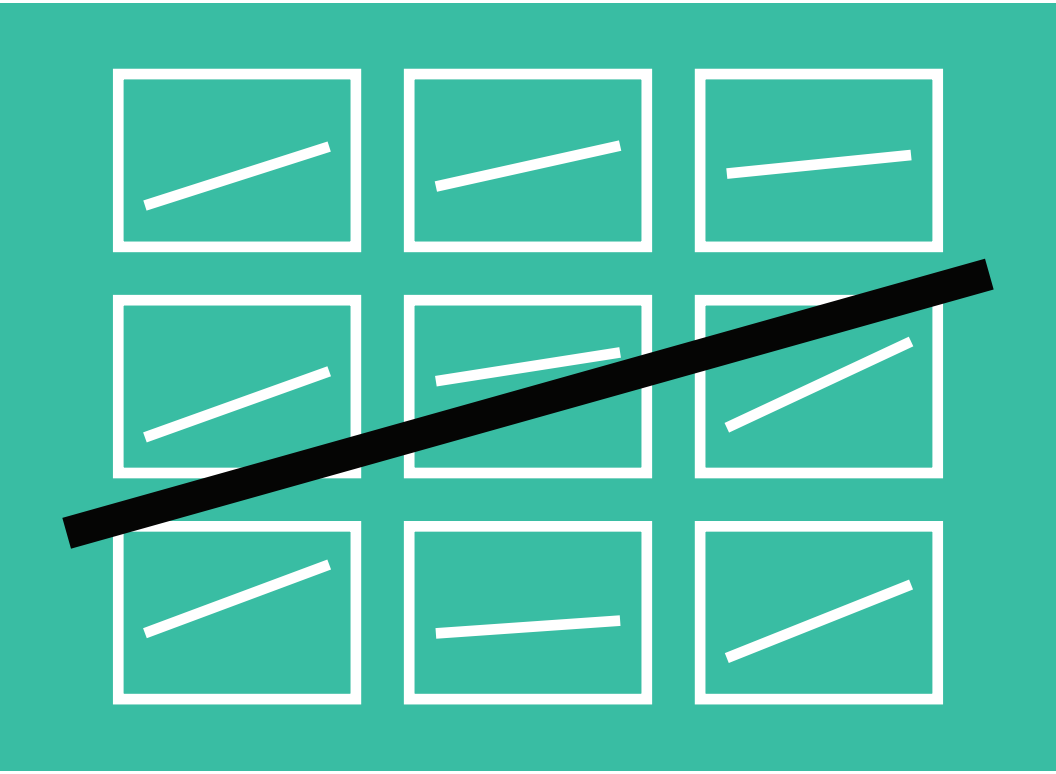
Cloudera Fast Forward Labs

Federated Learning



Cloudera Fast Forward Labs

Federated Learning

https://www.fastforwardlabs.com

New York, NY

To the future—

Contents

CHAPTER 1

Introduction

To train a machine learning model you generally need to move all the data to a single machine or, failing that, to a cluster of machines in a data center. This can be difficult for two reasons.

First, there can be privacy barriers. A smartphone user may not want to share their baby photos with an application developer. A user of industrial equipment may not want to share sensor data with the manufacturer or a competitor. And healthcare providers are not totally free to share their patients' data with drug companies.

Second, there are practical engineering challenges. A huge amount of valuable training data is created on hardware at the edges of slow and unreliable networks, such as smartphones, IoT devices, or equipment in far-flung industrial facilities such as mines and oil rigs. Communication with such devices can be slow and expensive.

This research report and its associated prototype introduce *federated learning*, an algorithmic solution to these problems.

In federated learning, a server coordinates a network of nodes, each of which has training data that it cannot or will not share directly. The nodes each train a model, and it is that model which they share with the server. By not transferring

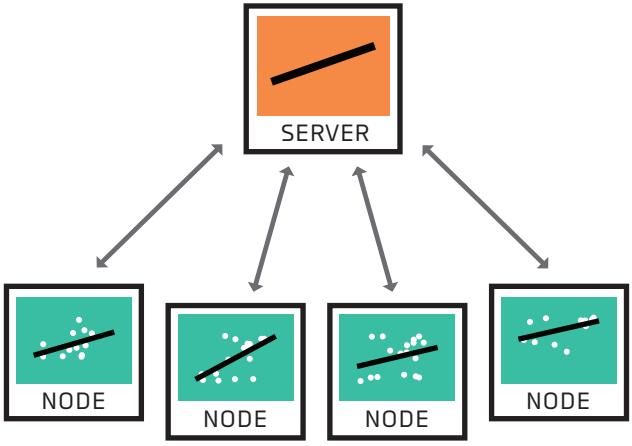


FIGURE 1.1 *In federated learning, a network of nodes shares models rather than training data with a server.*

the data itself, federated learning helps to ensure privacy and minimizes communication costs.

In this report, we discuss use cases ranging from smartphones to web browsers to healthcare to corporate IT to video analytics. Our working prototype focuses in particular on industrial predictive maintenance with IoT data, where training data is a sensitive asset.

CHAPTER 2

The Problem and the Solution

In this chapter, we will describe three hypothetical scenarios that seem very different, but which share characteristics that make them a great fit for federated learning. We'll then introduce a specific federated learning algorithm, and explain how it helps. Finally, we'll address the practical systems problems that complicate its use.

2.1 User Data on Smartphones

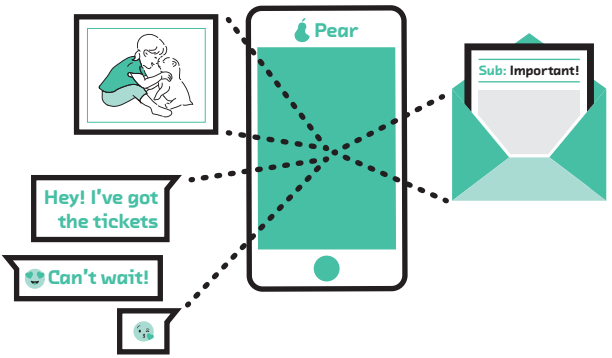


FIGURE 2.1 *Smartphones generate a wealth of data including pictures, text messages and emails.*

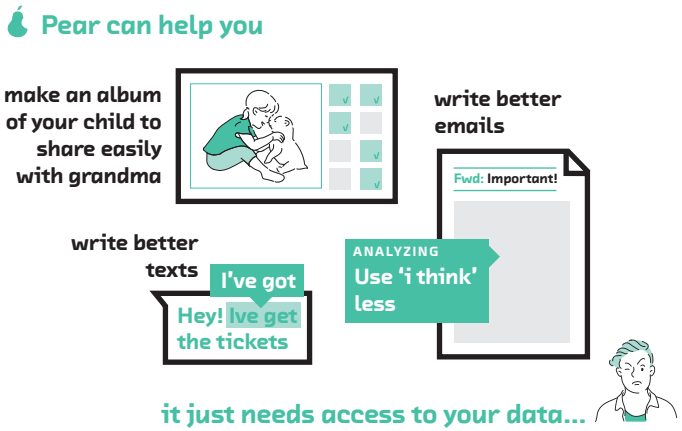


FIGURE 2.2 *Smartphone manufacturers could use that data to personalize a user's experience—but they require access to do so.*

Our first scenario concerns Pear, a company that makes a popular smartphone. Users love it. They use it to take pictures of their kids, email their colleagues, and write quick text messages to friends. All of this activity on millions of Pear phones generates data that, if it were combined, would allow Pear to train models to make its phones even better.

Pear's phones could learn to spot particularly good baby photos and proactively offer to share them with friends and family. They could make it easier to write emails that are more likely to receive quick replies. And they could make composing text messages even quicker and easier by accurately suggesting the next phrase, whatever the language.

The difficulty is that many users are not comfortable sharing the training data these examples would require (baby photos, work emails, personal text messages) with a

multinational corporation. Even among those who are not sensitive to privacy concerns, some will still refuse to share their data because they don't want to waste their bandwidth uploading data that will primarily benefit a private company. And among those who do choose to share their data, that data is often protected by laws that place significant administrative burdens on companies that wish to use it.

Pear understands these concerns, and has earned a reputation as a company that takes privacy more seriously than its competitors. How can Pear add new predictive features to its phones while respecting its users' privacy?

2.2 Predictive Maintenance

TurbineCorp sells turbines for installation in power stations. These machines are profitable to run but expensive to maintain, and very expensive to repair. TurbineCorp wants to differentiate itself from its competitors by offering customers access to a predictive model of turbine failure.

This model would use readings from sensors installed on each turbine as input, and return an estimate of its remaining useful life. A good model would reduce the likelihood of an expensive failure by prompting the owner to maintain a turbine before it fails. It would also help avoid the almost equally expensive mistake of maintaining a turbine too early and too often.

This model needs training data—but testing lots of turbines until they failed in order to acquire that data would be an expensive endeavor for TurbineCorp. It would be less costly for TurbineCorp if its customers were to send it such data. More importantly, the failures actual customers experience

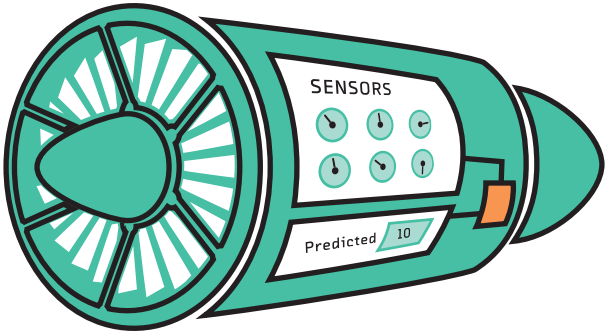


FIGURE 2.3 *Turbine sensor data could be used to train a predictive maintenance model.*

will be more representative of real-world use than those TurbineCorp would see in factory experiments. In short, training data acquired from customers would be both cheaper and better.

But there are several problems. Some of their customers are reluctant to share details about turbine failures in their facilities. Furthermore, some operate in countries such as China, where power stations are considered strategic assets, and are therefore legally prevented from exporting sensor data. And, as a practical matter, the volume of data generated by the dozens of sensors on each turbine is enormous, which makes streaming it back to TurbineCorp an engineering challenge.

How can TurbineCorp build an accurate predictive maintenance model without direct access to the best and cheapest

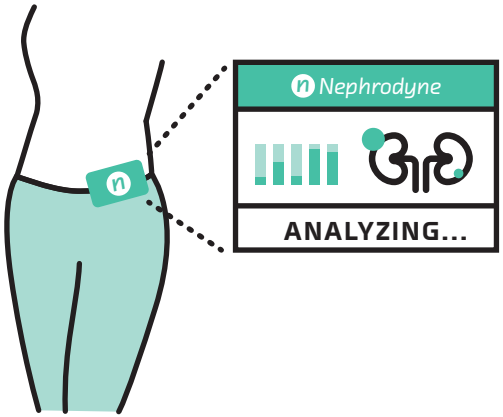


FIGURE 2.4 *Wearable medical devices collect data that could save lives.*

training data? (This scenario is the focus of our prototype, discussed in 3 Prototype.)

