Summary of “Applied Machine Learning at Facebook: A Datacenter Infrastructure Perspective”

Hengjia Zhang (hengjia), Jiaqing Ni (jiaqni)

Problem and Motivation

Facebook connects more than two billion people as of December 2017. Meanwhile, the past several years have seen a revolution in the application of machine learning to real problems at this scale, building upon the virtuous cycle of machine learning algorithmic innovations, enormous amounts of training data for models, and advances in high-performance computer architectures. At Facebook, machine learning provides key capabilities in driving nearly all aspects of user experience including services. This diversity has implications at all layers in the system stack. In addition, the sizable fraction of all data stored at Facebook flows presents significant challenges in delivering data, and computational requirements are also intense. Addressing these and other emerging challenges continues to require diverse efforts that span machine learning algorithms, software, and hardware design.

Hypothesis

At Facebook, machine learning should be applied pervasively across nearly all services and the set of machine learning approaches used at Facebook is incredibly diverse. Tremendous amounts of data should be funneled through Facebook’s machine learning pipelines and evaluates new hardware solutions from a performance-per-watt perspective.

Solution Overview

Machine Learning at Facebook

The machine learning flow at Facebook consists of 2 phases shown in Figure 1:

1. Training phase (offline): Build the model.
2. Inference phase (online): Run the trained model in production and make a real-time prediction.

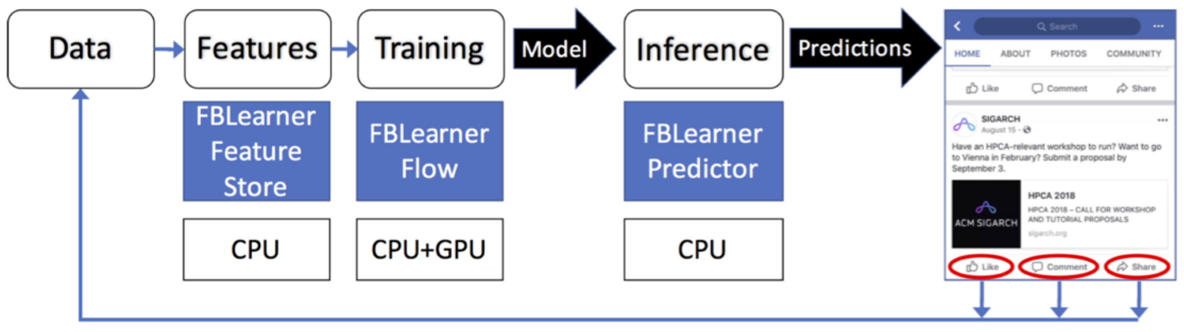


Fig. 1. Example of Facebook's Machine Learning Flow and Infrastructure.

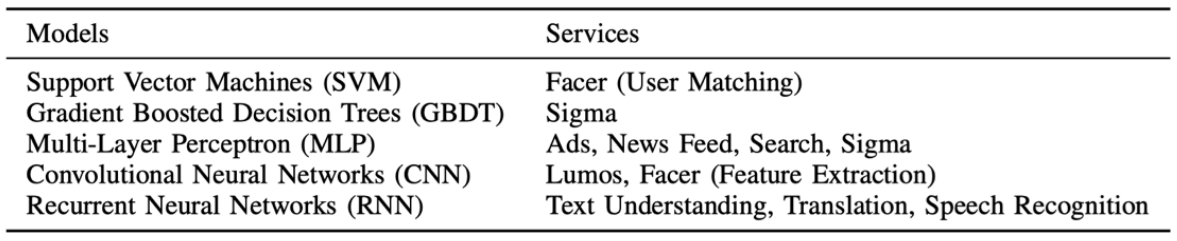
Major Services Leveraging Machine Learning:

The following the major services leveraging machine learning at Facebook:

1. **News Feed** ranking algorithm: Help people see the stories that matter most to them first, every time they visit Facebook.
2. **Ads**: Leverage ML to determine which ads to display to a given user.
3. **Search**: Launch a series of distinct and specialized sub-searches to the various verticals, e.g., videos, photos, people, events, etc.
4. **Sigma**: The general classification and anomaly detection framework that is used for a variety of internal applications including site integrity, spam detection, payments, registration, unauthorized employee access, and event recommendations.
5. **Lumos**: The framework used at Facebook to extract high-level attributes and embeddings from an image and its content, enabling algorithms to automatically understand it.
6. **Facer**: Facebook's face detection and recognition framework.
7. **Language Translation**: The service that manages internationalization of Facebook content. Currently it supports translations for more than 45 languages.
8. **Speech Recognition**: The service that converts audio streams into text.

