2024 INFONET Presentation

Journal Club Presentation

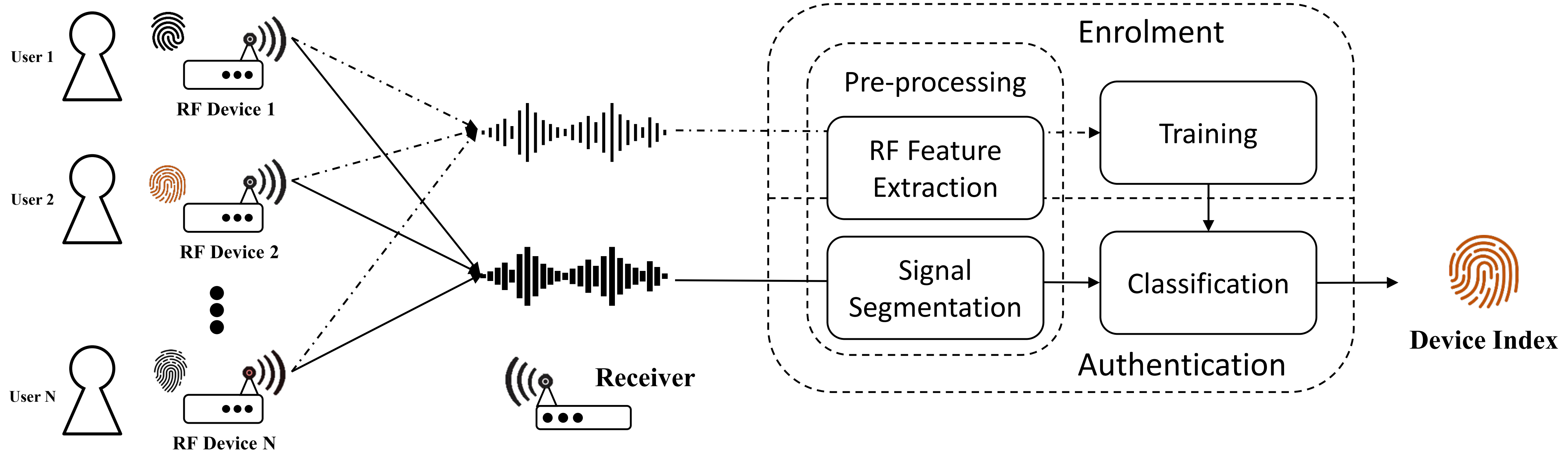
Large Scale Distributed Deep Networks, NIPS, 2012

Jusung Kang

2024.03.22

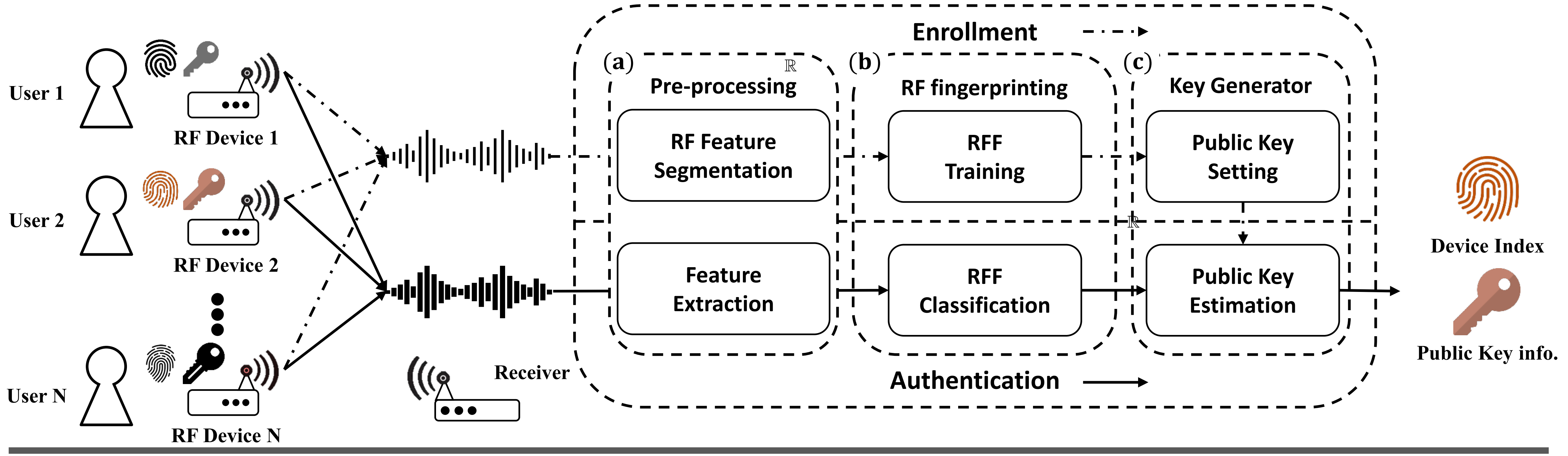
Preview - RF Fingerprinting

- **RF Feature 기반 Physical Layer Authentication 시스템**
  - ‘Intrinsic imperfection’ 으로 인한 RF domain 에서의 신호 특성
    - Transient, Preamble, I/Q imbalance, etc.
    - ***Real domain, $\mathbb{R}$, 에서 동작하는 RF Fingerprinting.***



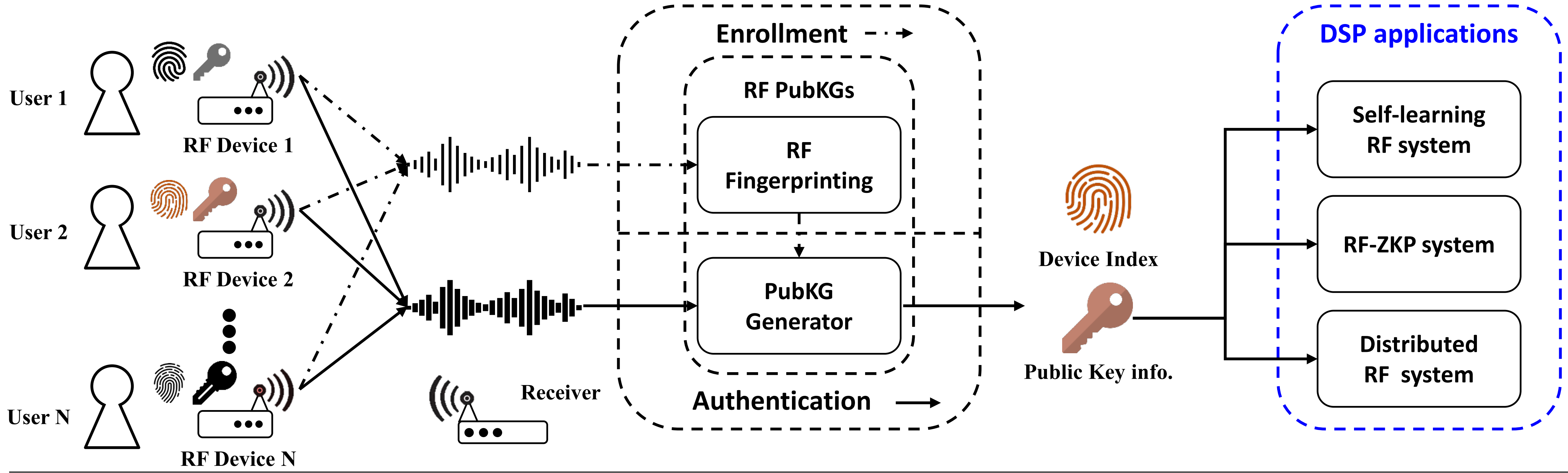
Preview - RF based Cryptographic Sequence Generator

