CERN

Distributed Deep Learning
with Apache Spark and Keras

Joeri Hermans (Technical Student)
Departement of Knowledge Engineering
Maastricht University

IT-DB-SAS
CERN

Distributed Deep Learning

Problem: How do we reduce the training time of our (large) models, while training them on very large datasets? (like use-cases in CMS and ATLAS)

- Jeff Dean et al. (Google) proposes 2 different paradigms:
  - Model parallelism
  - Data parallelism

Our focus: Data parallelism

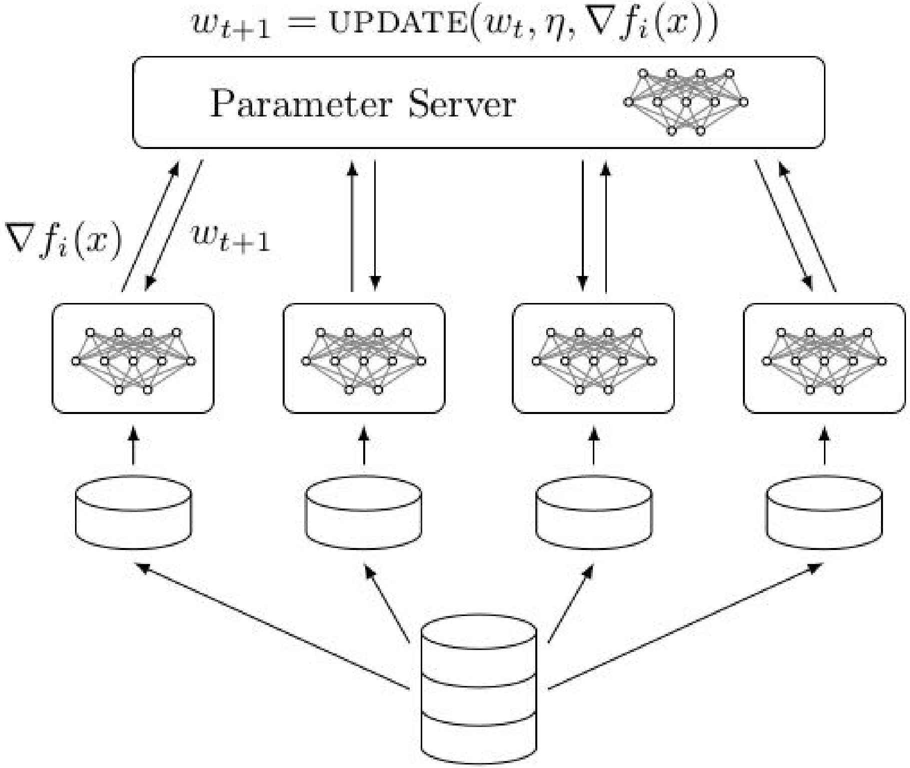
Data Parallelism

- ***n*** compute nodes (or processes)
- Data is split into ***n*** data shards.
- Model is copied to compute nodes.
- **Objective**: optimize center model.

Ideally: time is reduced by factor ***n***

However:

- Communication constraints
- Computational overhead

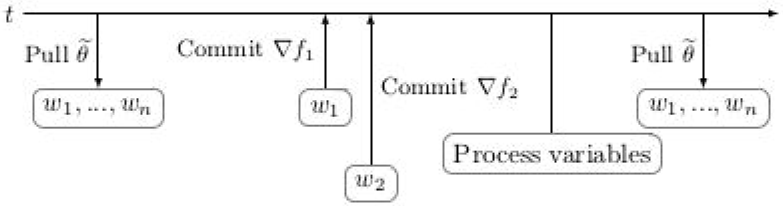


Approaches and techniques

- How to optimize the center model (or *center variable*) using data parallelism?
  - **Synchronous Data Parallelism**
    - Model Averaging
    - Elastic Averaging SGD (Zhang et al.)
  - **Asynchronous Data Parallelism**
    - Asynchronous Elastic Averaging SGD (Zhang et al.)
    - DOWNPOUR (Dean et al.)
    - ADAG

