Federated Learning GBoard

Overview

Federated learning is a relatively new technology allowing the machine learning models deployed on edge devices to be updated using locally available data. Incredibly, this can be achieved without communicating the local data to a centralized server, making it a privacy-preserving form of distributed machine learning. In this reading, we will review how federated learning works, and how it is used by Google to update GBoard, the phone keyboard for their Android mobile operating system.

Recap of Federated Learning

Instead of working with individual data samples, federated learning works with client devices. That is, a randomly selected sample used in traditional machine learning will instead be cast as a randomly selected client containing *N* training data samples. Then, instead of sending the data that is locally stored, the weights and biases of the local model can be sent instead. Let's walk through the procedure step-by-step.

First, the global model, stored in a centralized repository, is broadcast to all devices (or those that meet certain criteria, such as operating system or country of origin). Then, the model is locally trained on newly acquired data over time - essentially, continuous training. However, since there will likely not be enough information on a single device to optimize performance, federated learning allows us to crowdsource data from other users' devices to help improve the performance of our model. Our local model's weights and biases are then communicated to the centralized server, which aggregates these localized weights with those from other users' devices (such as by averaging their values). The updated model can then be rebroadcast to devices, and will hopefully demonstrate superior performance to the prior model.

Pros & Cons of Federated Learning

There are pros and cons to the federated learning approach. Federated learning is inherently privacy-preserving since no raw user data is transferred. Furthermore, it exhibits less communication overhead, since weights and biases are generally smaller in size than the corresponding data, which would also then take up space in the centralized server. However, the performance of a federated model is typically lower compared to one built on the full dataset (which can be a larger model which is subsequently compressed using model distillation to fit on the end-user device), which might make it unsuitable for some applications.

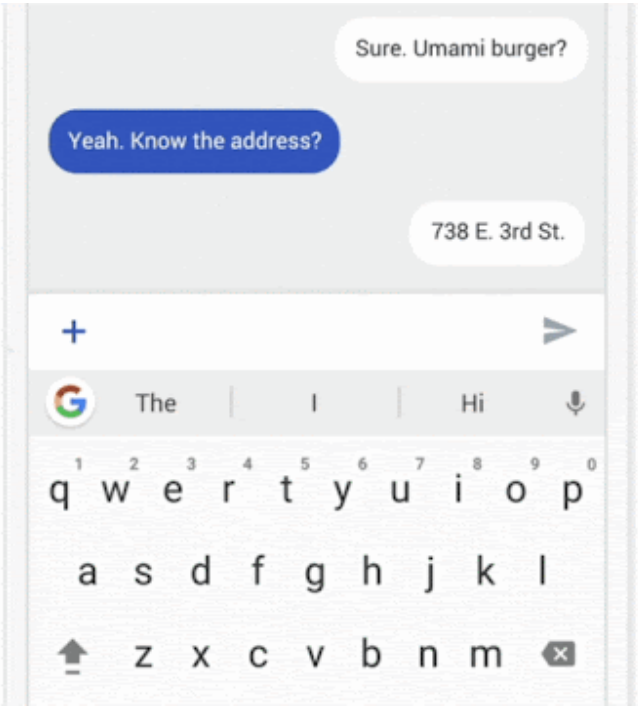
The aggregation procedure is one of the most important aspects of federated learning. There are two common aggregation procedures: fedSGD and fedAVG. When using fedAVG, the procedure is as described above, where our model is trained multiple times using local data, and the weights and biases of our network are updated. These updated weights and biases are then sent to the centralized server and then averaged across available client devices. In

fedSGD, the model weights are not necessarily updated, but the gradients for updating are determined based on local data. These gradients are then sent to the centralized server, which are then averaged and used for updating. Using fedSGD provides superior performance since it guarantees convergence, but has a higher communication overhead since the gradients must be sent at every iteration. In comparison, fedAVG has a fairly low communication cost but cannot guarantee convergence.

As you may be thinking, these aggregation procedures are quite naive implementations. This area is still an active research area, and so new methods are being developed continuously. For the interested reader, some promising methods worth looking into are federated matched averaging (FedMA) and probabilistic federated neural matching (PFNM).

Google's GBoard for Android and iOS devices

A great example of federated learning on edge devices is Google's GBoard. GBoard is the keyboard used in a number of Android and iOS devices. The system works by recording what the user writes (i.e., the context), what recommendations were shown, and whether the user clicked the suggestion.



Since the application needed to provide training updates across millions of devices, Google developed their own federated learning algorithm to minimize communication overhead. Updating was also only done on devices that were charging and connected to WiFi. They also developed a secure aggregation protocol to further improve privacy, ensuring that no user's average update can be inspected before averaging takes place.

Back To TinyML

While GBoard does not explicitly use TinyML, it is a good example of how federated learning can be applied to TinyML applications. Most resource-constrained devices that fall under the purview of TinyML do not have sufficient resources to train a model from scratch but do have access to small amounts of localized data. Federated learning provides an alternative route wherein local TinyML models could be updated by sending gradient updates to a centralized server and aggregated.

However, there are some additional challenges associated with the implementation of federated learning in TinyML. Most notably, some devices may have insufficient resources to model binary representations which allow gradient updates to be performed accurately. Despite these challenges, this is likely to be an important area of focus given its potential utility, and is thus well worth knowing about!