A Gentle Introduction to Decentralized ML

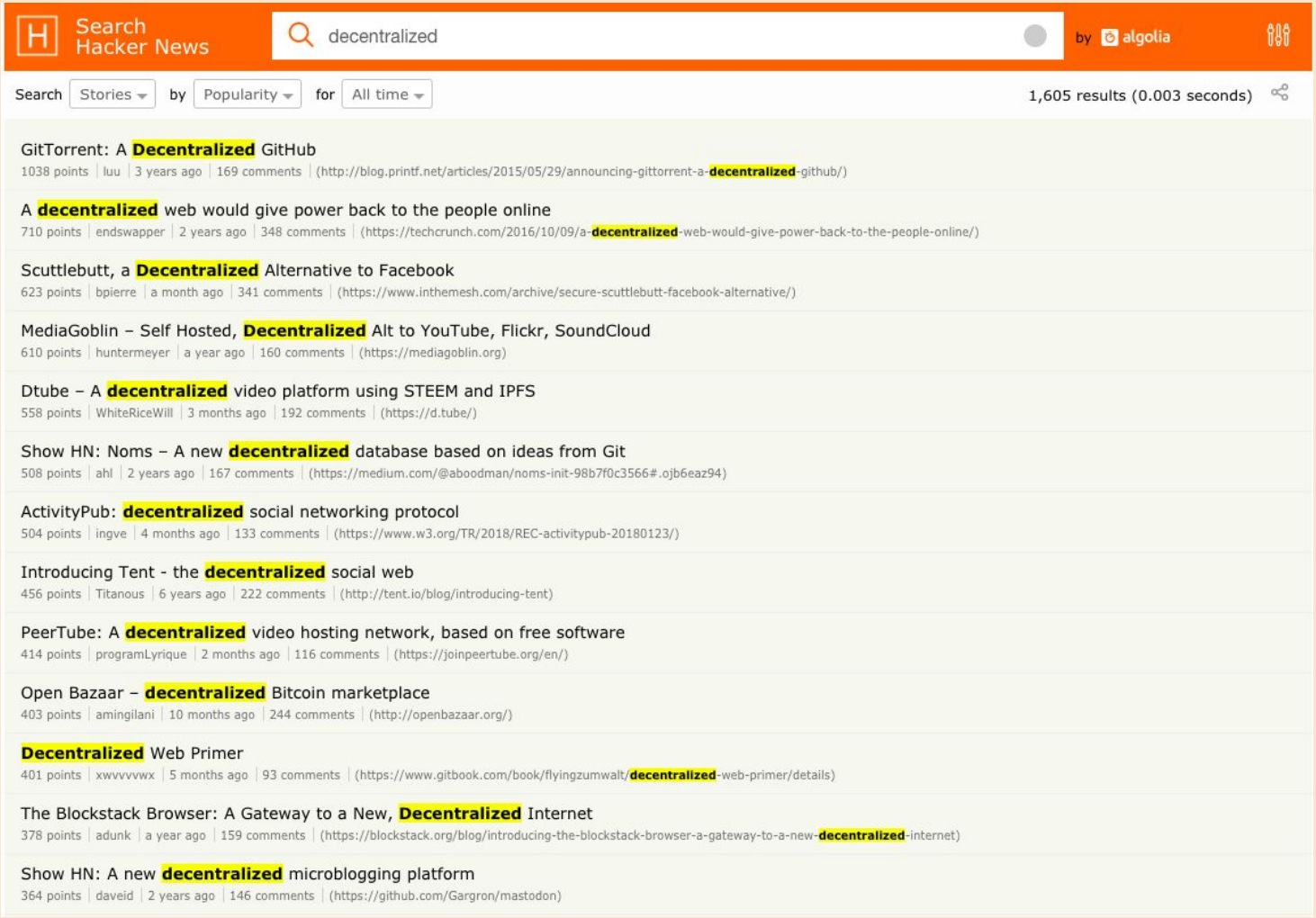
{ SAMSUNG NEXT } mynameisvinn

1. History

1. History
2. Benefits

1. History
2. Benefits
3. Implementation

1. History
2. Benefits
3. Implementation
4. Remaining Challenges (Opportunities)


Search Hacker News
decentralized
by algolia
Search Stories by Popularity for All time
1,605 results (0.003 seconds)
GitTorrent: A Decentralized GitHub
1038 points | luu | 3 years ago | 169 comments | (http://blog.printf.net/articles/2015/05/29/announcing-gittorrent-a-decentralized-github/)
A decentralized web would give power back to the people online
710 points | endswapper | 2 years ago | 348 comments | (https://techcrunch.com/2016/10/09/a-decentralized-web-would-give-power-back-to-the-people-online/)
Scuttlebutt, a Decentralized Alternative to Facebook
623 points | bpierre | a month ago | 341 comments | (https://www.inthemesh.com/archive/secure-scuttlebutt-facebook-alternative/)
MediaGoblin – Self Hosted, Decentralized Alt to YouTube, Flickr, SoundCloud
610 points | huntermeyer | a year ago | 160 comments | (https://mediagoblin.org)
Dtube – A decentralized video platform using STEEM and IPFS
558 points | WhiteRiceWill | 3 months ago | 192 comments | (https://d.tube/)
Show HN: Noms – A new decentralized database based on ideas from Git
508 points | ahl | 2 years ago | 167 comments | (https://medium.com/@aboodman/noms-init-98b7f0c3566#.ojb6eaz94)
ActivityPub: decentralized social networking protocol
504 points | ingve | 4 months ago | 133 comments | (https://www.w3.org/TR/2018/REC-activitypub-20180123/)
Introducing Tent - the decentralized social web
456 points | Titanous | 6 years ago | 222 comments | (http://tent.io/blog/introducing-tent)
PeerTube: A decentralized video hosting network, based on free software
414 points | programLyrique | 2 months ago | 116 comments | (https://joinpeertube.org/en/)
Open Bazaar – decentralized Bitcoin marketplace
403 points | amingilani | 10 months ago | 244 comments | (http://openbazaar.org/)
Decentralized Web Primer
401 points | xwvvvvwx | 5 months ago | 93 comments | (https://www.gitbook.com/book/flyingzumwalt/decentralized-web-primer/details)
The Blockstack Browser: A Gateway to a New, Decentralized Internet
378 points | adunk | a year ago | 159 comments | (https://blockstack.org/blog/introducing-the-blockstack-browser-a-gateway-to-a-new-decentralized-internet)
Show HN: A new decentralized microblogging platform