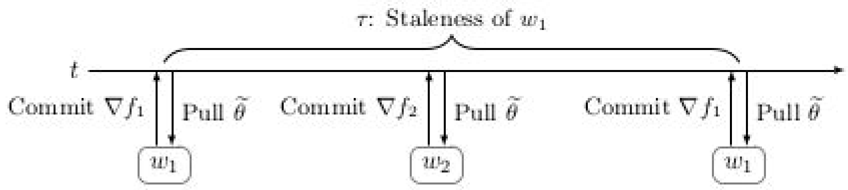
Methods are available in our framework.

Synchronous Data Parallelism



Problem: As fast as the slowest compute node due to blocking.

Asynchronous Data Parallelism



- Solves the blocking issue of synchronous data parallelism.
- **Problems:**
    - Gradient updates can be based on older values of the center variable (staleness)
    - Introduces a simple queuing model of gradient updates (*implicit momentum*, see next slide)

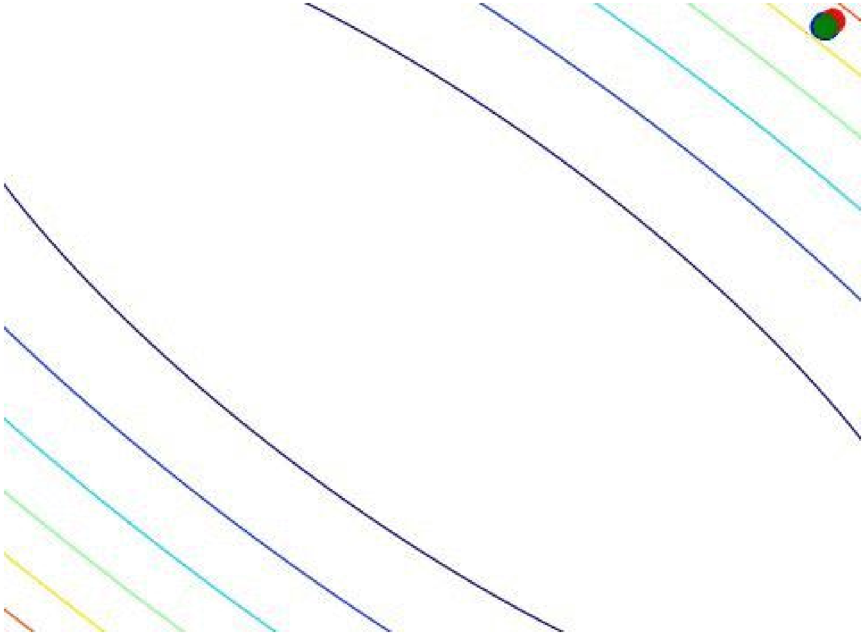
Note: basically the definition of the DOWNPOUR optimization scheme introduced by Dean et al.