Machine Learning Model

As said in the Hypothesis part, the machine learning approaches should be diverse. Machine learning algorithms used at Facebook include Logistic Regression (LR), Support Vector Machines (SVM), Gradient Boosted Decision Trees (GBDT), and Deep Neural Networks (DNN). The table below shows different models used in different services:



Models	Services
Support Vector Machines (SVM)	Facer (User Matching)
Gradient Boosted Decision Trees (GBDT)	Sigma
Multi-Layer Perceptron (MLP)	Ads, News Feed, Search, Sigma
Convolutional Neural Networks (CNN)	Lumos, Facer (Feature Extraction)
Recurrent Neural Networks (RNN)	Text Understanding, Translation, Speech Recognition

TABLE I
MACHINE LEARNING ALGORITHMS LEVERAGED BY PRODUCT/SERVICE.

ML-as-a-Service Inside Facebook

Facebook uses FBLearner, Caffe2 and PyTorch to simplify the tasks of leveraging machine learning within Facebook products. We will summarize each of them as follows:

1. FBLearner:

   FBLearner is a suite of three tools, each of which focuses on different parts of the machine learning pipeline. FBLearner leverages an internal job scheduler to allocate resources and schedule jobs on a shared pool of GPUs and CPUs. The three different tools in FBLearner are:

- FBLearner Feature Store: It's a catalog of several feature generators that can be used both for training and real-time predication. It serves as a marketplace that multiple teams can use to share and discover features.
- FBLearner Flow: It is Facebook's machine learning platform for model training, which executes a workflow describing the steps to train and/or evaluate a model and the resources required to do so.
- FBLearner Predictor: It is Facbeook's internal inference engine that uses the models trained in Flow to provide predictions in real time.

2. Caffe2:

   Caffe2 is the product framework choice at Facebook. It focuses on optimization for production, performance and cross-platform support.
3. PyTorch:

   PyTorch is the research framework choice at Facebook. It is not optimized for production and mobile deployments, but it has a frontend that help users flexibly debug and enable rapid experiments.
4. ONNX:

   ONNX is a format to represent deep learning models in a standard way to enable interoperability across different frameworks and libraries. Traditionally, when a research projects produce valuable results and the model needs to be transferred to production, it is always accomplished via rewriting the training pipeline in a product environment with other frameworks. With building ONNX toolchain, Facebook can simplify the transfer process.

Resource Implications of Machine Learning

Summary of hardware Resources at Facebook

Facebook created Big Basin, a latest-generation GPU servers to accelerate their progress of training larger and deep neural networks. It included eight NVIDIA Tesla P100 GPU accelerators.

Before Big Basin, Facebook was using BIG SUr GPU server, which was the first widely deployed, high-performance AI compute platform designed to support NVIDIA M40 GPU.

Resource Implications of Offline Training

Different services leverage different compute resources, where Lumos, Speech Recognition and Language Translation are trained on GPUS, News Feed and Sigma are trained on CPUS, and Facer and Search are trained on both.

In addition, since the data needed for training is increasing at Facebook, hardware limitations may result in an unacceptable increase in overall training latency. Facebook was using distributed training as one solution to overcome these hardware limitations. They found that Ethernet-based networking to be sufficient to provide near-linear scaling capability. With 50G Ethernet NIC, the Big Basin server has allowed Facebook to scale out training vision models without inter-machine synchronization being a bottleneck.

Machine Learning at Datacenter Scale

There're two other concerns besides resource requirements, which are significant data requirement and reliability in the face of natural disasters.

Getting Data to the Models

Facebook decouples the data workload from the training workload, where the data workload is complex, ad-hoc and changing fast and the training workload is usually regular and stable. Facebook develops a data processing machines (readers) to read the data from storage, process them and send to the training machines (trainers).

Leveraging Scale

From diurnal load observation, a large pool of servers are often idle at certain periods in time, which provides an enormous pool of compute resources available during off-peak hours. It provides a prime opportunity to take advantage of distributed training mechanisms that can scale to a large number of heterogeneous resources. Facebook uses scheduler to balance the load properly across heterogeneous hardware and consider the network topology and synchronization cost when training spans multiple hosts as well.

