

인텔 금융업계 인공지능 실전 매뉴얼

인텔® 제온® 확장성 플랫폼
고효율 실행 AI



AI 실행을 가속화하고 싶다면, www.intel.com/ai 를 방문해 주십시오





목차 CONTENTS

업계 동향

6

- 금융업계 혁신을 지속적으로 실현하는 인공지능

실전편

12

- 빅데이터 + AI 기반 고효율 실시간 금융 거래 사기 방지 솔루션

13

- 인텔과 금융 사용자가 협력해 모색한 RNN 모델 이용 사용자 행동 특징 학습 사례
- 중국은행 응용 사례

19

- AI 기반 고효율 연체 대출 리스크 예측 솔루션

22

- 인텔과 금융 사업자가 협력하고 AI 모델을 이용해 진행한 연체 대출 리스크 예측
- 대형 상업은행 응용 사례

23

- AI 기반 금융 업계 정밀 마케팅 전략

32

- AI 기반 금융 업계 정밀 마케팅 전략 탐색
- 응용 사례

34

- AI 영상 분석 능력 가속화, 보험 업계에서의 AI 잠재력 촉진

35

- AI로 보험 업계 영상 분석 능력 가속화
- 중국 평안보험 응용 사례

44

48

49

53

기술편

58

- 하드웨어 제품

- 2 세대 인텔® 제® 온확장성 CPU

60

- 인텔® 옵테인™ 데이터 센터급 영구 메모리

62

- 인텔® 옵테인™ SSD 와 인텔® QLC 3D NAND 기술 기반 인텔® SSD

63

- 소프트웨어와 아키텍처

- DNN 지향 인텔® MKL

64

- 소스가 동일한 빅데이터 분석 +AI 플랫폼 Analytics Zoo

65

- 인텔® 아키텍처 최적화 지향 Caffe

66

- 인텔® 아키텍처 최적화 지향 TensorFlow

67

- 인텔® 아키텍처 최적화 지향 Python 배포 패키지

68

- OpenVINO™ 툴킷



업계 동향

금융업계 혁신을
지속적으로
실현하는
인공지능

오랜 시간 발전을 거듭해온 인공지능 (Artificial Intelligence, AI)은 현재 새로운 국면에 접어들었습니다. 인류 경제와 사회 생활에 혁신적인 영향을 미치는 인공지능 기술을 선택하는 기업이 점차 증가하면서 디지털 변화의 새로운 장이 열린 것입니다. 그 중 가장 도드라지는 분야는 금융 업계입니다. 사실 금융 업계 거물들에게 조금만 관심을 갖고 시선을 돌려도 그들이 지난 십 여 년간 빅데이터와 로봇, 그리고 클라우드 컴퓨팅에 얼마나 많은 자금을 투자했으며, 투자의 지지도 또한 매우 높음을 쉽게 알 수 있습니다. 다음은 신랑재경에 실린 기사 중 일부분입니다.¹

2000년에는, 월스트리트의 대표 투자은행 골드만 삭스의 뉴욕 본사 거래소에서 600명의 트레이더가 대형 거래처의 주문에 따라 주식을 사고 팔았다. 하지만 지금은 그 많은 트레이더 중 고작 3명만이 남았다.

미국 비즈니스 일간지 비즈니스 인사이더 (Business Insider)의 통계에 따르면, 현재 골드만 삭스의 전체 임직원 33,000명 중 9,000명 이상이 프로그래머와 엔지니어이며, 최근 몇 년간 골드만 삭스의 CEO 역시 골드만 삭스는 이제 하나의 기술 회사로서 그 포지션이 과거와 달라졌음을 공식적인 자리에서 여러 차례 강조한 바 있다.

월스트리트의 또 다른 거물인 JP 모건스 역시 유사한 전략을 취하고 있다. JP 모건스는 일찍이 기술 센터를 설립하고 4만 명의 기술인력을 고용해 빅데이터와 로봇, 그리고 클라우드 인프라를 전문적으로 연구해 왔다. 기술 예산은 \$ 96 억에 달하며, 이는 기업 총수익의 9%를 차지한다. 작년에는 세계 최초로 로봇을 이용한 글로벌 주식 알고리즘 거래를 진행하겠다고 발표한 바 있다. 그 전에도 JP 모건스는 유럽에서 연구개발한 인공지능 프로젝트 LOXM으로 이미 재미를 본 기업이다.

그렇다면 인공지능은 어떻게 금융 업계의 새로운 총아가 된 것일까? 그 원인을 분석해보면 인공지능과 금융은 여러 측면에서 태생적으로 밀접한 관계임을 발견할 수 있습니다. 첫째, 인공지능과 금융 모두 다량의 데이터를 기반으로 한다는 점이 인공지능의 모형 트레이닝과 예측 추론에 일종의 비옥한 토양을 제공했습니다. 둘째, 인공지능은 전통적인 방식에서 고객관계관리에 사용되는 다양한 서비스 비용을 절감해 줄 수 있습니다. 셋째, 인공지능은 금융 기업이 더 다양하고 정밀하게 타겟을 겨냥할 수 있도록 돋기 때문에 업무의 질을 향상시켜 줍니다. 사실 이보다 더 중요한 점은 다양한 혁신적 요소를 지닌 금융 업무가 인공지능과 결합하면 업계를 확대 발전시키고 새로운 가능성을 더 많이 창출할 수 있다는 것입니다.

예를 들어, 은행을 인공지능과 융합하면 방대한 양의 업무 데이터를 지능적인 방식으로 분석하고 예측할 수 있어 혁신을 가져올 수 있습니다. 또한 그를 통해 새로운 통찰력과 더욱 빠르고 효율이 높은 비즈니스 모델을 획득할 수 있을 뿐만 아니라 연체 대출, 금융 거래 사기 등 리스크를 미연에 방지하기 때문에 미래 경쟁 속에서 기선을 잡을 수 있게 될 것입니다.

¹ 관련 기사는 <http://finance.sina.com.cn/world/2018-05-22/doc-ihawmaua4464623.shtml>을 참조하십시오

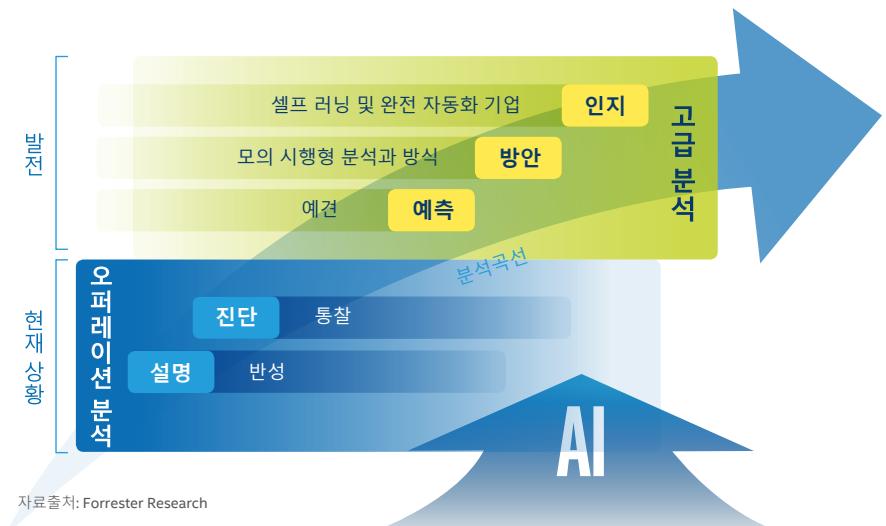


그림 1-1 기업 데이터 분석 기술의 발전

데이터 분석 기술은 단번에 인공지능으로 발전한 것이 아닙니다. 그림에서 보는 바와 같이 기업은 데이터 분석 기술이 발전하면서 일반적으로 규모와 성숙도 측면에서 설명, 진단, 예측, 솔루션 설계, 인지까지 다섯 단계를 거쳐 성장했습니다. 성숙한 인공지능 능력은 바로 이 다섯 단계에 모두 적용되며, 데이터 분석의 성숙도를 높이고 규모를 확대하는데 중요한 원동력입니다.

최근, 금융 업계가 보유한 데이터 양과 해시레이트, 알고리즘 모두 큰 진전을 이룩했으며, 금융 기업이 고품질 데이터 분석과 업무 예측을 실현하는데 인공지능이 중요한 수단이 될 것이라는 점은 논쟁할 여지가 없습니다. 또한, 컴퓨팅 기술은 금융 업계에서 응용하면서 거대한 가치를 창출하게 될 것입니다.

- **데이터 흥수 현상에 대응 :** 사물인터넷(IoT)의 광범위한 응용과 스마트 인터커넥트 시설의 빠른 보급으로 구조적 데이터와 비정형 데이터가 점차 증가하면서 데이터 흥수 현상이 이미 세계를 훔친 바 있습니다. 2020년에는 사물인터넷 중 스마트, 인터커넥트 시설의 수량이 2,000억 개, 이를 통해 생성되는 데이터는 2년마다 배로 늘어나 40ZB(1ZB=10조억 바이트)에 달할 것으로 예상합니다². 이처럼 방대한 데이터는 인공지능 알고리즘을 트레이닝하고 새로운

통찰력을 발굴하는데 초석을 마련할 것입니다.

• **계산력 돌파:** 무어의 법칙에 따라 칩 제조 기술과 아키텍처 혁신이 기존의 계산력을 끊임없이 추월하면서 계산집약형 알고리즘에 대한 수요를 만족시켰습니다. 예를 들어, 인공신경망을 통한 딥러닝의 개념은 적어도 지난 20년간 존재했지만 최근 몇 년까지는 컴퓨팅 기술이 인공지능에 필요한 고밀도, 고속의 연산 능력을 갖추어야만 계산집약형 알고리즘을 실행할 수 있었습니다.

• **혁신 추진:** 계산력과 데이터에 만의존해 인공지능을 발전시키기에는 턱없이 부족합니다. 인공지능을 대폭 발전시키는 주요 원동력은 바로 혁신으로, 혁신은 인공지능이 연구 단계에서 범용 단계로 뛰어넘을 수 있도록 추진할 것입니다. 인공지능 알고리즘이 혁신을 이룰 때마다 더 다양한 응용 가능성을 예시하고 더 많은 혁신기를 응용과 개발 대열에 끌어들일 수 있다는 점은 이미 실행을 통해 증명된 바 있습니다. 예를 들어, 20세기 90년대 뉴럴 네트워크의 혁신은 인공지능에 대해 새롭게 정의를 내렸고, 연구를 실현했습니다. 2009년과 2012년에는 음성 인식과 화상 인식 영역에서 획기적인 발전을 이루면서 현재 인공지능 혁신의 물결을 형성하는데

촉매제 역할을 하기도 했습니다.

그리하여 현재 인공지능 기술은 끊임없는 발전을 거쳐 금융업계의 프론트엔드, 미드엔드, 백엔드에서 모두 성숙한 응용 방안을 갖추게 되었습니다.

프론트엔드에서는 인식형 기술(컴퓨터 비전, 음성 인식, 자연어 처리 등)이 끊임없이 발전하고 있으며, 대표적인 기술로는 챗봇과 신원 인식 등이 있습니다.

챗봇은 고객 지원의 원칙을 준수하면서 머신 러닝 알고리즘을 빌어 대화를 관찰하고 고객의 의도를 파악합니다. 곧 한 상황이 발생하면 이슈를 인공지능에 발송해 처리하고 답변을 학습하면서 고객 서비스 품질을 지속 향상시키고 서비스 비용을 절감합니다. 신원 인식은 음성 인식과 안면 인식 등의 방식을 통해 사용자의 음성과 안구, 안면의 특징을 분석해 신원을 검증하는 기술입니다. 이러한 유형의 검증 방법은 기존의 보안 문제나 비밀번호 검증보다 효율이 높을 뿐만 아니라 사용자가 비밀번호를 암기할 필요가 없기 때문에 고객 경험 측면에서도 대폭 개선된 기술입니다.

미드엔드에서 인공지능은 정보 기반 분석 효율을 향상시켜 사용자가 더 순쉽게 비즈니스 기회를 잡을 수 있도록 도와 줍니다. 기존의 비즈니스 인텔리전스 (Business Intelligence, BI) 와 전통적인 방식의 데이터 분석은 경향 분석과 원인 규명, 데이터 발굴 및 예측 측면에 대부분 머물러 있습니다. 하지만 여기에 인공지능이 도입되면 분석의 범위를 확장하고 분석의 깊이까지 업그레이드할 수 있을 것입니다. 인공지능은 끊임없는 학습과 개선을 통해 연관성과 특이성을 향상시켜 "개인 맞춤형 분석"을 실현함으로써 리스크 관리와 마케팅, 서비스 등 분야에 지능형 분석과 솔루션을 제공합니다.

² 관련 기사는 다음을 참조하십시오. <https://cloud.tencent.com/info/5a0335164080e583fd6a2d4a735e7def.html>

예를 들어, 인공지능은 SNS의 신용 평가에 기반해 기준 평가 점수를 최적화하거나 신용 기록이 없는 사람의 신용도를 평가합니다. 뿐만 아니라 자연어 분석(Natural Language Processing, NLP)의 방식을 통해 분석 보고서를 만들 수 있으며, 재무 자료까지 분석 및 평가할 수 있습니다. 그 밖에도 인공지능은 금융 거래 사기 방지 상황을 능동적으로 탐지할 수 있으며, 실시간으로 일어나는 복잡한 거래 속에서 리스크를 발견하고 예방할 수 있습니다. 그보다 더 뛰어난 점은 인공지능이 고객의 행동을 연구해 개인 맞춤형 재무 컨설팅을 진행한다는 것입니다. 따라서 금융 기업은 인공지능을 이용해 고객과 제품 DNA에 따라 어디에도 없는 개인 맞춤형 마케팅 서비스를 제공할 수 있게 됩니다.

백엔드에서는 금융 업계의 법률 준수와 IT, 재무 등 지원 기능 중, 양이 방대하고 중복성이 높은 업무가 인공지능 응용에 중요한 분야 중 하나가 됩니다. 중복성이 높은 업무를 단계적으로 인수하고 있는 현재 단계에서 인공지능은 자연스레 다양한 응용 기회와 잠재력을 지니게 됩니다.

종합해 보면 프론트 앤드와 미드 앤드, 백 앤드에서 인공지능 응용은 주로 리스크 & 법률 준수, 고객 경험, 마케팅 의사결정과 스마트 운용 이 4 가지에 초점이 맞추어져 있습니다.

그림에서 보는 바와 같이 리스크와 법률 준수는 주로 사기 리스크와 신용 리스크, 매크로 리스크, 자금세탁방지와 법률 준수 정책 자료 분석 등을 포함하고 있으며,

인공지능 기술은 주로 머신 러닝, 딥러닝 기반 안면 인식과 음성 인식, 지식 그래프 등과 관련이 있습니다. 고객 경험은 주로 로봇 어드바이저, AI 보험금 지급 시스템, AI 기반 고객 서비스, 신원 인식과 가축 인식 등이 포함되며, 인공지능 기술 응용은 주로 딥러닝 기반 안면 인식과 음성 인식, 행동 패턴 인식, NLP 및 로봇 기술 등과 관련이 있습니다. 마케팅 의사결정은 고객 심층묘사, 정밀 마케팅, 자금 흐름 모니터링, 수량화 분석 등을 주로 포함하며, 인공지능 기술 응용은 딥러닝, 머신 러닝 등과 관련이 있습니다. 스마트 운용은 주로 네트워크 AI 레이아웃, 문서 식별, AIOps 등을 포함하며, 인공지능 기술은 주로 딥러닝 기반 화상 인식, 지식 그래프 등과 관련이 있습니다.

리스크 & 법률 준수

사기 리스크

신용 리스크

매크로 리스크

자금세탁방지

법률 준수 정책
자료 분석

고객 경험

로봇 어드바이저

AI 보험금 지급 시스템
(자동차보험/의료보험)

AI 기반 고객 서비스

신원 인식
(안면, 성문)

가축 인식

마케팅 의사결정

고객 심층묘사

정밀 마케팅

자금 흐름 모니터링

수량화 분석

스마트 운용

네트워크 AI
레이아웃

영수증 인식

AIOps

그림 1-2 금융 업계의 AI 응용 환경

다음 "실전편"에서는 금융 거래 사기 방지와 리스크 예측, 고객 마케팅, AI 언더라이팅 등 여러 환경에서 중국은련*, 차이나 생명보험 상하이 데이터 센터*, 마스터 카드*와 중국 핑안보험* 등 파트너사가 응용한 사례를 통해 실제 적용 예시를 상세히 설명할 예정입니다. 특히 인텔 연관 기술과 제품을 실제 환경에 응용하고 최적화할 수 있는 방안을 제시하고 하드웨어, 소프트웨어를 최적 상태로 구성해 실행하는 방안을 추천할 것입니다.



실전편

빅데이터 + AI 기반 고효율 실시간 금융 거래사기방지 솔루션



인텔과 금융 사용자가 협력해 모색한 RNN 모델이용 사용자 행동 특징 학습 사례

금융 거래 사기 방지 모델의 발전

금융 고객과 협력하는 과정 중, 우리는 금융 업계의 금융 거래 사기 방지 응용 모델 설계에 다음과 같은 문제가 있음을 발견했습니다.

- 사용자 행동 학습 알고리즘을 위한 실행 사례가 충분하지 않음
- 데이터 양에 대한 조건이 까다로운 전통적인 딥러닝 방식과 달리 실제 금융 기업은 모든 사용자의 행동 패턴 이력 데이터를 추출하기 힘든 상황
- 데이터 불균형 (Imbalance ratio) 이 심각한 상황. 대다수의 트레이닝 데이터는 모두 정상적인 거래 행위에서 유래하며, 정상/비정상 데이터 비율은 10 만 ~100 만대 1임.

전통적으로 금융 기업과 기관은 규칙을 기반으로 한 방식으로 금융 거래 사기 방지 리스크 모델링을 하는게 일반적입니다. 이러한 방식은 사용자의 행동 특징을 끊임없이 구축하고 갱신하는 규칙 베이스이기 때문에. 거래가 발생하면 시스템이 규칙 엔진을 통해 해당 거래에 잠재해 있는 리스크를 모니터링합니다.

예를 들어, 출장이 잦은 비즈니스맨의 경우 거래 규칙 베이스에서 거액의 해외 거래를 진행하는 것은 지극히 정상적인 일입니다. 하지만 거주지를 잘 벗어나지 않는 노인의 경우는 거주 지역에서 소액 거래를 자주 이용하는 것이 일반적이므로 거액의 거래 상황이 발생하면 규칙 엔진에 매칭되어 사용자에게 주의를 주게 됩니다. 타지에서 비정상적인 거액의 거래 기록이 여러 차례 발생하게 되면 해당 계정은 리스크 시스템에 의해 모니터링 범주에 포함되어 사후 감시 작업이 진행됩니다.

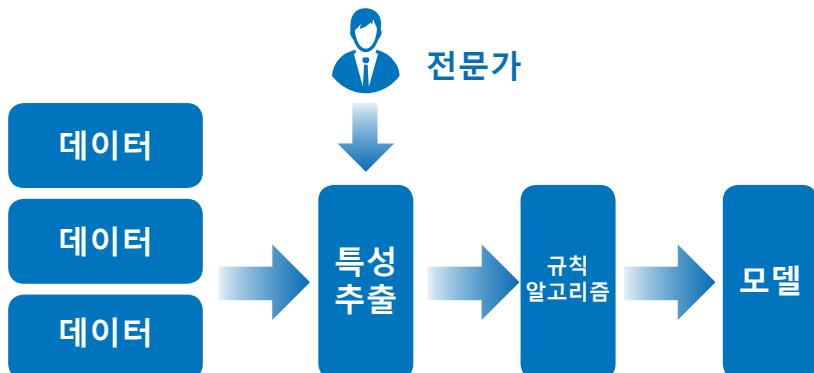


그림 1-1-1 전통적인 금융 거래 사기 방지 모델 방안

규칙 기반 리스크 관리 시스템은 효과적이긴 하지만 리버스 시스템이기 때문에 기존의 업무 사례를 통해 규칙 베이스를 끊임 없이 요약 해줘야 하는 단점이 있습니다. 이는 곧 사용자가 일정 시간마다 다양한 자원을 들여 업무를 요약하고 규칙을 간신해야 함을 의미합니다. 또한 업무 환경이 확장되면 거래 규칙 역시 더욱 복잡해지게 되고 결국 리스크 시스템의 자원 소모 상황과 모니터링 지연 현상이 지속적으로 증가할 것입니다. 이러한 문제를 해결하기 위해 금융 기관은 고효율의 금융 거래 사기 방지 모델을 구축할 수 있도록 인공지능 도입을 시도하기 시작했으며, 인공지능은 머신 러닝과 딥러닝 등 여러 방식을 바탕으로 하고 있습니다.

규칙 기반 방식과 비교했을 때, AI 금융 거래 사기 방지 방안은 객관성과 정확성이 더 높으며, 규칙을 셀프 러닝 할 수 있다는 점이 가장 주목을 끄는 점입니다. 통속적으로 규칙 기반 방식은 사전 알림 시스템으로 A 방법과 B방법이 틀리면 알림이 작동합니다. 하지만 머신 러닝과

금융 업계의 인공지능 시나리오

- ✓ 거래 특성에 따라 위조 카드, 현금깡, 허위 상점 등 사기를 식별할 수 있습니다



거래 원스 애기 반 해분 석 한 알고리즘 (계정 레벨)

- ✓ 계정의 비정상 거래 행위에 따라 도용 등 사기거래를 발견하고, 고객 심증묘사도 진행할 수 있습니다



그림 1-1-2 현재 금융 거래 사기 방지 모델에서 자주 볼 수 있는 알고리즘

딥러닝 방식은 다량의 데이터 이력을 학습 표본으로 삼고, 다양한 컴퓨팅 유닛을 통해 트레이닝을 진행해 하나의 평가 모델을 만듭니다. 새로운 거래가 이 단계에 진입하면 시스템은 거래의 적합성을 스스로 판단할 수 있게 됩니다.

그림 1-1-2 와 같이 현재 머신 러닝 중로지스틱 회귀 (Logistic Regression, LR), 랜덤 포레스트 (Random Forest, RF), 그라데이션 부스팅 의사결정 트리 (Gradient Boosting Decision Tree, GBDT) 등과 같은 일부 우수한 정렬 알고리즘은 이미 금융 거래 사기 방지 모델에 광범위하게 사용되고 있습니다.

RNN 기반 딥러닝 방식

딥러닝 방식은 금융 거래 사기 방지 AI 응용 중 많이 쓰이는 방법이며, 순환신경망 (Recurrent Neural Networks, RNN)은 금융 거래 사기 방지 모델 중 가장 자주 쓰이는 딥러닝 중 하나입니다. 전형적인 RNN 구조는 아래 그림 1-1-3 과 같으며, RNN은 시시각각 이루어지는 입력 데이터에 현재 모델의 상태를 결합해 결과를 출력합니다. 그림처럼 RNN의 주구조 A의 입력은 입력층의 X 외에도 순환을 거쳐 현재 시점의 상태가 제공되며, A의 상태 역시 현재 단계에서 다음 단계로 이전됩니다.

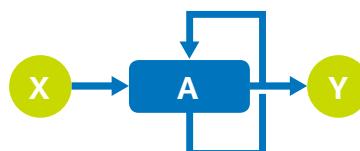


그림 1-1-3 전형적인 RNN 구조

RNN은 크게 LSTM(Long Short-Term Memory)과 게이트 순환 유닛 (Gated Recurrent Unit, GRU) 두 가지로 나뉘며, LSTM은 3 개의 특수 "게이트" 구조를 통해 설계되어 있기 때문에 RNN 네트워크 구조의 의존 문제를 예방하고 메모리 시간을 대폭 향상시킵니다. LSTM의 "게이트"는 Sigmod 신경망과 곱셈의 결합체로 Sigmod 를 활성화 함수로 한 완전 연결 신경망층이 0부터 1 까지의

값을 출력해 현재 해당 구조를 통과할 수 있는 입력 정보양을 설명합니다. Sigmod 출력값이 1 이면, 전체 정보가 통과됩니다. 반대로 Sigmod 출력값이 0이면, 어떠한 정보도 통과되지 않습니다. GRU 는 LSTM 의 업그레이드 버전으로 LSTM의 "게이트" 3 개를 2 개로 통합하기 때문에 현재 새로운 정보를 계산하는 방식이 LSTM 과 달라집니다.

RNN 기반 딥러닝 방식은 시퀀스 기반 분석 업무에서 줄곧 좋은 결과를 보여 왔습니다. 라벨링 기술을 기반으로 한 RNN 딥러닝 학습 방식은 다른 계정의 실시간 행동 상태를 분석하는데 사용할 수 있습니다.

하지만 딥러닝 방식만으로 방대한 양의 거래 데이터에 대해 금융 거래 사기 방지 모델링을 진행한다면, 그 효과는 결코 이상적이지 않을 것입니다. 그 이유는 RNN 과 같은 뉴럴 네트워크가 거래 시퀀스 간의 특징은 학습하지만, 단일 거래 학습 능력은 부족해 예상 목표를 달성할 수 없기 때문입니다.

예를 들어, 금융 거래 사기 방지 과정 중 늦은 밤 발생한 거액의 장외 거래"는 아주 특징되는 거래 행위로 인식하게 되지만, "장외", "거액", "늦은 밤"이 단독으로 등장 할 때에는 일반적인 특징으로 간주하기 때문입니다. 이러한 특징의 조합은 딥러닝 방식만으로는 이상적인 결과를 얻기 힘듭니다.

모델 실현 및 최적화

RNN 딥러닝은 Keras*와 TensorFlow* 등 딥러닝 플랫폼을 통해 실현할 수 있습니다. Keras 의 백엔드는 TensorFlow, Theano*, CNTK* 등 여러 가지 구조로 가능합니다. 다음의 알고리즘 모델은 TensorFlow 를 Keras 의 백엔드로 선택한 예를 설명한 것이며, \$HOME/.keras/keras.json 의 부분을 수정해 설정을 진행할 수 있습니다.

1. "backend": "tensorflow"

■ 인텔® 아키텍처 최적화를 지향하는 TensorFlow 설치 방법

TensorFlow는 현재 딥러닝 영역에서 주요 구조 중 하나로 딥러닝 개발자가 효율적으로 컴퓨팅 리소스를 이용해 딥러닝 모델을 구축할 수 있도록 도와줍니다. 인텔® 아키텍처를 충분히 이용하고 더 높은 성능을 실현하기 위해 TensorFlow는 이미 딥러닝 신경망 지향의 인텔® MKL-DNN(Intel® Math Kernel Library for Deep Neural Networks, 인텔® MKL-DNN)을 사용해 최적화를 진행했습니다.

Anaconda*는 Python*의 배포 버전으로 사용자가 인텔® MKL-DNN을 사용해 TensorFlow(TensorFlow v1.9부터 해당 기능 지원)를 구축하도록 돕고 인텔® 아키텍처의 CPU에 더 높은 성능을 제공합니다. 사용자가 Conda* 소프트웨어 패키지 관리자를 사용해 환경과 소프트웨어 패키지를 관리하려면 가상 환경에 Anaconda.org의 TensorFlow 소프트웨어 패키지를 설치하기만 하면 됩니다.

Anaconda에 TensorFlow 명령을 설치하는 방법은 다음과 같습니다.

1. conda install tensorflow

사용자의 Anaconda 채널이 기본적인 최상위 등급의 채널이 아닐 경우, 아래 명령을 사용해 인텔® 아키텍처 최적화 지향 TensorFlow를 획득할 수 있습니다.

1. conda install -c anaconda tensorflow

위의 설치 방법 외에도 인텔® 아키텍처 최적화 지향 TensorFlow는 wheel과 docker 이미지 또는 Conda 소프트웨어 패키지 형태로 배포할 수 있습니다.

그 밖에도 사용자는 PIP를 통하여 기존 Python 환경에 인텔® 아키텍처 최적화 지향 TensorFlow를 설치할 수 있으며 명령은 다음과 같습니다.

1. pip install intel-tensorflow

또는 사용자는 기존의 Python 환경에 wheel을 설치할 수 있으며, 이 경우 인텔® Python 배포 패키지 사용 하길

권장합니다. Python2.7과 Python3.6에 1.13.1 버전을 설치하는 명령은 각각 다음과 같습니다.

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

■ 인텔® 아티텍처 지향 TensorFlow 최적화 방법

인텔® 아티텍처 지향 플랫폼으로 TensorFlow를 최적화하면 모델 업무의 효율을 효과적으로 향상시킬 수 있습니다. 최적화 방법은 다음과 같습니다. CPU 핵심 수량을 조정하고 NUMA(Non-Uniform Memory Access Architecture) 기술 및 인텔® MKL-DNN을 도입합니다. 최적화 절차는 아래와 같습니다.

■ 환경 변수 설정

우선, 환경 변수를 설정해야 하며 명령은 다음을 포함합니다. 시스템의 캐시를 정리해 CPU를 최고 성능 모드로 설정하고 최고 주파수에서 실행해 CPU의 터보 부스트를 엽니다. 설정 명령은 아래 보는 바와 같습니다.

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

- KMP_BLOCKTIME를 1로 설정합니다. 이는 스레드 현재 임무를 실행 완료하고 휴면 상태에 들어가기 전까지 대기하는 시간 설정으로 일반적으로 1밀리초로 설정합니다.
- KMP_AFFINITY를 Compact로 설정합니다. 이는 해당 모드에서 스레드 바인딩이 컴퓨팅 코어의 컴퓨팅에 따라 우선 순위를 정해 동일한 코어를 우선 바인딩하고, 그 다음으로 동일 CPU의 다음 코어를 바인딩하는 것을 의미합니다. 이러한 바인딩 방식은 스레드 사이에 데이터 교류가 이루어지거나 공용 데이터를 갖춘 상황에 적용되며, 다양한 계층의 캐시를 충분히 이용할 수 있다는 점이 장점입니다.
- OMP_NUM_THREADS를 20으로 설정하면 스레드의 수량을 일정 물리 코어 수로 병행 설정하게 됩니다.

■ 테스트 코드에 스레드 컨트롤 추가

스레드 컨트롤 추가 코드는 다음과 같습니다.

```
1. config = tf.ConfigProto()
2. config.allow_soft_placement = True
3. config.intra_op_parallelism_threads = FLAGS.num_intra_threads
4. config.inter_op_parallelism_threads = FLAGS.num_inter_threads
```

그림에서 보는 바와 같이 `tf.ConfigProto()`를 초기 설정할 때, `intra_op_parallelism_threads` 파라미터와 `inter_op_parallelism_threads` 파라미터를 설정해 연산자 `op`로 병렬 컴퓨팅한 모든 스레드 개수를 컨트롤할 수도 있습니다. 두 방법의 차이는 다음과 같습니다.

- `intra_op_parallelism_threads`는 연산자 `op`의 내부 병렬을 컨트롤합니다. 연산자 `op` 가 단일 연산자이고 행렬 곱셈, `reduce_sum`과 같이 내부에서 병렬 실행이 가능한 경우, `intra_op_parallelism_threads` 파라미터를 설정해 병렬을 실행할 수 있으며, `intra`는 내부를 의미합니다.
- `intra_op_parallelism_threads`는 여러 연산자 `op` 사이의 병렬 컴퓨팅을 컨트롤합니다. 연산자 `op`가 여럿이고 서로 독립적이며 연산자와 연산자 사이에 직접적인 경로 없이 `Path` 할 경우, TensorFlow는 컴퓨팅을 시도하게 되고, `inter_op_parallelism_threads` 파라미터로 수량 중 하나의 스레드 풀을 컨트롤합니다.

통상적으로 `intra_op_parallelism_threads`는 단일 CPU의 물리적 코어 수량으로 설정하지만 `inter_op_parallelism_threads`는 1 또는 2로 설정합니다.

