Deepak Dilipkumar
@deepakdilipkumr
Machine Learning Engineer

Justina Chen
@jtinachen
Product Manager

Only on Twitter
@Twitter
#OnlyOnTwitter

Infrastructure

A SplitNet architecture for ad candidate ranking

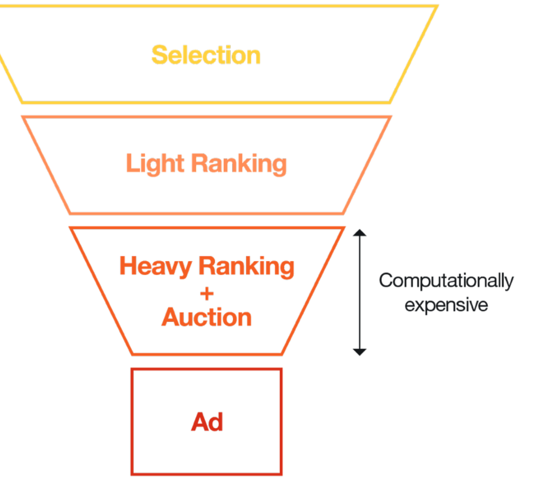
By Deepak Dilipkumar and Justina Chen
Tuesday, 25 June 2019

Twitter connects advertisers to customers by delivering the most relevant ads to their target audience. Whenever users refresh their timeline, browse another user's profile, or search for a Tweet, our ad serving system receives ad requests. Over 100 million people use the platform each day, so our ad serving system is designed to handle a huge amount of real-time traffic at all times. As our business has expanded, we've faced new challenges of operating our ad server at growing scale while maintaining reliability and improving performance.

For each incoming ad request, we must evaluate hundreds of thousands of potential ad candidates to identify the best ad to serve to the user. We do this through two main phases: candidate selection and candidate ranking. In the first phase, candidate selection, we identify a subset of all ad candidates that satisfies the advertisers' targeting criteria such as the user's age, gender, interests, and so on.

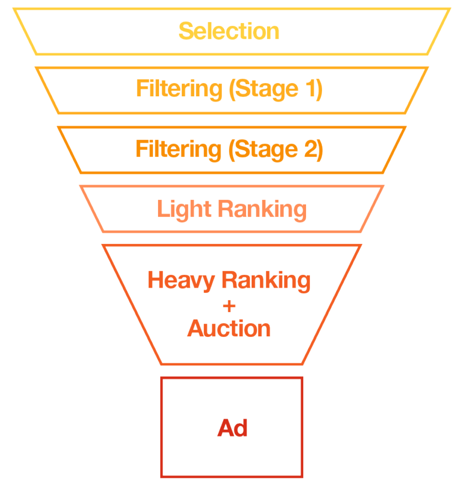
The second phase, candidate ranking, ranks these ads to find the most relevant ones for the user. The candidate ranking stage initially had just two substages: light ranking and heavy ranking. In light ranking, a computationally efficient ML model ranks all the eligible ads from the candidate selection phase. The top-K ad candidates from the light ranking stage are passed to the heavy ranking stage, where more expensive computations are performed to select the best ad.

This entire pipeline is described in detail in a previous blog post; the high-level diagram is shown below.



Motivation

As our advertising business grew and we evolved other parts of the pipeline, it became too expensive to evaluate all the eligible ad candidates even at the light ranking stage. In response, we patched the system with additional filtering stages before light ranking to reduce computational costs. With the patches we introduced, the pipeline looked like this:



We introduced two additional filtering stages, which optimized for system efficiency by eliminating low-value candidates earlier in the pipeline using simple heuristics. We refer to this entire stack, consisting of the two heuristic filters and the ML-based light ranking model, as "early filtering."