2.3 Medical AI

Nephrodyne, a medical device company, wants to offer a wearable device that detects kidney problems in users before they become acutely serious. The company knows from lab trials that its prototype rule-based device performs fairly well, but it does not have the accuracy required for use by nonexperts outside a hospital. Nephrodyne is confident that a machine learning model would perform even better than the rule-based model, and its business team has determined that Germany would be the best place to launch this product.

The problem is, as a private company based in the United States, Nephrodyne is having trouble getting access to

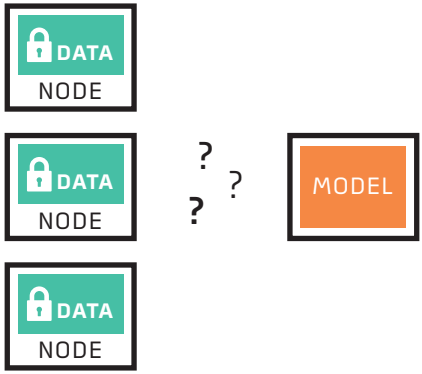


FIGURE 2.5 *Federated learning helps when the data cannot be moved.*

the representative training data required to build the model. Data protection regulations in the European Union place significant regulatory burdens on companies working with personal data, especially those outside the EU. Likewise, HIPAA regulations in the US make it difficult to work with US patient data (and in any case, that American patient data would differ systematically from that of the target customers in Germany). And finally, anywhere in the world, but especially in Germany, patients are typically unwilling to share their personal data with a private medical device company. Patients do, however, share their data with their healthcare providers.

Given these constraints, how can Nephrodyne work with multiple healthcare providers to train the accurate model it needs to build its new product?

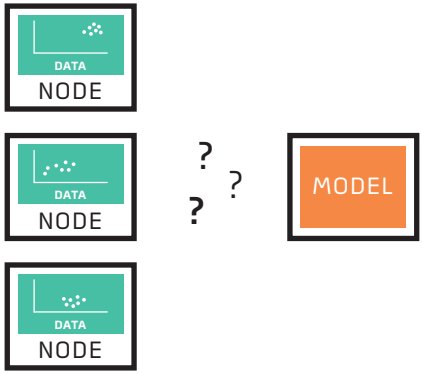


FIGURE 2.6 *Federated learning helps when the data on each node is different.*

2.4 The Federated Learning Setting

These three scenarios share two common characteristics. These characteristics comprise what machine learning experts call the *setting* of federated learning.

First, the most representative (and cheapest) training data cannot be moved away from its source. This characteristic is the most important for identifying a problem that might be a good fit for federated learning. The reasons for this constraint can include privacy concerns, regulatory impediments, and practical engineering challenges.

Second, each source of potential training data is different from every other. Each of these sources will show biases relative to the overall dataset. For example, Pear wants to predict which emails will receive replies, but perhaps one user almost always gets a reply, and one user almost never does. Neither

user is typical. Or perhaps two of TurbineCorp's customers stress their turbines in different ways and observe different failure modes. And maybe one of Nephrodyne's users simply doesn't have much data. The fact that any one source's data is biased and small relative to the total dataset means that it's difficult to build a good global model based on data from a single source.

Here we'll introduce the term *node*, used in this report to refer to a source of training data. A node might be a physical device, a person, a facility or other geographic location, a legal entity, or even a country. If each node will not or cannot share its data directly, or if you are concerned that any one node might produce a biased subset of data, then you have a problem that is potentially a good fit for federated learning!

Distributed Machine Learning

Federated learning is one example of distributed machine learning, but there exists another variety that is currently more common and more mature: distributed machine learning in the datacenter.[1] In this setting, most of the practical constraints of federated learning go away. In particular, we are free to move training data between nodes, and communication between them is relatively fast and cheap.

Algorithms for distributed machine learning in the

1 *One of the earliest uses of the term "federated learning" draws this distinction in its title: see Jakub Konečný, Brendan McMahan, and Daniel Ramage's paper "Federated Optimization: Distributed Optimization Beyond the Datacenter" (https://arxiv.org/abs/1511.03575).*

absence of the constraints of federated learning differ in various ways that trade off speed, complexity, and accuracy. In most circumstances, it's best to let the tool you're using decide these algorithmic details. The cluster computing framework Apache Spark supports distributed machine learning out of the box and works well for many use cases. Dask, a library for parallel computing in Python, also supports machine learning and may be a good choice for custom use cases or experimentation. Deep learning packages like TensorFlow and PyTorch provide distributed implementations and can additionally be layered on management platforms like YARN and Kubernetes[2] for robustness and fault tolerance.

2.5 A Federated Learning Algorithm

So, how does federated learning work?

The crucial insight is to realize that the nodes, which are the sources of training data, are not only data storage devices. They are also computers capable of training a model themselves. The federated solution takes advantage of this by training a model on each node.

The server first sends each node an instruction to train a model of a particular type, such as a linear model, a support vector machine (SVM), or, in the case of deep learning, a particular network architecture.

On receiving this instruction, each node trains the model on its subset of the training data. Full training of a model would

---

2 *E.g., TonY (https://github.com/linkedin/TonY) and Kubeflow (https://www.kubeflow.org/).*

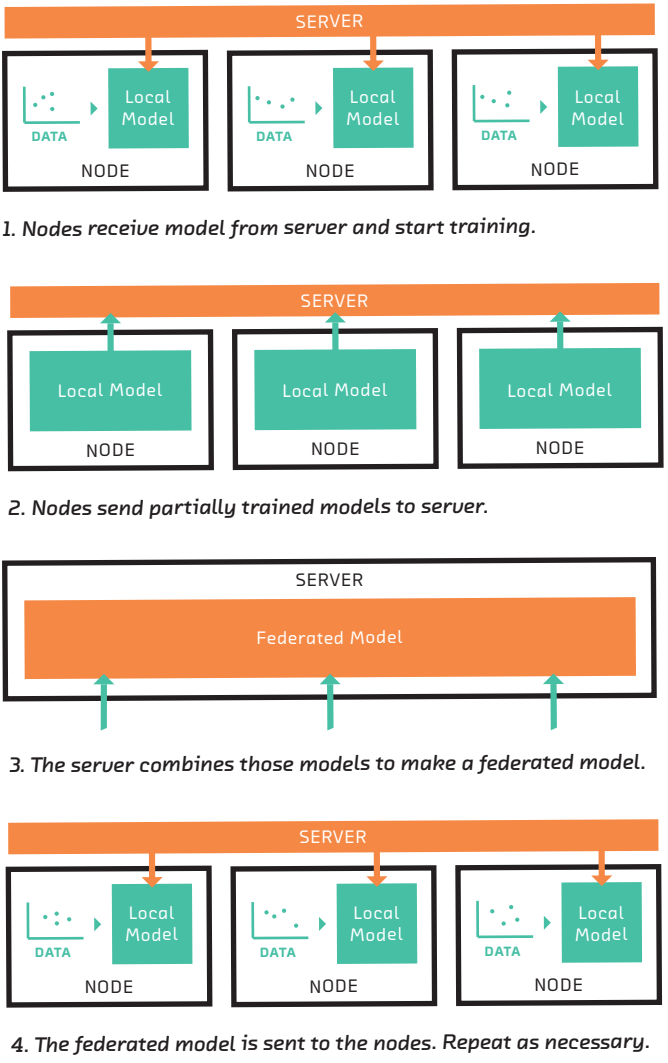


FIGURE 2.7 *A federated learning algorithm.*

require many iterations of an algorithm such as gradient descent, but in federated learning the nodes train their models for only a few iterations. In that sense, each node's model is partially trained after following the server's instruction.

The nodes then send their partially trained models back to the server. Crucially, they do not send their training data back.

The server combines the partially trained models to form a federated model. One way to combine the models is to take the average of each coefficient, weighting by the amount of training data available on the corresponding node. This is called *federated averaging*.

The combined federated model is then transmitted back to the nodes, where it replaces their local models and is used as the starting point for another round of training. After several rounds, the federated model converges to a good global model. From round to round, the nodes can acquire new training data. Some nodes may even drop out, and others may join.

This algorithm makes it possible to train a model using all the training data spread across the nodes, without moving that data off the nodes.

It is difficult (although not in general impossible) for the server to reconstruct the training data from the trained models it receives from each node. We will discuss this possibility in 5 Ethics, but for now, we simply note that the server does not have or need access to the training data. Indeed, no training occurs on the server, so it doesn't need specialized training hardware such as a GPU.

The amount of data the nodes have to send to the server is much less than it would be if the training data were shared

directly. This is because a machine learning model is generally much smaller than the data on which it was trained.

2.6 Applicability

Federated learning is more complicated than working with all the data on one machine. If moving the data is an option, you should try that before you try federated learning. But even if moving the data is not an option, there are a few things you should consider before you take the leap.

2.6.1 The Data on the Nodes

Federated learning gives you access to more training data. But if that extra training data won't help you make better predictions than the data you already have, then federated learning will not help. For more training data to help, two things must be true.

First, there must be room for improvement. If the model structure lacks the flexibility to capture the patterns in the training data, then it doesn't matter how much training data you throw at it.[3]

Second, the additional training data must be predictive of the shared task. This is often assumed in non-federated learning. For example, a language model trained with the text of 1,000 English books will be better than a language model trained with 10 English books. But the nodes in a federated model could be so completely different or uncorrelated that the data they have to offer is not helpful. Two nodes

---

3 *You can check if this is true by constructing a learning curve using the data you do have.*

are unlikely to train a good language model together if one of them uses 10 English books and the other uses 10 Japanese books.

As a more realistic example, suppose the potential nodes of a federated network are businesses that want to predict how many servers they will need next year. It's quite possible that the server demands of these businesses are driven by the same external factors, and therefore correlated. In this case federated learning may help. But if their server demands are uncorrelated, each business will get results that are just as good (or better), with less expense, by building its own model using only its own data.

2.6.2 Walk Before You Can Run

Before doing something new in machine learning, it is a best practice to establish a well-understood performance benchmark. In the context of federated learning, that means training a non-federated model on representative proxy data. For example, you might first train a prototype predictive text model on a Wikipedia corpus, or a predictive maintenance model on laboratory stress tests.

This benchmark is valuable because it gives you an idea of the performance you can expect from a federated model before you build it. That's because, while federated learning is almost as good as having direct access to all the training data, it is not usually better. The non-federated model therefore places an upper limit on the performance of a federated model trained *on the same data*.

The only situation in which a federated model could beat a non-federated benchmark is if, in the move to federated

learning, you gain access to more or qualitatively better training data. This is not necessarily a rare situation, but relying on it is a risk you should carefully assess.

2.6.3 Not Only Deep Learning

It is possible to train any kind of machine learning model using federated learning, provided there exists a meaningful way to combine two models. This is true of at least linear models, SVMs, and neural networks. In particular, despite what you may have heard, federated learning is not only applicable to deep learning.