■ NUMA 특성을 이용해 CPU 컴퓨팅 리소스 사용 컨트롤

데이터 센터가 사용하는 서버는 일반적으로 2대 이상의 CPU를 구성하며, 대다수가 NUMA 기술을 사용해 여러 서버를 단일 시스템처럼 운영합니다. CPU가 자신의 로컬 기억 장치에 접근하는 속도는 타지의 기억 장치에 접근하는 속도보다 빠릅니다. 이러한 시스템에서 더 뛰어난 컴퓨팅 성능을 획득하기 위해서는 일부 특정 명령을 통해 컨트롤해야 합니다. `numactl`이 바로 과정을 컨트롤하고 기억을 공유하는 일종의 기술 매커니즘으로 Linux* 시스템에서 광범위하게 사용되는 컴퓨팅 리소스 컨트롤 방식입니다. 구체적인 사용 방법은 아래와 같습니다.

```
(root@worker105: tf_cnn_benchmarks]# numactl -H
available: 2 nodes (0-1)
node 0 cpus: 0 1 2 3 4 5 6 7 8 9 10 11 12 13 14 15 16 17 18 19 40 41 42 43 44 45 46 47-48 49-50 51 52 53-54 55-56 57-58 59
node 0 size: 56366 MB
node 0 free: 56366 MB
node 1 cpus: 20 21 22 23 24 25 26 27 28 29 30 31 32 33 34 35 36-37 38 39 48 61 62 63 64-65 66-67 68-69 70-71 72 73-74 75-76 77-78 79
node 1 size: 58384 MB
node 1 free: 55939 MB
node distances:
node 0  1
 0: 10 21
 1: 73 10
```

그림 1-1-4 NUMA 특성을 이용해 CPU 컴퓨팅 리소스 사용 컨트롤

Python 명령을 사용한 `numactl` 실행 명령은 다음과 같습니다.

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

명령 중 `test.py` 이 실행 중일 경우는 CPU#CPU0 중 0-19 와 40-59 코어만 사용했으며, 사용한 메모리 역시 CPU#CPU0 에 대응하는 메모리만 사용했습니다.

■ Keras/TensorFlow 다층 LSTM/GRU 실현

Keras에서 다층의 LSTM/GRU를 실현하는 코드는 다음과 같습니다.

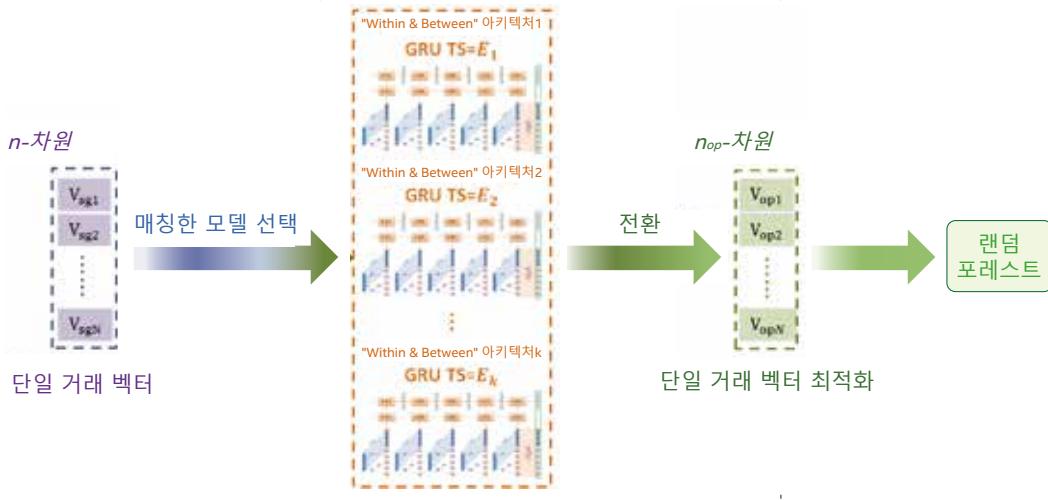
```
1. classifier.add(GRU(number_of_cells=128,
2.   input_shape=(timesteps, data_dim),
3.   recurrent_dropout=0.3,
4.   activation='sigmoid',
5.   recurrent_activation='hard_sigmoid',
6.   return_sequences=True))
7. classifier.add(GRU(number_of_cells=64,
8.   recurrent_dropout=0.3,
9.   activation='sigmoid',
10.  recurrent_activation='hard_sigmoid'))
```

* 인텔® 아키텍처 최적화를 지원하는 TensorFlow 기술과 관련한 더 자세한 사항은 해당 매뉴얼 기술편 소개 부분을 참조하십시오.

"샌드위치" 다층 금융 거래 사기 방지 모델

■ 모델 소개

그림 1-1-5에서 보는 바와 같이 "GBDT—>GRU—>RF" 3 층 구조로 이루어집니다. 우선, 이 구조는 단일 딥러닝 방식 (예: RNN)이 단일 거래에서 학습 능력이 부족한 상황을 탐지하고 있으며, 인텔이 제공하는 Analytics Zoo 툴을 통해 아키텍처의 프론트엔드에 GBDT 모델을 도입해 특징을 최적화합니다. 최적화가 완료된 특징은 인공 특징과 결합해 GRU 네트워크의 입력 요소로 삼아 시퀀스 간의 특징을 학습하고 단일 거래의 특징을 시간순서화 합니다. 이러한 과정을 통해 데이터를 효과적으로 필터링할 수 있으며, 이를 통해 백엔드의 GRU 모델에 유용한 데이터를 제공할 수 있습니다.

그림 1-1-5 GBDT—>GRU—>RF "샌드위치" 아키텍처 금융 거래 사기 방지 모델³

중간층은 GRU 네트워크 출력을 직접적인 금융 거래 사기 방지 판단 기준으로 사용하지 않고 그것을 시퀀스 사이의 특징을 학습하는 하나의 서클로 삼습니다. 학습을 통해 얻은 시퀀스 사이의 특징은 기존 거래의 특징과 결합해 거래 특징의 벡터를 형성합니다.

마지막으로는 시퀀스 특징을 한 단계 업그레이드해 융합 학습할 수 있도록, 아키텍처의 마지막 부분에 마지막 금융 거래 사기 방지 판별 분류기로 최고급 RF모델을 추가했습니다.

■ 소프트웨어 스택

"샌드위치" 다층 금융 거래 사기 방지 모델의 소프트웨어 스택은 그림1-1-6과 같으며, 인텔® 제온® 골드 CPU(6000 시리즈)를 기초로 구성한 하드웨어 인프라입니다. RedHat® Linux 운영체제(Centos7.4 Kernel 3.10.0-957.12.1.el7.x86_64)로 가상화 소프트웨어로 만든 가상 컴퓨터입니다. 가상 컴퓨터에 인텔® MKL-DNN 또는 인텔® MKL을 배치했고, 인텔® MKL-DNN 최적화 지향 TensorFlow1.10과 인텔® Python 배포 패키지도 설치했습니다. 가장 윗부분에는 금융 거래 사기 방지 애플리케이션을 배치했습니다.



그림1-1-6 "샌드위치" 다층 금융 거래 사기 방지 모델 소프트웨어 스택

■ 인텔® Python 배포 패키지 설치

다음의 각기 다른 명령을 실행해 python3/python2 환경에서 인텔® Python 배포 패키지 코어 패키지를 설치할 수 있습니다.

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

또는 인텔® Python 배포 패키지 풀 패키지 설치:

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

애플리케이션 활성화 환경:

Linux/MacOS :

1. `source activate idp`

Windows* :

1. `activate idp`

Conda 명령을 사용해 sympy 등과 같은 소프트웨어 패키지를 별도 설치합니다.

1. `conda install sympy`

이상 과정에 대한 세부 정보는 다음을 참조하십시오.

<https://software.intel.com/zh-cn/articles/using-intel-distribution-for-python-withanaconda>

* 인텔® Python 배포 패키지과 관련한 더 자세한 사항은 해당 매뉴얼 기술편 소개부분을 참조하십시오.

³ 샌드위치 아키텍처 금융 거래 사기 방지 모델 기술 세부 정보는 다음을 참조하십시오. Transaction Fraud Detection Using GRU-centered Sandwich-structured Model

■ 데이터 불균형 처리 방법

데이터 불균형은 금융 거래 사기 방지 애플리케이션이 자주 맞닥뜨리는 문제입니다. 이 환경에서 사기 표본과 정상 표본의 비율은 10 만~100만대 1에 달합니다. 데이터 불균형 상황을 기준으로 표본 추출과 트레이닝 진행할 경우에는 다음 원칙을 근거로 할 수 있습니다.

1. 정상 표본의 데이터 양이 충분히 많기 때문에 정상 표본 추출만으로도 트레이닝 조건을 만족 시킬 수 있습니다.
2. 트레이닝을 진행할 때 사기 표본 데이터의 가중치를 향상시킵니다.

```

1. class_weights= dict()
2. class_weights[0] = 1 #正常样本权重
3. class_weights[1] = 20 #欺诈样本权重
4. classifier.fit(X_train,
5.     y_train,
6.     batch_size=BS,
7.     epochs=runEpoch,
8.     class_weight=class_weights)

```

3. 화상과 음성 등 데이터가 쉽게 태그되는 경우와 달리, 사기 데이터는 쉽게 생성되거나 태그되지 않습니다. 그렇기 때문에 우리는 랜덤으로 순서를 뒤바꾸고 사기 표본을 여러 차례 트레이닝하지만 사실 불균형 문제를 해결하려면 정상 표본을 한번만 사용하면 됩니다.

■ 알고리즘의 정확성 제고 방법

■ 다른 방식으로 조합한 시퀀스와 정확성과의 관련성:

특징에서 특성을 뽑아낸 알고리즘과 특징 사이에서 특성을 뽑아낸 알고리즘을 결합할 때, 시퀀스가 다르면 정확성 또한 달라집니다. 우리는 테스트를 통해 특징에서 특성을 뽑아낸 방법 2 가지 사이에 특징 사이에서 특성을 뽑아낸 방법으로 획득한 정확성을 추가해야 가장 정확하다는 것을 알 수 있습니다.

■ 바이패스를 사용해 특징 재이용 강화:

Densenet* 알고리즘과 마찬가지로 "샌드위치" 역시 다른 방식 사이에 연결 관계를 구성했으며, 이러한 바이패스 사용 방식은 특징 재이용 강화를 통해 전체 트레이닝 효과를 향상시켜 줍니다.

소프트웨어 구성 제안

데이터 프로세스화 모델링과 다층 금융 거래 사기 방지 모델 구축 관련 검증 업무는 아래 인텔® 아키텍처 플랫폼 기반 소프트웨어 구성을 참조할 수 있습니다.

명칭	사양
운영 체제	Centos7.4
Linux 커널	3.10.0-957.12.1.el7.x86_64
워크로드	GRU
컴파일러	GCC5.4
베이스	인텔® MKL-DNN 최신 버전
아키텍처	인텔® 아키텍처 최적화 지향 TensorFlow 배포 버전*
Hadoop 배포 버전	Cloudera CDH-5.9.0 또는 그 이상의 버전
Spark 버전	Apache Spark-2.1.0. 또는 그 이상의 버전
기타 소프트웨어 구성	Anylitics Zoo 인텔® Python* 배포 패키지

중국은련 응용 사례

배경

금융 업무가 빠른 속도로 확장되면서 위험 지표 역시 끊임없이 증가하고 있습니다. 은행카드와 신용카드 등의 영역이 특히 두드러지며, 사기로 인한 손실률을 역시 사기 손실 금액이 증가함에 따라 매년 상승하고 있는 상황입니다. 이런 상황이다 보니 금융 거래 사기 방지의 자연스레 금융 업계의 리스크 관리가 발전해 나가야 할 중요한 방향이 되었습니다.

리스크는 형식적으로 전통 리스크와 새로운 유형의 리스크가 서로 뒤엉켜 있습니다. 리스크는 형식적으로 전통 리스크와 새로운 유형의 리스크가 서로 뒤엉켜 있습니다. 오랜 세월 끊이지 않고 발생하는 신용 사기, 도용 사기, 현금강, 보험 사기 등과 같은 전통적인 금융 사기 외에도 인터넷 발달로 등장한 개인 정보 누출, 피싱 사이트 등의 문제는 빈도가 더 찾고 더 치밀해진 금융 사기 범죄입니다.

이러한 문제에 대응하기 위해 여러 금융 기관에서 세심하고 다양한 방식으로 금융 거래 사기 방지 방안을 내놓았습니다. 정보화 기술, 특히 인공지능 기술이 끊임없이 발전하며 한계를 돌파함에 따라 날이 갈수록 금융 리스크 관리 시스템과 결합하는 인공지능 능력이 점차 많아지면서 더 효과적인 금융 거래 사기 방지 모델을 형성했습니다.

중국은련(이하 "은련")은 전문인 금융 거래 서비스를 제공하며, 카드 발급량과 거래량의 시장 점유율이 세계 최고인 금융 기관입니다. 중국은련은 인공지능 기술 도입에 전력을 다한 끝에 고효율 금융 거래 사기 방지 모델을 구축하는데 성공했습니다. 이번 사례는 중국은련과 인텔이 딥러닝 기반 금융 거래 사기 방지 기술 공동 연구한 사례입니다.

중국은련은 규칙과 머신 러닝 등 금융 거래 사기 방지 모델에서 얻은 경험을 결합하고, "샌드위치" 아키텍처의 다층 모델 및 인텔® 아키텍처의 다방면 최적화를 기초로 고효율의 금융 거래 사기 방지 방안을 구축했습니다. 현재 이 방안은 위조 카드/현금강 등의 사기 방지 시나리오에서

테스트를 진행했으며 만족스러운 결과를 얻었습니다.

솔루션

중국은련은 GBDT—>GRU—>RF "샌드위치" 아키텍처를 이용해 고효율 금융 거래 사기 방지 모델을 구축했습니다. 우선, 은련은 Analytics Zoo 및 Spark* pipeline을 기초로 데이터에 프로세스화 모델링을 진행했습니다. 통상적으로 AI 트레이닝 모델은 사용자의 모든 거래와 행동을 분석하고, 알고리즘을 통해 모든 카드 소유자의 소비 패턴을 학습해야 합니다. 이상 여부를 분석해야 하며, 이상 거래 행위를 발견하면 차단 조치가 작동되는데 이때 시스템은 다양한 거래 데이터가 필요해집니다. 또한, 트레이닝 모델은 사용자의 과거 거래를 학습해야 하기 때문에 인당 최소 수백 건에 달하는 비정상 거래 데이터가 학습용으로 모델에게 제공되어야 합니다.

이러한 상황 때문에 중국은련은 Hadoop* 기반 스토리지 플랫폼을 구축해 Analytics Zoo 등으로 트레이닝 데이터를 프로세스화 모델링하고 있습니다. 학습할 과거 거래 데이터가 부족한 문제점의 경우, 모델링 과정을 이용하면 플랫폼이 소량의 최초 필드에서 수백 개의 특정 인자를 파생해내 현재/이전 거래, 장기/단기 통계 및 신뢰할 수 있는 특징 변수 등 6가지

차원으로 귀납할 수 있습니다. 또한 이 프로세스를 통하면 모델이 학습하기 더 수월해집니다. 은련은 "GBDT—>GRU—>RF" 3 층 아키텍처 모델을 기반으로 수백 개의 노드로 구성된 트레이닝 클러스터에 금융 거래 사기 방지 모델을 구축했으며 그 결과 만족스러운 효과를 보았습니다.

다방면의 검사와 평가를 통해 새로운 다층 금융 거래 사기 방지 모델은 리콜률과 정확도에서 모두 기대치를 만족시켰습니다. 전통적인 머신 러닝, 또는 단일 RNN 방식과 비교했을 때, F1 값 (F1 Score, 정확도와 리콜률 가중치 평균값으로 모델의 성능을 표시하는데 사용됨)이 향상했으며, 업무 배치의 임계점을 돌파했습니다.

그림 1-2-1 중 왼쪽 그림과 같이 GBDT—>GRU—>RF "샌드위치" 아키텍처 금융 거래 사기 방지 모델은 기타 머신 러닝, 딥러닝 모델 또는 다층 모델과 비교했을 때, 가장 뛰어난 정확도- 리콜(Precision - Recall, PR) 곡선(가장 윗부분의 곡선은 GBDT—>GRU—>RF "샌드위치" 아키텍처 금융 거래 사기 방지 모델 테스트값)을 형성하고 있습니다. 오른쪽 그림에서는 데이터의 불균형(Imbalance ratio) 상황이 증가함에 따라, GBDT—>GRU—>RF "샌드위치" 아키텍처 금융 거래 사기 방지 모델의 F1 값이 가장 완만하게 하락하는 것을 알 수 있습니다.

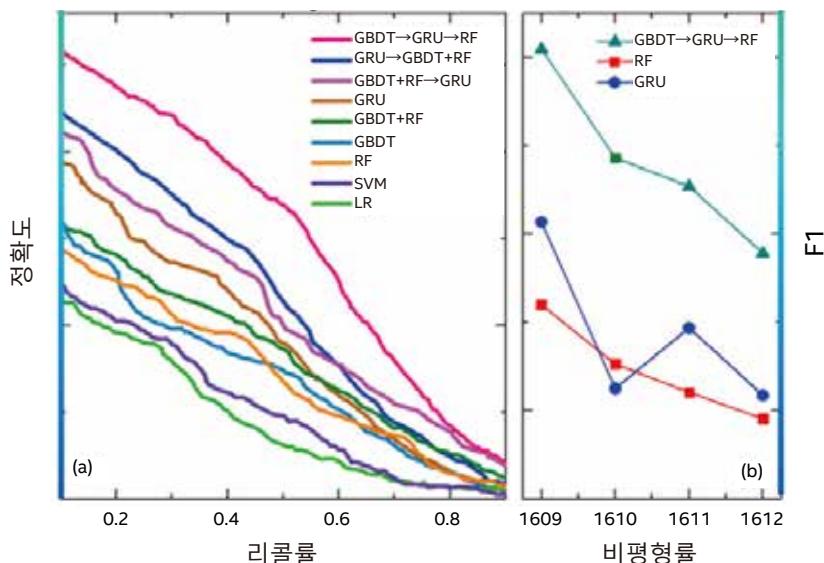


그림 1-2-1 GBDT—>GRU—>RF "샌드위치" 아키텍처 금융 거래 사기 방지 모델 평가 효과

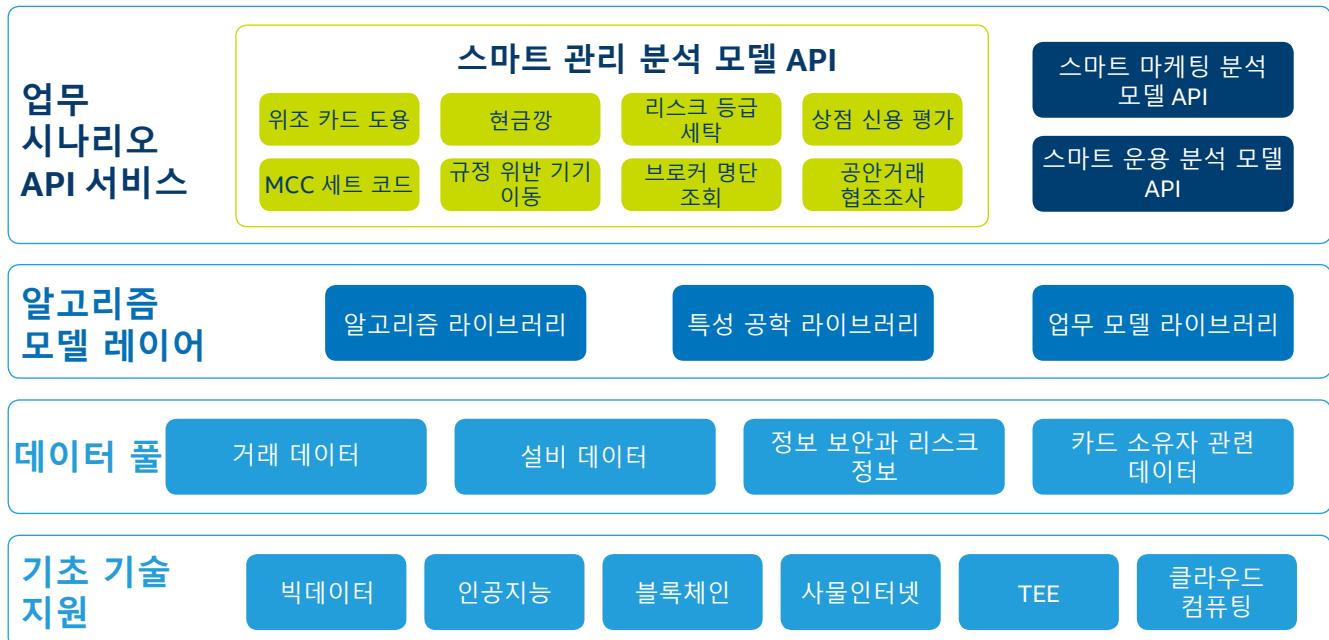


그림 1-2-2 중국은련 전자 결제 연구원 스마트 분석 서비스 플랫폼* 구조도

프로세스화 모델링과 다중 금융 거래 사기방지 모델 구축을 완료한 뒤 은련은 이를 통합하고 포장해 API 인터페이스 방식으로 end-to-end 의 인공지능 분석 솔루션에 제공했고, 그 덕분에 직원들에게 더 좋은 서비스를 제공하게 되었습니다. 그림1-2-2와 같이 사용자는 API 인터페이스 등의 방식을 통해 파라미터를 제공하며, 인공지능 모델 연산 분석을 거친 결과 지표를 획득할 수 있습니다. 샌드위치의 금융 거래 사기 방지 모델을 예로 들면, 샌드위치는 위조 카드와 현금깡 등 사기 방지 시나리오에서 기본 모델을 제공해 지원하기 때문에 업무 관련 직원은 복잡한

모델을 연구하느라 애쓸 필요가 없습니다. 데이터 규범에 따라 상위층의 API 를 호출하기만 하면 됩니다.

은련의 트레이닝 클러스터에는 모두 인텔® 제온® CPU 기반 플랫폼을 사용했습니다. 이 플랫폼은 커널, 고속 캐시 등에서 뛰어날 뿐만 아니라 여러 하드웨어 강화 기술 측면에서도 아키텍처의 성능을 향상시킬 수 있습니다. 기본적인 컴퓨팅 능력을 통한 지원 외에도 플랫폼은 은련 AI 금융 거래 사기 방지 모델 구축에 다음과 같은 능력을 지원합니다.

- 엔트로피, 트리 순회, 축소 등과 같이 고도의 불규칙한 계산을 지원합니다.

- GRU 비선형 활성화, 일괄 처리 규범화 등과 같은 상용 계산 방식을 지원합니다.

뿐만 아니라 인텔® 제온® CPU/인텔® 제온® 확장성 CPU 가 집성한 인텔® 고급 벡터 확장 512(인텔® AVX-512) 기술은 은련의 "샌드위치" 아키텍처 금융 거래 사기 방지 모델에 훌륭한 병렬 컴퓨팅 기능을 제공했습니다.

요약

머신러닝에 기반한 방식으로 시퀀스화된 거래 특징을 학습하기에는 역부족입니다. 또한 단일 딥러닝 모델로 단일 거래의 특징을 학습하기에는 한계가 있습니다. 그렇기 때문에 은련은 인텔과 연합해 다층 머신러닝+딥러닝 모델을 내놓았고, 이 기술 혁신은 금융 거래 사기 방지 모델의 성능을 대폭 향상시켰습니다.

이런 과정 속에서 인텔은 고성능 CPU 제품을 알고리즘 엔진으로 제공했을 뿐만 아니라, 다양하고 확장성이 뛰어나며 전방위적인 기술 지원도 아끼지 않았습니다. 샌드위치 금융거래 사기 방지의 모든 층마다 맞춤형 최적화 수단과 툴을 제공함으로써 전체 금융 거래사기 방지 모델이 효율을 높일 수 있도록 도왔습니다.

인텔® 제온® CPU/인텔® 제온® 확장성 CPU에 기반한 하드웨어 플랫폼은 중국은련의 금융 거래 사기 방지 모델을 성공적으로 구축했고, 애플리케이션이 제공한 강한 알고리즘과 인텔이 제공한 최적화된 조치로 앞으로는 사용자 역시 성능이 뛰어날 뿐만 아니라 AI 영역에서 더 많은 최적화 방식을 보유한 2세대 인텔® 제온® 확장성 CPU 등의 하드웨어 업데이트 제품을 사용해 더 우수한 성능을 솔루션을 구축할 수 있게 될 것입니다.

AI 기반 고효율 연체 대출 리스크 예측 솔루션

인텔과 금융 사업자가 협력하고 AI 모델을 이용해 진행한 연체 대출 리스크 예측

도전에 직면한 연체 대출 리스크

대출은 금융 기관에게 있어 가장 중요한 자산 업무 중 하나입니다. 상업은행의 대출 업무 규모가 점차 확장됨에 따라 증가한 부실 채권은 은행의 이윤을 잠식할 뿐만 아니라 여신한도를 점용하기 때문에 은행의 대출 업무에 영향을 미쳐 정작 필요한 우수 사업에 대출을 지원할 수 없게 됩니다.

더 심각한 문제는 부실 채권이 일정 한도액을 초과하면 업무 경영과 운영에 지대한 영향을 미쳐 리스크를 가져온다는 점입니다. 뿐만 아니라 부실 채권이 다량 발생하면 전체 사회의 도덕적 위험을 유발하고, 부실 채권 처리 강도가 과도하게 크면 기업이 연쇄 파산할 수 있기 때문에 재정 리스크와 사회 신용 위기 확률이 증가합니다.

중국보험감독관리위원회 (이하 "중국보감회") 의 데이터에 따르면, 2018년 4분기 말부터 중국 상업은행의 부실 채권액이 2.03 조 위안, 부실 채권률이 1.83%⁴에 달한다고 합니다. 상황이 이렇다 보니 은행이 리스크 관리 시스템을 구축하는데 있어 고효율 대출 전후 리스크 관리는 매우 중요한 콘텐츠가 되었습니다.

현재 상업은행은 연체 대출 리스크 시나리오로 주로 두 가지를 예측합니다. 하나는 대출 전에 진행하는 대출 전 리스크 평가로 예측한 결과의 시효성과 해석 가능성에 중점을 두고 있습니다. 또 다른 하나는 대출 지급 후 진행하는 대출 후 리스크 예측으로 예측한 결과의 정확도와 해석 가능성에 중점을 두고 있습니다.

대출 전 리스크 예측의 경우, 상업은행이 대출 지급 전 주로 대상 기업이 속해 있는 업계의 발전적 특징과 기업의 실제 경영 상황, 자산 부채, 신용 상황 등에 따라 다방면으로 조사 연구를 진행하며 이를 근거로 대출 지급 리스크 등급을 평가합니다. 이러한 방식은 통상적으로 효율이 낮기 때문에 다량의 인력과 시간이 소모되고 주관적 판단이 개입된다는 단점이 있습니다.

대출 후 리스크 예측은 상업은행이 일반적으로 인력을 동원해 정기적 또는 비정기적으로 대출 기업이 속한 업계와 경영 특징에 따라 현장을 직접 또는 원격 검사합니다. 대부분을 통해 대출 기업의 재무 정보와 경영 상황을 분석하고 대출 자금 흐름을 모니터링해 발생할 수 있는 부도 리스크와 신용 리스크 요소를 파악해 연체 대출을 방지합니다.

⁴ 데이터 소스 출처는 중국보감회 홈페이지입니다.
<http://www.cbrc.gov.cn/chinese/newShouDoc/CDF5FDDEDAE14EFEB351CD93140E6554.html>

이처럼 인력 동원 방식은 비용 문제로 매월 또는 분기별 1회밖에 진행할 수 없다 보니 이슈가 발생했을 때 상부에 보고하고 리스크 관리 부서에서 결정이 내려와야 행동을 취할 수 있기 때문에 다음처럼 몇 가지 문제를 피할 수 없습니다.

- 인력이 많이 투입되고 예상 소요 시간이 길니다. 은행에서 근무하는 직원들은 매월 만기 예정 금액과 향후 3개월부터 6개월까지 만기 예정인 연체 대출 리스크를 예측하고 상부 보고해 처리를 기다려야 하는데 이 전체 과정이 한 달 가까이 걸립니다. 대출 관리 리스크를 컨트롤하기 위해 귀중한 시간을 할애해야 합니다.
- 인력을 통한 예측 작업은 품질 측면에서 사람마다 차이가 크다 보니, 어떤 사람은 경험이 많아 정확히 예측하는 것과 달리 어떤 사람은 경험 부족으로 리스크를 제거할 역량이 부족합니다.
- 예측에 영향을 미치는 요소가 여러 가지입니다. 인력을 통한 예측 과정 중 시장 환경의 다변성과 경제 활동의 주기성, 은행과 기업 정보의 불일치성 등의 요소에 간섭을 받게 되고, 더 나아가 리스크 예측 작업의 시효성과 정확성에 영향을 미치게 됩니다.

인력을 통한 예측은 정확성을 보장받을 수 없다는 점 외에도 지식 체계가 완전하지 않기 때문에 경험 누적만으로 예측 작업의 효율과 정확성을 제고하기 어렵다 보니 선순환 구조를 형성할 수 없게 됩니다.

대출 업무가 점차 확대됨에 따라 상업은행의 전통적 인력 기반 리스크 예측 방식이 직면하는 도전 역시 나날이 커지고 있습니다.

이러한 문제점에 매월 중국보감회가 연체 상황을 관찰하는 것까지 더해져 은행은 지대한 비용과 관리 스트레스를 받고 있는 게 현실입니다. 상황이 이렇다 보니 은행은 자연스레 자신의 풍부한 데이터 자원을 이용, AI 방식을 통해 더 효과적인 대출 연체 리스크 예측 시스템을 원하게 되었습니다. 완전한 연체 대출 리스크 예측 AI 아키텍처를 구축해 높은 정확성과 낮은 지연 시간, 해석 가능한 대출 연체 예측 방안을 마련하려면 우선 업무 데이터와 환경 데이터 분석 및 예측이 이루어져야 합니다.

전자인 업무 데이터는 금융 기관이 기업 사용자의 금융 자산 상황과 미래 수입, 자금 용도 등 데이터를 기록한 것으로, 현재 업계는 통상적으로 머신러닝이나 딥러닝의 방식으로 예측 모델을 구축합니다. 후자는 환경 데이터로 자연어 처리 (Natural Language Processing, NLP) 방식으로 연구와 예측을 진행할 수 있습니다.

인텔과 금융 사용자는 협력을 통해 LSTM과 전통적인 머신러닝을 기반으로 한 혼합 모델을 구축함으로써 정확도와 해석 가능성에 대한 사용자의 니즈를 만족시켰을 뿐만 아니라, 환경 데이터의 NLP 모델 구축을 위한 기반도 탐색했습니다.

연체 대출 리스크 예측 모델의 아키텍처 설계

■ 머신러닝 방식 기반

분류 회귀 트리 기반 머신러닝 방식은 연체 대출 리스크 예측 모델에서 자주쓰이는 기술로써 예측 결과의 해석 가능성이 비교적 뛰어난 것이 특징입니다. 그 중 XGBoost^{*}는 중요한 머신러닝 모델이자 boosting의 양상을 학습으로 다양한 분류 회귀 트리 (Classification And Regression Tree, CART)를 집성해 만든 강분류기입니다.

CART 회귀 트리는 이진 트리에 특징에 따라 끊임없이 가지치기를 하는 것입니다. 예를 들어, 현재 트리의 노드 J가 a개의 특징 값에 따라 가지치기를 한다면 특징 값 b의 표본은 좌 종속 트리로 구분되고 y보다 큰 표본은 우 종속 트리로 구분됩니다.

$$J_1(a, b) = \{X | X^{(a)} \leq b\} \quad (\text{좌子樹})$$

$$J_2(a, b) = \{X | X^{(a)} > b\} \quad (\text{우子樹})$$

CART 회귀 트리는 실질적으로 이러한 특징 차원에서 표본 공간을 구분한 것으로 전형적인 CART 회귀 트리가 생성한 목표함수는 다음과 같습니다.