Limitations and possible improvements

For distributed training, Facebook was using data parallelism, which requires synchronizing the gradient descent across all the nodes. Synchronous SGD requires an all-reduce operation, where the bandwidth requirements decrease exponentially with the recursion levels. Alternately, asynchronous SGD is harder and typically done via a shared parameter server. Facebook may improve the staleness of updates and reduce and pressure on parameter servers by designing a hybrid design, where the asynchronous updates happen within super-nodes with high bandwidth and synchronous updates happen across supernodes.

Summary of "Machine Learning at Facebook:

Understanding Inference at the Edge”

Problem and Motivation

As we discussed from the last paper, Facebook is training model offline and inferencing online to reduce latency and become less dependent on network connectivity. The challenge occurs when Facebook tries to enable machine learning inference locally on smartphones and other edge platforms, while the need is necessary since Facebook makes over 90% of its advertising revenue from mobile.

Hypothesis

Facebook tries to run inference on the edge by observing the inside hardware and software system stack the Facebook app is run on, researching the machine learning frameworks and tool sets and try to optimize the performance and efficiency of mobile inference.

Solution Overview

The hardware and software system stack

Facebook does researches on the distribution function of SoC market share and they found that the most commonly-used SoC accounts for less than 4% of all mobile devices. Moreover, the distribution shows an exceptionally long tail: there are only 30 SoCs with more than 1% market share and their joint coverage is only 51% of the market, which means that **There is no standard mobile chipset to optimize for!**

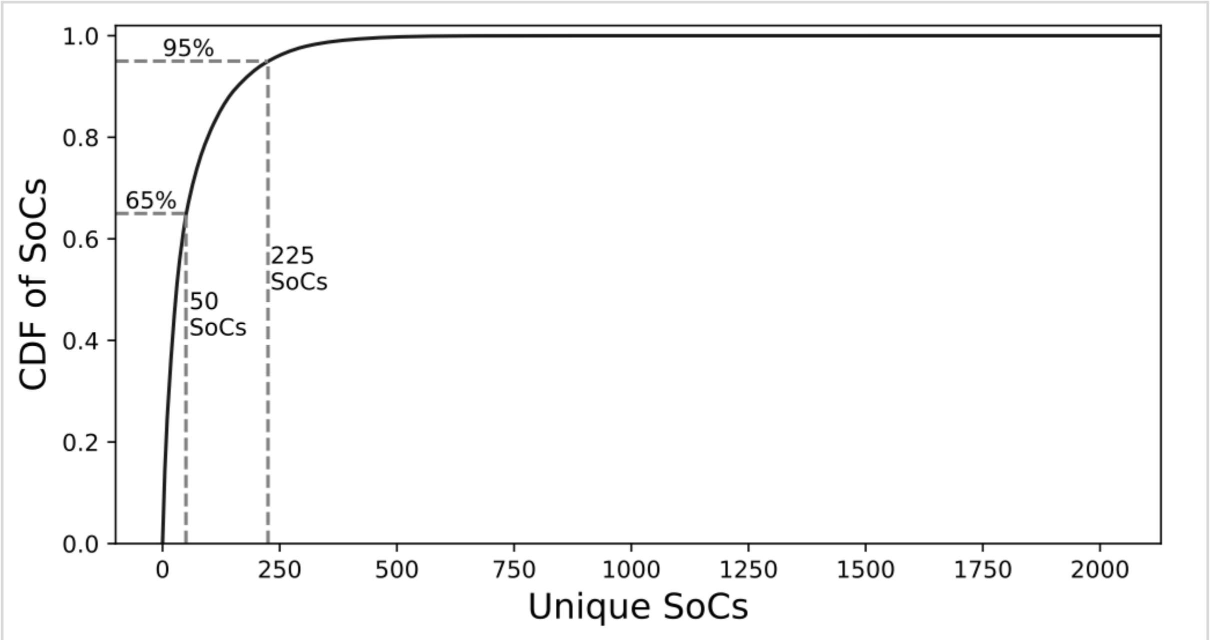


Figure 2: There is no standard mobile SoC to optimize for. The top 50 most common SoCs account for only 65% of the smartphone market.

When looking at the inside of mobile CPUs, Facebook find that mobile CPUs show little diversity: 72% of primary CPU cores being used in mobile devices today were designed over 6 years ago. Cortex A53 represents more than 48% of the entire mobile processors whereas Cortex A7 represents more than 15% of the mobile processors. They also find the design strategies difference between Android and iOS smartphones:iOS devices tend to use fewer, more powerful cores while Android devices tend to have more cores, which are often less powerful.

