGX Driver

下载链接：https://download.01.org/intel-sgx/ linux-<version>/<OS><OS version>

安装具体步骤请参考以下链接：

https://github.com/intel/linux-sgx/blob/master/README.md.

b) 安装 Graphene 工具：

从 github 上下载最新的 Graphene 工具：

```
git clone https://github.com/oscarlab/graphene.git
```

可以参考以下步骤，运行 Graphene-sgx 环境（可根据实际情况予以调整）：

1. 安装与 SGX 相关的依赖项

```
sudo apt-get install -y libprotobuf-c-dev protobuf-c-compiler \
    libcurl4-openssl-dev

sudo apt-get install -y python3-protobuf
```

2. 安装以 FSGSBASE 修补的 Linux 内核

```
git clone --single-branch --branch linux-5.4.y \
    https://git.kernel.org/pub/scm/linux/kernel/git/stable/linux.git
cd linux
git am <graphene-dir>/Pal/src/host/Linux-SGX/sgx-driver/fsgsbase_patches/*.patch
```

按照 Ubuntu Wiki（https://wiki.ubuntu.com/KernelTeam/GitKernelBuild）中的说明构建和安装内核。

```
uname -r
LD_SHOW_AUXV=1 /bin/true | grep AT_HWCAP2
```

3. 生成签名密钥

```
openssl genrsa -3 -out enclave-key.pem 3072
```

4. 编译

```
make SGX=1
```

更详细步骤说明，请参考链接：

https://graphene.readthedocs.io/en/latest/building.html

安装结束后，若能成功运行 Examples/python-scipy-insecure 样例，则证明安装成功。具体运行步骤可参考以下链接：

https://github.com/oscarlab/graphene/tree/master/Examples/python-scipy-insecure

c) 从以下链接下载 python.manifest.template

https://raw.githubusercontent.com/oscarlab/graphene/master/Examples/python-scipy-insecure/python.manifest.template

下载到 graphene/Examples/python-test 目录下并运行以下命令：

```
$  python3.6 -m venv venv
$  ./venv/bin/pip install --upgrade pip
$  ./venv/bin/pip install --upgrade setuptools
$  ./venv/bin/pip install --upgrade wheel
$  ./venv/bin/pip install numpy==1.16.5
$  ./venv/bin/pip install tensorflow==1.14
$  sudo chown -R root:root venv
```

对 python-test/python.manifest.template 文件作相应的修改，添加以下内容：

```
loader.exec = file:venv/bin/python3.6
loader.argv0_override = venv/bin/python3.6
…
sgx.enclave_size = 4G
sgx.thread_num = 64
sgx.file_check_policy = allow_all_but_log
…
sgx.allowed_files.keras = file:keras
sgx.allowed_files.venv = file:venv
```

d) 准备测试的 test_resnet.py 文件：

```
import ctypes.util
def fl(name):
    raise PermissionError("No permission")
ctypes.util.find_library = fl

import os
import posix
import platform

def static_os_uname():
    return posix.uname_result(('Linux', 'sgx256', '4.15.0-20-generic', '#21-Ubuntu SMP Tue Apr 24 06:16:15 UTC 2018', 'x86_64'))

def static_platform_uname():
    return platform.uname_result('Linux', 'sgx256', '4.15.0-20-generic', '#21-Ubuntu SMP Tue Apr 24 06:16:15 UTC 2018', 'x86_64', 'x86_64')

os.uname = static_os_uname
platform.uname = static_platform_uname

import time, traceback
import numpy as np
import tensorflow as tf
import tensorflow.keras as k
import h5py, io

def main():
    model = tf.keras.applications.resnet50.ResNet50(weights=None)

    x_test = np.zeros((400, 224, 224, 3))

    for _ in range(0,10):
        start_time = time.time()
        model.predict(x_test, batch_size=4, verbose=1)
        print("Evaluation took", time.time() - start_time, "seconds")

main()
```

在本例中，数据集采用 cifar10 开源数据集，构造 Resnet50 网络进行训练。训练的模型会保存在 python-test/scripts/saved_models 中。

在程序中加入以下语句，可使得模型能够正常保存。

```
os.environ["HDF5_USE_FILE_LOCKING"] = 'FALSE'
```

e) 在 python-test 目录下

```
$  make SGX=1
$  SGX=1 ./pal_loader python3.6.manifest.sgx ./scripts/test_resnet.py
```

可以直接运行。

软、硬件建议配置

以上 1+N 式的多源数据 AI 模型训练解决方案的构建，可以参考如下基于英特尔® 架构平台完成，环境配置如下：

硬件配置

名称	规格
处理器	英特尔® 至强® E-2200 处理器或更高
基础频率	最大睿频5.0GHz
核心/线程	8/16
HT	Off
Turbo	On
内存	最高 128GB DDR4 2666MHz
硬盘	英特尔 ® DC S3320 数据中心级固态盘 480GB

软件配置

名称	规格
操作系统	Ubuntu Linux release 18.04
Linux内核	4.15.0-74-generic
工作负载	LSTM, XGBoost
编译器	GCC 7.4
开源软件库	Graphene 1.0, Python 3.6.8, TensorFlow 1.14, XGBoost 0.9
SGX 软件开发套件	Intel SGX Linux 2.8 Release

技术展望

基于 TEE 的联邦学习架构是一个新兴的领域，也为其它联邦学习的实现方式提供了新的可选项，在本文中，介绍了以英特尔® SGX 为代表的 TEE 组件在联邦学习领域的实现价值。目前，英特尔正与诸多合作伙伴一起，推动将 TEE 纳入到联邦学习架构设计中去，和其它的实现方式一起，共同实现一个互相补充，有机结合，可落地的解决方案。

在未来，英特尔与合作伙伴还计划在以下方面进一步充实本文介绍的基于 TEE 的联邦学习架构：

1. 根据英特尔的产品路线图，把此架构实现在其它具有 SGX 功能的英特尔 ® 至强 ® 服务器里；
2. 实现算法优化，能够把更多的联邦学习算法加载到 SGX 飞地中；
3. 基于区块链来实现联邦学习密钥自主协商和分配。

中国平安应用案例

案例背景

作为平安集团旗下的科技解决方案专家，平安科技正基于人工智能、云计算，为 5 亿 + 生态用户提供覆盖金融、医疗、汽车、房产、智慧城市五大生态圈的端对端智能科技服务，助力企业实施智能化转型。

这一过程中，平安科技希望通过聚合多种数据源，为 AI 应用提供更多、更优质的训练数据集，从而提升 AI 平台在不同场景下的训练效果。但数据访问、聚合和交互过程中的安全风险，令用户的数据隐私安全无法获得充分保护。因此，平安科技亟待寻找一种更完善的解决方案，在保证数据隐私的前提下来充分挖掘数据的价值。

平安科技内重要的 AI 技术实践专家 -- 平安科技联邦学习技术团队（以下简称“联邦学习团队”）通过探索和运用联邦学习方法、聚合了更多数据，提升了其 AI 模型训练效果。在这一过程中，联邦学习团队也与英特尔开展了一系列深入合作，使用英特尔® SGX 技术，成功构建起基于数据隐私安全保护的多源数据 AI 训练一站式解决方案：蜂巢联邦学习平台。该方案在多个场景的运用中均获得了令人满意的成果，为探索多方数据协同实施 AI 训练提供了有益的实践。

中国平安蜂巢联邦学习平台

蜂巢联邦学习平台，是平安科技联邦学习团队结合英特尔® SGX 技术，为用户提供的基于数据隐私安全保护的多源数据 AI 训练一站式解决方案。平台特征如下：

- 提供多种加密方式，支持同态加密等多方安全计算机制；
- 支持单机和多机训练；
- 可直接使用英特尔架构处理器实施训练；
- 支持多种深度学习 / 机器学习算法和框架。

平台架构如图 2-5-14 所示。蜂巢联邦学习平台自下而上共分为四层，最底层是由一系列硬件设备组成的基础设施层，包括了基于英特尔® 架构的处理器等；其上是算子层，平台在这里为用户提供了丰富的深度学习框架支持，包括了常见的 TensorFlow，Keras，PyTorch，MXNet 等。同时，算子层集成了主要的联邦学习功能模块，例如样本对齐、特征对齐、梯度计算器等。用户的 AI 模型、数据以及参数，将通过算子层内的各个模块来调用英特尔® SGX 技术，在基础设施层内构建飞地，形成可信任的 AI 训练环境。

图 2-5-14 蜂巢联邦学习平台架构

在算法层，蜂巢联邦学习平台集成了同构逻辑回归、同构 RNN、异构随机森林等一大批常见的深度学习 / 机器学习算法，基本涵盖了金融领域在风险评估、反洗钱、投顾、投研、信贷、保险和监管等众多应用场景中的需求；而在顶层的表现层，平台则提供了一系列用户交互能力和任务管理能力，方便用户更有效地协调 AI 训练任务。

除了传统联邦学习平台所具有的功能外，蜂巢联邦学习平台推出的创新功能还包括：基于联邦学习的医疗影像数据平台、扩接融合用户特征与个性推荐系统和动态车险定价模型系统等。这些深入结合金融行业具体应用的功能，可以帮助用户获得更低的学习曲线，实现更快的业务融合。

平台应用场景

目前，蜂巢联邦学习平台在一系列业务场景中的广泛应用，均取得了良好的效果。以平台在保险行业的应用为例，以往，用户在投保时，业务人员只能根据用户的年龄、性别等基本信息来确定保费金额。而今，用户数据的数量和特征维度在信息化环境中有了巨大的增加。以健康类险种为例，业务系统如果能够利用海量的病历、家族病史数据等进行 AI 预测，并得到更加细分的健康评估类别，就能提升投保人健康评估结果的准确度。

但病历、病史等无疑是各个健康医疗机构中绝对不能公开且需要提升安全等级进行保护的绝对隐私数据。现在，通过结合英特尔® SGX 技术的蜂巢联邦学习平台的引入，企业可以在不触及用户数据的情况下开展保险定价模型的 AI 训练。在已有的实践中，来自一线的反馈表明，保险个性化定价效果得到了明显的提升。

除了在投保个性化定价系统中的应用，蜂巢联邦学习平台还在以下应用场景中获得了良好的效果：

- **车联网大数据智能分析**

蜂巢联邦学习平台能在满足数据隐私保护的合法合规的前提下，整合不同行业客户的特点（如：保险公司、运营车队、二手车评估平台等），为用户提供定制化的大数据发布服务（分析结果 / 报告等），充分体现“自有数据运营”的价值。也可以用实际数据来验证、评价、预测相关业务的正确性，更全面地为车厂创造更多价值。从经济性、环境适用性、可靠性、安全性等方面挖掘数据价值，向车厂研发部、质量控制部等提供业务决策依据。

- **精准完善的社会征信评分系统**

个人征信系统线上线下联动并逐步向完善的方向发展，是不可逆的趋势和潮流。蜂巢联邦学习平台让民间征信的数据纬度变得更加多元化，通过联合民生、金融、社交网络、电商、交通、和监管这六大领域的累计数据，更全面地勾勒出每个人在信用方面的画像，进而建立精准的社会征信用评分系统。

- **医疗临床决策支持系统**

蜂巢联邦学习平台所具备的联邦图像分析和识别技术，可帮助医疗联邦系统为基层医疗机构打造病医联体和医疗信息平台，利用识别医疗影像（X 光，CT，MRI）数据，或者挖掘医疗文献数据建立医疗专家数据库，为医护人员提供智能的解决方案，从而提高工作效率和诊疗质量。

- **智能语音管理系统**

蜂巢联邦学习平台能够支持金融领域应用级电话语音录入，为各参与方提供联合建立语音识别系统、电话语音转写记录以及语音数据智能统计分析的功能。通过优化后的语音识别系统，能对大量通话记录内容进行识别、统计、分析，从而帮助小微金融机构在最短时间内了解不同业务的话务结构，实现更高效准确的智能质检和价值信息提取，定位导致客户投诉、流失、话务异常等问题原因、并预测业务热点趋势、发现潜在客户。

小结

借助先进的联邦学习方法，英特尔® SGX 技术以及英特尔架构处理器等先进软硬件产品，正助力金融企业有效应对 AI 应用进阶时遇到的多源数据安全协同难题。这一过程中，针对金融企业在风险评估、反洗钱、投顾、投研、信贷、保险和监管等应用场景中的需求，英特尔与众多金融行业合作伙伴一起，利用基于联邦学习的 AI 应用构建起更有效的风控、营销和管理模型，有效识别信用卡盗刷、贷款逾期、金融欺诈等潜在金融风险，大为减少金融企业的经营风险，为联邦学习方法在金融行业的应用落地提供了有益的参考。

未来，英特尔还计划与更多金融企业深入开展技术合作，以先进的技术驱动数据资源在联邦学习中的安全运转和高效转化，在保证数据安全的前提下消除数据孤岛，推动联邦学习的快速发展和应用。

先进内存产品与创新算法模型　推动高可用、低 TCO 的金融 AI 解决方案落地

基于金融数据特征的 AI 落地解决方案

高维特征金融数据需要更优内存方案

■ 金融数据的高维特征

作为 AI 三驾马车之一，数据在 AI 应用和解决方案中起着至关重要的作用。在 AI 应用和解决方案的落地过程中，不同行业的数据特征表示能力会对所涉及的算法算力提出不同需求。例如在社交场景中，数据主要包括用户交互记录、用户身份等，数据关联性强，数据维度单一；在电商场景中，数据以销售记录、商品数据以及用户购买行为等为主，特点是数据维度少，数据特征相对复杂；而在工业制造领域，数据则主要是日志等过程数据，数据时序性强，特征相对集中。

在金融行业中，银行、保险等金融机构所面对和处理的数据同样也有其鲜明的特性。以常见的预测系统为例，如图 2-6-1 所示，某商业银行需要通过预测 A 市未来一段时间的房价走势来预判信贷风险，其基本流程是将所采集的样本数据集通过机器学习 / 深度学习方法训练得到相应模型，然后将其置于预测数据集中进行推理得到结果，结果与实际验证结果相校验后，再通过模型自我学习来进行优化。

上述过程中，数据集（样本数据集、预测数据集）特征的表示能力对机器学习 / 深度学习方法的性能至关重要。在金融行业中，数据特征的表示通常可分为宏观和微观两个层面。以个性化推荐系统为例，物品和系统的属性，例如价格、期限等，是全部用户的通用特征（宏观描述），而用户各自的属性，例如年龄、性别等则属于专项特征（微观刻画）；而对于反欺诈业务而言，银行平台层面的属性属于宏观层面的通用特征，而用户各自的交易和账号信息则属于微观层面的专项特征。以上用户数据特征经梳理和衍生后，最终可获得极为可观的数据维度。

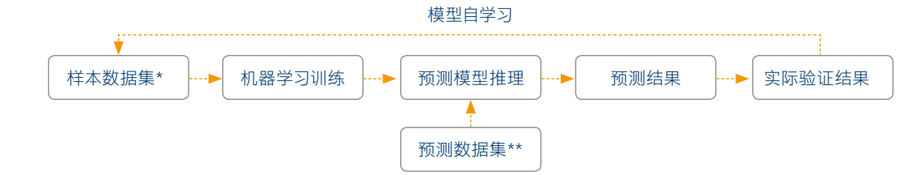
具有高维特征的金融数据在构建AI学习方法时，通常会遇到训练效率问题。究其原因，是数据集拥有的大量用户样本在微观层面虽有大规模专项特征，能更精确地表述个体行为，表示能力更强，但由于数据样本量巨大，因而在构建机器学习/深度学习模型时，会对系统的计算、存储性能提出更高要求。因此，金融AI应用在处理具有高维特征的数据时，既需要选择创新、独特的算法，也需要寻求更先进的硬件基础平台来提供强大的性能助力。

■ 金融高维数据模型对内存方案提出更高挑战

众所周知，机器学习 / 深度学习模型的训练样本量越大，数据维度越高，其蕴含的信息量也就越多，但相应的计算处理过程也会变得更复杂，这对基础硬件设施，尤其是内存的性能也提出了更高的要求。对于高维模型而言，其构建的金字塔型的巨型数据矩阵，底层数据维数可能高达上亿乃至数十亿。这一方面对处理平台的并行实时计算能力有着更高需求，需要处理器具备更高的主频、更多的核心 / 处理线程以及更优化的微架构；另一方面，高维数据模型也对大容量高性能内存有着迫切需求。

数据建模分析表明，当数据模型的维度在百万级时，模型文件大小一般为 GB 级，在十亿维度时，文件大小可至 TB 级。而当 AI 系统进行模型计算和更新时，会产生大量的中间结果数据用于迭代过程。

在传统系统设计中，这些有着低延迟、高吞吐性能要求的数据迭代任务都由高性能动态随机存取存储器（Dynamic Random Access Memory，DRAM）内存来承载。但随着金融数据的维度不断向上突破，DRAM 内存的使用就变得不再“实用”。



*样本数据集

特征	城市	地段	面积	楼高	...	价格
	A 市	二环	120	12	...	9,000,000
学习样本	A 市	三环	95	4	...	7,200,000
	...	...	...	...	...	...

** 预测数据集

特征	城市	地段	面积	楼高	...	价格
	A 市	二环	100	8	...	?
预测对象	A 市	二环	90	6	...	?
	...	...	...	...	...	?

图 2-6-1　基于 AI 方法的金融预测系统

以上文的推荐系统为例，其数据维度具有数百个原始特征，再进一步实施衍生后，可出现近亿列特征，单个推荐系统所使用的内存数据库所需内存容量可达数百 GB，当企业同时使用十余个推荐系统时，数据库所需内存将高达数 TB。TB 级的内存需求不仅给金融企业用户带来了沉重的成本开支，同时大容量的 DRAM 也需要企业部署更大规模的集群来应对，不仅给企业的 IT 运维和管理带来了巨大压力，也给 AI 解决方案的落地带来挑战。

为应对这一挑战，英特尔正与第四范式等合作伙伴一起，结合一系列创新高维特征算法以及 RTiDB 实时特征数据库等技术，以英特尔® 傲腾™ 持久内存和其他英特尔先进软硬件产品与技术为基础，为金融企业提供高维数据场景下的可参考金融 AI 应用范例。

先进内存产品结合创新算法为金融 AI 应用“降本增效”

■ 面向高维特征数据的创新算法

金融行业中常用的传统机器学习或深度学习算法，如逻辑回归（Logistics Regression，LR）、梯度提升回归机（Gradient Boosting Machine，GBM）以及深度神经网络（Deep Neural Network，DNN）等，在处理高维特征数据集时，往往面临内存资源消耗大、计算复杂度高、训练效率低等问题。为使英特尔 ® 傲腾™ 持久内存在此类场景中发挥出更大效能，英特尔与第四范式一起，针对高维数据特征场景对一系列模型算法进行了优化，包括：

- **HD-LR 线性模型**：其使用 FTRL（Follow The Regularized Leader）算法来迭代求解模型，从而拥有更快的求解速度，并对正则化算子予以了支持，非常适于处理含大量稀疏特征的超大规模数据集；
- **HD-GBM 梯度提升回归机**：其增加了一个 HD-Linear 模型，在处理高维稀疏特征数据时，决策树子模型可与 HD-Linear 子模型交替进行迭代提升，通过提高两者间的交互来获得更好的训练效果；
- **HD-DSN 深度稀疏网络算法**：相较传统 DNN 算法，HD-DSN 算法不仅能对多种超高维稀疏数据实施高层次特征抽象，还可自动学习嵌入式表达。目前 HD-DSN 算法所支持的数据维数已达十万亿级之多；
- **HD-He-Treenet 高维离散嵌入树模型**：在面对一些时序性较强的数据集时，高维离散嵌入树模型可通过一系列基于统计的算法，将高维稀疏离散特征实施嵌入，从而大幅降低特征维度，在不损失原特征所含信息量的同时将稀疏变为稠密。同时，变换后的特征在嵌入时序信息后，特征的信息量也得以进一步丰富，令训练效果获得进一步提升。

■ 适于高维特征的英特尔® 傲腾™ 持久内存

在传统金融企业信息化系统中，大规模中间业务数据的“落盘”任务，通常由硬盘（Hard Disk Drive，HDD）或固态盘（Solid State Drive, SSD）来承担，但随着推荐系统等金融 AI 应用所需处理的数据特征呈指数级增长，更高的实时性需求亟待系统能大幅提升数据访问速度，而其中最有效的方法之一，是将更多数据存储在更靠近处理器的位置（如图 2-6-2 所示，越向上越靠近处理器）。

近年来，随着大量数据存储围绕着 DRAM 内存展开，各类内存数据库已经成为 AI 应用中“热数据”处理不可或缺的载体。但大规模 DRAM 内存的使用不仅带来了更高的采购和维护成本，同时 DRAM 内存的易失性特征，也使系统在遭遇宕机等问题时，需要花上更多时间来使工作进程重新加载数据。

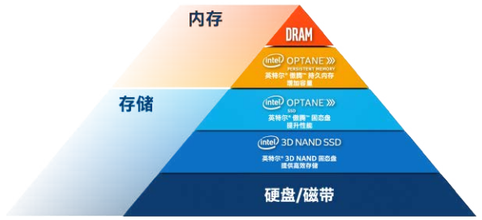


图 2-6-2　兼顾内存级性能和大容量持久化存储能力的英特尔® 傲腾™ 持久内存 [22]

为此，通过与上节中各创新算法的结合，英特尔针对高维特征模型处理过程中，大容量、低时延、以及高性能数据加载 / 读写等方面的要求，基于英特尔® 傲腾™ 技术提供了高可用、低 TCO 的持久内存（Persistent Memory Module，PMM）解决方案，来兼顾金融行业 AI 应用在高性能和大容量两方面的需求。

作为兼顾内存级性能和大容量持久化存储能力的英特尔® 傲腾™ 持久内存，其以独特的 3D XPoint ™ 存储介质构建，能够在密集、无晶体管以及可堆叠的设计中单独寻址内存单元，通过与其他英特尔先进系统内存和存储控制器、接口硬件以及软件增强功能相结合，可获得与 DRAM 内存相近的读写性能和访问延时。

[22] 图片引自英特尔官网：https://www.intel.cn/content/www/cn/zh/architecture-and-technology/intel-optane-technology/reimagine-memory-storage-in-the-data-center.html

当工作在内存模式（Memory Mode）下时，如图 2-6-3 所示，基于英特尔® 傲腾™ 技术构建的英特尔® 傲腾™ 持久内存不仅可以插入到标准的双列直插式存储模块（Dual-Inline-Memory-Modules， DIMM）内存插槽中与 DRAM 内存相兼容，同时与 DRAM 内存 不同的是，英特尔® 傲腾™ 持久内存还能提供了两项彻底改变内存和存储功能的重要特性：持久性和高密度。前者意味着即使断电或重启，数据仍会保留；后者则指容量高达 512GB / 每 DIMM 插槽，是当前 DRAM 内存最大密度的数倍。

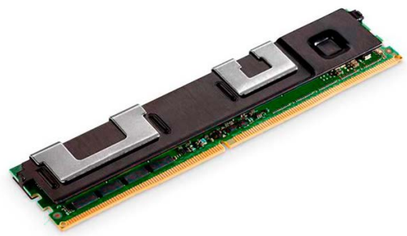


图 2-6-3　英特尔® 傲腾™ 持久内存 [23]

凭借这两种特性，一方面金融企业部署的内存数据库等在进行系统维护，或发生意外停机后，数据可以得到保留而无需重新加载，不仅避免因数据丢失引起的一系列问题，使 AI 系统的训练和推理不再受到意外的影响而保持强一致性，同时系统恢复时间也可压缩至数秒内；另一方面，更低的每 GB 内存成本，也使金融企业可以灵活地在数据中心配置更多大容量内存数据库，为虚拟机配备更大内存，或进一步加大虚拟机密度等，从各个方面提升面向金融数据的 AI 应用的工作效率，使其可更游刃有余地处理高维数据特征。