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

XGBoost로 다시 돌아와서, 핵심 사상을 이야기한다면 끊임없이 진행되는 특징

분열을 통해 생성되는 새로운 포크 트리라 할 수 있습니다. 트리가 한 개 추가될 때마다 새로 운 학습 하나를 학습해 이전에 예측한 나머지에 맞추기 때문에, XGBoost의 목표함수는 다음과 같이 정의 내릴 수 있습니다.

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

k 개의 표본 중 n번째 모델 예측의 결과는 다음과 같습니다.

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

GBT 등 머신러닝 방식과 비교하면 XGBoost의 장점은 다음과 같습니다.

- XGBoost는 병렬 컴퓨팅을 지원하기 때문에 CPU의 다중 스레드 기능을 충분히 이용할 수 있습니다. 특히 인텔® 아키텍처 플랫폼에서 작업할 경우에는 인텔® AVX-512 등 최신 명령어 조합의 강렬한 병렬 컴퓨팅 기능을 효과적으로 이용할 수 있습니다.
- XGBoost는 비용 함수에 정규화 항목을 도입했기 때문에 모델이 복잡해지는 것을 제어하고 과도한 맞춤을 방지할 수 있습니다.
- XGBoost는 칼럼 서브샘플링(column subsampling) 방식을 지원하기 때문에 과도한 맞춤을 방지할 수 있을 뿐만 아니라 컴퓨팅이 복잡해지는 것을 줄일 수 있습니다.
- GBT는 최적화를 진행할 때 일 차 도함수 정보만 사용하고, XGBoost는 비용 함수에 2차 테일러리를 사용하기 때문에 1차와 2차 도함수를 동시에 사용해 더 뛰어난 예측 효과를 볼 수 있습니다.

그러다 보니 XGBoost 머신러닝 모델은 이미 연체 대출 리스크 예측 솔루션으로 광범위하게 적용되고 있습니다. XGBoost를 사용한 대출 리스크 예측 절차는 그림 2-1-1과 같으며, 데이터 도입, 데이터 정리와 준비, 모델 구축, 모델 평가 및 모델 효과 대비 등 몇 가지 주요 절차로 나뉩니다.



그림 2-1-1 XGboost 를 사용한 연체 대출 리스크 예측

■ RNN/LSTM 기반 딥러닝 방식

딥러닝 방식 역시 연체 대출 리스크 예측 분야에서 나날이 광범위하게 응용되는 방안이며, 그 중 RNN은 전형적인 딥러닝 모델입니다. 전형적인 RNN 구조 중, 입력마다 현재 모델의 상태를 결합하면 출력력을 얻을 수 있으며, RNN의 주구조 A의 입력은 입력층의 X 외에 순환을 거쳐 현재 시점의 상태가 제공됩니다. 이와 동시에 A의 상태 역시 현재 단계에서 다음 단계로 점차 전달됩니다.

LSTM은 RNN이 발전한 모델로 특별한 "게이트" 아키텍처 설계를 통해 전형적인 RNN 아키텍처에 존재하는 장기 의존 문제를 방지함으로써 기억 시간을 대폭 향상시킵니다. LSTM 기반 딥러닝 방식은 시퀀스 기반 분석 작업 운용에 매우 적합합니다. 즉, 은행은 기업 경영 상황과 계정 수입 등 과거 일정 시간 동안 대출 지급 후 일련의 특징을 이용해 미래의 일정 기간에 대출이 직면할 수 있는 연체 리스크를 예측할 수 있습니다.

하지만 딥러닝 방식을 자체적으로 과정에 해석 가능성 이 부족하다는 결함을 갖고 있기 때문에 은행 등 금융 기관은 추론해 얻은 결과를 해석해야 할 때가 종종 있습니다. 즉, 모델을 어떤 정보와 조건에 따라 특정 예측 결과를 얻을지

파악해야 합니다. 이러한 해석은 금융 고객 업무 프로세스 개선, 고객 경험 개선 등 측면에서 가이드를 제공할 수 있습니다. 딥러닝 방식은 사용자에게 종종 블랙박스 상태로 나타나며, 이는 해석 가능한 딥러닝 방식이 향후 딥러닝 방식이 최적화를 실행할 수 있는 중요한 방향이 되었습니다.

■ 머신러닝과 딥러닝 집적 방법

해석 가능성과 정확성을 향상시키기 위해서는 다른 방법을 채택해야 합니다. 모델 융합은 매우 효과적인 기술로 대부분의 머신러닝 업무 중 회귀 또는 분류의 정확성을 향상시킬 수 있습니다. 다른 모델의 결과 문서를 바로 융합할 수도 있고, 어느 모델 하나의 예측 결과를 또 다른 모델의 특징으로 삼아 트레이닝을 진행한 뒤 새로운 예측 결과를 얻을 수도 있습니다. 모델의 유형이 다르면 학습 트레이닝 원리도 다르고, 학습한 지식 역시 달라지며, 이를 융합하면 트레이닝 효과를 향상시킬 수 있습니다. 예를 들어, 트리 모델 XGBoost 와 LSTM 딥러닝 모델 융합 방식은 예측 능력을 한 단계 강화시킴과 동시에 모델이 해석 가능성을 갖출 수 있게 해줍니다. 모델 융합의 전체 구조는 다음 그림과 같습니다.

연체 대출 리스크 예측 모델의 알고리즘 실현

연체 대출 리스크 예측 모델 트레이닝 데이터

연체 대출 리스크 예측 모델 트레이닝 데이터는 일반적으로 몇 년 사이에 고객이 은행에서 진행한 대출 거래 데이터와 당월 고객의 경영 상황을 양자화한 데이터를 포함합니다. 그 밖에 인력 평가 업무 논리 역시 고급 특징으로 데이터에 추가되어 집중적으로 트레이닝됩니다.

■ 연체 대출 리스크 예측 모델 소프트웨어 플랫폼

DMLC (Distributed Machine Learning Community)는 현재 인텔® Python 배포 패키지와 인텔® 아키텍처 최적화 지향 TensorFlow 딥러닝 아키텍처의 XGBoost 오픈 소스 패키지를 발표했습니다. XGBoost 오픈소스 패키지는 wrapper를 1개 제공하며, 모델이 Scikit-Learn* 아키텍처 중 다른 분류기 또는 리그레서와 함께 사용하는 것을 허용합니다.

XGBoost는 인텔® Python 배포 패키지를 통해 트레이닝과 추론 과정을 가속화할 수 있습니다. 인텔® Python 배포 패키지에는 인텔 이 디 이 터 분석 과 머신러닝을 타깃으로 한 가속 라이브러리 (Intel® Data Analytics Acceleration Library, Intel® DAAL)가 내장되어 있습니다. 이 가속 라이브러리는 머신러닝 과정을 가속화할 수 있으며, 인텔® 아키텍처의 하드웨어 자원을 충분히 이용할 수 있습니다.

머신러닝과 딥러닝의 융합

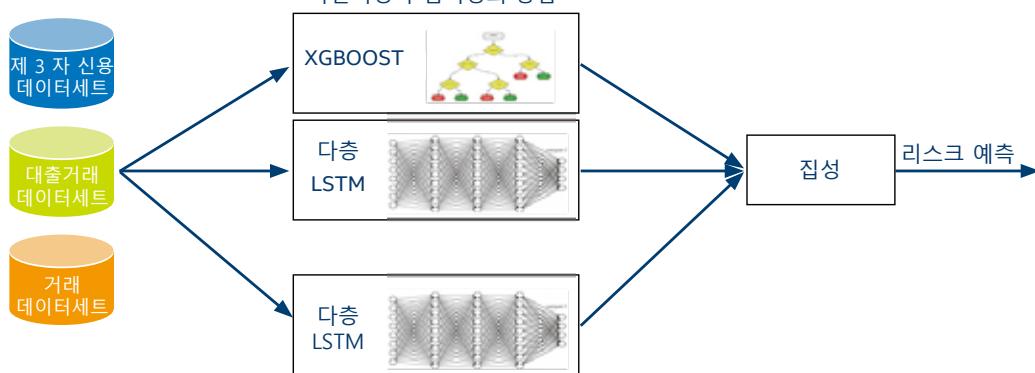


그림 2-1-2 모델 융합 전체 아키텍처

■ 인텔® 아키텍처 최적화 지향 TensorFlow

TensorFlow 업계를 선도하는 딥러닝 아키텍처로서 이미 여러 업계의 AI 애플리케이션에 광범위하게 응용되었습니다. TensorFlow 아키텍처가 인텔® 아키텍처 기반 플랫폼에서 최고 성능을 발휘하도록 하려면, 즉, 인텔® 아키텍처 최적화 지향 TensorFlow 를 최적화 방향으로 가도록 하려면 주로 세 가지 측면에서 진행해야 합니다.

- 1) 인텔® MKL-DNN 집적
- 2) 산출 그래프 최적화
- 3) Kernel 최적화.

위의 세 가지 측면을 최적화하면 가장 자주 사용하는 연산을 최적화된 MKL-DNN 요소에서 진행할 수 있을 뿐만 아니라 연산자 융합 방식을 통해 산출 그래프를 최적화할 수 있습니다. 또한, 여러 스레드 라이브러리를 최적화해 서로 CPU 자원을 쟁탈하지 않고 공존할 수 있게 해줍니다. 이렇게 소프트웨어 측면을 최적화하면 뉴럴 네트워크 모델을 변경하지 않은 상태에서 트레이닝하고 추론한 전체 성능을 눈에 띄게 향상시킬 수 있습니다. 구체적인 방법은 아래 문장 중 "XGBoost 모델 및 트레이닝" 소개를 참조하십시오.

* 인텔® MKL-DNN 과 관련한 더 자세한 사항은 해당 매뉴얼 기술편 소개 부분을 참조하십시오.

■ 데이터 전처리와 피쳐 엔지니어링

데이터의 전처리와 피쳐 엔지니어링은 구체적인 임무와 데이터에 따라 다르게 처리합니다. 예를 들어, One-hot 코딩은 카테고리형의 변수를 수치 변수로 전환할 수 있으며, 데이터 정규화는 주로 모델의 트레이닝과 컨버전스 속도를 가속화합니다.

```

1. test_df = pd.concat([test_df,pd.get_dummies(test_df[Feature 1],prefix='F1')],axis=1)
2. test_df = pd.concat([test_df,pd.get_dummies(test_df[Feature 2],prefix='F2')],axis=1)
3. ....
4. test_df = pd.concat([test_df,pd.get_dummies(test_df[Feature N],prefix='FN')],axis=1)
5. test_df.drop(['Feature 1'],axis=1,inplace=True)
6. test_df.drop(['Feature 2'],axis=1,inplace=True)
7. ....
8. test_df.drop(['Feature N'],axis=1,inplace=True)

```

카테고리형 변수를 수치형 전환하면 기존의 카테고리형 변수가 삭제되고, 전체 데이터를 정규화 처리합니다.

```

1. std_scale = preprocessing.StandardScaler()
2. for column in need_to_scale:
3.     test_df[column] = std_scale.fit_transform(test_df[column].reshape(-1, 1))
4.     print("0 mean = %f, var = %f" %format(column, std_scale.mean_, std_scale.var_))

```

■ XGBoost 모델 및 트레이닝

■ XGBoost 모델 실현

XGBoost 모델은 XGBoost 오픈소스 패키지를 채택해 배치하거나 최초 XGBoost 패키지로 실현합니다. 이런 방식은 SKLearn* 스타일의 프로그래밍 방식을 채택하면 더 간단하고 편리해집니다.

```

1. # fit model on training data
2. eval_set = [(X_train,y_train)]
3. # hyper parameters
4. h_param = {'learning_rate': 0.1,
5.             'n_estimators': 2000,
6.             'max_depth': 2,
7.             'min_child_weight': 10,
8.             'gamma': 0.0,
9.             'subsample': 0.8,
10.            'colsample_bytree': 0.8,
11.            'objective': 'binary:logistic',
12.            'eval_metric': 'auc',
13.            'scale_pos_weight': imbalance_weight,
14.            'reg_lambda': 0.7,
15.            'reg_alpha': 1.25,
16.            'cv': 10}
17. model1 = XGBClassifier(learning_rate=h_param['learning_rate'],
18.                         n_estimators=h_param['n_estimators'],
19.                         max_depth=h_param['max_depth'],
20.                         min_child_weight=h_param['min_child_weight'],
21.                         gamma=h_param['gamma'],
22.                         subsample=h_param['subsample'],
23.                         colsample_bytree=h_param['colsample_bytree'],
24.                         objective=h_param['objective'],
25.                         eval_metric=h_param['eval_metric'],
26.                         scale_pos_weight=h_param['scale_pos_weight'],
27.                         reg_lambda=h_param['reg_lambda'],
28.                         reg_alpha=h_param['reg_alpha'],
29.                         seed=randint(1,65535),
30.                         cv=h_param['cv'])
31. xgb_param = model1.get_xgb_params()
32. print("-----Parameters-----")
33. print(xgb_param)
34. # Fit the algorithm on the data
35. model1.fit(X_train,y_train,eval_metric='auc')

```

■ XGBoost 파라미터 튜닝

XGBoost 는 슈퍼 파라미터가 매우 다양하며 모든 파라미터를 다음 세 가지로 분류할 수 있습니다.

- 1) 제너럴 파라미터: 매크로 파라미터 제어, 이 부분의 파라미터는 기본적으로 조정이 필요 없습니다.
- 2) Booster 파라미터: 모든 단계의 Booster(tree/regression) 관련 파라미터를 제어하려면 상세히 조정해야 하고, 이는 최종 성능에 영향을 미칩니다.
- 3) 학습목표 파라미터: 트레이닝 목표를 제어한 결과물로 일반적으로 임무에 따라 확정되고 보통은 조정이 필요 없습니다.

조정해야 할 파라미터는 주로 Booster 와 관련이 있습니다. 아래 표를 참조하십시오.

파라미터	설명
max_depth	트리의 최대 깊이. 트리가 깊을수록 모델이 더 복잡해지고 과도하게 맞춤이 이루어지기 쉽습니다.
learning_rate	학습률 또는 축소 인자
n_estimators	에스티메이터 수
gamma	노드 분열에 필요한 최소 순실 함수 하락 값
min_child_weight	잎 노드가 필요한 최소 표본 가중치의 합
subsample	트리별 사용된 표본의 비율
colsample_bytree	트리별 사용된 특징의 비율
colsample_bylevel	트리가 층별 분열별 사용된 특징 비율
reg_alpha	L1/L0 정규 페널티 계수
reg_lambda	L2 정규 페널티 계수

인터넷 검색의 방식을 통해 위의 표에서 나열한 Booster 파라미터를 단계적으로 조정할 수 있습니다. 인터넷 검색은 교차 검증 방식으로 가장 우수한 파라미터를 선택해야 하기 때문에 여러 파라미터를 동시에 최적화할 때 매우 시간이 오래 걸리고, 그렇기 때문에 하나씩 또는 관련 파라미터를 그룹별로 최적화해야 합니다. 조정하면서 XGBoost 자체 CV 함수를 통해 트리의 수를 조정할 수 있습니다. 기타 파라미터는 XGBRegressor 또는 XGBClassifier (XGBoost의 Sklearn 패키지, 회귀와 분류 문제 조절 전략 일치)와 GridSearchCV를 이용해 조정합니다.

다음은 최적 트리 조정에 사용된 함수의 수를 정의했습니다.

```

1. def modelFit(alg, X_train, y_train, folds, useTrain=True, early_stopping_rounds=50):
2.     if useTrain:
3.         dtrain = xgb.DMatrix(data=X_train, label=y_train)
4.         params = alg.get_xgb_params()
5.         cvResult = xgb.cv(params=params, dtrain=dtrain, num_boost_round=params["n_estimators"], folds=folds,
6.                            early_stopping_rounds=early_stopping_rounds, metrics="rmse")
7.         alg.set_params(n_estimators=cvResult.shape[0])
8.         print("n_estimators is: {}".format([cvResult.shape[0]]))
9.         alg.fit(X_train, y_train)
10.        prediction = alg.predict(X_train)
11.        error = mean_squared_error(y_train, prediction)
12.        print("mean squared error is {}".format(error))
13.        feaImp = pd.Series(alg.get_booster().get_fscore()).sort_values(ascending=True)
14.        feaImp.plot(kind="barh", title="Feature Importance")

```

2 단계: 기본적인 XGBRegressor (조정해야 할 파라미터를 모두 나열할 수 있으며, 조정하기 편리함) 정의, 위에서 정의한 함수를 호출해 얻은 트리의 수:

```

1. xgbRegressor = XGBRegressor(
2.     n_estimators=2000,
3.     learning_rate=0.1,
4.     max_depth=6,
5.     min_child_weight=1,
6.     gamma=0,
7.     subsample=1,
8.     colsample_bytree=1,
9.     colsample_bylevel=1,
10.    reg_alpha=0,

```

```

11.    reg_lambda=1,
12.    objective='reg:linear',
13.    seed=123,
14.    n_jobs=-1)
15. kfold = KFold(n_splits=5, random_state=0, shuffle=True)
16. modelFit(xgbRegressor, X_train, y_train, kfold)

```

얻은 결과:

1. n_estimators is: [523]

3 단계: 여러 파라미터의 의미에 따라 나누어 조정합니다. 시작할 때에는 파라미터 보폭을 확대해 거칠게 조정하고, 결과가 나오면 보폭을 줄여 미세하게 조정합니다.

```

1. param_grid1 = {"max_depth":range(3,10,2), "min_child_weight":range(1,10,2)}
2. xgbRegressor1 = XGBRegressor(
3.     n_estimators=523,
4.     learning_rate=0.1,
5.     max_depth=6,
6.     min_child_weight=1,
7.     gamma=0,
8.     subsample=1,
9.     colsample_bytree=1,
10.    colsample_bylevel=1,
11.    reg_alpha=0,
12.    reg_lambda=1,
13.    objective='reg:linear',
14.    seed=123,
15.    n_jobs=-1)
16. gridcv1 = GridSearchCV(xgbRegressor1, param_grid=param_grid1, scoring="neg_mean_squared_error", cv=5, n_jobs=-1)
17. gridcv1.fit(X_train, y_train)
18. gridcv1.best_params_, gridcv1.best_score_

```

얻은 결과:

1. {[max_depth:9,min_child_weight:5]}

```

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

```

얻은 결과:

1. {[max_depth:9,min_child_weight:6]}

개별 조정은 최적화된 파라미터가 필요하며, 마지막에 얻은 최적의 파라미터는: gamma

```

1. param_grid4 = {"gamma": [x/10 for x in range(0,6)]}
2. xgbRegressor4 = XGBRegressor(
3.     n_estimators=473,
4.     learning_rate=0.1,
5.     max_depth=9,
6.     min_child_weight=6,
7.     gamma=0,
8.     subsample=1,
9.     colsample_bytree=1,
10.    colsample_bylevel=1,
11.    reg_alpha=0,
12.    reg_lambda=1,
13.    objective='reglinear',
14.    seed=123,
15.    n_jobs=-1)
16. gridcv4 = GridSearchCV(xgbRegressor4, param_grid=param_grid4, scoring="neg_mean_squared_error", cv=5, n_jobs=-1)
17. gridcv4.fit(X_train, y_train)
18. gridcv4.best_params_, gridcv4.best_score_

```

얻은 결과:

```
1. Out[20]: {'gamma': 0.1}
```

subsample	트리별 사용된 표본의 비율
colsample_bytree	트리별 사용된 특징의 비율
colsample_bylevel	트리가 층별 분열별 사용된 특징 비율

```

1. param_grid5 = {"subsample": [x/100 for x in range(70,90,5)], "colsample_bytree": [x/100 for x in range(70,90,5)]}
2.     "colsample_bylevel": [x/100 for x in range(70,90,5)]}
3. xgbRegressor5 = XGBRegressor(
4.     n_estimators=473,
5.     learning_rate=0.1,
6.     max_depth=9,
7.     min_child_weight=6,
8.     gamma=0,
9.     subsample=1,
10.    colsample_bytree=1,
11.    colsample_bylevel=1,
12.    reg_alpha=0,
13.    reg_lambda=1,
14.    objective='reglinear',
15.    seed=123,
16.    n_jobs=-1)
17. gridcv5 = GridSearchCV(xgbRegressor5, param_grid=param_grid5, scoring="neg_mean_squared_error", cv=5, n_jobs=-1)
18. gridcv5.fit(X_train, y_train)
19. gridcv5.best_params_, gridcv5.best_score_

```

얻은 결과:

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

다음 두 개의 파라미터는 함께 조정할 수 있습니다.

reg_alpha	L1/L0 정규 페널티 계수
reg_lambda	L2 정규 페널티 계수

```

1. 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]}
2. xgbRegressor7 = XGBRegressor(

```

```

3.     n_estimators=429,
4.     learning_rate=0.1,
5.     max_depth=9,
6.     min_child_weight=6,
7.     gamma=0,
8.     subsample=0.7,
9.     colsample_bytree=0.75,
10.    colsample_bylevel=0.95,
11.    reg_alpha=0,
12.    reg_lambda=1,
13.    objective='reglinear',
14.    seed=123,
15.    n_jobs=-1)
16. gridcv7 = GridSearchCV(xgbRegressor7, param_grid=param_grid7, scoring="neg_mean_squared_error", cv=5, n_jobs=-1)
17. gridcv7.fit(X_train, y_train)
18. gridcv7.best_params_, gridcv7.best_score_

```

얻은 결과:

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

마지막으로 최적화된 파라미터로 모델을 재구성합니다.

```

1. xgbRegressor9 = XGBRegressor(
2.     n_estimators=2000,
3.     learning_rate=0.05,
4.     max_depth=9,
5.     min_child_weight=6,
6.     gamma=0,
7.     subsample=0.7,
8.     colsample_bytree=0.75,
9.     colsample_bylevel=0.95,
10.    reg_alpha=1,
11.    reg_lambda=0.1,
12.    objective='reglinear',
13.    seed=123,
14.    n_jobs=-1)
15. modelFit(xgbRegressor9, X_train, y_train, kfold)

```

■ 모델 트레이닝과 추론

모델의 슈퍼 파라미터 전체를 확정하면, 지정 XGBooster 의 파라미터를 통해 Booster 를 생성해 모델을 트레이닝할 수 있고, 임무에 따른 평가 기준에 따라 모델의 트레이닝 효과를 평가할 수 있습니다.

```

1. model = XGBClassifier()
2. eval_set = [(X_test, y_test)])
3. model.fit(X_train, y_train)
4. # make predictions for test data
5. y_predictions = model.predict(X_test)
6. # evaluate predictions
7. accuracy = metrics.accuracy_score(y_test, y_predictions)
8. print("Accuracy: %.2f%%" % (accuracy * 100.0))
9. acc = metrics.accuracy_score(y_test, y_predictions)
10. recall = metrics.recall_score(y_test, y_predictions)
11. f1 = metrics.f1_score(y_test, y_predictions)
12. precision = metrics.precision_score(y_test, y_predictions)
13. print("ACC: %.4f" % (acc * 100.0))
14. print("Precision: %.4f" % (precision * 100.0))
15. print("Recall: %.4f" % (recall * 100.0))
16. print("F1-score: %.4f" % (f1 * 100.0))
17. metrics.confusion_matrix(y_test, y_predictions)

```

■ 인텔® 아키텍처 최적화 지향 TensorFlow 를 사용해 LSTM 실현

실현 방법

인텔® 아키텍처 최적화 지향 TensorFlow 는 인텔® MKL-DNN 을 집성했기 때문에 인텔® 아키텍처의 하드웨어에 하드웨어 자원을 충분히 이용할 수 있습니다. 벡터화, 병렬화, 인텔® 딥러닝을 이용한 가속 기술(VNNI 명령어 조합) 등 여러 최적화 수단을 통해 LSTM 등 네트워크 모델을 가속화할 수 있습니다.

■ 모델 아키텍처의 정의

2 층 LSTM 네트워크를 채택해 네트워크 아키텍처를 구축할 수 있습니다. 그 중 한 층의 LSTM 네트워크는 기본적인 LSTM 층에 Dropout 1 층을 추가해 구성한 것이며, LSTM 출력한 뒤 완전 접속망 3 층을 연결했습니다.

```

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

```

손실함수와 옵티마이저를 정의, 손실함수 최소화:

```

1. logits = LSTM(X, weights, biases)
2. weighted_logits = tf.multiply(logits, class_weight)
3. prediction = tf.nn.softmax(weighted_logits)
4. # Define loss and optimizer
5. loss_op = tf.reduce_mean(tf.nn.softmax_cross_entropy_with_logits(
6.     logits=weighted_logits, labels=Y))
7. optimizer = tf.train.GradientDescentOptimizer(learning_rate=learning_rate)
8. train_op = optimizer.minimize(loss_op)

```

■ 모델 트레이닝

트레이닝 서버의 구성에 따라 트레이닝을 설정한 시스템 구성 파라미터:

```

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

```

모델 트레이닝과 검증 (Tensorboard 를 통해 관찰해야 할 여러 파라미터를 모니터링할 수 있음), 동시에 트레이닝한 모델을 저장:

```

1. # Start training
2. with tf.Session(config=conf) as sess:
3.     saver = tf.train.Saver()
4.     # Create a summary to monitor cost tensor
5.     tf.summary.scalar("loss_op", loss_op)
6.     tf.summary.scalar("accuracy", accuracy)
7.     merged_summary_op = tf.summary.merge_all()
8.     summary_writer = tf.summary.FileWriter("./tf_logs_train", sess.graph)
9.     test_writer = tf.summary.FileWriter("./tf_logs_test", sess.graph)
10.    # Run the initializer
11.    sess.run(init)
12.    i = 1
13.    for epoch in range(training_epochs):
14.        for b in range(total_batches):
15.            offset = (b * batch_size) % (lab_tr.shape[0] - batch_size)
16.            batch_x = X_tr[offset:(offset + batch_size), :]
17.            batch_y = lab_tr[offset:(offset + batch_size), :]
18.            _, c, summary = sess.run([train_op, loss_op, merged_summary_op], feed_dict={X: batch_x, Y: batch_y, keep_prob_: 0.9})
19.            summary_writer.add_summary(summary, i)
20.            i+=1
21.    print("Optimization Finished!")
22.    saver.save(sess, "checkpoints/loan-lstm.ckpt")
23.    #用验证集验证模型效果:
24.    validation_batches = (X_vld.shape[0])//batch_size
25.    j = 0
26.    precision_score_arr = []
27.    recall_score_arr = []
28.    f1_score_arr = []
29.    # Calculate accuracy for validation set
30.    for b in range(validation_batches):
31.        offset = (b * batch_size) % (lab_vld.shape[0] - batch_size)
32.        batch_x = X_vld[offset:(offset + batch_size), :]
33.        batch_y = lab_vld[offset:(offset + batch_size), :]
34.        # Calculate batch loss and accuracy
35.        pred, pred_2, Y_1, loss, acc, summary_val = sess.run([prediction, pred_val, Y_1, loss_op, accuracy, merged_summary_op], feed_dict={X: batch_x, Y: batch_y, keep_prob_: 1.0})
36.        test_writer.add_summary(summary_val, i)
37.        j+=1
38.        if b % display_step == 0 or b == 1:
39.            print("Step "+ str(b) + ", Minibatch Loss= " + \
40.                  "{:.4f}".format(loss) + ", Validation Accuracy= " + \
41.                  "{:.3f}".format(acc))
42.            precision_val = precision_score(Y_1, pred_2)
43.            recall_val = recall_score(Y_1, pred_2)
44.            f1_val = f1_score(Y_1, pred_2)
45.            precision_score_arr.append(precision_val)
46.            recall_score_arr.append(recall_val)
47.            f1_score_arr.append(f1_val)
48.            print("Precision: {:.4f}.format(precision_val))")
49.            print("Recall: {:.4f}.format(recall_val))")
50.            print("F1 score: {:.4f}.format(f1_val))")
51.            print("confusion_matrix")
52.            print(confusion_matrix(Y_1, pred_2))
53.            fpr, tpr, thresholds = roc_curve(Y_1, pred_2)
54.            print("Validation precision mean = {:.4f}.format(sum(precision_score_arr)/len(precision_score_arr)))")
55.            print("Validation recall mean = {:.4f}.format(sum(recall_score_arr)/len(recall_score_arr)))")
56.            print("Validation f1 mean = {:.4f}.format(sum(f1_score_arr)/len(f1_score_arr)))")

```

■ 모델 추론

모델의 추론 단계는 테스트 집합 데이터를 통해 모델의 최종 추론 효과를 얻으며, 우선 기재 전 저장해야 하는 모델은:

```

1. with tf.Session(config=config) as sess:
2.     precision_score_arr = []
3.     recall_score_arr = []
4.     f1_score_arr = []
5.     # Restore
6.     saver.restore(sess, tf.train.latest_checkpoint('checkpoints'))
7.     batch_size = 100
8.     test_batches = (test_x.shape[0]//batch_size)
9.     i = 0
10.    print("test_batches = {}" .format(test_batches))
11.    # Calculate accuracy for validation set
12.    for b in range(test_batches):
13.        offset = (b * batch_size) % (test_y.shape[0] - batch_size)
14.        batch_x = test_x[offset:(offset + batch_size), :]
15.        batch_y = test_y[offset:(offset + batch_size), :]
16.        # Calculate batch loss and accuracy
17.        pred, pred_2, Y_1, acc, summary_val = sess.run(
18.            [prediction, pred_val, Y_accuracy, merged_summary
19.             _op], feed_dict={x:batch_x, y:batch_y, keep_prob_: 1.0})
20.        print("Step " + str(b) + ",",
21.              "Test Accuracy: " + str(acc))
22.        print("Y_1 shape = {}" .format(Y_1.shape))
23.        print("pred_2 shape = {}" .format(pred_2.shape))
24.        print("pred shape = {}" .format(pred.shape))
25.        print("Y_1: {}" .format(Y_1))
26.        print("pred_2: {}" .format(pred_2))
27.        print("pred: {}" .format(pred))
28.        precision_val = precision_score(Y_1, pred_2)
29.        recall_val = recall_score(Y_1, pred_2)
30.        f1_val = f1_score(Y_1, pred_2)
31.        precision_score_arr.append(precision_val)
32.        recall_score_arr.append(recall_val)
33.        f1_score_arr.append(f1_val)
34.        print("Precision: {}" .format(precision_val))
35.        print("Recall: {}" .format(recall_val))
36.        print("f1 score: {}" .format(f1_val))
37.        print("confusion_matrix")
38.        print(confusion_matrix(Y_1, pred_2))
39.        fpr, tpr, thresholds = roc_curve(Y_1, pred_2)
40.        print("Test precision mean = {}" .format(sum(precision
41.            _score_arr)/len(precision_score_arr)))
42.        print("Test recall mean = {}" .format(sum(recall_score
43.            _arr)/len(recall_score_arr)))
44.        print("Test f1 mean = {}" .format(sum(f1_score_arr)
45.            /len(f1_score_arr)))

```

연체 대출 리스크 혼합 예측 모델

■ 모델소개

LSTM과 전통적인 머신러닝에 기반한 연체 대출 리스크 혼합 예측 모델은 머신러닝과 딥러닝의 장점을 융합했습니다. 딥러닝을 통해 예측의 정확성을 보장했고, 머신러닝 방식을 통해 예측의 해석 가능성을 제공했습니다. 뿐만 아니라 이 혼합 모드는 인텔® 아키텍처 최적화 지향 TensorFlow와 인텔® Python 배포 패키지 등 고급 툴과 제품을 사용해 최적화를 실행하면서 상업은행 등 금융 기관에 고효율 예측 서비스를 제공할 수 있게 되었습니다.

이 모델의 기본 구조와 업무 프로세스는 그림 2-1-3 과 같습니다. 우선, 특징을 분석하고 데이터 전처리를 진행합니다. 이 단계에서는 금융 기관 해당 지역 빅데이터 플랫폼의 데이터 또는 서드 파티가 제공한 데이터(예: 신용 조회 기관의 데이터)가 시스템에서 처리됩니다. 결측 데이터 처리, 데이터 범위 처리, 데이터 불균형 처리 및 데이터의 중요한 특징 분석을 포함합니다. 또한 데이터 집약률이 증가하고 복잡해짐에 따라 이 모델은 여러 전처리 툴킷과 새로운 모델을 사용해 여러 유형의 데이터 입력에 대응할 수 있습니다.