In addition, the performance difference between a mobile CPU and GPU/DSP is narrow, which can be shown as the figure below:

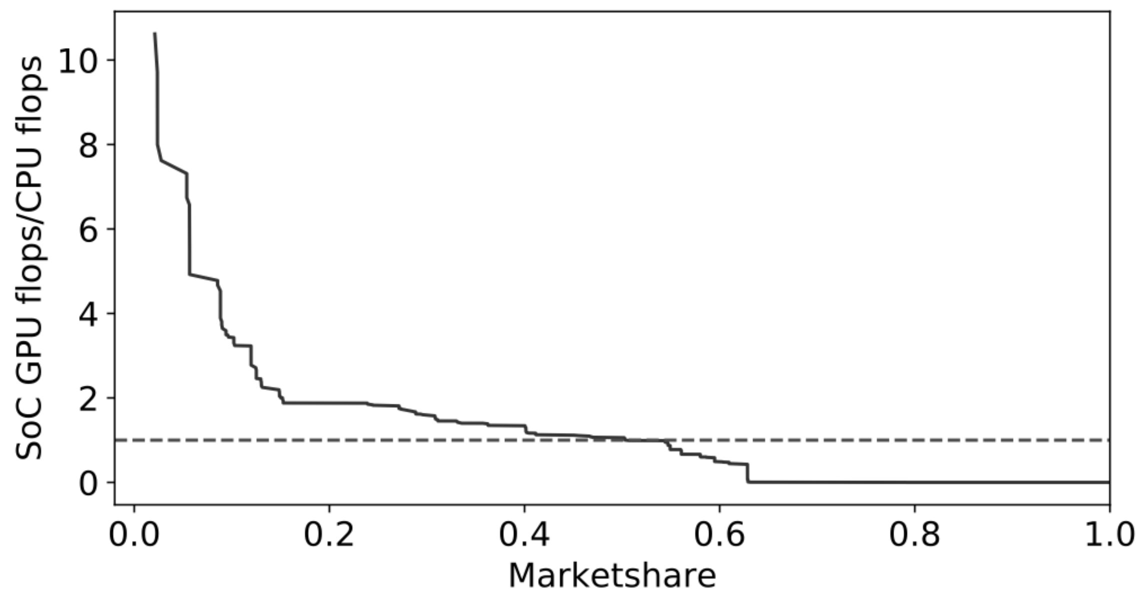


Figure 4: The theoretical peak performance difference between mobile CPUs and GPUs is narrow. In a median Android device, GPU provides only as much performance as its CPU. Only 11% of the smartphones have a GPU that is 3 times more performant than its CPU.

For the available co-processors: DSP and NPU, they are facing the challenge that few of them are avaibable on the SoCs that the Facebook apps run on.

For the software stack, the major API that the Facebook is working on program neural networks on mobile GPUs are OpeCL, OpenGL ES and Vulkan:

1. OpenCL is designed to enable general-purpose programs to run on programmable co-processors and thus doesn't provide graphics specific functionality.
2. OpenGL ES is a trimmed variant of the OpenGL API specifically for mobile and embedded systems: OpenGL 2.0 the first version of the API with a programmable graphics pipeline. OpenGL 3.0 is supported on 83% of Android devices and 3.1 is supported on 52% of Android devices. OpenGL 3.1 introduces compute shaders that provide similar functionalities available in OpenCL 1.x and early versions of CUDA.
3. Vulkan is a successor to OpenGL and OpenGL ES and it provides similar functionality to OpenGL ES 3.1 with a newer API targeted at minimizing driver overhead. It seems like a promising GPUGPUS API.

Machine Learning at Facebook

Machine learning models and frameworks are mostly introduced on the previous paper. Here is just a quick recap: Facebook is using PyTorch for research, Caffe2 for products and

ONNX for the model transferring from research to productions. For mobile inference in particular, Caffe2 is built with optimized mobile inference to deliver the best experience possible for a broad set of mobile devices.