In our prototype we do use a neural network, but the most prominent documented production example of federated learning (see 4.2.1 Firefox) uses an SVM. Each type of model has its usual advantages and disadvantages in a federated setting (flexibility, training data requirements, speed, etc.). Your choice may be restricted in practice if the nodes are unusual edge hardware (see 4.3 Tools and Vendors).

2.7 Systems Issues

An attempt to use any algorithm in the real world will face practical challenges due to the hardware, software, and network. Computer scientists call these *systems issues*. Sometimes they are mere details that can be addressed easily, but sometimes they render an elegant algorithm useless in practice. It is therefore important to be aware of them.

In some contexts where federated learning would otherwise make sense, it can be unsuitable because of its power demands. For example, Pear's smartphone users may not want to dedicate a portion of their battery to training a neural

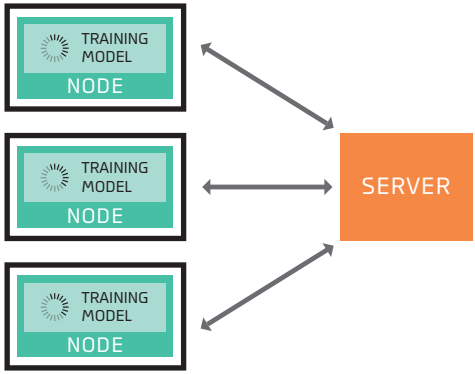


FIGURE 2.8 *Training models on client devices uses power.*

network in the background. And training isn't the only potential power sink: communication over a cellular network using the radio antenna is a particularly power-hungry task on a smartphone.[4] Federated learning is less demanding in this respect than a strategy that involves transmitting *all* the training data back to the server, but it is more demanding than shipping a static model to the phone just once. This trade-off may be worth making if the federated model is much better than a static model trained once on unrealistic data, but you'll need to consider whether this is the case.

There's also the possibility that a node may drop out. The most obvious example of this is a smartphone user who turns their phone off, but a healthcare provider participating in

4 *Power consumption and communication costs are closely related. See 6.1 Reducing Communication Costs for more on this topic.*

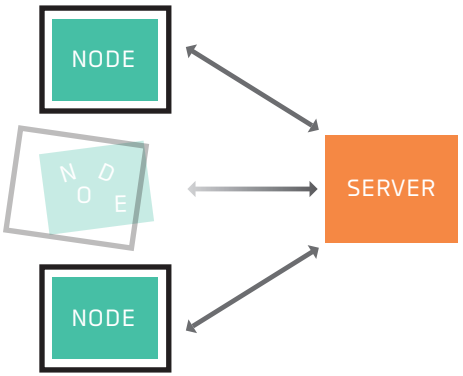


FIGURE 2.9 *Some nodes may drop out of the network.*

Nephrodyne's network or a power station participating in TurbineCorp's network might also choose or be forced to cease participation. The individual nodes cannot be counted on to have the uptime you can expect of a server in a datacenter. If federated learning is distributed across more than a handful of nodes, you can pretty much guarantee at least *one* of them will become temporarily or permanently unavailable at some point. Federated learning must be robust to this possibility.

Finally, even with plenty of power and no dropouts, the server is faced with the unavoidable reality that some nodes are likely to be stragglers that take longer to do their work, because they have either slow hardware or a slow network connection. Can this risk be reduced? At what point should the server abandon hope of receiving an update from a particular node? How should it handle an update if it arrives after the server has already combined the updates from the other

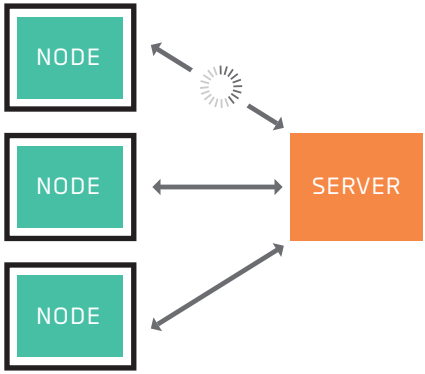


FIGURE 2.10 *Some nodes may take longer to do their share of the work.*

nodes? The answers to these questions depend on the details of your system and the particular federated learning algorithm you are using. In some cases, they are also the subject of current research (see 6 Future).

In any production application of federated learning it is essential to consider the real-world issues you will face and, where possible, mitigate the damage they do. In the next chapter we describe how we built a working product using federated learning. Our prototype avoids some of these systems issues, but is forced to deal with others.

Machine learning on the edge

In order for federated learning to be possible, the nodes must be able to *train* a model. The ability to do *inference* (i.e., apply a pretrained model) is not sufficient. Training

is not always possible on edge devices such as specialized IoT hardware. This is particularly true in the case of neural networks. This difficulty is due to the current state of the toolchain rather than anything fundamental, and we expect that limitation to go away in the near future.[5]

But even if you can't train on edge devices (and therefore can't use them for federated learning), machine learning inference on edge devices is still a very exciting possibility. Among the many compelling arguments for shifting the burden of data processing (including featurization and inference) to edge devices is *cost*. You save money because the owners of the edge devices (i.e., your users) provide the compute resources.

In a sense, along with the move to the cloud (and serverless in particular), moving machine learning to edge devices is part of a general trend toward minimizing the long-lived infrastructure of a system.[6]

---

5 *See 4.3.4 Mobile and Edge and "What Does It Take to Train Deep Learning Models On-Device?" (https://petewarden.com/2018/10/04/what-does-it-take-to-train-deep-learning-models-on-device/).*

6 *See "How We Built a Big Data Platform on AWS for 100 Users for Under $2 a Month" (https://read.acloud.guru/how-we-built-a-big-data-analytics-platform-on-aws-for-100-large-users-for-under-2-a-month-b37425b6cc4) for an example of migration of a backend system to serverless and the "AI frontend" to TensorFlow.js running on user devices. Together these changes resulted in a cost reduction of 3,700x.*

CHAPTER 3

Prototype

Turbofan Tycoon, the prototype we created for this report, is a federated learning solution to one of the scenarios we saw in the previous chapter: training a predictive maintenance model using real-world customer data. In this chapter we describe how we selected the problem, prepared the data, built the federated model, and created the user interface. If you'd like to skip the predictive maintenance part of the chapter and get straight to the federated learning, jump ahead to 3.5 The Federated CMAPSS Model.

3.1 Predictive Maintenance Primer

If a piece of equipment generates revenue, its failure costs money. The costs multiply if the failure makes a production line stop, wastes time-sensitive raw materials, or forces customers to wait for their orders. The equipment may fail in a way that requires it to be scrapped completely, or even in a way that damages other equipment. The owner of an expensive piece of equipment maintains it with the hope of avoiding these costly failures.

But maintenance also causes downtime. This downtime is planned, so it is hopefully less expensive than unplanned downtime due to equipment failure, but it is not free.

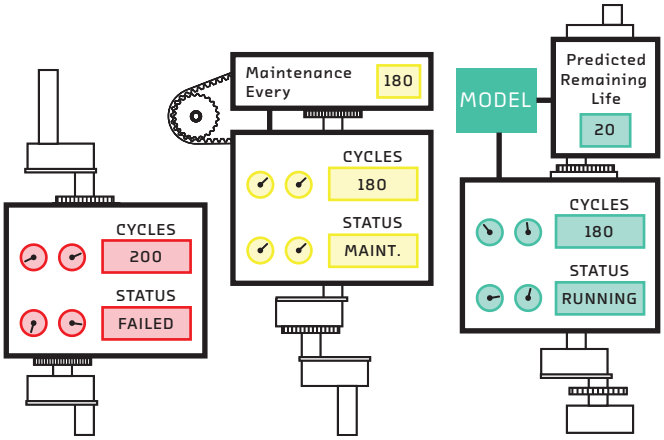


FIGURE 3.1 *Corrective maintenance waits for a failure. Preventative maintenance maintains on a schedule. Predictive maintenance uses machine learning to decide when to maintain.*

Undermaintenance and overmaintenance are therefore both expensive mistakes. How can we avoid them?

The simplest maintenance strategy is *corrective maintenance*. This is a fancy technical term that means "waiting for the equipment to break." This is what you're doing when you take your car to the repair shop *after* the engine catches fire. This strategy is suboptimal because it undermaintains.

The next step up the evolutionary tree is *preventative maintenance*. This is servicing on a schedule. The schedule is fixed and does not depend on detailed observations of the current state of your particular piece of equipment. Rather, the schedule is derived from average experience. Taking your car in for a service every 10,000 miles as recommended by its manufacturer is an example of preventative maintenance.

In order for preventative maintenance to work, its schedule must necessarily be conservative. So, while corrective maintenance undermaintains equipment, preventative maintenance tends to overmaintain.

The most intelligent maintenance schedule is *predictive maintenance*. Here, a machine learning model is applied to your particular piece of equipment to predict its remaining useful life (RUL). When that RUL estimate drops below some threshold, you can intervene by maintaining or replacing the equipment.

If the machine learning model at the heart of a predictive maintenance strategy is good, this most advanced approach can save a huge amount of money relative to the alternatives.

3.2 Why Federated Predictive Maintenance

It only makes sense to apply federated learning to machine learning problems where the training data is difficult or impossible to move (see 2.6 Applicability). We chose a predictive maintenance scenario for our prototype because it really is sometimes subject to this constraint.

As we discussed in the previous chapter, if a manufacturer wants to build a predictive model to share with its customers, then the training data that belongs to those customers is the gold standard. It is more diverse and representative of real-world use than data the manufacturer collects in laboratory stress tests and simulations. Assuming they can access it, collecting customer data is also cheaper for the manufacturer.

But sometimes competitive pressures can result in a business being unwilling to share data with suppliers (who may themselves be competitors, or may be suppliers to

competitors). There can also exist legal constraints that prevent a facility of strategic importance, such as a power station or military base, from exporting its data.[7] Even if a facility is willing and legally able to share data with a supplier, predictive maintenance data can be so voluminous that this goal presents a practical engineering challenge for the network. This challenge can be particularly acute in industries such as mining, when the facilities are in remote locations or the equipment is mobile.

The fact that federated learning makes it possible to train on a huge amount of private data while only sending small models over the network therefore makes its application to industrial predictive maintenance a very exciting possibility.

