CTI Project: Deep Networks as a Semantic Platform for Modeling User Behavior Data

Report: Final Results

Thomas Hofmann, ETH Zurich
Andreas Tschofen, 1plusX AG

October 18, 2019

1 Changes to Milestone 3+4

As requested in the last intermediate report, there has been a proliferation of general platforms for deep learning and specific libraries for learning embeddings. Platforms like TensorFlow[1] (Google) have become very popular and a *de facto* standard in research, but also in many industries, including digital marketing (most relevant for 1plusX). The Gensim library[2] is very popular in the area of language/text understanding and includes very efficient implementations of embedding algorithms. We have thus decided to build the required functionality on top of these frameworks. This has alleviated us of the need to define our own proprietary specification language (originally Milestone 3), which is also an advantage for customers who nowadays have data scientist trained in Python, Tensorflow, Gensim etc., and which can use our model interface without further training. This piggy-backing has already shown success with customers and becomes part of a now popular strategy to offer APIs into Data Management Platforms (DMPs).

2 Background

Let us briefly sketch some background on how the 1plusX predictive DMP works and how it uses machine learning and specifically embeddings. A sketch of the overall architecture is shown in Figure **??**. Fundamentally, URIs are extracted from interactions of users with Web pages that they have visited or items that have interacted with. These URIs are filtered (item candidates, items) and finally mapped to a latent embedding space, where each URI is represented by

[1] `https://www.tensorflow.org/`
[2] https://pypi.org/project/gensim/

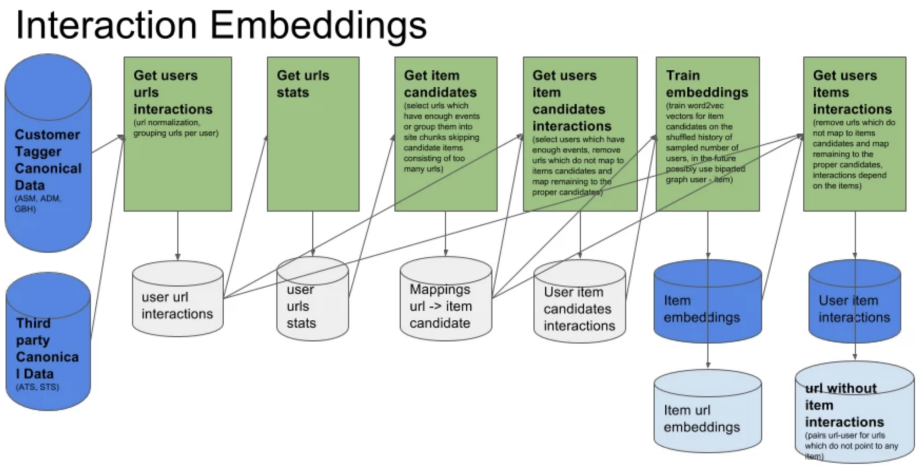

Figure 1: Data architecture for computing item embeddings.

a Euclidean vector, uri $\mapsto \mathbf{x}_{\text{uri}}$. Using aggregation, most basically averaging, user embeddings are created

$$\text{u} \mapsto \left[\sum_{\text{uri visited by u}} \mathbf{x}_{\text{uri}} \right] \Big/ \#\{\text{uris visited by user}\} \tag{1}$$

The input to the ML subsystem for embeddings is thus the set of events per userID that has been seen in the last 60-90 days. The collection of items (URIs) per user is considered as a document that describes the user. And the collection of all the users is considered as a document collection. This allows us to treat the URIs as words in a language model and use results from deep representation learning techniques as applied to Natural Language Processing (NLP). Using these models, we are able to learn a robust representation (embeddings) of both items and users which satisfy various desirable properties such as a complementarity, supplementarity, and compositionality.

The item embedding computation module returns item embedding vectors of a configurable pre-specified dimensionality. This module is built from components such as word2vec, Gensim or Tensorflow. This allows scaling-up to hundreds of thousands of items. In the event that there are more items, this can be handled via parallel execution of embedding computations (offered in Tensorflow for example) or by downsampling items accordingly by thresholding on frequency.

	gensim_25	tf_mini_batch_25	tf_optimized_25	gensim_50	tf_mini_batch_50	tf_optimized
runtime(seconds)	131	297	165	276	628	330
loss_validation	10.10	11.06	10.89	10.42	11.49	11.29

Table 1: Comparison of performance and accuracy between models trained with Gensim and Tensorflow. For evaluation a standard internal data set of user-page visits has been used.

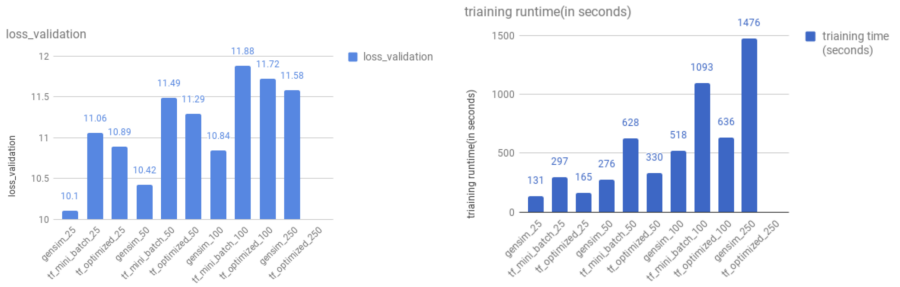

Figure 2: Comparison of accuracy/loss and runtime between models trained with Gensim and Tensorflow on different data set sizes.