- RF Feature 기반 Cryptographic Sequence 생성 시스템
  - ***Cryptographic sequence (w. GF(2)) 로의 mapping mechanism 제안***
    - Cryptographic application: Public Key Authentication.
    - ***Finite Field domain, $\mathbb{N}$, $GF(q)$, 에서 동작하는 RF based Public Key Sequences.***



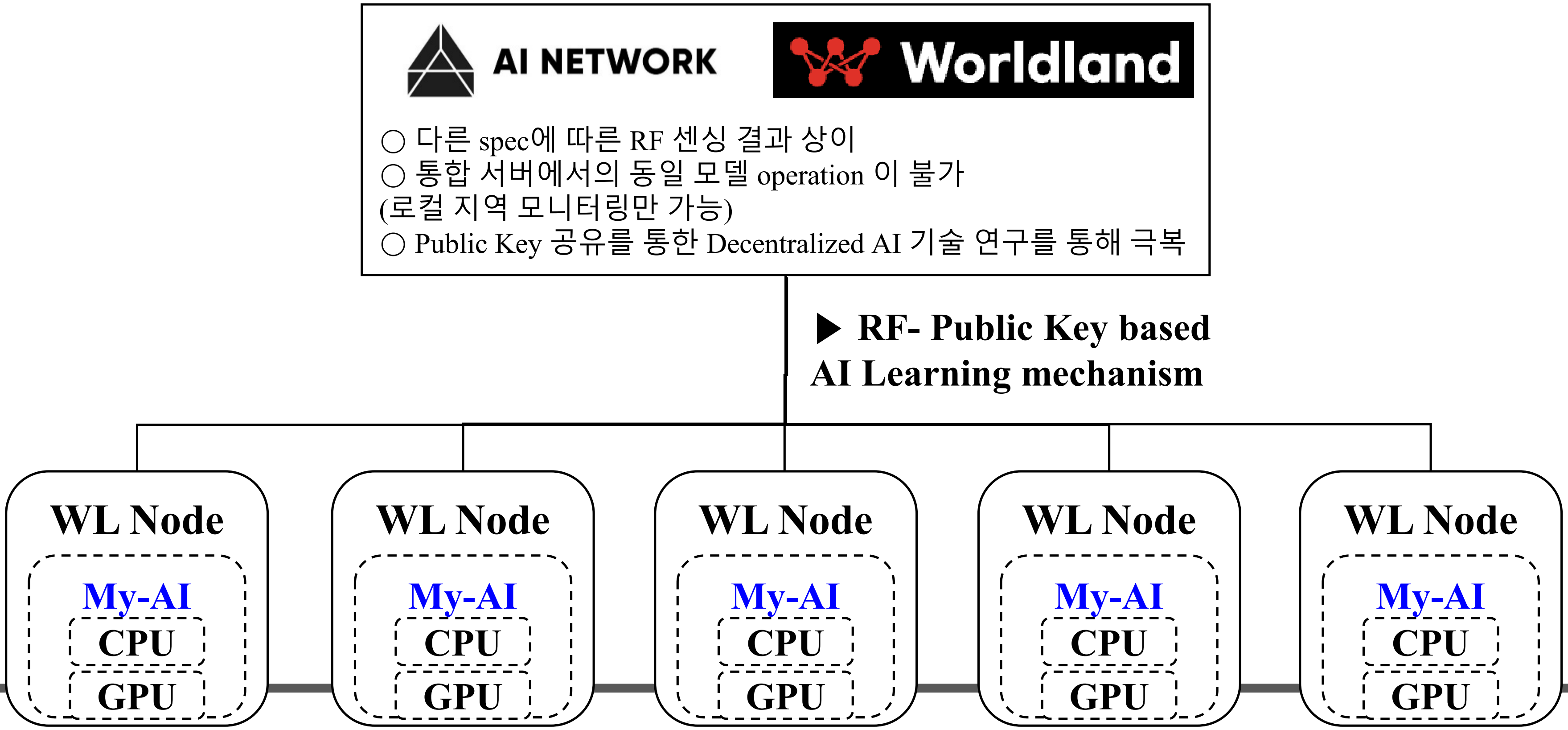
Preview – What's for next?

- **RF Fingerprinting application in Digital Signal Processing domain.**
  - ***마침내 Digital domain, ℕ, 에서 processing 가능하게 된 RF Intrinsic Features***
    - Digitized Feature Key 기반 Application research
      - Out of Distribution, Zero Knowledge Proof, Distributed AI, e.t.c.…



Preview – [Distributed AI] Distributed RF Fingerprinting system

- **[Distributed AI] Public key based Distributed RF Fingerprinting system.**
  - 1. 분산 노드 환경하에서의 모델 학습을 위한 AI 경량화 기술 개발
  - 2. Multi-agents 간의 상호 협력을 통한 Decentralized AI 기술 연구



○ 다른 spec에 따른 RF 센싱 결과 상이
○ 통합 서버에서의 동일 모델 operation 이 불가
(로컬 지역 모니터링만 가능)
○ Public Key 공유를 통한 Decentralized AI 기술 연구를 통해 극복

Node Spec 1 :
- 1G SPS, 5dbi antenna,
- Center Freq. 400MHz
- E.t.c.

Node Spec 2 :
- 100M SPS, 3dbi antenna,
- Center Freq. 10MHz
- E.t.c.

Titles

- **Dean, Jeffrey**, Greg S. Corrado, Rajat Monga, Kai Chen, Matthieu Devin, Quoc V. Le, Mark Z. Mao, Marc' Aurelio Ranzato, Andrew Senior, Paul Tucker, Ke Yang, **Andrew Y. Ng,** "***Large scale distributed deep networks***." in Proc. Adv. Neural Inf. Process. Syst. (NeurIPS), vol. 25, 2012, pp. 1223–1231.

  - **Search Keyword**
    - Distributed AI
    - Federated AI

  - **The aim of this research**
    - The problem of training a deep network with billions of parameters using tens of thousands of CPU cores.

  - **Warning!!!**
    - Paper at the 2012
      - Before the AlexNet, It consider the CPU cores,
    - We aim to consider how the distributed AI networks are considered at the time of beginning.

Before starting the presentation…

- **Aim of search**
  - Understanding the Origin of the Distributed AI research field
    - Concepts, Core ideas, Approach intuitions, etc….

    - Q) How many peoples are read this paper?

    - Most research paper ideas originate from the Origin paper in that field.
      - It’s importance of the origin paper
        - Has anyone read today’s presentation paper?
      - One way to understand new research field efficiently => Find the origin paper.
        - It can represent the CORE IDEA in paper writing efficiently without referencing others.