364 points | daveid | 2 years ago | 146 comments | (https://github.com/Gargron/mastodon)

Decentralize everything… including machine learning!

“The aim in federated learning is to fit a model to data, $\{\mathbf{X}_1, ..., \mathbf{X}_m\}$, generated by m distributed nodes. Each node, $t \in [m]$, collects data in a non-IID manner across the network, with data on each node being generated by a distinct distribution $\mathbf{X}_t \sim \mathbf{P}_t$. The number of data points on each node, n_t, may also vary significantly, and there may be an underlying structure present that captures the relationship amongst nodes and their associated distributions.”

Federated Optimization: Distributed Machine Learning for On-Device Intelligence

Jakub Konečný
University of Edinburgh
kubo.konecny@gmail.com

H. Brendan McMahan
Google
mcmahan@google.com

Daniel Ramage
Google
dramage@google.com

Peter Richtárik
University of Edinburgh
peter.richtarik@ed.ac.uk

October 11, 2016

Abstract

We introduce a new and increasingly relevant setting for distributed optimization in machine learning, where the data defining the optimization are unevenly distributed over an extremely large number of nodes. The goal is to train a high-quality centralized model. We refer to this setting as *Federated Optimization*. In this setting, communication efficiency is of the utmost importance and minimizing the number of rounds of communication is the principal goal.

A motivating example arises when we keep the training data locally on users' mobile devices instead of logging it to a data center for training. In federated optimization, the devices are used as compute nodes performing computation on their local data in order to update a global model. We suppose that we have extremely large number of devices in the network — as many as the number of users of a given service, each of which has only a tiny fraction of the total data available. In particular, we expect the number of data points available locally to be much smaller than the number of devices. Additionally, since different users generate data with different patterns, it is reasonable to assume that no device has a representative sample of the overall distribution.

We show that existing algorithms are not suitable for this setting, and propose a new algorithm which shows encouraging experimental results for sparse convex problems. This work also sets a path for future research needed in the context of federated optimization.