However, using less robust heuristics inevitably led to some high-value ad candidates being incorrectly eliminated before the heavy ranking stage. Additionally, the substages were introduced sequentially over time, so we unintentionally ended up with a system where the substages did not align with each other. Even if each stage selected the locally best ads, the overall system may not identify the globally optimal ad due to the unpredictable and complex interactions of the early filters. As a result, there was a lot of room to consolidate and improve the early filtering stack.

The light ranking stage within the early filtering stack itself also needed improvement. The goal of the light ranking model is to approximate the heavy ranking model as closely as possible while using a simpler architecture and fewer features. It used a simple logistic regression model to achieve this, built using an in-house machine learning framework called Lolly. While this approach was efficient and scalable, we could not experiment with state-of-the-art model architectures or novel loss functions.

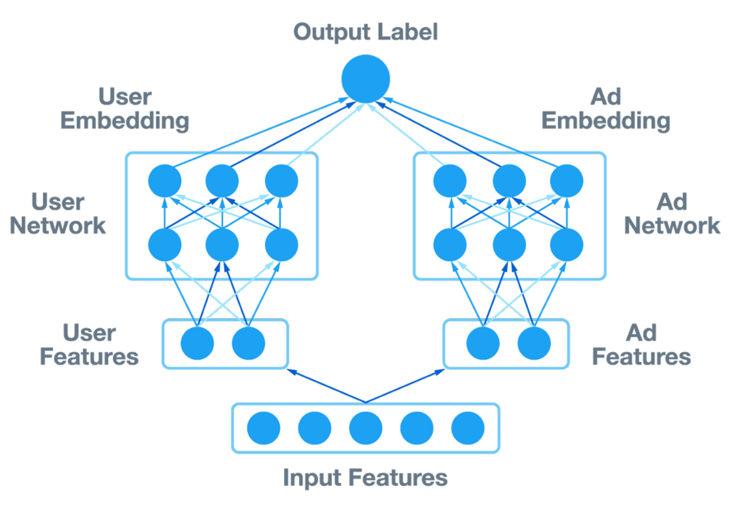
Last year, we shipped multiple improvements to these early filtering stages which significantly improved ad relevance. We were confident in huge potential improvements in this part of our system, but the existing piecemeal infrastructure encumbered progress. To capture further improvements, we had to ask ourselves one question: How would early filtering work if we could redesign it from scratch?

Modeling Improvements

Using Lolly, our in-house ML framework, for the light ranking stage, we were limited to a very small set of supported models (primarily linear and logistic regression). Additionally, because Lolly does not support automatic differentiation, we had to manually compute the gradients for any new loss function experiments. We decided to switch to the new ML framework at Twitter — DeepbirdV2, a wrapper for TensorFlow that interfaces smoothly with Twitter's existing data pipelines. This change created the freedom for us to experiment with any models and loss functions we wanted to try, improving both performance and productivity.

However, we could not simply replace the existing logistic regression model with a deep neural network. We have a tight latency budget to work with when serving real traffic, so as to ensure that users are shown the ads most relevant to them as soon as they open their timeline. While using a deeper model might deliver model performance gains in offline simulations, it would also lead to increased latency due to slower inference, and thus those gains would disappear in production. Instead, we needed a model architecture that would simultaneously allow for novel experimentation and efficient inference.

Due to the sparse nature of a lot of Twitter's data, embeddings that transform input features into a compressed representation are proving increasingly effective for Twitter's ML products. We concluded that an embedding-based architecture was ideal for our light ranking use case as well. Specifically, we learn an embedding for each user and ad with the following architecture:



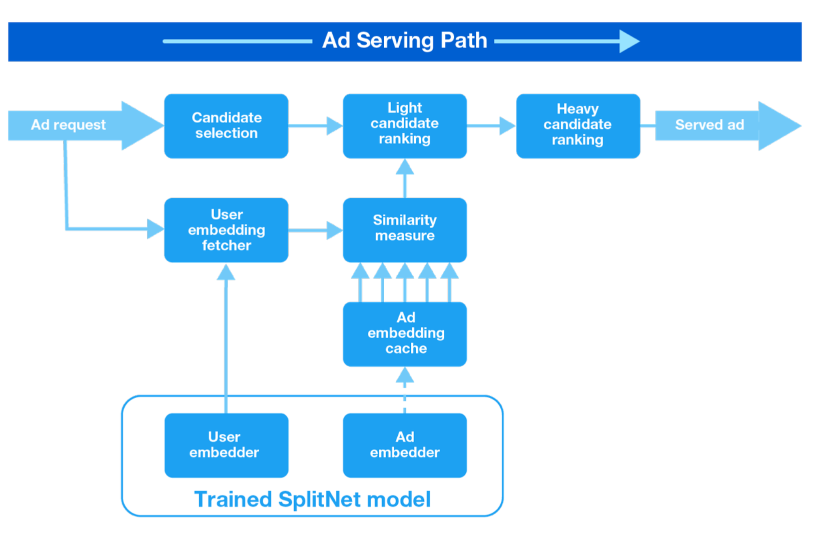
We call this a "SplitNet" model. It takes the full set of input features used in the previous model and splits them into two branches — user/request side features and ad side features. User/request features include properties of the user and device that the ad is served to, such as user interests, device ID, and time of day. Ad features include properties of the ad candidate under consideration, such as the ID of the advertiser who is serving the ad and the format of the ad. Splitting the features this way gives us computational benefits at serving time, which we discuss further in the "System Improvements" section.

The previous model used additional features that were dependant on both the user and ad, such as the past engagement history of this user with this or similar ads. These interaction features did not fit into either branch, and so we discarded them after verifying that removing them had no significant impact on model quality under the new architecture.

User and ad features are fed into separate deep neural networks, resulting in dense user and ad embeddings. These embeddings are combined to calculate the final model output, a score representing the probability of the user engaging with this particular ad. The score can be calculated in different ways, but we chose to simply take the dot product of the embeddings, which is both efficient and gives good results empirically. The user and ad embeddings are then learned jointly using a logistic loss function, with labels obtained directly from the downstream heavy ranking model.

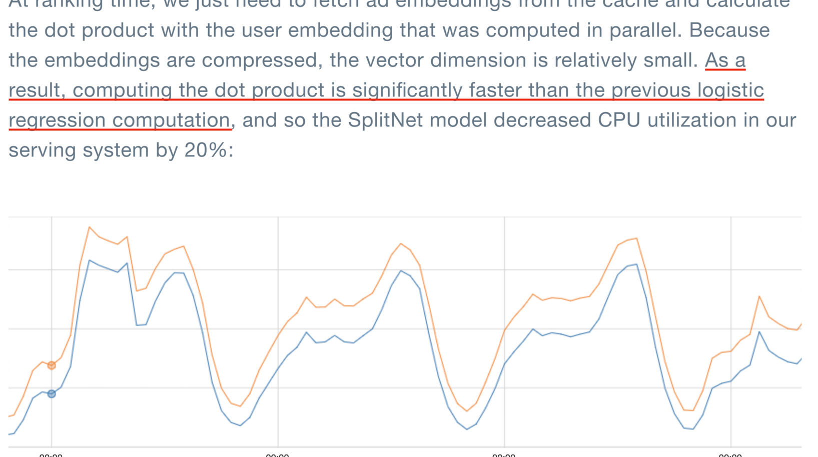
System Improvements

At serving time, the SplitNet model requires us to compute a user embedding and an ad embedding, and generates a score for the user/ad pair calculated as the dot product of these embeddings. Initially, we might expect these calculations to add latency to the system compared to the previous logistic regression model, as the dense embeddings are computed by passing user and ad features through a neural network. Fortunately, we already have all the required user features from the onset, and so we can begin computing the user embedding before starting the candidate selection phase. Therefore, we compute the user embedding in parallel with candidate selection, creating no additional latency to the overall system.