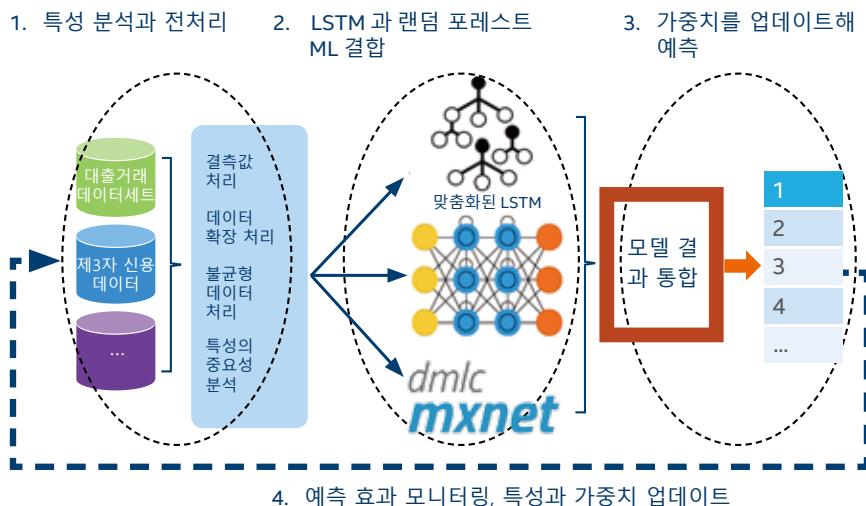


그림 2-1-3 LSTM과 전통적인 머신러닝 기반 혼합 모델

두 번째 단계는 딥러닝 모델 (LSTM)과 전통적인 머신러닝 모델 (XGBoost/RF)을 이용해 각각 표본 데이터에 트레이닝과 추론을 진행하고 각각의 관련 결과를 얻습니다. 그러면 혼합 모델이 결과물에 대해 각각 가중 처리하고 가중치를 갱신해 예측합니다.

마지막 단계는 이번 사이클의 예측 결과를 모델 헤드에 다시 유입한 뒤, 예측 효과에 따라 특징 값과 가중치를 갱신하고 다음 사이클 예측을 진행합니다.

■ 소프트웨어 스택

연체 대출 리스크 혼합 예측 모델 소프트웨어 스택은 그림 2-1-4와 같으며, 왼쪽 가장 아래쪽은 인텔® 제온® 골드 6130CPU와 인텔® 이더넷 융합 네트워크 어댑터 X710-DA2로 만든 하드웨어 인프라입니다. 그 위는 AI 기능 층으로 VMware* ESXi* 가 가상화 환경을 제공하고, RedHat Linux 운영 체제 (CentOS Linux release 7.4.1708)를 설치하고, 인텔® MKL-DNN 또는 인텔® MKL, 인텔® 아키텍처 최적화 지향 TensorFlow1.10 및 인텔® Python 배포 패키지를 배치했습니다. 오른쪽 가장 아래쪽은 인텔® 제온® 골드 6130CPU 와 인텔® 이더넷 융합 네트워크 어댑터 X710-DA2 로 만든 하드웨어 인프라입니다. 그 위는 데이터 층으로 RedHat Linux 운영 체제 (CentOS Linux release 7.4.1708)를 설치하고, Apache HBase* 분산형 데이터베이스 및 Hadoop 분산 파일 시스템 (Hadoop Distributed File System, HDFS)을 설치하고, 분산형 데이터 스토리지 읽기/쓰기 기능을 제공했습니다.

AI 기능 층과 데이터 층에는 연체 대출 리스크 혼합 예측 애플리케이션을 배치했습니다.



그림 2-1-4 연체 대출 리스크 혼합 예측 모델 소프트웨어 스택

■ 시스템 구조

실제 시스템에 적용했을 경우, 연체 대출 리스크 혼합 예측 방안은 그림 2-1-5 방식처럼 구성할 수 있습니다. 전체 시스템은 왼쪽 부터 오른쪽으로 외부

데이터 처리 서버 시스템과 온라인 시스템 및 오프라인 시스템으로 나뉩니다. 우선, 외부 데이터에 대해 시스템은 동일한 데이터 인터페이스를 통해 데이터 기획과 모니터링 플랫폼을 유입한 뒤, 서비스 인터페이스로 일부 데이터를 오프라인 시스템에 보냅니다.

그 다음, 오프라인 시스템에서는 외부 데이터 서버 시스템과 오프라인 시스템에서 가져온 일부 데이터가 하나의 데이터 마트 (Data Mart)에 유입됩니다. 이 데이터는 정리를 거친 뒤 오프라인의 모델 트레이닝과 알고리즘 배치 프로세서로 들어가며, 트레이닝을 거친 모델 알고리즘은 온라인 서버 시스템의 예측 시스템으로 유입됩니다.

온라인 서버 시스템에서는 전자 시스템이 우선 모델 트레이닝용으로 일부 데이터를 데이터 마트에 맡기고, 다른 데이터는 데이터 푸시 시스템을 통해 Storm 클러스터로 구성된 분산형 실시간 컴퓨팅 시스템에 들어가 예측 스케줄링 등 절차를 진행합니다. 그러면 데이터가 예측 시스템에 들어가 추론 예측을 진행하고, 얻은 결과는 테스트 플랫폼에 발송되어 검증을 진행하게 됩니다. 마지막 결과는 명단 관리 모듈 및 리스크 태그 생성 모듈에서 다시 오프라인 서버 시스템의 알고리즘 배치와 모델 트레이닝 모듈로 돌아가 알고리즘을 반복하게 됩니다.

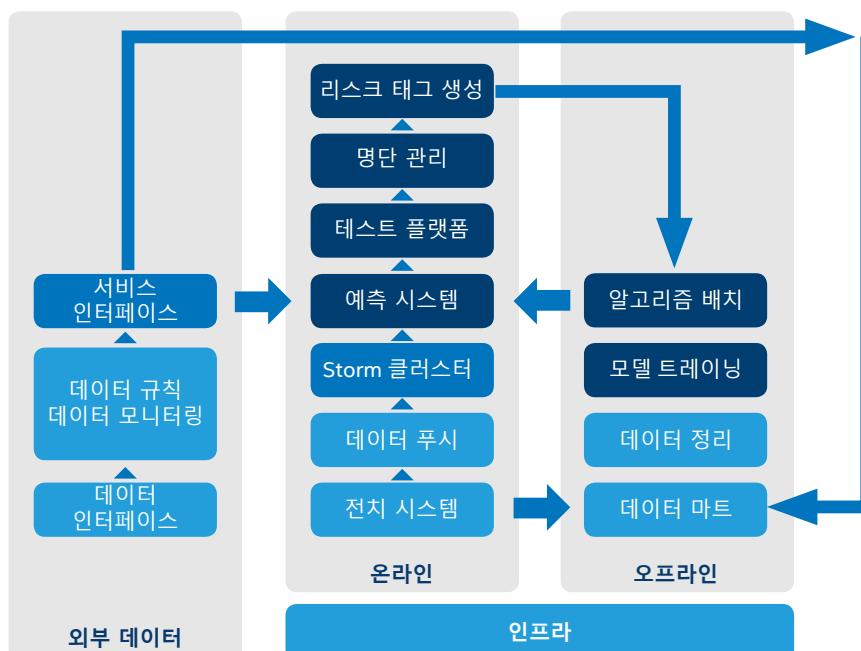


그림 2-1-5 연체 대출 리스크 혼합 예측 방안 시스템 아키텍처

소프트웨어/하드웨어 구성 제안

이상의 연체 대출 리스크 예측 모델 구성은 다음의 인텔® 아키텍처 플랫폼 환경 기반 구성을 참조할 수 있습니다.

하드웨어:

명칭	사양
CPU	튜웨이 인텔® 제온® 골드 6230 CPU 또는 그 이상
기저 주파수	2.10GHz
코어/스레드	16/32
HT	On
Turbo	On
메모리	16GB DDR4 2666MHz* 12
하드 디스크	인텔® DC S3320 데이터 센터급 SSD 480GB
BIOS	SE5C620.86B.00.01.0013.030920180427

소프트웨어:

명칭	사양
운영 체제	CentOS Linux release 7.4.1708 (Core)
Linux 커널	3.10.0-957.12.1.el7.x86_64
워크로드	LSTM/XGBoost
컴파일러	GCC 5.4
베이스	인텔® MKL 최신 버전
아키텍처	인텔® 아키텍처 최적화 지향 TensorFlow 배포 버전
기타소프트웨어 구성	인텔® Python 배포 패키지

대형 상업은행 응용 사례

성과

이 사용자의 실제 배치 검증에서 알 수 있듯이 마지막 혼합 방안은 예측 정확도를 효과적으로 향상시키고 예측 지연을 대폭 하락시킬 수 있습니다. 이는 인력 투입 예측 방안과 비교했을 때, LSTM 방식의 정확성이 1 배, 혼합 모델 방안의 예측 정확성은 2 배 이상 향상시킬 수 있으며, 예측 지연 시간은 2일(예측 효율 10 배 이상 향상) 단축되었음을 데이터로 나타내고 있습니다. 그 밖에도 온라인 예측 방안 (대출 가능 리스크 예측)의 예측별 시간이 모두 1 초보다 짧아지면서 고객 만족도 역시 눈에 띄게 향상됐습니다.



그림 2-2-1 예측 방안별 결과 비교

이 상업은행 사용자는 혼합 모델로 구성한 트레이닝과 추론 클러스터를 기반으로 하고 있으며, 모두 인텔® 아키텍처 플랫폼이 베이스입니다. 인텔® 아키텍처 CPU 플랫폼은 커널, 고속 캐시 등 측면에서 우수할 뿐만 아니라 다양한 하드웨어 강화 기술로 시스템 성능을 향상시킬 수 있습니다. 예를 들어, 인텔® 제온® CPU와 인텔® 제온® 확장성 CPU가 구비한 인텔® AVX-512 는 XGBoost 모델에 특별한 병렬 컴퓨팅 기능을 제공합니다.

요약

다양화된 인공지능 방식을 이용해 금융 사용자에게 적합한 솔루션을 제공하기 위해, 인텔은 자체 기술을 결합해 업계에 맞춤형 솔루션을 내놓는 좋은 시도를 했습니다. LSTM 과 전통적인 머신러닝을 혼합한 예측 모델은 예측 정확도에서 만족스러운 효과를 얻었을 뿐만 아니라 예측 과정 해석 가능성에 대한 고객의 니즈도 충분히 만족시켰습니다.

인텔은 이 신형 혼합 예측 모델에 고성능 CPU 제품과 인텔® 아키텍처 최적화 지향 TensorFlow, 인텔® Python 배포 패키지 등 다양한 소프트웨어 최적화 기능을 제공함으로써 업무 효율을 향상시켰습니다. 현재, 이 혼합 예측 모델은 어느 상업은행에 실제 배치되어 사용자에게 효율이 높고 정확한 연체 대출 리스크 예측 기능을 제공하고 있습니다. 앞으로 양측은 NLP 를 이용한 전체 사회 환경 데이터 분석 및 예측 방안을 탐색하고, 예측 효과를 더 전면적이고 정확하게 만들 계획입니다.

기존의 사례에는 인텔® 제온® CPU/인텔® 제온® 확장성 CPU 기반 서버를 사용했지만. 향후에는 사용자가 성능이 더 강하고, 인공지능 영역에 더 최적화된 2 세대 인텔® 제온® 확장성 CPU를 사용해 솔루션을 구성할 수 있을 것입니다.

AI 기반 금융 업계 정밀 마케팅 전략



AI 기반 금융 업계 정밀 마케팅 전략 탐색

배경 소개

금융 업계는 예전부터 줄곧 인공지능 기능을 적극 활용해 업무를 가속화하고 마케팅 효율의 패러다임을 향상시켜왔습니다. 그 이유는 다음과 같습니다. 첫 번째, 금융 업계의 기업은 종종 완전한 정보화 시스템을 구비하고 있고, 업무 데이터 채집과 누적을 중시하며, 이를 통해 누적된 다양한 데이터는 인공지능 응용에 건고한 기초를 제공합니다. 두 번째, 은행과 보험, 증권사 등 금융 관련 업무는 모두 데이터를 기반으로 전개되기 때문에 방대한 양의 번거로운 데이터 처리 업무 효율을 향상시키려면 반드시 인공지능의 도움이 필요합니다. 세 번째, 딥러닝이 빠르게 발전하면서 인공지능과 금융 업계의 융합이 더 다양한 시나리오에서 실행되었습니다. 그 중, AI 기반 금융 업계 정밀 마케팅 전략은 점점 더 많은 사람들이 관심을 보이는 부분입니다.

금융 업계의 높은 정보화 수준과 데이터 우위는 업계 내 기업의 유형별 추천 시스템 구성을 가속화했으며, "각인각색", "전체 사용자 이미지" 등의 방식으로 정밀 마케팅과 개인 맞춤형 마케팅 등 주요 응용 업무를 추진했습니다. 금융 기업은 다양한 구조적/비정형 데이터를 이용해 일련의 마케팅 의사결정을 구성해 단말 사용자의 취향, 사용 경험 및 구매 의도 등을 심도 있게 분석할 뿐만 아니라 시세 전망을 추측해 관련 금융 제품 또는 비즈니스 거래에 맞춤형 조언을 함으로써 마케팅 혁신에 새로운 원동력을 제공했습니다.

이러한 추세에 부응하기 위해, 여러 금융 업계의 기업은 인텔과 함께 탐색하는 과정 중에 있으며, 인텔이 개발한 "빅데이터 분석+AI" 플랫폼 Analytics Zoo 는 신경협력필터 (Neural Collaborative Filtering, NCF) 모델, WAD(Wide and Deep) 등 딥러닝 모델을 통해 고효율 업무 추천 시스템을 구성했습니다.

추천 시스템

■ 일반적인 추천 시스템

추천 시스템 (Recommender System, RS) 은 일종의 정보 필터 툴로 기업이 개인 맞춤형 방식으로 다양한 옵션 중 소비자의 취향을 발견하도록 인도합니다. 그를 통해 고객의 소비 경험을 개선하고, 기업의 마케팅 효과를 향상시킬 뿐만 아니라 목표 마케팅 제품/계획의 정확성 측면에서 중요한 역할을 합니다. 예를 들어, 구매 잠재력이 가장 높은 소비자에게 혜택을 주는 것은 누가 봐도 더 효과적인 일입니다.

현재 추천 시스템은 이미 여러 업계에서 매출과 서비스를 확장하는데 있어 중요한 툴이 되었습니다. 예를 들어, 사용자 중 80% 가 Netflix* 추천을 통해 다음에 볼 영화 5 를 선택하고, YouTube*는 그 60%⁶ 가 선택을 합니다. 또한 딥러닝 기반 추천 시스템은 추첨 품질 면에서 점점 더 인정⁷ 을 받고 있습니다.

⁵ Carlos A Gomez-Uribe and Neil Hunt. 2016. Netflix 추천 시스템: 알고리즘, 비즈니스 가치와 혁신입니다. 관리 정보 시스템의 ACM 사무(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. 4 회 ACM 추천 시스템 회의록 29페이지부터 29페이지까지(RecSys '10)입니다.

⁷ Shuai Zhang, Lina Yao, and Aixin Sun. Deep learning-based Recommender System: A Survey and New Perspectives. arXiv preprint arXiv: 1707.07435, 2017.

추천 모델은 일반적으로 협력 필터, 내용 중심, 그리고 혼합 시스템으로 나뉩니다. 협력 필터 기반 추천 알고리즘은 사용자는 이전에 구매한 상품과 유사한 제품을 선택할 확률이 크다는 관점을 기준으로, 사용자와 상품의 과거 조합을 학습하고 명시적인 수치(예를 들어 사용자의 이전 평가 등급) 또는 암묵적인 피드 백(예를 들어 사용자 구매 후 평가)을 이용해 의견을 제시합니다. 이 방식은 특징 값을 스크리닝할 필요가 없기 때문에 초기 모델로 비교적 적합합니다.

내용 중심의 추천 알고리즘은 사용자는 일반적으로 어느 항목의 콘텐츠와 관심을 가졌던 제품과 유사한 제품을 좋아한다는 것을 기본 원리입니다. 예를 들어 재테크 상품 A를 구매했다면, 내용 중심의 추천 알고리즘은 이전에 구매한 A와 수익률 또는 연한이 유사한 재테크 상품 B를 찾아내 추천해줍니다. 이러한 방법은 추천 시스템의 쿨타임 문제를 방지할 수 있다는 장점이 있지만, 때때로 중복 추천할 수 있다는 단점도 있습니다. 또한 이 알고리즘 역시 다양한 특징 값에 기반한 분석 방식입니다.

지금은 딥러닝이 고효율 추천 모델 구성에 점점 더 많이 쓰이고 있습니다. 전통적인 머신러닝 알고리즘은 이전의 솔루션에서 매우 중요한 역할을 하고 있지만, 모델과 피쳐 엔지니어링이나 날이 복잡해지다 보니 최근 들어서는 마케팅 활동의 효율성을 향상시키기 위해 여러 사람들이 딥러닝 기반 신경 추천 모델을 제시하고 있습니다.

■ 추천 시스템 구성 과정

그림 3-1-1과 같이 추천 시스템 구성 과정은: 데이터 정리, 피쳐 엔지니어링, 모델링, 평가 조정으로 나뉩니다.

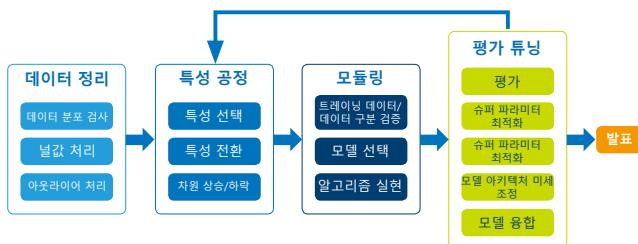


그림 3-1-1 추천 시스템 구성 과정

- 일반적으로 초기 데이터는 여러 오순 데이터를 포함하고 있기 때문에 모델 트레이닝과 예측의 정확도에 매우 영향을 미칩니다. 데이터 정리는 데이터를 다시 심사하고 검사해 데이터 일치를 보장하는 과정입니다. 데이터 정리는 주로 데이터 분포 검사와 아웃라이어 처리, 널값 처리 등 기능을 포함합니다.
- 피쳐 엔지니어링은 데이터에서 결과 예측에 필요한 정보를 추출하는 과정으로, 차원 전환을 진행해 마지막으로 특징 벡터를 형성합니다. 특징 선택, 특징 전환과 차원 상승/하락 등 주요 기능을 포함합니다.

- 모델링은 모델 선택, 모델 트레이닝과 알고리즘 실현을 포함하고 있습니다.
- 평가와 조정은 슈퍼 파라미터 최적화 모델 구조 조정 그리고 교차 검증과 모델 융합 등 작업이 포함됩니다. 조정 후에는 조정 결과에 따라 초기 데이터 정리와 피쳐 엔지니어링 부분으로 돌아갈지 여부를 판단해야 합니다.

Analytics Zoo

Analytics Zoo는 인텔이 개발한 "빅데이터 분석+AI" 플랫폼으로 TensorFlow, Keras, PyTorch, BigDL, Ray, Spark 및 Flink* 등 소프트웨어와 애플리케이션을 하나의 체계로 빈틈 없이 집성하고 대형 Apache Hadoop/Spark 클러스터로 확장해 딥러닝에 필요한 분산형 트레이닝 또는 예측에 사용합니다.

Analytic Zoo는 인텔® 제온® 확장형 CPU 기반 클러스터에서 운용해 기업의 딥러닝 수요를 만족시킵니다. Analytic Zoo는 사용자가 Apache Hadoop/Spark와 같은 빅데이터 인프라에서 딥러닝 애플리케이션을 개발하고 운용하는 것을 허용합니다. Analytic Zoo는 Plain Old Java Object(POJO) 와 해당 지역의 Java* API 또는 Scala* / Python 모델 로딩 API를 사용해 Web서비스 (Spark Streaming*, Kafka* 등)에 빈틈 없이 집성할 수 있습니다.

사용자는 Analytics Zoo를 통해 다음 작업을 진행할 수 있습니다.

- Spark로 데이터 처리 및 분석
- TensorFlow, Keras 또는 PyTorch로 딥러닝 모델 개발
- Spark와 BigDL에서 분산형 트레이닝/추론 진행

Analytics Zoo end-to-end, 일관된 빅데이터 분석+AI 플랫폼

응용 사례	추천	비정상행위 탐지	텍스트 분류	문자 매칭	
	영상 분류	객체 탐지	seq2seq	변환기	BERT
특성 공정	영상	3D 영상	텍스트	시계열	
하이레이어 어셈블리 플랫폼	tfpark : Spark 의 분산형 TF nnframes: 딥러닝에 사용하는 Spark 데이터 프레임과 머신러닝 어셈블리 플랫폼	Spark 의 분산형 자동 솔빙 그래디언트를 장착한 Keras	분산형 모델 서비스 (다양 처리, 어셈블리 처리와 온라인 처리)		
백 엔드/ 라이브러리	TensorFlow Ray	Keras 인텔® MKL-DNN	PyTorch OpenVINO™ 둘것	BigDL NLP Architect Apache Spark Apache Flink	인텔® 옵테인™ 데이터 센터급 영구 메모리 인텔® 딥러닝 가속

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

그림 3-1-2 Analytics Zoo 다양한 end-to-end 분석 기능과 AI 지원 제공

Analytics Zoo는 아래와 같은 다양한 end-to-end 분석 기능과 AI 지원도 제공합니다.

- 쉽게 사용할 수 있는 고급 분석 어셈블리 라인 API (예: 트랜스미션 러닝, 자동 프로그래밍 작업, Spark DataFrame, ML Pipelines, 온라인 모델 서비스 API 등)
- 영상, 텍스트, 3D 영상 등 자주 볼 수 있는 피쳐 엔지니어링 작업
- 다량의 내장 딥러닝 모델(예: 객체 탐지, 영상 분류, 추천, 비정상행위 탐지, 텍스트 매칭, seq2seq 등)
- 다양한 참고 사례(예: 비정상행위 탐지, 감성 분석, 금융 거래 사기 방지, 이미지 유사성 등).

기업이 Analytics Zoo를 사용하면 다음과 같은 경쟁력을 얻을 수 있습니다.

- 분석을 이동하거나 복제하지 않고 동일한 빅데이터 클러스터의 빅데이터 (HDFS, Apache HBase 와 Apache Hive 등)에 저장
 - 딥러닝 기능을 재구성하지 않고 기존의 분석 애플리케이션과 머신러닝 어셈블리 라인에 추가
 - 기존의 빅데이터 클러스터와 인프라(자원 할당, 부하 관리와 기업급 모니터링) 이용
 - 트레이닝 단계에서 교차 검증을 진행할 때, 딥러닝 알고리즘은 지수적 성장의 임베디드 특성을 생성하고, 내부 특성 선택과 최적화를 자동 실행하기 때문에 피쳐 엔지니어링 작업량이 현저히 감소합니다. 모델을 구성할 때 알고리즘은 미리 정의한 미끄럼 특성과 사용자 지정 중복 특성에만 중점을 두어 대부분의 Long Time Variable(LTV) 프리컴퓨테이션을 삭제하기 때문에 다양한 시간과 자원을 절감할 수 있습니다.
 - 전통적인 머신러닝 (ML) 방식은 전문가에 의존해 모델을 최적화했다면, Analytics Zoo는 더 다양한 옵션을 제공해 더 우수하고, 안정적인 실행 구성을 모색하기 때문에 자동 모델 최적화 기능을 대폭 향상시켰습니다.
 - Analytic Zoo는 인텔® 제온® CPU의 표준 Spark 프로세스로 운용할 수 있기 때문에 배치 또는 조작 비용이 제로입니다.
- * **Analytic Zoo 기술과 관련한 더 자세한 사항은 해당 매뉴얼 기술편 소개 부분을 참조하십시오.**

전형적인 AI 추천 딥러닝 모델

■ 신경 협업 필터링 (NCF) 모델

NCF⁸모델은 현재 자주 볼 수 있는 딥러닝 기반 추천 아이템 중 하나입니다⁹. 앞에서 서술한 바와 같이, 협업 필터링 알고리즘은 보통 명시적 피드백과 암시적 피드백에 의존하지만, 실제 적용 상황에서는 명시적 피드백은 두드러지지 않고, 암시적 피드백이 더 많이 드러납니다. 암시적 피드백 데이터에는 행렬 분해(MF)로 추상화해 문제를 추천할 수 있습니다. 하지만 전통적인 MF 모델은 잠재변수(latent factor)의 선형 모델로 사용자와 상품 간의 상호 작용에는 반응할 수는 있지만 사용자가 해당 상품을 실제로 좋아하는지에 대해서는 반응할 수 없습니다.

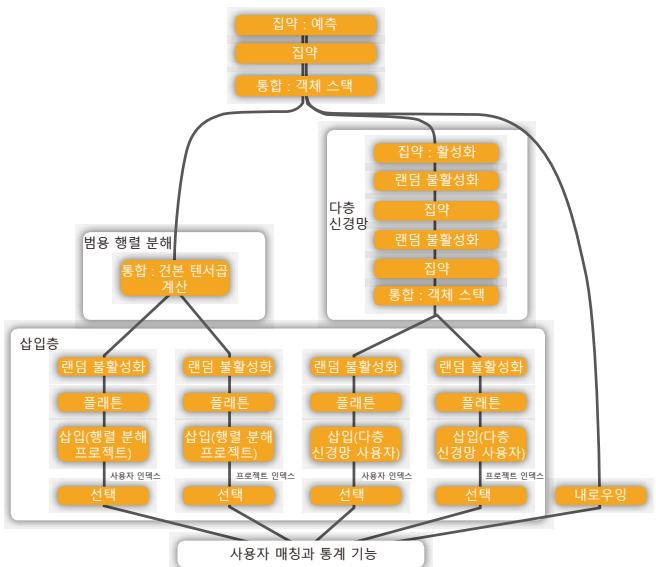


그림 3-1-3 신경 협업 필터링 (NCF) 모델 예시

NCF는 DNN을 도입해 이 문제를 해결합니다. DNN을 사용해 데이터에서 작용 함수를 학습해 MF 모델의 한계를 삭제합니다. 그림 3-1-3과 같이 Embedding Layer를 이용해 입력 레이어의 희소 표현을 하나의 새로운 잠재 벡터로 매핑한 뒤, 사용자 입력과 상품 입력을 각각 다중 신경망 아키텍처에 발송합니다. 왼쪽을 보면, 모델은 범용 행렬분석 (GMF) 구조를 사용해 선형 상호작용을 처리하고, 오른쪽을 보면, 다중 신경망(MLP)을 사용해 비선형 상호작용을 처리합니다. 그런 다음 두 개를 융합해 더 훌륭한 추천 효과를 획득하게 됩니다. 이제 사용자는 Analytics Zoo로 손쉽게 NCF 모델을 만들 수 있습니다.

⁸ NCF 기술 관련 설명은 다음을 참조하십시오. <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.

■ 와이드 앤 딥 모델

와이드 앤 딥 러닝 모델은 2016년에 출시된 DNN-Linear 혼합 모델로, 와이드 컴포넌트 모델과 딥 컴포넌트 모델 두 부분으로 나뉩니다. 그림 3-1-4 중 오른쪽 부분에서 볼 수 있듯이 와이드 컴포넌트 모델은 퍼셉트론이자 일반형 선형 모델입니다. 전통적인 추천 시스템을 불연속형 특성 기반 선형 알고리즘을 통한 방식과 비교하면 딥 컴포넌트 모델은 사용자가 과거에 클릭했던 페이지, 구매했던 상품 등 과거 행동 데이터를 얻은 뒤 코드를 통해 불연속형 특성을 구성해 컴퓨팅을 진행합니다.

와이드 컴포넌트 모델 추천 방식은 대규모의 희귀 데이터에 효과가 좋을 뿐만 아니라 모델에 강한 설명성을 지니고 있습니다. 로지스틱 회귀(LF)를 예로 들어보면 모든 불연속형 특성은 모델의 가중치와 대응되며, 특성의 가중치는 결과에 미치는 특성의 영향과 밀접한 관계가 있습니다. 하지만 와이드 컴포넌트 모델의 특성은 다량의 인위적인 간섭과 전문가의 경험이 개입되어야 파생이 가능하고 예측 효과 또한 낮습니다.

딥 컴포넌트 모델은 신경 협업 필터링 모델과 유사한 다층 신경망으로 딥러닝을 통해 일련의 벡터를 획득하고, 이 벡터를 특성의 일부분으로 삼아 트레이닝에 참여합니다. 딥 컴포넌트 모델로 생성된 특성은 다음과 같은 장점을 가지고 있습니다. 하나는 인위적으로 추출한 특성으로 야기된 차원적 한계를 메꾸고 예측의 정확도를 향상시킨다는 점입니다. 또 다른 하나는 이 특성은 딥러닝 아키텍처로 자동 생성되기 때문에 인력이 개입되지 않아도 트레이닝 효율을 향상시킬 수 있다는 점입니다. 하지만 딥 컴포넌트 모델로 생성된 벡터는 암시적 특성이다 보니 예측 과정에서 해석 가능성이 명확하지 않은 경우가 종종 발생합니다.

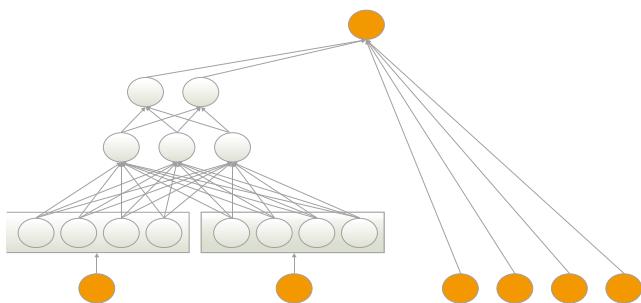


그림 3-1-4 와이드 앤 딥 모델 그림

그래서 WAD 모델은 추천 시스템의 효율을 높이기 위해 와이드 컴포넌트 모델과 딥 컴포넌트 모델을 결합했습니다. WAD 모델은 SparseTensor 와 희소 데이터를 사용해 명확하게 설계한 레이어로 SparseLinear, SparseJoinTable 등이 있습니다.

Analytics Zoo 는 WAD 모델을 적극 지원하며, DataFrame 과 RDD(Resilient Distributed Datasets) 두 개의 인터페이스를 데이터 준비와 트레이닝에 사용해 사용자의 환경에 따라 유연하게 적용합니다. 뿐만 아니라 Analytics Zoo 의 WAD 모델은 Spark 1.5 를 최신 버전에 호환되도록 허용합니다.

Analytics Zoo 기반 모델 실현

■ 신경 협업 필터링 모델 시스템 실현

아래는 Spark 의 Analytics Zoo 와 BigDL 을 기반으로 명시적 피드백 기반 NCF를 구성하는 방법을 설명하고 있습니다.

시스템 환경은 다음과 같습니다.

- Python 2.7/3.5/3.6
- JDK 8
- Spark 1.6.0/2.1.1/2.1.2/2.2.0 (컴파일 Analytics Zoo 의 Spark 버전과 일치해야 함)
- Analytics Zoo 0.5.0
- Jupyter Notebook 4.1

■ Analytics Zoo 다운로드 또는 설치

pip로 analytics-zoo 를 설치하거나 프리컴파일 패키지 (prebuilt package) 를 다운로드합니다. <https://analytics-zoo.github.io/master/#PythonUserGuide/install/> (<https://analytics-zoo.github.io> 홈페이지의 왼쪽 인덱스에서 User Guide -> Python -> Install 선택도 가능)을 참조하십시오.

PIP로 설치 후 운용

사용자는 다음 명령을 사용해 손쉽게 운용할 수 있습니다.

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

다음 주소를 참조하면 PIP 설치 후 운용 가이드에 대해 더 자세히 알 수 있습니다.

