Prediction-serving Systems

BY DAN CRANKSHAW AND JOSEPH GONZALEZ

**EXPERT-CURATED
GUIDES TO
THE BEST OF
CS RESEARCH**

Research for Practice combines the resources of the ACM Digital Library, the largest collection of computer science research in the world, with the expertise of the ACM membership. In every RfP column experts share a short curated selection of papers on a concentrated, practically oriented topic.

This installment of Research for Practice features a curated selection from Dan Crankshaw and Joey Gonzalez, who provide an overview of machine learning serving systems. What happens when we wish to actually deploy a machine learning model to production, and how do we serve predictions with high accuracy and high computational efficiency? Dan and Joey's selection provides a thoughtful selection of cutting-edge techniques spanning database-level integration, video processing, and prediction middleware. Given the explosion of interest in machine learning and its increasing impact on seemingly every application vertical, it's possible that systems such as these will become as commonplace as relational databases are today. Enjoy your read!

—Peter Bailis

Machine learning is an enabling technology that transforms data into solutions by extracting patterns that generalize to new data. Much of machine learning can be reduced to learning a model—a function that maps an input (e.g., a photo) to a prediction (e.g., objects in the photo). Once trained, these models can be used to make

predictions on new inputs (e.g., new photos) and as part of more complex decisions (e.g., whether to promote a photo). While thousands of papers are published each year on how to design and train models, there is surprisingly less research on how to manage and deploy such models once they are trained. It is this later, often overlooked, topic that this article addresses.

Before examining the recent work on how to manage and deploy machine-learning models, let's first briefly review the three phases of machine-learning application development: model development, training, and inference.

Model development typically begins with collecting and preparing training data. This data is then used to design new feature transformations and choose from a wide range of model designs (e.g., logistic regression, random forest, convolutional neural network, etc.) and their corresponding training algorithms. Even after a model and training algorithm are selected, there are often additional hyperparameters (e.g., smoothing parameters) that must be tuned by repeatedly training and evaluating the model.

The result of model development is typically a training pipeline that can be run at scale. The training phase executes the training pipeline repeatedly as new data arrives to produce new trained models that can be used to render predictions as part of some application or service.

The final phase of rendering predictions is often referred to as prediction serving, model scoring, or inference. Prediction serving requires integrating machine-learning software with other systems including user-facing application code, live databases, and high-volume data streams. As such, it comes with its own set of challenges

and tradeoffs and is the domain of the emerging class of prediction-serving systems.

While prediction serving has been studied extensively in domains such as ad targeting and content recommendation, because of the domain-specific requirements these systems have developed highly specialized solutions without addressing the full set of systems challenges critical to developing high-value machine-learning applications. Here we have selected four complementary papers, each of which provides practical lessons for developing machine-learning applications, whether you are developing your own prediction-serving system or using off-the-shelf software.

Putting models in the database

MauveDB: Supporting Model-Based User Views in Database Systems. Amol Deshpande and Samuel Madden. In SIGMOD '06; <https://dl.acm.org/citation.cfm?id=1142483>.

MauveDB is an ambitious effort to incorporate machine-learning models into a traditional relational database while preserving the declarativity of SQL-based query languages. MauveDB (for model-based user views, in an homage to a well-known Dilbert cartoon; <http://dilbert.com/strip/1995-11-17>) starts with the observation that the modeling process is fundamentally rooted in data, yet traditional database management systems provide little value for those seeking to create and manage models. The extent of database support for models at the time the paper was written was the ability to use a trained model

as a UDF (user-defined function). This allows users to bring the model to the data but is insufficient for integrating the model into a query optimizer or enabling the database to maintain the model automatically.

MauveDB observes that a model is just a way of specifying a complex materialized view over the underlying training data. The SQL view mechanism was extended to support declaratively specifying models as views that the database engine can understand and optimize. As a result, the database can automatically train and maintain models over time as the underlying data evolves. Furthermore, by integrating the models as views instead of user-defined functions, the query optimizer can use existing cost-based optimization techniques to choose the most efficient method for querying each trained model.