■ 面向金融数据的英特尔® PMDK工具包

在先进硬件产品之外，英特尔也通过英特尔® 持久化内存开发工具包（Persistent Memory Development Kit，PMDK）为用户提供了面向英特尔® 傲腾™ 持久内存的编程模型和环境，通过与高性能内存数据库相配合，构建面向金融数据的持久化内存解决方案，使金融企业可以便捷地将高维数据转化为持久化内存式数据结构，用于后续的模型训练和推理。

在这一解决方案中，由于持久化内存的数据生命周期往往大于程序进程的生命周期，所以传统的基于易失性内存（例如 DRAM 内存）的编程模型，包括内存分配与管理、基于虚拟地址的指针等将不再适用。英特尔® PMDK 工具包通过引入一种新的持久性内存编程模型，即以最新的 libpmemobj-cpp 来实现持久化的数据结构，打造 KV（key-value）存储引擎，从而帮助用户用尽量小的代码改动将已有的、基于传统内存模型的程序平滑过渡到持久化内存中，并保证所有数据在任何情况下（重启、软硬件错误导致的系统崩溃等）的一致性。下文将对这一方案中，PmemStore-KV 存储引擎的设计与实现，以及其与高性能 RTiDB 实时特征数据库的整合进行简要描述。

底层数据结构的设计与实现

底层数据结构决定了 key 的组织方式，或 value 的索引方式（index），将直接影响到 KV 引擎在不同工作负载下的性能，如点查询（point query）较多时，hash-based index 会比较合适，而范围查询（range search）更适合采用 b+tree, skiplist 等方式。

持久性数据结构的 debug 和数据一致性测试是当今推广持久化内存编程最重要的难题之一，业界仍然处于探索阶段，目前已有一些受到广泛认可的工具，如 pmemcheck（in valgrind）、pmreorder 等，这些对基于英特尔® PMDK 的程序查错都有一定帮助。

KV 引擎架构设计与实现

基于 key → pmem_pointer, pmem_pointer → value 的架构，英特尔® PMDK 工具包可以帮助用户实现一个基本的本地 KV 引擎，此时如何管理 value 将成为影响性能的关键。为此方案采取了以下设计：

- 基于英特尔® PMDK 的持久化内存管理，其可以有效解决 value 的空间分配回收管理；
- 提供不同性能的数据一致性策略供选择，用户可以根据性能要求进行取舍。

高性能 RTiDB 实时特征数据库

传统关系数据库或内存数据库，如 MySQL、Redis 等，并非为时序抽取而设计，在性能上无法达到毫秒级的高维时序特征抽取速度，更难以应对因高维特征抽取而导致的 I/O 爆发。而即便是特定的一些特定的时序数据库（Time Series DB），在性能上也无法满足 TP99（即满足 99% 的请求所需最低时延）为 5 毫秒的金融硬实时场景需求。

而由第四范式推出的 RTiDB 是以基于排序的核心数据结构、读写之间的非阻塞执行以及少事务性作为基本设计理念，通过内置的高维特性抽取引擎 Feature Extractor 等，在高维特征数据处理上，有着巨大的优势。与常见的内存数据库相比，时序抽取性能可以达到其 5-10 倍，而 PUT 性能则能达到其 2 倍 [24]。因此非常适于金融行业反欺诈、反洗钱、营销、推荐等多种场景。

[23] 图片引自英特尔官网：https://www.intel.cn/content/www/cn/zh/architecture-and-technology/intel-optane-technology/reimagine-memory-storage-in-the-data-center.html

[24] 性能数据援引自《第四范式 AI 数据平台架构》

从 RTiDB 的系统设计可以看出，其核心的 Tablet Server 模块由基于内存的存储引擎以及独特的内存恢复模型构成。前者需要大容量、高性能内存予以支持，传统 DRAM 内存相对昂贵的价格，以及相对受限的单机内存密度无疑是其发挥更大作用的瓶颈。而后者则有赖于持久化的内存式数据结构。因此让 RTiDB 发挥更大效能的关键，是找到一种兼顾大容量和持久性特性的内存产品与其配合，而英特尔® 傲腾™ 持久内存无疑是非常符合这一需求的先进内存产品。

目前，通过英特尔® PMDK 工具包提供的 KV 引擎架构设计，RTiDB 实时特征数据库已经与英特尔® 傲腾™ 持久内存实现了良好的整合。目前英特尔正与第四范式一起围绕性能优化探索更多优化方案。虽然基于 DRAM 内存的 RTiDB 相比基于英特尔® 傲腾™ 持久内存的 RTIDB 仍有性能优势，但通过移除 snapshot/binlog 与磁盘的同步写，能使得基于英特尔® 傲腾™ 持久内存的 RTIDB 在新的持久化模型下，对写操作有着 TP9999/TP99999（即满足 99.99%/99.999% 的请求所需最低延迟）延迟优化。

更多英特尔® PMDK 工具包详情，请参阅 https://pmem.io/pmdk/

■ 利用PMDK实现数据库快速恢复实例

首先，系统管理员和应用程序开发人员可以通过访问 https://pmem.io/ 来获得 PMDK 工具包，其安装方法如下：

```
# install pmdk

sudo yum install ndctl-devel daxctl-devel
git clone https://github.com/pmem/pmdk
cd pmdk
git checkout tags/1.8
make -j4
sudo make install prefix=/usr/local
```

同时还需要安装额外的 libpmemobj-cpp 库：

```
# install libpmemobj-cpp

git clone https://github.com/pmem/libpmemobj-cpp
cd libpmemobj-cpp
git checkout tags/1.9
mkdir build && cd build
cmake .. -DTBB_DIR=$TBBROOT/cmake -DCMAKE_INSTALL_PREFIX=/usr/local -
DUSE_TBB=1
make -j4
sudo make install
```

纳入必要的头文件

```
#include "libpmemobj++/container/concurrent_hash_map.hpp"
#include "libpmemobj++/p.hpp"
#include "libpmemobj++/persistent_ptr.hpp"
#include "libpmemobj++/pool.hpp"
#include "libpmemobj++/utils.hpp"
#include "libpmemobj++/make_persistent.hpp"
#include "libpmemobj++/make_persistent_array.hpp"
#include "libpmemobj++/transaction.hpp"
```

进行初始化时，需要创建或打开 pmempool。参考代码如下：

```
try {
    if (!rtidb::base::IsExists(FLAGS_pmem_pool_file)) {
        LOG(INFO, "pmem pool file does not exist, start to create a new one");
        pop_ = pool<::rtidb::storage::PmemRoot>::create(FLAGS_pmem_pool_file,
                PMEM_LAYOUT, FLAGS_pmem_pool_size,S_IWUSR | S_IRUSR);
        LOG(INFO, "new pmem pool file created, size = %llu", FLAGS_pmem_pool_size);
    } else {
        LOG(INFO, "pmem pool file found, use the existing one");
        pop_ = pool<::rtidb::storage::PmemRoot>::open(FLAGS_pmem_pool_file, PMEM_LAYOUT);
    }
} catch (pmem::pool_error &e) {
    LOG(ERROR, "failed to open/create pmem pool file");
    return false;
}
```

PMemRoot 是保存在 pmempool 的 root 位置，也是找到所有 RTiDB 表数据的入口。可以用 libpmemobj-cpp 提供的 persistent concurrent hash map 来实现 table id → tabledata 的 mapping table。参考代码如下：

```
tabledata 的 mapping table
class PmemRoot {
public:
    using hashmap_type = concurrent_hash_map<p<uint64_t>, persistent_ptr<PmemTableData>>;
    persistent_ptr<hashmap_type> pmem_tables;
};
```

在创建 RTiDB 新表时，需要先查看 root 中的 mapping table 是否存在，如果不存在则需要预先创建。然后初始化 RTiDB 表的必要数据（PmemTableData），并插入到 mapping table 中。参考代码如下：

```
if (pop_.root()->pmem_tables == nullptr) {
    LOG(INFO, "pop_.root()->pmem_tables is null, start to create one");
    try {
        pmem::obj::transaction::run(pop_, [&] {
            pop_.root()->pmem_tables = make_persistent<PmemRoot::hashmap_type>();
        });
    } catch (const pmem::transaction_error &e) {
        LOG(ERROR, "failed to allocation pmem for root->pmem_tables, %s", e.what());
        return false;
    }
} else {
    LOG(INFO, "pop_.root()->pmem_tables not null, start to initialize");
    pop_.root()->pmem_tables->runtime_initialize();
}
PmemRoot::hashmap_type::accessor acc;
bool res = pop_.root()->pmem_tables->find(acc, tid_pid);
if (res) {
    LOG(WARNING, "pmem tid %u pid %u exist", id_, pid_);
    return false;
} else {
    LOG(INFO, "start to create pmem_table_data");
    try {
        pmem::obj::transaction::run(pop_, [&] {
            meta_ = make_persistent<PmemTableMeta>();
            persistent_ptr<persistent_ptr<PmemSegmentData>[]> segs_data
                = make_persistent<persistent_ptr<PmemSegmentData>[]>(idx_cnt_*seg_cnt_);
            pmem_table_data = make_persistent<PmemTableData>(meta_, segs_data);
            for (uint32_t i = 0; i < idx_cnt_; i++) {
                for (uint32_t j = 0; j < seg_cnt_; j++) {
                    segs_data[i * seg_cnt_ + j] = segments_[i][j]->Init();
                }
            }
        });
    } catch (const pmem::transaction_error &e) {
        LOG(ERROR, "failed to allocation pmem for root->pmem_tables, %s", e.what());
        return false;
    }
    LOG(INFO, "create pmem_table_data success, start to insert to pmem_tables");
    pop_.root()->pmem_tables->insert(PmemRoot::hashmap_type::value_type(tid_pid, pmem_ta
}
```

在打开已有的 RTiDB 表时，也是先查看 root 中的 mapping table 是否存在，如果不存在则需要预先创建。然后在 mapping table 中找到 RTiDB 表的必要数据（PmemTableData），用于对表进行即时恢复。参考代码如下：

```
if (pop_.root()->pmem_tables == nullptr) {
    try {
        pmem::obj::transaction::run(pop_, [&] {
            pop_.root()->pmem_tables = make_persistent<PmemRoot::hashmap_type>();
        });
    } catch (const pmem::transaction_error &e) {
        LOG(ERROR, "failed to allocation pmem for root->pmem_tables, %s", e.what());
        return false;
    }
} else {
    pop_.root()->pmem_tables->runtime_initialize();
}
PmemRoot::hashmap_type::accessor acc;
bool res = pop_.root()->pmem_tables->find(acc, tid_pid);
if (res) {
    pmem_table_data = acc->second;
    meta_ = pmem_table_data->meta_;
    for (uint32_t i = 0; i < idx_cnt_; i++) {
        for (uint32_t j = 0; j < seg_cnt_; j++) {
            segments_[i][j]->Load(pmem_table_data->segs_data_[i * seg_cnt_ + j]);
        }
    }
} else
{
    PDLOG(WARNING, "pmem table tid %u pid %u does not exist", id_, pid_);
    return false;
}
```

■ 基于英特尔® 傲腾™ 持久内存的性能和TCO测评

为验证英特尔® 傲腾™ 持久内存与 RTiDB 等产品与技术相整合后，对金融行业 AI 场景带来的性能与 TCO 提升，英特尔与第四范式一起进行了一项 DRAM 内存与英特尔® 傲腾™ 持久内存在性能和 TCO 等指标上的对比测试，测试配置如下表所示：

系统配置		
处理器	双路英特尔® 至强 ® 铂金 8280L 处理器	
处理器频率	2.70GHz（Turbo Boost @ 4.0GHz）	
L1/L2/L3 缓存	1.75MB/28MB/38.5MB	
DRAM 内存	384GB（12*32GB DDR4 2666MHz）	
英特尔® 傲腾™ 持久内存	2TB（8*256GB 2666MHz）	
存储	750GB 英特尔® 傲腾™ 固态盘 DC P4800X	
操作系统	CentOS-7.6（Kernel 5.1.9-1.el7）	
英特尔® 傲腾™ 持久内存固件	01.02.00.5375	
RTiDB 配置		
DRAM 内存	Volatile Skiplist@ DRAM 内存	日志存储于英特尔® 傲腾™ 固态盘
英特尔® 傲腾™ 持久内存	Persistent Skiplist@ 英特尔® 傲腾™ 持久内存（AD 模式）	日志无需额外存储

英特尔® 傲腾™ 持久内存测评配置

测试流程：

1. 数据准备

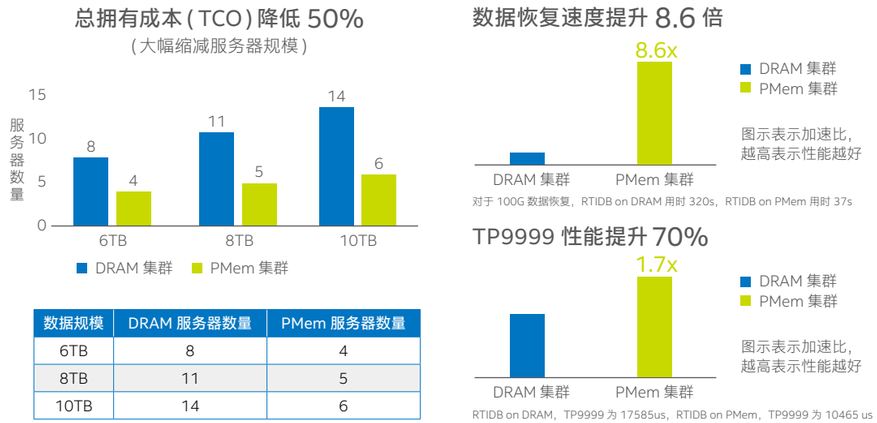
```
##Schema
ttl_type: kAbsoluteTime
ttl: 0
partition_num: 8
replica_num: 3
columns:
-
name: "card"
type: "string"
add_ts_idx: true
-
name: "mcc"
type: "string"
-
name: "amt"
type: "string"
-
name: "num"
type: "int64"
-
name: "coll"
type: "double"
-
name: "ts1"
type: "int64"
is_ts_col: true
-
name: "ts2"
type: "int64"
is_ts_col: true
column_keys:
-
index_name: "combined_key1"
col_name: ["card"]
ts_name: ["ts1","ts2"]
-
index_name: "combined_key2"
col_name: ["mcc","amt"]
ts_name: ["ts1"]
-
index_name: "combined_key3"
col_name: ["num"]
ts_name: ["ts1"]
```

2. get&scan 操作

- get 与 scan 合并操作；
- 操作 * key * 50 条数据；
- 压力测试：采用 1-128 个线程，压测时间为 1 小时。

3. put 操作

- 在原有表存在情况下，另新建一表，键值对（key-value）数量随机。
- 压力测试：采用 128 个线程，压测时间为 10 分钟。



数据规模	DRAM 服务器数量	PMem 服务器数量
6TB	8	4
8TB	11	5
10TB	14	6

图 2-6-4　DRAM 内存与英特尔® 傲腾™ 持久内存的性能和 TCO 对比

测试结果如图 2-6-4 所示，在不同的数据规模下，英特尔® 傲腾™ 持久内存可比 DRAM 内存综合降低 50% 的 TCO，同时数据恢复速度提升 8.6 倍，TP9999 性能 (即满足 99.99% 的请求所需最低时延) 提升 70%。

更多测试详情，请参阅第四范式内部报告 Sigmod2020 Paper Experimental Results: https://wiki.4paradigm.com/display/PlatformRD/Sigmod2020+Paper+Experimental+Results

应用场景

通过创新算法、RTiDB 以及英特尔® 傲腾™ 持久内存的结合，面向高维特征数据的 AI 解决方案已在多个领域的金融场景中进行应用和推广。包括：

交易欺诈侦测：一方面，新方案可以结合银行、支付机构的反欺诈场景，建立超高维机器学习模型来实施欺诈防控。另一方面，方案也可结合实时信息处理，实现实时信息处理和在线服务，为银行、电商、支付机构提供实时交易反欺诈能力。与传统专家系统相比，识别准确率较原有规则提升数倍，并可实现毫秒级的事中交易实时阻断。

信贷风控：方案可联合知识图谱、自然语言处理 (Natural Language Processing，NLP) 等技术，为信贷全生命周期中的各个关键业务环节保驾护航。涵盖风险预警、信息验真、人机博弈、欺诈识别、贷后管理、催收预警、失联修复等多个业务场景。

供应链金融：通过对供应链数据、企业数据、市场数据以及金融机构数据实施深度分析与实时计算监测，方案可使金融机构及时获取企业资金需求，并精准认知企业经营能力与风险状态，从而提供更优质的服务；另一方面，方案也可将金融服务融合到供应链交易管理过程中，与企业的生产经营管理深度结合，让中小企业简单、顺畅地享受到金融服务。

智能客服：方案可使用海量数据建立对话模型，结合多轮对话与实时反馈自主学习，精准识别用户意图，支持文字、语音、图片的交互，实现了多领域的语义解析和多形式的问答对话，将智能客服由此前的 “辅助问答” 向 “主动问答” 转变，提升用户体验。

软、硬件建议配置

以上基于英特尔® 傲腾™ 持久内存的机器学习解决方案的构建，可以参考如下基于英特尔 ® 架构的平台完成，环境配置如下：

硬件配置

名称	规格
处理器	双路英特尔® 至强 ® 铂金 8280L 处理器或更高
基础频率	2.70GHz (Turbo Boost @ 4.0GHz)
核心/线程	28/56
HT	On
Turbo	On
内存	2TB (8*256GB 2666MHz) 英特尔® 傲腾™ 持久内存或更高
存储	750GB 英特尔® 傲腾™ 固态盘 DC P4800X 或更高

软件配置

名称	规格
操作系统	CentOS-7 或其他 Linux 系统
Linux 内核	5.1.9-1.el7
工作负载	第四范式先知内置特征数据库 RTiDB
编译器	GCC 5.4
库	Jdk (jdk-8u121-linux-x64.tar.gz) Zookeeper3.4

技术展望

随着金融业务对实时性、交互性，以及 AI 应用能力提出更高的要求，内存数据库正在金融企业中获得更广泛的部署。而作为一种创新的内存技术，英特尔® 傲腾™ 持久内存产品以其更具经济性的容量以及对数据持久性的良好支持，正助力金融企业更有效地围绕内存数据库，开展基于高维时序数据的一系列机器学习 / 深度学习方法，为金融业务提供卓有成效的 AI 能力支撑。

在未来，更多金融行业用户正计划将英特尔® 傲腾™ 持久内存与更多技术和业务场景相融合，打造更有效的高级数据分析能力和 AI 应用。例如随着 5G 时代的到来，超低延迟、超大带宽的网络体验也促使金融业务能够提供与之匹配的，更快捷的数据分析和 AI 应用反应速度。但传统构建在数据中心、或云上的数据处理能力，其响应速度无疑会受到网络质量、带宽等因素的影响。

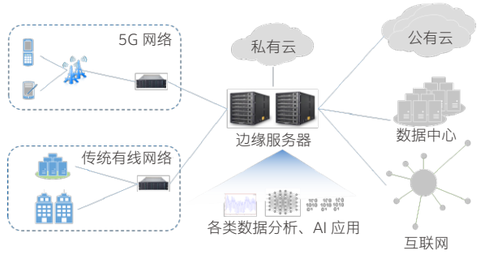


图 2-6-5　典型边缘计算节点部署架构

为此，越来越多的金融企业正在网络边缘部署边缘计算 (MEC) 节点来应对以上挑战。典型的边缘计算节点部署架构如图 2-6-5 所示，边缘服务器能帮助企业将低时延、大带宽和本地化业务，尤其是需要进行大量数据处理、存储和传输的应用下沉到网络边缘，令这些业务的用户感知和体验更友好。

但边缘服务器由此也会承受更大的性能、容量以及可用性挑战，而英特尔® 傲腾™ 持久内存产品在这三个方面，无疑比传统选用 DRAM 内存的边缘服务器更具优势。一方面，目前最大规格的英特尔® 傲腾™ 持久内存单条容量已达 512GB，而常见 DRAM 内存单条容量仅为 32GB，内存密度相差巨大。更多的内存意味着金融企业可以在边缘服务器中部署更多的虚拟机或业务进程，并减少服务器数量，降低 TCO。另一方面，英特尔® 傲腾™ 持久内存内置的 AES 256 bit 硬件加密技术，也能在边缘侧为金融业务带来更高的数据安全性。

为进一步帮助用户提升数据处理能力，目前英特尔已经推出了全新第二代英特尔 ® 傲腾™ 持久内存，如图 2-6-6 所示，新一代产品的内存带宽平均增幅较上代可提升 25%[25]，每路内存容量容量高达 4.5TB[26]，在一些场景中，可帮助用户缩短数据库启动时间达 12.5 倍[27]。因此可为用户提供更为强劲的处理性能和更大的数据处理空间，为金融行业提供更优质的数据处理舞台。

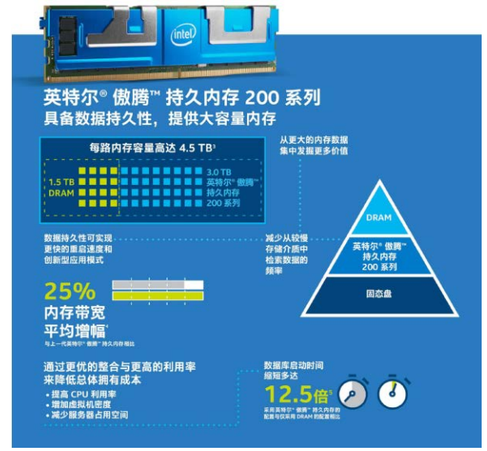


图 2-6-6　全新第二代英特尔® 傲腾™ 持久内存 [28]

[25] 数据测试配置为：基准配置：单节点，搭载 1 颗英特尔® 至强 ® 8280L 28C@2.7GHz 处理器的 Neon City，单个持久性内存模块配置 (6 个 32GB DDR4 DRAM; 1 个 {128GB, 256GB, 512GB} 英特尔® 傲腾™ 持久内存 100 系列模块，15W)，ucode 0x04002f00，用以运行 Fedora29 内核 5.1.18-200.fc29.x86_64 和 App-Direct 模式下的 3.8 版本 MLC。数据来源 2020ww18_CPX_BPS_DI。英特尔于 2020 年 4 月 27 日测试。新配置：单节点，搭载 1 颗英特尔® 至强 ® 预生产 CPX6 28C @2.9GHz 处理器的 Cooper City，单个持久性内存模块配置 (6 个 32GB DDR4 DRAM；1 个 {128GB, 256GB, 512GB} 英特尔® 傲腾™ 持久内存 200 系列模块，15W), ucode 预生产，用以运行 Fedora29 内核 5.1.18-200.fc29.x86_64 和 App-Direct 模式下的 3.8 版本 MLC。数据来源：2020ww18_CPX__BPS_BG。英特尔于 2020 年 3 月 31 日测试。

[26] 配置为 6 个 512 GB 英特尔 ® 傲腾™ 持久内存 (3,072 GB) + 6 个 256 GB DDR4 DRAM (1,536 GB) = 每路总内存 4,608 GB。

[27] 数据基于2018年5月30日进行的测试。SAP HANA模拟工作负载，使用SAP BW版本的SAP HANA标准应用基准测试版本2（2018年5月30日）。传统DRAM的基准配置：联想ThinkSystem SR950服务器，搭载8颗英特尔® 至强® 铂金8176M处理器 (28颗内核，165瓦，2.1 GHz) 。总内存包括48个16 GB TruDDR4 2,666 MHz RDIMM和5个ThinkSystem 2.5英寸PM1633a 3.84TB 容量 SAS 12GB 热插拔固态盘 (SSD) , 用于SAPHANA存储。操作系统是SUSE Linux Enterprise Server 12 SP3 , 使用 SAP HANA 2.0 SPS 03，带有6TB数据集。表预加载10次迭代后所有已完成数据的平均启动时间：50分钟。