Asynchrony induces momentum

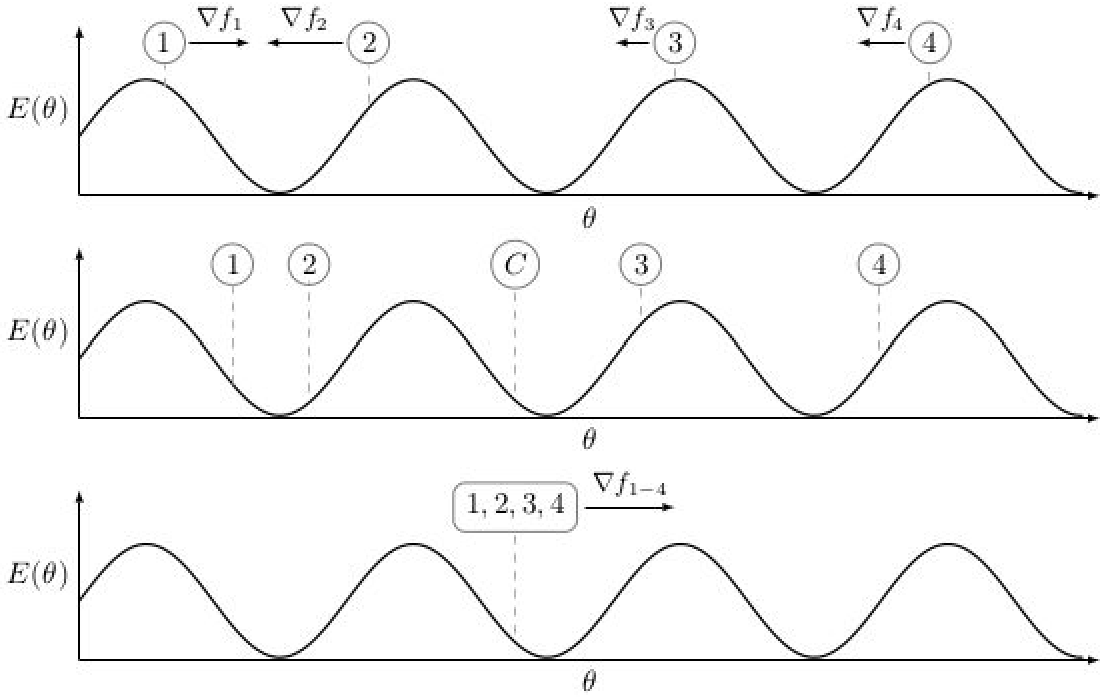
- Or rather, something that behaves like momentum.
- Too many workers causes decay in performance or even *divergence*! (unless optimizer is able to handle this)

Simulation of DOWNPOUR (right)

Green Regular Gradient Descent
Blue Parallel worker
Red Center variable



Model Averaging (inherently synchronous)

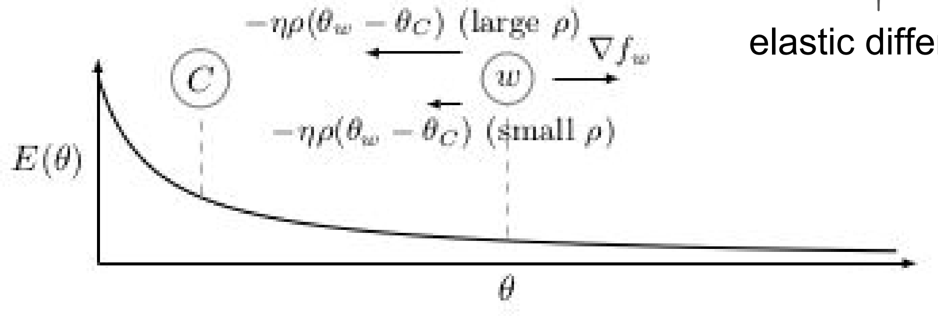


Note: gradients are pointing in the opposite direction to make the figure more intuitive.

Elastic Averaging SGD

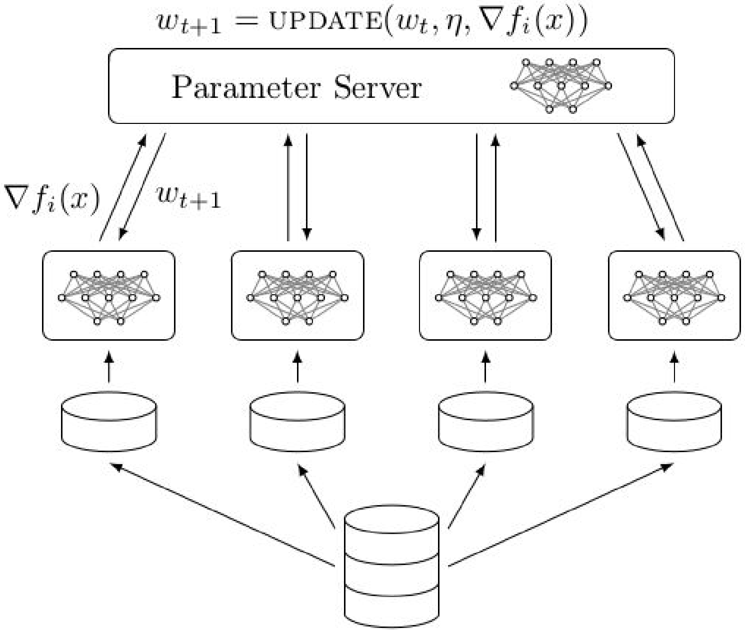
- What to do under communication constraints (e.g., heavily used networks)?
    - Let workers do more iterations before communicating with the PS (exploration).
    - Too much exploration, workers do not “agree on neighbourhood” anymore.
    - **Answer:** “elasticity”.