<https://analytics-zoo.github.io/master/#PythonUserGuide/run/#run-after-pip-install>

프리컴파일 패키지로 설치 후 운용

로컬 모드 (master=local[*]) 또는 클러스터 모드의 spark에 다음 명령을 운용합니다.

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

다음 주소를 참조하면 PIP 비설치 후 운용 가이드에 대해 더 자세히 알 수 있습니다.

<https://analytics-zoo.github.io/master/#PythonUserGuide/run/#run-without-pip-install>

■ 명시적 피드백 기반 NCF 실현

다음은 뉴럴 네트워크 추천 시스템과 명시적 피드백 기반 NCF 구성 방법을 설명하고 있습니다. 추천 시스템의 API를 사용해 Analytics Zoo에 모델을 구성할 수 있고, 대응하는 옵티마이저를 사용해 모델을 트레이닝할 수도 있습니다.

시스템 (추천 시스템: 원칙, 방법과 평가¹⁰)은 일반적으로 시스템 인터페이스를 통해 사용자에게 프로젝트 점수를 제공해 모델을 구성하고 개선합니다. 따라서 추천의 정확도는 사용자가 제공하는 평가 수량에 따라 결정됩니다.

NCF 다중 신경망을 이용해 사용자-프로젝트 간 상호작용 기능을 학습함과 동시에 해당 아키텍처에서 행렬 분해를 표현하고 확대할 수 있습니다. includeMF (불리언)은 행렬 분해를 갖추거나 갖추지 않은 NCF를 구성하도록 사용자에게 제공됩니다.

본 예시에서 사용한 데이터세트는 movieens-1M¹¹으로 4,000 편의 영화에 대한 사용자 6,000 명의 100 만 개의 평점 데이터로 5 개의 등급으로 나뉩니다. 우리는 이제 모든 세트 (사용자, 영화)를 5 개의 유형으로 구분하고 평균절대오차 (MAE) 알고리즘을 사용으로 계산 방식을 경험할 수 있게 됩니다.

참고문헌 :

- 영화 추천 케라스 실현 방법은 다음을 참조하십시오.

<https://github.com/ririw/ririw.github.io/blob/master/assets/Recommendingmovies.ipynb> 와 <http://blog.richardweiss.org/2016/09/25/movie-embeddings.html>

- NCF 관련 논문은 다음을 참조하십시오. <https://www.comp.nus.edu.sg/~xiangnan/papers/ncf.pdf>

들여오기전에 필요한 라이브러리 :

```
1. In [1]:
2. from zoo.pipeline.api.keras.layers import *
3. from zoo.models.recommendation import UserItemFeature
4. from zoo.models.recommendation import NeuralCF
5. from zoo.common.nncontext import init_nncontext
6. import matplotlib
7. from sklearn import metrics
8. from operator import itemgetter
9. from bigdl.dataset import movielens
10. from bigdl.util.common import *
11. matplotlib.use('agg')
12. import matplotlib.pyplot as plt
13. %pylab inline
14. Populating the interactive namespace from numpy and matplotlib
```

NN 앞뒤 문장을 초기화하면 BigDL 성능을 최적화 성능 구성에 쓰이는 SparkContext를 얻을 수 있습니다.

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

■ 데이터 준비: 1M 크기의 movielens 데이터를 다운로드 및 리드 인:

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

데이터 중 모든 기록의 형식은 (userid, movieid, rating_score)입니다. 사용자 ID의 범위는 1부터 6,040 사이이며, 영화 ID의 범위는 1부터 3,952 사이입니다. 평점은 5 레벨 (전체 레벨에만 국한됨)로 향후 사용할 수 있도록 사용자 수와 영화 수를 기록합니다.

```
1. In [4]:
2. min_user_id = np.min(movielens_data[:,0])
3. max_user_id = np.max(movielens_data[:,0])
4. min_movie_id = np.min(movielens_data[:,1])
5. max_movie_id = np.max(movielens_data[:,1])
6. rating_labels = np.unique(movielens_data[:,2])
7. print(movielens_data.shape)
8. print(min_user_id, max_user_id, min_movie_id, max_movie_id, rating_labels)
9. (1000209, 3)
10. (1, 6040, 1, 3952, array([1, 2, 3, 4, 5]))
```

초기 데이터를 RDD12 형식의 표본으로 전환합니다. 본 예시에서는 BigDL의 옵티마이저를 사용해 바로 모델을 트레이닝했으며, RDD 형식으로 데이터를 제공해야 합니다. 다음 예시는 하나의 BigDL 데이터 구조로 각각 2개의 numpy 배열과 feature, label을 사용할 수 있습니다. 여기에서 API 인터페이스는 Sample.from_ndarray (feature, label)이며, 태그를 1에서 시작해 0으로 전환하는데 쓰입니다.

```
1. In [5]:
2. def build_sample(user_id, item_id, rating):
3.     sample = Sample.from_ndarray(np.array([user_id, item_id]), np.array([rating]))
4.     return UserItemFeature(user_id, item_id, sample)
5. pairFeatureRdds = sc.parallelize(movielens_data)\n6. .map(lambda x: build_sample(x[0], x[1], x[2]-1))
```

¹⁰ 세부 정보는 다음을 참조하십시오. <http://www.sciencedirect.com/science/article/pii/S1110866515000341>

¹¹ 데이터세트는 <https://grouplens.org/datasets/movielens/1m/>를 참조하십시오

¹² RDD 기본에 대한 구체적인 설명은 다음을 참조하십시오. <https://bigdl-project.github.io/master/#APIGuide/Data/#sample>

```

7. pairFeatureRdds.take(3)
8. Out[5]:
9. [<zoo.models.recommendation.recommender.UserItemFeature at 0x11473ffd0>,
10. <zoo.models.recommendation.recommender.UserItemFeature at 0x124eb9110>,
11. <zoo.models.recommendation.recommender.UserItemFeature at 0x11473fed0>]

```

데이터는 시퀀스 (80%) 와 검증 (20%) 으로 랜덤 구분됩니다.

```

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

3. valPairFeatureRdds.cache()
4. train_rdd= trainPairFeatureRdds.map(lambda pair_feature: pair_feature.sample)
5. val_rdd= valPairFeatureRdds.map(lambda pair_feature: pair_feature.sample)
6. val_rdd.persist()
7. Out[6]:
8. PythonRDD[3] at RDD at PythonRDD.scala:48
9. In [7]:
10. print(train_rdd.count())
11. train_rdd.take(3)
12. 799923
13. Out[7]:
14. [Sample: features: [JTensor: storage: [ 1. 661], shape: [2], float], labels: [JTENSOR: storage: [2],
   , shape: [1], float],
15. Sample: features: [JTENSOR: storage: [ 1. 914], shape: [2], float], labels: [JTENSOR: storage: [2],
   , shape: [1], float],
16. Sample: features: [JTENSOR: storage: [1.000e+00 3.408e+03], shape: [2], float], labels: [JTENS or: storage: [3], shape: [1], float]]

```

■ 모델 구성

Analytics Zoo 에서는 Neuracf API를 손쉽게 호출해 NCF모델을 구성할 수 있습니다 (데이터에 따라 사용자 계수, 프로젝트 계수와 분류 번호를 지정한 뒤 필요에 따라 숨겨진 층을 추가할 수 있고, 네트워크에 행렬 분해를 포함할 수도 있음). 이 모델에는 BigDL의 옵티마이저 또는 Analytics Zoo 의 NNClassifier를 입력할 수 있습니다. 다음 예시에서는 BigDL 옵티마이저의 사용 방법을 설명하고 있습니다.

```

1. In [8]:
2. ncf = NeuralCF(user_count=max_user_id,
3.                 item_count=max_movie_id,
4.                 class_num=5,
5.                 hidden_layers=[20, 10],
6.                 include_mf = False)
7. creating: createZooKerasInput
8. creating: createZooKerasFlatten
9. creating: createZooKerasSelect
10. creating: createZooKerasFlatten
11. creating: createZooKerasSelect
12. creating: createZooKerasEmbedding
13. creating: createZooKerasEmbedding
14. creating: createZooKerasFlatten
15. creating: createZooKerasFlatten
16. creating: createZooKerasMerge
17. creating: createZooKerasDense
18. creating: createZooKerasDense
19. creating: createZooKerasDense
20. creating: createZooKerasModel
21. creating: createZooNeuralCF

```

■ 모델 컴파일

특정한 옵티마이저, 손실과 평가 지표에 따라 모델을 컴파일하면 옵티마이저는 트레이닝 세트에서 가중치/편차의 순실 관련 뉴럴 네트워크를 최대한 감소시킵니다. BigDL 에 옵티마이저를 생성하려면 적어도 model (뉴럴 네트워크 모델), criteria (손실함수), traing_dd (트레이닝 데이터세트)와 batch size 를 포함한 다음 파라미터를 지정해야 합니다.

고효율 옵티마이저 생성과 관련한 자세한 정보는 다음의 프로그래밍 가이드와 옵티마이저 매뉴얼을 참조하십시오.
 프로그래밍 가이드 : <https://bigdl-project.github.io/master/#ProgrammingGuide/optimization/>
 옵티마이저 : <https://bigdl-project.github.io/master/#APIGuide/Optimizers/Optimizer/>

```

1. In [9]:
2. ncf.compile(optimizer= "adam",
3.               loss= "sparse_categorical_crossentropy",
4.               metrics=[accuracy])
5. creating: createAdam
6. creating: createZooKerasSparseCategoricalCrossEntropy
7. creating: createZooKerasSparseCategoricalAccuracy

```

■ 수집 일지

tensorboard 로 요약을 조회할 수 있습니다.

```

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

```

■ 트레이닝 모델

```

1. In [11]:
2. ncf.fit(train_rdd,
3.          nb_epoch= 10,
4.          batch_size= 8000,
5.          validation_data=val_rdd)

```

■ 예측

Analytics Zoo 모델은 model.predict (val-rdd) API 로 지정된 데이터를 추론합니다. RDD 결과 반환, Predict_class 유형으로 예측 태그를 반환합니다.

```

1. In [12]:
2. results = ncf.predict(val_rdd)
3. results.take(5)
4.
5. results_class = ncf.predict_class(val_rdd)
6. results_class.take(5)
7. Out[12]:
8. [5, 5, 4, 4, 4]

```

Analytics Zoo 에서는 3 개의 독특한 API 로 예측한 사용자 항목 세트를 제공하고, 사용자 또는 지정된 옵션 항목에 추천을 제시합니다.

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

사용자마다 3 개의 항목을 추천하고, RDD 특성에 후보 항목을 제시합니다.

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

항목마다 3 개의 사용자를 추천하고, RDD 특성에 후보 항목을 제시합니다.

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

■ 평가

드로잉 트레이닝과 손실 곡선 검증:

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

드로잉 정확도:

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

와이드 앤 딥 네트워크 실시 사례

아래 문장에서는 Analytics Zoo 를 사용한 추천 API를 넓은 선형 모델과 DNN 을 만듭니다. 즉, 와이드 앤 딥 네트워크를 가리키며, BigDL 옵티マイ저를 사용해 네트워크를 트레이닝합니다. 와이드 앤 딥 모델은 기억 강도와 보편화를 결합해 대규모의 회귀와 분류 문제에 사용될 수 있습니다. 돌발적인 입력 특성 (예: 다량의 가능성 값을 지닌 유형 특성)을 포함합니다.

시스템 환경은 다음과 같습니다.

- Python 2.7/3.5/3.6
- DK 8
- Spark 1.6.0/2.1.1/2.1.2/2.2.0 (컴파일 Analytics Zoo 의 Spark 버전과 일치해야 함)
- Analytics Zoo 0.5.0
- Jupyter Notebook 4.1

Analytics Zoo 다운로드 또는 설치는 38 페이지의 운용 가이드를 참조하십시오.

■ 초기화

필요한 라이브 들여오기

```
1. In [1]:
2. from zoo.models.recommendation import *
3. from zoo.models.recommendation.utils import *
4. from zoo.common.nncontext import init_nncontext
5. import os
6. import sys
7. import datetime as dt
8. import matplotlib
9. matplotlib.use('agg')
10. import matplotlib.pyplot as plt
11. %pylab inline
12. Populating the interactive namespace from numpy and matplotlib
```

NN 앞뒤 문장을 초기화하면 BigDL 성능을 최적화 성능 구성에 쓰이는 SparkContext 를 얻을 수 있습니다.

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

■ 데이터 준비

movielens 1M 평점 데이터를 다운로드 및 리드 인해 차원을 파악합니다.

```

1. In [3]:
2. from bigdl.dataset import movielens
3. movielens_data = movielens.get_id_ratings("/tmp/movielens/")
4. min_user_id = np.min(movielens_data[:,0])
5. max_user_id = np.max(movielens_data[:,0])
6. min_movie_id = np.min(movielens_data[:,1])
7. max_movie_id = np.max(movielens_data[:,1])
8. rating_labels= np.unique(movielens_data[:,2])
9. print(movielens_data.shape)
10. print(min_user_id, max_user_id, min_movie_id, max_movie_id, rating_labels)
11. (1000209, 3)
12. (1, 6040, 1, 3952, array([1, 2, 3, 4, 5]))
```

평점 데이터를 데이터 프레임으로 전환해 사용자와 프로젝트 데이터를 데이터 프레임으로 읽습니다. 태그를 1부터 시작해 0으로 전환합니다.

```

1. In [4]:
2. sqlContext = SQLContext(sc)
3. from pyspark.sqltypes import *
4. from pyspark.sql import Row
5. Rating = Row("userId", "itemId", "label")
6. User = Row("userId", "gender", "age", "occupation")
7. Item = Row("itemId", "title", "genres")
8.
9. ratings = sc.parallelize(movielens_data)\.
10. .map(lambda t: (int(t[0]), int(t[1]), int(t[2])-1))\.
11. .map(lambda r: Rating(*r))
12. ratingDF = sqlContext.createDataFrame(ratings)
13.
14. users= sc.textFile("/tmp/movielens/ml-1m/users.dat")\
15. .map(lambda t: t.split("\t")[0:4])\
16. .map(lambda t: (int(t[0]), t[1], int(t[2]), int(t[3])))\
17. .map(lambda r: User(*r))
18. userDF = sqlContext.createDataFrame(users)
19.
20. items = sc.textFile("/tmp/movielens/ml-1m/movies.dat")\
21. .map(lambda t: t.split("\t")[0:3])\
22. .map(lambda t: (int(t[0]), t[1], t[2].split("%")))\.
23. .map(lambda r: Item(*r))
24. itemDF = sqlContext.createDataFrame(items)
```

데이터를 연결 및 전환합니다. 예를 들어, 성별은 분류 특성으로, 직업과 성별은 교차 특성으로 쓰입니다.

```

1. In [5]:
2. from pyspark.sql.functions import col, udf
3.
4. gender_udf = udf(lambda gender: categorical_from_vocab_list(gender, ["F", "M"], start=1))
5. bucket_cross_udf = udf(lambda feature1, feature2: hash_bucket(str(feature1) + "_" + str(feature2), bucket_size=100))
6. genres_list = ["Crime", "Romance", "Thriller", "Adventure", "Drama", "Children's",
7. "War", "Documentary", "Fantasy", "Mystery", "Musical", "Animation", "Film-Noir", "Horror",
8. "Western", "Comedy", "Action", "Sci-Fi"]
9. genres_udf = udf(lambda genres: categorical_from_vocab_list(genres, genres_list, start=1))
10.
11. allDF = ratingDF.join(userDF, ["userId"]).join(itemDF, ["itemId"])\.
12. .withColumn("gender", gender_udf(col("gender")).cast("int"))\.
```

```

13. .withColumn("age-gender", bucket_cross_udf(col("age"), col("gender")).cast("int"))\.
14. .withColumn("genres", genres_udf(col("genres")).cast("int"))
15. allDF.show(5)
16. +-----+-----+-----+-----+
17. |itemid|userid|label|gender|age|occupation| title|genres|age-gender|
18. +-----+-----+-----+-----+
19. | 26| 3391| 3| 2| 18| 4|Othello (1995)| 5| 24|
20. | 26| 1447| 4| 2| 18| 4|Othello (1995)| 5| 24|
21. | 26| 5107| 3| 1| 45| 0|Othello (1995)| 5| 5|
22. | 26| 2878| 3| 1| 50| 20|Othello (1995)| 5| 47|
23. | 26| 1527| 1| 2| 18| 10|Othello (1995)| 5| 24|
24. +-----+-----+-----+-----+
25. only showing top 5 rows
```

와이드 앤 딥 모델이 공유한 특수 데이터 특성 정보 및 그 특성 생성입니다. 여기에서 우리는 직업 성별을 광범위한 기초 부분으로, 연령과 성별을 광범위한 교차 부분으로, 유파와 성별을 지표로, 사용자 ID와 프로젝트 ID를 임베디드에 씁니다.

```

1. In [6]:
2. bucket_size = 100
3. column_info = ColumnFeatureInfo(
4.     wide_base_cols=["occupation", "gender"],
5.     wide_base_dims=[21, 3],
6.     wide_cross_cols=["age-gender"],
7.     wide_cross_dims=[bucket_size],
8.     indicator_cols=["genres", "gender"],
9.     indicator_dims=[19, 3],
10.    embed_cols=["userId", "itemId"],
11.    embed_in_dims=[max_user_id, max_movie_id],
12.    embed_out_dims=[64, 64],
13.    continuous_cols=["age"])
```

데이터를 표본의 RDD로 전환하면 BigDL의 옵티마이저를 사용해 모델을 트레이닝할 수 있습니다. RDD(sample) 형식으로 데이터를 제공해야 합니다. 다음 예시는 하나의 BigDL 데이터 구조로 각각 2 개의 numpy 배열과 feature, label을 사용할 수 있습니다. API 인터페이스는 sample.from_ndarray(feature, label)입니다. 와이드 앤 딥 모델은 2 개의 입력 텐서가 필요한데, 하나는 와이드 모델의 스파스 텐서고 또 다른 하나는 딥 모델의 인텐시브 텐서입니다.

```

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

■ 와이드 앤 딥 모델 생성

AnalyticsZoo에서는 와이드 앤 딥 API를 호출해 와이드 앤 딥 모델을 손쉽게 만들 수 있습니다. 데이터에 따라 모델 유형, 분류 번호 및 특성의 행렬 정보를 지정합니다. 또한 숨겨진 층과 같은 네트워크의 기타 기본값을 변경할 수 있습니다. 이 모델은 BigDL의 옵티마이저, AnalyticsZoo의 NNClassifier를 입력할 수 있습니다. 다음 사례는 BigDL의 옵티마이저 사용 방법을 설명하고 있습니다.

```

1. In [8]:
2. wide_n_deep = WideAndDeep(5, column_info, "wide_n_deep")
3. creating: createZooKerasInput
4. creating: createZooKerasInput
5. creating: createZooKerasInput
6. creating: createZooKerasInput
7. creating: createZooKerasSparseDense
8. creating: createZooKerasFlatten
9. creating: createZooKerasSelect
10. creating: createZooKerasEmbedding
11. creating: createZooKerasFlatten
12. creating: createZooKerasFlatten
13. creating: createZooKerasSelect
14. creating: createZooKerasEmbedding
15. creating: createZooKerasFlatten
16. creating: createZooKerasMerge
17. creating: createZooKerasDense
18. creating: createZooKerasDense
19. creating: createZooKerasDense
20. creating: createZooKerasDense
21. creating: createZooKerasMerge
22. creating: createZooKerasActivation
23. creating: createZooKerasModel
24. creating: createZooWideAndDeep

```

■ 트레이닝 모델 생성 및 최적화:

```

1. In [9]:
2. wide_n_deep.compile(optimizer = "adam",
3.                      loss="sparse_categorical_crossentropy",
4.                      metrics=['accuracy'])
5. creating: createAdam
6. creating: createZooKerasSparseCategoricalCrossEntropy
7. creating: createZooKerasSparseCategoricalAccuracy
8. In [10]:
9. tmp_log_dir = create_tmp_path()
10. wide_n_deep.set_tensorboard(tmp_log_dir, "training_wideanddeep")

```

네트워크를 완료될 때까지 트레이닝하고 트레이닝 완료된 모델을 획득합니다.

```

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

```

■ 예측과 추천

Analytics Zoo 모델은 model.predict (val-rdd) API로 지정된 데이터를 추론합니다. RDD 결과를 반환합니다. Predict_class 유형으로 예측 태그를 반환합니다.

```

1. In [12]:
2. results = wide_n_deep.predict(test_data)
3. results.take(5)
4.
5. results_class = wide_n_deep.predict_class(test_data)
6. results_class.take(5)
7. Out[12]:
8. [4, 2, 4, 5, 2]

```

Analytics Zoo에서는 3 개의 독특한 API로 예측한 사용자 항목 세트를 제공하고, 사용자 또는 지정된 옵션 항목에 추천을 제시합니다.

```

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

```

사용자마다 3 개의 항목을 추천하고, RDD 특성에 후보 항목을 제시합니다.

```

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

```

사용자마다 3 개의 항목을 추천하고, RDD 특성에 후보 항목을 제시합니다.

```

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

```

■ 수축 곡선 드로잉:

```

1. In [16]:
2. #retrieve train and validation summary object and read the loss data into ndarray's.
3. train_loss = np.array(wide_n_deep.get_train_summary("Loss"))
4. val_loss = np.array(wide_n_deep.get_validation_summary("Loss"))
5. #plot the train and validation curves
6. # each event data is a tuple in form of (iteration_count, value, timestamp)
7. plt.figure(figsize = (12,6))
8. plt.plot(train_loss[:,0],train_loss[:,1],label='train loss')
9. plt.plot(val_loss[:,0],val_loss[:,1],label='val loss',color='green')
10. plt.scatter(val_loss[:,0],val_loss[:,1],color='green')

```

```

11. plt.legend();
12. plt.xlim(0,train_loss.shape[0]+10)
13. plt.grid(True)
14. plt.title("loss")
15. Out[15]:
16. Text(0.5,1,'loss')

```

드로잉 정밀도:

```

1. In [17]:
2. plt.figure(figsize = (12,6))
3. top1 = np.array(wide_n_deep.get_validation_summary
   ("Top1Accuracy"))
4. plt.plot(top1[:,0],top1[:,1],label="top1")
5. plt.title("top1 accuracy")
6. plt.grid(True)
7. plt.legend();
8. plt.xlim(0,train_loss.shape[0]+10)
9. Out[17]:
10. (0, 1010)
11.
12. In [18]:
13. valPairFeatureRdds.unpersist()
14. Out[18]:
15. PythonRDD[82] at RDD at PythonRDD.scala:48
16. In [19]:
17. sc.stop()

```

소프트웨어/하드웨어 구성 제안

이상은 AI 기반 정밀 마케팅 전략 모델 구성 방법으로 아래의 인텔® 아키텍처 기반 플랫폼을 참조할 수 있습니다. 환경 구성은 다음과 같습니다.

하드웨어 구성

명칭	사양
CPU	투웨이 인텔® 제온® CPU E5-2650 v4 또는 그 이상
기저 주파수	2.20GHz
기저 주파수	12/24
HT	BIOS 기본 설정 (enabled 또는 disabled 모두 가능)
Turbo	BIOS 기본 설정 (enabled 또는 disabled 모두 가능)
메모리	384GB
하드디스크	21TB
BIOS	출하 설정 또는 후속 업그레이드의 모든 버전
기타 하드웨어 구성	10GbE 네트워크 대역폭

소프트웨어 구성

명칭	사양
운영 체제	Ubuntu 14.04 LTS * 최신 지원 운영 체제 버전, https://analytics-zoo.github.io/ 을 참조하십시오
Linux 커널	3.14
워크로드	Analytics Zoo based NCF, WAD, ALS model training & model inference.
컴파일러	gcc 4.8
베이스	Analytics Zoo-bigdl_0.6.0-spark_2.2.0 (인텔® MKL 포함) • Spark MLlib 2.2.0
아키텍처	Analytics Zoo、BigDL
기타 소프트웨어 구성	<ul style="list-style-type: none"> Hadoop 배포버전 : Cloudera Distributed Hadoop (CDH) 5.12.1 Spark 버전 : 2.2 Java 플랫폼, 표준버전 개발 툴킷 (JDK *) 1.8

응용사례

차이나생명보험상하이 데이터센터 생명보험업무 재발견 실현

배경

보험료가 4,000 억 위안을 초과하는 초대형 보험 기업 휘하의 중요한 일원으로서 차이나 생명보험 상하이 데이터 센터는 앞선 인공지능 기능을 구축하기 위해 힘쓰고 있으며, 서포터는 그 덕분에 고객에게 개인 맞춤형 보험을 효율적으로 추천할 수 있어 업무 규모와 보험 종류 규모 확대로 인한 문제를 해결 할 수 있습니다.

이전에는 마케팅 직원 개인의 업무 경험에 근거해 현재 회사가 집중 판매하는 보험만 고객에게 추천하다 보니 고객의 니즈를 고려하지 못했습니다. 그렇다 보니 두 가지 문제를 야기했습니다. 하나는 고객의 니즈를 만족시키지 못했다는 점이고, 또

다른 하나는 계약 취소 또는 보험 해약으로 인해 회사 수익에 손실을 끼친다는 점입니다.

판매원이 보험을 추천할 때 방법론이 부족한 것 이 이 문제의 주요 원인 이었습니다. 특히 경험 이 없는 젊은 판매원의 경우에는 오해를 살만한 판매 가자주 일어났습니다. 그래서 차이나 생명보험 상하이 데이터 센터는 판매원이 효율적으로 보험을 추천해 고객의 만족도를 높일 수 있도록 데이터를 베이스로 AI 기반 추천 모델을 구성했습니다.

방법과 성과

차이나 생명보험 상하이 데이터 센터 업무 추천 시스템 플랫폼 아키텍처는 그림 3-2-1과 같습니다. 이 플랫폼은 Analytics Zoo를 기반으로 하며, 빅데이터 플랫폼은 CDH 5.10 버전을 채택했습니다. Sqoop으로 업무 시스템의 데이터를 HDFS에 들여오기하며, 데이터 정리와 부분 전처리는 Hive / Impala를 사용해 진행하거나 Python, Scala를 사용해 데이터 전처리를 진행하기도 합니다. 그런 다음에는 전처리한 데이터를 IMPALA 또는 HIVE에 저장합니다. 마지막으로 Spark On Hive를 사용해 구조적 형식으로 데이터를 읽고 BigDL을 호출해 모델 트레이닝을 진행합니다.

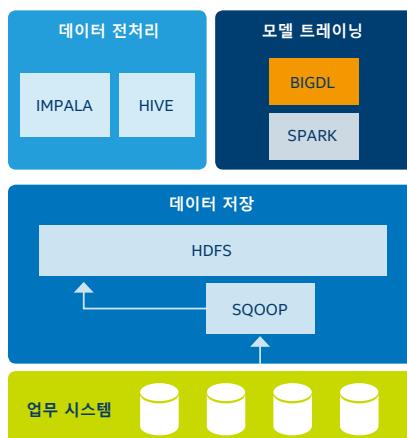


그림 3-2-1 차이나 생명보험 상하이 데이터 센터 업무 추천 시스템 플랫폼 아키텍처

딥러닝 기반 차이나 생명보험 상하이 데이터 센터 업무 추천 시스템은 주로 NCF 모델 관련 소개에서 설명한 바와 같은 NCF 모델을 채택하며, 모델은 왼쪽과 오른쪽으로 나뉩니다. 왼쪽은 범용 행렬 분해로 입력 벡터에 곱셈을 진행합니다. 오른쪽은 MLP 으로 입력한 user 와 item 의 특성을 한데 모아 다층 변환을 진행한 뒤 상위 레이어에서 두 개의 모델을 결합하고, sigmoid 방식으로 이 특성을 마지막 막 사용자의 경향 값으로 전환합니다. 이 사례에서 NCF 파라미터의 설정은 다음과 같습니다.

- Embedding 초기화 평균값 0, 편방 편차 0.01 의 정규 분포
- Batch Size 는 2800 으로 설정
- 튜닝 방법은 Adam

모델의 출력은 모든 보험에 대한 사용자의 평점으로, 이 평점으로 순서를 배열해 높은 점수를 득점한 몇 개의 보험을 사용자에게 추천합니다.

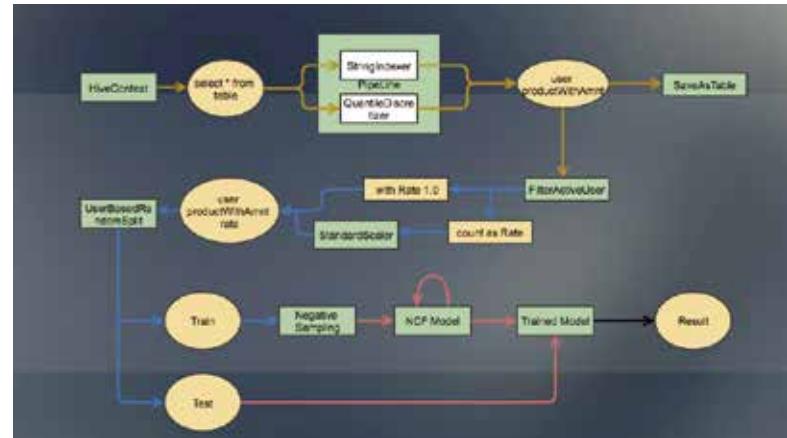


그림 3-2-2 차이나 생명보험 상하이 데이터 센터 추천 모델 기본 처리 프로세스

그림 3-2-2 와 같이 추천 모델의 기본 처리 프로세스는 다음 절차로 나뉩니다.

1. 데이터 전처리 과정: HiveContext를 사용, SQL 방식으로 Hive에서 직접 데이터를 읽습니다.
2. 데이터를 읽으면, Spark 의 DataFrame 객체로 저장합니다. 데이터가 뉴럴 네트워크에 적용할 수 있도록 Spark 중 PipeLine 인터페이스를 통하고 String indexer 를 사용해 데이터를 불연속형 데이터에 매핑합니다.
3. (user, productWithAmnt) 와 같은 데이터 세트를 획득한 뒤, 재작업을 진행합니다. 데이터마다 사용자 구매 취향 평가를 추가합니다. 예를 들어 구매한 적 있는 물건의 평가 rate 를 1 로 설정합니다.
4. 트레이닝 세트와 테스트 세트를 구분하고, Spark API 를 통해 트레이닝 데이터에 부정 데이터를 추가합니다.
5. 마지막에 트레이닝 세트는 데이터 트레이닝에 쓰이고, 테스트 세트는 모델 트레이닝 결과 검증에 쓰입니다.

이 절차는 Analytics Zoo 에 구성된 플랫폼을 통하고 구비한 다양한 고급 분석 어셈블리 API 와 특성에 따라 Spark DataFrame, ML Pipelines 등을 강력하게 지원해 전체 프로세스의 업무 효율을 향상시킵니다.

차이나 생명보험 상하이 데이터 센터는 두 가지 주요 지표를 통해 추천 시스템의 결과를 평가합니다. 두 가지 지표는 명중률 (Hit Rate) 과 NDCG(Normalized Distributed Cumulative Gain) 입니다. 이 예시에서는 그림 3-2-3 에서 보는 바와 같이 차이나 생명보험 상하이 데이터 센터 추천 시스템의 Hit Rate 이 99.8%, NDCG는 0.66 에 달합니다. 이 결과는 예상치를 초과하며, 이는 곧 추천 시스템이 훌륭한 효과를 지니고 있음을 의미합니다.

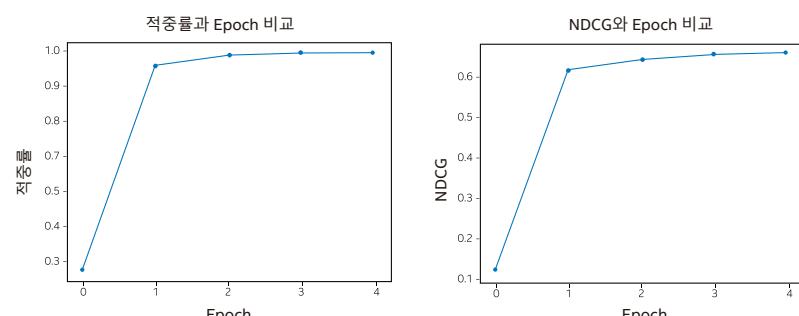


그림 3-2-3 차이나 생명보험 상하이 데이터 센터 추천 시스템 효과의 평가 결과

마스터 카드 추천 서비스 최적화

■ 배경과 도전

세계를 선두하는 페이먼트 솔루션 업체 마스터 카드 (MasterCard*) 는 26 억장의 신용카드를 보유하고 있고, 연 거래량도 560 억에 달합니다. 현재 마스터 카드는 AI를 플랫폼에 집성해 더 나은 고객 서비스를 제공하려 하고 있으며, 그 과정에서 마스터 카드는 다음과 같은 도전에 직면해 있습니다.

- 배치 시간이 길고, 다양한 딥러닝 모듈을 모두 마스터 카드의 기존 시스템에 재구성해야 합니다.
- 마스터 카드 기존의 ETL, 데이터 웨어하우스와 기타 분석 관련 데이터 기술 및 툴킷 등 다른 기업 정보화 모듈과의 호환성이 떨어집니다.
- 데이터를 여러 모듈 사이에서 빈번하게 복제해야 하다 보니 I/O 성능이 병목 현상이 일어납니다.

이러한 도전에 맞서기 위해 마스터 카드는 인텔과 협작해 Analytics Zoo "빅데이터 분석+AI" 플랫폼을 도입했고, 이를 통해 딥러닝 기반 추천 알고리즘을 구축했습니다. 이는 최신 연구와 업계 실제 적용 사례를 기반으로 한 방안으로 NCF와 와이드 앤 딥 WAD 모델을 추천의 두 가지 모델로 삼았으며, Analytics Zoo의 Keras 스타일 API 역시 Python과 Scala를 기반으로 딥러닝 모델을 구축하는데 사용되었습니다.

모델 구축 완료 후, 마스터 카드는 Analytics Zoo의 서비스 API를 이용해 이미 딥러닝과 모델 서비스 프로세스를 Apache NiFi* 를 기반으로 구축한 기업 데이터 어셈블리 라인에 임베디드했습니다.

Analytics Zoo를 기반으로 구성한 딥러닝 추천 알고리즘을 검증하기 위해, 마스터 카드는 Spark 머신러닝과 Analytics Zoo의 BigDL 모델을 벤치마크 테스트했습니다. 전자는 Spark MLlib 방식의 ALS(Alternating Least Squares) 모델을 선택했습니다. 딥러닝 모델과 ALS 모델 방식 비교는 그림 3-2-4에 나타나 있습니다.

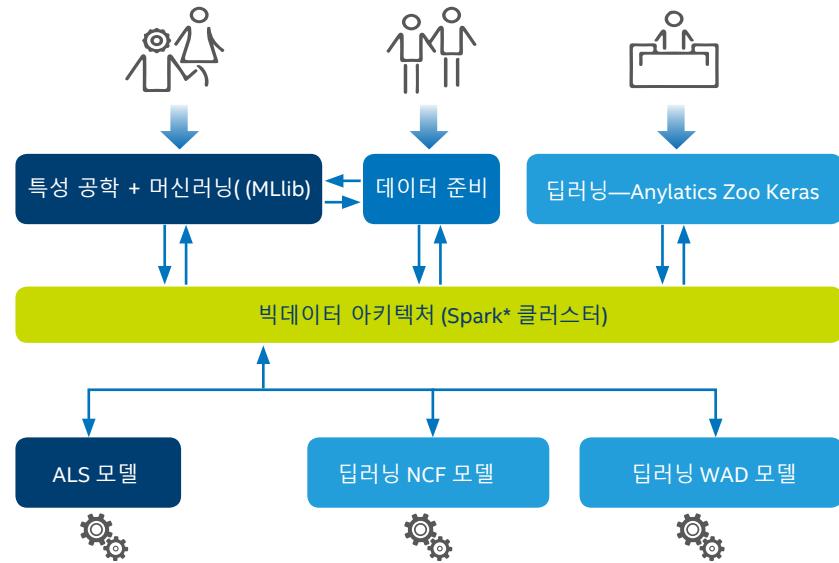


그림 3-2-4 딥러닝 모델과 ALS 모델 비교

■ 방안 구성과 성과

비교 방안은 아래와 같이 과거 3년 동안 마스터 카드가 특정 채널을 통해 수집한 데이터를 채택해 진행했습니다.

- 합격 소비자 수 : 675,000
- 기준이 되는 타깃 상점 (우대 또는 광고 계열) : 2,000
- 알고 있는 거래 : 14 억(최초 데이터 53 GB)
- 소비 시간 : 12-24 개월은 트레이닝, 1-2 개월은 검증 기간

ALS 모델 대비 마스터 카드 추천 시스템의 딥러닝 모델 효과는 주로 아래의 네 가지 지표를 기반으로 합니다.

1. ROC 곡선 면적 (ROC AUG)
2. 정확도와 리콜률 곡선 면적 (PR AUC)
3. 정확도와 리콜률
4. 모든 고객의 상위 20% 정확도

딥러닝 모델이 ALS 모델보다 눈에 띄게 개선되었음을 검증 결과를 통해 알 수 있습니다.

	NCF 模型	WAD 模型
ALS 대비, 리콜률 개선	29% ↑	26% ↑
ALS 대비, 리콜률 개선	18% ↑	21% ↑
ALS 대비, 상위 20% 정확도 증가	14% ↑	16% ↑

표 3-2-1 ALS 모델 대비 딥러닝 모델의 개선 결과

요약

금융은 데이터와 프로세스를 중시하는 전통적인 업계로서 다년 간 다양한 데이터를 축적해 왔으며, 여기에 인공지능 기술을 응용하면 더 다양한 가치를 발굴해 업무 범위를 넓히도록 도울 수 있습니다. 또한 단말 사용자에게 더 다양한 개인 맞춤형 서비스를 제공해 사용자의 경험을 향상시킬 수 있습니다.

Analytics Zoo 가 제공한 end-to-end AI 와 빅데이터 분석 기능 및 다양한 모델과 API 를 이용하면 금융 기업은 자신의 데이터 자원을 빠르게 활용해 Hadoop, Spark 와 같은 기존의 빅데이터 플랫폼에 NCF, WAD 등에 딥러닝 모델 기반 추천 시스템을 구축할 수 있습니다. 그러면 처음부터 새로 시작할 필요가 없기 때문에 금융 기업이 업무 추천 시스템을 구축하는데 드는 비용과 시간을 대폭 절감할 수 있습니다.

차이나 생명보험 상하이 데이터 센터, 마스터 카드 등의 사례에서 솔루션은 모두 인텔® 제온® CPU/인텔® 제온® 확장성 CPU를 기초 하드웨어 플랫폼으로 채택했습니다. 앞으로 사용자는 성능이 더 강하고, 인공지능 영역에 더 최적화된 방식의 2 세대 인텔® 제온® 확장성 CPU 등 혁신 1 세대 하드웨어 제품을 선택해 성능이 더 뛰어나고, AI 트레이닝/추론 기능이 더 강한 솔루션을 구성할 수 있습니다.

AI 영상 분석 능력 가속화, 보험 업계에서의 AI 잠재력 촉진



AI로 보험 업계 영상 분석 능력 가속화

보험 업계에서의 영상 분석

보험 업계는 보험 종류가 무엇이든 영상 분석에 대한 수요가 큽니다. 예를 들어, 자동차보험의 보험 가입과 사고발생은 보험 가입자가 보험 가입 시스템에 신분증과 운전면허증, 자동차등록증 등 증명서를 업로드하면 다시 직원의 심사를 거쳐야 합니다. 자주 사용되는 여러 증명서와 서명날인은 많게는 수십 개까지 달하다 보니 모든 것을 사람이 심사하려면 힘들 뿐만 아니라 오류가 발생하기 쉽습니다. 또한 나날이 관심이 확대되고 있는 건강보험 역시 X-ray, CT 등 영상 자료 판독을 통해 보험 가입자의 최근과 과거 건강 상황을 정확하게 평가해야 합니다.

AI 능력을 강화해 사용자 경험 향상

현재 안면 인식, 영상 분할 등 일련의 영상 분석 기반 AI 응용이 보험 업계에서 점점 더 광범위하게 활용되고 있습니다. AI 영상 분석을 보험 업무 경영과 리스크 관리, AI 기반 고객 서비스 및 내부 컨트롤 등 모든 프로세스에 도입하면 리스크를 효과적으로 포착하고, 업무 프로세스를 최적화하며, 보험 업계의 AI 잠재력을 실현할 수 있습니다. 예를 들어, 위에서 거론한 자동차보험과 건강보험을 처리할 때, AI 영상 분석을 통해 NLP 기술을 결합하면 필요한 보험료 지급 자료를 빠르게 선별해 심사 정보를 자동적으로 추출할 수 있습니다. 그런 다음 보험료 지급 규칙과 리스크 관리 모델을 통해 보험금을 지급함으로써 해당 작업을 자동, 고효율적으로 완료합니다.

보험 업계의 AI 응용 수요에 맞추어 인텔은 얼굴 감지, 비교 대조, 인식, 생체 조직 검사 등 여러 모듈에 맞는 알고리즘과 모델을 제공할 수 있습니다. 예를 들어, 인텔이 출시한 OpenVino™ 툴 키트은 이미 수십 개의 사전 트레이닝된 AI 모델을 제공해 사용자가 처음부터 다시 시작하지 않아도 얼굴 감지 인식 등 AI 응용을 구축할 수 있도록 했습니다.

* OpenVino™ 툴 키트 기술과 관련한 더 자세한 사항은 해당 매뉴얼 기술편 소개 부분을 참조하십시오.

다시 얼굴 감지 기능을 예로 들어 보면, FeatherNet*은 인텔이 화증과기대*와 합작해 안면 인식 금융 거래 사기 방지 응용에 맞춰 연구 개발한 컨볼루션 뉴럴 네트워크 (Convolutional Neural Networks, CNN)입니다. 전통적인 CNN과 비교하면, 컨볼루션 신경망은 두 가지 특징을 지니고 있습니다. 첫째, 스트리밍 모듈 (Streaming Model)이 GAP(Global Average Pooling)를 대체했습니다. GAP는 여러 DNN에서 차원 하락과 과도한 맞춤 방지에 사용할 수 있지만, 지역 가중치 구분 능력이 부족하기 때문에 안면 인식 시나리오에서 정확도가 쉽게 하락됩니다. 하지만 FeatherNet에는 DWConv 레이어를 함유한 스트리밍 모듈을 추가해 GAP를 대체하기 때문에 정확도면에서 대폭 향상되었습니다. 둘째, FeatherNet은 멀티 모달리티 데이터 융합에 맞추어 새로운 융합 분류기를 구축했기 때문에 멀티 모달리티 데이터에서 학습한 모델을 조합하고 직렬 배치해 모델의 정확도를 향상시키는데 쓰입니다.

¹³ FeatherNet 기술과 성능 관련 설명은 <https://arxiv.org/pdf/1904.09290.pdf>를 참조하십시오

ResNet 기반 딥러닝 방식

■ ResNet 소개

DNN는 현재 AI 영상 분석에서 가장 광범위하게 사용되는 네트워크 모델 중 하나입니다. 전형적인 DNN은 네트워크 레이어 수가 많을수록 추출할 수 있는 숨겨진 레이어의 특성이 더 풍부해질 뿐만 아니라, 깊은 네트워크일수록 추출한 특성을 더 추상적이고 다양한 의미 정보를 가질 수 있도록 합니다.

하지만 심도가 증가함에 따라 저하 (Degradation) 문제 역시 발생하기 때문에 정확도가 우선 포화 상태까지 상승한 뒤 심도가 지속적으로 증가하고 정확도가 점차 하락하게 됩니다. 레지듀얼 네트워크 (Residual Net, ResNet)는 이 문제를 효과적으로 해결할 수 있습니다. 그림 4-1-1 에서 보는 바와 같이, ResNet에 여러 잔차 블록 구조를 구성할 수 있으며, 입력과 기대 출력이 대등해 일종의 항등사상 관계를 형성하게 됩니다.

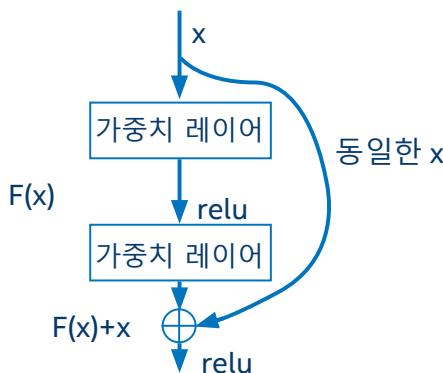


그림 4-1-1 ResNet 잔차 블록 구조

이런 구조는 DNN이 심도가 끊임없이 증가해도 정확도를 유지할 수 있게 해줍니다. 현재 ResNet은 이미 화상 인식 등 AI 응용 시나리오에서 광범위하게 사용되고 있습니다.

■ 모델 실현

인텔® 아키텍처 최적화 지향 Caffe* 는 RESNET50 네트워크에 최적화 버전의 caffe prototxt 파일을 제공하며 그 위치는 다음과 같습니다.

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

dummy 데이터를 사용한 prototxt 파일의 위치 :

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

■ 인텔® 제온® CPU 플랫폼에서 코드 운용 효율 최적화

■ 인텔® 아키텍처 최적화 지향 Caffe 설치

인텔® 아키텍처 최적화 지향 Caffe 1.1.6 설치 방법은 다음과 같습니다.

```
1. git clone -b 1.1.6 --recursive https://github.com/intel/caffe.git intelcafe
2. cd intelcafe
3. ./scripts/prepare_env.sh
4. mkdir build
5. cd build
6. cmake ..
7. build -j
```

인텔® 아키텍처 최적화 지향 Caffe의 python 목록을 Pythonpath의 환경 변수에 추가합니다.