3.3 The CMAPSS Dataset

To build our prototype, we used the CMAPSS turbofan degradation dataset. A turbofan is a kind of jet engine. This dataset is to the predictive maintenance community as ImageNet or MNIST is to the computer vision community.[8]

We used data for 400 engines, each of which was allowed to run until failure. There were 24 time series (21 sensor readings and 3 operational settings) for each engine. The sensor readings are temperatures, pressures, and fan rotation speeds for various components of the jetfan. The task was to predict

---

7 *See "Data Protection: The Growing Threat to Global Business" (https://www.ft.com/content/6f0f41e4-47de-11e8-8ee8-cae73aab7ccb), The Financial Times, May 13, 2018.*

8 *It is available from the NASA website (https://ti.arc.nasa.gov/tech/dash/groups/pcoe/prognostic-data-repository/).*

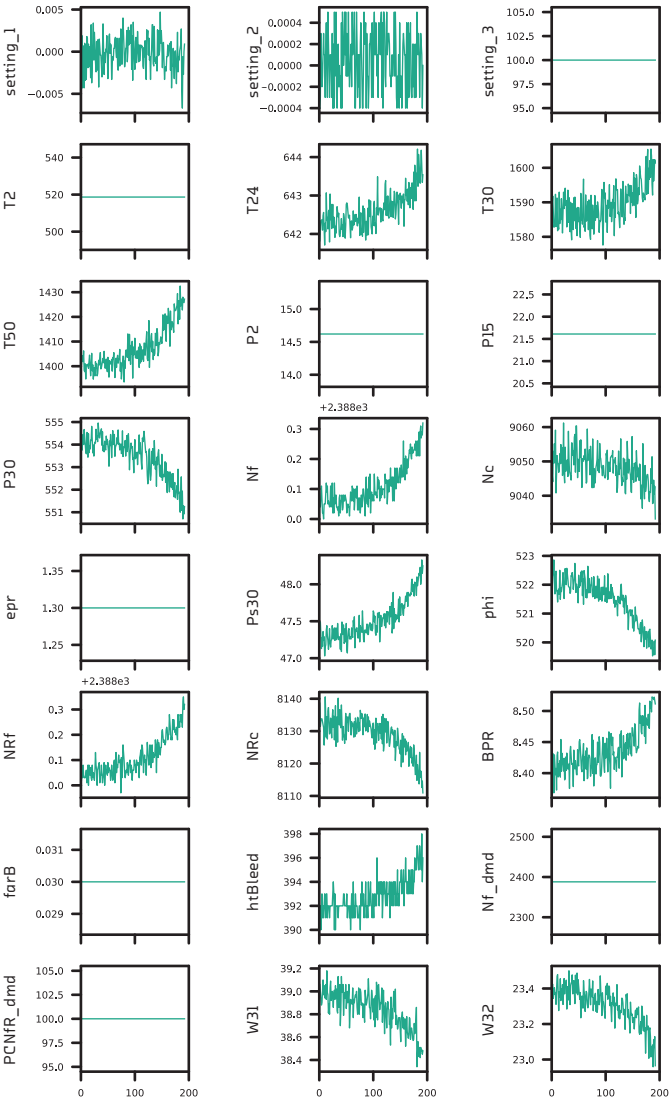


FIGURE 3.2 *Sensor data for one turbofan.*

the remaining useful life of the engine at a randomly chosen time before failure using the available history of sensor readings and settings. We used 16 of the 24 sensors and settings in constructing our model.

3.4 Modeling the CMAPSS Data

The CMAPSS training data can be modeled in several ways and, as discussed in 2.6.3 Not Only Deep Learning, federated learning does not in itself place many restrictions on the kind of model we build.

Because the focus of this report is federated learning rather than predictive maintenance, and because federated learning gives us considerable freedom to choose our modeling approach, our prototype uses the simplest modeling approach that performs significantly better than a naive model. A naive model of the CMAPSS dataset predicts RUL by assuming that every engine in the test set has a total useful life of 181 time steps (the median life of engines in the training set).[9] This naive model has a root mean squared error of 94.2.

The simplest modeling approach is to treat each time step as a training example. In this approach, an engine that has a total useful life of 314 time steps becomes 314 training

---

9 *In a more familiar classification context, the naive model is usually a dummy classifier that always predicts the most prevalent class in the training set. Such a dummy classifier will have an accuracy equal to the prevalence of that class. At a minimum, a trained model should have an accuracy greater than this. (For example, if 90% of emails are spam, a spam classifier with an accuracy less then 90% is doing worse than chance!)*

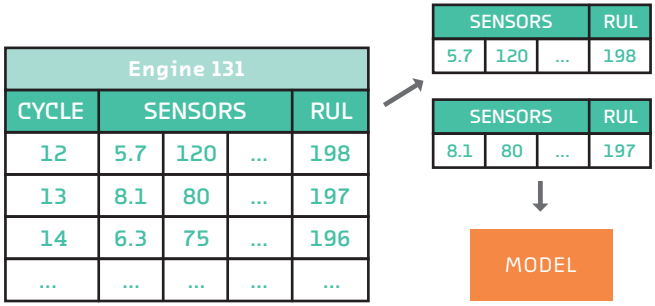


FIGURE 3.3 *Each time step is treated as a training example.*

examples. The features are the instantaneous values of the 16 sensors at each time, and the target variable is the remaining useful life at each time. The model does not have access to the history of the sensor data. It can only use the sensor readings at each instant to predict the remaining useful life.

Despite its simplicity, this formulation results in a model that is comfortably better than the naive model, so we use it rather than, e.g., modeling the full sequence with a recurrent

neural network.[10] The 400 engines in the set became 58,798 training examples and 16,245 test examples, each with 16 features. We standardized the features and target variable for numerical stability.

The model that learns how to predict the RUL from the 16 input features is a simple neural network with one hidden layer. The non-federated version of this model (i.e., trained with direct access to all the data on a single machine) has a root mean squared error of 62.4.

3.5 The Federated CMAPSS Model

In our federated version of the problem, we randomly partitioned the 320 engines that comprised the training set across 80 nodes. You can think of each of the nodes as a factory that has observed four turbofan failures. Assuming a factory is unwilling or unable to share the sensor data from these failures, it can choose from four maintenance strategies:

- A corrective maintenance strategy (waiting for the engines to fail)

---

10 *See* FF04: Summarization *for more on recurrent neural networks. For the particular problem of modeling the time until an event (e.g., customer churn, engine failure), we like the "Weibull Time-to-Event RNN" (https://ragulpr.github.io/2016/12/22/WTTE-RNN-Hackless-churn-modeling/). But some machine learning experts have found that an autoregressive feed-forward neural network (i.e., a regular or convolutional neural network) also works well for sequence problems. See "When Recurrent Models Don't Need to be Recurrent" (https://bair.berkeley.edu/blog/2018/08/06/recurrent/).*

- A preventative maintenance strategy (maintaining at a fixed time, hopefully some time before the average engine fails)
- A local predictive maintenance machine learning model, trained on the factory's four failed engines
- A federated predictive maintenance machine learning model, trained on the collective data of all 80 factories using federated learning

To train the federated model we wrote federated.py, an implementation of federated averaging, in about 100 lines of PyTorch. This implementation is a *simulation* of federated learning in the sense that no network communication takes place. The server and the nodes all exist on one machine. However, it is an algorithmically faithful implementation: the server and nodes communicate only by sending copies of their models to each other.

This approach makes it possible to experiment rapidly with very large numbers of nodes, without getting bogged down in network issues. And despite the simplification, we can reproduce many of the challenges discussed in 2.7 Systems Issues. For example, we can dynamically flag nodes as non-participants (which means they train on their local data, but do not send or receive model updates) or failed (which means they stop training altogether). We show the most important few lines of of `federated.py` in 8 Appendix: federated.py.

The models trained on each node (and the federated model that is their average) are simple feed-forward neural networks with one hidden layer. Including biases, the models are defined by 865 weights. The federated model has a final root

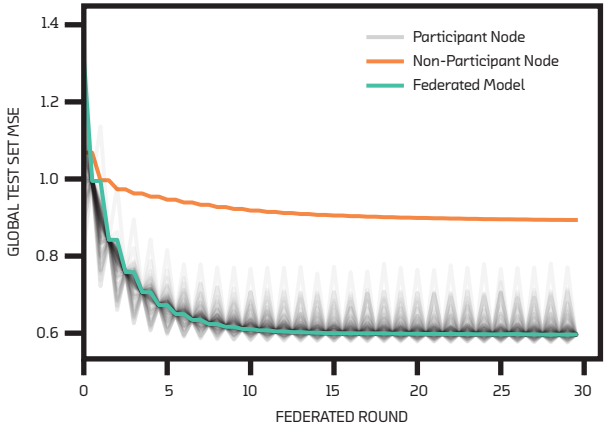


FIGURE 3.4 *Mean squared error loss (in standardized units) for the federated model, 79 nodes that participate in the federated model, and 1 node that does not.*

mean squared error of 64.3. This is almost as good as the 62.4 of the non-federated model trained on the same data without federated learning (and much better than the 94.2 of the naive model).

The federated model requires around 10 rounds of federated learning to reach its final error (where each node trains for one epoch per round). This is slower than the non-federated model. We show the loss curve in FIGURE 3.4. The characteristic sawtooth shape of the loss curves of the nodes is due to their training between rounds of communication.

We marked one node in our network as a non-participant. This is a node that trains as often as every other, but does not contribute to or benefit from the averaging process. This model is therefore effectively a non-federated model trained

on just four engines. With such a small amount of training data, it is vulnerable to overfitting. Its exact performance is very sensitive to exactly which four engines it uses (some are more typical than others, and result in a model that generalizes fairly well, while others result in a catastrophically bad model). But regardless of this randomness, it is almost always significantly worse than the federated model. We show the loss curve for the non-participant model we use as the "local" model in the prototype in FIGURE 3.4.

3.6 Product: Turbofan Tycoon

We knew that the model trained using federated learning made better predictions. The challenge for the frontend was how to communicate that fact to the prototype user in a form more visceral than a static chart. Our solution was to make the user a factory owner in charge of four running turbofans. We simulate a running turbofan by playing through the data step by step. The user decides the strategy for when a turbofan is maintained, and the effectiveness of that strategy determines the factory's profit. By simulating the lifespan of each turbofan we made the prototype dynamic, and by assigning a dollar value to accurately predicting that lifespan we were able to dramatize the payoff of federated learning.

In Turbofan Tycoon, every time a turbofan completes a cycle (one hour) the factory makes $250. If a turbofan breaks down it costs $60,000 to repair, and it takes 60 hours to get it running again. Turbofan maintenance can head off a breakdown, but it costs $10,000 and takes 20 hours to perform. When maintenance is performed is determined by which of

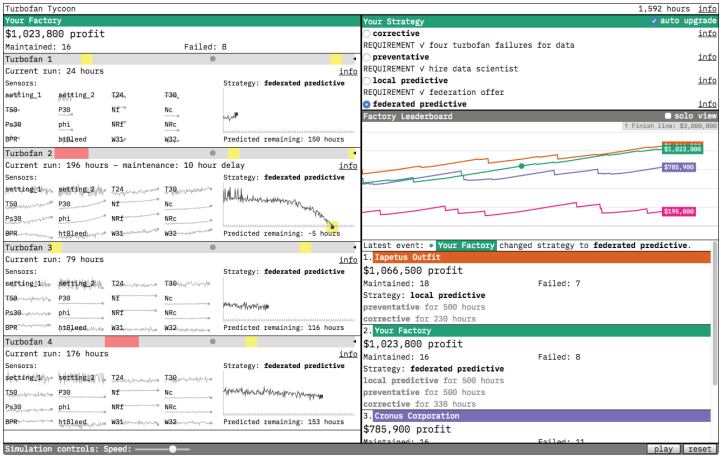


FIGURE 3.5 *The Turbofan Tycoon prototype.*

the four maintenance strategies the user selects: corrective, preventative, local predictive, or federated predictive.