Introduction - contributions

- **In rescent days, the use of GPUs has shown a significant advance in the training of deep networks**
  - It works well in a condition of non-bottleneck within the cpu-to-gpu process
  - Less attractive for the problem of large-sized networks

- **Contributions:** Proposal of optimization algorithms for large-scale clusters of machnes (distributed learning systems).
  - The DistBelief – Software framework
    - **Model parallelism**
      - Within a machine (via multithreading)
      - Across machine (via message passing)

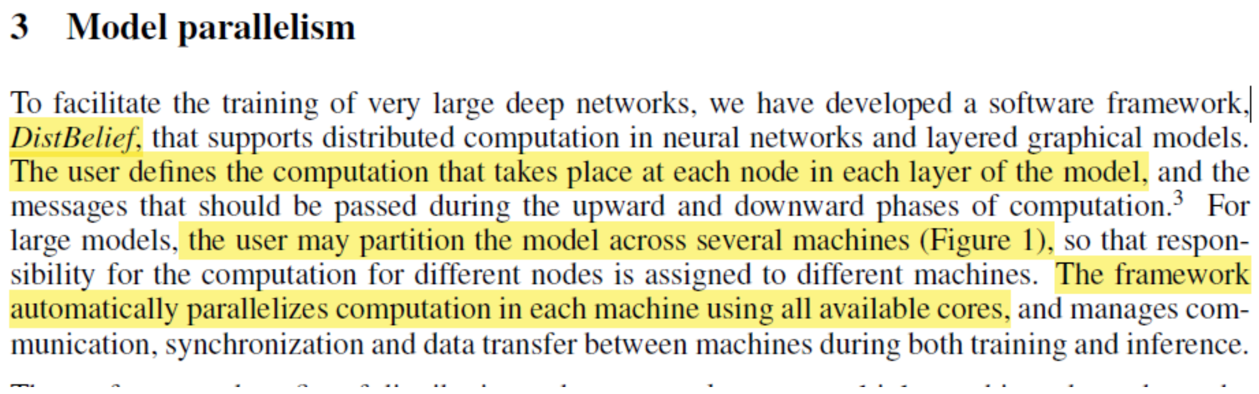
    - **Data parallelism**
      - Downpour SGD:
        - An asynchronous SGD method supporting replica learning.
      - Sandblaster L-BFGS:
        - An implementation method of Sandblaster optimization-based L-BFGS (Limited memory Broyden-Fletcher-Goldfarb-Shanno) supporting batch optimization for model parallelism.

Introduction - contributions

- **In rescent days, the use of GPUs has shown a significant advance in the training of deep networks**
  - It works well in a condition of non-bottleneck within the cpu-to-gpu process
  - Less attractive for the problem of large-sized networks
- **Findings** – for solving the large-scale nonconvex optimization;
  - Asynchronous SGD works ‘very‘ well for large scale nonconvex optimization with Adagrad.
  - L-BFGS is competitive with the SGD approaches

Model parallelism

- How we can partition the model into multiple machines? **- Model parallelism**
  - Users can (may) define the partitions of the model within the framework.
    - Performance depends on **the connectivity structure** and **computation cost**
      - 144 partitions for a large model
      - 8 or 16 partitions for a modestly size model
        - It depends on the user….??
    - Issue) Waiting for the single slowest machine
      - Optimization process applied (Data parallelism)



3 Model parallelism

To facilitate the training of very large deep networks, we have developed a software framework, *DistBelief*, that supports distributed computation in neural networks and layered graphical models. The user defines the computation that takes place at each node in each layer of the model, and the messages that should be passed during the upward and downward phases of computation.[3] For large models, the user may partition the model across several machines (Figure 1), so that responsibility for the computation for different nodes is assigned to different machines. The framework automatically parallelizes computation in each machine using all available cores, and manages communication, synchronization and data transfer between machines during both training and inference.

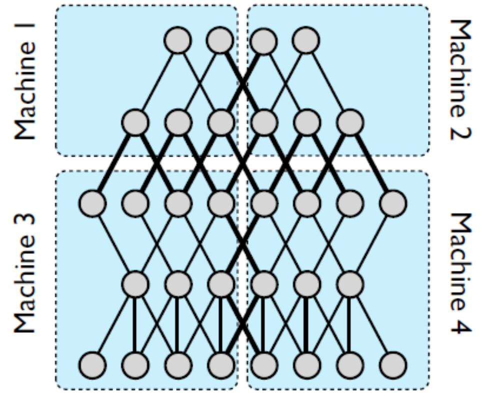


Figure 1: An example of model parallelism in DistBelief. A five layer deep neural network with local connectivity is shown here, partitioned across four machines (blue rectangles). Only those nodes with edges that cross partition boundaries (thick lines) will need to have their state transmitted between machines. Even in cases where a node has multiple edges crossing a partition boundary, its state is only sent to the machine on the other side of that boundary once. Within each partition, computation for individual nodes will the parallelized across all available CPU cores.

Data parallelism – Distributed Optimization algorithms

- How should the data be considered to solve the slowest machine issues? – **Data parallelism**
  - Distribution problem of the training procedure across multiple model instances
    - A centralized sharded parameter server
    - [Goal] To tolerate variance in the processing speed and the wholesale failure of the model.
      - 1) Simultaneously process the distinct training examples
      - 2) combine their results to optimize the object function
  - Two large-scale distributed optimization procedure
    - Downpour SGD: Online method
    - Sandblaster L-BFGS: Batch method