Additionally, instead of computing each ad candidate embedding at serving time, we continuously compute and cache these embeddings, and then simply look them up during serving. Whenever we find an ad candidate without an embedding in the cache, we compute and store it so that the embedding is immediately available for the next request that requires it

At ranking time, we just need to fetch ad embeddings from the cache and calculate the dot product with the user embedding that was computed in parallel. Because the embeddings are compressed, the vector dimension is relatively small. As a result, computing the dot product is significantly faster than the previous logistic regression computation, and so the SplitNet model decreased CPU utilization in our serving system by 20%:



The system performance gain allowed us to remove one of the heuristic filters in our early filtering stack, making the overall system easier to understand and improve. Furthermore, since we migrated the bulk of the model's computation outside of the request path to a standalone service, we now have the ability to introduce more complex ML models without concerns of memory usage or latency.

Future Directions

SplitNet was designed as an investment in modeling infrastructure that maximizes productivity. The launch opens up a number of opportunities for additional improvements to the ads ranking funnel.

Relevance score: One immediate opportunity involves replacing our cosine similarity-based embedding aggregation with an additional linear layer that learns elementwise weights while computing the dot product. This overparametrization technique has shown significant model quality gains in offline and online experiments.

Model training: The current SplitNet model is trained on uniformly sampled light model candidates that are labeled by the heavy ranking model. While this ensures that our training and test distributions for the light ranking model are consistent, in practice most of the training data consists of low-value candidates that can overwhelm the model. We have access to an alternate data source, consisting of the high-value candidates that were allowed to pass through the light ranking model. Training on both datasets together has shown potential in initial offline experiments.

System efficiency: An ambitious future experiment might explore using approximate nearest neighbor (ANN) based search to rank our ad candidates efficiently. Our current system exhaustively scores all surviving ad candidates to find the top-K candidates. Instead, we could precompute an index over the dense embedding space so that we can efficiently return the ad candidates closest to a particular user embedding. This efficiency improvement might allow us to remove more heuristic filters, score more ad candidates with similar resource usage, and simplify the stack further.

Conclusion

Redesigning our early filtering system from scratch improved our ad relevance models and system performance while simplifying our ranking stack. It also paves the way for accelerated productivity and future improvements that capitalize on cutting-edge ML technology.

Twitter's revenue team is in the middle of a similar large scale redesign of the ad server. This effort, called Project Tao, aims to revamp our ad serving infrastructure by separating our product and ranking logic into different services. Tao unlocks even more opportunities for our early filtering stack and the rest of our ranking models. These changes pose immense technical challenges but will also underpin the future of Twitter. If any of the topics outlined in this blog interest you, consider joining the flock!

Acknowledgements

The SplitNet system was developed by Jiyuan Qian, Rui Zhang, Hongjian Wang, Deepak Dilipkumar, Sandeep Sripada, James Gao and James Neufeld. We would like to thank Yi Zhuang, Pavan Yalamanchili, Ira Ktena, Joost van Amersfoort, Briac Marcatté, Priyank Jain, Jean-Pascal Billaud, Scott Chen and the rest of Revenue Science for their contributions to the project and to the blog, and a special mention to leadership for supporting this ambitious project - Steven Yoo, Gary Lam, Srinivas Vadrevu and Luke Simon. Finally, thanks to Jen Satzger and Alicia Vartanian for designing the illustrations in this post.

Tags: ads machine learning

More from Infrastructure

Testing Twitter.com: achieving reliable test results at scale
By Anna Goncharova on Wednesday, 17 July 2019

Building the new Twitter.com
By Charlie Croom and Gregory Baker on Monday, 15 July 2019

MetricsDB: TimeSeries Database for storing metrics at Twitter
By Satish Kotha and Ilho Ye on Monday, 13 May 2019

Partly Cloudy: The start of a journey into the cloud
By Joep Rottinghuis on Monday, 8 April 2019