3.6.1 The Life of a Turbofan

We visualize the turbofan lifespan with a collection of graphs. On the left we show the data from the 16 sensors used by the local and federated predictive models to predict the remaining useful life of the turbofan. If you watch the sensor data very closely and turn the simulation speed down very low, you, like the predictive models, may be able to make a guess about when a turbofan failure is coming, but it's a tough job (one better left to algorithms). We included the sensor data because we wanted to give a sense of the information the predictive models are distilling into strategy.

On the right side of the turbofan slot is a visualization of the maintenance strategy. For the corrective and preventative

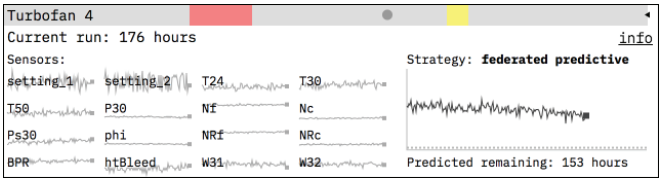


FIGURE 3.6 *The turbofan visualization shows the sensor data and maintenance strategy.*

strategies, there isn't a whole lot to see. For corrective, it's simply a line marking how many cycles that turbofan has gone through. For preventative, the preselected threshold (193 hours) at which maintenance is performed is shown as a dotted line. The local predictive and federated predictive graphs are richer. For each cycle each model predicts the remaining useful life of the turbofan. Maintenance is performed when that prediction drops below 10 cycles. The local predictive and federated predictive strategy graphs are visualized in the same way but differ in their predictions because of the different amounts of data they are trained on. The federated predictive model's larger training dataset makes it the more accurate predictor.

To guide the user through the different maintenance strategies, we set up the simulation so that they become available

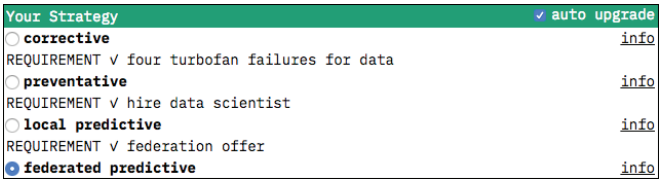


FIGURE 3.7 *More advanced strategies become available as the simulation progresses.*

one by one, as their requirements are satisfied. These requirements (four turbofan failures for data to upgrade to preventative, hiring a data scientist to upgrade to local predictive, getting a federation offer to upgrade to federated predictive) have some basis in the real world but are vastly simplified so as not to distract from the prototype's main purpose: to demonstrate the effectiveness of the different strategies.

3.6.2 Visualizing Strategy Effectiveness

The effectiveness of each strategy is visible in the factory's maintenance and failure counts. We further condense each of those measurements into one number: profit. The formula for profit calculation is the active turbofan cycles multiplied by the profit per cycle minus the total failure and maintenance costs for each turbofan slot in the factory. Because of how it's

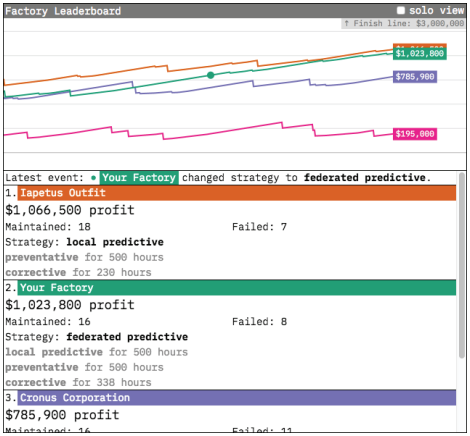


FIGURE 3.8 *The Factory Scoreboard shows the effectiveness of different strategies over time.*

calculated, the factory's profit shows the effectiveness of each strategy over time.

As a measure of the effectiveness of the different maintenance strategies, the factory's profit is only a useful indicator in the context of strategy paths not taken. In our prototype, these alternate paths are shown as the user's competitor factories. The other factories follow the same upgrade rules as your own factory, except they each max out at a different strategy level (corrective, preventative, and local predictive). As long as you follow the prototype's strategy upgrade suggestions, the strategy caps on the other factories mean you'll pull away from the pack as the simulation progresses and the effectiveness of the federated predictive strategy reveals itself.

3.6.3 The Simulation Trade-off

Behind the scenes, the simulation works by selecting a new random turbofan from the dataset each time maintenance or failure occurs. This gives the simulation a realistic variety: some turbofans run over 400 cycles, some cut out at 150. The law of large numbers means that, over enough time, each strategy's effectiveness will reveal itself. But, just like in life, streaks of good or bad luck can make a strategy look more or less effective than it actually is (if a factory using the corrective strategy gets a lucky streak of long-lasting turbofans, for example). Ultimately we decided the drama of the simulation was worth the potential of temporarily misleading results, especially because it contains a real-world lesson: if your competitor takes the lead with a subpar strategy, don't panic! Stick with what you know is effective, and over time you'll win out.

CHAPTER 4

Landscape

In this chapter we discuss use cases for federated learning. Most of these are natural and obvious but, for now at least, speculative. As a very new technology, federated learning is not yet in production use at many companies willing to discuss their experiences. We share what we've learned from the exceptions in 4.2.1 Firefox, 4.2.2 Owkin, and 4.2.3 OpenMined. Finally, we discuss the tools and vendors you can use if you decide to explore federated learning.

4.1 Use Cases

4.1.1 Smartphones and Browsers

Machine learning has huge potential to improve the user experience with smartphones and the web in general. In part that is because the training data is so abundant: users generate vast amounts of potentially valuable on-device data in their normal day-to-day interactions with these devices.

But aside from the practical challenge of getting this data off a device with a slow connection, the personal aspect of some of this data (what people type, where they travel, what websites they visit) makes it problematic. Users are reluctant to share this sensitive data, and possessing it exposes technology companies to security risks and regulatory burdens.

These characteristics make it a great fit for federated learning. The use case is so compelling that it comes as no surprise that Google researchers[11] are usually credited with its invention, and Samsung engineers have also contributed significant ideas.[12]

This use case is discussed throughout this report, including in 2.1 User Data on Smartphones, 4.2.1 Firefox, 6.1 Reducing Communication Costs, and 6.2 Personalization.

4.1.2 Healthcare

The healthcare industry offers huge financial incentives to develop effective treatments and predict outcomes. But the training data required to apply machine learning to these problems is of course extremely sensitive. The consequences of actual and potential privacy violations can be serious. For example, Facebook was recently forced to distance itself from press reports that it was sharing anonymized data with hospitals,[13] and a British hospital was reprimanded for sharing the health records of 1.6 million patients with Google DeepMind

---

11 *https://ai.googleblog.com/2017/04/federated-learning-collaborative.html*

12 *https://arxiv.org/abs/1712.07473*

13 *The project is on hiatus so that Facebook can focus on "other important work, including doing a better job of protecting people's data." See "Facebook Sent a Doctor on a Secret Mission to Ask Hospitals to Share Patient Data" (https://www.cnbc.com/2018/04/05/facebook-building-8-explored-data-sharing-agreement-with-hospitals.html), CNBC, April 5, 2018.*

as part of a trial to test an alert, diagnosis, and detection system for acute kidney injury.[14]

By keeping the training data in the hands of patients or providers, federated learning has the potential to make it possible to collaboratively build models that save lives and make huge profits. The European Commission's Innovative Medicine Institute is soliciting proposals to build a federated, private, international platform for drug discovery.[15]

In 2.3 Medical AI we described a wearable medical device use case for federated learning (inspired in part by the Google DeepMind controversy). In 4.2.2 Owkin we share what we learned from our conversation with Owkin, an ambitious and advanced startup that makes it possible for healthcare data owners to collaborate.

4.1.3 Industrial IoT

As discussed in 3 Prototype, industrial applications of decentralized machine learning can be subject to privacy concerns: if the nodes are in competition with one another, they may be unwilling to share training data because of what it reveals about their operations. Our predictive maintenance prototype explores this scenario.

---

14 *See "Royal Free - Google DeepMind Trial Failed to Comply with Data Protection Law" (https://ico.org.uk/about-the-ico/news-and-events/news-and-blogs/2017/07/royal-free-google-deepmind-trial-failed-to-comply-with-data-protection-law/), Information Commissioner's Office, July 3, 2017.*

15 *https://ec.europa.eu/research/participants/portal/desktop/en/opportunities/h2020/topics/imi2-2018-14-03.html*

Communication costs are also an issue. The sheer volume of relevant sensor data can be a challenge, even for fixed factory equipment with a stable, fast internet connection. For mobile equipment and equipment at remote facilities, such as offshore infrastructure or mines, the problem is even harder. By transferring models rather than training data (and thereby saving considerable bandwidth), federated learning can help.

Federated learning's use of the computational resources of the nodes can also save money. Nodes can be inexpensive commodity edge IoT hardware or can even be someone else's financial responsibility entirely (see 4.3.4 Mobile and Edge).

This use case is discussed throughout this report, including in 2.2 Predictive Maintenance and 3 Prototype.

4.1.4 Video Analytics

Machine learning is increasingly being applied to video data. Applications include home security cameras and baby monitors, in-store video for retail analytics, and autonomous vehicles. It's expensive to move lots of video from multiple sources over a network.[16] If those sources do not have fixed connections (such as in the case of autonomous vehicles), it can be almost impossible. Assuming you can move the data,

---

16 *"Why the Future of Machine Learning Is Tiny" (https://petewarden.com/2018/06/11/why-the-future-of-machine-learning-is-tiny/) tells the story of a person with an in-home camera system who experienced significantly higher ISP usage in the month of December than in the rest of the year. Why? Because "his blinking Christmas lights caused the video stream compression ratio to drop dramatically, since so many more frames had differences."*

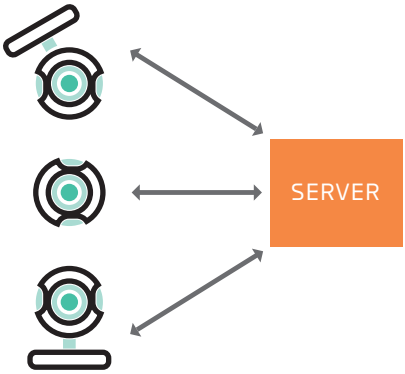


FIGURE 4.1 *Moving video data to a server requires a lot of bandwidth.*

it can be difficult to store it once you get it to the datacenter because of its volume. And when the video is from a home security device, privacy is also a particular concern. These situations present engineering and privacy challenges that make federated learning a great fit.