[28] 图引用自英特尔官网：https://www.intel.cn/content/dam/www/public/cn/zh/images/20200715-turbocharge-your-ai-and-analytics-cn.jpg

第四范式创新算法在某商业银行应用案例

案例背景

手机银行 APP 已成为各大商业银行最重要的线上渠道之一，如何在功能基本雷同的一众 APP 中脱颖而出？针对不同用户属性，在 APP 页面中部署 “千人千面” 的个性化推荐位，无疑是赢得客户青睐的好方法。

在金融 AI 道路上耕耘多年的某商业银行，亦希望在其手机银行 APP 的推荐展位上，向客户推荐合适的理财产品，从而提升客户响应率，增加理财销售收入。但一般地，金融企业会通过在 APP 中以“埋点”的形式，获取客户的 APP 使用习惯、使用轨迹、行为热力图等，从而感知客户的偏好和关注点，进而选取合适的推荐位向客户进行推荐。

由于该银行的手机银行 APP 开发较早，APP 中并没有埋点，同时，银行还希望这一推荐系统可以与该银行其他渠道的历史理财购买行为相互关联，提高推荐成功率。通过数据分析可知，各个渠道累计的海量历史数据具有典型的高维、稀疏特性，这在提供丰富信息的同时，也对 AI 推荐方案的构建提出了挑战。

英特尔与第四范式一起，将创新算法、RTiDB 与英特尔® 傲腾™ 持久内存结合，利用机器学习方法对具有高维特性的银行理财历史数据进行训练和推理，从而精准判断客户的理财需求，为 APP 提供合适的产品推荐方案。

方案架构及部署

新的基于机器学习的银行营销推荐系统架构如图 2-6-7 所示，在方案中，选取了银行综合理财系统中最近一年内的用户数据作为样本集，包括用户基本信息、理财产品信息、理财购买记录以及用户功能信息等。

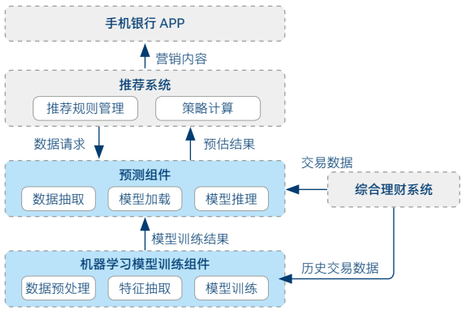


图 2-6-7　某商业银行营销推荐系统架构

样本集首先会在架构底层的机器学习模型训练组件中进行数据预处理、特征抽取以及模型训练。模型训练得到的结果会被加载到预测组件中，执行数据抽取、模型加载和推理工作。推理得到的预估结果会经由银行的推荐系统，最后呈现在手机银行 APP 上。

这其中，样本定义、样本构造等几个步骤是影响方案效能的关键。样本定义如图 2-6-8 所示，方案设定客户营销日期开始的 7 天内购买了某款理财产品作为正样本客户，而 7 天内如果客户没有购买某款理财产品，则作为该款产品的负样本客户。

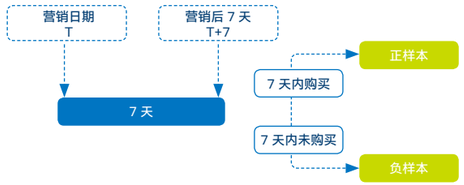


图 2-6-8　样本定义流程

在样本构造过程中，如图 2-6-9 所示，方案将成功购买理财产品的记录作为正样本，营销时间按照购买时间往前推 7 天(防止穿越，配合回收策略)，并记录为：
<cust_id,prd_id,mkt_time,1>;

而负样本则以产品为维度，从历史上没有该款产品的客户中，按照正负比例 1:50 (这一比例是权衡理财产品购买平均响应率，生成样本的代价以及训练运行时间后得出) 选择客户作为负样本客户，营销时间从该款产品的正例营销时间中随机选取，记为：
<cust_id,prd_id,mkt_time,0>

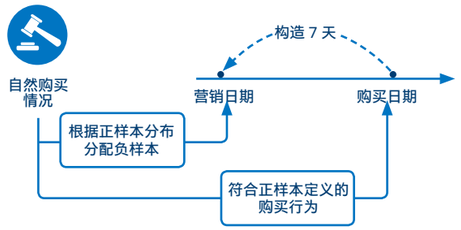


图 2-6-9　样本构造流程

此处负样本客户包含两种情况，一是历史上从未没有购买过理财产品的客户，用于区分客户是否会购买理财产品；二是历史上没有购买 A 产品，但是购买过 B 产品的客户，此部分用户会作为 A 产品的负样本客户，用于模拟客户对产品的选择性。

接下来，方案会进行特征抽取，根据对综合理财系统中历史数据的分析，方案将数据分为用户状态类、产品属性类、用户行为类、统计类等不同的特征大类，每个大类下又细分出几十上百种不同的特征小类。进而衍生出近亿特征列。

在模型训练过程中，方案通过筛选 trans_time，考虑 22 天的数据可得性 gap，得到正样本的流水，同时方案也控制每款理财产品的比例保持一致，负样本的客户则从没购买过该产品的客户群中进行筛选 (例如 A 产品的负样本客户从没有购买过 A 产品的客户群中筛选，B 产品的负样本客户从没有购买过 B 产品的客户群中筛选)，最后筛选得到 2 个月的流水，通过机器学习方法训练得到 17021921 个样本。最后，方案选取了该商业银行在售的一款众享型长期产品对预测效果进行了评估。

方案成效

为验证基于机器学习的推荐方法是否更具优势，方案中设置了一组随机推荐方法 (随机组)、一组基于专家规则的推荐方法 (专家规则组) 与机器学习方法 (机器学习组) 进行了对比测试。三种推荐方法的对比方案设计如图 2-6-10 所示。

图 2-6-10　三种推荐方法对比方案设计

对比方案采用 A/B 测试方式，比较专家规则组、随机组与机器学习组各自的响应率。测试方式首先筛选出符合该款产品购买条件的对私客户约 150 万。然后专家规则组、随机组和机器学习组分别对该名单进行筛选，对符合条件的客户进行短信营销。三组方法各自按照自己的规则生成 20,000 个客户名单用于真实的短信营销，并在营销结束后统计结果。

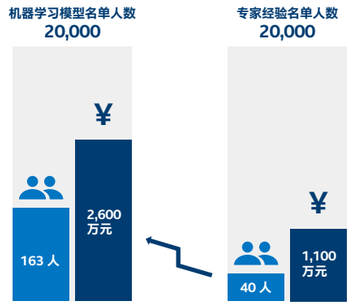


图 2-6-11　不同推荐方法营销效果对比

测试结果如图 2-6-11 所示，机器学习组的营销效果远优于专家规则组和随机组。其中机器学习组共有 163 名客户购买，占所有购买客户的 29%，销售额累计 2,601 万元，占所有购买客户的 23%；而专家经验足共有 40 名客户购买，占所有购买客户的 7%，销售额累计 1,107 万元，占所有购买客户的 10%；最后，随机营销组 2 万名客户中仅有 10 名客户购买，占所有购买者的 2%，销售额累计 87 万元，占所有购买客户的不足 1%。

值得一提的是，基于机器学习方法的推荐方法可以完全通过在线方式进行训练、推理和效果评估。与传统的专家规则推荐方法相比，新方案可以让用户实时地了解到推荐效果的变化，并及时做出策略调整，这对于“寸时寸金”的金融行业而言，无疑是巨大的利好。

小结

通过创新算法、RTiDB 以及先进英特尔® 傲腾™ 持久内存产品的结合，英特尔与第四范式等合作伙伴一起，构建了面向金融数据特性的实时线上机器学习方法，助力金融企业更多地将 AI 应用推广到营销推荐、风险评估等领域。这其中，一系列创新的算法有效降低了高维、稀疏特征带来的计算复杂度和耗时，提升了训练效果；高性能的 RTiDB 实时特征数据库以高维特性抽取引擎提供了强有力的时序特征抽取性能。更为关键的是，由英特尔提供的英特尔® 傲腾™ 持久内存，不仅以大容量、高性能的内存产品特性，解决了高维、稀疏特征数据处理所需的大内存，更以其持久性特性，提供了良好的数据恢复速度，为机器学习方法的在线运行提供有力保障。

未来，英特尔还计划与更多合作伙伴一起，基于金融数据特性展开更多技术探讨，以先进的软硬件产品与创新算法模型相结合，推动高可用、低 TCO 的金融 AI 解决方案快速落地和应用。

巧妙运用“新芯”动力，以知识图谱助力金融行业挖掘更多高价值信息

知识图谱在金融行业的应用

金融行业中的知识图谱

■ 知识图谱简介

作为 AI 技术的重要分支，知识图谱（Knowledge Graph）以其对复杂信息关系的准确勾勒，正在各行各业中获得越来越多的部署与应用。如图 2-7-1 所示，知识图谱是由一系列实体、实体属性以及实体间关系构成的 AI 系统。

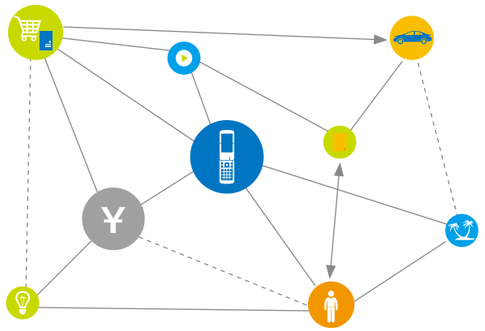


图 2-7-1 勾勒复杂信息关系的知识图谱

通过知识图谱，用户能更精准地剖析互联网时代数据海洋中蕴含的高价值信息，以及信息之间的内在关系，从而获得更有深度，更具效能的解决方案。现在，知识图谱系统已在信息检索、推荐系统、信息反欺诈、舆情监测、市场细分、社交链等领域获得了大量成功应用。

例如在信息检索中，搜索引擎可利用知识图谱对信息进行精准聚合和匹配、加深对关键词的理解和对搜索意图的语义分析，提升搜索效率；在推荐系统中，知识图谱能作为一种辅助信息工具，集成在电子商务等场景中，精准匹配用户的购买意愿和商品候选集合，为用户提供更精准的推荐选项；在社交领域，知识图谱可提供可视化的关系表示，优化好友推荐体验、喜好聚合等功能。

■ 金融行业知识图谱系统构建

在金融行业，知识图谱同样也获得了广泛的运用，例如在金融风控领域，金融机构可以通过知识图谱来分析实体之间的关系，进而分析金融风险等级，便于制定应对措施；在金融营销领域，金融机构可以通过知识图谱来分析挖掘客户潜在商机，开拓更多关联优质客户，使业务人员在传统关系型营销获客模式外，增添更多获客模式，扩大潜在客户源。

如图 2-7-2 所示，金融行业的知识图谱系统通常可由数据采集、数据预处理、实体和实体关系抽取、图展现、数据存储 & 管理、任务调度以及各类对内对外的服务接口等模块组成。

在数据采集步骤，一方面，系统通过合法合规的方式从多个内外数据源中采集到企业和自然人的公开，或内部数据信息。公开信息例如可从互联网获得的企业相关信息、各类舆情新闻等，内部信息例如金融机构各个信息化系统提供的内部担保数据、抵押数据等，这些数据需要运用高效成熟的 NLP 技术来实施采集；另一方面，系统也需要接入大量非文本化数据信息，例如存在于图像、视频的信息，这需要系统具备强有力的光学字符识别 (Optical Character Recognition，OCR) 能力。

在其后的数据预处理模块，首先，系统会在数据清洗步骤将一些不规范数据、错误数据以及重复数据予以剔除，然后再通过数据融合，将同一个实体下来自内外部不同源的数据进行融合和对齐。

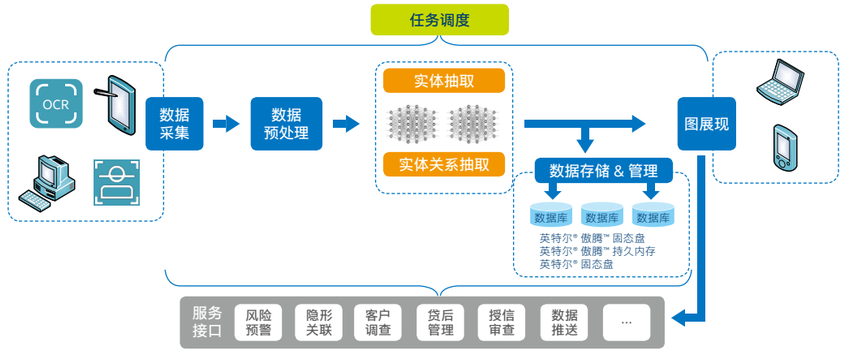


图 2-7-2 常见金融行业知识图谱系统组成

在核心的实体 & 实体关系抽取模块，系统会先通过 AI 算法，构建出企业、自然人等实体；然后，实体属性挖掘模块和关系挖掘模块会利用 AI 算法引擎对实体数据进行分析，获取实体间的关系和属性，并最终转化成具有实体、实体属性以及实体关系的知识图谱数据结构。

如图 2-7-2，展现模块通常能以可视化的方式，在各类终端上向用户展示上述的实体、属性和关系。展现的方式一般有网格状方式和树状方式，其中网格状方式在展示与中心实体的关联关系外，还可展示其他关联实体之间的关联关系；而树状方式则是展示与中心实体关联的对应实体。

通过实体 & 实体关系抽取后的数据，既可以通过数据存储 & 管理模块保存到金融机构的数据库中，也能以服务接口的方式，供内外部系统在不同场景下进行调用，例如可以通过内部 Wiki 系统供信息筛选检索；也可以通过集成插件的形式，对外提供统一数据服务，例如在企业门户网站中供实时信息展示。

实体抽取方法

如上节所述，在面向金融行业构建的知识图谱系统中，核心模块是进行实体和实体关系的抽取。BERT (Bidirectional Encoder Representations from Transformers) 是目前较为常见的实体抽取模型，其作为一种双向 Transformer 编码器，本质上是采用了对语言表征进行预训练 (Pre-train) 的方法，即通过大量文本语料库训练获得通用的语言理解模型。

经典的 BERT 结构如图 2-7-3 左侧所示，其采用了 Transformer Encoder 模型作为语言模型。Transformer 模型结构如图 2-7-3 右侧所示，模型抛弃了经典的 RNN/CNN 等深度学习模型结构，而是采用 Attention 机制来进行 input-output 之间关系的计算。由于采用了 Transformer Encoder，也就是每个时刻的 Attention 计算都能够得到全部时刻的输入。BERT 模型的优势在于可通过无监督的学习掌握了很多自然语言的一些语法和语义知识，在少量数据集上也能通过微调 (Fine Tune) 方式来取得较好的推理效果。

更多 BERT 模型信息，可参阅 Jacob Devlin, Ming-Wei Chang, Kenton Lee, Kristina Toutanova, BERT: Pre-training of Deep Bidirectional Transformers for Language Understanding https://arxiv.org/pdf/1810.04805.pdf

在此基础上，用户可以根据金融行业普遍的结构化数据特征，以及英特尔® 架构软硬件产品所提供的优化能力，采用一些衍生优化的模型，例如图 2-7-4 所示的 “BERT 字向量 + BiLSTM + CRF” 三层实体识别模型。

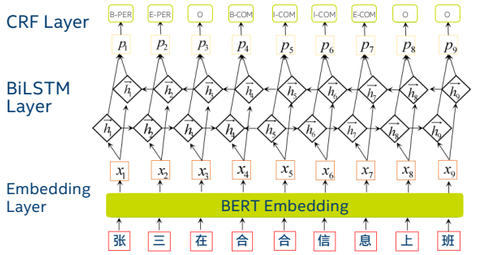


图 2-7-4 BERT 字向量 + BiLSTM + CRF 三层模型结构图

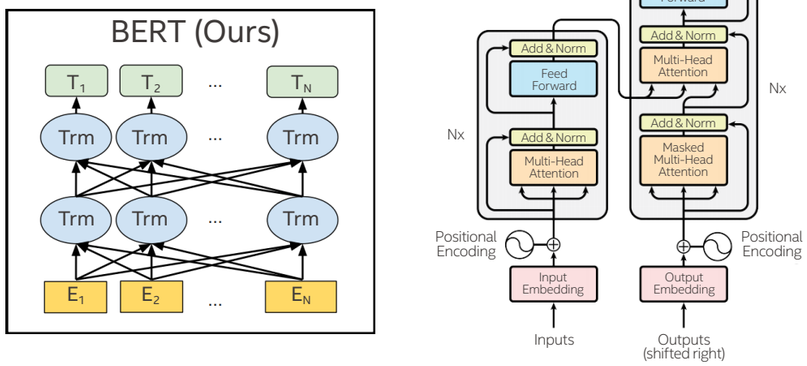


图 2-7-3 BERT 结构与 Transformer 模型 [29]

这一模型主要包含了三层网络结构。底层是 Embedding Layer，其利用 BERT 字向量对每个字符进行语义表示；第二层是 BiLSTM Layer，即双向的 LSTM 模型结构。如前文章节所述，LSTM 是 RNN 重要的衍伸模型，其可以通过特别的门结构设计来避免长期依赖问题，大幅提升记忆时长。基于 LSTM 的网络结构非常适合运用在基于序列的分析任务中。双向 LSTM 同时具有前向和后向传播的能力，这使得网络能够更好的学习上下文语义的信息；第三层是 CRF 层，CRF 层通过训练学习到条件随机场中的转移概率矩阵，从而可以增加标签间的约束关系特征。

实体关系抽取方法

实体关系抽取作为信息抽取的重要任务，是指再识别实体的基础上，抽取出预定义的实体关系。如图 2-7-5 所示，实体对的关系可以用形式化的描述为关系三元组 < 实体 M：关系 X：实体 N>。

图 2-7-5 实体关系抽取架构图

对于实体关系抽取，常见的深度学习方法有 TextCNN 模型、知识增强语义表示 (Enhanced Representation from kNowledge IntEgration, ERNIE) 模型等。其中 TextCNN 模型的输入为序列的词向量，即将词向量通过卷积层、池化层得到最终的特征向量 (这里也可以设置多层卷积)，其最后一层接入全连接的 Softmax，输出实体关系的类别。

而 ERNIE 模型则是在经典 BERT 模型上进行了改进，首先其通过在 Masked LM 中通过对词和实体概念等语义单元进行 mask 来预训练模型，使模型对语义知识单元的表示更贴近现实，其次这一模型也引入了更多源化的数据语料来进行训练。当其应用于实体关系抽取时，可采用 ERNIR + Softmax 分类模型，以相关实体所在句子组成相关段落作为输入加以训练，从而预测出实体间的相互关系。

■ 知识图谱在金融行业中的应用

知识图谱在金融行业中的广泛应用

随着知识图谱技术的不断发展，其应用范围和市场规模也在加速扩展。来自行业研究报告的数据显示，2019 年涵盖知识图谱领域及自然语言处理应用的大数据智能市场规模约为 106.6 亿元，而到 2023 年，这一数字预计将突破 300 亿元。这其中，目前又以金融领域占比最大 [30]。

通过数十年的建设与积累，金融机构的信息化系统及其他相关系统往往采集和保存了海量的结构化业务数据。这些数据蕴含着大量多维度的企业和个人信息，是金融行业推动运营效率，提升用户体验，降低企业风险的财富宝库。

利用这些数据，在高效的基础硬件设施上推动知识图谱系统的构建，可以帮助金融机构直接抽取出数据中实体、实体属性、以及实体间的关系，以简洁明了的方式向用户展示网格化的信息链，有效避免传统关系型数据库在交互式查询时遇到的性能短板。

例如从基础工商数据、业务日志等数据中，可以抽取出企业名称、企业董监高人员等实体，也可以利用股东关系、对外投资、资金往来、担保等业务数据，构建实体间的关系。如图 2-7-6 所示，金融机构能以庞大的图数据库为基础，运用针对性的大数据分析算法，可视化展示企业幕后的关联关系。从而更高效地规避企业间的关联风险。

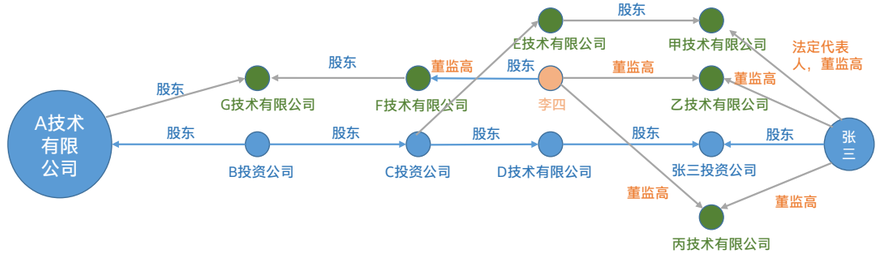


图 2-7-6 利用图谱关系挖掘实体间关系 (源自合合信息的 “企业图谱” 产品)

[29] 经典 BERT 模型相关描述援引自 Jacob Devlin, Ming-Wei Chang, Kenton Lee, Kristina Toutanova, BERT: Pre-training of Deep Bidirectional Transformers for Language Understanding: https://arxiv.org/pdf/1810.04805.pdf

[30] 数据援引自艾瑞网《2020 年中国知识图谱行业研究报告》: http://report.iresearch.cn/report_pdf.aspx?id=3553

目前，知识图谱在金融行业中有以下的典型应用场景：

- **基于图谱关系挖掘与风险挖掘技术实现金融风险防控**

错综复杂的企业股权关系，使商业银行等金融机构在进行授信风控、贷款审核，信贷预期防控等工作时，难免有所错漏。利用大数据平台，构建企业画像，并基于知识图谱数据库围绕"实体、关系、事件、属性"进行高效存储和管理，能更有效规避以上风险，帮助客户经理、风险管理条线、授信审批条线等工作人员更好地发现隐藏在复杂网络之下的风险关系网络，以及隐性关联关系。

- **基于司法关系抽取构建金融防控系统**

司法关系与金融风险息息相关，随着我国司法数据逐步公开，金融机构可通过各种方式获得越来越多的案件判决信息，但目前存在于互联网的裁判文书往往是大段文本，不利于案件的检索、分析和利用。利用知识图谱对各类裁判文书中涉及的多种实体（案件类型、原告、被告、涉案金额等）、实体关系（原告 - 负责人、原告 - 委托代理人等主要关系）进行快速梳理，有助于金融机构在进行授信、关联担保等业务时，对涉及的企业和个人信用等级做出准确判断，提前对可能的司法隐患进行"排雷"。

- **基于互联网信息实施企业经营风险防控**

任何突发或热点事件，都可能会对企业经营造成影响，带来隐形的经营风险。例如 A 地区突发的虫害，可能影响农作物 B 的产量，进而降低相应农业机械 C 的销售量，最终影响机械厂商 D 的贷款偿还能力。这种长链分析能力，过去需要由经验丰富的专家来执行，且准确率和时效性不高。现在，通过 NLP 技术与知识图谱系统的加入，金融企业可以迅速采集和辨识包含在互联网中的各种细微信息，并做出相应应对措施。

金融行业典型应用场景：企业图谱关系挖掘与风险挖掘

企业作为商业银行等金融机构的最主要客户群，其生产经营状况的好坏，对商业银行在信贷、授信等业务上有着巨大的影响。现代企业往往有着非常复杂的股权关系，彼此交错形成关联共同体；同时，企业风险是在生产、经营等一系列活动过程中产生和发展的，是一个动态的过程，并且具有很强的传导性。风险通过特定的传导机制累积、放大甚至突发，最后可能会引发连锁型的危机。因此金融机构如果只了解单一企业，单一部门的生产经营状况，并无法判定风险的全貌。目前，越来越多的金融机构正选择知识图谱来实施企业图谱关系挖掘与风险挖掘。