This deep integration of models into the database, however, has some significant limitations. In particular, MauveDB is focused on modeling sensor data and thus considers only two types of models—regression and interpolation—that are widely used in that context. Even for these two relatively simple models, the view definitions become complex to account for all of the available modeling choices. Declaratively specifying models also restricts the user to using only existing database functionality. Any custom preprocessing operations or model specialization must be written as UDFs, defeating the purpose of the tight integration between model and database. Finally, the various access methods and materialization strategies for the optimizer to choose from must be studied and developed separately for each training algorithm. As a result, the addition of new types

of model-based views requires developing new access methods and incremental maintenance strategies, as well as modification to the database engine itself—tasks that ordinary users are typically neither willing nor able to do without significant effort.

The key insight in this paper is that by finding and exposing the semantics of your model to the applications in which they are embedded, you can make your end-to-end machine-learning applications both faster and easier to maintain. But this tight integration comes at the cost of generality and extensibility by making it much harder to change the modeling process or apply these techniques to new domains.

Prediction serving at scale

LASER: A Scalable Response Prediction Platform for Online Advertising. Deepak Agarwal, Bo Long, Jonathan Traupman, Doris Xin, and Liang Zhang. In WSDM '14; <https://dl.acm.org/citation.cfm?id=2556195.2556252>.

The LASER system, developed at LinkedIn, explores a holistic approach to building a general platform for both training and serving machine-learning models. LASER was designed to power the company's social-network-based advertising system but found wide use within the company. The LASER team deliberately restricted the scope of models that it supports—generalized linear models with logistic regression—but took an end-to-end approach to building a system to support these models throughout the entire machine-learning lifecycle. As a consequence,

this paper has many insights that can be applied broadly when developing new machine-learning applications. By restricting the classes of models supported, the authors are able to build all of the techniques they discuss directly into the platform itself. But these same ideas (e.g., those around caching or lazy evaluation) could be applied on a per-application basis on top of a more general-purpose serving system as well. The paper describes ideas for improving training speed, serving performance, and usability.

LASER uses a variety of techniques for intelligent caching and materialization in order to provide realtime inference (these are similar to the view-maintenance strategies discussed in §3.3.2 of the MauveDB paper). The models described in LASER predict a score for displaying a particular ad to a particular user. As such, the model includes linear terms that depend only on the ad or user, as well as a quadratic term that depends on both the user and the ad. LASER exploits this model structure to partially prematerialize and cache results in ways that maximize cache reuse and minimize wasted computation and storage. The quadratic term is expensive to compute in realtime but precomputing the full cross product matching users to ads (a technique described in the literature as full prematerialization) would be wasteful and expensive, especially in a setting such as online advertising when user preferences can change quickly, and ad campaigns frequently start and stop.

Instead, the paper describes how LASER leverages the specific structure of its generalized linear models to prematerialize part of the cross product to accelerate

inference without incurring the waste of precomputing the entire product. LASER also maintains a partial results cache for each user and ad campaign. This factorized cache design is particularly well suited for advertising settings in which many ad campaigns are run on each user. Caching the user-specific terms amortizes the computation cost across the many ad predictions, resulting in an overall speedup for inference with minimal storage overhead. The partial prematerialization and caching strategies deployed in LASER could be applied to a much broader class of models (e.g., neural features, word embeddings, etc.).

LASER also uses two techniques that trade off short-term prediction accuracy for long-term benefits. First, LASER does online exploration using Thompson sampling to explore ads with high variance in their expected values because of small sample sizes. Thompson sampling is one of a family of exploration techniques that systematically trade off exploiting current knowledge (e.g., serving a known good ad) and exploring unknown parts of the decision space (serving a high-variance ad) to maximize long-term utility.

Second, the LASER team adopted a philosophy it calls “*Better wrong than late.*” If a term in the model takes too long to be computed (e.g., because it is fetching data from a remote data store), the model will simply fill in the unbiased estimate for the value and return a prediction with degraded accuracy rather than blocking until the term can be computed. In the case of a user-facing application, any revenue gained by a slightly more accurate prediction is likely to be outweighed by the loss in engagement caused by a web page taking too long to load.

There are two key takeaways from the LASER paper: first, trained models often perform computation whose structure can be analyzed and exploited to improve inference performance or reduce cost; second, it is critical to evaluate deployment decisions for machine-learning models in the context of how the predictions will be used rather than blindly trying to maximize performance on a validation dataset.

Applying cost-based query optimization to deep learning

NoScope: Optimizing Neural Network Queries Over Video at Scale. Daniel Kang, John Emmons, Firas Abuzaid, Peter Bailis, and Matei Zaharia. 2017. In *Proceedings of the VLDB Endowment* 10(11); <https://dl.acm.org/citation.cfm?id=3137664>.