4.1.5 Corporate IT

The application of machine learning to corporate data is challenging because access to the training data is typically restricted by competitive or legal constraints. If enterprise users install software on their own hardware or in their own cloud, the developer of that software will find it hard to get training data from those users. If the ability to participate in a federated learning network is added as an (optional) feature to such

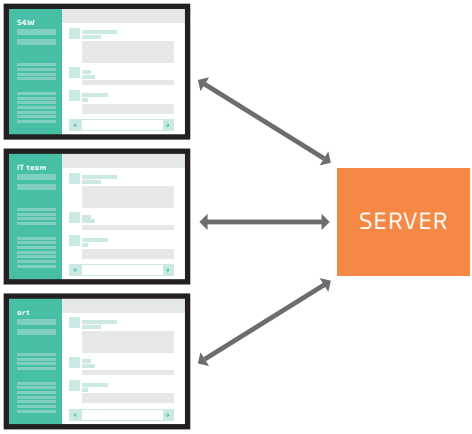


FIGURE 4.2 *Companies are reluctant to share their internal discussions with the developers of corporate IT products.*

software, the developer can train new intelligent features on real user data.

Collaboration tools like Jira and Slack are good examples of this. Atlassian (the developer of Jira) does not have direct access to the data its users (other companies) generate on local installations of Jira. The simplest workaround for Atlassian is to train models using data from its own internal Jira instance. This data has the virtue of being available, but obviously Atlassian's internal use of its product is not typical, and a model trained on this data may perform poorly for the average user. Federated learning could offer the machine learning teams at companies like Atlassian indirect access to the much more representative data generated by their paying customers.

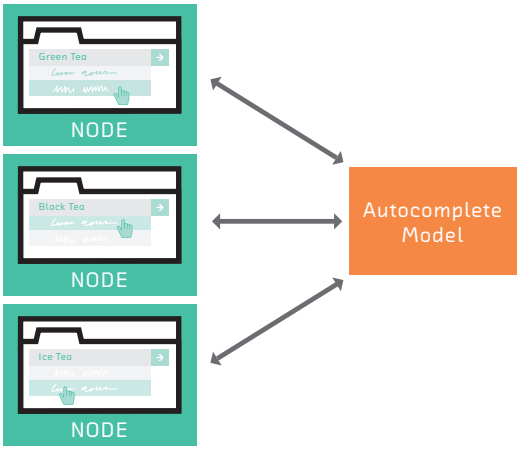


FIGURE 4.3 *URL autocompletion in Firefox.*

4.2 Users

4.2.1 Firefox

Google employees have published a huge amount of academic research on federated learning, but Google's public discussion of its own production use has so far been very high-level. By far the most detailed description of federated learning in a production context in any industry is of its use in the open source web browser Firefox.[17] That article is well worth reading if you plan to use federated learning in a production setting, but we'll point out some notable aspects here.

Like all modern browsers, Firefox offers to autocomplete URLs as you type them. The exact equation that determines the rank of a suggestion currently uses arbitrarily chosen

17 https://florian.github.io/federated-learning-firefox/

coefficients. These would be better determined by machine learning, but Firefox users (understandably) do not want to share their browser history with Firefox's developers. The Firefox federated learning project treats the browsers as nodes in a federated network that learns the optimal autocompletion strategy. Users can opt in to participate in the network, but they do not need to share their browser history with Firefox.

The Firefox project is also a great demonstration of the fact that you don't need to use deep learning to do federated learning (see 2.6.3 Not Only Deep Learning). The model is an SVM, which gets around the difficulty of training deep models on edge devices (see 4.3.2 General-Purpose Tools). This project also benefits from Firefox's built-in telemetry system, which handles the network communication.

4.2.2 Owkin

Owkin[18] is a French healthcare startup with federated learning at the heart of its business model. Owkin sets up the hardware and software infrastructure to train models onsite at hospitals and other medical research institutions, connects these systems into networks, and uses federated learning to combine models across institutions working on similar problems. For example, it might form a federation of hospitals treating patients with a certain medical condition—say, a certain type of cancer—and research institutions working on therapies for that type of cancer. Owkin also uses federated learning to share research models with

18 *https://owkin.com/*

pharmaceutical companies researching therapies for those medical conditions.

Owkin uses federated learning to protect the privacy of medical data, comply with strong European data protection laws, and safeguard commercially sensitive information in federations that may include competing pharmaceutical companies. It puts substantial resources into security and is constantly on the lookout for new threat models. Owkin also faces challenges in regularizing models across institutions that collect somewhat different types of data. This is part of the reason it suits the company to install its own hardware and software, to help maintain consistency at edge sites.

Owkin was founded in 2016 and has about 40 employees. It currently supports around a dozen hospitals and research institutions, and is adding more.

4.2.3 OpenMined

OpenMined[19] is a community of open-source developers who are working to build tools for secure, privacy-preserving machine learning. The core mission of OpenMined is to establish a marketplace that makes it possible to train models on data that is kept private and owned by clients, while allowing models to be shared securely amongst multiple owners. To achieve these broad goals, the community seeks to combine federated learning, differential privacy, and advanced cryptographic techniques in a complete software ecosystem.

The idea is that this three-sided market will enable sources of training data to monetize that data while retaining privacy,

19 https://openmined.org/

will enable sources of compute power to monetize those cycles for training of models, and will enable users of models to pay for the data and compute resources necessary to create them. OpenMined needs to do all this while guarding against bad actors who seek to violate the privacy of users of the marketplace, to poison models with bad training training data, or to steal models.[20] These are ambitious goals! Some of the project's progress is visible in PySyft (see 4.3.1 PySyft).

4.3 Tools and Vendors

At the time of this writing we could not identify any commercial vendors of federated learning solutions. Commercial federated learning requires a rigorous approach to preserving data privacy, which is still an area of intense research. Fortunately, where there is some level of trust between members of a federated network, it is possible to build a federated learning solution using existing tools.

Federated learning is a combination of machine learning and network communication. Rich ecosystems of tools already exist for doing both, and can be used to build a federated learning solution. There are also tools that specialize in federated learning that aim to make it easier to combine the individual parts.

---

20 *See "How to Backdoor Federated Learning" (https://arxiv.org/abs/1807.00459) by Eugene Bagdasaryan et al.*

4.3.1 PySyft

PySyft[21] is an open-source Python library, built by OpenMined (4.2.3 OpenMined), for private federated learning. It provides abstractions on top of popular machine learning libraries that make it easy to move computations to remote clients where the data resides. PySyft currently provides first-class support only for PyTorch, but integration with TensorFlow is on the roadmap.

PySyft is currently under very active development and is more of a research platform than a production-ready federated learning library. Because of its dependence on PyTorch, it is not suitable for federated learning on edge devices, mobile devices, or browser-based clients. It is more suitable in settings where clients are server-based, with larger resource pools, such as private cloud environments where data has a single owner but cannot be moved to a centralized location.

4.3.2 General-Purpose Tools

Federated learning requires software that can distribute computation across multiple physical machines and manage communication between them. Many machine learning libraries already provide this functionality and can, in theory, run on any device with a CPU and a network connection. However, the key difference between federated learning and distributed machine learning is that the machines that participate in federated learning may be heterogeneous, unreliable, and highly resource-constrained. The libraries that currently

21 https://github.com/OpenMined/PySyft

exist for distributed machine learning were not designed for the federated setting and therefore may perform poorly.

We do not expect this functionality gap to last long, though, and while performance may vary, there are still some cases today where federated learning can directly leverage existing machine learning tools.

4.3.3 Browser-Based

Web browsers are a common interface to billions of client devices, each of which has data and computational resources that can be put to use. Libraries like TensorFlow.js make it possible to run machine learning workloads directly in the browser, so that a user's training data never has to leave their machine. Additionally, because training happens on the client's machine, the model can still learn even when the user has no network connection.

TensorFlow.js is a JavaScript library that provides a TensorFlow-like API for doing machine learning directly in the browser. It supports both training and inference and can leverage client-side GPUs for fast computation. The library is maintained under the TensorFlow family of projects, but shares no code with TensorFlow itself and only provides a subset of the full TensorFlow functionality.

4.3.4 Mobile and Edge

Because mobile and edge devices have low computational power and small energy budgets, the computations they run need to be heavily optimized. This usually means the software they run is written in Java or C++, but most existing machine learning tools require Python. This gap has led to the creation

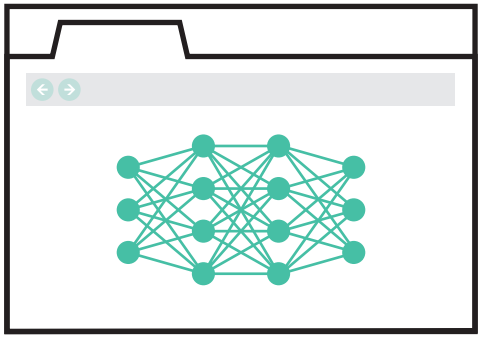


FIGURE 4.4 *Browser-based machine learning tools could open up more distributed possibilities.*

of several lightweight machine learning libraries that are optimized for mobile and have interfaces in Java and C++. Examples are TensorFlow Lite, Caffe2, and Core ML.

Currently these lightweight libraries only support the inference stage of machine learning. They don't provide a way of computing the model updates on-device, as is required for federated learning. One solution is to directly use the full-featured machine learning libraries that support training and provide Java or C++ APIs for mobile development. A second option is to develop custom code for updating your federated learning model.

Federated learning can be applied to any model type that has a proper notion of an incremental update, but among existing tools deep learning libraries are typically best suited for federated learning. Libraries like TensorFlow, Caffe2, MXNet,

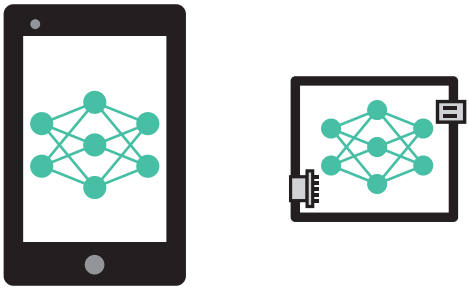


FIGURE 4.5 *Training models on the edge devices is exciting, but library support is immature.*

and Deeplearning4j all make it easy to compute model updates (gradients) and provide APIs in C++ and Java that are amenable to mobile development. Using these full-featured libraries will not be performant in general, but they may still be viable options when on-device resources are less constrained.

While developing customized code is generally undesirable, it may be the best or only viable option given the current state of federated learning tools. The burden of implementing custom deep learning logic may be deemed too high, but simpler solutions like linear models, SVMs, or clustering can be done with relatively little custom code. If a federated learning solution enables applications that were previously impossible, even these simple models can provide significant gains.

CHAPTER 5

Ethics

5.1 Privacy

The macOS and iOS installer contains a dialog box that tells the user:

> Apple believes privacy is a fundamental human right, so every Apple product is designed to:
>
> - Use on-device processing wherever possible
> - Limit the collection and use of data
> - Provide transparency and control over your information
> - Build on a strong foundation of security

Along similar lines, the European Union's General Data Protection Regulation[22] requires that:

> - Data must be collected for a specific purpose.
> - That purpose must be one to which the user has consented.

