Major AI, ML, and DL use cases and architectures

NetApp Solutions

NetApp
October 20, 2023

Table of Contents

Major AI, ML, and DL use cases and architectures

Previous: Use cases summary.

Major AI, ML, and DL use cases and methodology can be divided into the following sections:

Spark NLP pipelines and TensorFlow distributed inferencing

The following list contains the most popular open-source NLP libraries that have been adopted by the data science community under different levels of development:

- Natural Language Toolkit (NLTK). The complete toolkit for all NLP techniques. It has been maintained since the early 2000s.
- TextBlob. An easy-to-use NLP tools Python API built on top of NLTK and Pattern.
- Stanford Core NLP. NLP services and packages in Java developed by the Stanford NLP Group.
- Gensim. Topic Modelling for Humans started off as a collection of Python scripts for the Czech Digital Mathematics Library project.
- SpaCy. End-to-end industrial NLP workflows with Python and Cython with GPU acceleration for transformers.
- Fasttext. A free, lightweight, open-source NLP library for the learning-of-word embeddings and sentence classification created by Facebook's AI Research (FAIR) lab.

Spark NLP is a single, unified solution for all NLP tasks and requirements that enables scalable, high-performance, and high-accuracy NLP-powered software for real production use cases. It leverages transfer learning and implements the latest state-of-the-art algorithms and models in research and across industries. Due to the lack of full support by Spark for the above libraries, Spark NLP was built on top of Spark ML to take advantage of Spark's general-purpose in-memory distributed data processing engine as an enterprise-grade NLP library for mission-critical production workflows. Its annotators utilize rule-based algorithms, machine learning, and TensorFlow to power deep learning implementations. This covers common NLP tasks including but not limited to tokenization, lemmatization, stemming, part-of-speech tagging, named-entity recognition, spell checking, and sentiment analysis.

Bidirectional Encoder Representations from Transformers (BERT) is a transformer-based machine learning technique for NLP. It popularized the concept of pretraining and fine tuning. The transformer architecture in BERT originated from machine translation, which models long-term dependencies better than Recurrent Neural Network (RNN)-based language models. It also introduced the Masked Language Modelling (MLM) task, where a random 15% of all tokens are masked and the model predicts them, enabling true bidirectionality.

Financial sentiment analysis is challenging due to the specialized language and lack of labeled data in that domain. FinBERT, a language model based on pretrained BERT, was domain adapted on Reuters TRC2, a financial corpus, and fine-tuned with labeled data (Financial PhraseBank) for financial sentiment classification. Researchers extracted 4, 500 sentences from news articles with financial terms. Then 16 experts and masters students with finance backgrounds labeled the sentences as positive, neutral, and negative. We built an end-to-end Spark workflow to analyze sentiment for Top-10 NASDAQ company earnings call transcripts from 2016 to 2020 using FinBERT and two other pre-trained pipelines (Sentiment Analysis for Financial News, Explain Document DL) from Spark NLP.

The underlying deep learning engine for Spark NLP is TensorFlow, an end-to-end, open-source platform for

machine learning that enables easy model building, robust ML production anywhere, and powerful experimentation for research. Therefore, when executing our pipelines in Spark `yarn cluster` mode, we were essentially running distributed TensorFlow with data and model parallelization across one master and multiple worker nodes, as well as network- attached storage mounted on the cluster.

Horovod distributed training

The core Hadoop validation for MapReduce-related performance is performed with TeraGen, TeraSort, TeraValidate, and DFSIO (read and write). The TeraGen and TeraSort validation results are presented in TR-3969: NetApp Solutions for Hadoop for E-Series and in the section “Storage Tiering” (xref) for AFF.

Based upon customer requests, we consider distributed training with Spark to be one of the most important of the various use cases. In this document, we used the Hovorod on Spark to validate Spark performance with NetApp on-premises, cloud-native, and hybrid cloud solutions using NetApp All Flash FAS (AFF) storage controllers, Azure NetApp Files, and StorageGRID.

The Horovod on Spark package provides a convenient wrapper around Horovod that makes running distributed training workloads in Spark clusters simple, enabling a tight model design loop in which data processing, model training, and model evaluation are all done in Spark where training and inferencing data resides.

There are two APIs for running Horovod on Spark: a high-level Estimator API and a lower-level Run API. Although both use the same underlying mechanism to launch Horovod on Spark executors, the Estimator API abstracts the data processing, model training loop, model checkpointing, metrics collection, and distributed training. We used Horovod Spark Estimators, TensorFlow, and Keras for an end-to-end data preparation and distributed training workflow based on the Kaggle Rossmann Store Sales competition.