The next paper, from Kang et al. at Stanford, presents a set of techniques for significantly reducing the cost of prediction serving for object detection in video streams. The work is motivated by current hardware trends—in particular, that the cost of video data acquisition is dropping as cameras get cheaper, while state-of-the-art computer vision models require expensive hardware accelerators such as GPUs to compute predictions in realtime for a single video stream.

To reduce this cost imbalance, the authors developed a system called NoScope (<https://github.com/stanford-futuredata/noscope>) to reduce the monetary cost of processing videos by improving model-inference performance. The authors developed a set of techniques to reduce the number of frames on which a costly deep-learning model must be evaluated when querying a video

stream, and then developed a cost-based query optimizer that selects which of these techniques to use on a per-query basis. (Note that in the NoScope work, the use of the term *query* refers to a streaming query to identify the periods of time in which a particular object is visible in the video.) As a result, while NoScope is restricted to the domain of binary classification on fixed-location cameras, it can automatically select a cost-optimal query plan for many models and applications within that domain.

The paper presents two techniques used in combination to reduce the number of frames that require a state-of-the-art model for accurate classification. First, the authors use historical video data for the *specific camera feed* being queried to train a much smaller, specialized model for the query. While this model forgoes the generality of the more expensive model, it can often classify frames accurately with high confidence. The authors use the more expensive model only if the specialized model returns a prediction below a specific confidence threshold. This approach is similar to prior work on model cascades, first introduced by Viola and Jones (<http://lieeexplore.ieee.org/document/990517/>). It also bears some similarities to work on model distillation by Hinton, Vinyals, and Dean (<https://arxiv.org/abs/1503.02531>), although in the case of distillation the goal is to train a cheaper model to *replace* the more expensive one rather than supplement it.

NoScope combines these specialized models with a technique called *difference detectors*, which exploit the temporal locality present in fixed-angle video streaming to skip frames altogether. If the difference detectors find

that the current frame is similar enough to an existing frame that has already been labeled, NoScope skips inference completely and simply uses the label from the previously classified frame. NoScope uses a cost-based optimizer to select the optimal deployment for a particular video stream, query, and model from the set of possible specialized model architectures and difference detectors.

NoScope's key insight is the identification of a domain-specific structure that can be exploited to accelerate inference in a range of settings within that domain. While the specific structure NoScope leverages is limited to fixed-location object detection, identifying temporal and spatial redundancy to reduce the load on expensive state-of-the-art models has the potential to be exploited in many different prediction-serving settings.

A general-purpose prediction-serving system

Clipper: A Low-latency Online Prediction-serving System.

Daniel Crankshaw, Xin Wang, Giulio Zhou, Michael J. Franklin, Joseph E. Gonzalez, and Ion Stoica. In NSDI'17; <https://dl.acm.org/citation.cfm?id=3154630.3154681>.

The last paper included here describes the Clipper (<http://clipper.ai/>) prediction-serving system. Rather than making any assumptions or restrictions on the types of models that can be served as the previous papers did, Clipper starts with the design goal of easily serving any trained model at interactive latencies. From this starting point the paper explores techniques for optimizing both inference

performance and accuracy, while encapsulating the models in a uniform, black-box prediction interface.

To support the uniform prediction interface, Clipper adopts a modular, layered architecture, running each model in a separate Docker container and interposing an intermediate layer between the models and the querying applications. This distributed architecture enables the system to serve models with conflicting software and hardware requirements at the same time (e.g., serving models written in different programming languages running on a mix of CPUs and GPUs). Furthermore, the architecture provides process isolation between different models and ensures that a single model failure does not affect the availability of the rest of the system. Finally, this disaggregated design provides a convenient mechanism for horizontally and independently scaling each model via replication to increase throughput.

Clipper also introduces *latency-aware batching* to leverage hardware-accelerated inference. Batching prediction requests can significantly improve performance. Batching helps amortize the cost of system overheads (e.g., remote procedure call and feature method invocation) and improves throughput by enabling models to leverage internal parallelism. For example, many machine-learning frameworks are optimized for batch-oriented model training and therefore capable of using SIMD (single instruction, multiple data) instructions and GPU accelerators to improve computation on large input batches. While batching increases throughput, however, it also increases inference latency because the entire batch must be completed before a single prediction is returned.