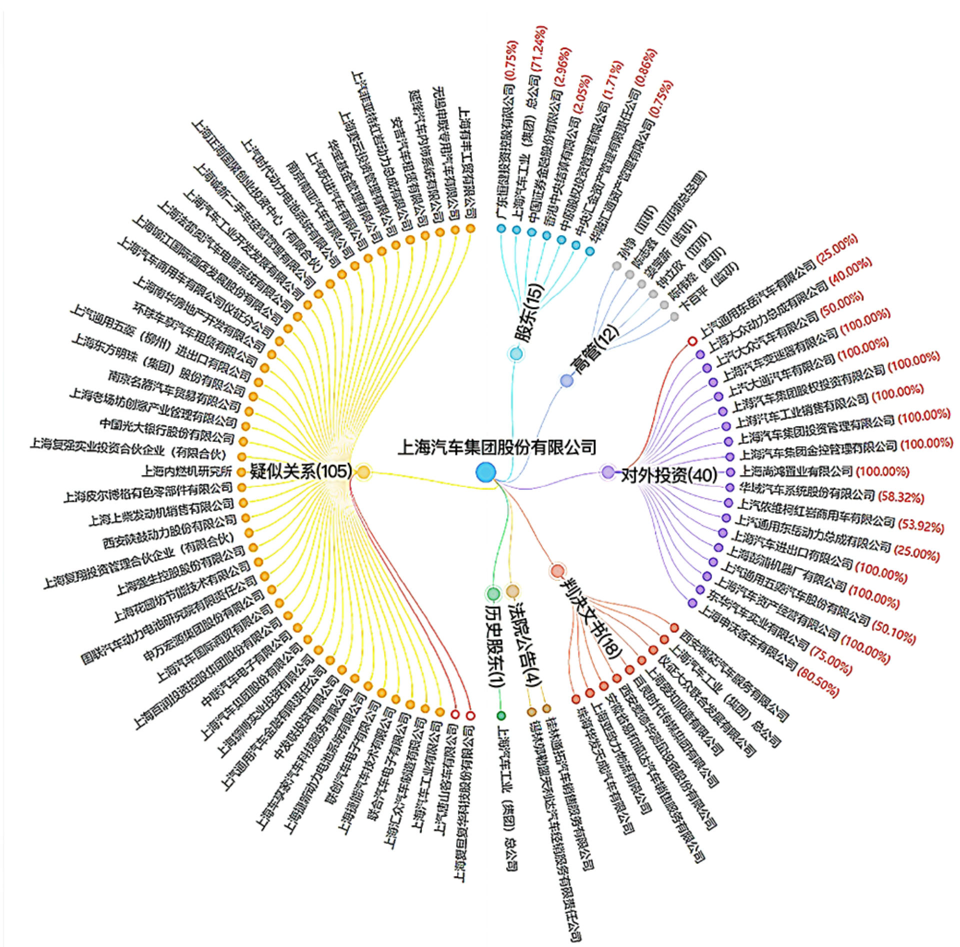


图 2-7-7　企业图谱示例图（源自合合信息 "启信宝" 产品）

一般地，企业关联风险可分为以下两种方式：

- **内部风险传导**：在集团内部，部分分公司出现风险问题会对集团各分公司产生风险传导影响，以及对整个集团产生影响；
- **外部风险传导**：集团外部的供应商、经销商以及合作企业的产生的风险会对集团及集团下各分公司产生风险传导影响；

因此，金融机构首先要通过知识图谱系统来捋清企业图谱关系，然后再构建企业关联风险传导模型，来清晰地展现风险传导状况。如图 2-7-7 所示，企业图谱可展示了企业全貌，是对企业相关信息的一次性全面解剖图示，其包括股东、高管、对外投资、法院判决、法院公告、历史股东及疑似关系等 7 个方面。

以商业银行在开展信贷、授信以及资金监管等业务时，较为关心的 "实际控制人" 和 "企业关联关系" 等信息为例，传统上通过人工调查和维护的模式，在企业规模越来越庞大时，会变得耗时耗力，每增加一家关联企业或股权发生变化时，相关信息都需要很长时间才会传导到金融机构。现在，金融机构一方面可以通过企业图谱可以向上追溯企业股东，查看各个股东的出资比例，深度剖析企业的实际控制人；另一方面也可通过查看关联股东的出资比例、风险等信息，监控当前公司的运营情况，并向下挖掘企业的对外投资，多层次展示企业的投资路径，分析企业股权关系，发现投资关系的交叉持股现象。

在获得企业图谱后，金融机构就可以进一步基于企业风险预警信号体系、企业指标库等能力，有针对性地建设企业关联风险传导模型，例如，以图谱技术、半监督的标签传播聚类、邻近网络算法等，逐步迭代数据和模型，实现事件 + 产业链风险传导模型分析。

■ 金融行业运用知识图谱所面临的挑战

不断发展的知识图谱技术，在为金融等行业用户提供深度高价值信息挖掘能力的同时，也对相关的算法、算力和数据能力提出了更高要求，这使金融企业传统的信息化系统面临着更多的挑战，这些挑战包括：

- 金融数据中越来越多的实体、实体关系信息，对金融机构既有信息化系统的计算、存储和传输能力带来更大挑战。一般地，高维度的结构化金融数据，模型迭代时的中间结果数据规模往往达到 GB 级别乃至 TB 级别，需要核心处理平台具备更高的主频、更多的核心数量以及线程数量，同时还需要内存具有更大的容量和性能；
- 金融机构构建知识图谱系统，涵盖了训练与推理、高性能存储、机器视觉等一系列技术要素，需要一个完整的、包括运用深度学习技术在内的端到端技术链来支撑整个场景；
- 构建高精度的实体语料库需要投入大量人力物力，在实操中须对标注人员进行大量培训；同时高速增长的金融业务数据使实体语料库不断变化，新旧语料往往覆盖不全。一些大型金融机构每天可能新增千万条数据，因此，金融行业在构建知识图谱系统时，需要引入新的深度学习方法，尤其是无监督的深度学习方法来应对以上变化；
- 同时，更多深度学习方法的加入，也需要基础硬件平台针对新方法的需求，针对处理器平台、深度学习框架等提出新的优化方案。

英特尔从云到端的全栈平台包含丰富的软硬件产品，正好来应对这些知识图谱应用上的各种挑战，接下来的章节会详细介绍英特尔软硬件产品如何对各种深度学习方法提供支持来解决上述的问题。

先进软硬件产品为基于深度学习的知识图谱提供支撑

可以看到，金融企业知识图谱系统要发挥更佳效能，就需要在系统的每个环节，数据采集 / 处理、知识图谱构建 / 优化、数据存储 / 管理以及最终的图展现上，都部署更优的计算、存储硬件设备以及相应的软件优化技术。为此，如图 2-7-8 所示，英特尔以其完整的技术链，为知识图谱系统提供整体性的技术支撑。

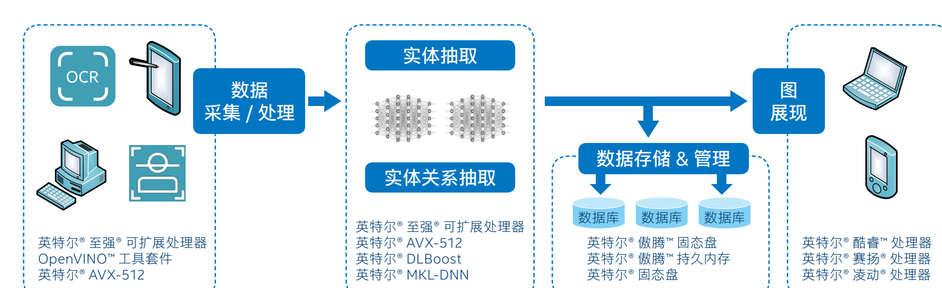


图 2-7-8　英特尔为知识图谱系统提供整体性技术支撑

■ OpenVINO™ 工具套件助力提升数据采集效率

在数据采集/处理阶段，一方面针对数据采集中涉及的大量互联网舆情信息，新系统需要依赖各类爬虫技术进行抓取。高速爬虫系统通常会采用大规模并行处理的方式，而英特尔为之提供的，具有众核、高频能力的英特尔® 至强® 可扩展处理器可在此发挥长处。同时，处理器所集成的英特尔® AVX-512 技术也可加速系统的并行处理能力。另一方面，对于大量的非文本化信息，英特尔也通过 OpenVINO™ 工具套件等开源技术的加入，为常用的 OCR 检测算法提供性能助力。

OpenVINO™工具套件是由英特尔推出的，旨在加速图像视频处理、深度学习推理及部署效率的开源软件工具套件，其不仅内置了大量 OpenCV、OpenXV 视觉库的传统 API 来实现图像视频处理的加速与优化，也加入了深度学习部署工具包，使系统能够更充分地利用英特尔® 架构处理器的计算能力，从硬件指令集层面加速深度学习模型的运行效率。

在一项针对基于 ResNet101 模型的 FPN-SSD 上做延时优先的测试时，开源 ONNX Runtime 引擎使用与未使用 OpenVINO™ 工具套件插件（详情请参阅：https://github.com/microsoft/onnxruntime/releases）之间进行对比，测试结果如图 2-7-9 所示，使用 OpenVINO™ 工具套件对比 ONNX Runtime 引擎在 OpenMP 线程数为 24 的时候有着 2.10 倍的性能优势。

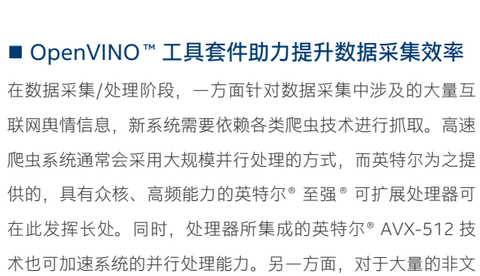


图 2-7-9　OpenVINO™ 工具套件归一化性能对比 [31]

■ 基于英特尔® 架构的处理器为模型推理提供强劲算力

在知识图谱构建 / 优化阶段，英特尔® 至强® 可扩展处理器可以有效提升知识图谱系统中，实体 / 实体关系抽取等深度学习模型的推理速度。新一代的英特尔® 至强® 可扩展处理器不仅具有多达 56 个处理器内核，112 个线程，微架构也进行了全面升级优化，配备了更快、效率更高的高速缓存来提升处理效能，并可支持高达 36TB 的系统级内存容量 [32]。同时，新一代英特尔® 至强® 处理器所集成的英特尔® AVX-512 指令集也可为系统提供更宽的矢量计算功能，供用户在进行系统设计时，能针对性地进行更多底层浮点数计算优化。

而矢量神经网络指令（VNNI）的引入，也使新一代英特尔® 至强® 可扩展处理器在深度学习推理速度上有着耀眼的表明，与上一代产品相比，其推理性能提升高达 30 倍 [33]，有力提升了知识图谱系统的应用效率。

同时，英特尔也为各个主流深度学习框架，如 TensorFlow、Caffe 等都提供了丰富的优化函数库，例如英特尔® MKL 和英特尔® MKL-DNN，这些函数库的引入，可以大幅提升基于深度学习方法的知识图谱系统在英特尔® 架构处理器平台上运行时的性能。

更多英特尔® MKL 和英特尔® MKL-DNN 介绍，请参阅本手册技术篇相关介绍。

■ 存储新技术为系统提供有效数据支撑

知识图谱能为用户提供高价值信息的根源，是其基于深度学习方法，对海量数据开展训练和推理。随着数据规模的不断增加，尤其在金融行业内，大型金融机构每日新增的结构化数据可达数千万条，数据增量超过 100GB。同时在实体 / 实体关系抽取等深度学习模型的训练和推理过程中，金融数据的高维特性，也会使系统产生 GB 级，乃至 TB 级的中间文件。

在传统存储架构中，这类大容量存储主要由硬盘（Hard Disk Drive，HDD）或固态盘（Solid State Drive, SSD）来承担，但知识图谱系统所需的实时性（尤其在金融风控等关键应用场景中），对存储的高性能、低延时提出了更高要求。此时采用更多的 DRAM 内存，无疑可以获得更好的性能，但由此也会给用户带来急剧提升的成本。

如图 2-7-10 所示，基于英特尔® 傲腾™ 技术构建的各级存储产品，可以满足知识图谱系统在以上数据存储中的不同需求。一方面，基于 3D XPoint™ 存储介质构建的英特尔® 傲腾™ 持久内存，不仅拥有与 DRAM 内存相近的读写性能、访问延时，以及相比固态盘更强的耐用性，在金融行业各类高并发的应用场景中，也能有着不亚于 DRAM 内存的性能表现。同时，它还可凭借大容量的特性，帮助金融机构轻松构建起 TB 级的内存数据库。而且其具备的非易失性特性，也能为系统带来更好的可用性，例如在遇到宕机、断电等情况时，利用英特尔® 傲腾™ 持久内存的非易失性特性，系统可以在极短的时间内予以恢复。

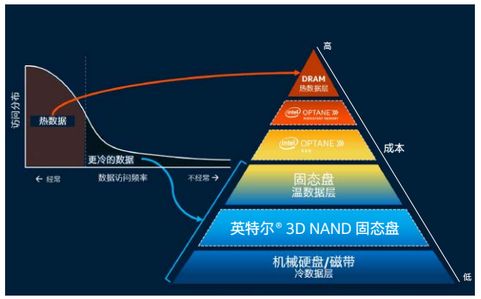


图 2-7-10 各类存储硬件的性能与成本比较

而英特尔® 傲腾™ 固态盘则可从另一维度，帮助知识图谱系统获得高效的存储性能支撑。与英特尔® 傲腾™ 持久内存相比，英特尔® 傲腾™ 固态盘更注重性能、容量和成本的平衡。利用一系列先进内存控制器、接口硬件和软件技术的组合，英特尔® 傲腾™ 固态盘在低延迟、高稳定等多方面均有着良好表现。以英特尔® 傲腾™ 固态盘 DC P4800X 为例，其具有高达 55 万 IOPS 的随机读写能力，低至 10 微秒的读写延迟，可更好地应对金融行业中多用户、高并发的各类应用场景。同时，其优异的写入寿命（Drive Writes Per Day，DWPD），也能赋予系统更长的生命周期，确保系统具有更佳的经济性。

合合信息知识图谱在某商业银行应用案例

案例背景

作为国内领先的商业银行之一，某商业银行一直积极以先进的信息化技术、AI 能力为抓手，通过信息技术实力以及电子银行平台打造自身竞争优势，并以产品创新和卓越的客户服务树立高美誉度的品牌。

随着该银行投资主体多元化的不断推进，业务范围跨行业、跨区域的集团客户越来越多。企业间的关联关系日趋错综复杂，信用状况参差不齐，银行因关联企业识别不充分所造成的各种风险隐患或实际损失也屡见不鲜。在一项银行内部开展的在集团关系验证中，行内抽取 10 个集团的内部企业家谱进行识别，发现有 86 家企业有错误录入问题，同时有 92% 的成员企业未覆盖完全，这对银行业务的开展而言，显然增加了大量不可控风险。

为有效控制金融风险，保证银行信贷资产安全，银行希望通过构建基于知识图谱技术的风险门户系统，对关联企业实施高效识别，防止出现集团客户多头授信、过度授信、关联担保等经营风险。针对新系统的构建，银行提出了以下的需求：

- 关联关系图谱能提供门户页面、关系插件、数据接口三种对外服务类型，行内用户能快捷高效地获取想要部分的内容；
- 可支持全维度搜索企业，展现全量全维度企业信息，行内用户无需在多个系统与站点中来回切换查询，增加业务操作效率；
- 可在同一门户中快速便捷地查看所有企业的各类关系信息，发现疑似风险信息。同时可与银行已有风险信息系统相结合，对行内的风险前置和风险预警作补充。
- 能实现图谱数据统一查询，帮助行内用户全面掌握客户信息，避免由于孤立数据等造成的信息不一致、信用重复、信息不完整等问题。同时，可利用知识图谱的属性和关系挖掘功能，挖掘出一些可以用于信贷审核的隐藏信息。
- 可深度挖掘行内数据信息，如交易、担保等数据内隐含的大量高价值信息，帮助银行达到提前发现与规避风险的目的。
- “开箱即得”式的便捷部署模式，可视化的关系展现模式，并避免二次开发工作；

为帮助银行顺利完成这一构想，深耕知识图谱与 NLP 技术多年的合合信息与英特尔一起，基于英特尔® 至强® 可扩展处理器、英特尔® AVX-512、OpenVINO™ 工具套件等英特尔 ® 架构软硬件产品与技术，为其量身打造全新的风险门户系统并顺利投入使用。新系统在实践中展现出良好的工作效能，并获得了银行内外部使用者的一致好评。

关于合合信息

一直专注于商业智能化之路的上海合合信息科技股份有限公司，正率先将各类 AI 深度学习技术应用到传统模式识别、企业知识图谱系统构建等场景中去，结合丰富的客户风险管理、营销管理等业务场景知识，帮助各类用户，尤其是金融行业用户快速构建覆盖企业内外部数据知识的知识图谱分析与应用能力。

更多信息，请访问合合信息官网：https://www.intsig.com/

[31] 数据源于合合信息内部测试，测试配置为：处理器：英特尔® 至强® 金牌 6248R 处理器，主频 3.0GHz，24 核心 /48 线程，固定输入尺寸：[1,3,768,768]；迭代次数：50 次；测试对象：FPS

[32] 相关参数援引自英特尔官网：https://ark.intel.com/content/www/cn/zh/ark/products/194146/intel-xeon-platinum-9282-processor-77m-cache-2-60-ghz.html

[33] 数据援引自英特尔官网：https://www.intel.com/content/www/us/en/technology-provider/products-and-solutions/xeon-scalable-family/moving-ai-to-the-edge-article.html

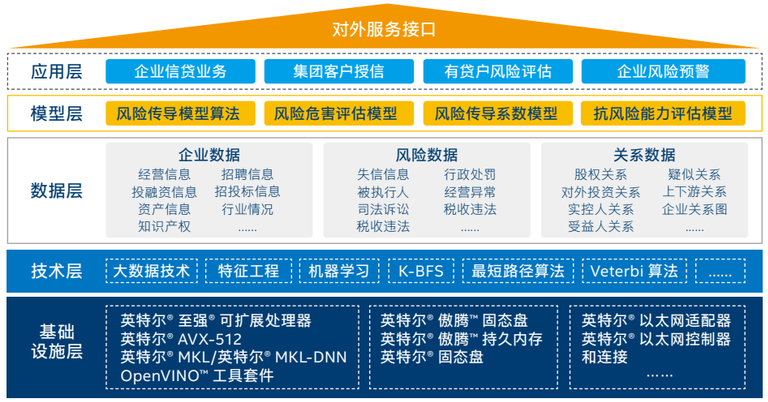


图 2-7-11　某商业银行风险门户系统架构

合合信息知识图谱方案架构及优化方案

■ 方案总体架构

如图 2-7-11 所示，新系统的整体架构至上而下依次分为基础设施层、技术层、数据层、模型层以及应用层。

- 在基础设施层，由英特尔提供的一系列软硬件产品，包括英特尔® 至强® 可扩展处理器、英特尔® AVX-512、英特尔® MKL、OpenVINO™ 工具套件等为整个架构提供了强有力的计算、存储和传输能力；
- 在技术层，是由合合信息针对英特尔软硬件平台特点为更上层应用引入的 BERT、TextCNN 等深度学习和大数据技术模型；
- 在数据层，是系统对接的各类内外部多源化数据；
- 在模型层，则是合合信息结合客户风险管理、营销管理等各类业务场景知识，所部署的风险传导模型算法、风险传导系数模型等。
- 在顶层，系统能以封装 API、集成插件等形式，为银行各系统输出各类风控能力。

值得一提的是，在新项目中，合合信息以 2.3 亿家全国全量全维度企业数据为基础，帮助用户构建了十大客户关联关系图谱，如图 2-7-12 所示，图谱包括了企业的股权结构、对外投资、实际控制人、集团关系、疑似、涉诉、地址、事件关系等 8 种外部数据关系，同时也结合了银行内的交易担保数据，构建企业的交易、担保关系。

图 2-7-12　客户关联关系图谱构建

■ 基于经典 BERT 模型的实体识别模块优化

针对新方案中，所涉及的不同数据特征，合合信息也与英特尔一起，基于经典 BERT 模型和 TextCNN 模型，分别在实体抽取和实体关系抽取模块上开展了大量针对性的优化方法。

其中，实体抽取模型上主要优化方法包括：

- 对经典 BERT 模型进行微调，加入 BiLSTM+CRF 层。因为在序列标注任务中，位置信息是有必要的，甚至方向信息也很重要；而经典 BERT 模型弱化了位置信息，因此加入 BiLSTM 可以更好的学习观测序列上的依赖关系；该方法在项目中常见的企业名实体（通常该类型实体长度较长）类型上可提升约 0.5% 的准确率（F1 Score）；
- 设置合理的学习率 (learning rate)，对 BERT 层与 CRF 层设置不同的学习率；CRF 层设置更大一些，这可以让 CRF 层快速学习；整体平均准确率（F1 Score）提升约 0.2%；
- 使用 1 层 BiLSTM。BERT 本身具有很强的学习能力，底层的已经特征足够丰富，因此 1 层 BiLSTM 便可以取得较好的效果。

在实施优化后，模型与基准模型（基于字符的 BiLSTM+CRF 模型结构）相比较，整体准确率（F1 Score）提升约 4%[34]；而与单独使用 BERT+CRF 模型相比，整体平均准确率（F1 Score）的提升约为 0.8%[35]；

■ 基于 TextCNN 模型的实体关系分类模块优化

主要优化方法包括：

- 在 Embedding 层引入更丰富的特征，进行特征融合，包括字特征，词特征，词性特征等；与单独使用词特征相比，分类准确率（F1 Score）约提升 0.9%；
- 利用多个不同的 kernel_size 来提取文本中的关键信息，从而更好的学习局部相关性；kernel_sizes = [3,4,5,6]，分类准确率（F1 Score）提升约 1%；
- 加入多层卷积，并将 BatchNorm 合并到卷积层中，可以进行归一化并加速训练数据的拟合 0.3%；
- 类别分布不均衡，对数据进行重采样并结合同义词替换，目的是增加数据的多样性；其中少类别提升约 2%。

在实施优化后，模型与基准模型（基于词的 TextCNN 模型结构，kernel_size=3，一层卷积且无 BatchNorm）相比较，整体准确率（F1 Score）提升约为 4.5%[36]。

方案成效

以知识图谱技术为核心的风险门户项目在该商业银行上线后，获得了银行内各个部门的积极响应与反馈。系统通过逐次迭代不断优化，应用效果不断获得提升。其中基于知识图谱企业数据库、大数据挖掘和企业风险实时监控技术，已成功帮助某分行及时获悉被抵押股权遭冻结信息，及时风险预警，及时阻断因信息不对称而引发的风险控制问题。

银行内部统计数据表明，累计使用的分行、子公司总计 55 家；被查看过的企业数量总计 75,596 家；总访问人次总计 91,423 次；主动推送的重点信息达 1,971 条；如图 2-7-13 所示，其中企业基本信息、风险信息、关注信息分别成为客户经理、风险经理、审贷官们最受欢迎的模块。

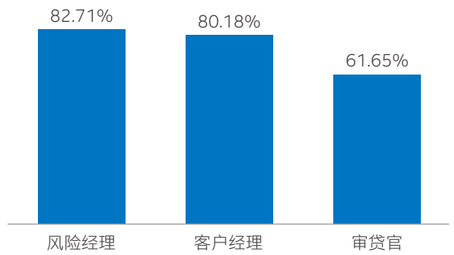


图 2-7-13　某商业银行不同用户对新系统使用覆盖率

同时，系统也帮助银行在舆情预测应用上获得了良好的效果，来自实际使用的数据表明，在基于英特尔® 至强® 可扩展处理器平台上，预测时间仅为 0.232 秒，超过 0.5 秒的业务需求。预测精确率（Precision）为 0.968，召回率（Recall）为 0.952，F1 值达 0.959，完全达到了银行的预期 F1=0.95 的目标[37]。