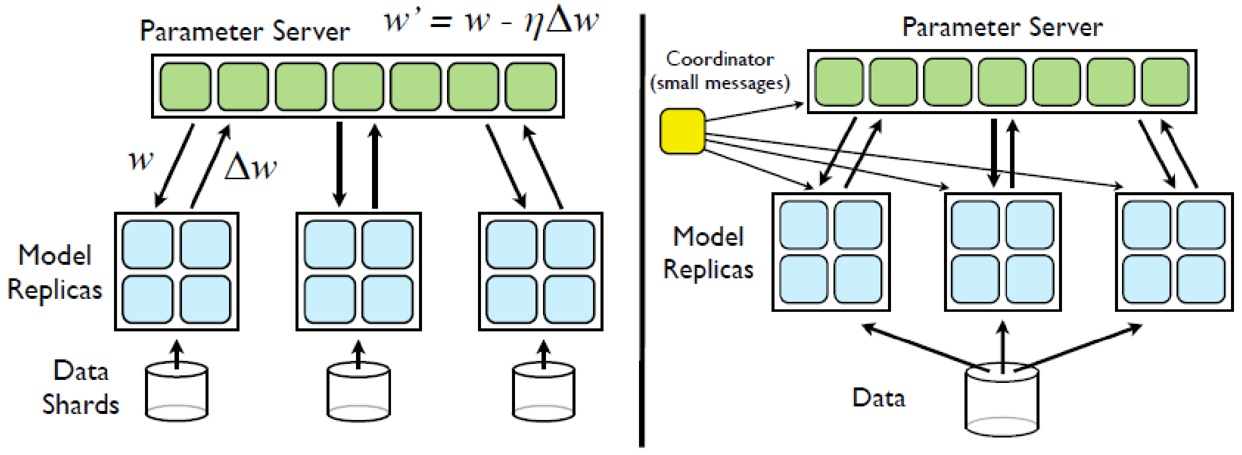
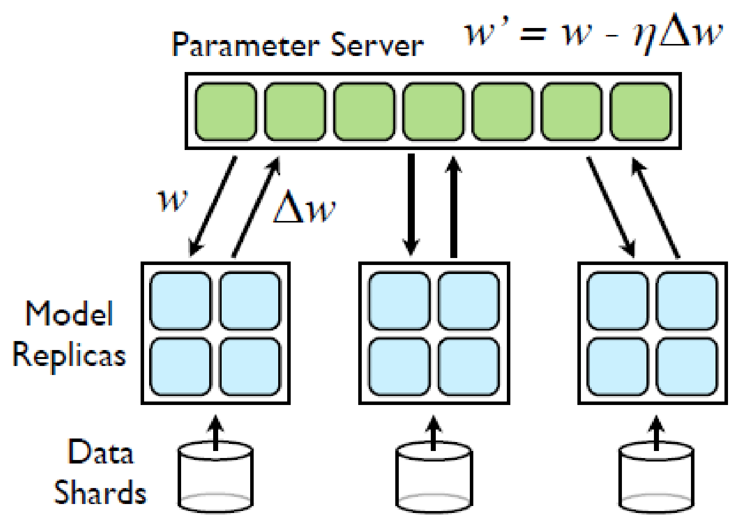
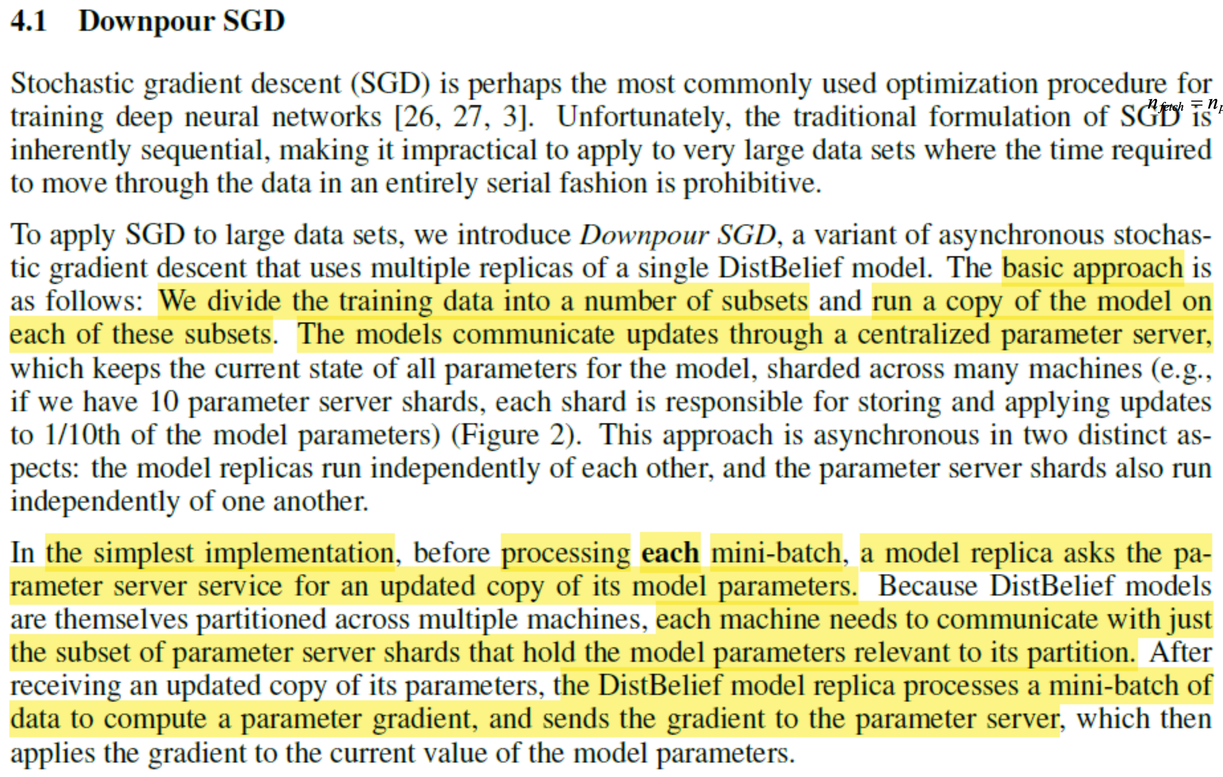


Figure 2: Left: Downpour SGD. Model replicas asynchronously fetch parameters w and push gradients Δw to the parameter server. Right: Sandblaster L-BFGS. A single 'coordinator' sends small messages to replicas and the parameter server to orchestrate batch optimization.

Data parallelism – Downpour SGD



- **Downpour SGD**
  - [Concepts] Divide the data (or mini-batch) to the multiple machines
  - Fetches and Pushes are equal (vs SGD)
    - $n_{fetch} = n_{push} = 1$

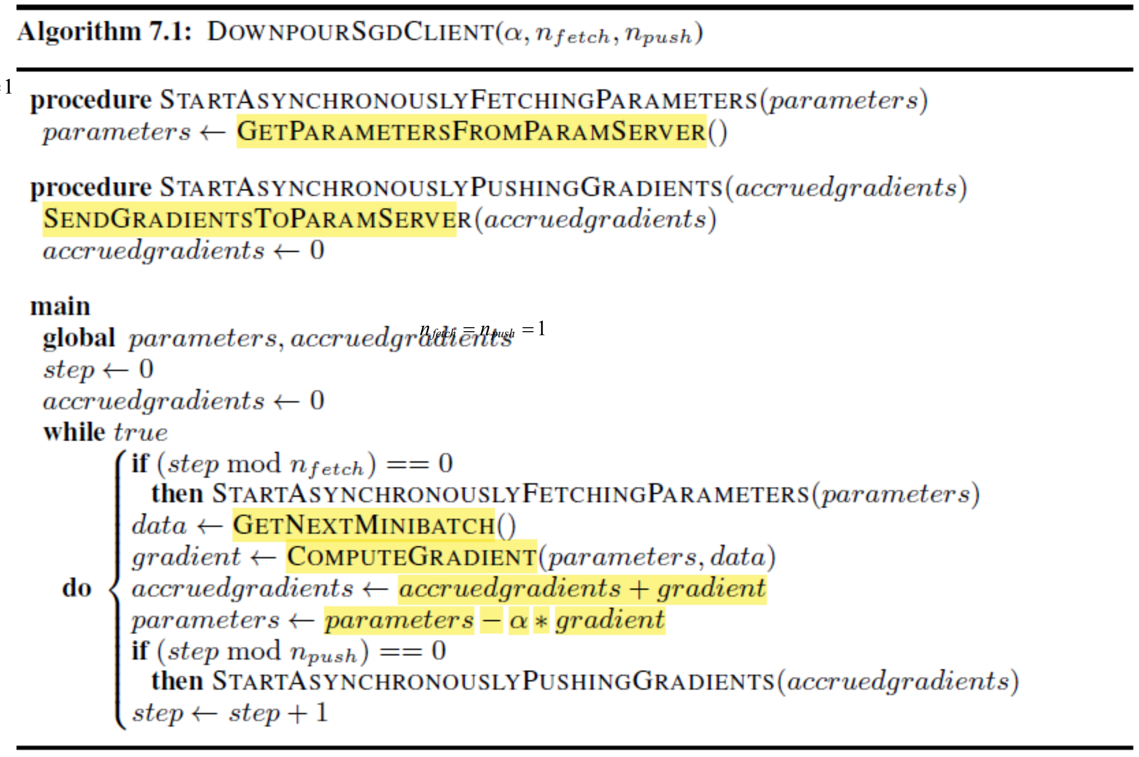


4.1 Downpour SGD

Stochastic gradient descent (SGD) is perhaps the most commonly used optimization procedure for training deep neural networks [26, 27, 3]. Unfortunately, the traditional formulation of SGD is $n_{fetch} = n_{push} = 1$ inherently sequential, making it impractical to apply to very large data sets where the time required to move through the data in an entirely serial fashion is prohibitive.

