Bayesian Dropout Approximation (BDA) Engine design

Overview

The BDA Engine acts as a back end for modeling and analysis of data stored in the CESMII Smart Manufacturing Information Platform (SMIP). It is meant to be deployed as a cloud service, running on a Kubernetes cluster. An instance of the Service and the K8 cluster on which it runs is expected to be single-tenant and created by an individual instance of a third-party app (TPA) that wishes to make use of the Service's modeling capabilities and which already interacts with the SMIP. An instance of the Service will receive and fulfill requests through a RESTful interface, and will access data from the SMIP through the SMIP's GraphQL interface, using a bearer token provided by the user.

In response to requests, the Engine operates in two modes: training mode and inference mode.

In training mode, the Service will receive and authenticate requests from the TPA to train a model based on a specified set of data from the SM Platform. The Service will then fetch the data and use it to train a model, and return an encapsulated data package called a Model Kernel Object (MKO), which represents all the information necessary to compute predictions with the model and is expected to be retained by the TPA. On-Service caching or remote storage of the MKO is a possible future feature.

In inference mode, the Service will receive and authenticate requests from the TPA to use a provided MKO and provided model inputs to compute samples of possible model outputs. These samples can then be returned to the TPA or used in a variety of ways by the Service's "post-processing" toolset, with the resulting analysis being returned to the TPA. The tools in this toolset will include capabilities to create histogram images, form probability distribution functions (PDFs), create statistical summaries, integrate certain quantities over PDFs, and visualize model predictions.

Design Considerations

- TPAs may have high latency or intermittent connections to their instance of the BDA Service.
- The SM Platform service is highly available, low latency, and GraphQL based.
- Instances of the BDA Service are likely not persistent for the useful lifetime of a trained model.
- A typical training dataset from the SM Platform service may be large, 10^5 - 10^6.
- The typical number of degrees of freedom in an MKO is order of magnitude 10^3 - 10^4.

- Post training, internal data structures in the Service will be a few hundred MB at the outside, typically much smaller.
- Training task times will be from seconds to multiple hours.
- Inference task times will be from seconds to multiple days of processor time, with typical values being in tens of seconds.
- MKOs should embed provenance and metadata describing what system it represents and which data it is derived from.

Approach

The Service is implemented as a hybrid microservices architecture within a Kubernetes cluster, likely running on a cloud service.