---

22 *This informal language is taken from "On Purpose and by Necessity: Compliance Under the GDPR" (https://blog.acolyer.org/2018/03/21/on-purpose-and-by-necessity-compliance-under-the-gdpr/).*

> - The data must be necessary to achieve that purpose.
> - It must be deleted when it is no longer necessary for any purpose.

The 2012 White House report "Consumer Data Privacy in a Networked World" argues for a principle of "focused collection":

> Consumers have a right to reasonable limits on the personal data that companies collect and retain. Companies should collect only as much personal data as they need to accomplish [clearly specified purposes]. Companies should securely dispose of or de-identify personal data once they no longer need it, unless they are under a legal obligation to do otherwise.

Most simply, Maciej Ceglowski gave the following advice about data to the Strata 2015 audience in his keynote address "Haunted by Data":

> - Don't collect it.
> - Don't store it.
> - Don't keep it.

By keeping data on a device that belongs to the user, federated learning makes it easier to use machine learning while following these (hopefully self-evident) ethical imperatives and legal requirements. Because the model owner does not have direct access to the data, concerns and liabilities seem to fall away.

But there is a problem with this rosy picture: it is sometimes possible to infer things about the training data from a model. This problem is not unique to federated learning—any time you share the predictions of a trained model you open up this possibility. However, it is worth considering in detail in the context of federated learning for two reasons. First, preserving privacy is one of federated learning's main goals. Second, by distributing training among (potentially untrustworthy) participants, federated learning opens up new attack vectors.

The most indirect way to infer information about the training data requires only the ability to query the model several times. Anyone with indirect access to the model via an API can attempt to attack it in this way. This attack vector is not unique (or any more dangerous) in federated learning.[23] The usual protection against it is *differential privacy*. Differential privacy is a large and mathematically formal field with

---

23 *See e.g., "Membership Inference Attacks Against Machine Learning Models" (https://arxiv.org/abs/1610.05820) by Reza Shokri et al. and "Model Inversion Attacks that Exploit Confidence Information and Basic Countermeasures" (https://dl.acm.org/citation.cfm?id=2813677) by Matt Fredrikson, Somesh Jha, and Thomas Ristenpart.*

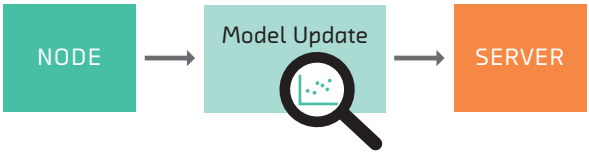


FIGURE 5.1 *It can be possible to infer information about the data on a node from the models it sends to the server.*

applications far beyond machine learning.[24] In a production machine learning context, its application generally means that the server adds noise to the weights of the model before opening it up to users.

In a federated learning setting where the server and nodes are justified in trusting each other, this type of attack is the

---

24 *For a modern, accessible, ML-oriented introduction we recommend "Privacy and Machine Learning: Two Unexpected Allies?" (http://www.cleverhans.io/privacy/2018/04/29/privacy-and-machine-learning.html) by Nicolas Papernot and Ian Goodfellow. For a more general overview, we recommend the short and mercifully informal three-part series "Understanding Differential Privacy and Why It Matters for Digital Rights" (https://www.accessnow.org/understanding-differential-privacy-matters-digital-rights/).*

FIGURE 5.2 *Training data (left) can be reconstructed (right) by a malicious node (images taken from "Deep Models Under the GAN: Information Leakage from Collaborative Deep Learning" (https://arxiv.org/abs/1702.07464) by Briland Hitaj, Giuseppe Ateniese, and Fernando Perez-Cruz).*

only concern. But if the server or nodes are not trustworthy, other kinds of attacks are possible.

For example, the server must be able to directly inspect the node's model in order to average it. But for certain classes of model, the mere fact that a weight has changed tells you a particular feature was present in the training data. For example, suppose a model takes a bag of words as features, and the tenth word in the vocabulary is "dumplings." If a node returns a model where the tenth coefficient of the model has changed, the server or an intermediary may be able to infer that the word "dumplings" was present in the training data on that node.

This attack is more difficult to carry out against modern (and more complex) models in practice. The risk can be mitigated by differential privacy (again) or secure aggregation. The differential privacy approach has each node add noise to its model before sharing it with the server.[25] Secure aggregation protocols make it possible for the server to compute the average of the node models using encrypted copies which it does not have the ability to decrypt. Both these approaches add communication and computation overhead, but that may be a trade-off worth making in highly sensitive contexts.

To sum up our discussion of privacy: by leaving the data at its source, federated learning plugs the most obvious and gaping security hole in distributed machine learning. But it is not a silver bullet. Differential privacy and secure aggregation will both almost certainly be a piece of the security puzzle. These techniques are already in production use—but as with any computer security risk, adversaries will make progress too. Production deployment of federated learning in highly sensitive contexts must be done with care. Eliminating the many threats to a shared model is a work in progress.

5.2 Consent

A node is surrendering computational resources and, as we saw in the previous section, risking the privacy of its data. It is therefore hopefully self-evident that you need to get a

---

25 *See Naman Agarwal et al.'s "cpSGD: Communication-Efficient and Differentially-Private Distributed SGD" (https://arxiv.org/abs/1805.10559).*

user's consent before making them a node in your federated learning network.

But there are more formal, legal arguments for consent. The European Union's General Data Protection Regulation doesn't directly address federated learning (or indeed apply worldwide), but it provides a lens through which to make decisions about machine learning generally. Among other things, the GDPR requires informed and active consent. *Informed* consent is tricky in the context of federated learning because the network asks individuals to share an abstraction of their data and to give up some of the processing power on their devices. These may turn out to be difficult ideas to get across in a way that will withstand legal scrutiny.

The requirement to obtain *active* consent complicates federated learning. If it is a legal requirement to allow a node to withdraw, an active federated learning network must be robust against this possibility in the algorithmic sense (see 2.7 Systems Issues). This is in addition to one of the challenges that the GDPR's "right to be forgotten" poses to machine learning in all forms: what happens to the trained model if the source of the training data withdraws consent?[26]

---

26 *See e.g., Section IV.B of "Slave to the Algorithm? Why a Right to an Explanation Is Probably Not the Remedy You Are Looking For" (http://dx.doi.org/10.2139/ssrn.2972855), by Lilian Edwards and Michael Veale.*

5.3 Environmental Impact

Internet-connected devices will be responsible for 20% of the world's electricity consumption by 2025.[27] In addition to the proliferation of such devices in technologically advanced countries, access to the internet is increasingly considered a human right.[28] With billions of people coming online, it is important to consider the environmental impact of those devices and the data centers required to support them.

In this sense, federated learning is good news. By reducing the amount of information that needs to be sent over the network, this approach to distributed machine learning significantly reduces the power consumed by edge devices (see 6.1 Reducing Communication Costs). And by reducing the amount of data that needs to be stored in data centers, federated learning reduces the need for long-lived, climate-controlled infrastructure.

---

27 *"Tsunami of Data Could Consume One Fifth of Global Electricity by 2025"* (*https://www.theguardian.com/environment/2017/dec/11/tsunami-of-data-could-consume-fifth-global-electricity-by-2025*), The Guardian, *December 11, 2017.*

28 *http://digitallibrary.un.org/record/845728?ln=en*

CHAPTER 6

Future

We have already discussed perhaps the most active area of federated learning research (5.1 Privacy). In this chapter we discuss two more economically and technologically significant open questions: how to further reduce communication costs and personalize models.

6.1 Reducing Communication Costs

The basic premise of federated learning—transferring the models rather than the data—reduces the communication costs of distributed training significantly. But reducing data transfer even further is an area of active research.

Why is it so important to further reduce communication? First, bandwidth (especially upload) is an expensive resource, particularly on consumer mobile connections. Users will refuse to participate in a federated network that uses up all their bandwidth.

Second, avoiding communication is synonymous with conserving power on battery-powered devices. That's because the radio antenna is the most power-hungry part of smartphones and specialized edge hardware. It consumes more energy than a device's sensors or processor, by a factor of tens,

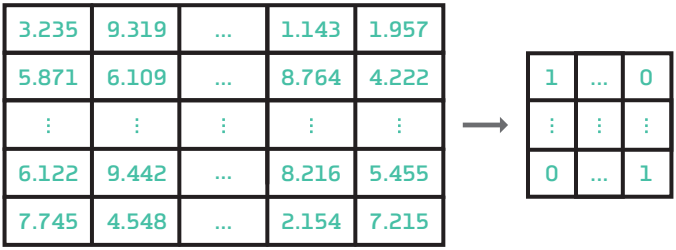


FIGURE 6.1 *It is sometimes possible to compress the array of numbers that define a model, which saves bandwidth.*

hundreds, or even thousands.[29] Users will refuse to participate in a federated network that uses up all their battery.

Clearly, then, it's valuable to reduce data transfer. But how? One set of approaches reduces the size of the trained model that is transferred to and from each node.[30] Options to do this include structured updates (where the training process restricts the model parameters such that they can be parameterized more compactly) and sketched updates (where

29 *See "Why the Future of Machine Learning Is Tiny" (https://petewarden.com/2018/06/11/why-the-future-of-machine-learning-is-tiny/) for more detailed numbers.*

30 *See "Federated Learning: Strategies for Improving Communication Efficiency" (https://arxiv.org/abs/1610.05492) by Jakub Konečný et al.*

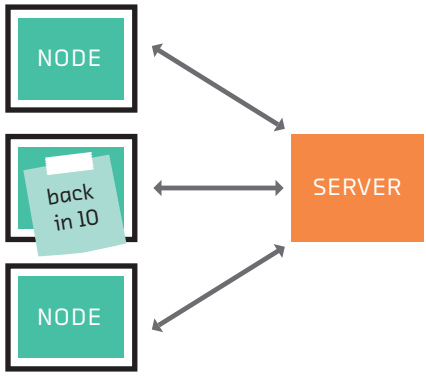


FIGURE 6.2 *A node that is not participating in a particular round of federated learning saves bandwidth.*

the model can be trained in the usual way, but is lossily compressed—i.e., *quantized*—for communication).

Another set of approaches modifies the simple protocol defined in 2.5 A Federated Learning Algorithm to reduce the number of rounds of communication a given node participates in. This not only saves on communication by transferring fewer copies of the model, but speeds up the overall process by eliminating the latency associated with establishing a connection. Some researchers have proposed eliminating rounds by instructing nodes to report back only when they

have accumulated significant changes to the model.[31] Others have suggested decreasing the probability that a given node is asked to participate in a particular round by the server (perhaps in proportion to its power or connection speed).[32]

This is an area in which rapid progress was being made even as we wrote this report. We look forward to further advances that make it easier to use federated learning in parts of the world without robust internet infrastructure, and without wasting energy. We expect significant progress in months rather than years.

6.2 Personalization

In all the training scenarios we've described so far in this report, the server's goal was to use the data on every node to train a single global model. But in situations where a node plans to *apply* the model (not just to contribute to its creation), it will usually care much more that the model captures the patterns in *its* data than any other node's. In other words, each node will care more about the model's accuracy on its test set than on any other.