The script `keras_spark_horovod_rossmann_estimator.py` can be found in the section "Python scripts for each major use case." It contains three parts:

- The first part performs various data preprocessing steps over an initial set of CSV files provided by Kaggle and gathered by the community. The input data is separated into a training set with a `Validation` subset, and a testing dataset.
- The second part defines a Keras Deep Neural Network (DNN) model with logarithmic sigmoid activation function and an Adam optimizer, and it performs distributed training of the model using Horovod on Spark.
- The third part performs prediction on the testing dataset using the best model that minimizes the validation set overall mean absolute error. It then creates an output CSV file.

See the section “Machine Learning” for various runtime comparison results.

Multi-worker deep learning using Keras for CTR prediction

With the recent advances in ML platforms and applications, a lot of attention is now on learning at scale. The click-through rate (CTR) is defined as the average number of click-throughs per hundred online ad impressions (expressed as a percentage). It is widely adopted as a key metric in various industry verticals and use cases, including digital marketing, retail, e-commerce, and service providers. See our TR-4904: Distributed training in Azure - Click-Through Rate Prediction for more detail on the applications of CTR and an end-to-end Cloud AI workflow implementation with Kubernetes, distributed data ETL, and model training using Dask and CUDA ML.

In this technical report we used a variation of the Criteo Terabyte Click Logs dataset (see TR-4904) for multi-worker distributed deep learning using Keras to build a Spark workflow with Deep and Cross Network (DCN) models, comparing its performance in terms of log loss error function with a baseline Spark ML Logistic Regression model. DCN efficiently captures effective feature interactions of bounded degrees, learns highly

nonlinear interactions, requires no manual feature engineering or exhaustive searching, and has low computational cost.

Data for web-scale recommender systems is mostly discrete and categorical, leading to a large and sparse feature space that is challenging for feature exploration. This has limited most large-scale systems to linear models such as logistic regression. However, identifying frequently predictive features and at the same time exploring unseen or rare cross features is the key to making good predictions. Linear models are simple, interpretable, and easy to scale, but they are limited in their expressive power.

Cross features, on the other hand, have been shown to be significant in improving the models' expressiveness. Unfortunately, it often requires manual feature engineering or exhaustive search to identify such features. Generalizing to unseen feature interactions is often difficult. Using a cross neural network like DCN avoids task-specific feature engineering by explicitly applying feature crossing in an automatic fashion. The cross network consists of multiple layers, where the highest degree of interactions is provably determined by layer depth. Each layer produces higher-order interactions based on existing ones and keeps the interactions from previous layers.

A deep neural network (DNN) has the promise to capture very complex interactions across features. However, compared to DCN, it requires nearly an order of magnitude more parameters, is unable to form cross features explicitly, and may fail to efficiently learn some types of feature interactions. The cross network is memory efficient and easy to implement. Jointly training the cross and DNN components together efficiently captures predictive feature interactions and delivers state-of-the-art performance on the Criteo CTR dataset.

A DCN model starts with an embedding and stacking layer, followed by a cross network and a deep network in parallel. These in turn are followed by a final combination layer which combines the outputs from the two networks. Your input data can be a vector with sparse and dense features. In Spark, both ml and mllib libraries contain the type `SparseVector`. It is therefore important for users to distinguish between the two and be mindful when calling their respective functions and methods. In web-scale recommender systems such as CTR prediction, the inputs are mostly categorical features, for example `'country=usa'`. Such features are often encoded as one-hot vectors, for example, `'[0,1,0, …]'`. One-hot-encoding (OHE) with `SparseVector` is useful when dealing with real-world datasets with ever-changing and growing vocabularies. We modified examples in DeepCTR to process large vocabularies, creating embedding vectors in the embedding and stacking layer of our DCN.

The Criteo Display Ads dataset predicts the ads click-through rate. It has 13 integer features and 26 categorical features in which each category has a high cardinality. For this dataset, an improvement of 0.001 in logloss is practically significant due to the large input size. A small improvement in prediction accuracy for a large user base can potentially lead to a large increase in a company's revenue. The dataset contains 11GB of user logs from a period of 7 days, which equates to around 41 million records. We used Spark `dataFrame.randomSplit()function` to randomly split the data for training (80%), cross-validation (10%), and the remaining 10% for testing.

DCN was implemented on TensorFlow with Keras. There are four main components in implementing the model training process with DCN:

- **Data processing and embedding.** Real-valued features are normalized by applying a log transform. For categorical features, we embed the features in dense vectors of dimension 6×(category cardinality)1/4. Concatenating all embeddings results in a vector of dimension 1026.
- **Optimization.** We applied mini-batch stochastic optimization with the Adam optimizer. The batch size was set to 512. Batch normalization was applied to the deep network and the gradient clip norm was set at 100.
- **Regularization.** We used early stopping, as L2 regularization or dropout was not found to be effective.
- **Hyperparameters.** We report results based on a grid search over the number of hidden layers, the hidden layer size, the initial learning rate, and the number of cross layers. The number of hidden layers ranged from 2 to 5, with hidden layer sizes ranging from 32 to 1024. For DCN, the number of cross layers was