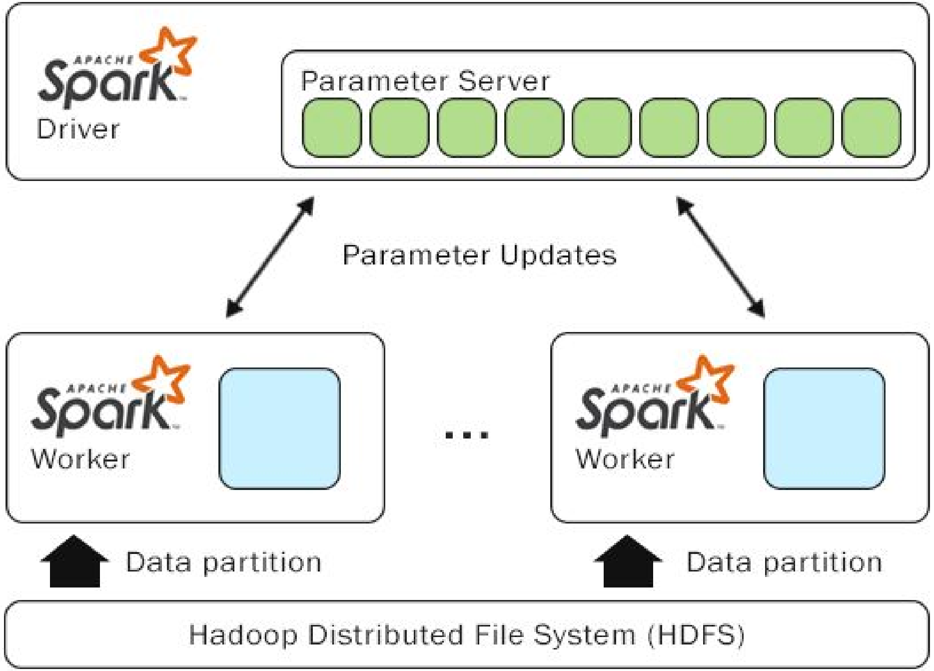
$$\theta_{t+1}^{i} = \theta_t^i - \eta\nabla f(\theta_t^i) \underbrace{- \eta\rho(\theta_t^i - \theta_t^c)}_{\text{elastic difference}}$$



However, EASGD requires some fine-tuning (rho). And has difficulties converging when communication window is small (why?).
But scales very well (almost ideally)!