The following is the execution flow of Facebook's machine learning for mobile inference:

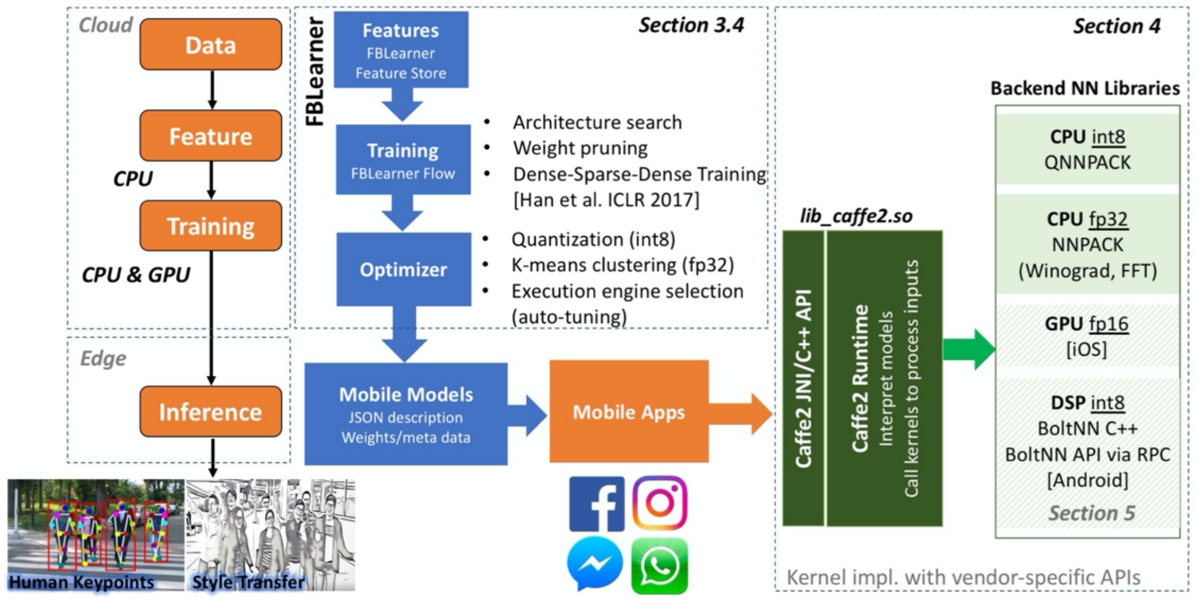


Figure 6: Execution flow of Facebook's machine learning for mobile inference.

1. First, features are collected and selected for any ML modeling tasks from FBLearner Feature Store.
2. Then, a workflow describing architectures of a model and steps for the model training and evaluation is built with FBLearner Flow.
3. After model training and evaluation, the next step is to export and publish the model so it can be served in one of Facebook's production inference tiers.
4. Before models are deployed for edge inference, optimization techniques, such as quantization, can be applied in the Optimizer.

Making Inference on Smartphones:

To make the best possible use of limited computing resources on mobile devices, Caffe2 Runtime integrates two in-house libraries, NNPACK (Neural Networks PACKage) and QNNPACK (Quantized NNPACK) to implements asymptotically fast convolution algorithms.

Limitations and possible improvements:

Currently the majority of mobile inference on run on CPUs and most of they are at least six years old. Maybe Facebook can think ahead and optimize the algorithms for newer CPU.

Also, the programmability is a huge roadblock to using mobile co-processors/accelerators, it maybe helpful to introduce a new framework for this so that the algorithms and hardware can be more easily implemented and integrated. Maybe the integration of PyTorch Mobile and TVM may be helpful in this case.

Summary of Class Discussion

Q: Disaster recovery for less frequently trained services (e.g. Facer, trained once every few years)
A: Most likely multiple data centers maintained copies of the same model, like any other modern databases.

Q: Why favor CPU for certain processes over GPU
A: Too many CPU hardwares (for different purposes) support paralyzing and enable better performance; models that GPU has no overall improvements when preprocessing is significantly time consuming.

Q: Why only CPU for inference?
A: Large amount of idle CPUs during off peak time; data communication over the network is the dominant factor of delay, CPU and GPU doesn't affect overall; quantization CPU is less costly

Q: ONNX covers optimization or users have to do it themselves?
A: ONNX doesn't optimize for the users. The model is only translated between different formats for platforms to be able to train and inference.

Q: Mobile hardware choice between CPU or GPU or DSP or NPU?
A: Majority inference uses CPU because the core processors are not very prevalent on mobile devices. Low performance benefit resulting from high effort of creating these features.

Q: Have PyTorch and ONNX been used in Facebook workflow mobile end inference?
A: PyTorch new release of mobile inference framework. Different flow from FBLearner in this paper.

Q: Mobile end has 2 clusters: high performance and energy efficiency. The paper mentioned using high performance cluster for neural network. Can we use both clusters in a neural network?

A: This is a great project to explore how to effectively apply the two clusters both in a neural network.