To apply SGD to large data sets, we introduce *Downpour SGD*, a variant of asynchronous stochastic gradient descent that uses multiple replicas of a single DistBelief model. The basic approach is as follows: We divide the training data into a number of subsets and run a copy of the model on each of these subsets. The models communicate updates through a centralized parameter server, which keeps the current state of all parameters for the model, sharded across many machines (e.g., if we have 10 parameter server shards, each shard is responsible for storing and applying updates to 1/10th of the model parameters) (Figure 2). This approach is asynchronous in two distinct aspects: the model replicas run independently of each other, and the parameter server shards also run independently of one another.

In the simplest implementation, before processing **each** mini-batch, a model replica asks the parameter server service for an updated copy of its model parameters. Because DistBelief models are themselves partitioned across multiple machines, each machine needs to communicate with just the subset of parameter server shards that hold the model parameters relevant to its partition. After receiving an updated copy of its parameters, the DistBelief model replica processes a mini-batch of data to compute a parameter gradient, and sends the gradient to the parameter server, which then applies the gradient to the current value of the model parameters.

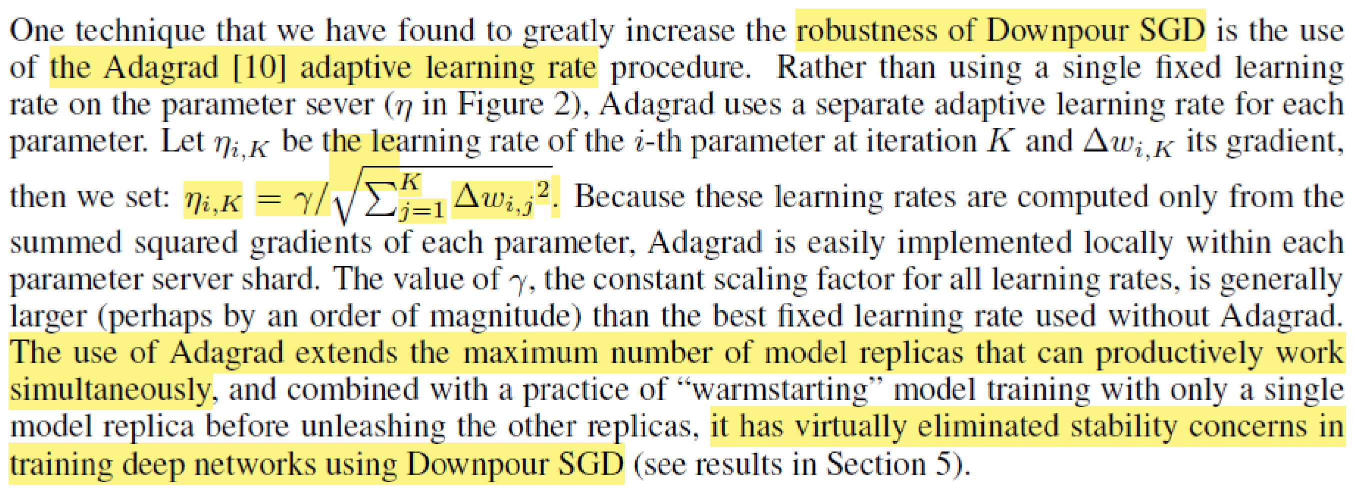


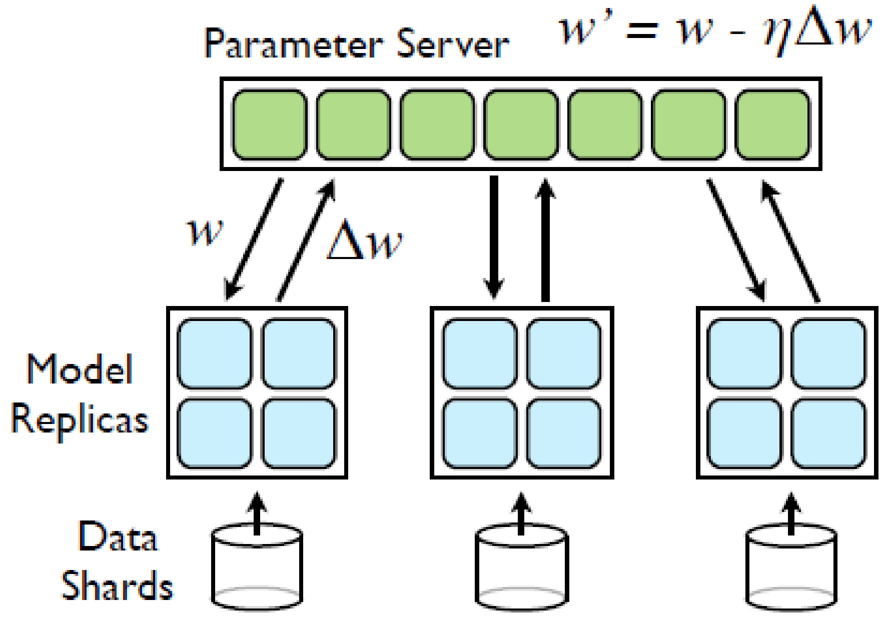
Algorithm 7.1: DOWNPOURSGDCLIENT($\alpha, n_{fetch}, n_{push}$)

```
procedure STARTASYNCHRONOUSLYFETCHINGPARAMETERS($parameters$)
  $parameters \leftarrow$ GETPARAMETERSFROMPARAMSERVER()

procedure STARTASYNCHRONOUSLYPUSHINGGRADIENTS($accruedgradients$)
  SENDGRADIENTSTOPARAMSERVER($accruedgradients$)
  $accruedgradients \leftarrow 0$

main
  global $parameters, accruedgradients$   $n_{fetch} = n_{push} = 1$
  $step \leftarrow 0$
  $accruedgradients \leftarrow 0$
  while $true$
    do
      if ($step$ mod $n_{fetch}$) == 0
        then STARTASYNCHRONOUSLYFETCHINGPARAMETERS($parameters$)
      $data \leftarrow$ GETNEXTMINIBATCH()
      $gradient \leftarrow$ COMPUTEGRADIENT($parameters, data$)
      $accruedgradients \leftarrow accruedgradients + gradient$
      $parameters \leftarrow parameters - \alpha * gradient$
      if ($step$ mod $n_{push}$) == 0
        then STARTASYNCHRONOUSLYPUSHINGGRADIENTS($accruedgradients$)
      $step \leftarrow step + 1$
```