dist-keras: architecture



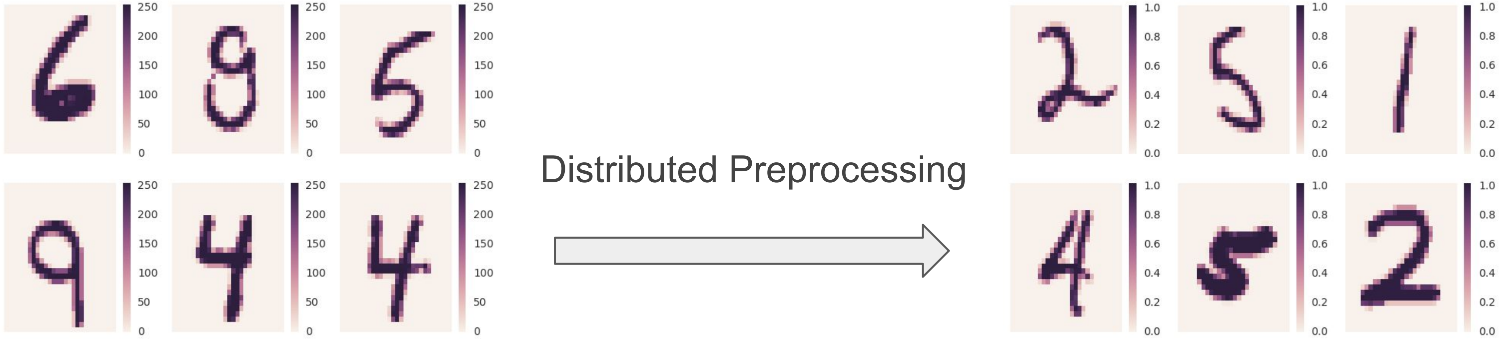


Why Apache Spark?

- We use Apache Spark mainly as a distribution mechanism for the training.
- Strong data preprocessing framework and libraries.
- Bigger than memory datasets.
- Large community and active development.
- CERN Hadoop Service has several clusters available.
- Integration with Spark Streaming to do on-line predictions.

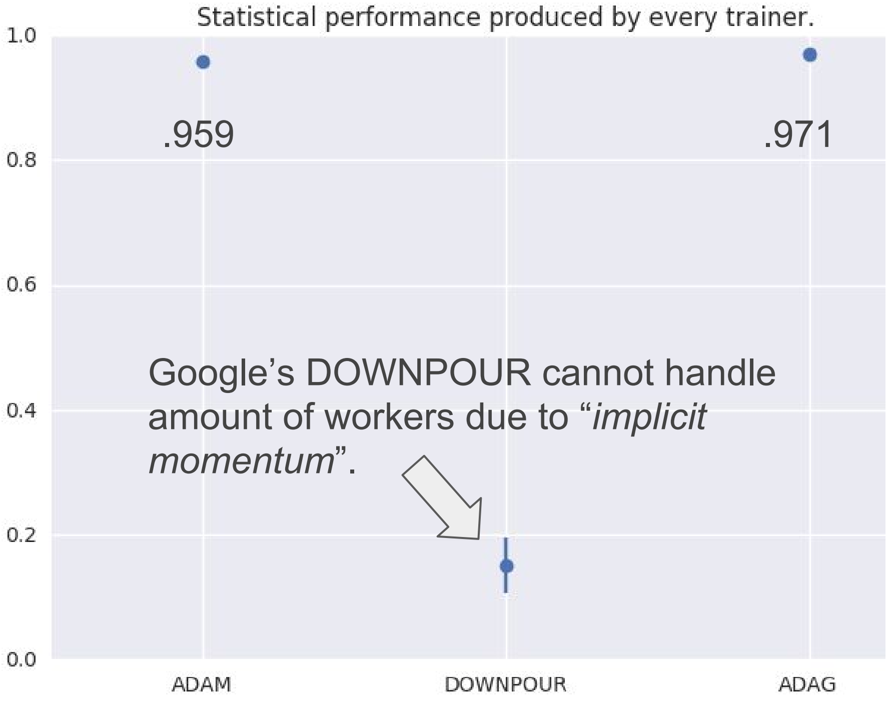
Experiments

- 2 networks: a multilayer perceptron and convolutional network.
- Both have ~1 000 000 trainable parameters (~32 MB per model).
- 4 sample mini-batches, 1 epoch.
- **Dataset**: MNIST.
- **Optimizers**: Adam (sequential), EASGD, DOWNPOUR, ADAG (distributed)
- 20 parallel workers:
  - 10 compute nodes with 10 Gbps network cards
  - 2 processes per node (32 cores per node)