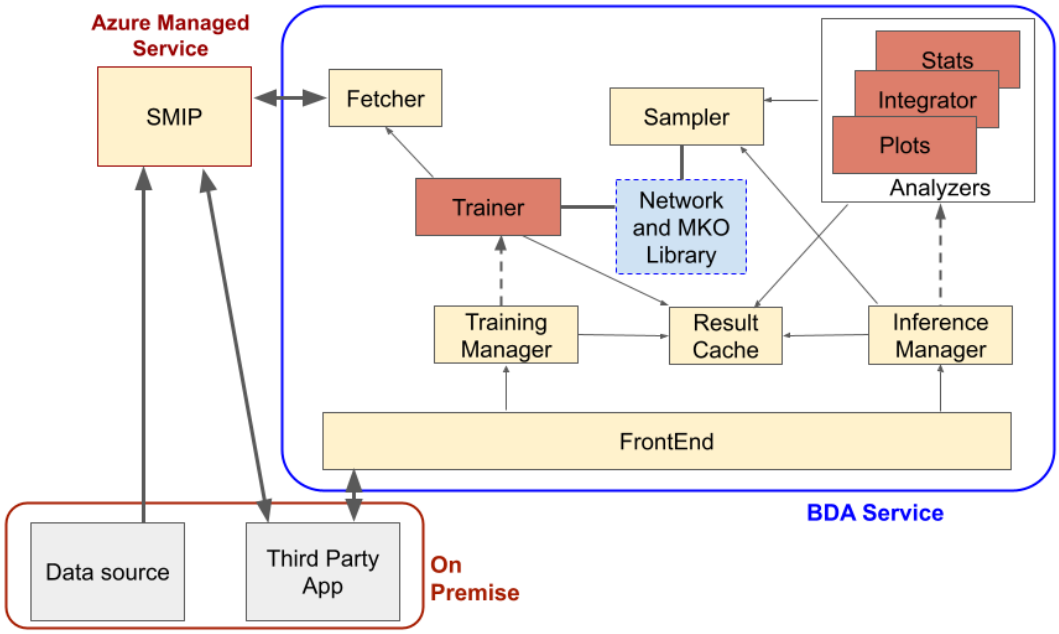

Figure 1: Architecture of BDA Service and external points of contact. Microservices within the BDA Service may be scaled out to meet demand. Solid arrows indicate RESTful web APIs, dotted arrows indicate separate executables created via system call, and solid thick lines indicate posix function calls to libraries.

External communications

Requests to the BDA Service are made to the FrontEnd via REST API at the Kubernetes ingress point, with java web tokens (JWTs) used to manage ongoing authentication and service continuity. Many types of requests will require long-running operations to return results. In these cases, a token, an expected wait time, and an expiration date will be returned, which can be used to retrieve the forthcoming result.

Communication with the SMIP will be via its GraphQL interface. Authentication will be via a JWT provided in the TPA's request.

Internal communications

Communications between microservices internal to the Service will be via REST APIs, with two major exceptions. First, Trainers and Analyzers are executed via system call and given control parameters via standard input. Second, as the neural network functionality is be provided by the TensorFlow 2.x framework, TensorFlow functions are called directly by the Trainer service and by the Sampler service. Subsequently, the TensorFlow data structures are encoded into and decoded from MKOs directly in memory.

Individual Services

FrontEnd - REST API. The FrontEnd service listens on the cluster's exposed ingress port and takes HTTPS requests. It handles logins and a user database, with permissions and access lists. After authentication and permission validation, FrontEnd forwards requests to the Training Manager, the Inference Manager, or the Results Cache to handle a request.

Training Manager - REST API. The Training Manager service sanity checks requests for training a model, determines model hyperparameters, makes estimates for time to complete, and reports errors. It creates instances of Trainers and handles returning MKOs for both short and long operation results. It is responsible for retrieving long-operation MKOs from the Results Cache and returning them to the TPA.

Trainer - POSIX executable. Instances are created by Training Manager to instantiate a TensorFlow network and train the network parameters. Employs Fetcher to get training data. Posts MKOs to the Results Cache service and exits.

Fetcher - REST API. Fills requests to retrieve data from SM Platform.

Result Cache - REST API. Accepts various results from the Trainer and the Generators and holds them until retrieved by Training Manager or Inference Manager or until the expiration date.

Inference Manager - Rest API. The Inference Manager service sanity checks requests for inference with a model, makes estimates for time to complete, and reports errors. It creates an instance of the appropriate Generator via system call, passing it an array of MKOs and control parameters, and handles returning inference results for both short and long operation results. It is responsible for retrieving long-operation inferences from the Result Cache and returning them to the TPA.

Generators - Posix executables. A set of analysis tools that all operate on model samples. Samples are obtained by passing an MKO and control parameters to Sampler. There may be multiple calls to multiple Samplers. Responsibly written Generators release their Samplers when they are done.

Sampler - REST API. Fills requests for obtaining samples, by creating and initializing a network to the state represented by an MKO. Session based, so that a network can be built and persist across multiple requests for samples.

Assumptions and Risks

Communication of data specification - There must be a data specification, a path to the piece of equipment and the appropriate time series of data, available to the TPA and communicable to Fetcher. This requirement is implemented via Attribute IDs in the SMIP.

The FrontEnd is not a bottleneck - Although kubernetes can duplicate the frontend service, because the frontend also provides authentication via an internally provided database, replication is ill-advised. If the frontend does prove to be a bottleneck, a separate database service will have to be created or subscribed to, likely in the form of a containerized MongoDB service incorporated into the architecture.

Low latency and availability from the TPA cannot be assumed. The overhead associated with the Results Cache is in response to the lack of a good connection to the TPA.