Clipper employs a latency-aware batching mechanism that automatically sets the optimal batch size on a per-model basis in order to maximize throughput, while still meeting latency constraints in the form of user-specified service-level objectives.

To improve prediction accuracy, Clipper introduces a set of *selection policies* that enable the prediction-serving system to adapt to feedback and perform online learning on top of black-box models. The selection policy uses reward feedback to choose between and even combine multiple candidate models for a given prediction request. By selecting the optimal model or set of models to use on a per-query basis, Clipper makes machine-learning applications more robust to dynamic environments and allows applications to react in realtime to degrading or failing models. The selection policy interface is designed to support ensemble methods (https://en.wikipedia.org/wiki/Ensemble_learning) and explore/exploit techniques that can express a wide range of such methods, including multiarmed bandit techniques and the Thompson sampling algorithm used by LASER.

There are two key takeaways from this paper: the first is the introduction of a modular prediction-serving architecture capable of serving models trained in any machine-learning framework and providing the ability to scale each model independently; the second is the exploitation of the *computational structure* of inference (as opposed to the mathematical structure that several of the previous papers exploit) to improve performance. Clipper exploits this structure through batching, but there is potential for exploiting other kinds of structures,

particularly in approaches that take more of a gray- or white-box approach to model serving and thus have more fine-grained performance information.

Emerging systems and technologies

Machine learning in general, and prediction serving in particular, are exciting and fast-moving fields. Along with the research described in this article, commercial systems are actively being developed for low-latency prediction serving. TensorFlow Serving (<https://www.tensorflow.org/serving/>) is a prediction-serving system developed by Google to serve models trained in TensorFlow. The Microsoft Custom Decision Service (<https://azure.microsoft.com/en-us/services/cognitive-services/custom-decision-service/>), with accompanying paper (<https://arxiv.org/abs/1606.03966>), provides a cloud-based service for optimizing decisions using multiarmed bandit algorithms and reinforcement learning, with the same kinds of explore/exploit algorithms as the Thompson sampling used in LASER or the selection policies of Clipper. Finally, Nvidia's TensorRT (<https://developer.nvidia.com/tensorrt>) is a deep-learning optimizer and runtime for accelerating deep-learning inference on Nvidia GPUs.

While the focus of this article is on systems for prediction serving, there have also been exciting developments around new hardware for machine learning. Google has now created two versions of its TPU (Tensor Processing Unit) custom ASIC. The first version, announced in 2016, was developed specifically to increase the speed and decrease the power consumption of its deep-learning inference workloads. The TPUv2, announced

in 2017, supports both training and inference workloads and is available as part of Google's cloud offering. Project Brainwave (<https://www.microsoft.com/en-us/research/blog/microsoft-unveils-project-brainwave/>) from Microsoft Research is exploring the use of FPGAs (field-programmable gate arrays) to perform hardware-based prediction serving and has already achieved some exciting results demonstrating simultaneously high-throughput and low-latency deep-learning inference on a variety of model architectures. Finally, both Intel's Nervana chips and Nvidia's Volta GPUs are new, machine-learning-focused architectures for improving the performance and efficiency of machine-learning workloads at both training and inference time.

As machine learning matures from an academic discipline to a widely deployed engineering discipline, we anticipate that the focus will shift from model development to prediction serving. As a consequence, we are anxious to see how the next generation of machine-learning systems can build on the ideas pioneered in these papers to drive further advances in prediction-serving systems.

Dan Crankshaw *is a Ph.D. student in the UC Berkeley CS department working in the RISELab. After cutting his teeth doing large-scale data analysis on cosmology simulation data and building systems for distributed graph analysis, he turned his attention to machine learning systems. His current research interests include systems and techniques for serving and deploying machine learning, with a particular emphasis on low-latency and interactive applications.*

Joseph Gonzalez is an assistant professor at UC Berkeley and co-director of the UC Berkeley RISELab where he studies the design of algorithms, abstractions, and systems for scalable machine learning. Before joining UC Berkeley, Joseph co-founded Turi Inc. [formerly GraphLab] to develop AI tools for data scientists and later sold Turi to Apple. Joseph also developed the GraphX framework in Apache Spark and is a member of the Apache Spark PMC. Joseph received his Ph.D. in Machine Learning from Carnegie Mellon University.

Copyright © 2018 held by owner/author. Publication rights licensed to ACM.