```
1. export PYTHONPATH=../../intelcafe/python:$PYTHONPATH
```

■ 인텔® 아키텍처 최적화 지향 Caffe*의 메모리 최적화¹⁴

기본적인 상황에서 인텔® 아키텍처 최적화 지향 Caffe는 모든 레이어에 단독 출력 버퍼 존을 배치합니다. 하지만 출력 버퍼 존이 로컬 메모리 캐시에 있지 않고 각기 다른 메모리 주소를 사용하기 때문에 전달 중인 여러 메모리 조사가 잠재 캐시를 명중시키지 못하게 됩니다. 그러므로 원형 버퍼 공유 메커니즘, 즉, 숨겨진 층 간 재사용 우선 배치된 메모리 버퍼 존 방법은 인텔® 아키텍처 최적화 지향 Caffe에서 캐시 손실률을 낮추는데 사용할 수 있습니다. 컴파일 단계에서는 그래프 순회를 통해 출력 버퍼 존의 최대 크기를 식별하고, 실행 단계에서는 한 개의 레이어 실행을 완료하면, 해당 레이어의 메모리 버퍼 존이 재사용으로 원형 큐에 릴리스 및 리턴됩니다.

또한 여러 실제 사례에서도 가중치 공유 기술을 이용해 시스템에 더 뛰어난 성능을 제공할 수 있습니다. 가중치 공유 메커니즘은 동일한 NUMA 노드 내 여러 가지 프로세스에서 가중치 버퍼 존을 공유해 CPU 3 레벨 캐시 (L3 Cache) 와 메모리 간의 캐시 명중률을 향상시킵니다.

■ NUMA 특성을 이용해 CPU 컴퓨팅 리소스 사용 컨트롤

데이터 센터는 일반적으로 NUMA 기술을 도입해 여러 서버를 단일 시스템처럼 운영합니다. CPU는 자체 로컬 메모리에 접근하는 속도가 비로컬 메모리에 접근하는 속도보다 빠릅니다. 이러한 시스템에서 더 뛰어난 컴퓨팅 성능을 획득하기 위해서는 일부 특정 명령을 통해 컨트롤해야 합니다. numactl 이 바로 과정을 컨트롤하고 기억을 공유하는 일종의 기술 매커니즘으로, Linux 시스템에서 광범위하게 사용되는 컴퓨팅 리소스 컨트롤 방식입니다. 인텔® 아키텍처 최적화 지향 Caffe는 추론 시 numactl 명령을 사용해 컴퓨팅 효율을 높임으로써 스루풋을 향상시킵니다.

¹⁴ 자세한 기술 설명은 다음을 참조하십시오. <https://arxiv.org/abs/1805.08691>

구체적인 사용 방법은 아래와 같습니다.

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

실행 시에는 CPU#CPU0 중 0-19 와 40-59, CPU#CPU1 에 상응하는 근단 메모리를 사용했고, CPU#CPU1에서도 유사한 명령을 운용할 수 있습니다.

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

또한 수많은 코드가 복잡한 정도를 대폭 향상시키기 어렵기 때문에 임무를 더 작은 CPU 코더에서 운용하면 효율이 더 높아집니다. 그러므로 numactl 방식으로 CPU 코어를 바인딩해 더 많은 실제 사례를 운용하면, 더 높은 스루풋을 얻을 수 있게 됩니다. 랙 시간이 다소 증가하긴 하지만 일반적으로는 수용할 수 있는 범위 내에 있게 됩니다.

3D V-Net 디바이딩 네트워크 기반 딥러닝 방식

V-Net은 end-to-end 트레이닝의 완전한 콘볼루션 네트워크로 3D 영상 데이터를 처리해 영상 분할 업무를 완료합니다. 네트워크 모델은 데이터를 다시 자르지 않고 3D 콘볼루션 네트워크 레이어를 사용해 3D 데이터를 직접 처리합니다. 또한 유사한 계수로 커스텀한 목표 함수로 네트워크 트레이닝을 지도해 트레이닝을 최적화함으로써 속도를 향상시킬 수 있습니다.

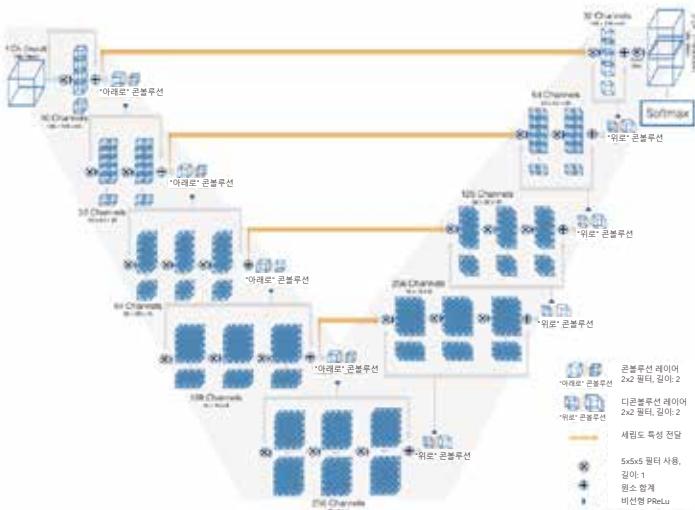


그림 4-1-2 V-Net 콘볼루션 뉴럴 네트워크 설명도

데이터 규모를 $128 \times 128 \times 64$ 크기의 3D 데이터로 제한하고 Batch_Size=1로 설정해 콘볼루션 네트워크의 메모리 로드를 추산합니다. 파라미터를 계산할 때 메모리를 점유할 경우 bias 를 주의해야 하며, 데이터에 필요한 메모리 로드는: $313.7864M \times 4\text{Bytes} = 1255.1456\text{MB} \approx 1.23\text{GB}$ (이곳은 포워드 계산 과정뿐이며, 백워드 계산 과정은 이 메모리의 약 2배가 필요함)입니다.

- 가중치에 필요한 메모리 로드 : $77.44695M \times 4\text{Bytes} = 309.7878\text{MB} \approx 0.303\text{GB}$;
- 전체 네트워크 모델에 필요한 메모리 로드 크기: $1.23 + 0.303 \approx 1.53\text{GB}$;
- 배의 메모리 임시 저장 장치를 추가하려면 : 3.06GB 。

Layer (batch-size=1)	Data (M)	Weight (M)
Input	1.048576	0
Conv1	16.77722	0.002016
	16.77722	0.032016
	16.77722	0.032016
Pool1	4.194304	0.004128
Conv2	4.194304	0.128032
	4.194304	0.128032
	4.194304	0.128032
Pool2	1.048576	0.016448
Conv3	1.048576	0.512064
	1.048576	0.512064
	1.048576	0.512064
Pool3	0.262144	0.065664
Conv4	0.262144	2.048128
	0.262144	2.048128
	0.262144	2.048128
Pool4	0.065536	0.2624
Bottom	0.065536	8.192256
	0.065536	8.192256
	0.065536	8.192256
Deconv4	0.524288	0.524544
	0.524288	8.192256
	0.524288	8.192256
	0.524288	8.192256
	0.524288	8.192256
Deconv3	0.2097152	0.262272
	0.2097152	2.048128
	0.2097152	2.048128
	0.2097152	2.048128
	0.2097152	2.048128
Deconv2	0.1101005	0.0656
	0.1101005	0.512064
	0.1101005	0.512064
	0.1101005	0.512064
	0.1101005	0.512064
Deconv1	0.3355443	0.016416
	0.3355443	0.128032
	0.3355443	0.128032
	0.3355443	0.128032
	0.3355443	0.128032
Softmax	2.097152	0.000066
	2.097152	0
합계	313.7864	77.44695

인텔® 아키텍처를 통한 성능 향상

■ 인텔® 제온® 확장성 CPU의 AI 강화 특성

혁신적인 마이크로아키텍처를 보유한 차세대 인텔® 제온® 확장성 CPU는 커널이 더 다양하고, 스레드 병렬 배포도가 더 높으며 고속 캐시가 더 왕성한 특징이 있습니다. 또한 다양한 하드웨어 강화 기술을 보유하고 있는데, 특히 인텔® AVX-512 등 기술은 AI 추론 과정에서 강력한 병행 컴퓨팅 기능을 제공해 사용자의 딥러닝 효과를 더 좋게 만들 수 있습니다.

이 뿐만 아니라 인텔® 아키텍처 CPU는 BVLC Caffe, TensorFlow, Apache MXNet* 등과 같은 다양한 AI 아키텍처에 맞추어 다량의 최적화 작업을 진행합니다. 인텔® 아키텍처 최적화 지향 Caffe를 예로 들면, BVLC Caffe와 비교했을 때 인텔® 제온® 확장성 CPU의 장점을 릴리즈 15 해 1+1>2 효과를 실현할 수 있습니다.

■ 인텔® 아키텍처 최적화 지향 Caffe 방식과 코드

인텔® 아키텍처 최적화 지향 Caffe는 레이어 내부에서 인텔® MKL-DNN의 API를 호출해 최적화된 명령어 조합을 호출해 프로세스의 명령 병렬화 효과를 대폭 향상시킵니다. 인텔® MKL-DNN은 인텔® 제온® 확장성 CPU에 내장한 인텔® AVX-512 명령어 조합 및 2세대 인텔® 제온® 확장성 CPU 내장 딥러닝 가속 기술 (VNNI 명령어 조합)을 자동 호출합니다.

* 2세대 인텔® 제온® 확장성 CPU 및 VNNI 명령어 조합 기술과 관련한 더 자세한 사항은 해당 매뉴얼 기술편 소개 부분을 참조하십시오.

레이어 융합

BN+Scale, Conv+Sum, Conv+Relu, BN InPlace와 Sparse Fusion 등 레이어 융합은 딥러닝 성능을 향상시키는데 쓰입니다. 레이어 융합 기술을 인텔® 아키텍처 최적화 지향 Caffe 아키텍처와 융합하면 ResNet 등 콘볼루션 뉴럴 네트워크로 인텔® 제온® 확장성 CPU 플랫폼에서 2D 도형 추론을 진행할 때, 성능이 기존의 플랫폼 수준을 넘어설 정도로 뛰어나집니다. 또한 VNNI 명령어 조합에서 최적화 지원 INT8 정밀도를 얻어 지원합니다. 아키텍처가 제공한 calibration* 등의 툴은 사용자가 뉴럴 네트워크를 빈틈 없이 INT8로 전환하도록 도와 성능을 대폭 향상시킬 수 있습니다.

일부 데이터를 통해 알 수 있듯이 BVLC Caffe를 사용한 경우와 비교했을 때, 인텔® 아키텍처 최적화 지향 Caffe는 인텔® 제온® 확장성 CPU에서 운용함과 동시에 레이어 융합 기술을 추가했습니다. 또한 ResNet50 콘볼루션 뉴럴 네트워크를 사용해 동등한 테스트 환경에서 AI 추론을 실행하면, 그림 4-1-3과 같이 단위 시간의 추론 성능을 전자의 51배 이상 향상시킬 수 있으며, 추론 시간은 전자의 4.7%¹⁶까지 단축됩니다.

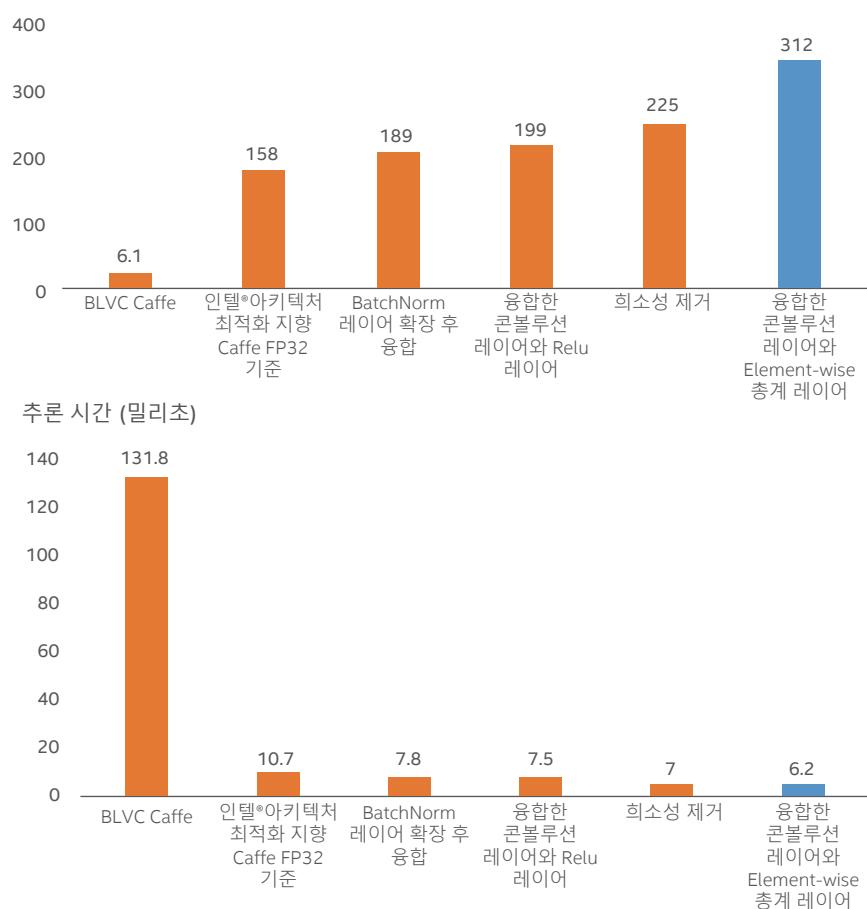


그림 4-1-3 인텔® 아키텍처 최적화 지향 Caffe를 인텔® 제온® 확장성 CPU에서 최적화 방안을 추가한 이후 추론 슬루프과 추론 시간 및 성능 BLVC Caffe와 비교

* 인텔® 아키텍처 최적화 지향 Caffe 기술과 관련한 더 자세한 사항은 해당 매뉴얼 기술편 소개 부분을 참조하십시오.

¹⁵ 인텔® 아키텍처 최적화 지향 Caffe 홈페이지: <https://github.com/intel/caffe>

¹⁶ 이 데이터의 출처는 《Highly Efficient 8-bit Low Precision Inference of Convolutional Neural Networks with IntelCaffe》에서 추출한 것입니다. <https://arxiv.org/pdf/1805.08691.pdf>, 테스트 구성은 다음과 같습니다. 콘볼루션 모델: ResNet50, 하드웨어: AWS single-socket c5.18xlarge

소프트웨어/하드웨어 구성 제안

이상은 AI 영상 분석 모델 기반 구성 방법이며, 아래의 인텔® 아키텍처 기반 플랫폼을 참조할 수 있습니다. 환경 구성은 다음과 같습니다.

하드웨어 구성

名稱	規格
노드 수	1
Socket	2
CPU	인텔® 제온® 골드 6148 CPU 또는 그 이상
기저 주파수	2.40Ghz
코어/스레드	20/40
HT	On
Turbo	On
메모리	12 slots/192G/2666
BIOS	1.46

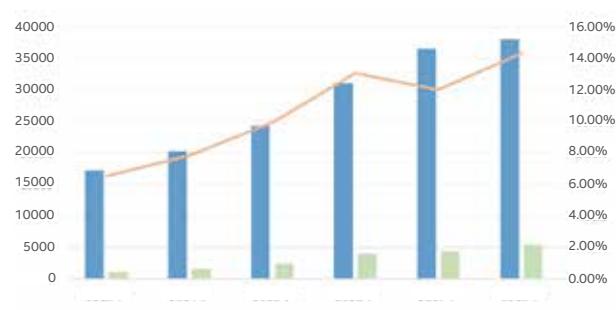
하드웨어 구성

名稱	規格
운영 체제	Ubuntu 16.04
Linux 커널	3.10.0-693.21.1.el7.x86_64
워크로드	Resnet50/VNet
기저 주파수	Gcc 4.8.5
딥러닝 가속 라이브러리	인텔® MKL-DNN 최신 버전
딥러닝 아키텍처	인텔® 아키텍처 최적화 지향 Caffe 배포 버전

중국 평안보험 응용 사례

배경

건강보험은 상업보험 중 중요한 종류 중 하나입니다. 사람들의 의료를 향한 관심이 점차 높아짐에 따라 건강보험의 보험료와 종류, 보험 가입 범위 등 역시 점차 확대되고 있습니다. 중국보감회의 통계 데이터에 따르면, 2018년 말까지 건강보험 관련 보험료 수입이 5448.13 억 위안 달성, 동기 대비 24.12% 증가함으로써 전체 기준 보험료 수입의 14.33%¹⁷ 차지했다고 합니다.



이미 어느 정도 성숙한 보험 시장과 비교했을 때, 이 숫자는 더 크게 성장할 여지를 지니고 있습니다. 미국의 경우, 건강보험이 보험총액에서 차지하는 비율이 40%¹⁸나 된다는 것만 봐도, 중국의 보험 시장에서 건강보험이 성장할 잠재력은 매우 크다는 걸 예상할 수 있습니다. 현재 건강보험이 매우 빠르게 발전하고 있는 것은 사실이지만 제약이 없는 것은 아닙니다.

다른 종류의 보험과 비교했을 때, 건강보험은 피보험자의 건강을 소재로 한 상품이다 보니, 보험 회사는 피보험자의 질병과 상해 상황을 정확하고 명백하게 평가해 건강보험 경영 리스크를 줄여 보험자금을 제어해야 하는 어려움이 있습니다. 게다가 이 업무의 기술 난이도와 관리 난이도가 다른 종류의 보험보다 복잡해서 담당자는 매우 전문적인 지식과 실전 경험을 보유해야 하는 실정입니다.

의료 영상은 의료 기관이 가장 자주 사용하는 의료 근거이자 보험 기관이 피보험자의 건강 상황을 판단하는 중요한 근거이기도 합니다. 의료 영상 서비스는 이미 다양한 형태의 의원에서 광범위하게 사용되고 있지만, 의료 영상을 정확하게 판독하는 문제는 새로운 도전에 직면한 상황입니다. 담당 의사들은 임상의학, 의료영상학 등 다방면의 전문 지식을 갖추어야 할 뿐만 아니라 방사선학, CT, 핵자기공명, 초음파 공학 등 관련 기술을 마스터함과 동시에 여러 영상 진단 기술을 운용해 질병을 판단해야 합니다. 결국 경험이 풍부한 영상의학전문의만 정확히 판독할 수 있는 능력을 지니게 되는 것입니다.

¹⁷ 이 데이터의 출처는 중국보감회 홈페이지입니다. <http://bxjg.circ.gov.cn/web/site0/tab5257/info4132154.htm>

¹⁸ 이 데이터의 출처는 한 대중 매체의 보도입니다. <http://www.chinairn.com/news/20150211/140425992.shtml>

이제는 앞서가는 AI 기술을 이용해 의료 영상 판독을 진행할 수 있게 되었습니다. 영상을 분할하고 조정할 수 있을 뿐만 아니라 영상을 심층 분석해 육안으로는 발견하기 어려운 미세한 병리적 특성을 발견해 유형별 악성 질환 조기 발견 확률을 향상시킬 수 있습니다. AI 기반 2D/3D 의료 영상 트레이닝과 추론은 여러 의료기관이 의료 효율을 향상시킬 수 있도록 도울 뿐만 아니라 보험 기관이 더 정확하게 건강 평가를 진행할 수 있도록 효율적인 방법을 제공합니다.

솔루션

머신러닝 또는 딥러닝 개념 중, 트레이닝해 획득한 AI 모델이 새로운 데이터에 응용되는 과정을 우리는 추론이라 부릅니다. 이 트레이닝으로 얻은 모델은 의료 영상 판독 작업에서 병리 특징을 추론 판단하는데 사용되고, 그렇기 때문에 추론 효율은 의료 영상 판단 효율 정도에 따라 좌지우지 됩니다.

이번 사례에서는 중국평안보험의 앞에서 소개한 2D 영상 분류와 감지, 조정 측면에서 특히 우수한 ResNET과 최첨단 3D 영상 분할 모델 V-Net 분할 네트워크와 인텔® 아키텍처 최적화 지향 Caffe 등 딥러닝 아키텍처를 이용해 2D/3D 의료 영상에 대해 AI 추론을 진행합니다.

그림 4-2-2 처럼 중국평안보험 의료 영상 응용 추론 과정은 주로 네 개 단계로 나뉩니다. 전처리 단계에서는 시스템이 의료 영상 분할, ROI 지역 선택, 데이터 강화, 정규화 입력 준비 등 작업을 진행합니다. 추론 엔진에서는 시스템이 ResNet 네트워크와 3D V-Net 분할 네트워크를 이용해 인텔® 아키텍처 최적화 지향 Caffe 아키텍처에서 입력한 의료 영상을 추론하고 처리 결과를 얻습니다. 후처리 단계에서는 시스템이 수요에 따라 다양한 형태학을 진행하고 예측 결과 통합 등 작업을 실행합니다. 마지막 애플리케이션 층 처리 단계에서는 시스템이 XML 등 방식으로 대외 출력하고 결과를 표시할 수 있습니다.



그림 4-2-2 중국평안보험 의료 영상 AI 추론 프로세스

중국평안보험 AI 팀은 ResNet 네트워크와 3D V-Net 분할 네트워크를 이용해 고효율의 AI 추론 작업을 진행해 만족스러운 성과를 거두었습니다. 그림 4-2-3처럼, 첨단 기술을 사용해 노년의 황반변성 등 안과 질병의 광 간섭성 단층촬영기술 (Optical Coherence tomography, OCT) 영상을 병변 분할한 뒤, 결과를 통해 환자의 망막 내 수증 (노란선 부분)을 확실히 발견할 수 있었습니다. 망막 아래 수증 (빨간선 부분)과 망막 아래 HR (보라색 부분)은 모두 스마트앱에 의해 뚜렷하게 드러났습니다. 경험이 풍부한 의사가 힘을 들여야 완료할 수 있던 과거와 달리 이제는 AI 앱을 이용하면 몇 초 만에 완료할 수 있게 되었습니다.

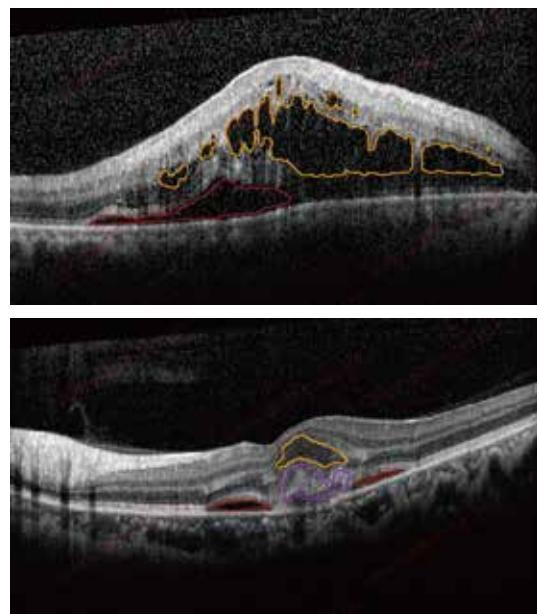


그림 4-2-3 스마트앱으로 실현한 OCT 병변 분할 결과

실험실과 임상 반복 트레이닝 및 추론을 통해 중국평안보험 스마트 의료 영상 분석은 이미 여러 시나리오에서 자랑스러운 업적을 달성했습니다. 예를 들어, 사르코이드증 검사 부분에서는 폐암 사망률을 초기에 낮출 수 있는 훌륭한 수단이 되었습니다. 저선량 CT 검사를 통하여 고위험군을 살살이 조사해 일찌감치 위험 요소를 찾아낼 수는 있지만 방대한 양의 CT 영상 자료 때문에 선별 효율은 대대적으로 낮아집니다. 그런데 사르코이드증 검사에 AI를 이용하면 이러한 문제를 해결하고 효율을 높일 수 있습니다. 2018년 초 사르코이드증 분석 (LUng Nodule Analysis, LUNA) 평가에서 중국평안보험은 "중국평안보험 사르코이드증 스마트 판독 기술"로 우승을 거머쥐었으며, 정밀도 95.1% 와 96.8%로 "사르코이드증 검사"와 "위양성 검사" 부분에서 세계적인 기록19을 남겼습니다. 의료 영상 분석은 조사한 보험의 사기 여부도 분석할 수 있기 때문에 중국평안보험의 금융 거래 사기 방지 능력 역시 대대적으로 향상시켰습니다.

¹⁹ 이 데이터의 출처는 Luna 홈페이지입니다.<https://luna16.grand-challenge.org/Results/>

요약

AI 기반 영상 분석은 금융 기관이 업무 처리 효율을 높이고, 사기 리스크를 막을 뿐만 아니라 사용자 경험을 개선시켜 줍니다. Caffe, TensorFlow 등 딥러닝 아키텍처는 보험 업계의 스마트 인수심사 프로세스 중 병리 영상 판독, 영수증 처리 등 시나리오에서 광범위하게 사용되고 있습니다.

인텔® 제온® 확장성 CPU 와 인텔® 아키텍처에 맞춰 최적화된 딥러닝 아키텍처를 이 스마트앱에 도입하면 스마트앱의 주론 효율을 효과적으로 높일 수 있을 뿐만 아니라 앱의 랜딩 능력과 배치 가능성을 효과적으로 강화하고, 보험 업계 내 AI 응용을 가속화합니다.

기존의 사례에 대한 솔루션은 모두 인텔® 제온® CPU/인텔® 제온® 확장성 CPU 플랫폼을 채택했습니다. 더 나은 영상 분석 효과를 얻기 위해, 사용자는 더 강한 성능에 AI 영역에서 더 다양한 최적화 방식을 지닌 2 세대 인텔® 제온® 확장성 CPU 를 솔루션으로 선택하길 권장합니다.



A dark blue background featuring a complex network graph composed of numerous small, glowing teal and white circular nodes connected by thin, light-colored lines.

기술편

2 세대 인텔® 제온® 확장성 CPU



2 세대 인텔® 제온® 확장성 CPU는 데이터 센터 현대화를 위해 전문 설계되었습니다. 이 전 제품 보다 성능¹이 2.5% - 3.5% 높고, 다양한 옵션과 새로운 특성을 갖추었습니다. 유연성과 안전성을 향상시키고 메모리 성능을 강화해 사용자가 인프라, 기업 앱과 기술 컴퓨팅 운용 효율을 향상시킬 수 있도록 도와 성능이 더 강력하고 서비스가 더 민첩해 가치가 높은 기능을 형성합니다. 더 나아가 총소유비용 (Total Cost of Ownership, TCO)을 개선해 생산력을 높입니다.

인텔® 제온® 골드 CPU 6200 시리즈, 특히 주류의 인텔® 제온® 골드 6248 CPU, 인텔® 제온® 골드 6240 CPU, 인텔® 제온® 골드 6230 CPU는 인텔® 제온® 확장성 CPU 플랫폼의 기동으로서 더 빠른 메모리와 더 강한 메모리 용량과 포웨이 확장 가능성을 지원합니다. 성능과 고급 신뢰성, 하드웨어 증강형 보안 기술 측면에서도 눈에 띄게 개선되었으며, 조건이 까다로운 주류 데이터 센터, 클라우드, 네트워크와 스토리지 등 워크로드에 최적화를 진행했기 때문에 더 복잡하고 다양화된 응용 시나리오에 적응할 수 있게 되었습니다. 또한, 인텔® 제온® 골드 6200 시리즈는 듀얼 FMA 채널을 최초로 지원하며, 이는 FMA 성능이 2 배로 향상되었음을 의미합니다.

2 세대 인텔® 제온® 확장성 CPU는 딥러닝 가속 기술 (VNNI)을 집성하고 인텔® AVX-512를 확장해 플랫폼에 더 다양하고 더 강한 AI 기능을 부여함으로써 인공지능과 딥러닝 추론을 가속화하고 워크로드를 최적화할 수 있었고, 그 결과 AI 가속 능력을 보유한 CPU 아키텍처가 되었습니다. 이 아키텍처를 기반으로

대다수의 추론 업무가 워크로드 또는 애플리케이션에 집성되어 있기 때문에 사용자는 가속화된 성능과 높아진 유연성 등의 장점을 누림으로써 데이터가 중심인 현대 사회에서 클라우드와 AI 엣지 사이에서 성능 전환과 AI 개발, 응용을 방해 받지 않고 고효율적으로 진행할 수 있습니다.

차세대 제온 확장성 플랫폼의 "핵심"으로서 2 세대 인텔® 제온® 확장성 CPU는 인텔® 옵테인™ 데이터 센터급 영구 메모리라는 새로운 제품을 지원합니다. 하지만 인텔® 옵테인™ 데이터 센터급 영구 메모리는 2 세대 인텔® 제온® 골드 및 실버 CPU와 결합해 DRAM 메모리의 강력한 보완 조치로 시스템 성능, 가속 워크로드 처리와 서비스 전달(구체적인 내용은 《인텔® 옵테인™ 데이터 센터급 영구 메모리》부분 참조)을 뚜렷하게 향상시킬 수 있습니다.

기능 특성 :

- 더 높아진 커널별 성능 : 많게는 56 커널 (9200 시리즈)과 28 커널 (8200 시리즈)에 다다르며, 컴퓨팅과 스토리지, 네트워크 응용에서 컴퓨팅 밀집형 워크로드에 더 높은 성능과 확장성을 제공합니다.
- VNNI 기반 인텔® 딥러닝 가속 (인텔® DL Boost) 기술: CPU에서 인공지능 추론의 표현을 향상시켰습니다. 이전 세대 제품과 비교하면 성능이 30 배까지 향상되면서 데이터 센터에서 엣지까지 AI를 배치 및 응용하는데 도움이 됩니다.
- 업계를 선도하는 메모리와 스토리지, 더 커진 메모리 대역폭 / 용량 :

인텔® 옵테인™ 데이터 센터급 영구 메모리를 지원하며, 전통적인 DRAM과 결합해 사용하면 최고 36 TB 시스템급 메모리 용량까지 지원할 수 있습니다. 메모리 대역폭과 용량이 50%² 향상되어, 각각 메모리 채널 6 개와 최대 4TB DDR4 메모리, 최고 2933 MT/s (1 DPC) 속도를 지원합니다. 그 밖에도 인텔® 옵테인™ 데이터 센터급 SSD 와 인텔® QLC 3D NAND SSD를 지원해 데이터 밀집형 워크로드와 획기적인 메모리와 스토리지 메모리 혁신 효율과 성능을 두드러지게 향상시켰습니다.

- 인텔® Infrastructure Management 기술 (인텔® IMT) : 이 자원 관리 아키텍처는 인텔의 여러 능력을 결합해 플랫폼급 탐지와 보고, 구성을 효과적으로 지원할 수 있습니다.
- 데이터 센터 지향의 인텔® Security Libraries (인텔® SecL-DC) : 이 소프트웨어 라이브러리와 모듈은 인텔 하드웨어를 기반으로 한 보안 기능입니다.

제온® 플랫폼의 혁신작으로서, 2 세대 인텔® 제온® 확장성 CPU는 획기적인 설계를 기반으로 플랫폼에서 컴퓨팅, 메모리, 스토리지, 네트워크 및 보안 등 기능을 융합하면서 새로운 수준으로 올라설 수 있었습니다.

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

스마트앱에 적용되는 2 세대 인텔® 제온® 확장성 CPU

* 특정 CPU에서만 지원을 받습니다.

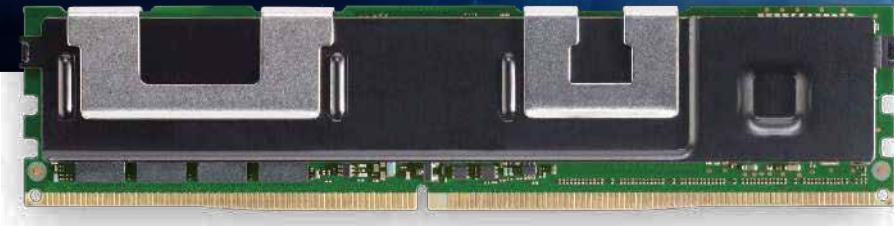
	인텔® 제온® 골드 CPU (6200 시리즈)	인텔® 제온® 골드 CPU (6200 시리즈)			인텔® 제온® 골드 CPU (8200 시리즈)	인텔® 제온® 골드 CPU (9200 시리즈)
		인텔® 제온® 골드 6230 CPU	인텔® 제온® 골드 6240 CPU	인텔® 제온® 골드 6248 CPU		
유비쿼터스의 성능과 안전성						
지원하는 최대 커널 수	24	20	18	20	28	56
지원하는 최고 주파수	4.4 GHz	3.90 GHz	3.90 GHz		4.0 GHz	3.8 GHz
지원하는 CPU 수	최대 4 개				최대 8 개	최대 2 개
인텔® Ultra Path Interconnect (UPI)	3	3	3	3	3	4
인텔® UPI Speed	10.4 GT/s				10.4 GT/s	10.4 GT/s
인텔® 고급 벡터 확장 512 (인텔® AVX-512)	2 FMA	2 FMA	2 FMA	2 FMA	2 FMA	2 FMA
지원한 최고 메모리 속도 (DDR4)	2933 MT/s	2933 MT/s	2933 MT/s	2933 MT/s	2933 MT/s	2933 MT/s
지원하는 최고 메모리 용량*	1 TB、2 TB、4.5 TB	1 TB	1 TB		1 TB、2 TB、4.5 TB	3.0 TB
16 Gb DDR4 DIMM 지원	●	●	●	●	●	●
VNNI 를 채택한 인텔® 딥러닝 가속 (인텔® DL Boost)	●	●	●	●	●	●
인텔® 옵테인™ 데이터 센터급 영구 메모리 모듈 지원*	●	●	●	●	●	
인텔® Omni-Path 아키텍처 (독립식 PCIe* 카드)	●	●	●	●	●	●
인텔® QuickAssist 기술 (칩셋에 집성)	●	●	●	●	●	
인텔® QuickAssist 기술 (독립식 PCIe* 카드)	●	●	●	●	●	●
인텔® 옵테인™ 데이터 센터급 SSD	●	●	●	●	●	●
인텔® SSD 데이터 센터 제품군 (3D NAND)	●	●	●	●	●	●
PCIe 3.0	●	●	●	●	●	●
인텔® QuickData 기술 (CBDMA)	●	●	●	●	●	●
NTB	●	●	●	●	●	●
인텔® 터보 가속 2.0	●	●	●	●	●	●
인텔® 하이퍼스레딩 기술 (인텔® HT 기술)	●	●	●	●	●	●
노드 제어 장치 지원	●	●	●	●	●	●

* 특정 CPU에서만 지원을 받습니다.

2 세대 인텔® 제온® 확장성 CPU 정보는 다음을 참조하십시오.

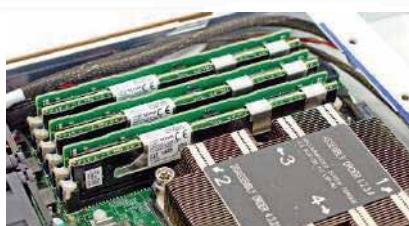
<https://www.intel.cn/content/www/cn/zh/products/processors/xeon/scalable.html>

인텔® 옵테인™ 데이터 센터급 영구 메모리



인텔® 옵테인™ 데이터 센터급 영구 메모리는 인텔의 혁명적인 상품입니다. 새로운 스토리지 라이너를 만들어 메모리와 스토리지 간의 성능 차이를 메워 전통적인 메모리-스토리지 아키텍처를 뒤엎고, 합리적인 가격으로 영구적인 대형 메모리 레이어를 제공하기 때문에 메모리 밀집형 워크로드, 가상 머신 밀도와 라피드 스토리지에 더 뛰어난 성능과 효율, 경제성을 가져다 줄 수 있습니다. 더 나아가 사용자가 이전보다 빨라진 분석과 클라우드 서비스, 인공지능 트레이닝과 추론 및 차세대 통신 서비스를 통해 IT 변화를 가속화함으로써 데이터 시대의 수요를 만족시킬 수 있습니다.

인텔® 옵테인™ 데이터 센터급 영구 메모리는 비용과 비휘발성, 성능 등 특성을 고루 고려하기 때문에 단일 모듈은 128/256/512 GiB 세가지 중 선택할 수 있으며, DDR4 슬롯과는 호환할 수 있습니다. 전통적인 DDR4 DRAM 메모리와 함께 2세대 인텔® 제온® 확장성 CPU 기반 플랫폼에 응용하면, 적은 비용으로 Eight way 시스템에서 최고 24TiB 용량(way당 최고 구성 가능 용량이 3TiB인 옵테인 데이터 센터급 영구 메모리)을 실현할 수 있습니다. 더 나아가 사용자가 CPU의 메모리 시스템에서 규모가 이전을 훨씬 초과하는 데이터 세트를 처리하도록 주기 때문에 메모리 데이터 세트 분석, AI 추론과 반복 등 대형 메모리에 조건이 까다로운 앱 로딩 수요를 만족시킴으로써 더 많은 데이터 처리와 분석을 실시간 처리할 수 있게 됩니다.



메모리와 스토리지 특성을 기반으로 융합한 옵테인™ 데이터 센터급 영구 메모리는: 메모리 모드와 App Direct 모드, 두 가지 업무 모드가 있습니다. 서로 다른 이 두 가지 업무 모드를 통해 고객은 여러 개의 워크로드를 유연하게 넘나들 수 있기 때문에 시스템 성능이 현저히 향상됩니다.

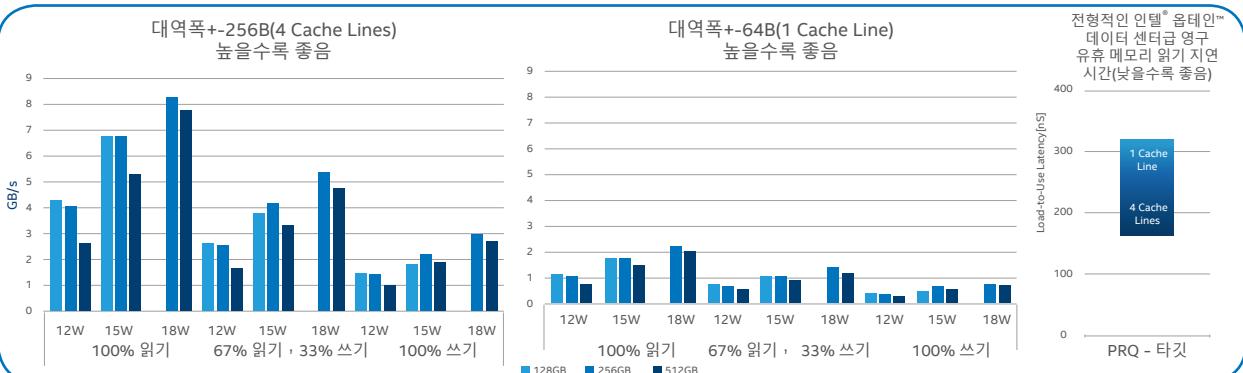
메모리 모드는 대용량 메모리에 매우 적합하며, 애플리케이션을 변경할 필요가 없습니다. 메모리 모드에서는 CPU 메모리 제어 장치가 인텔® 옵테인™ 데이터 센터급 영구 메모리를 주소지정방식의 주기억장치로 삼아 운영 체제가 지원하는 사용 가능 휘발성 메모리 용량을 확장하고, DRAM을 캐시로 삼습니다. 이는 가상화와 컨테이너 기술이 직접적인 이익을 얻을 수 있게 해주며, 저렴한 비용으로 단일 물리적 서버에서 가상 컴퓨터나 컨테이너의 밀도를 향상시키거나 모든 가상 컴퓨터와 컨테이너에 더 큰 용량의 메모리를 제공할 수 있기 때문에 소프트웨어를 새로 프로그래밍할 필요가 없습니다.

App Direct 모드에서는 운영 체제가 DRAM 메모리와 인텔® 옵테인™ 데이터 센터급 영구 메모리를 두 개의 독립적인 메모리

풀로 간주하기 때문에 인텔® 옵테인™ 데이터 센터급 영구 메모리가 메모리처럼 주소를 지정하고 스토리지처럼 영구적인 데이터를 보유할 수 있도록 해줍니다. 이러한 데이터의 영구적 특성은 시스템이 유지 또는 재부팅 기간일 때에도 영구 메모리가 데이터를 유지함으로써 시스템의 업무 탄력성을 증가시키고, 재부팅 시간을 단축하며, 그로 인한 비용을 절감할 수 있습니다.

이중 모드는 App Direct의 부분 집합으로 프로비저닝을 통해 인텔® 옵테인™ 데이터 센터급 영구 메모리 부분을 메모리 모드에, 나머지 부분은 App Direct 모드에 놓이게 할 뿐만 아니라 이중 조건을 갖춘 워크로드 또는 응용 시나리오의 수요를 만족시킵니다.

인텔® 옵테인™ 데이터 센터급 영구 메모리는 다른 인텔® 제온® 확장성 CPU 플랫폼 상품과 함께 시장에 출시되었으며, 2세대 인텔® 제온® 확장성 CPU를 최적화했습니다. 메모리만의 독특한 장점 덕분에 출시 초기부터 관심을 끌었고, 수많은 ISV 파트너들이 관련 소프트웨어를 최적화하게 만들었을 뿐만 아니라. 클라우드 인프라 및 데이터 분석 사용자를 보유하고, 인텔® 옵테인™ 데이터 센터급 영구 메모리를 도입하게 만들었습니다. 협력 파트너와 사용자는 앱 투팅과 앱 실전 과정에서 인텔® 옵테인™ 데이터 센터급 영구 메모리의 응용 가치를 일찍이 증명했습니다.



제품군		인텔® 옵테인™ 데이터 센터급 영구 메모리										
외형 사양		영구 메모리 모듈 (PMM)										
DCPMM SKU ¹	128 GiB	256 GiB			512 GiB							
사용자 용량	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						
상품 코드	NMA1XXD128GPSU4	NMA1XXD128GPSUF	NMA1XXD256GPSU4	NMA1XXD256GPSUF	NMA1XXD512GPSU4	NMA1XXD512GPSUF						
모델 문자열	NMA1XXD128GPS		NMA1XXD256GPS		NMA1XXD512GPS							
기술	인텔® 옵테인™ 기술											
수리 기간	5 년											
연평균 고장률 (AFR)	≤ 0.44											
내구성 100% 쓰기 15W 256B	292 PBW	363 PBW			300 PBW							
내구성 100% 쓰기 15W 256B	91 PBW	91 PBW			75 PBW							
내구성 100% 쓰기 15W 64B	6.8 GB/s	6.8 GB/s			5.3 GB/s							
내구성 100% 쓰기 15W 256B	1.85 GB/s	2.3 GB/s			1.89 GB/s							
내구성 100% 쓰기 15W 64B	1.7 GB/s	1.75 GB/s			1.4 GB/s							
내구성 100% 쓰기 15W 64B	0.45 GB/s	0.58 GB/s			0.47 GB/s							
DDR 주파수	2666、2400、2133、1866 MT/s											
최대 열 설계 전력 (TDP)	15W	18W										
온도 (TJMAX)	≤ 84° C (85°OFF, 83°C 기본) 중간 온도											
온도 (주변 온도)	10W: 54° C @ 2.4m/s											
온도 (주변 온도)	12W: 49° C @ 2.4m/s											
온도 (주변 온도)	15W: 44° C @ 2.7m/s											
온도 (주변 온도)	N/A	18W: 40° C @ 3.7m/s										

주석 : ¹GiB = 2³⁰; GB = 10⁹

더 많은 정보는 다음을 참조하십시오.

<https://www.intel.cn/content/www/cn/zh/products/memory-storage/optane-dc-persistent-memory.html>

인텔® 옵테인™ SSD와 인텔® QLC 3D NAND 기술 기반 인텔® SSD



인텔® 옵테인™ SSD 와 인텔® QLC 3D NAND 기술을 채택한 인텔® SSD 는 혁신적인 스토리지 아키텍처로 데이터 센터가 미래로 향하고 혁신과 혁신을 가속화하도록 돕습니다.