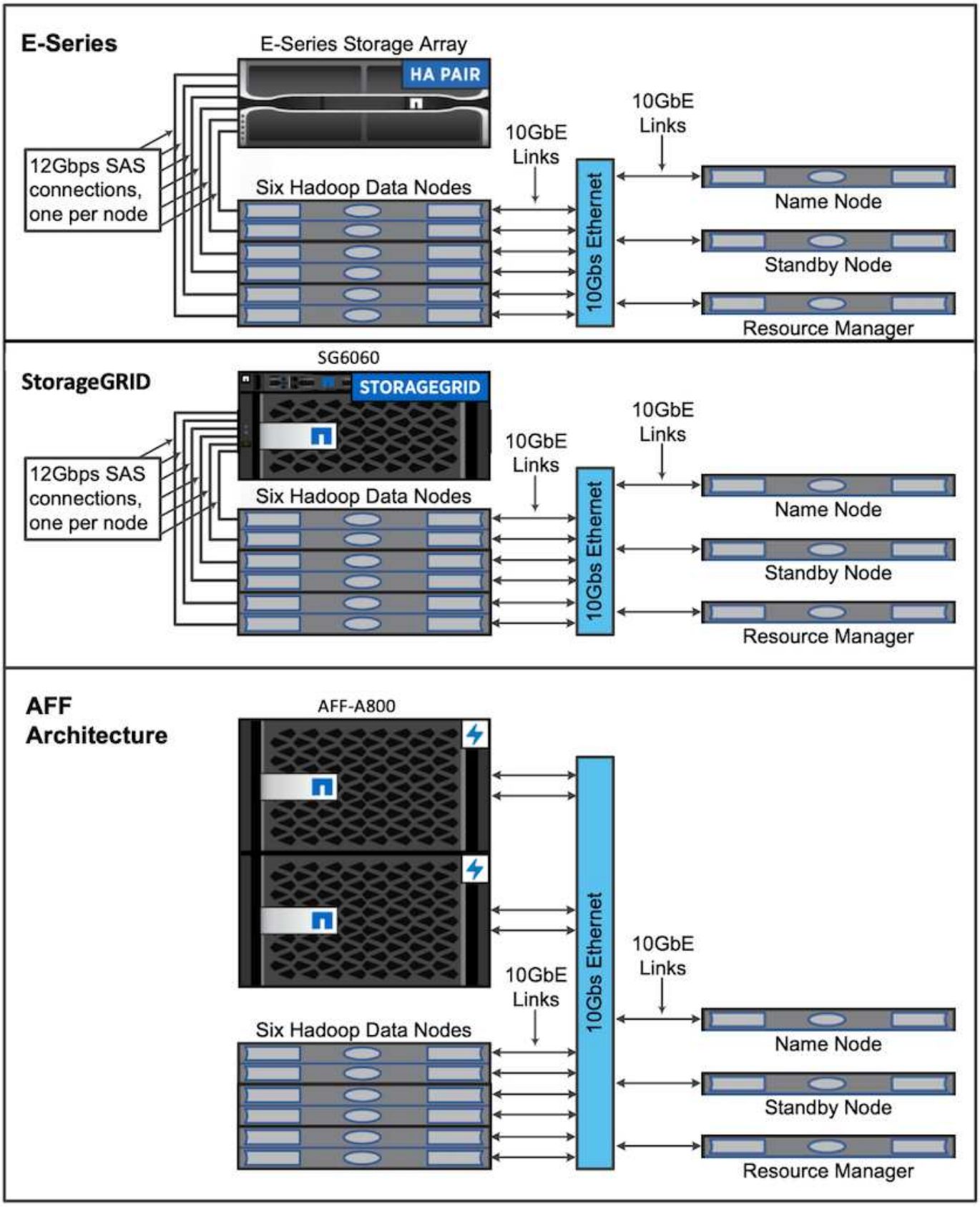
from 1 to 6. The initial learning rate was tuned from 0.0001 to 0.001 with increments of 0.0001. All experiments applied early stopping at training step 150,000, beyond which overfitting started to occur.

In addition to DCN, we also tested other popular deep-learning models for CTR prediction, including DeepFM, xDeepFM, AutoInt, and DCN v2.

Architectures used for validation

For this validation, we used four worker nodes and one master nodes with an AFF-A800 HA pair. All cluster members were connected through 10GbE network switches.

For this NetApp Spark solution validation, we used three different storage controllers: the E5760, the E5724, and the AFF-A800. The E-Series storage controllers were connected to five data nodes with 12Gbps SAS connections. The AFF HA-pair storage controller provides exported NFS volumes through 10GbE connections to Hadoop worker nodes. The Hadoop cluster members were connected through 10GbE connections in the E-Series, AFF, and StorageGRID Hadoop solutions.



Next: Testing results.

Copyright information

Trademark information