1 Introduction

Communication-Efficient Learning of Deep Networks from Decentralized Data

H. Brendan McMahan **Eider Moore** **Daniel Ramage** **Seth Hampson** **Blaise Agüera y Arcas**
Google, Inc., 651 N 34th St., Seattle, WA 98103 USA

Abstract

Modern mobile devices have access to a wealth of data suitable for learning models, which in turn can greatly improve the user experience on the device. For example, language models can improve speech recognition and text entry, and image models can automatically select good photos. However, this rich data is often privacy sensitive, large in quantity, or both, which may preclude logging to the data center and training there using conventional approaches. We advocate an alternative that leaves the training data distributed on the mobile devices, and learns a shared model by aggregating locally-computed updates. We term this decentralized approach *Federated Learning*.

We present a practical method for the federated learning of deep networks based on iterative model averaging, and conduct an extensive empirical evaluation, considering five different model architectures and four datasets. These experiments demonstrate the approach is robust to the unbalanced and non-IID data distributions that are a defining characteristic of this setting. Communication costs are the principal constraint, and we show a reduction in required communication rounds by 10–100× as compared to synchronized stochastic gradient descent.

promise of greatly improving usability by powering more intelligent applications, but the sensitive nature of the data means there are risks and responsibilities to storing it in a centralized location.

We investigate a learning technique that allows users to collectively reap the benefits of shared models trained from this rich data, without the need to centrally store it. We term our approach *Federated Learning*, since the learning task is solved by a loose federation of participating devices (which we refer to as *clients*) which are coordinated by a central *server*. Each client has a local training dataset which is never uploaded to the server. Instead, each client computes an update to the current global model maintained by the server, and only this update is communicated. This is a direct application of the principle of *focused collection* or *data minimization* proposed by the 2012 White House report on privacy of consumer data [39]. Since these updates are specific to improving the current model, there is no reason to store them once they have been applied.

A principal advantage of this approach is the decoupling of model training from the need for direct access to the raw training data. Clearly, some trust of the server coordinating the training is still required. However, for applications where the training objective can be specified on the basis of data available on each client, federated learning can significantly reduce privacy and security risks by limiting the attack surface to only the device, rather than the device and

FEDERATED LEARNING: STRATEGIES FOR IMPROVING COMMUNICATION EFFICIENCY

Jakub Konečný, **H. Brendan McMahan, Felix X. Yu, Ananda Theertha Suresh & Dave Bacon**
Google
{konkey,mcmahan,felixyu,theertha,dabacon}@google.com

Peter Richtárik
King Abdullah University of Science and Technology (KAUST), Thuwal, Saudi Arabia
University of Edinurgh, Edinburgh, Scotland
Peter.Richtarik@kaust.edu.sa

ABSTRACT

Federated Learning is a machine learning setting where the goal is to train a high-quality centralized model while training data remains distributed over a large number of clients each with unreliable and relatively slow network connections. We consider learning algorithms for this setting where on each round, each client independently computes an update to the current model based on its local data, and communicates this update to a central server, where the client-side updates are aggregated to compute a new global model. The typical clients in this setting are mobile phones, and communication efficiency is of the utmost importance.

In this paper, we propose two ways to reduce the uplink communication costs: *structured updates*, where we directly learn an update from a restricted space parametrized using a smaller number of variables, e.g. either low-rank or a random mask; and *sketched updates*, where we learn a full model update and then compress it using a combination of quantization, random rotations, and subsampling before sending it to the server. Experiments on both convolutional and recurrent networks show that the proposed methods can reduce the communication cost by two orders of magnitude.

1 INTRODUCTION

As datasets grow larger and models more complex, training machine learning models increasingly requires distributing the optimization of model parameters over multiple machines. Existing machine learning algorithms are designed for highly controlled environments (such as data centers) where the data is distributed among machines in a balanced and i.i.d. fashion, and high-throughput

Differentially Private Federated Learning: A Client Level Perspective

Robin C. Geyer[1,2], **Tassilo Klein**[1], **Moin Nabi**[1]
SAP SE[1], ETH Zurich[2]
geyerr@ethz.ch*, tassilo.klein@sap.com, m.nabi@sap.com

Abstract

Federated learning is a recent advance in privacy protection. In this context, a trusted curator aggregates parameters optimized in decentralized fashion by multiple clients. The resulting model is then distributed back to all clients, ultimately converging to a joint representative model without explicitly having to share the data. However, the protocol is vulnerable to differential attacks, which could originate from any party contributing during federated optimization. In such an attack, a client's contribution during training and information about their data set is revealed through analyzing the distributed model. We tackle this problem and propose an algorithm for client sided differential privacy preserving federated optimization. The aim is to hide clients' contributions during training, balancing the trade-off between privacy loss and model performance. Empirical studies suggest that given a sufficiently large number of participating clients, our proposed procedure can maintain client-level differential privacy at only a minor cost in model performance.