软、硬件建议配置

以上金融系统知识图谱解决方案的构建，可以参考如下基于英特尔® 架构的平台完成，环境配置如下：

硬件配置

名称	规格
处理器	英特尔® 至强® 金牌 6248R 处理器或更高
基础频率	3.00GHz（Turbo Boost @ 4.00GHz）
核心/线程	24/48
HT	On
Turbo	On
内存	384G (32G DDR4 2933MHz x12)
存储	英特尔 ® 固态盘 D5 P4320 系列及以上

软件配置

名称	规格
操作系统	Centos 7.7.1908
Linux 内核	3.10.0-1062.1.2.el7.x86_64
工作负载	ResNet101 + FPN/ BERT NER
编译器	gcc 7.5.0
库	LLVM OpenMP runtime: Libomp-dev 5.0.1-1
OpenVINO™ 工具套件	OpenVINO 2020 R3

小结

利用知识图谱的方法，更深挖掘海量业务数据中蕴含的高价值信息，是合合信息与英特尔一起合作，面向金融行业用户进行的有益探索。基于深度学习方法构建的知识图谱系统，能让金融行业用户不仅可通过图谱关系挖掘、风险挖掘技术、司法关系抽取等能力的建设，有效构筑起针对信贷风控、授信风控、企业经营防控等金融风险的樊篱，还可以从互联网信息中快速判别突发或热点事件对企业经营带来的影响，并作出相应措施。

英特尔不仅为全新基于深度学习方法的知识图谱系统提供了高性能的英特尔® 至强® 处理器平台作为其计算核心，还提供了英特尔® AVX-512、英特尔® MKL、OpenVINO™ 工具套件等一系列软硬件技术，对系统工作效能提供了全方位的优化。目前，新的知识图谱系统已在某商业银行获得实践部署，在为用户带去便捷、高效的金融风控能力的同时，获得了行内用户的一致好评。未来，合合信息还计划就知识图谱在各行业的应用，与英特尔携手开展进一步的探索。合合信息希望能够依托英特尔从云到端的技术优势和全栈平台，进一步拓展英特尔 AI 创新生态，在各行业各场景中，推动 AI 技术的发展和突破，加速智能应用落地，助力产业智能变革。

[34]、[35] 数据源自合合信息的内部测试。

[36] 数据源自合合信息的内部测试。

[37] 测试配置为：处理器：英特尔® 至强® 金牌 6248R 处理器，主频 3.00GHz，24 核心 /48 线程；
测试场景为随机选取 100 篇舆情新闻，共计 1,327 个句子，Batch Size 设置为 6，测试结果为单一批次平均预测时间及整体的准确率。

端到端统一大数据 AI 平台，助力金融行业实施大数据到深度学习的“无缝切换”

基于金融大数据的深度学习方法探索

深度学习方法为更多金融业务提供 AI 助力

■ 大数据已成为金融行业的基础设施

金融行业与移动互联网、大数据、5G、云计算等新兴技术的不断融合，正催生出网上银行、在线支付等新型金融业务模式，也驱动金融机构启动以智能化、智慧化转型为目标的新一轮信息化建设。这其中，如现代化城市建设之初需要进行“七通一平”工程一样，大数据平台以及其上的应用能力建设，也成为金融机构开展新业务、新功能之前，所必要的“基建”项目。

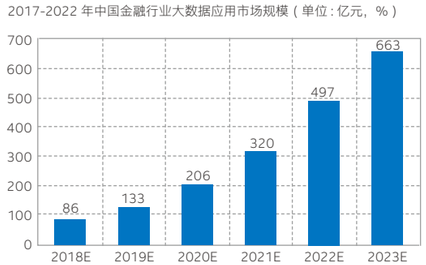


图 2-8-1　中国金融行业大数据规模不断扩大

如图 2-8-1 所示，一项数据表明，中国金融行业大数据规模正以惊人的速度不断扩张。同时，一些分析报告的观点也指出，面对不断创新的业务场景，金融大数据平台也在积极变革，在目前其通常需具备下几个特性：

- **良好的实时计算支持**：更复杂的业务场景以及更高的监管要求，例如实时风控、交易预警、反欺诈等，对大数据平台的联机事务处理（On-Line Transaction Processing，OLTP）性能提出了更高要求。通过英特尔® 架构软硬件产品等构建的基础设施，以及高效的计算、存储引擎，正助力大数据平台提供毫秒、乃至微秒级的实时计算、存储能力；
- **拥抱云平台**：更多金融机构正根据业务需要，选择不同类型的云平台（公有云、私有云以及混合云）部署其大数据服务，以便于形成多维度的数据安全防护体系和异地容灾方案。同时，云化的大数据服务也便于按需交付，并形成一点接入、多点互联的便捷性；
- **分布式架构与数据湖**：新一代的大数据平台往往选择 Hadoop + Apache Spark 等为代表的分布式系统架构。基于 x86 服务器集群的大数据平台，为其提供了良好的横向扩展能力、线性存储方式以及计算资源，能够助其大幅消减计算和 I/O 资源瓶颈。为快速推动变革，一些金融机构甚至已着手建设混合式架构的数据湖 (Data Lake) 方案，为深层次的联机分析处理（On-Line Analytical Processing，OLAP）需求提供助力。

以上趋势明确显示，与 AI 能力的融合无疑正成为金融行业非常关注，且扮演越来越重要角色的部分。众所周知，AI 的发展离不开算法、算力和数据的支撑，而金融行业大数据可以作为金融 AI 应用的天然载体，为一系列深度学习 / 机器学习方法提供丰富的训练数据集。近年来，基于金融业务大数据以及英特尔® 架构软硬件产品构建的基础设施，诸多金融机构已在反欺诈、金融风控、信贷风险预测、智能客服等方面构建了多样化的 AI 应用能力。这些解决方案前面章节已有详细分析，这里不做赘述。

■ 金融大数据平台以及AI能力建设

为满足海量金融业务数据的存储，很多金融机构都已建设了基于 Hadoop/Spark 分布式系统存储架构的大数据平台。并以之为基础，开展各类大数据应用或 AI 能力建设。

典型的金融机构大数据平台如图 2-8-2 所示，其底层是由客户数据、交易数据、账户数据等结构化、非结构化数据构成的数据源。其上是由 Hadoop 服务器集群构建的数据服务能力，为平台提供分布式文件系统、资源 / 任务调度以及计算服务。数据服务之上是金融机构根据自身的需求，构建的各类数据应用。

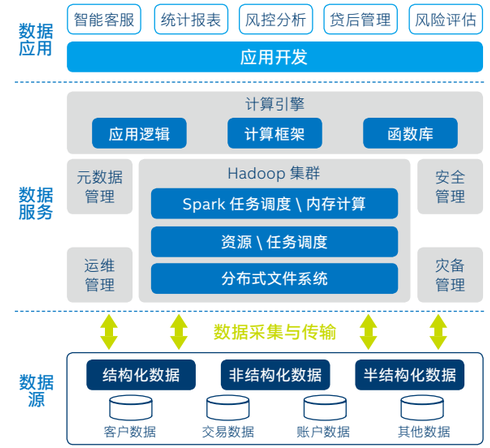


图 2-8-2　典型的金融机构大数据平台

作为整个平台的核心，Hadoop 集群可为海量的金融业务数据提供高效的分布式文件系统、计算框架、数据仓库以及调度能力。近年来，在英特尔、Cloudera 等企业的推动下，越来越多的金融机构开始采用 Spark 作为大数据平台的计算引擎。与 Hadoop 中的 MapReduce 相比，Spark 能对不同来源、不间断输入数据进行准实时的流式处理，同时其也能使用大规模、复杂的机器学习 / 深度学习和图计算，对海量数据进行深度挖掘和分析。另外，Spark 还可以使用分布式高速内存数据缓存，用以支持交互式、迭代计算和数据分析，使数据计算速度大幅提升。

基于以上大数据平台，很多金融机构开展了大量 AI 应用的探索与实践。例如一些商业银行正利用 XGBoost 算法搭建贷款风险预测模型，并取得了良好的成果。但值得注意的是，目前金融机构在中间业务领域，例如代收代付、结算、信用卡发放等金融中间业务的 AI 应用上，主要还是采用机器学习方法。究其原因是，商业银行等既有的用户、资金数据，一般是宽表、结构化数据，且有着显著的序列化特征，因此从机器学习方法着手，更易于开启 AI 应用探索。

随着 AI 应用规模不断扩大，传统单一的机器学习方法也显露出短板。以典型的资金流转预测场景为例，当资金流转的规模和关联性达到一定程度，其呈现出的高智能性、强相关性、紧耦合性以及随机性，使它成为一个复杂的非线性动力系统，而深度学习的方法在预测场景上有着更好表现。

另外，传统机器学习方法也难以实现自动化学习，无法从复杂数据中自我学习，以及通过迭代来优化 AI 效果。同时，金融机构大数据平台拥有的海量数据，也能为深度学习训练提供所需的大量数据集。而这些都为金融机构开展基于金融大数据的深度学习方法探索，提供了必要的前置条件。

■ 基于既有大数据平台构建深度学习方法遇到的挑战

但金融机构从大数据平台向深度学习方法的延伸也并非一蹴而就。这一过程中，金融机构的数据专家和 AI 应用工程师可能会遇到以下问题：

- 既有的 Hadoop/Spark 分布式系统存储架构，在提供强大存储能力的同时，也加深了深度学习框架获取、利用数据的复杂度。从既有大数据平台上构建深度学习模型，如果缺乏端到端的可用平台，将间接抬升金融机构开展深度学习方法研究与探索的门槛；
- 针对不同的业务场景，金融机构在构建 AI 模型时，会根据自身情况选择不同的深度学习框架，以及配套的软硬件基础设施，包括数据平台、计算平台等，这些都会影响预测效率和准确率，并带来大量调试成本；
- 既有大数据平台往往缺乏统一的软硬件集成体系，在使用不同的深度学习框架时，无法有效地为底层算力提供优化、加速。同时，深度学习框架、代码自身的调优，也会消耗大量的人力和时间。

为帮助更多金融机构高效地将其既有大数据平台与深度学习方法探索结合起来，英特尔在为大数据平台、深度学习方法提供一系列软硬件产品和技术方案外，也通过与金融行业伙伴开展的深度合作，为以上从大数据到深度学习的 “无缝切换” 提供了以 Analytics Zoo 为代表的端到端方法、流程和工具平台，并在多个用户的部署实践中，获得了令人满足的成果。

基于金融大数据的深度学习方法探索

■ 构建资金流转预测场景下的深度学习方法

金融机构要在既有金融大数据平台上实现从机器学习到深度学习的 “切换”，需要在一系列的实践中，通过解决以下问题，来形成一套行之有效的方法论以及转换流程：

- 如何根据金融机构业务场景，选择合适的深度学习方法、算法；
- 新的深度学习方法、算法如何与既有大数据平台对接，使大数据平台与深度学习系统形成有效的数据互动；
- 如何利用金融机构信息化系统既有算力和资源，选择合理的优化库，为深度学习方法提供加速；
- 如何降低深度学习方法中，所需的超参数调优等工作给金融机构带来的成本负担。

因此，如图 2-8-3 所示，英特尔要帮助金融机构基于大数据平台，开展深度学习方法的探索，就需要与用户一起构建端到端的方法、流程与工具平台。

首先，在深度学习方法的选择上，虽然已经有越来越多的深度学习方法被应用于金融领域的各个业务场景，例如，利用深度学习方法在影像分析上的优势，开发人脸检测识别、图像分割等 AI 应用，运用于保险理赔、智能客服等领域。但在资金流转预测等场景中，还是采用机器学习方法为主，这是因为传统的深度学习方法在结构化金融数据这类离散化的数据领域中并不占优。同时，其还面临着需要使用大量标签数据进行训练、理论分析复杂、参数调整要求高等问题。

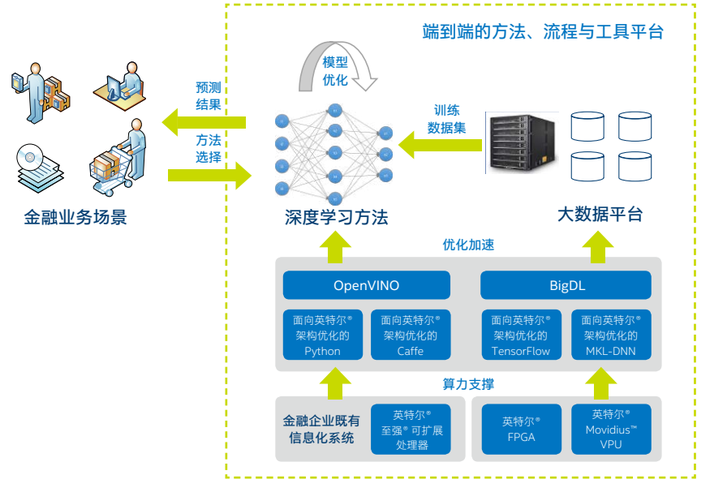


图 2-8-3　基于大数据平台构建深度学习方法需要端到端的方法、流程与工具平台

随着深度学习算法、算力和相关优化加速能力的持续提升，以及日趋成熟的金融大数据平台可为深度学习方法提供更多的优质训练数据集，越来越多的金融机构开始探索深度学习方法在资金流转预测等领域中的应用。这些方法包括多层感知机 (Multilayer Perceptron，MLP) 、差分整合移动平均自回归 (Autoregressive Integrated Moving Average Model，ARIMA) 、DNN、CNN、LSTM 等不同的算法模型。金融机构可以根据自己的应用目标和数据类型，选择最适宜的算法模型。下文，将对广泛地应用于资金流转预测解决方案的 MLP 模型做简单介绍。

其次，金融大数据平台往往采用分布式架构部署，因此在部署深度学习方法时，也需要有相应的应对方案。一些金融机构正尝试引入分布式深度学习框架，例如 BigDL，其是由英特尔开源、基于 Spark 计算引擎的分布式深度学习库。通过 BigDL 的引入，用户可以将资金流转预测等深度学习应用程序作为标准 Spark 程序，直接运行在现有大数据平台的 Hadoop/Spark 集群上。

深度学习方法的构建也需要大量的计算力资源予以支持，这既需要对用户既有信息化系统的资源 (具备丰富的通用处理器算力) 加以利用，避免重复投资，降低用户 CAPEX 成本；也需要根据处理器特性，开展针对性的优化，提升计算力效率。例如，当 AI 解决方案采用 TensorFlow 框架时，用户可以引入面向英特尔® 架构优化的 TensorFlow。优化的框架既可以利用英特尔® MKL 函数库对图运算过程进行优化，也可以对多个线程库进行优化，进而提升计算资源的利用率，加速深度学习方法的执行。

最后，在传统机器学习 / 深度学习方法中，需要开展大量的训练数据打标签、超参数调优等工作，这会耗费用户大量的人力资源。因此，为面向未来开展的深度学习方法探索，金融行业用户也迫切希望新方案具备深度学习方法自动调优能力。

■ 基于MLP模型的深度学习方法

MLP 模型因其简洁、高效和易于部署的特性，而被广泛地应用于资金流转预测解决方案中。典型的 MLP 模型架构如图 2-8-4 所示，其是一个由全连接层组成的神经网络，并至少含有至少一个隐藏层 (图 2-8-4 作为示例，只有一个隐藏层) ，且每个隐藏层的输出通过激活函数来进行变换，MLP 模型的超参数是网络层数和各隐藏层中隐藏的单元个数。

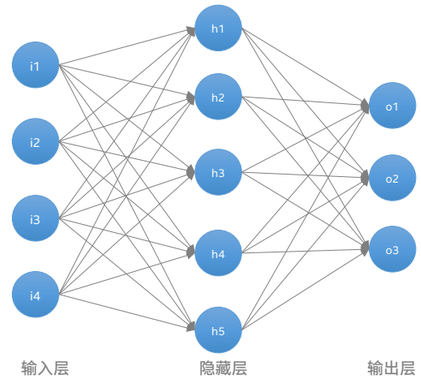


图 2-8-4　典型的 MLP 模型架构

MLP 模型中常用的激活函数包括了 ReLU 函数、Sigmoid 函数和 tanh 函数，其中：

- ReLU 函数提供了一个简单的非线性变换，对于给定元素 x，函数定义为 ReLU (x) = Max (x,0)。因此，ReLU 函数只保留正数元素，并将负数元素清零；
- sigmoid 函数可将元素的值变换到 0 和 1 之间，即 sigmoid(x) =1/(1+exp(− x))，根据链式法则，sigmoid 函数的导数为 sigmoid′ (x) = sigmoid(x)(1− sigmoid(x))；
- tanh 函数可以将元素的值变换到 -1 和 1 之间，即 tanh(x) = (1−exp(− 2x))/(1+exp(− 2x))。根据链式法则，tanh 函数的导数为 tanh′(x) =1 − tanh^2(x)，当输入为 0 时，tanh 函数的导数达到最大值 1；当输入越偏离 0 时，tanh 函数的导数越接近 0。

与其他深度学习方法相比，MLP 模型一方面有着良好的非线性全局作用和容错性，并具有联想记忆功能，在训练数据集规模较大的情况下，可获得优异的预测效果；另一方面，该模型也具有良好的并行处理能力，而这正是拥有众核、高频能力的英特尔® 架构处理器最为擅长之处，金融机构可以利用其既有信息化系统中大量的英特尔® 架构处理器算力来提升 AI 应用的效能。

在 MLP 模型之外，英特尔同时也在帮助金融机构利用其他深度学习模型，例如 DNN、CNN 以及 LSTM 等，在不同金融业务场景中开展预测、推荐等 AI 应用的探索。

■ 英特尔为深度学习方法探索提供端到端统一工具平台

除了需要确定方法、流程，金融机构在基于大数据平台开展深度学习方法探索时，还需要统一的、端到端工具平台，来承载这些方法和流程。由英特尔推出的开源 Analytics Zoo “大数据分析 +AI” 平台，一方面可以无缝地将 Spark、PyTorch、TensorFlow、Keras 等软件与框架集成到一个统一的体系中，并方便地扩展到 Hadoop/Spark 集群中，同时也融合了多种软件库，如英特尔® MKL、英特尔® MKL-DNN 等，能够充分释放第二代英特尔® 至强® 可扩展处理器等计算平台所集成的向量和深度学习指令，大幅提高 AI 应用的训练和推理速度。

通过内部集成的 Spark、BigDL 等模块，Analytics Zoo 能够与金融机构既有大数据平台架构“无缝融合”。如图 2-8-5 所示，一方面 Analytics Zoo 可以帮助用户将资金流转预测解决方案中，大数据平台和深度学习方法所需的 Spark、PyTorch 等软件和框架无缝集成到同一流程，有助于用户将数据存储、数据处理以及训练推理的流水线整合到统一的基础设施，从而大幅提升方案的部署效率、资源利用率和可扩展性，并减少硬件管理以及系统运维成本。

另一方面，Analytics Zoo 也针对不同的计算处理器平台，提供了大量多样化的优化函数库，如英特尔® MKL、英特尔® MKL-DNN，也针对不同的深度学习框架和代码语言，提供了基于处理器平台的优化版本，例如面向英特尔® 架构优化的 TensorFlow、面向英特尔® 架构优化的 Caffe 以及面向英特尔® 架构优化的 Python 等；另外，Analytics Zoo 还面向不同 AI 应用场景，预置了丰富的功能组件。例如，对于资金流转预测场景，Analytics Zoo 能够提供：

- **预测方法中常见的深度学习模型：** MLP、DNN、CNN、LSTM、MTNet、ARIMA 等；
- **预测方法中常用的数据预处理和特征工程：** Datetime features、Time diff、Log-transform、Rolling window 等；
- **预测方法中普遍的异常探测方法：** Percentile、Distribution -based、Uncertainty based、Autoencoder 等。

最后，在最新版本的 Analytics Zoo 中也加入了 AutoML 框架，用于自动化特征选择、模型选择和超参调优等，令预测模型工作效率获得进一步提升。

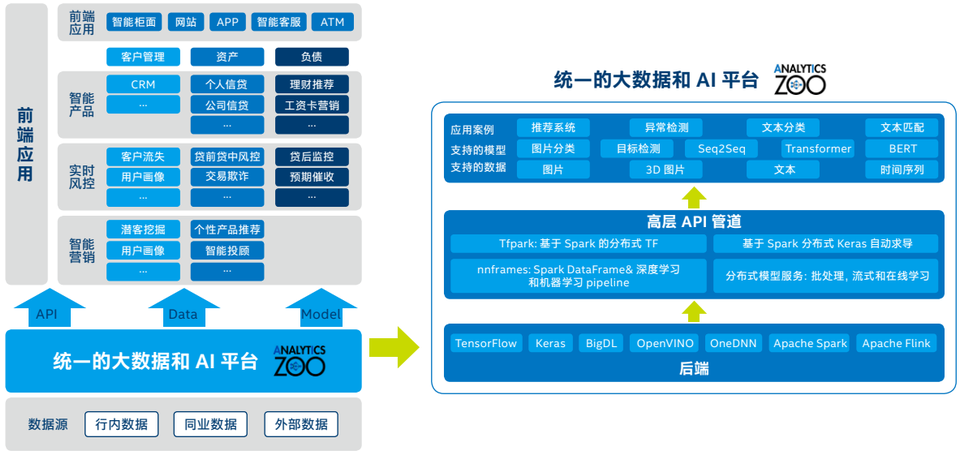


图 2-8-5　与金融大数据平台相结合的 Analytics Zoo

■ 面向未来的AutoML框架

一般地，传统深度学习或机器学习模型中的数据预处理、模型优化等工作，都需要富有经验的数据科学家来完成，这无疑提升了金融机构的系统维护压力和人力成本。为此，英特尔在其最新的 Analytics Zoo “大数据分析 +AI” 平台中，加入基于开源 Ray 分布式框架的 AutoML 框架，使特征生成、模型选择和超参数调优等流程实现了自动化，帮助用户进一步提升预测效率。

如图 2-8-6 所示，AutoML 框架主要由 FeatureTransformer、Model、SearchEngine 和 Pipeline 等组件构成，其中：

- FeatureTransformer 定义了特征工程流程，包括了特征生成、特征缩放和特征选择等操作；
- Model 定义了模型以及所使用的优化算法；
- SearchEngine 用于搜索 FeatureTransformer 和 Model 的最佳超参数组合，是控制模型训练过程的核心；
- Pipeline 配置了 FeatureTransformer 与 Model 的最佳端到端数据分析流水线，可反复加载使用。

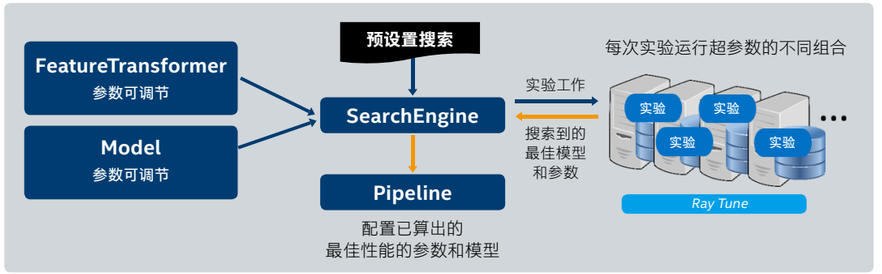


图 2-8-6　典型的 AutoML 执行流程

以金融行业的资金流转预测为例。首先，AutoML 会将参数调整后的 FeatureTransformer 和 Model 送入 SearchEngine 中进行实例化，SearchEngine 随后会借助 Ray Tune 在右侧集群中进行多轮的试验 (trail jobs)，每轮试验都会使用不同的超参数组合，进行特征工程以及模型训练。最后，系统会选出最优的一组超参数和模型的组合 (best model/parameters) 返回给 SearchEngine，然后送入 Pipeline 供后续的预测训练与推理使用。