Data parallelism – Downpour SGD

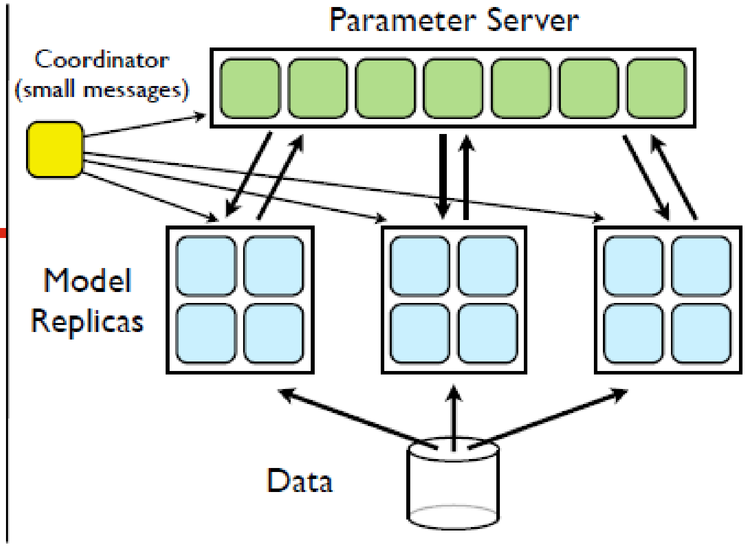
- **Downpour SGD – w. Adagrad**
  - With Adagrad adaptive learning rate
    - Increasing Robustness
    - Adaptive learning rate for each separate parameter can increase the stability of the model training.
      - Increase the maximum number of machines
      - Eliminates stability concerns


One technique that we have found to greatly increase the robustness of Downpour SGD is the use of the Adagrad [10] adaptive learning rate procedure. Rather than using a single fixed learning rate on the parameter sever (η in Figure 2), Adagrad uses a separate adaptive learning rate for each parameter. Let $\eta_{i,K}$ be the learning rate of the i-th parameter at iteration K and $\Delta w_{i,K}$ its gradient, then we set: $\eta_{i,K} = \gamma / \sqrt{\sum_{j=1}^{K} \Delta w_{i,j}{}^2}$. Because these learning rates are computed only from the summed squared gradients of each parameter, Adagrad is easily implemented locally within each parameter server shard. The value of γ, the constant scaling factor for all learning rates, is generally larger (perhaps by an order of magnitude) than the best fixed learning rate used without Adagrad. The use of Adagrad extends the maximum number of model replicas that can productively work simultaneously, and combined with a practice of "warmstarting" model training with only a single model replica before unleashing the other replicas, it has virtually eliminated stability concerns in training deep networks using Downpour SGD (see results in Section 5).



Data parallelism – Sandblaster L-BGFS

- **Sandblaster optimization based on L-BGFS**



A key idea in *Sandblaster* is distributed parameter storage and manipulation. The core of the optimization algorithm (e.g L-BFGS) resides in a coordinator process (Figure 2), which does not have direct access to the model parameters. Instead, the coordinator issues commands drawn from a small set of operations (e.g., dot product, scaling, coefficient-wise addition, multiplication) that can be performed by each parameter server shard independently, with the results being stored locally on the same shard. Additional information, e.g the history cache for L-BFGS, is also stored on the parameter server shard on which it was computed. This allows running large models (billions of parameters) without incurring the overhead of sending all the parameters and gradients to a single central server. (See the Appendix for pseudocode.)

In typical parallelized implementations of L-BFGS, data is distributed to many machines and each machine is responsible for computing the gradient on a specific subset of data examples. The gradients are sent back to a central server (or aggregated via a tree [16]). Many such methods wait for the slowest machine, and therefore do not scale well to large shared clusters. To account for this problem, we employ the following load balancing scheme: The coordinator assigns each of the N model replicas a small portion of work, much smaller than 1/Nth of the total size of a batch, and assigns replicas new portions whenever they are free. With this approach, faster model replicas do more work than slower replicas. To further manage slow model replicas at the end of a batch, the coordinator schedules multiple copies of the outstanding portions and uses the result from whichever model replica finishes first. This scheme is similar to the use of "backup tasks" in the MapReduce framework [24]. Prefetching of data, along with supporting data affinity by assigning sequential

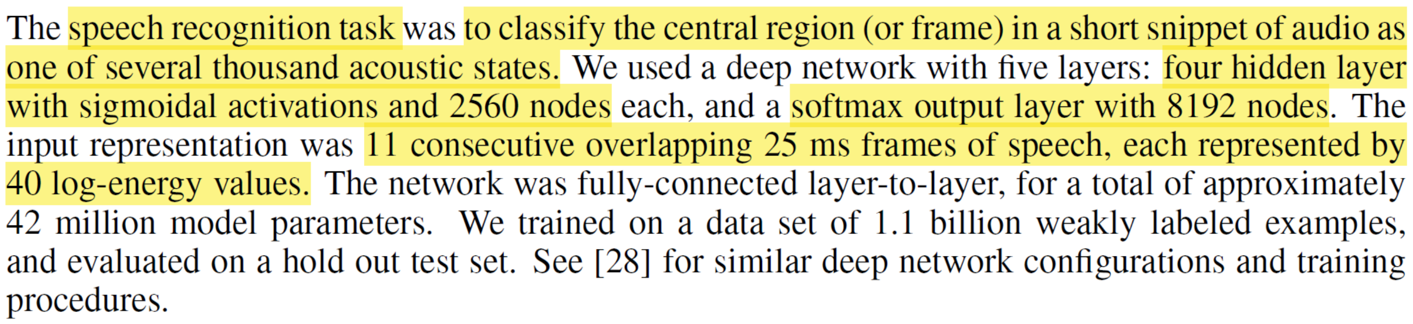
portions of data to the same worker makes data access a non-issue. In contrast with Downpour SGD, which requires relatively high frequency, high bandwidth parameter synchronization with the parameter server, Sandblaster workers only fetch parameters at the beginning of each batch (when they have been updated by the coordinator), and only send the gradients every few completed portions (to protect against replica failures and restarts).

Algorithm 7.2: SANDBLASTERLBFGS()

```
procedure REPLICA.PROCESSPORTION($portion$)
  if ($!hasParametersForStep$)
    then $parameters \leftarrow$ GETPARAMETERSFROMPARAMSERVER()
  $data \leftarrow$ GETDATAPORTION($portion$)
  $gradient \leftarrow$ COMPUTEGRADIENT($parameters, data$)
  $localAccruedGradients \leftarrow localAccruedGradients + gradient$

procedure PARAMETERSERVER.PERFORMOPERATION($operation$)
  $PerformOperation$

main
  $step \leftarrow 0$
  while $true$
    do
      comment: PS: ParameterServer
      $PS.accruedgradients \leftarrow 0$
      while ($batchProcessed < batchSize$)
        do
          for all ($modelReplicas$) comment: Loop is parallel and asynchronous
            if ($modelReplicaAvailable$)
              then
                REPLICA.PROCESSPORTION($modelReplica$)
                $batchProcessed \leftarrow batchProcessed + portion$
            if ($modelReplicaWorkDone$ and $timeToSendGradients$)
              then
                SENDGRADIENTS($modelReplica$)
                $PS.accruedGradients \leftarrow PS.accruedGradients + gradient$
      COMPUTELBFGSDIRECTION($PS.Gradients, PS.History, PS.Direction$)
      LINESEARCH($PS.Parameters, PS.Direction$)
      PS.UPDATEPARAMETERS($PS.parameters, PS.accruedGradients$)
      $step \leftarrow step + 1$
```

Experiments – Models

- **Experimental models – Speech Recognition**
  - Speech Recognition Task
    - Time series audio signals
    - Central region: Find a region where meaningful audio exist
      - What kind of region contains the meaning?