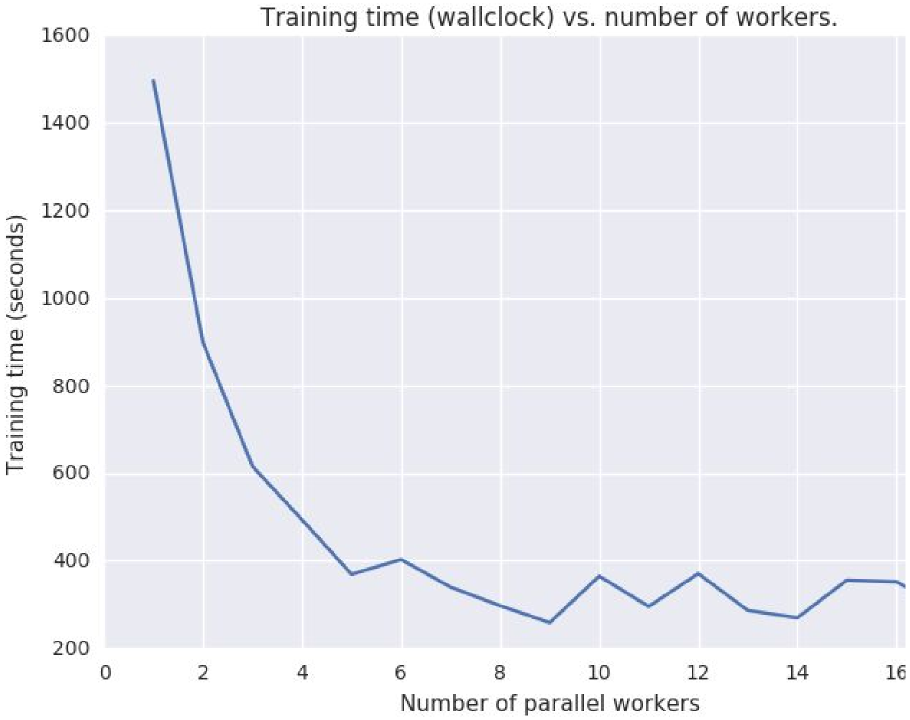
Experiments (1)

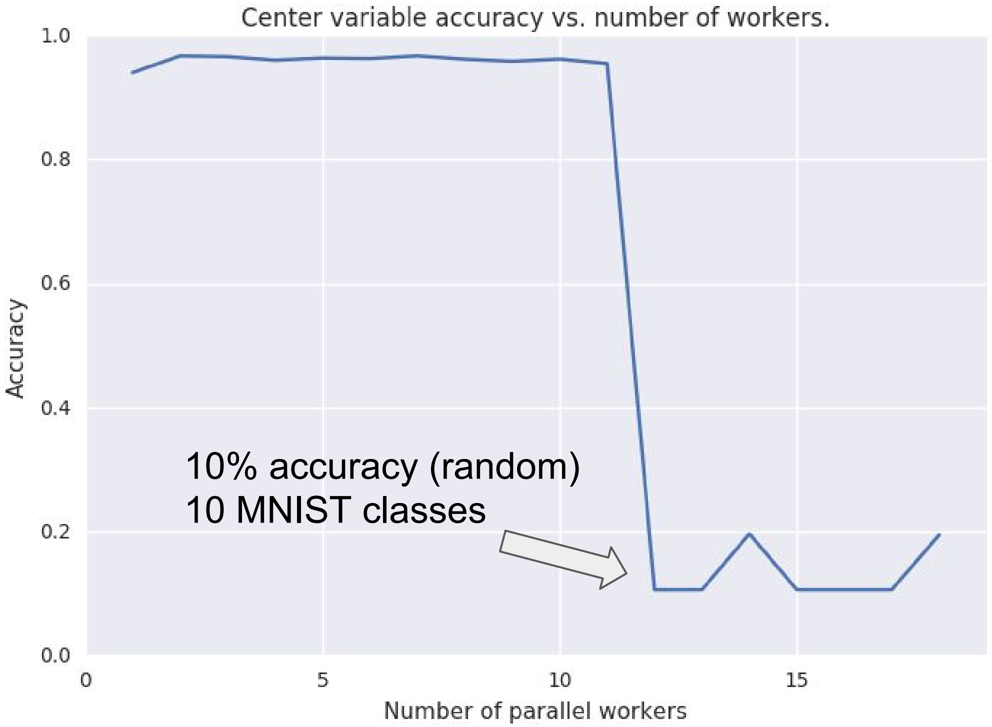
- 30 experiments for every optimization scheme (multilayer perceptron).



