On-device Training: Limitations and Opportunities

About This Reading

Machine learning models are great when they work, but once deployed their performance can often degrade over time due to the phenomenon of both concept drift and data drift. One of the ways to tackle these issues and ensure models continue to remain high-performing is to update them with new data. To do this, we have to continuously monitor the ML model deployments, which in our case can be costly as communication is costly in terms of energy consumption. An alternative way to keep the models updated is through on-device training.

What Is On-Device Training?

On-device training refers to the training of a machine learning algorithm on the device where it will be deployed. That is, the device was not trained on an external device (like a laptop or desktop computer) and exported to the deployment device, which is what we have done in all of our applications thus far. There are additional challenges that come with on-device training that make it far from simple to implement, especially on the resource-constrained devices we work with in the field of TinyML.

Smartphones and other devices have been able to perform on-device training for a while. However, when it comes to resource-constrained devices like microcontrollers, this is considerably more challenging.

Tiny In-Device Training Challenges

Firstly, in order to train an algorithm, we need data to be able to achieve this. Unfortunately, this also means we need a place to store that data, which can be difficult if only a few kB of memory is available for data storage. This would ultimately limit how much data could be stored and would dictate how often the algorithm would need updating.

Secondly, training the algorithm would also be difficult due to the lack of numerical precision available on most resource-constrained devices. Propagating gradients in the computational graph of a neural network requires high precision because these values are usually quite small. The 8-bit precision of a microcontroller is not really sufficient for this procedure and will result in vanishing or exploding gradients due to the limited numerical precision available.

Another important issue is the ability to perform training itself. Training requires additional features in the runtime that are not necessarily required during inference. By incorporating training characteristics, this will add bloat to the library required to generate the runtime, limiting the size of the network that can be deployed alongside the runtime, while still fitting into the

same limited memory. Furthermore, the peak memory loads are often higher during training which might also limit the size and performance of the deployed model.

Clearly, on-device training is not currently feasible on some of the more resource-constrained devices. However, this is still an active area of research, and the hope is that one day training will be able to be performed on all manner of devices, regardless of resource constraints. For the time being, we will have to rely on some of the more resource-rich devices that contain sufficient memory, compute power, and numerical precision to perform on-device training.