The speech recognition task was to classify the central region (or frame) in a short snippet of audio as one of several thousand acoustic states. We used a deep network with five layers: four hidden layer with sigmoidal activations and 2560 nodes each, and a softmax output layer with 8192 nodes. The input representation was 11 consecutive overlapping 25 ms frames of speech, each represented by 40 log-energy values. The network was fully-connected layer-to-layer, for a total of approximately 42 million model parameters. We trained on a data set of 1.1 billion weakly labeled examples, and evaluated on a hold out test set. See [28] for similar deep network configurations and training procedures.

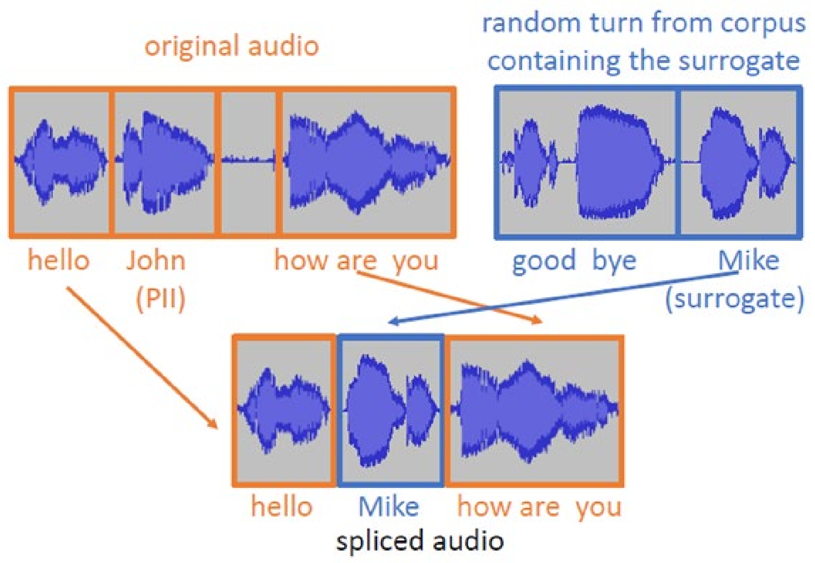


Figure 3: *Audio generation by splicing original speech and snippets from the corpus.*

Results – Model Parallelism

- **Model parallelism benchmarks**
  - Metric: Tr. time for single machine / Tr. Time for Multiple machines.
    - Moderately sized model (speech)
      - 2.2x faster @ 8 machines
      - Network overhead, less work per machine issues
    - Larger model (speech)
      - 12x faster @ 81 machines
      - Continues increasing as more model implemented

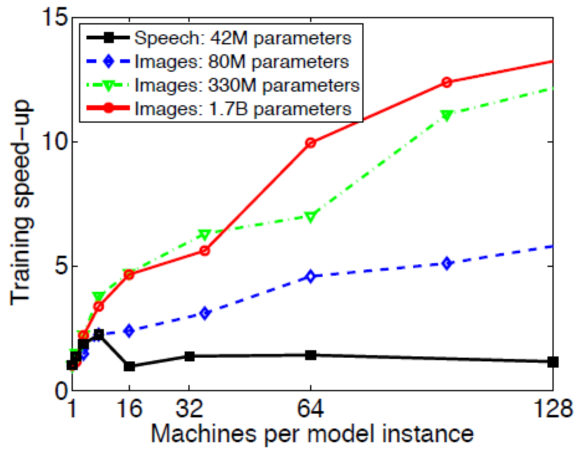


Figure 3: Training speed-up for four different deep networks as a function of machines allocated to a single DistBelief model instance. Models with more parameters benefit more from the use of additional machines than do models with fewer parameters.

Results – Data Parallelism (1/2)

- **Optimization method comparison (Speech model)**
  - [Goal] Obtain the maximum test set acc. In the minimum tr. Time, regardless of resource req.
    - Sandblaster L-BFGS, Downpour SGD w. Adagrad can faster than GPU based processing.

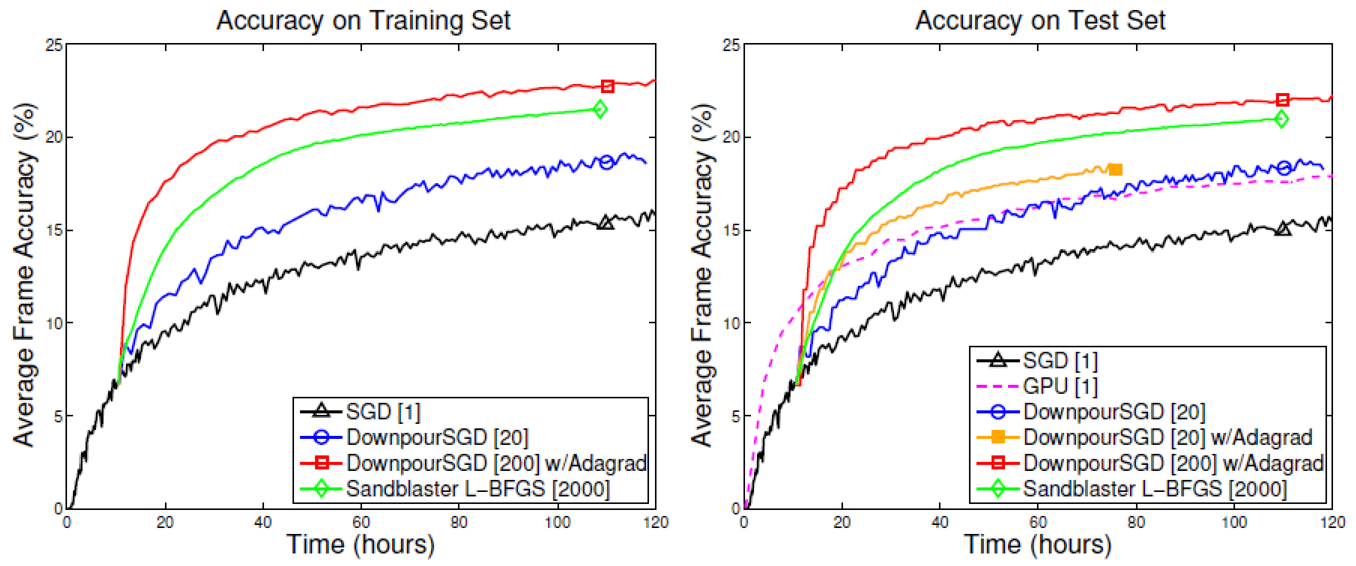
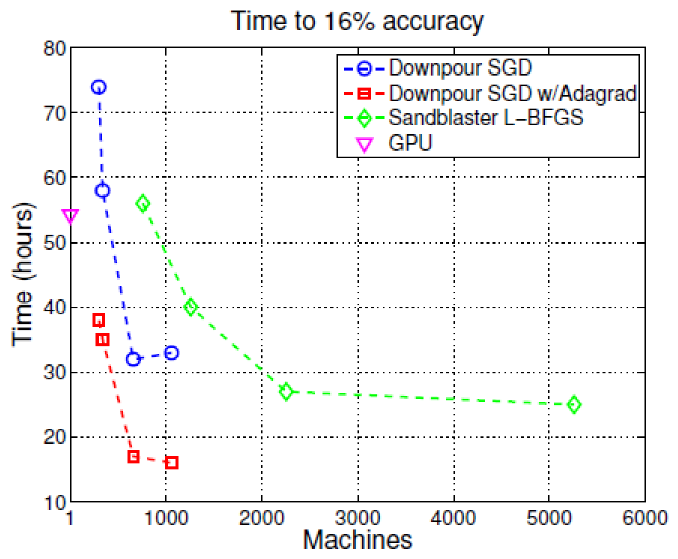


Figure 4: Left: Training accuracy (on a portion of the training set) for different optimization methods. Right: Classification accuracy on the hold out test set as a function of training time. Downpour and Sandblaster experiments initialized using the same ∼10 hour warmstart of simple SGD.