更多 AutoML 的代码、Demo 和文档，请参阅：

- 在 Analytics Zoo Repo 中的 branch @ https://github.com/intel-analytics/analytics-zoo/tree/automl
- AutoML 自述文档 @ https://github.com/intel-analytics/analytics-zoo/blob/automl/pyzoo/zoo/automl/README.md
- Demo 手册 @ https://github.com/intel-analytics/analytics-zoo/blob/automl/apps/automl/nyc_taxi_dataset.ipynb

■ 优化方法示例

为帮助金融机构用户更快、更便捷地利用 Analytics Zoo，开展基于金融大数据平台的深度学习方法探索，并获得良好的应用效果，英特尔为用户提供了多样化的代码优化实例。以下为一段供参考的优化示例代码，并主要分为几个主要功能块：

- 定义 HADOOP_CONF_DIR 以及初始化 yarn；
- 数据的预处理，包括数据清洗去重、数据的拆分等；
- 定义模型的各种超参数，为后续模型的训练和预测做准备：
- 模型的准备工作，包括模型定义、特征集获取、准备交叉验证数据集等：
- 利用 Analytics Zoo 进行训练。

定义 HADOOP_CONF_DIR 和初始化 yarn：

```
from zoo.common.nncontext import *

if os.environ.get('PYTHONHOME') is not None:
    sc = init_spark_on_local(cores=1, conf={"spark.driver.memory": "20g"})
else:
    hadoop_conf_dir = os.environ.get('HADOOP_CONF_DIR')
    num_executors = 2
    num_cores_per_executor = 1
    os.environ['ZOO_MKL_NUMTHREADS'] = str(num_cores_per_executor)
    os.environ['OMP_NUM_THREADS'] = str(num_cores_per_executor)
    sc = init_spark_on_yarn(
        hadoop_conf=hadoop_conf_dir,
        conda_name=os.environ["ZOO_CONDA_NAME"], # conda 环境名称
        num_executor=num_executors,
        executor_cores=num_cores_per_executor,
        executor_memory="20g",
        driver_memory="20g",
        driver_cores=1,
        spark_conf={"spark.rpc.message.maxSize": "1024",
                    "spark.task.maxFailures": "1",
                    "spark.scheduler.minRegisteredResourcesRatio": "1",
                    "spark.scheduler.maxRegisteredResourcesWaitingTime": "100s",
                    "spark.driver.extraJavaOptions": "-Dbigdl.failure.retryTimes=1"})
```

数据的预处理：

```
from pyspark.sql import functions as F
from scipy.stats import mode

# 数据集预处理
import pandas as pd
data=pd.read_csv("gen_data.csv").set_index('Cust_Id')
data=data.fillna(0)

for i in data.columns:
  if data[i].dtype != 'float64':
    print(i,data[i].dtype)

# 去除非特征列
fea_notF = ['df_dt','core_cust_lev_cd','indus_cd','Rating','City_Cd']
data.drop(data[fea_notF],axis=1, inplace=True)


# 去除同值特征列
equi_fea = []
for i in data.columns:
  try:
    mode_value = mode(data[i][data[i].notnull()])[0][0]
    mode_rate = mode(data[i][data[i].notnull()])[1][0] / data.shape[0]
    if mode_rate > 0.6:
      equi_fea.append([i, mode_value, mode_rate])
  except Exception as e:
    print(i, e)
e = pd.DataFrame(equi_fea, columns=['col_name', 'mode_value', 'mode_rate'])
e.sort_values(by='mode_rate')

same_val_fea_to_drop = list(e.col_name.values)

for i in same_val_fea_to_drop:
  if i == 'if_lc':
    print(i)
    same_val_fea_to_drop.remove('if_lc')

data.drop(same_val_fea_to_drop, axis=1, inplace=True)

data[['if_lc']]=data[['if_lc']].astype('double')

from sklearn.preprocessing import StandardScaler

data_feature = data[data.columns[1:]]
data_label = data[data.columns[0]]

# 拆分数据集为训练集与验证集
from sklearn.model_selection import train_test_split
train_X,test_X,train_y,y_test = train_test_split(data_feature,data_label,test_size=0.3)
```

定义模型的各种超参数：

```
epochs = 30
batch_size = 1024
classes = 1
learning_rate = 0.01

scaler = StandardScaler()
train_X = scaler.fit_transform(train_X)
test_X = scaler.transform(test_X)
```

模型的准备工作：

```
import torch
import torch.nn.functional as F
from torch.utils.data import TensorDataset, DataLoader
from sklearn.metrics import roc_auc_score
from torch.autograd import Variable
from sklearn.model_selection import KFold
import numpy as np

# 定义模型
class MLP(torch.nn.Module):
    def __init__(self, n_feature, n_hidden, n_output, dropout=0.5):
        super(MLP, self).__init__()
        self.dropout = torch.nn.Dropout(dropout)
        self.hidden_1 = torch.nn.Linear(n_feature, n_hidden)
        self.bn1 = torch.nn.BatchNorm1d(n_hidden)
        self.hidden_2 = torch.nn.Linear(n_hidden, n_hidden//2)
        self.bn2 = torch.nn.BatchNorm1d(n_hidden//2)
        self.hidden_3 = torch.nn.Linear(n_hidden//2, n_hidden//4)
        self.bn3 = torch.nn.BatchNorm1d(n_hidden//4)
        self.hidden_4 = torch.nn.Linear(n_hidden // 4, n_hidden // 8)
        self.bn4 = torch.nn.BatchNorm1d(n_hidden // 8)
        self.out = torch.nn.Linear(n_hidden//8, n_output)

    def forward(self, x):
        x = F.relu(self.hidden_1(x))
        x = self.dropout(self.bn1(x))
        x = F.relu(self.hidden_2(x))
        x = self.dropout(self.bn2(x))
        x = F.relu(self.hidden_3(x))
        x = self.dropout(self.bn3(x))
        x = F.relu(self.hidden_4(x))
        x = self.dropout(self.bn4(x))
        x = self.out(x)
        return x

def sigmoid(x):
    return 1 / (1 + np.exp(-x))

# 将训练集分割为 5 折
folds = KFold(n_splits=5, shuffle=True, random_state=2019)
NN_predictions = np.zeros((test_X.shape[0], ))
oof_preds = np.zeros((train_X.shape[0], ))
x_test = np.array(test_X)
x_test = torch.tensor(x_test, dtype=torch.float)
test = TensorDataset(x_test)
test_loader = DataLoader(test, batch_size=batch_size, shuffle=False)
avg_losses_f = []
avg_val_losses_f = []

from bigdl.util.common import Sample

# 获取 FeatureSet
def get_featureset(x, y, shuffle=True):
    x = np.split(x.data.numpy(), x.shape[0])
    y = np.split(y.data.numpy(), y.shape[0])
    print(x[0].shape)
    print(y[0].shape)
    samples = [Sample.from_ndarray(np.squeeze(x[i]), np.squeeze(y[i])) for i in range(len(x))]
    sample_rdd = sc.parallelize(samples)
    return FeatureSet.sample_rdd(sample_rdd, shuffle=shuffle)

# 模型训练
for fold_, (trn_, val_) in enumerate(folds.split(train_X)):
    print("fold {}".format(fold_ + 1))
    x_train = Variable(torch.Tensor(train_X[trn_.astype(int)]))
    y_train = Variable(torch.Tensor(train_y[trn_.astype(int), np.newaxis]))

    x_valid = Variable(torch.Tensor(train_X[val_.astype(int)]))
    y_valid = Variable(torch.Tensor(train_y[val_.astype(int), np.newaxis]))

    model = MLP(x_train.shape[1], 512, classes, dropout=0.4)
    print(x_train.shape[1])
    loss_fn = torch.nn.BCEWithLogitsLoss()
```

利用 Analytics Zoo 进行训练：

```
from zoo.pipeline.api.keras.optimizers import AdamWeightDecay
    from zoo.pipeline.api.torch import TorchModel, TorchLoss
    from zoo.feature.common import FeatureSet
    from zoo.pipeline.estimator import *

    zooOptimizer = AdamWeightDecay(lr=learning_rate, weight_decay=1e-5)
    zooModel = TorchModel.from_pytorch(model.cpu())
    zooLoss = TorchLoss.from_pytorch(loss_fn)
    train_featureSet = get_featureset(x_train, y_train, shuffle=True)
    val_featureSet = get_featureset(x_valid, y_valid, shuffle=False)

    estimator = Estimator(zooModel, optim_methods=zooOptimizer)

    from bigdl.optim.optimizer import MaxEpoch, EveryEpoch
    from zoo.pipeline.api.keras.metrics import AUC
    estimator.train(train_featureSet, zooLoss, end_trigger=MaxEpoch(epochs),
                                checkpoint_trigger=EveryEpoch(),
                                validation_set=val_featureSet,
                                validation_method=[AUC()], batch_size=batch_size)
    torchModel = zooModel.to_pytorch()

# 预测
test_preds_fold = np.zeros((len(test_X)))
for i, (x_batch,) in enumerate(test_loader):
    y_pred = torchModel (x_batch).detach()
    test_preds_fold[i * batch_size:(i + 1) * batch_size] = sigmoid(y_pred.cpu().numpy())[:, 0]
    NN_predictions += test_preds_fold / folds.n_splits
for i in oof_preds:
  if i>0.5:
    print()

threshold = 0.5
result = []
for pred in NN_predictions:
    result.append(1 if pred > threshold else 0)

threshold = 0.5
result = []
for pred in oof_preds:
    result.append(1 if pred > threshold else 0)

# 计算 acc
from sklearn.metrics import accuracy_score
acc = accuracy_score(train_y, result)
```

某商业银行应用案例

案例背景

■ 大数据已成为金融行业的基础设施

代发工资是商业银行重要的业务之一，其是接受企业等用人单位的委托，将所需支付的劳动报酬等款项，通过转账划入指定的员工账户。这一业务的开展，可为商业银行带来了以下的收益：

- 企业对公账户和个人工资账户中的活期存款沉淀，一直是银行优质低息的纳储来源，通过代发工资业务，可以增加银行存款，帮助银行获得资金储备；
- 为银行注入流量，增加新的有效客户数，并带来后续设立信用卡、各类信贷业务收入 (汇兑、短信通知等等) 以及销售理财产品的机会，乃至为银行带来贷款用户；
- 代发工资业务，以及其配套的增值服务，可为银行带来持续的手续费收入，比如账户管理费、企业网银服务费、短信费等。同时，拥有工资卡的个人也能为银行带来年费、工本费、账户小额托管费等收入；
- 通过代发工资业务的开展，银行还可基于此推动更多业务的开展，创造延伸性服务空间，例如为企业和员工提供专项附加扣除信息收集等全流程服务，提升用户体验。

传统上，工资代发只是一种金融中间业务，很多个人客户在收到工资到账的信息后就把钱直接转走，银行并没有获得低成本高质量的资金沉淀，也无法开展服务延伸并从中产生利益，所以工资代发的后期管理就显得非常重要。在过去，商业银行通过客户经理对接、柜台业务推荐、广告手册等方式提高用户资金留存率。随着 IT 技术的不断进步，越来越多的商业银行正运用信息化技术，乃至 AI 能力，来构建资金流转预测系统，提升银行对金融中间业务的后期管理能力，实现对资金流转的精准把控，并助力赢得客户，提升竞争优势。

作为全球排名前二十的知名金融机构，某股份制商业银行也一直积极发展工资代发等金融中间业务。随着银行数字化服务的不断升级，某具有大量用户的商业银行也围绕客户信息、账户信息、资金流转信息等业务数据，在英特尔、Cloudera 等合作伙伴的助力下，构建了高效的金融大数据平台。并在此基础上，通过聚类算法、XGBoost 等机器算法的引入，开发部署了资金流转预测、业务推荐、主动营销等 AI 应用。

随着该行业务规模，尤其是个人客户规模的不断扩大，其金融中间业务涉及的数据量也大幅增加。同时，围绕工资代发业务开展的信用卡、个人理财、信用贷等业务，所具有的大量非线性、强相关和紧耦合特性，也使资金流转预测的复杂度急剧抬升。

为有效应对以上变化，对金融科技和数字化创新始终有着高度敏感性的该商业银行，与英特尔一起，在日益成熟的 Cloudera 金融大数据平台上探索构建深度学习方法，并将之应用于资金流转预测场景。为帮助用户更好地推进这一应用，英特尔除了为方案中的深度学习方法提供强劲的算力资源，以及软件调优方法，还为其提供了端到端的统一大数据 AI 平台 Analytics Zoo。

方案架构及部署成效

在银行看来，工资代发不仅是其吸纳存款、保证资金流通的重要来源，也是开展其他高质量业务，诸如信用卡、理财、个人贷款等的重要抓手。因此，银行希望基于深度学习方法，对工资代发后用户三天内的资金流转情况进行预测，从而可以根据不同的用户行为特征，制定更有效的产品营销方案。

通过充分的沟通和交流，在预测系统新方案的设计、构建与部署上，双方总结出了以下几个方面的需求：

- 新方案需要和银行既有的 Cloudera 大数据平台无缝对接；
- 新方案可以有效利用其信息化系统中英特尔® 架构处理器的计算力；
- 新方案可以面向不同应用场景，灵活使用不同的深度学习框架，例如其原有推荐系统使用的是 TensorFlow，但在预测系统中双方计划使用 PyTorch，方案需要对不同的框架都有良好的支持；
- 预测系统新方案需要达到 80% 的预测准确率（AUC 值）。

基于以上需求，如图 2-8-7 所示，英特尔与银行一起，规划设计了基于 Cloudera 大数据平台的深度学习架构，其核心是通过引入开源 Analytics Zoo 平台，在银行既有 Hadoop/Spark 大数据平台和基于深度学习的 AI 应用之间，构建起一条高速通道。

方案借由 Analytics Zoo，在 Cloudera 大数据平台之上构建 5 个主要能力，分别是深度学习框架、机器学习框架、模型训练 / 增量学习集群、模型分布式推理服务以及 AutoML 框架。

首先，面向大数据平台的分布式架构，Analytics Zoo 平台通过 BigDL、模型分布式推理服务的引入，实现了分布式的工作流对接。其次，针对预测系统中用到的 Pytorch 框架，用户可以将其直接部署在 Analytics Zoo 平台中。另外，Analytics Zoo 平台也为新系统提供了一系列易于使用的抽象和 API，例如传输学习支持、签名操作、Spark 数据帧、在线模型服务 API 等，用于开展模型训练和推理。

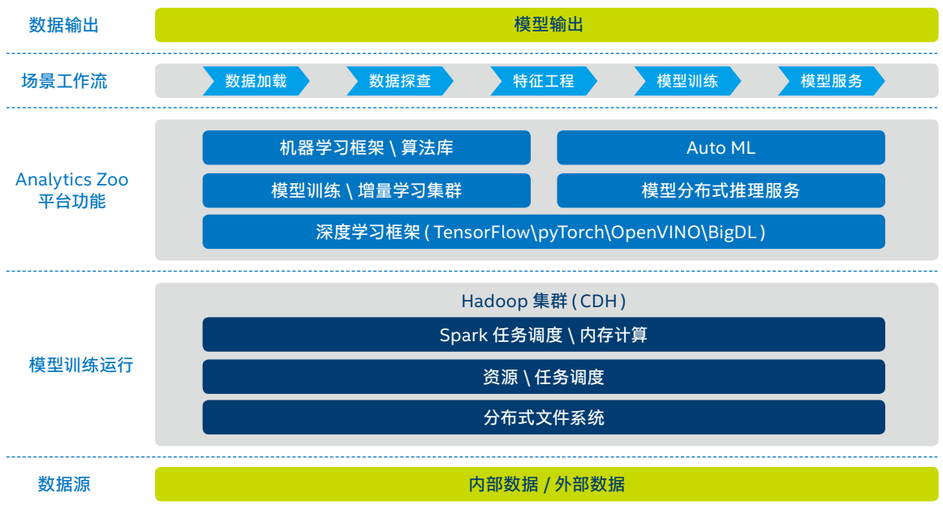


图 2-8-7　某商业银行基于大数据平台开展深度学习

同时，为了更好地利用其既有信息化系统中英特尔® 架构处理器提供的算力，Analytics Zoo 平台通过内置的各函数库，提供加速能力，提升其在预测系统的推理计算性能。

为了让用户更快地在 Analytics Zoo 平台上构建起基于深度学习方法的预测系统，英特尔帮助用户使用 PyTorch 框架，在平台上使用 MLP 模型进行预测系统代码的重构，并根据测试结果进行了多轮迭代优化。如图 2-8-8 所示，在经过两轮优化后，预测效果（AUC 值）达到 87%[38]，满足了用户的预设需求。

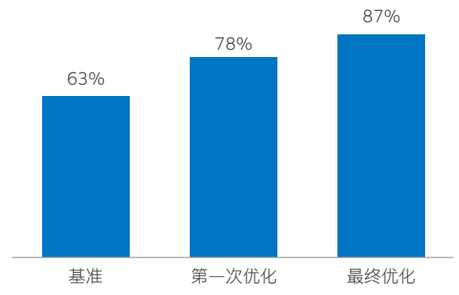


图 2-8-8　某商业银行优化效果 AUC 值

软、硬件建议配置

以上资金流转预测解决方案的构建，可以参考如下基于英特尔® 架构的平台完成，环境配置如下：

硬件配置

名称	规格
处理器	双路英特尔® 至强® 金牌 6248 处理器或更高
基础频率	3.00GHz（Turbo Boost @ 4.0GHz）
核心/线程	24/48
HT	On
Turbo	On
内存	96G (8G DDR4 2666MHz x12) （或根据具体数据量和模型尺寸相应匹配）
存储	1TB（或根据具体数据量和模型尺寸相应匹配）

软件配置

名称	规格
操作系统	CentOS Linux release 7.6.1810/Ubuntu16.04.6
Linux 内核	3.10.0-957.10.1.el7.x86_64
工作负载	CNN MLP
编译器	GCC 4.8.5
框架	PyTorch/Analytics Zoo 0.9.dev
Hadoop 版本	CDH5.15/CDH6.2.0

小结

资金流转管理在商业银行运营中占有重要的地位，是银行实现收益最大化的一大保证，也是抵御风险的基础之一。通过合作开发基于深度学习方法的资金流转预测解决方案，英特尔与用户一起，对金融行业实施从大数据到深度学习的“无缝切换”进行了探索，并获得了宝贵经验。这其中，围绕着 Analytics Zoo 平台开展的一系列工作流程、方法论以及工具平台的应用实践，被证明可以有效提升金融行业基于大数据平台探索深度学习方法的效率，并可以获得良好的预测效果。

帮助金融机构利用其既有的大数据平台和信息化系统资源，运用深度学习方法，在更多的金融业务场景中部署 AI 应用，是英特尔与合作伙伴们共同的探索目标。在未来，英特尔还计划与更多合作伙伴携手，进一步释放先进信息化产品与技术能力，更好挖掘金融数字化洪流蕴含的价值。

已有的案例解决方案，都采用了英特尔® 至强® 处理器 / 英特尔® 至强® 可扩展处理器平台。为获得更佳的影像分析效果，建议用户选择性能更强，在 AI 领域有着更多优化方法的第二代英特尔® 至强® 可扩展处理器来构建其解决方案。

[38] 测试配置：处理器：双路英特尔® 至强® 金牌 6248R 处理器，主频 3.00GHz，24 核心 /48 线程；内存：8GB DDR4 2666 x12；操作系统：CentOS Linux release 7.6.1810/Ubun16.04.6；Linux 内核：3.10.0-957.10.1.el7.x86_64；编译器：GCC 4.8.5；Hadoop 版本：CDH5.15/CDH6.2.0；工作负责：CNN MLP；框架版本：PyTorch/Analytics Zoo 0.9.dev

技术篇

第二代英特尔® 至强® 可扩展处理器

第二代英特尔® 至强® 可扩展处理器专为数据中心现代化变革而设计，提供比前代产品高出 25%-35% 的性能[1]，且具备多项新特性，提升了灵活性与安全性，增强了内存性能，能够帮助用户提高各种基础设施、企业应用及技术计算的运行效率，打造性能更强的敏捷服务和更具价值的功能，进而改善总体拥有成本（Total Cost of Ownership, TCO），提升生产力。

英特尔® 至强® 金牌处理器 6200 系列，特别是主流的英特尔® 至强® 金牌 6248 处理器、英特尔® 至强® 金牌 6240 处理器、英特尔® 至强® 金牌 6230 处理器，作为英特尔® 至强® 可扩展处理器平台的中流砥柱，能够支持更高的内存速度、增强的内存容量和四路可扩展性，并在性能、高级可靠性和硬件增强型安全技术方面取得了显著改进，且针对要求苛刻的主流数据中心、多云、网络和存储等工作负载进行了优化，能够适应更复杂、更多样化的应用场景。此外，英特尔® 至强® 金牌 6200 系列首次支持双 FMA 通道，意味着 FMA 性能提升了 2 倍[2]。

此外，第二代英特尔® 至强® 可扩展处理器集成深度学习加速技术（矢量神经网络指令 VNNI），扩展了英特尔® AVX-512，赋予平台更多、更强的 AI 能力，可加速人工智能和深度学习推理，并针对工作负载进行了优化。这使其拥有了集成 AI 加速能力的 CPU 架构。基于这一架构，大多数推理工作被集成在工作负载或应用程序中，让用户可以获得加速带来的性能和更高的灵活性等优势，在以数据为中心的时代，帮助在多云与智能边缘之间高效进行无障碍性能切换，以及 AI 开发与应用。

作为新一代至强® 可扩展平台的“核心”，第二代英特尔® 至强® 可扩展处理器支持英特尔® 傲腾™ 持久内存这一全新产品类别。而英特尔® 傲腾™ 持久内存通过与第二代英特尔® 至强® 金牌以及铂金处理器搭配，可以作为 DRAM 内存的有力补充，来显著提高系统性能，加速工作负载处理和服务交付。（具体请参考《英特尔® 傲腾™ 持久内存》部分）。

功能特性：

- 更高的每核性能：多达 56 核（9200 系列）和多达 28 核（8200 系列），在计算、存储和网络应用中，为计算密集型工作负载提供更高的性能和可扩展性。
- 基于 VNNI 的英特尔® 深度学习加速（英特尔® DL Boost）技术：提高了在 CPU 上运行人工智能推理的表现，与上一代产品相比，性能提升高达 30 倍[3]，有助于从数据中心到边缘，充分支持 AI 部署和应用。
- 业界领先的内存和存储支持，更大的内存带宽 / 容量：支持英特尔® 傲腾™ 持久内存，与传统 DRAM 结合使用可支持高达 36TB 的系统级内存容量；内存带宽和容量提高 50%[4]，每路支持 6 个内存通道和多达 4TB DDR4 内存，速度高达 2933 MT/s（1 DPC）。还支持英特尔® 傲腾™ 固态盘和英特尔® QLC 3D NAND 固态盘，对于数据密集型的工作负载，突破性的内存和存储内存创新可以显著提高其效率和性能。
- 英特尔® Infrastructure Management 技术（英特尔® IMT）：该资源管理框架能够将英特尔的多种能力结合起来，有效支持平台级检测、报告和配置。
- 面向数据中心的英特尔® Security Libraries（英特尔® SecL-DC）：该软件库和组件实现了基于英特尔硬件的安全功能。

作为至强® 平台的创新之作，第二代英特尔® 至强® 可扩展处理器，基于突破的设计，从平台层面融合计算、内存、存储、网络以及安全等功能，并将它们提升到了新的高度。

[1] https://www.intel.cn/content/www/cn/zh/technology-provider/products-and-solutions/xeon-scalable-family/2gen-data-centric-computing-article.html
[2]、[3]、[4] https://www.intel.cn/content/www/cn/zh/products/docs/processors/xeon/2nd-gen-xeon-scalable-processors-brief.html