3 Milestone 4

3.1 Comparative Evaluation

We have built major functionality into an existing Gensim implementation at 1plusX for learning embeddings for items and users. In the context of this project, we have also build from scratch a Tensorflow implementation of all machine learning models and algorithms, resulting in a more or less equivalent system. We have benchmarked different Gensim and different Tensorflow implementations to learn embeddings. The key metrics are summarized in Table **??**. One observes that Gensim has slight advantages in both, performance and final accuracy. However, the Tensorflow implementation comes close and has the advantage to be much more extensible as the framework is more powerful and flexible. We have drilled down a bit deeper and looked at different data set sizes (more "words" = visited URLs). Results in terms of accuracy and runtime are shown in Figure **??**. Details on the data set preparation are shown in Table **??**. As one can see these data sets are quite big, ranging up to about 60M events.

	gensim_25	tf_mini_batch_25	tf_optimized_25	gensim_50	tf_mini_batch_50	tf_optimized_50	gensim_100	tf_mini_batch_100	tf_optimized_100	gensim_250
#total words	15153544	15153544	15153544	30249286	30249286	30249286	60451791	60451791	60451791	151033284
#unique words	724927	724927	724927	1063129	1063129	1063129	1582097	1582097	1582097	2643524
#vocabulary	62441	62441	62441	95880	95880	95880	146939	146939	146939	250330
words/sec	204036	around 90000	around 170000	203404	around 85000	around 180000	214461	around 85000	around 190000	204969
training_loss	NA	around 4	NA	NA	3.52	NA	NA	around 4	NA	NA
training_steps	NA	16142	16224	NA	33238	33417	NA	16117	68279	NA
#sentences	196480	196480	196480	391747	391747	391747	782650	782650	782650	1955801

Table 2: Characteristics and dimensions of different data sets used in the experiments.

3.2 Tensorflow Implementation

We have built upon the Tensorflow programming environment shown in Figure **??**. In a first step, (1) we encapsulate the training data, which gets extracted from the logs in a form that is digestible for Tensorflow. Then one can use a simple jupyter notebook to (2) load pre-trained embedding models, (3) train and evaluate pre-made tensorflow estimators (e.g. classifiers) as a baseline, (4) build and train custom estimators with tf.keras, (4) export and deploy the models to the Google Cloud ML Engine, and (5) request inferences for single samples or for a batch from the cloud, using python or the command line.

This provides a powerful extension to 1plusX offering as it opens-up data and foundational models (i.e. embeddings, pre-trained representations) to the Tensorflow ecosystem. The above pipeline has been tested and validated internally at 1plusX over various data sets from different existing customers. The internal library is called TFplusX and a workflow demo can be found online at this public Github URL.

4 Milestone 5

We have worked with an 1plusX customer, who does not come from the core segment of publishers and media resellers and who has a non-standard set of use cases and who needs custom models to be adapted to their needs.[3] This customer was interested in age, gender, and interest prediction, audience expansion (finding users similar to a given core group), use of topic-based targeting, used two standard activation channels, and required custom events.

We have pursued the approach outlined above and made processed data as well as embedding models and classifiers available as outlined above. The customer ran a pilot project in late 2018. This pilot was deemed successful and the customer has continued towards productionizing the setup. It has not yet made use of the full Tensorflow capabilities due to legacy systems, but we expect the pilot customer as well as other customers to move in a direction that makes

[3]The customer identity is not publicly known as hence can not be disclosed, but it is a large financial institution.

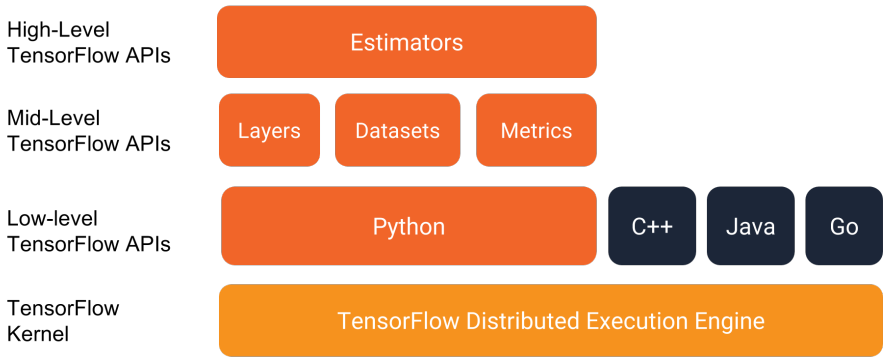

Figure 3: Tensoflow programming environment

more use of the Tensorflow platform in the future (see also implementation plan).