Results – Data Parallelism (2/2)

- **Optimization method comparison (Speech model)**
  - For comparison to a fixed resource budget. (Trade-off : Resource vs Performance) @ 16% test acc.
    - Downpour SGD w. Adagrad
      - Takes less time and Fewer resources
    - Sandblaster L-BFGS
      - Promising the fastest training time if the computing resources are enough.



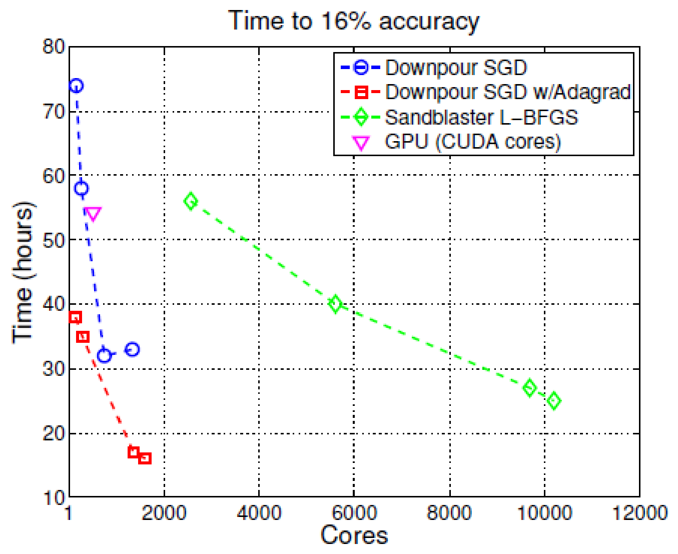
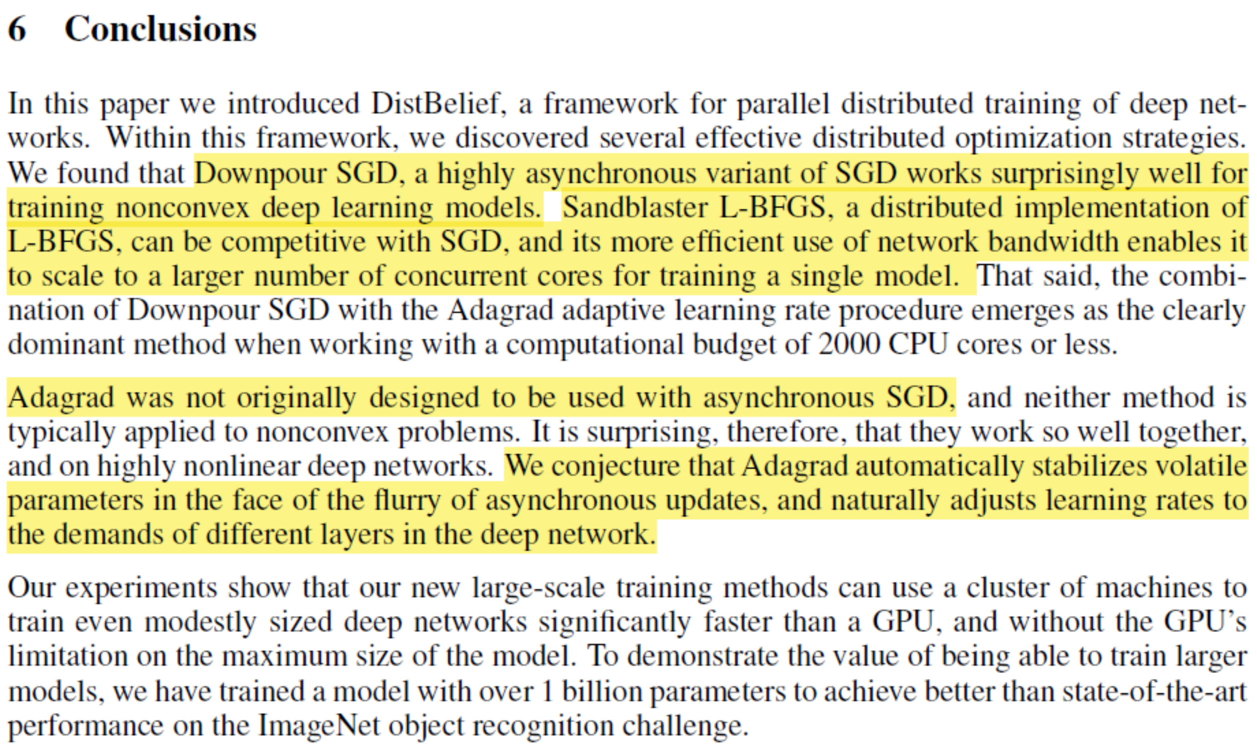


Figure 5: Time to reach a fixed accuracy (16%) for different optimization strategies as a function of number of the machines (left) and cores (right).

Conclusion



6 Conclusions

In this paper we introduced DistBelief, a framework for parallel distributed training of deep networks. Within this framework, we discovered several effective distributed optimization strategies. We found that Downpour SGD, a highly asynchronous variant of SGD works surprisingly well for training nonconvex deep learning models. Sandblaster L-BFGS, a distributed implementation of L-BFGS, can be competitive with SGD, and its more efficient use of network bandwidth enables it to scale to a larger number of concurrent cores for training a single model. That said, the combination of Downpour SGD with the Adagrad adaptive learning rate procedure emerges as the clearly dominant method when working with a computational budget of 2000 CPU cores or less.

Adagrad was not originally designed to be used with asynchronous SGD, and neither method is typically applied to nonconvex problems. It is surprising, therefore, that they work so well together, and on highly nonlinear deep networks. We conjecture that Adagrad automatically stabilizes volatile parameters in the face of the flurry of asynchronous updates, and naturally adjusts learning rates to the demands of different layers in the deep network.

Our experiments show that our new large-scale training methods can use a cluster of machines to train even modestly sized deep networks significantly faster than a GPU, and without the GPU's limitation on the maximum size of the model. To demonstrate the value of being able to train larger models, we have trained a model with over 1 billion parameters to achieve better than state-of-the-art performance on the ImageNet object recognition challenge.

- **We consider the oration paper for starting the research of decentralized AI**
  - Data / Model parallelism
  - The most important thing is
    - How efficiently construct the parallelized model architecture?
    - If possible, then it can override the performance of single shallow networks.

- **Q) (In Decentralized RFF) Distributed models are different. – Multi-agent cases?**
  - Future works – Try to implement the model parallelism for a decentralized RFF system.