适用于人工智能应用的第二代英特尔® 至强® 可扩展处理器

* 仅在特定处理器上受支持。

	英特尔® 至强® 金牌处理器（6200 系列）	英特尔® 至强®金牌处理器（6200 系列）						英特尔® 至强® 铂金处理器（8200 系列）	英特尔® 至强® 铂金处理器（9200 系列）
		英特尔® 至强® 金牌 6230 处理器	英特尔® 至强® 金牌 6230R 处理器	英特尔® 至强® 金牌 6240 处理器	英特尔® 至强® 金牌 6240R 处理器	英特尔® 至强® 金牌 6248 处理器	英特尔® 至强® 金牌 6248R 处理器		
普适的性能和安全性									
支持的最大内核数	24 核	20 核	26 核	18 核	24 核	20 核	24 核	28 核	56 核
支持的最高频率	4.4 GHz	3.90 GHz	4.00 GHz	3.90 GHz	4.00 GHz	3.90 GHz	4.00 GHz	4.0 GHz	3.8 GHz
支持的 CPU 路数	多达 4 路	多达 4 路	多达 2 路	多达 4 路	多达 2 路	多达 4 路	多达 2 路	多达 8 路	多达 2 路
英特尔® 超级通道互连（UPI）	3	3	2	3	2	3	2	3	4
英特尔® UPI Speed	10.4 GT/s	10.4 GT/s	10.4 GT/s	10.4 GT/s	10.4 GT/s	10.4 GT/s	10.4 GT/s	10.4 GT/s	10.4 GT/s
英特尔® 高级矢量扩展 512（英特尔® AVX-512）	2 FMA	2 FMA	2 FMA	2 FMA	2 FMA	2 FMA	2 FMA	2 FMA	2 FMA
支持的最高内存速度（DDR4）	2933 MT/s	2933 MT/s	2933 MT/s	2933 MT/s	2933 MT/s	2933 MT/s	2933 MT/s	2933 MT/s	2933 MT/s
每路支持的最高内存容量*	1 TB，2 TB，4.5 TB	1 TB	1 TB	1 TB	1 TB	1 TB	1 TB	1 TB，2 TB，4.5 TB	3.0 TB
16 Gb DDR4 DIMM 支持	●	●	●	●	●	●	●	●	●
采用矢量神经网络指令（VNNI）的英特尔® 深度学习加速（英特尔® DL Boost）	●	●	●	●	●	●	●	●	●
英特尔® 傲腾™ 持久内存模块支持*	●	●	●	●	●	●	●	●	
英特尔® Omni-Path 架构（独立式PCIe* 卡）	●	●	●	●	●	●	●	●	●
英特尔® QuickAssist 技术（集成在芯片组中）	●	●	●	●	●	●	●	●	
英特尔® QuickAssist 技术（独立式PCIe* 卡）	●	●	●	●	●	●	●	●	●
英特尔® 傲腾™ 固态盘	●	●	●	●	●	●	●	●	●
英特尔® 固态盘数据中心家族（3D NAND）	●	●	●	●	●	●	●	●	●
PCIe 3.0	●	●	●	●	●	●	●	●	●
英特尔® QuickData技术（CBDMA）	●	●	●	●	●	●	●	●	●
非透明桥（NTB）	●	●	●	●	●	●	●	●	●
英特尔® 睿频加速技术 2.0	●	●	●	●	●	●	●	●	●
英特尔® 超线程技术（英特尔® HT 技术）	●	●	●	●	●	●	●	●	●
节点控制器支持	●	●	●	●	●	●	●	●	●

* 仅在特定处理器上受支持。

了解更多第二代英特尔® 至强® 可扩展处理器信息，请访问：
https://www.intel.cn/content/www/cn/zh/products/processors/xeon/scalable.html

英特尔® 傲腾™ 持久内存

英特尔® 傲腾™ 持久内存是英特尔的革命性产品。其通过创建新的存储层来填补内存和存储之间的性能差距，从而颠覆传统的内存 - 存储架构，以合理的价格提供大型持久性内存层级，可为内存密集型工作负载、虚拟机密度和快速存储带来更优的性能，以及更加出色的效率和经济性。进而帮助用户通过比以往更快的分析、云服务、人工智能训练与推理以及下一代通信服务，来加速 IT 转型，满足数据时代的需求。

英特尔® 傲腾™ 持久内存兼顾了成本、容量、非易失性和性能等特性，且单一模块可提供 128/256/512 GiB 三种选择，并兼容 DDR4 插槽。在与传统 DDR4 DRAM 内存一同应用在基于第二代英特尔® 至强 ® 可扩展处理器的平台上时，可以更低的成本在 8 路系统上实现高达 24TiB 的容量（每路最高可配置容量达 3TiB 的傲腾持久内存），进而帮助用户在靠近处理器的内存系统上加载规模远超以往的数据集，满足包括内存数据库、更大规模数据集分析，以及 AI 推理与迭代等对大内存有苛刻要求的应用负载需求，让更多数据的处理和分析走向实时化。

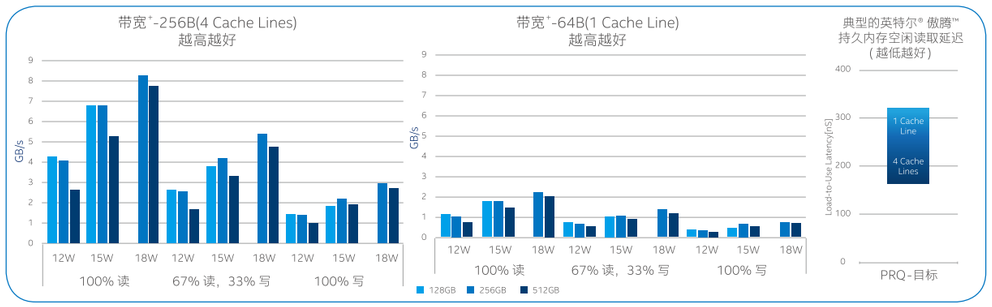
基于将内存和存储特性的融合，傲腾™ 持久内存具有两种不同的工作模式：内存模式和 App Direct 模式。通过这两种截然不同的工作模式， 客户可以灵活地跨越多个工作负载，显著提升系统性能。

内存模式。内存模式非常适合提供大容量内存，且不需要对应用进行更改。在内存模式下，CPU 内存控制器会将英特尔® 傲腾™ 持久内存用作可寻址的主内存，来扩展操作系统支持的可用易失性内存容量，而将 DRAM 用作其缓存。这可让虚拟化及容器技术直接受益，因为它可以借此以更低的成本在单个物理服务器上提升虚拟机或容器的密度，或为每个虚拟机及容器提供更大的内存容量，且无需重新编写软件。

App Direct 模式。在这一模式下，操作系统会将 DRAM 内存和英特尔® 傲腾™ 持久内存视为两个独立的内存池，使得英特尔® 傲腾™ 持久内存可以像内存一样寻址，并像存储设备一样具备数据持久性。这一数据持久特性，使得系统即使在维护或重启期间，持久内存也能保留数据，从而能够增加系统的业务弹性，缩短重启时间，并减少由此带来的成本。

双重模式。属于 App Direct 的子集，可通过预配置，让英特尔® 傲腾™ 持久内存部分处于内存模式，其余部分则处于 App Direct 模式，进而来满足具备双重需求的工作负载或应用环境需求。

英特尔® 傲腾™ 持久内存随其他英特尔® 至强 ® 可扩展处理器平台产品一并面市，并针对第二代英特尔® 至强 ® 可扩展处理器做了优化。其独到的应用优势在其上市伊始就吸引了很多 ISV 合作伙伴着手对旗下相关软件进行调优。同时，也有云基础设施及数据分析用户，也导入了英特尔® 傲腾™ 持久内存。这些合作伙伴和用户在相对应的应用调优和应用实践中，已初步见证了英特尔® 傲腾™ 持久内存在多重模式下的应用价值。



产品家族	英特尔® 傲腾™ 持久内存					
外形规格	持久内存模块 (PMM)					
PMem SKU[1]	128 GiB		256 GiB		512 GiB	
用户容量	126.4 GiB[8]		252.4 GiB[8]		502.5 GiB[8]	
MOQ	4	50	4	50	4	50
MM#	999AVV	999AVW	999AVX	999AVZ	999AW1	999AW2
产品代码	NMA1XXD128GPSU4	NMA1XXD128GPSUF	NMA1XXD256GPSU4	NMA1XXD256GPSUF	NMA1XXD512GPSU4	NMA1XXD512GPSUF
模型字符串	NMA1XXD128GPS		NMA1XXD256GPS		NMA1XXD512GPS	
技术	英特尔® 傲腾™ 技术					
有限保修	5 年					
平均年故障率 (AFR)	≤ 0.44					
耐用性 100% 写入 15W 256B	292 PBW		363 PBW		300 PBW	
耐用性 100% 写入 15W 256B	91 PBW		91 PBW		75 PBW	
耐用性 100% 读取 15W 64B	6.8 GB/s		6.8 GB/s		5.3 GB/s	
耐用性 100% 写入 15W 256B	1.85 GB/s		2.3 GB/s		1.89 GB/s	
耐用性 100% 读取 15W 64B	1.7 GB/s		1.75 GB/s		1.4 GB/s	
耐用性 100% 写入 15W 64B	0.45 GB/s		0.58 GB/s		0.47 GB/s	
DDR 频率	2666、2400、2133、1866 MT/s					
最大热设计功耗 (TDP)	15W		18W			
温度 (TJMAX)	≤ 84℃（85℃ 关闭，83℃ 默认）介质温度					
温度 （环境温度）	10W: 54℃ @ 2.4m/s					
温度 （环境温度）	12W: 49℃ @ 2.4m/s					
温度 （环境温度）	15W: 44℃ @ 2.7m/s					
温度 （环境温度）	N/A		18W: 40℃ @ 3.7m/s			

注：[1]GiB = 2^{30}；GB = 10^9

更多信息请参阅：
https://www.intel.cn/content/www/cn/zh/products/memory-storage/optane-dc-persistent-memory.html

英特尔® 傲腾™ 固态盘与基于英特尔® QLC 3D NAND 技术的英特尔® 固态盘

英特尔® 傲腾™ 固态盘和采用英特尔® QLC 3D NAND 技术的英特尔® 固态盘以创新的存储架构，助力数据中心面向未来，加速变革与跨越。

作为英特尔在固态盘产品线上的高端成员，英特尔® 傲腾™ 固态盘采用创新的 3D XPoint™ 存储介质，并结合了一系列的先进系统内存控制器、接口硬件和软件技术，在低延迟、高稳定等多方面均有上佳表现，可帮助消除数据中心存储瓶颈，并允许使用更大型、更经济实惠的数据集，进而加快应用程序速度、降低延迟敏感型工作负载的事务处理成本，并改善数据中心的 TCO。英特尔® 傲腾™ 固态盘这一更全面、更优秀，也更为均衡的 IT 基础设施能力，无疑能够为数据密集型的 AI 模型训练和推理带来更高的效率。以英特尔® 傲腾™ 固态盘 DC P4800X 为例，其具有高达 55 万 IOPS 的随机读写能力，低至 10 微秒的读写延迟，可更好地应对多用户、高并发场景，帮助应用方案获得更强的性能表现。同时，其写入寿命（Drive Writes Per Day, DWPD）高达 60，远超 NAND 固态盘，能够赋予存储系统更长的生命周期，也带来了更佳的经济性。

英特尔® 固态盘采用具有突破意义、可信的 3D NAND 技术来提升存储经济性，进而为替代传统硬盘（HDD）提供了性价比更高的选择，能够帮助用户改善体验、提升应用与服务的性能，并降低 TCO。作为固态盘中的“新兴生力军”，英特尔® 固态盘 D5-P4320 系列依托英特尔领先的 64 层 3D NAND 技术，可使得 QLC 固态盘单盘容量达到 7.68TB（TeraByte，万亿字节），从而有效应对数据中心等基础设施用户对“大容量”存储的需求。同时，其随机读取的 IOPS 高达 42.7 万，通过与第二代英特尔® 至强® 可扩展处理器搭配，尤其适用于 AI 训练等应用场景中对于“一写多读”的性能需求，为支持复杂的多样化工作负载提供了高效性、高稳定性和低能耗的存储框架。

了解更多信息，请访问：

- https://www.intel.cn/content/www/cn/zh/products/memory-storage/optane-memory/optane-memory-h10-solid-state-storage.html
- https://www.intel.cn/content/www/cn/zh/products/memory-storage/solid-state-drives/data-center-ssds.html

开源的统一的大数据分析+AI平台 Analytics Zoo

Analytics Zoo 是一个统一的大数据分析与 AI 开源平台，是为方便用户开发基于大数据、端到端的深度学习应用而推出。

Analytics Zoo 可帮助用户在 Apache Spark/Flink 和 Ray 之上，实现分布式的 TensorFlow、Keras、PyTorch 和 BigDL 程序，以及日后可能需要支持的其它框架等，无缝集成到一个管道之中；将这些模型透明地扩展到成百上千节点规模的大数据集群，进行分布式训练或推理，从而进一步简化了 AI 解决方案开发，且无需额外的专用基础设施。

为了提高计算性能，Analytics Zoo 融合了多种软件库，如英特尔® MKL 和英特尔® MKL-DNN。在硬件方面，它基于英特尔® 至强® 处理器平台，充分释放第二代英特尔® 至强® 可扩展处理器已集成的向量和深度学习指令，可大幅提高训练和推理速度。

将数据存储和处理流水线整合到一个统一的基础设施中，而无需移动数据的好处显而易见——不仅可提高开发部署效率和可扩展性、减少硬件管理和开发者学习新语言的时间、提高开发部署效率、资源利用率和灵活性，且降低总拥有成本，还不会影响计算效率与性能。开发人员需要的只是在扩展其 AI 解决方案时，充分利用 Analytics Zoo 提供的丰富特性和功能，以及多种分析和 AI 工具，即可实现大数据分析和 AI 的高效融合与部署、应用。

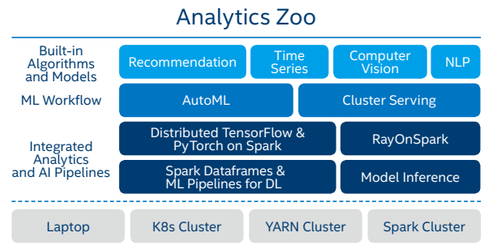
Analytics Zoo 平台所支持的众多 AI 框架中，BigDL 是英特尔自行研发和开源的。BigDL 是一个基于 Apache Spark 的分布式深度学习框架，可以无缝、直接运行在现有的 Apache Spark 和 Hadoop 集群之上，而不需要对集群做任何修改。基于 BigDL，开发者可以使用 Scala 或 Python 语言编写深度学习应用程序，并可以充分发挥 Spark 群集在可扩展方面的强大能力，推动大数据分析与 AI 的融合。在过去几年已经熟悉和启用 BigDL 的用户，可以通过 Analytics Zoo 直接调用 BigDL，无缝切换。

Analytics Zoo 技术特性：

- **端到到的流水线，应用 AI 模型（TensorFlow, PyTorch, OpenVINO™ 工具套件，等等）到分布式的大数据上：**
  - 以 Spark 代码为 TensorFlow 或 PyTorch 实现分布式的训练和预测
  - 在 Spark ML 流水线中，支持原生的深度学习（TensorFlow / Keras / PyTorch / BigDL）
  - 通过 RayOnSpark，在大数据集群上直接运行 Ray 的程序
  - 提供 Plain Java/Python APIs（TensorFlow/PyTorch/BigDL/OpenVINO™）以服务于 Model Inference
- **高抽象的 ML 工作流实现机器学习任务的自动化**
  - Cluster Serving 实现自动化分布式（TensorFlow/PyTorch/Caffe/OpenVINO™）的模型推理
  - 可扩展的 AutoML 服务于时序数据分析的预测
- **内建的模型服务于推荐系统，时序分析，计算机视觉和自然语言处理应用**

使用 Analytics Zoo 的理由：

- 可以轻松地将 AI 模型（例如 TensorFlow, Keras, PyTorch, BigDL, OpenVINO™ 工具套件等）应用于分布式大数据上。
- 可以通过“零”代码更改将 AI 应用程序从一台笔记本电脑透明地扩展到大型集群。
- 可以将 AI 流水线部署到现有的 YARN 或 K8S 集群，而无需对集群进行任何修改。
- 可以使应用机器学习的过程自动化（例如特征工程，超参数调整，模型选择，分布式推理等）。



了解更多信息，请访问：
https://software.intel.com/zh-cn/blogs/2018/09/10/analytics-zoo-unifying-analytics-ai-for-apache-spark?elq_cid=4287274&erpm_id=7282583

英特尔® 数据分析加速库

作为人工智能的一个分支，机器学习现在正获得极大的关注，基于机器学习的高级分析也越来越流行，其原因在于，与其它分析方法相比，机器学习能够帮助 IT 人员、数据科学家、各种业务团队及其组织迅速释放优势。并且机器学习提供了许多新的商用和开源解决方案，为开发人员提供了一个丰富的生态系统。同时开发人员可以选择各种开源机器学习库，比如 Scikit-learn，Cloudera 和 Spark MLlib 等。

英特尔® 数据分析加速库（英特尔®DAAL）

英特尔为行业用户部署机器学习，也推出了一套高性能系统化方案，涵盖处理器、经优化的软件和开发人员支持，以及强大的生态系统等丰富资源。

机器学习需要强劲的计算能力。英特尔® 至强® 处理器提供了一个可扩展的基准，专门用于满足机器学习所特有的高度并行工作负载，及其对内存和架构（网络）的需求。在英特尔的一项测试中，该处理器使系统训练时间减少了 50 倍 [1]。

此外，英特尔还提供了软件支持，包括：

- 在英特尔® 至强® 处理器上优化的库、语言以及构件模块，oneDNN 和**英特尔® 数据分析加速库（Intel® Data Analytics Acceleration Library，英特尔® DAAL）**，以及面向英特尔® 架构优化的 Python 分发包。
- 可简化开发的优化框架，包括 Apache Spark、Caffe、Torch 和 TensorFlow。英特尔支持开源软件和商用软件，使用户能够迅速利用市场上可获得的最新处理器和系统功能。

英特尔® DAAL 是一套旨在帮助数据科学家和分析师们快速建立从数据预处理，到数据特征工程、数据建模和部署的端到端软件方案。它提供了建立机器学习和分析所需的各种数据分析及算法所需的高性能构建模块。目前已经支持线性回归、逻辑回归、LASSO、AdaBoost，贝叶斯分类器、支撑向量机、K 近邻、Kmeans 聚类、DBSCAN 聚类、各种决策树、随机森林、Gradient Boosting 等经典机器学习算法。这些算法经过高度优化，可在英特尔® 处理器上实现高性能，如中国一家领先的大数据分析技术和服务提供商，使用这些资源已将多个数据挖掘算法提高了 3 到 14 倍 [2]。

为了开发人员在基于英特尔环境中的机器学习应用中更加方便地使用英特尔® DAAL，英特尔开源了整个项目：https://github.com/intel/daal，并针对不同的大数据使用场景，提供全内存的、流式的和分布式的算法支持。比如 DAAL Kmeans 可以很好地和 Spark 结合，在 Spark 集群上进行多节点聚类。另外，英特尔® DAAL 提供了 C++、Java 和 Python 接口。

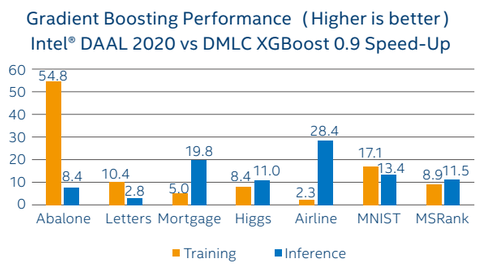
DAAL4py

为了更好地支持 Python 广泛应用最 Scikitlearn，英特尔® DAAL 提供了非常简单的 Python 接口 DAAL4py（开源地址：https://github.com/IntelPython/daal4py），它可以和 Scikitlearn 无缝的结合，在底层提供机器学习的算法加速。开发者无需修改 Scikitlearn 代码，即可利用自动向量化和多线程化的优势。目前 DAAL4py 在 Scikitlearn 中支持算法有：

- sklearn. 线性回归，sklearn. 岭回归，逻辑回归
- PCA
- KMeans
- pairwise_ 距离
- SVC（SVM 分类）

英特尔优化的 XGBoost

XGBoost 是一个基于递进 Gradient Boosting 的机器学习开源库，被广泛应用于各种分类和决策业务中。为了进一步加强其性能，英特尔优化和开源了代码库 [3]，最新的优化成果已经集成到 XGBoost 1.0 及之后的版本。相比 XGBoost 0.9 版，新版本性能提升 2 倍以上，最高达 54 倍。[4]



了解更多信息，请访问：https://software.intel.com/en-us/daal

[1、2] https://software.intel.com/zh-cn/articles/meritdata-speeds-up-a-big-data-platform
[3] Performance optimizations for Intel CPUs : https://github.com/dmlc/xgboost/pull/3957/files
[4] https://software.intel.com/daal

英特尔®深度神经网络库（oneDNN）

英特尔® 深度神经网络库（其前身是面向深度神经网络的英特尔® 数学核心函数库，Intel® MKL-DNN）是一款面向深度学习应用的开源性能增强库，也是英特尔为了帮助开发人员充分利用英特尔® 架构，推进深度学习的研究和应用而创建的基础库。（源代码地址：https://github.com/intel/mkl-dnn）

oneDNN 作为专为在英特尔® 架构上加快深度学习框架的运行速度而设计的一个性能增强库，包含了高度矢量化和线程化的构建模块，支持利用 C 和 C++ 接口实施深度神经网络，具备广泛的深度学习研究、开发和应用生态系统。目前已支持：TensorFlow、 PyTorch、MXNet、Caffe、Spark BigDL、OpenVINO ™ 工具套件等丰富的深度学习软件产品。

为了有效提升深度学习模型在英特尔® 架构基础设施上的运行速度，以及提升各类神经网络中其他性能敏感型应用的效率，oneDNN 提供了众多优化的深度学习运行和操作基元，可应用于不同的深度学习框架，以确保通用构建模块的高效实施。这些模块包括：

- 矩阵乘法和卷积：1D/2D/3D, Winograd 2D
- RNN 基元；
- 内积；
- 池化：最大、最小、平均；
- 标准化：跨通道局部响应归一化（LRN），批量归一化；
- 激活：修正线性单元（ReLU）；
- 数据操作：多维转置（转换）、拆分、合并、求和和缩放。

这些高效的函数模块可以应用于广泛的深度学习模型，如：

应用类型	拓扑结构
图像识别	AlexNet, VGG, GoogleNet, ResNet, MobileNet
图像分割	FCN, SegNet, MaskRCNN, U-Net
体积分割	3D-Unet
目标检测	SSD, Faster R-CNN, Yolo
机器翻译	GNMT
语音文字识别	DeepSpeech, WaveNet
对抗网络	DCGAN, 3DGAN
强化学习	A3C

为大幅提升了深度学习在 CPU 上的性能，英特尔还和众多开源社区合作，把该库集成进各种深度学习框架。如早在 2016 年，经过 oneDNN 优化的 Caffe，利用英特尔® 至强® 处理器 E5-2697 v3，相对于原始的 Caffe 性能获得高达 10 倍的提升 [1]。在 2019 年，经过优化后的 ResNet-50 也在英特尔® 至强® 铂金 9282 处理器上实现了每秒 7,736 张图像的领先性能 [2]。

oneDNN 目前已成为众多深度学习框架在 CPU 上运行时的基本配置。开发者可在深度学习框架的安装和应用中，直接获取 oneDNN 带来的性能提升。

了解更多信息，请访问：

- https://software.intel.com/zh-cn/articles/intel-mkl-dnn-part-1-library-overview-and-installation
- https://software.intel.com/zh-cn/articles/introducing-dnn-primitives-in-intelr-mkl