SSD 상품 중 고급 라인에 속하는 인텔® 옵테인™ SSD는 혁신적 인 3D XPoint™ 스토리지 매개체를 채택하고 일련의 첨단 시스템 메모리 제어 장치와 인터페이스 하드웨어, 소프트웨어 기술을 결합해 낮은 지연과 높은 안전성 부분에서 만족스러운 결과를 보임으로써 데이터 센터 스토리지의 병목 현상을 제거할 수 있습니다. 더 크고, 경제적이며 실용적인 데이터 세트를 사용하기 때문에 애플리케이션 속도를 가속화하고, 랙 민감형 워크로드 사무 처리 비용을 절감하며 데이터 센터의 TCO를 개선할 수 있습니다. 인텔® 옵테인™ SSD는 더 전면적이고 우수하며 균형적인 IT 인프라 기능을 지닌 덕분에 데이터 밀집형 AI 모델 트레이닝과 주론 효율 역시 더 높아졌습니다. 인텔® 옵테인™ SSD DC P4800X 를 예로 들면, 최고 55 만 IOPS 의 랜덤 읽기쓰기 기능을 갖추었으며, 읽기쓰기 랙 시간이 10 마이크로초 밖에 되지 않기 때문에 사용자가 많고 여리 번 교부해야 하는 시나리오에서 더 뛰어난 성능을 보일 수 있게 도와줍니다. 또

DWPD (Drive Writes Per Day) 가 높게는 NAND SSD 를 한참 초과하는 수치인 60 까지 달하기 때문에 스토리지 시스템에 더 오랜 생명 주기를 부여할 수 있어 경제적으로도 뛰어납니다.

인텔® SSD는 획기적인 의미를 지니고, 신뢰할 수 있는 3D NAND 기술을 채택해 스토리지 경제성을 향상시키고, 더 나아가 전통적인 하드 디스크(HDD)를 대체하기 때문에 높은 가성비를 선택할 수 있게 합니다. 그 덕분에 사용자는 경험을 개선하고 응용과 서비스 성능을 향상시키며 TCO 를 절감할 수 있습니다. SSD 의 신예 부대라 할 수 있는 인텔® SSD D5-P4320 시리즈는 인텔의 첨단 64 레이어 3D NAND 기술을 채택해 QLC SSD 단일 디스크 용량이 7.68TB (TeraByte)에 달하기 때문에 데이터 센터 등 인프라 사용자의 "대용량" 스토리지에 대한 수요를 만족시킬 수 있습니다. 게다가 랜덤으로 읽은 IOPS가 42.7 만에 달하며, 2세대 인텔® 제온® 확장성 CPU와 매칭을 통해 AI 트레이닝 등 응용 시나리오에 적용되어 "한 번 기록 가능 여러 번 판독 가능" 성능에 대한 수요와 복잡하고 다양화된 워크로드에 고효율, 고안전성을 지닌 저에너지 스토리지 아키텍처를 제공했습니다.

더 많은 정보는 다음을 참조하십시오.

- <https://www.intel.cn/content/www/cn/zh/products/memory-storage/optanememory/optane-memory-h10-solidstate-storage.html>
- <https://www.intel.cn/content/www/cn/zh/products/memory-storage/solidstate-drives/data-center-ssds.html>

適用於深度神經網路的 Intel® 數學核心函數庫

DNN 지향 인텔® MKL(Intel® Math Kernel Library, 인텔® MKL-DNN)은 딥러닝 응용 기반 오픈소스 인핸스먼트 라이브러리이다, 인텔이 개발자가 인텔® 아키텍처를 충분히 이용해 딥러닝 연구와 응용을 추진할 수 있도록 만든 기본 라이브러리입니다. ([소스코드주소 : https://github.com/intel/mkl-dnn](https://github.com/intel/mkl-dnn))

인텔® MKL-DNN은 인텔® 아키텍처에서 딥러닝 아키텍처의 운용 속도를 가속화하기 위해 전문 설계된 하나의 성능 인핸스먼트 라이브러리로서, 고도의 벡터화와 스레드화의 구성 모듈을 포함하고 있으며 C와 C++ 인터페이스 이용 DNN을 지원하고 광범위한 딥러닝 연구와 개발, 응용 생태계를 보유하고 있습니다. Caffe, TensorFlow, PyTorch Apache, Mxnet, BigDL, CNTK, OpenVINO™ 툴킷 등 여러 딥러닝 소프트웨어 상품에 적용됩니다.



인텔® 아키텍처에서 딥러닝 모델이 운용되는 속도를 효과적으로 향상시키고, 유형별 뉴럴 네트워크 중 기타 성능 민감형 애플리케이션의 효율을 향상시키기 위해 인텔® MKL-DNN은 여러 최적화된 딥러닝 요소를 제공해 범용 구성 모듈이 높은 효율로 실행되도록 다른 딥러닝 아키텍처에서도 응용할 수 있게 만들었습니다. 이 모듈은 행렬 곱셈과 콘볼루션 외에도 다음을 포함합니다.

- 다이렉트 배치 콘볼루션
- 내적

- 풀링: 최대, 최소, 평균
- 표준화: 크로스 채널 부분 응답 정규화(LRN), 대량 정규화
- 활성화: 선형 유닛 설정(ReLU)
- 데이터 작업: 다차원 전위(전환), 분해, 통합, 합산과 크기 조절.

이 고효율의 함수 모듈은 딥러닝 모델에 광범위하게 응용될 수 있습니다.

응용 유형	위상학적 구조
화상 인식	AlexNet、VGG、GoogleNet、ResNet、MobileNet
영상 분할	FCN、SegNet、MaskRCNN、U-Net
부피 분할	3D-Unet
객체 탐지	SSD、Faster R-CNN、Yolo
기계 번역	GNMT
음성 문자 인식	DeepSpeech、WaveNet
적대 신경망	DCGAN、3DGAN
강화 학습	A3C

CPU에서 딥러닝의 성능을 대폭 향상시키기 위해 인텔은 여러 오픈소스 공동체와 협력해 인텔® MKL-DNN을 다양한 딥러닝 아키텍처에 집성했습니다. 일찍이 2016년에는 인텔® MKL-DNN 최적화를 거친 Caffe가 인텔® 제온® CPU E5-2697 v3를 이용하면서 기존의 Caffe보다 그 성능이 10이나 높아졌습니다.¹ 최적화를 거친 ResNet-50 역시 인텔® 제온® 실버 9282 CPU에서 초당 7736 장 그림을 선보이는 성능²을 실현했습니다.

인텔® MKL-DNN은 현재 여러 딥러닝 아키텍처를 CPU에서 운용할 때 사용하는 기본 구성이 되어 있기 때문에, 개발자는 딥러닝 아키텍처 설치와 응용에서 인텔® MKL-DNN의 업그레이드된 성능을 바로 활용할 수 있습니다.

더 많은 정보는 다음을 참조하십시오.

- <https://software.intel.com/zh-cn/articles/intel-mkl-dnn-part-1-library-overview-and-installation>
- <https://software.intel.com/zh-cn/articles/introducing-dnnprimitives-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>

소스가 통일된 빅데이터 분석 +AI 플랫폼 Analytics Zoo

Analytics Zoo 는 하나의 통일된 빅데이터 분석과 인공지능 오픈소스 플랫폼으로서 사용자가 손쉽게 빅데이터 기반, end-to-end 딥러닝 애플리케이션을 개발할 수 있게 돋기 위해 출시되었습니다.

Analytics Zoo는 사용자가 Spark, TensorFlow, Pytorch, Keras 와 BigDL 프로세스와 향후 지원이 필요할 수 있는 기타 아키텍처 등을 하나의 채널에 빙름 없이 집성할 수 있도록 돋습니다. 이 모델은 수백 수천에 달하는 노드 규모의 빅데이터 클러스터에 그대로 확장되고, 분산형 트레이닝이나 추론을 진행함으로써 인공지능 솔루션 개발을 간소화하기 때문에 별도의 전용 인프라가 필요하지 않습니다.

컴퓨팅 성능을 향상시키기 위해 Analytics Zoo 는 인텔® MKL 과 인텔® MKL-DNN 같은 여러 소프트웨어 라이브러리를 융합했습니다. 하드웨어 측면에서는 인텔® 제온® CPU 플랫폼을 기반으로 2세대 인텔® 제온® 확장성 CPU 가 집성한 벡터와 딥러닝 명령을 충분히 릴리즈해 트레이닝과 추론 속도를 대폭 향상시킬 수 있었습니다.

데이터 스토리지와 처리 어셈블리 라인을 하나의 통일된 인프라에 통합했기 때문에 데이터를 이동할 필요가 없게 되었습니다. 덕분에 개발 배치 효율과 확장성을 향상시키고, 하드웨어 관리와 개발자가 새로운 언어를 학습하는데 소요되는 시간을 절감했습니다. 개발 배치 효율과 자원 이용률 그리고 유연성을 향상시켰고, 전체 비용을 절감했으며, 컴퓨팅 효율과 성능에도 영향을 미쳤습니다. 개발자에게 필요한 것은 인공지능 솔루션을 확장할 때 Analytics Zoo 가 제공한 다양한 특성과 기능, 여러 분석과 인공지능 툴을 충분히 이용하는 것 뿐이며, 그렇게 만 하면 빅데이터 분석과 인공지능을 고효율로 융합하고 배치하며 응용할 수 있습니다.

Analytics Zoo 플랫폼이 지원하는 여러 AI 아키텍처 중 BigDL 은 인텔이 자체 개발한 것입니다. BigDL 은 Apache Spark 기반의 분산형 딥러닝 아키텍처로 클러스터를 수정할 필요 없이 기존의 Apache Spark 와 Hadoop 클러스터에서 바로 운용할 수 있습니다. 개발자는 BigDL 을 기반으로 하고 Scala나 Python 언어를 사용해 딥러닝 애플리케이션을 프로그래밍하고, Spark 클러스터가 확장성 측면에서 강력한 능력을 충분히 발휘할 수 있도록 빅데이터 분석과 인공지능을 융합할 수 있습니다. 지난 몇 년 간 이미 BigDL 에 친숙해진 사용자라면 Analytics Zoo로 BigDL을 바로 호출해 틈 없이 전환할 수 있습니다.

Analytics Zoo 기술 특성 :

- 손쉽게 사용할 수 있는 추상화 API(예: 트랜스미션 러닝 지원, 서명 작업, Spark 데이터 프레임과 ML 채널 지원, 온라인 모델 서비스 API 등)는 모델 트레이닝과 서비스에 쓰입니다.
- 범용 기능 공정 작업은 영상, 텍스트, 3D 영상 등에 쓰입니다.
- 내장 딥러닝 모델(예: 객체 탐지, 영상 분류, 제안, 비정상행위 탐지, 텍스트 매칭, seq2seq 등).
- 참고 사례 (예 : 비정상행위 탐지, 감성 분석, 금융 거래 사기 방지, 이미지 유사성 등).

Analytics Zoo end-to-end, 일관된 빅데이터 분석 +AI 플랫폼

응용 사례	추천	비정상행위 탐지	텍스트 분류	문자 매칭
모델	영상 분류	객체 탐지	seq2seq	변환기
특성 공정	영상	3D 영상	텍스트	시계열
하이레이어 어셈블리 플랫폼		tfpark : Spark 의 분산형 TF Spark 의 분산형 자동 솔빙 그레디언트를 장착한 Keras nnframes: 딥러닝에 사용하는 Spark 데이터 프레임과 머신러닝 어셈블리 플랫폼 분산형 모델 서비스 (다량 처리, 어셈블리 처리와 온라인 처리)		
백 엔드/ 라이브러리		TensorFlow	Keras	PyTorch
		BigDL	NLP Architect	Apache Spark
		Apache Flink	Ray	인텔® MKL-DNN
		OpenVINO™ 툴킷	인텔® 옵테인™ 데이터 센터급 영구 메모리	인텔® 딥러닝 가속

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

더 많은 정보는 다음을 참조하십시오.

https://software.intel.com/zh-cn/blogs/2018/09/10/analytics-zoo-unifying-analytics-aifor-apache-spark?elq_cid=4287274&erpm_id=7282583

인텔® 아키텍처 최적화 지향 Caffe

최적화

인텔® 아키텍처 최적화 지향 Caffe 는 당시 최신 버전인 인텔® MKL을 집성한 최신 버전에서 당시 제온® CPU 제품이 이미 집성한 인텔® AVX 2 와 인텔® AVX-512 를 전문적으로 최적화한 것입니다. BVLC Caffe와 완전하게 호환될 뿐만 아니라 BVLC Caffe 의 모든 장점까지 갖추고 있습니다. CPU 최적화 기능도 보유하고 있고, 인텔® 아키텍처 CPU 에서 아주 뛰어난 성능을 보였을 뿐만 아니라 여러 노드 분포 프로세스 트레이닝도 지원합니다.

인텔® 아키텍처 최적화 지향 Caffe 는 완벽한 Post-training 양자화 방안을 지원하며, 여러 CNN 모델에서 성과를 보았습니다. 특히 2 세대 인텔® 제온® 확장성 CPU 기반 플랫폼 (CPU 소개 부분 참조)에서 INT8에 대해 지원한 인텔® 딥러닝 가속 기술(VNNI 명령어 조합)을 최적화해 예측 정확도에 영향을 미치지 않는 상황에서 여러 딥러닝 모델이 INT8 을 사용할 때 주론 속도가 FP32 를 사용할 때 속보다 2 ~ 4 배 (아래 그림 참조)¹⁾ 가 되도록 했습니다. 덕분에 사용자는 딥러닝 응용 업무 효율을 대대적으로 향상시켰습니다.

더 많은 정보는 다음을 참조하십시오.

- <https://software.intel.com/en-us/articles/caffe-optimized-for-intelarchitecture-applying-modern-codetechniques>
- <https://software.intel.com/zh-cn/videos/what-is-intel-optimization-for-caffe>

최적화된 딥러닝 아키텍처와 툴 패키지

인텔® 딥러닝 가속 기술을 채택한 ResNet-50 개인

2 세대 인텔® 제온® 플래티넘 8280 CPU vs 인텔® 제온® 플래티넘 8180 CPU

인텔® 제온® 확장성 CPU	2 세대 인텔® 제온® 확장성 CPU	mxnet	PYTORCH	TensorFlow	Caffe	OpenVINO®
FP32	INT8 W/ 인텔® DL Boost	3.0x	3.7x	3.9x	4.0x	3.9x
INT8	INT8 W/ 인텔® DL Boost	1.8x	2.1x	1.8x	2.3x	1.9x

¹⁾ 구성 정보는 다음을 참조하십시오. <https://www.intel.cn/content/www/cn/zh/benchmarks/server/xeon-scalable/xeon-scalable-artificial-intelligence.html>

適用於 Intel® 架構最佳化的 TensorFlow

최적화

인텔® 아키텍처 최적화 지향 TensorFlow는 인텔이 CPU에서 딥러닝 모델을 운용할 때 맞닥뜨리는 성능 측면의 도전에 대응하기 위해 출시한 최적화 버전으로 딥러닝 유형의 워크로드가 여러 상황에서 인텔® MKL-DNN 기본 알고리즘 유닛을 이용해 효율적으로 운용할 수 있게 했습니다.

성능을 현저히 향상시키기 위해, 인텔은 다른 조치를 취해 TensorFlow를 최적화했습니다.

산출 그래프 최적화

인텔은 CPU 운용 시, 기본 TensorFlow 작업을 인텔 최적화 버전으로 손쉽게 교체해 사용자가 뉴럴 네트워크를 변경하지 않고 기존의 Python 프로그래밍을 사용한 상황에서 성능을 업그레이드할 수 있도록 다양한 산출 그래프 최적화 채널을 출시했습니다. 또한 불필요한 고가의 데이터 구성 전환을 제거하고, 여러 개의 연산을 통합해 CPU에서 고속 캐시를 고효율적으로 중복 사용하고, 빠르게 뒤로 전파되는 중간 상태를 처리합니다.

산출 그래프를 최적화 조치하면 성능이 한 단계 더 향상되기 때문에 TensorFlow 프로그래밍을 따로 신경 쓸 필요가 없습니다. 그 밖에 데이터 구성 최적화는 성능을 최적화하는 중요한 조치 중 하나입니다. 이전에는 TensorFlow에서 나온 로컬 형식의 데이터 구성은 내부 형식으로 전환해 삽입해 CPU에서 연산을 실행하고, 연산 출력을 TensorFlow로 전환하면 그로 인해 성능 오버헤드 문제가 발생했습니다. 하지만 지금은 인텔® MKL을 이용해 연산 실행한 서브그래프를 최적화하기 때문에 서브그래프 연산 중 전환 문제를 제거할 수 있습니다. 자동 삽입된 전환 노드는 서브그래프 경계에서 데이터 구성은 전환하고, 채널 융합과 같은 또 다른 중요한 최적화를 통해 여러 개의 연산을 고효율적으로 운영되는 단일 인텔® MKL 연산으로 자동 융합합니다.

기타 최적화

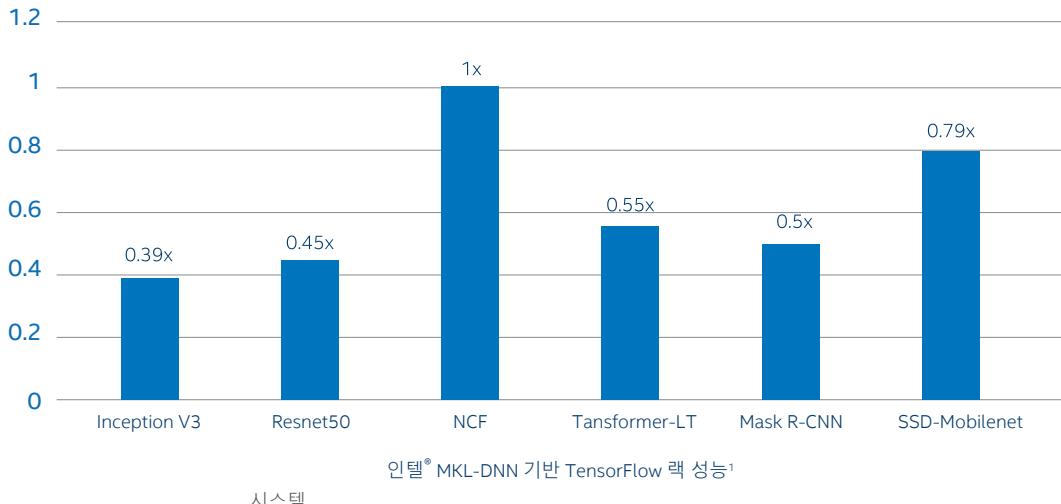
다양한 딥러닝 모델에서 최고의 CPU 성능을 실현할 수 있도록, 인텔은 수많은 TensorFlow를 겨냥해 아키텍처를 조정했습니다. 예를 들어,

TensorFlow에 원래 있는 메모리 풀 스플리터를 사용해 사용자 지정 메모리 풀 스플리터를 개발하고 인텔® MKL과 동일한 메모리 풀(인텔® MKL imalloc 기능 사용)을 공유함으로써 메모리를 너무 일찍 운영 체제로 반환할 필요가 없기 때문에 고비용이 드는 페이지 결실과 페이지 제거 현상을 막을 수 있게 되었습니다. 뿐만 아니라 여러 개의 스레드 라이브러리(TensorFlow가 사용한 pthread와 인텔® MKL이 사용한 OpenMP)페이지에서 딥 최적화를 진행해 공존할 수 있게 함으로써 서로 CPU 리소스를 쟁탈하는 현상을 방지해 리소스 종합 이용률을 향상시켰습니다.

더 많은 정보는 다음을 참조하십시오.

- https://www.intel.ai/tensorflow/?_ga=2.231295069.330745958.1563951842597697079.1551333838&elq_cid=4287274&erpm_id=7282583
- <https://www.intel.ai/improvingtensorflow-inference-performance-on-intel-xeon-processors/#gs.vOkayg>

지연 시간의 결과 (인텔® MKL-DNN/Eigen 라이브러리) — 낫을수록 좋습니다



¹ 시스템 구성: 인텔® 제온® 플래티넘 8180 CPU @2.50GHz, OS: CentOS Linux 7(Core), TensorFlow 소스 코드: <https://github.com/tensorflow/tensorflow>, TensorFlow 버전 번호: 355cc566efd2d86fe71fa9d755ceabe546d577a

인텔® 아키텍처 최적화 지향 Python 배포 패키지

인텔® 아키텍처 최적화 지향 Python 배포 패키지는 인텔이 주도적으로 개발하고 강력한 기능을 보유한 소프트웨어 개발 툴킷으로 C 와 Fortran 컴파일러, MKL과 애널라이저 같이 프로그래밍 Python 최초 확장에 필요한 모든 것을 제공합니다. 뿐만 아니라 NumPy, SciPy, Scikit-learn, Pandas, Jupyter, matplotlib, mpi4py 같은 여리 종류의 고성능 데이터 분석과 MKL까지 집성했습니다.

인텔® Python 배포 패키지는 인텔® Parallel Studio XE 의 중요한 툴킷 중 하나로 효율성이 높고 다양한 특성을 지니고 있습니다.

- 숫자와 과학, 데이터 분석, 머신러닝 등이 포함된 계산집약형 응용을 가속화할 수 있게 개봉해 바로 사용하는 uMath, NumPy, SciPy와 Scikit-learn 등 툴을 제공했습니다.
- 인텔® AVX-512 와 다중 스레드 명령, Numba 와 Cython 같은 인텔® 성능 라이브러리(예: 인텔® MKL)를 집성하고, 최신 벡터화 명령을 내장하고, 다중 스레드에 구축한 모듈 라이브러리 인텔® TBB의 컴포저를 병렬을 응용해

Python 의 다핵성 CPU 기반 병렬 응용 기능을 해제하고 더 나아가 인텔® 아키텍처에 기반한 플랫폼에서 Python 프로그래밍 운영 성능을 향상 시킴으로써 시스템 호환성을 보장하기 때문에 코드를 변경할 필요가 없습니다.

- Python2.7, Python3.6 과 최신 버전의 인텔® 아키텍처 CPU 를 지원하고, 벡터 컴퓨터 (SVM) 와 K-means 예측, 랜덤 포레스트와 XGBoost 알고리즘 등, 최적화된 TensorFlow, Caffe 등 딥러닝 라이브러리와 머신러닝 라이브러리를 제공하기 때문에 과학 기술 계산과 머신러닝 등 워크로드 구축과 양산 준비 알고리즘 확장이 편리합니다.

벤치마크 테스트를 거쳐 인텔® Python 배포 패키지와 기타 오픈 소스 Python 중 Scikit-learn 툴 패키지의 효율을 비교해보면 인텔® 아키텍처 최적화 지향 Python 지표가 눈에 띄게 향상 (그림 참조). 효율이 높을 수록 함수 속도가 빠르고 로컬 C언어 속도가 가까워짐)되게 나타납니다. 예를 들어, 인텔® Python 배포 패키지 K-means 클러스터, 선형 회귀 등

알고리즘의 효율은 인텔® 데이터 분석 가속 라이브러리 (Intel® Data Analytics Acceleration Library, 인텔® DAAL) 중 C 언어 효율의 90% 를 달성할 수 있습니다.

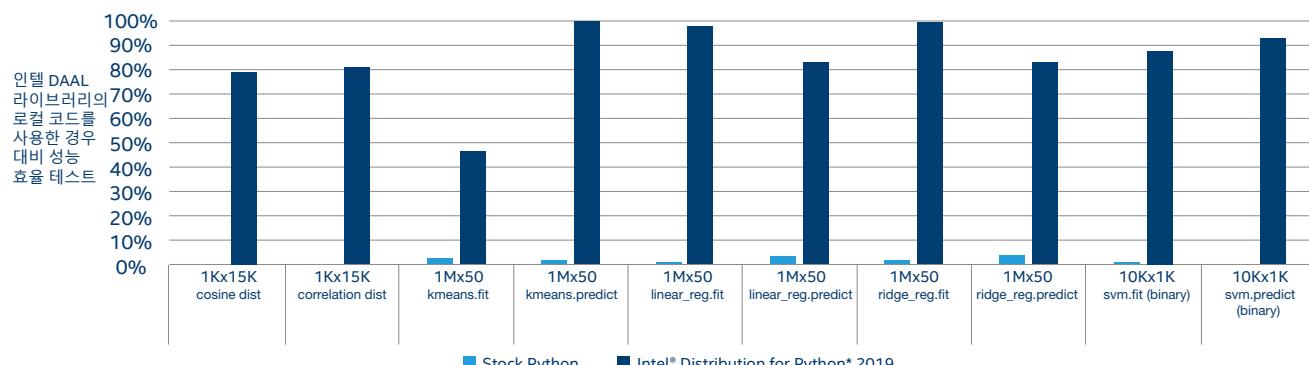
배치가 간단해 사용이 편리하다는 점 역시 인텔® Python 배포 패키지의 특성입니다. Conda 소프트웨어 패키지 관리자와 Anaconda 클라우드를 사용하면 사용자는 명령 하나만 실행하면 코어 Python 환경을 설치할 수 있습니다.

- `conda install intelpython3 -c intel`

인텔® Python 배포 패키지는 사전에 구축한 이진법 파일로 pip, Docker images, YUM 과 APT repos 등 여러 채널을 통해 획득할 수 있습니다.

더 많은 정보는 다음을 참조하십시오.

<https://software.intel.com/en-us/distribution-for-python>



인텔 최적화는 scikit-learn 프로그래밍 패키지의 효율을 향상시켜 인텔® 제온®CPU 에서 로컬 코드가 운영되는 속도와 가까워지게 했습니다

OpenVINO™ 툴킷

OpenVINO™ 툴킷은 인텔이 출시한 딥러닝 추론과 배치를 가속화하는 소프트웨어 툴킷으로 고성능 컴퓨터 비전 처리와 응용을 가속화하는데 쓰입니다. 이 툴은 이종 실행을 허용하고, Windows 와 Linux 시스템 및 Python/C++ 언어를 지원하기 때문에 컴퓨터 비전 기술이 스마트 카메라, cctv, 로봇, 지능형 교통, 스마트 의료 등 영역에서 응용될 수 있도록 추진할 수 있습니다.

이 툴킷은 컴퓨터 비전 솔루션의 성능을 향상시키고, 개발 시간을 단축하며, 인텔이 제공한 다양한 하드웨어 옵션에서 효익을 얻는 경로를 간소화했습니다. 이 옵션은 성능을 향상시키고, 전력을 낮추어 하드웨어 이용률을 최대화함으로써 사용자가 낮은 리소스로 높은 수익을 획득할 수 있게 하며, 새로운 제품을 개인 맞춤형으로 디자인할 수 있게 합니다.

딥 콘볼루션 신경망(CNN)을 기반으로 인텔® 하드웨어 (가속기 포함)의 워크로드를 확장함으로써 OpenVINO™ 툴킷이 인텔® 아키텍처 CPU 가 집성한 그래픽카드 (Integrated GPU), FPGA, 인텔® Movidius™ VPU 등 칩에 따라 비주얼 시스템의 기능과 성능을 강화할 수 있게 합니다. 최신 발표한 OpenVINO™ 버전은 2 세대 인텔® 제온® 확장성 CPU를 지원할 수 있고, 인텔® AVX-512 및 VNNI 의 인텔® 딥러닝 가속 (인텔® DL Boost) 기술을 채택해 추론 성능을 향상시킴으로써, 고객이 소프트웨어를 변경하지 않는 상황에서 하드웨어 제품 업그레이드와 알고리즘 이식을 빠르게 완료하고 경계에서 고성능 컴퓨터 비전과 딥러닝 응용 개발을 빠르게 실현할 수 있게 해줍니다.

- 인텔® 아키텍처 플랫폼에서 컴퓨터 비전 관련 딥러닝 성능이 19 배 이상¹ 향상됩니다.
- CNN-based 의 네트워크를 종단 장치의 성능 장애에 릴리즈합니다.

- OpenCV, OpenVX 비주얼 라이브러리의 전통적인 API에 가속화와 최적화를 실현합니다.
- 범용 API 기반 인터페이스를 CPU, GPU, FPGA 등 설비에서 운용합니다.
- 인텔® 플랫폼 기반 최적화 OpenVINO™ 툴킷은 주로 딥러닝 배치 툴 패키지와 전통적인 컴퓨터 비전 툴 패키지 두 부분을 포함합니다. 딥러닝 배치 툴 패키지는 모델 옵티마이저 (Model Optimizer) 와 추론 엔진 (Inference Engine) 두 개의 핵심 부분을 포함합니다.
- 모델 옵티마이저 (Model Optimizer) : 지정 모델을 기본 중간 표현 (Intermediate Representation, IR)으로 전환하고, 모델을 최적화하며 ONNX, TensorFlow, Caffe, MXNet, Kaldi* 등 딥러닝 아키텍처를 지원합니다.
- 추론 엔진 (Inference Engine) : 하드웨어 명령어 조합 측면의 딥러닝 모델 가속 운용을 지원하며, 다음의 하드웨어를 지원합니다. CPU, GPU, FPGA, VPU.



또한 전통적인 OpenCV 이미지 처리 라이브러리에도 명령어 조합 최적화를 진행해 성능과 속도를 눈에 띄게 향상시킵니다. 컴퓨터 비전 툴 :

- OpenCV(3.3 버전): OpenCV와 최신 인텔 CPU 최적화 비주얼 라이브러리(Intel Photography Vision Library)를 프리컴파일하고, 얼굴 감지/인식, 눈 깜빡임 감지, 미소 감지 기능 등을 보유하고 있습니다.
- OpenVX : 도형 기반 OpenVX를 실현하고, 전통적인 CV 작업과 CNN 요소를 지원합니다. Khronos OpenVX 뉴럴 네트워크 확장 1.2를 지원합니다.
- 기타 : OpenCL™ 드라이버와 런타임, 미디어 드라이버, 인텔® Media SDK와 인텔® SDK OpenCL™ 애플리케이션을 함께 작업한 컴퓨터 비전 SDK, 인텔® MKL-DNN, CLDNN 모두 포함되며, 단독 다운로드할 필요가 없습니다.

¹ <https://software.intel.com/en-us/articles/a-guide-for-setting-up-docker-based-openvino-development-environment-with-ubuntu-system>

그 밖에, 여러 애플리케이션에서 컴퓨터 비전과 딥러닝을 빠르고 효율적으로 실행하기 위해 출시된 툴 패키지로, 인텔® 플랫폼 기반 최적화 OpenVINO™ 툴킷은 현재 VGG-16, VGG-19, SqueezeNet, ResNet, Inception, CaffeNet, SSD, Faster-RCNN, FCN8 등과 같은 사전 전환한 Caffe, TensorFlow, Mxnet 모델의 MO 파일을 제공하며, 20 개 이상의 사전 트레이닝 모델을 갖추고 있습니다. 소프트웨어 개발자와 데이터 과학자는 이 툴을 이용해 개인 맞춤형 딥러닝 애플리케이션을 빠르게 실행할 수 있고, OpenCV, OpenVX 의 기본 라이브러리를 사용해 특정 알고리즘을 만들고 맞춤형 및 신형 응용 개발을 진행할 수 있습니다.

OpenVINO™ 툴킷은 인텔 플랫폼에서 시작 자료가 현실이 되게 만들어 여러 사용자가 컴퓨터 비전 애플리케이션을 손쉽게 개발하고 빠르게 배치할 수 있도록 도왔습니다. 여러 딥러닝 애플리케이션 시나리오에서는 인공지능 솔루션이 내포한 잠재력을 이미 선 보였습니다.

더 많은 정보는 다음을 참조하십시오.

<https://software.intel.com/zh-cn/openvino-toolkit>

해당 매뉴얼 관련 전문 어휘집

영문 약자	영문 풀네임	중문 풀네임
ALS	Alternating Least Squares	교대최소제곱
API	Application Programming Interface	애플리케이션 프로그래밍 인터페이스
CART	Classification and Regression Tree	분류 회귀 트리
CNN	Convolutional Neural Networks	콘볼루션 뉴럴 네트워크
Data	Mart Data Mart	데이터 마트
DDR SDRAM	Double Data Rate Synchronous Dynamic Random-Access Memory	DDR SD 랜덤 기억 장치
Decision Tree	Decision Tree	의사결정 트리
DL	Deep Learning	딥러닝
DMLC	Distributed Machine Learning Community	분산형 머신러닝 커뮤니티
GBDT	Gradient Boosting Decision Tree	그래디언트 부스팅 의사결정 트리
GMF	Generalized Matrix Factorization	범용(일반화) 행렬 분해
GRU	Gated Recurrent Unit	게이트 순환 유닛
HDFS	Hadoop Distributed File System	분산형 파일 시스템
HMM	Hidden Markov Model	은닉 마르코브 모델
IAaas	Intelligent Analysais as a service	서비스 플랫폼형 지능 분석
Imbalance Ratio	Imbalance ratio	비평형
Intel DAAL	Intel Data Analytics Acceleration Library	인텔 데이터 분석과 머신러닝 가속 라이브러리
Layer Fusion	Layer Fusion	레이어 융합
LF	Logistic Regression	로지스틱 회귀
LR	Logistic Regression	로지스틱 회귀
LSTM	Long Short-Term Memory	LSTM
LTV	Long Time Variable	
MF	Matrix Factorization	행렬 분해
MKL	Math Kernel Library	MKL
MKL-DNN	Math Kernel Library for Deep Neural Networks	DNN 지향 MKL
ML	Machine Learning	머신러닝
MLP	Multi-Layer Perception	다층 신경망
NBM	Naive Bayesian Model	단순 베이지안 모델
NCF	Neural Collaborative Filtering	신경 협업 필터링
NLP	Natural Language Processing	자연어 처리
NNs	Neural Networks	뉴럴 네트워크
NUMA	Non-Uniform Memory Access Architecture	NUMA
One-Hot	One-Hot	원-핫 인코딩
POJO	Plain Ordinary Java Object	간단한 Java 객체
PR	Precision – Recall	정밀도- 리콜
RDD	Resilient Distributed Datasets	RDD
ResNet	Residual Net	레지듀얼 네트워크
RF	Random Forest	랜덤 포레스트
RNN	Recurrent Neural Network	순환 신경망
RS	Recommender System	추천 시스템
SQL	Structured Query Language	구조화 조회 언어
Streaming Model	Streaming Model	스트리밍 모듈
SVM	Support Vector Machine	서포트 벡터 머신
WAD	Wide and Deep	와이드 앤 딥
XGBoost	eXtreme Gradient Boosting	익스트림 그래디언트 부스팅

성능 테스트에서 사용한 소프트웨어와 워크로드는 인텔 마이크로프로세서에서 성능 최적화를 진행했으며, SYSmark 와 MobileMark 등 테스트는 모두 특정 컴퓨터 시스템, 하드웨어, 소프트웨어, 운영 체제 및 기능을 기반으로 합니다. 위의 어떤 요소라도 변경하면 테스트 결과가 변할 수 있습니다. 기타 정보 및 성능 테스트 (기타 제품 결합 사용 시 운용 성능 포함)를 참조해 대상 제품에 대해 전면적인 평가를 진행합니다. 더 많은 정보는 www.intel.com/benchmarks 를 참조하십시오.

특정 시스템의 특수 테스트에서 부품 성능을 테스트합니다. 하드웨어와 소프트웨어 또는 구성 차이는 실제 성능에 영향을 미칩니다. 따라서 구매할 때 기타 정보 출처를 열람해 성능을 평가하십시오. 성능과 기본 테스트 프로그래밍 결과에 대한 정보는 www.intel.com/benchmarks 를 참조하십시오.

인텔의 기술 특성과 장점은 시스템 구성에 따라 결정되며, 활성화하려면 하드웨어와 소프트웨어 또는 서비스 지원이 필요할 수 있습니다. 제품의 성능은 시스템 구성 변화를 기반으로 합니다. 어떤 제품이나 부품도 절대적으로 안전한 것은 없습니다. 더 많은 정보는 최초 설비 제조업체 또는 유통업체를 통해 얻거나 intel.com 을 참조하십시오.

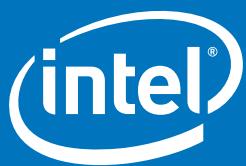
최적화 설명서: 인텔 컴파일러의 인텔 마이크로프로세서 최적화 정도는 인텔 마이크로프로세서의 최적화 정도에 따라 달라질 수 있습니다. 이 최적화는 SSE2, SSE3 와 SSSE3 명령어 조합과 기타 최적화를 포함합니다. 인텔 마이크로프로세서가 아닌 제품에 대한 최적화 존재 여부와 기능 또는 효력에 대해 인텔은 그 어떤 보장도 하지 않습니다. 이 제품 중 마이크로프로세서의 최적화를 결정하는 것은 인텔 마이크로프로세서를 겨냥한 것입니다. 인텔 마이크로 아키텍처의 특성을 보유하지 않은 특정 최적화는 인텔 마이크로프로세서에 보류합니다. 적용한 제품 사용자와 참고 가이드를 참조해 이 설명서와 관련한 구체적인 명령어 조합 정보를 획득하십시오.

설명서 버전 :#20110804

어떤 제품이나 부품도 절대적으로 안전한 것은 없습니다.

설명한 원가 절감 시나리오는 모두 특정 상황과 구성 중 특정 인텔 제품이 향후 원가에 어떻게 영향을 미치고 원가를 절약하는지 예를 들어 설명한 것입니다. 상황은 모두 다릅니다. 인텔은 모든 비용 또는 비용 절감을 보장하지는 않습니다.

인텔은 제 3 자 데이터를 컨트롤하거나 감사하지 않습니다. 이 내용을 감사하고, 기타 출처를 의논하고, 데이터의 정확성을 확인 및 검토하십시오.



AI 적용을 가속화하려면 다음 사이트를 방문하십시오.



홈페이지
Intel.com/ai



웨이보
@Intel Business



위챗
인텔 비즈니스 채널

© 2019 인텔 회사 저작권 소유. 인텔, Intel, 제온, 옵테인은 인텔 회사의 미국 또는 다른 국가의 상표입니다.
인텔 상표 또는 상표 및 브랜드 명칭 데이터뱅크의 전체 명단은 intel.com 의 상표를 참조하십시오.
* 다른 명칭과 브랜드는 다른 소유자의 자산일 수 있습니다.