If the global model has an appropriately flexible

---

31 *See "Deep Gradient Compression: Reducing the Communication Bandwidth for Distributed Training" (https://arxiv.org/abs/1712.01887) by Yujun Lin et al. and "LAG: Lazily Aggregated Gradient for Communication-Efficient Distributed Learning" (https://arxiv.org/abs/1805.09965) by Tianyi Chen et al.*

32 *See Takayuki Nishio and Ryo Yonetani's "Client Selection for Federated Learning with Heterogeneous Resources in Mobile Edge" (https://arxiv.org/abs/1804.08333).*

architecture and was trained on lots of good training data, then it may be better than any local model trained on a single node because of its ability to capture many idiosyncrasies and generalize to new patterns. But it is true that, in principle and sometimes in practice, the user's goal (local performance) can be in tension with the server's (global performance).

Of course each node has the option to use their own local model for inference if they choose, but it would perhaps be better to offer *personalized* federated models. These models would benefit from all the data on the network, but be optimized for application by a particular node. This is an area of current academic research. In "Federated Multi-Task Learning,"[33] Virginia Smith and collaborators frame personalization as a multi-task problem where each user's model is a task, but there exists a structure that relates the tasks. In "Differentially Private Distributed Learning for Language Modeling Tasks,"[34] a team from Samsung consider the example of predictive text on smartphones. They describe training a central model for new users, and allowing each user's model to diverge a little from that central model—but not so far that the model exhibits "catastrophic forgetting" (which in this context would be, e.g., forgetting standard English because a user uses lots of emojis).

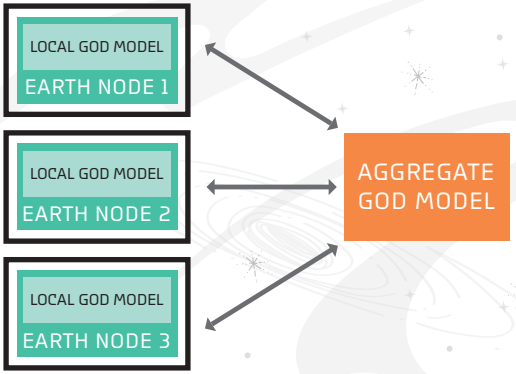
We were not able to learn about the details of any non-research use of personalized federated learning, but it is doubtless coming soon to smartphones (if it isn't already in production).

---

33 https://arxiv.org/abs/1705.10467

34 https://arxiv.org/abs/1712.07473

6.3 Sci-fi Story: Merrily, Merrily, Merrily, Merrily



A short story written by Luc Rioual. Inspired by federated learning.

Here is a god. Her name is Geoff and she is at work.

Geoff has been a god for a very, very long time, and her tour will come to an end at some point soon, once the Earth—her charge—has ceased to be useful.

You see, Geoff is but one of many.

It is not so much that one denomination or another got it right, but that they all got parts right—which is itself, too, a cliché. How do I put this? Gods, like you, tire. They are not architects; their professions do not have built-in respites; they are caretakers at best, which is not meant to badmouth. You try sweeping deserts, combing fields, growing trees. Do you know the time it takes? The will? The water? The focus?

Contrary to popular belief, there is no seventh day. No eighth. Ninth. It is all one long day.

Tours last who knows how long.

At the end there is another contraction, like the first, and Geoff withdraws, vacates for Kokomo, for steel drums, etc., and in her absence unfolds a replacement: an omniscient, a custodian. We don't know her name or temperament, but we know what she'll know, not the facts and figures themselves but their shape: everything that is the case and is not. How is this? It's complicated. It takes time.

Geoff likes her job. She likes her coworkers. Takes no breaks because she does not need them. Geoff has stamina.

Besides her obligations as terrestrial steward, she is charged with prep.

Omniscience is not inbred; it is, like a coat, presented to, put on, twirled in; hands will find tips left in the pockets, tags of authorship sewn by the collar, room in the size, restraint in the cut. Omniscience is not so much about power as it is access, which two are, on Earth, one and the same. Here it is different.

Geoff's job is, then, to some extent, archival, but deeper, more integral. Yes, she stores the daily runoffs and ephemera, receipts and bills and dog-ears, but this is small potatoes, you see, to the bigger crop: her manner of resolution. Yield, here, is second to method. Anyone can keep newspapers, clip bits, but it takes practice to properly sort them, to learn to do it better. This is her purpose: best practice.

As I've said, Geoff is not alone. There are Sarah, Jean, Ida, Kevin, Ross, Jack, Toby, Tara, Ben, Heidi, Tom, Thom, Thomas, Tommy Boy, Diana, Nell, Kristen, and others, whose roles are

all the same: to oversee. There are not yet infinite numbers of gods, realms, so on and so forth, though there is infinite potential; they have not all yet begun, but have, like members of a chorus singing in a round, begun in succession. They are not on different planes but in different spaces, homogeneous in law, hodgepodge in orchestration. Up is always up. Gravity pulls. Time's arrow flies. What the laws stipulate, of course, too, is that no cross-pollination can occur. One would contaminate the other, tear holes in complicated fabrics, bring about Universal End—Geoff would cease; Sarah, Jean, Ida, poof, all, too. Bye bye birdie, yellow brick road, etc. No one knows precisely what another knows. They keep—as pertains practicals—to themselves.

And yet, as has been said—this is chief—Geoff et al. know everything that is the case and is not, Ludwig just a man, Geoff a god. We are talking gods, for god's sake! Omniscience does not mean, here, infallibility. To know everything is not to always make the right decision. Resolution is hard. Will is what we share with Geoff, choice our first gift. Decisions require choices. Choices require knowledge. Knowledge can overwhelm. Geoff's job is to care: verb, "look after and provide for the needs of." Latin states one cannot end a sentence on a preposition, but I can, and I will.

Geoff is, in some respects, a guinea pig, perhaps. There will be more like her. They will not be named Geoff. This will all happen again, but better, in hi-def.

Consider this moral dilemma: you watch someone litter. Do you pick up after them? Scold? Snap? Now multiply that. Multiply that by however many people there might be on a given planet, subtracting a bit for the morally robust, and you

still have several billion people littering. How do you manage this? Where do you intervene? When do you give up? You need more than just what you have before you. You need everything, everyone, a team effort. Supreme courts have nine justices, not one. Everything that is the case and not.

Tommy Boy has no namesake film—Chris Farley a periodontist—Heidi no Princess Di. In Kevin's there's no cutlery to speak of so people everywhere eat with their hands, and in Tara's no one keeps pets in their homes. Jean has blue ducks. Kristen no caves. Sarah lacks yoga. And in Jack's everyone sleeps for days. You would not believe their skin. Their happiness. Jack is proud. All there is is difference.

Things happen. The gods respond (or don't). Effectiveness is measured. Adjustments made to what we might call protocols of assessment—they become smarter. Humans can be tricky. Geoff uploads. Geoff downloads. They all incorporate. What-ifs build out the properties of what-nows. Ida, Ross, Ben, et al.—Geoff and her lively cohort—how shrinks cannot talk, per se, cannot share what they know of their constituents, their worlds, so they share process, they share technique; together, they learn; together, they build best practice; they hone their craft. They pass it down. There is a Foreman, somewhere else, who keeps track of it all. They only call him that: the Foreman. Some have said his name is Evan, but who knows. He disperses updates with consistency and briefs the newbies.

And yet, they are not perfect. Only so much can be done. Geoff is tired, too. What can be done? Drop interest rates? Frogs? Bombs? They do what they can to protect us from ourselves. A nudge here, a net there.

Geoff's replacement will know so much. Everything and more.

People do not read. The poor can go hungry. Buffoons swell and rise like dough. Everything happens in rounds. Row, row, row your boat.

CHAPTER 7

Conclusion

As we discuss throughout this report, particularly in 5.1 Privacy and 6.1 Reducing Communication Costs, federated learning is not a perfect solution to the two problems mentioned in 1 Introduction: training data is often private and communication is often expensive. But federated learning *is* a good start!

It addresses problems that affect the most regulated, competitive, and profitable industries. It also has the potential to be of huge benefit to worldwide healthcare outcomes. For this reason, we expect distributed, privacy-preserving, communication-minimizing machine learning to remain an active area of research and commercial experimentation for many years.

In moving the majority of the work to the edge, federated learning is part of the trend to move machine learning out of the data center, for reasons that include speed and cost. But in federated learning, the edge nodes create and improve the model (rather than merely applying it). In this sense, federated learning goes far beyond what people usually mean when they talk about edge AI.

Federated learning points to a future in which we work collectively to apply machine learning to some of toughest problems that humanity faces, while each retaining control over our own data.

APPENDIX

federated.py

In this appendix we illustrate the core functionality of `federated.py`, our PyTorch federated learning simulation module. The module consists of two classes, one representing a node and the other the server. Each node object has a subset of the training data, and a `train` method that initiates a single epoch of training then returns its model and the amount of training data it used.

The server object manages connections to many nodes. It has two important methods. The user of the module calls `round` repeatedly to do federated learning:

```
def round(self):
    """
    Do a round of federated learning:

     - instruct each node to train and return its model
     - replace the server model with the weighted average of the node models
     - replace the node models with the new server model

    Nodes with `participant=False` train but are not
    included in the weighted average and do not receive a
    copy of the server model.
    """
    updates = [node.train() for node in self.nodes]
    self.fedavg([u for u, node in zip(updates, self.nodes) if
    node.participant])
    self.push_model(node for node in self.nodes if node.participant)
```

The method `fedavg` implements the federated averaging calculation:

```
def fedavg(self, updates):
    """
    Replace the server model with the weighted average of
    the node models. `updates` is a list of dictionaries,
    one for each node, each of which has `state_dict` (the
    weights on that node) and `n_samples` (the amount of
    training data on that node).
    """
    N = sum(u["n_samples"] for u in updates)
    for key, value in self.model.state_dict().items():
        weight_sum = (u["state_dict"][key] * u["n_samples"]
        for u in updates)
        value[:] = sum(weight_sum) / N
```

Aside from the systems and networking issues discussed in this report, these two methods are a complete federated learning implementation.

About Cloudera Fast Forward Labs

Cloudera Fast Forward Labs is an applied machine learning research group. We help organizations recognize and develop new product and business opportunities through emerging technologies.

The Cloudera Fast Forward Labs Federated Learning report is brought to you by Alice Albrecht, Grant Custer, Brian Goral, Micha Gorelick, Seth Hendrickson, Mike Lee Williams, Hilary Mason, Ryan Micallef, Emanuel Moss, Nisha Muktewar, Bethann Noble, Justin Norman, Shioulin Sam, Friederike Schüür, Danielle Thorp, and the rest of the Cloudera team. Illustrations by Beste Miray Doğan.

https://fastforwardlabs.com

Federated Learning makes it possible to build machine learning systems without direct access to training data. The data remains in its original location, which helps to ensure privacy and reduces communication costs. Federated learning is a great fit for smartphones and edge hardware, healthcare and other privacy-sensitive use cases, and industrial applications such as predictive maintenance.