[1] https://software.intel.com/es-es/node/604830?language=en
[2] https://www.intel.com/content/dam/www/public/us/en/images/diagrams/rwd/xeon-scalable-max-inference-rwd.png

面向英特尔® 架构优化的Caffe

面向英特尔® 架构优化的 Caffe，从最初版本即集成了当时最新版的英特尔® MKL 并专门面向当时英特尔® 至强® 处理器产品已经集成的英特尔® AVX 2 和 英特尔® AVX-512 做了优化。它完全兼容 BVLC Caffe，并且包含 BVLC Caffe 的所有优势，且具备处理器优化功能，在基于英特尔® 架构的处理器上展现了非常出色的性能，并支持多节点分布程序训练。

面向英特尔® 架构优化的 Caffe 支持完备的 Post-training 量化方案，并在大量 CNN 模型中得到实践。特别是在基于第二代英特尔® 至强 ® 可扩展处理器的平台 (参考处理器介绍章节)，利用其集成的、对 INT8 有优化支持的英特尔® 深度学习加速技术 (VNNI 指令集)，可在不影响预测准确度的情况下，使多个深度学习模型在使用 INT8 时的推理速度能够加速到使用 FP32 时的 2-4 倍之多 (见下图)[1]，从而大大提升了用户深度学习应用的工作效能。

了解更多信息，请访问：

- https://software.intel.com/en-us/articles/caffe-optimized-for-intel-architecture-applying-modern-code-techniques
- https://software.intel.com/zh-cn/videos/what-is-intel-optimization-for-caffe

优化的深度学习框架和工具包

采用英特尔® 深度学习加速技术的 ResNet50 增益

第二代英特尔® 至强® 铂金8280处理器 vs 英特尔® 至强® 铂金8180处理器

英特尔®至强® 可扩展处理器	第二代英特尔® 至强® 可扩展处理器	mxnet	PYTORCH	TensorFlow	Caffe	OpenVINO
FP32	INT8 W/ 英特尔® DL Boost	3.0x	3.7x	3.9x	4.0x	3.9x
INT8	INT8 W/ 英特尔® DL Boost	1.8x	2.1x	1.8x	2.3x	1.9x

[1] 配置信息，请参见：https://www.intel.cn/content/www/cn/zh/benchmarks/server/xeon-scalable/xeon-scalable-artificial-intelligence.html

面向英特尔® 架构优化的TensorFlow

面向英特尔® 架构优化的 TensorFlow，是英特尔为了应对在 CPU 上运行深度学习模型面临的性能挑战而推出的优化版，能够确保深度学习类工作负载在各种情况下都可利用英特尔® MKL-DNN 基本运算单元高效运行。

为了显著提升性能，英特尔还持续采用其他举措对 TensorFlow 进行了优化。

计算图优化

英特尔推出了大量计算图优化通道，以便在 CPU 上运行时，将默认的 TensorFlow 操作替换为英特尔优化版本，确保用户能运行现有的 Python 程序，并在不改变神经网络模型的情况下提升性能；同时，消除不必要且昂贵的数据布局转换，以及通过将多个运算融合在一起，确保在 CPU 上高效地重复使用高速缓存，并处理支持快速向后传播的中间状态。

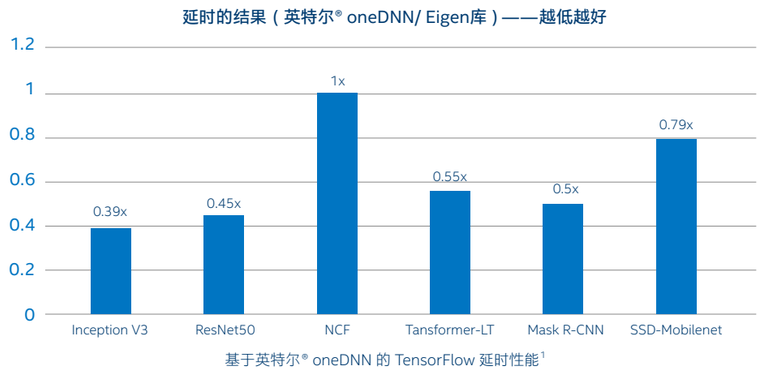
这些计算图优化措施进一步提升了性能，且没有为 TensorFlow 编程人员带来任何额外负担。此外，数据布局优化是一项关键的性能优化措施。此前，将来自 TensorFlow 本地格式的数据布局转换运算插入内部格式，在 CPU 上执行运算，并将运算输出转换回 TensorFlow 格式时，会因为转换造成性能开销的问题。而今利用英特尔® MKL 优化运算完全执行的子图，可消除子图运算中的转换。自动插入的转换节点能在子图边界执行数据布局转换，并通过另一个关键优化，即融合通道，将多个运算自动融合为高效运行的单个英特尔® MKL 运算。

其他优化

为确保在多种深度学习模型上实现最高的 CPU 性能，英特尔有针对性地调整了众多 TensorFlow 框架组件。比如，使用 TensorFlow 中现成的内存池分配器开发了一款自定义内存池分配器，确保其与英特尔® MKL 共享相同的内存池 (使用英特尔® MKL imalloc 功能)，从而不必过早地将内存返回至操作系统，避免了昂贵的页面缺失和页面清除。此外，对多个线程库 (TensorFlow 使用的 pthread 和英特尔® MKL 使用的 OpenMP) 页进行了深度优化，以使其能共存，避免了互相争抢 CPU 资源的情况，提升了资源的综合利用率。

了解更多信息，请访问：

- https://www.intel.ai/tensorflow/?_ga=2.231295069.330745958.1563951842597697079.1551333838&elq_cid=4287274&erpm_id=7282583
- https://www.intel.ai/improving-tensorflow-inference-performance-on-intel-xeon-processors/#gs.v0kayg



基于英特尔® oneDNN 的 TensorFlow 延时性能[1]

[1] 系统配置：英特尔® 至强® 铂金 8180 处理器 @2.50GHz; OS: CentOS Linux 7 （ Core ）; TensorFlow 源代码 : https://github.com/tensorflow/tensorflow; TensorFlow 版本号 : 355cc566efd2d86fe71fa9d755ceabe546d577a

面向英特尔® 架构优化的 Python分发包

面向英特尔® 架构优化的 Python 分发包，是一个由英特尔主导开发，功能强大的软件开发工具套件，提供了编写 Python 原生扩展所需的一切，包括 C 和 Fortran 编译器、数学库和分析器，并且集成了多个高性能数据分析和数学库，如 NumPy、SciPy、Scikit-learn、Pandas、Jupyter、matplotlib、mpi4py。

英特尔 ® Python 分发包是英特尔 ® Parallel Studio XE 的重要工具集之一，具备多项特性与高效率：

- 通过提供开箱即用的 uMath、NumPy、SciPy 和 Scikit-learn 等工具，加速包括数字、科学、数据分析、机器学习等在内的计算密集型应用。
- 集成英特尔® 性能库（如英特尔® MKL），内置最新的矢量化指令，如英特尔 ® AVX-512 和多线程指令、Numba 和 Cython，并应用对多线程构建模块库英特尔 ® TBB 的可组合并行，解锁 Python 基于多核处理器的并行应用功能，进而在基于英特尔® 架构的平台上提升 Python 程序运行性能，保证良好的系统兼容性，且不需要对代码进行任何更改。
- 支持 Python2.7、Python3.6 和最新一代英特尔 ® 架构处理器，提供优化的 TensorFlow、Caffe 等深度学习库和机器学习库，如支持向量机（SVM）和 K-means 预测、随机森林和 XGBoost 算法等，便于面向科学计算和机器学习等工作负载构建和扩展生产就绪算法。

经过基准测试，对比英特尔® Python 分发包与其它开放源码 Python 中 Scikit-learn 工具包（一个广泛用于数值计算、科学计算和机器学习的工具包）的效率，显示面向英特尔® 架构优化的 Python 指标有显著提升（见下图，效率越高，函数速度越快，与本机 C 语言速度越接近），如英特尔® Python 分发包 K-means 聚类、线性回归等算法的效率可以达到英特尔 ® DAAL 中 C 语言效率的 90%。

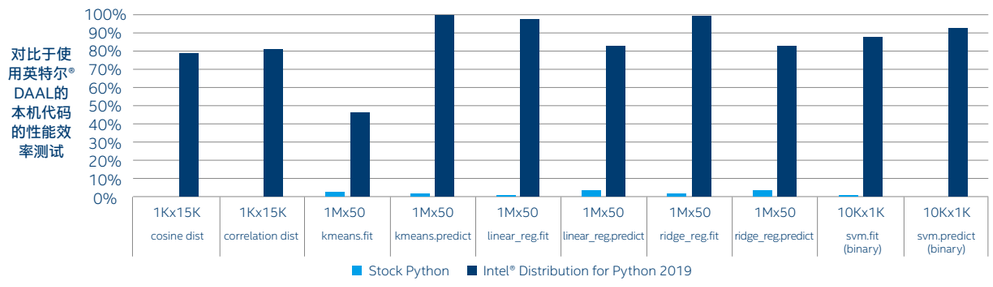
部署简单、易于使用也是英特尔® Python 分发包的一大特性——使用 Conda 软件包管理器和 Anaconda 云，用户只需执行一个命令，既可安装核心 Python 环境：

- conda install intelpython3 -c intel

此外，英特尔® Python 分发包是预先构建的二进制文件，也可以通过 pip、Docker images、YUM 和 APT repos 等各种渠道获得。

了解更多信息，请访问：

https://software.intel.com/en-us/distribution-for-python



英特尔的优化提升了 scikit-learn 程序包的效率，使其更加接近于本机代码在英特尔® 至强® 处理器上的运行速度

面向英特尔® 架构优化的 PyTorch

PyTorch 是一款开源的深度学习库，旨在帮助数据科学家和开发人员提高深度学习的训练和推理效率，具备高度的灵活性、易用性和训练、推理高速度，也深受业界青睐。

PyTorch 依托简洁、灵活的结构，提供了多项优异特性。比如，提供自动求导功能，使得模型搭建简单快捷；采用动态模型，在执行过程的任意环节都可以进行调试，大幅提升了易用性；能够自动计算梯度，并更新网络参数；训练方式灵活，允许用户自定义通讯方式，以实现新的通讯算法；提供了 Python、C++ 混合前端，可用 Python 前端做训练，而用 C++ 前端做部署。此外，PyTorch 具备强有力的社区支持，资源丰富。

为了基于 CPU 获得高性能、高效率，英特尔推出面向英特尔 ® 架构优化的 PyTorch。该优化版利用英特尔® MKL，使其能够充分运用 CPU 的并行计算能力和向量化技术，优来化矩阵乘法性能；采用英特尔® MKL-DNN，使其通过 CPU 的并行计算能力及向量化技术，并高效利用 CPU 片上缓存，来最大限度提高卷积、标准化等深度学习中常用运算性能。针对深度学习网络的层次或节点级的计算图特性，在节点级，英特尔优化了各种层，如卷积、矩阵乘法、ReLU、池等，最大限度地减少数据传输，并确保有效使用 SIMD 指令、执行单元、寄存器和内存缓存层次结构；在计算图级，英特尔优化措施使用多种数据顺序策略和层融合，来优化节点组，例如，将 ReLU 融合为卷积，以便数据在寄存器中的最后一个卷积周期仍执行 ReLU 操作。此外，借由英特尔® MKL-DNN API 关键的 DNN 层，还将英特尔 ® MKL-DNN 集成到 PyTorch 后端。

基于多项深度优化，面向英特尔® 架构优化的 PyTorch 经测试表明，基于英特尔® 至强® 铂金 8280 处理器，在 ResNet50、Faster R-CNN 和 RetinaNet 上的性能分别提高分别 7.7 倍、47 倍和 23.6 倍。[1]

此外，使用面向英特尔® 架构优化的 PyTorch 版本，通常不需要用户修改原有 Pythorch 脚本和代码。

了解更多信息，请访问：

https://software.intel.com/content/www/cn/zh/develop/articles/intel-and-facebook-collaborate-to-boost-pytorch-cpu-performance.html?wapkw=Pytorch&elq_cid=4287274&erpm_id=7282583

[1] https://software.intel.com/content/www/cn/zh/develop/articles/intel-and-facebook-collaborate-to-boost-pytorch-cpu-performance.html?wapkw=Pytorch&elq_cid=4287274&erpm_id=7282583

OpenVINO™ 工具套件

OpenVINO™ 工具套件是英特尔推出的一款加速深度学习推理及部署的软件工具套件，用以加快高性能计算机视觉处理和应用。该工具允许异构执行，支持 Windows 与 Linux 系统，以及 Python/C++ 语言，能够有效推进计算机视觉技术在从智能摄像头、视频监控、机器人，到智能交通、智能医疗等领域的深入应用。

本工具套件提高了计算机视觉解决方案的性能，缩短了开发时间，简化了从英特尔提供的丰富硬件选项中获得效益的途径，而这些选项可以提高性能、降低功耗并最大化硬件利用率——让用户可以低资源获得高收益，并为新的产品设计提供个性化空间。

通过基于深度卷积神经网络（CNN），扩展英特尔® 硬件（包括加速器）的工作负载，使得 OpenVINO™ 工具套件可依托英特尔® 架构处理器集成的显卡（Integrated GPU）、FPGA、英特尔® Movidius™ VPU 等芯片，来增强视觉系统的功能和性能。最新发布的 OpenVINO ™ 版本已能支持第二代英特尔® 至强® 可扩展处理器，并通过英特尔® AVX-512 以及采用 VNNI 的英特尔® DL Boost 技术来提升推理性能，可帮助客户在不改变软件的基础上，快速完成硬件产品升级和算法移植，从而助其在边缘侧快速实现高性能计算机视觉与深度学习应用的开发：

- 在英特尔® 架构平台上提升计算机视觉相关深度学习性能达 19 倍以上[1]；
- 释放 CNN-based 的网络在边缘设备的性能瓶颈；
- 对 OpenCV、OpenXV 视觉库的传统 API 实现加速与优化；
- 基于通用 API 接口在 CPU、GPU、FPGA 等设备上运行；
- 基于英特尔® 平台优化的 OpenVINO™ 工具套件主要包括深度学习部署工具包和传统的计算机视觉工具包两大部分。深度学习部署工具包括模型优化器（Model Optimizer）和推理引擎（Inference Engine）两个核心组件；
- 模型优化器：将给定的模型转化为标准的中间表示（Intermediate Representation，IR），并对模型优化，支持 ONNX、TensorFlow、Caffe、MXNet、Kaldi 等深度学习框架；
- 推理引擎：支持硬件指令集层面的深度学习模型加速运行，支持的硬件设备：CPU、GPU、FPGA、VPU。

同时，对传统的 OpenCV 图像处理库也进行了指令集优化，实现了性能与速度的显著提升。计算机视觉工具包括：

- OpenCV：预编译 OpenCV 和全新英特尔® 优化视觉库（Intel® Photography Vision Library），具有人脸检测 / 识别、眨眼检测、微笑检测等功能；
- OpenVX：基于图形的 OpenVX 实现，支持传统的 CV 操作和 CNN 原语。支持 Khronos OpenVX 神经网络扩展 1.2；
- 其他：包括 OpenCL™ 驱动程序和运行库，以及媒体驱动程序，以简化与英特尔® Media SDK 和英特尔® SDK OpenCL™ 应用程序一起工作的计算机视觉 SDK，英特尔® MKL-DNN、CLDNN 都包含在其中，不需要单独下载。

另外，作为旨在快速、有效地在多个应用中实施计算机视觉和深度学习而推出的工具包，基于英特尔® 平台优化的 OpenVINO ™ 工具套件目前提供预先转换的 Caffe、TensorFlow、MXNet 模型的 MO 文件，如 VGG-16, VGG-19、Squeezenet、ResNet、Inception、CaffeNet、SSD、Faster-RCNN、FCN8 等，还具备超过上百个预先训练的模型。软件开发人员和数据科学家等可以利用这些工具，快速实现个性化的深度学习应用，且可以使用 OpenCV、OpenVX 的基础库，去创建特定的算法，进行定制化和创新型应用的开发。

OpenVINO ™ 工具套件在英特尔平台上让视觉成为现实，已帮助众多用户轻松开发和快速部署计算机视觉应用程序，在多种深度学习应用场景展示了人工智能解决方案所蕴藏的巨大潜力。

了解更多信息，请访问：
https://software.intel.com/zh-cn/openvino-toolkit

提供跨平台工具，支持计算机视觉和深度学习推理加速

[1] https://software.intel.com/en-us/articles/a-guide-for-setting-up-docker-based-openvino-development-environment-with-ubuntu-system

英特尔® 软件防护扩展（英特尔® SGX）

英特尔® 软件防护扩展（Intel® Software Guard Extensions，英特尔® SGX）是一组新的指令集扩展与访问控制机制，支持英特尔® 至强® 处理器 E3-1500 第 5 和第 6 版、英特尔® 至强® 处理器 E2100 系列、第六代以上英特尔® 酷睿™ 处理器系列，以及英特尔® 赛扬® 处理器 J4105 或 J4005（带有支持英特尔® SGX 的 BIOS 型号）。其旨在通过基于硬件的隔离和内存加密方式，有效地防止应用程序代码和数据被泄漏或修改。通过英特尔® SGX，开发人员可将应用程序和敏感数据置入在特定硬件中划出的 “飞地”，来帮助获得硬件辅助的机密性和完整性保护，有效阻止来自更高特权级别进程的访问，进而增强安全级别，并具备多项特点：

- 增强机密性和完整添加保护
- 远程鉴证 & 供给
- 低学习曲线
- 助力显著减少攻击面

突破应用程序安全性限制

长期以来，开发人员受限于平台提供商为应用程序开发配置的安全性能。而今，英特尔® SGX 采用全新模型，既可以充分利用平台和操作系统（OS）优势来提升安全性，又帮助开发人员充分了解和决定哪些应用程序和机密数据需要额外保护，提升安全防范的自主性，进而使其充分释放基于芯片级安全技术所特有的全新性能优势，有效帮助应用程序根据开发人员的设置进行自我保护。

提供安全新途径

为助力解决众多的安全性漏洞和系统威胁问题，英特尔开发了基于硬件的可信执行环境（TEE），来减小攻击面。英特尔® SGX 提供全新的英特尔® 架构指令，既可以让应用程序使用这些指令划分更具隐私性的区域，也可用以选择代码和数据，防止存储在内存中正在执行的代码或数据遭到直接攻击。

增强联邦学习效率

基于英特尔® SGX 技术实施联邦学习时，AI 模型和训练数据都部署在受保护的硬件 “飞地”，能够大幅降低因应用程序和数据加解密带来的通信和计算成本，使执行效率更高。此外，采用英特尔® SGX 技术的应用程序可基于特定英特尔® 架构处理器平台进行开发、集成和执行，开发人员只需安装相关驱动并进行 SDK 适配，而无需熟悉额外的软硬件环境，编程方式也无需更改，无疑学习曲线更低。同时，与基于安全多方计算、同态加密等技术的联邦学习实现方法相比，基于英特尔® SGX 技术的硬件 TEE 方案运行效率更高。

阻止对应用程序实施内部攻击，更好地保护代码和数据

英特尔® SGX 应用程序

英特尔® SGX 应用程序包含两部分，即一个不可信任的启动组件和一个可信的组件，生产代码在后者 “飞地” 中运行。众多解决方案都可从英特尔® SGX 提供的额外保护中受益，包括人工智能（AI）和机器学习（ML）处理、密钥管理、专用算法、生物识别保护等等。开发人员可以通过建立协同运行的 1-n 个 “飞地”，来支持分布式架构。在程序运行时，英特尔® SGX 指令建立 “飞地”，并将其执行至特定的加密内存区域，开发人员可设置限定该区域的进入 / 退出，来防止数据泄露。

“飞地” 鉴证和数据密封

英特尔® SGX 支持 “飞地” 间本地鉴证或依赖方远程鉴证，确保应用程序不受损坏。其中，应用程序的一部分被加载到 “飞地”，进行代码和数据测量。然后，“飞地” 报告被发送到远程应用程序所有者的服务器上，进而服务器验证 “飞地” 报告是否由真实的英特尔处理器生成。

数据中心鉴证

英特尔® SGX 数据中心鉴证源码（英特尔® SGX DCAP）允许企业、数据中心和云服务提供商自行构建并交付鉴证服务，无需第三方提供商远程鉴证。

赋能安全性模型创新

英特尔® SGX 的基本功能是为平台的操作系统（OS）、应用程序及硬件的程序代码、数据、核心互联网协议（IP）提供更高级别的隔离和鉴证，以显著降低软件受到攻击的可能性，已被用于增强多种使用案例和应用程序的安全性，包括：

- 密钥管理
- 区块链
- 增强隐私性的分析和工作负荷
- 运行时的应用程序
- 硬件增强的内容保护
- 增强的应用程序和数据保护
- 边缘计算
- 数字钱包
- 通信和消息传递

英特尔® SGX 资源

英特尔® SGX 商业许可证信息：
https://software.intel.com/sgx/request-license

英特尔® SGX 软件开发工具包（SDK）：
https://software.intel.com/sgx/sdk

英特尔® SGX 文件和软件开发工具包（SDK）下载地址：
https://software.intel.com/sgx

本手册涉及的专业词汇表

英文缩写	英文全称	中文全称
ALS	Alternating Least Squares	交替最小二乘法
API	Application Programming Interface	应用程序编程接口
CART	Classification and Regression Tree	分类回归树
CNN	Convolutional Neural Networks	卷积神经网络
Data Mart	Data Mart	数据集市
DDR SDRAM	Double Data Rate Synchronous Dynamic Random-Access Memory	双倍速率同步动态随机存储器
Decision Tree	Decision Tree	决策树
DL	Deep Learning	深度学习
DMLC	Distributed Machine Learning Community	分布式机器学习社区
GBDT	Gradient Boosting Decision Tree	梯度提升决策树
GMF	Generalized Matrix Factorization	通用 (广义) 矩阵分解
GRU	Gated Recurrent Unit	门控循环单元
HDFS	Hadoop Distributed File System	Hadoop 分布式文件系统
HMM	Hidden Markov Model	隐马尔可夫模型
IAaas	Intelligent Analysais as a service	智能分析即服务平台
Imbalance Ratio	Imbalance ratio	非平衡性
Intel DAAL	Intel Data Analytics Acceleration Library	英特尔针对数据分析和机器学习的加速库
Layer Fusion	Layer Fusion	层融合
LR	Logistic Regression	逻辑回归
LSTM	Long Short-Term Memory	长短期记忆
MF	Matrix Factorization	矩阵分解
MKL	Math Kernel Library	数学核心函数库
MKL-DNN	Math Kernel Library for Deep Neural Networks	面向深度神经网络的数学核心函数库
ML	Machine Learning	机器学习
MLP	Multi-Layer Perception	多层神经网络
NBM	Naive Bayesian Model	朴素贝叶斯模型
NCF	Neural Collaborative Filtering	神经协同过滤
NLP	Natural Language Processing	自然语言处理
NNs	Neural Networks	神经网络
NUMA	Non- Uniform Memory Access Architecture	非统一内存访问架构
One-Hot	One-Hot	独热编码
POJO	Plain Ordinary Java Object	简单的 Java 对象
PR	Precision – Recall	精度 - 召回
RDD	Resilient Distributed Datasets	弹性分布式数据集
ResNet	Residual Net	残差网络
RF	Random Forest	随机森林
RNN	Recurrent Neural Network	循环神经网络
RS	Recommender System	推荐系统
SQL	Structured Query Language	结构化查询语言
Streaming Model	Streaming Model	流模块
SVM	Support Vector Machine	支持向量机
WAD	Wide and Deep	宽深
XGBoost	eXtreme Gradient Boosting	极端梯度提升

加速AI实践，请访问：

官网
Intel.cn/ai

微博
@ Intel Business

微信
英特尔商用频道