1 Introduction

Lately, the topic of security in machine learning is enjoying increased interest. This can be largely attributed to the success of big data in conjunction with deep learning and the urge for creating and processing ever larger data sets for data mining. However, with the emergence of more and more machine learning services becoming part of our daily lives, making use of our data, special measures must be taken to protect privacy. Unfortunately, anonymization alone often is not sufficient [7, 2] and standard machine learning approaches largely disregard privacy aspects.

In federated learning [5] a model is learned by multiple clients in decentralized fashion. Learning is shifted to the clients and only learned parameters are centralized by a trusted curator. This curator then distributes an aggregated model back to the clients.

Clients not revealing their data is an advance in privacy protection, however, when a model is learned in conventional way, its parameters reveal information about the data that was used during training. In order to solve this issue, the concept of differential privacy (dp) [3] for learning algorithms was proposed by [1]. The aim is to ensures a learned model does not reveal whether a certain data point was used during training.

Why Decentralize ML?

Latency

Latency

Regulation

Latency

Regulation

Privacy

Latency

Regulation

Privacy

Offline

Latency

Regulation

Privacy

Offline

Power

Latency

Regulation

Privacy

Offline

Power

Sensors

But How Does It Actually Work?

Algorithm 4 Federated SVRG (FSVRG)

parameters: h = stepsize, data partition $\{\mathcal{P}_k\}_{k=1}^K$,
diagonal matrices $A, S_k \in \mathbb{R}^{d\times d}$ for $k \in \{1,\dots,K\}$
for $s = 0, 1, 2, \dots$ **do** ▷ Overall iterations
  Compute $\nabla f(w^t) = \frac{1}{n}\sum_{i=1}^n \nabla f_i(w^t)$
  for $k = 1$ to K **do in parallel** over nodes k ▷ Distributed loop
    Initialize: $w_k = w^t$ and $h_k = h/n_k$
    Let $\{i_t\}_{t=1}^{n_k}$ be random permutation of $\mathcal{P}_k$
    for $t = 1, \dots, n_k$ **do** ▷ Actual update loop
      $w_k = w_k - h_k\left(S_k\left[\nabla f_{i_t}(w_k) - \nabla f_{i_t}(w^t)\right] + \nabla f(w^t)\right)$
    end for
  end for
  $w^t = w^t + A\sum_{k=1}^K \frac{n_k}{n}(w_k - w^t)$ ▷ Aggregate
end for

Pseudocode

Until Converged:

1. Select a random subset (e.g. 1000) of the (online) clients

Pseudocode

Until Converged:

1. Select a random subset (e.g. 1000) of the (online) clients
2. Send current parameters θ_t to those clients

Pseudocode

Until Converged:

1. Select a random subset (e.g. 1000) of the (online) clients
2. Send current parameters θ_t to those clients

Selected Client K

1. Receive θ_t from server

Pseudocode

Until Converged:

1. Select a random subset (e.g. 1000) of the (online) clients
2. Send current parameters θ_t to those clients

Selected Client K

1. Receive θ_t from server
2. Run some number of minibatch SGD steps on local {X,y}, producing θ'

Pseudocode

Until Converged:

1. Select a random subset (e.g. 1000) of the (online) clients
2. Send current parameters θ_t to those clients

Selected Client K

1. Receive θ_t from server
2. Run some number of minibatch SGD steps on local {X,y}, producing θ'
3. Return θ' to server

Pseudocode

Until Converged:

1. Select a random subset (e.g. 1000) of the (online) clients
2. Send current parameters θ_t to those clients

   Selected Client K

   1. Receive θ_t from server
   2. Run some number of minibatch SGD steps on local {X,y}, producing θ'
   3. Return θ' to server

3. $\theta_{t+1} = \theta_t$ + data-weighted average of client updates

Remaining Challenges

Founders: if you can solve these problems, please find me or McCall :)

Remaining Challenges

- Massively Distributed
  - Training data is stored across a very large number of devices

Remaining Challenges

- Massively Distributed
  - Training data is stored across a very large number of devices
- Limited Communication
  - Only a handful of rounds of unreliable communication with each devices

Remaining Challenges

- Massively Distributed
  - Training data is stored across a very large number of devices
- Limited Communication
  - Only a handful of rounds of unreliable communication with each devices
- Unbalanced Data
  - Some devices have few examples, some have orders of magnitude more

Remaining Challenges

- Massively Distributed
  - Training data is stored across a very large number of devices
- Limited Communication
  - Only a handful of rounds of unreliable communication with each devices
- Unbalanced Data
  - Some devices have few examples, some have orders of magnitude more
- Non-IID Data
  - Data on each device reflects one individual's usage pattern

Remaining Challenges

- Massively Distributed
  - Training data is stored across a very large number of devices
- Limited Communication
  - Only a handful of rounds of unreliable communication with each devices
- Unbalanced Data
  - Some devices have few examples, some have orders of magnitude more
- Non-IID Data
  - Data on each device reflects one individual's usage pattern
- Unreliable Compute Nodes
  - Devices go offline unexpectedly; expect faults and adversaries

Remaining Challenges

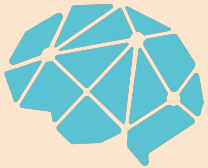
- Massively Distributed
  - Training data is stored across a very large number of devices
- Limited Communication
  - Only a handful of rounds of unreliable communication with each devices
- Unbalanced Data
  - Some devices have few examples, some have orders of magnitude more
- Non-IID Data
  - Data on each device reflects one individual's usage pattern
- Unreliable Compute Nodes
  - Devices go offline unexpectedly; expect faults and adversaries
- Dynamic Data Availability
  - The subset of data available is non-constant, e.g. time-of-day vs. country

Thank you!

Let’s see the code!

(github.com/mynameisvinn/aidecentralized)

Frameworks, Protocols, Services

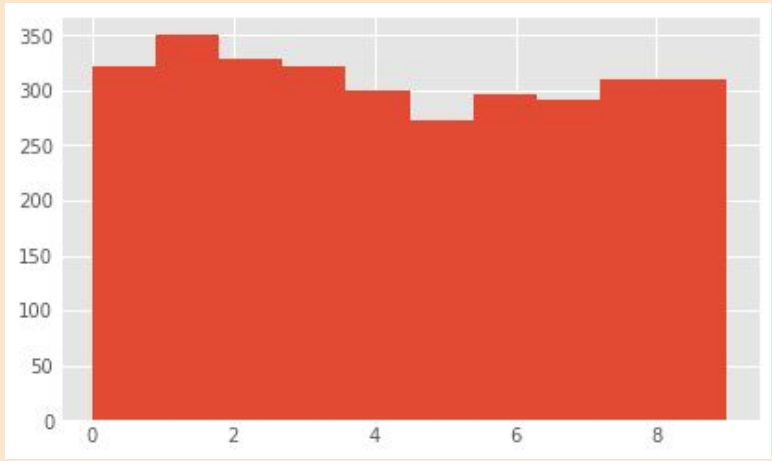


MNIST, the “hello world” of machine learning

```python
class Client(object):
    def __init__(self, sess, n_features, n_classes):
        self.sess = sess

        self.X_ = tf.placeholder(tf.float32, shape=[None, n_features])
        self.y_ = tf.placeholder(tf.float32, shape=[None, n_classes])
        self.w1 = tf.Variable(tf.random_uniform([n_features, n_classes]), name="w1")

        z1 = tf.matmul(self.X_, self.w1)
        probs = tf.nn.softmax(z1)

        self.loss = tf.losses.log_loss(labels=self.y_, predictions=probs)
        self.op = tf.train.AdamOptimizer(0.01).minimize(self.loss)

        predictions = tf.argmax(probs, 1)
        correct_prediction = tf.equal(predictions, tf.argmax(self.y_, 1))
        self.accuracy = tf.reduce_mean(tf.cast(correct_prediction, "float"))
```

```
with tf.Session() as sess:
    Master = Client(sess, n_features, n_classes)
    sess.run(tf.global_variables_initializer())
    print("untrained accuracy: ", Master.score(X_test, y_test))
    Master.load(w1_new)  # use average weights from clients
    print("accuracy, using updated weights: ", Master.score(X_test, y_test))
```

```
untrained accuracy:  0.110964
accuracy, using updated weights:  0.886042
```


350
300
250
200
150
100
50
0
0
2
4
6
8