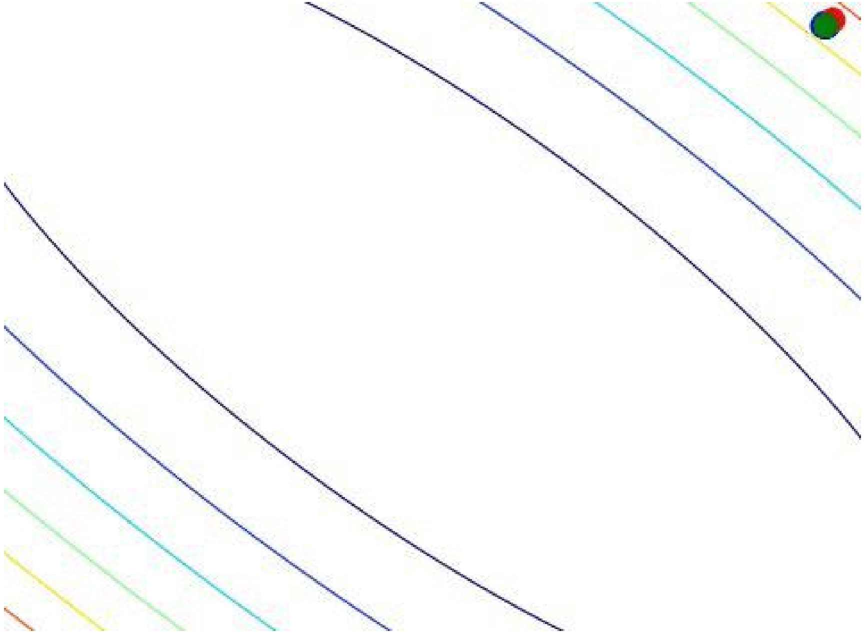
Experiments (2)

Optimization algorithm: DOWNPOUR

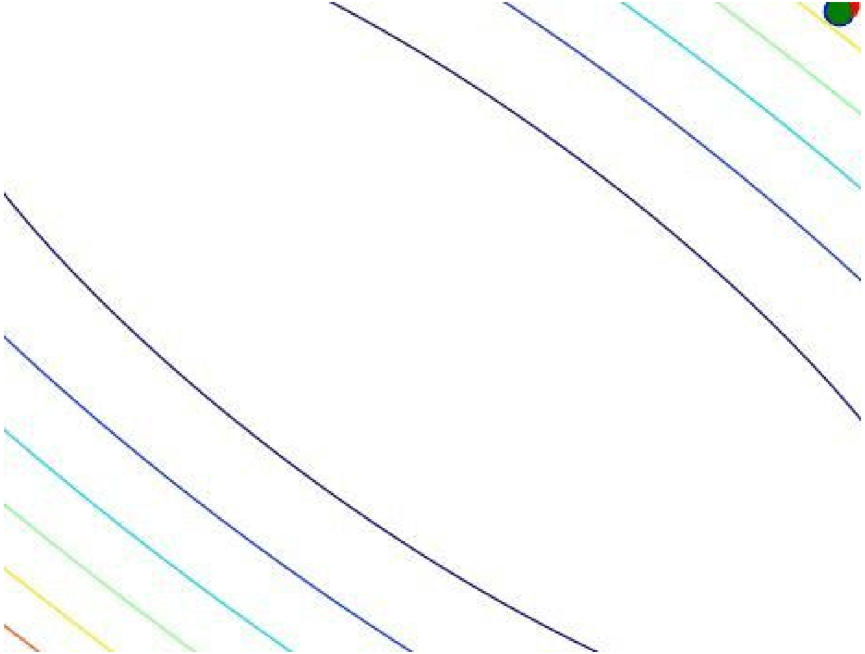




Divergence due to the number of parallel workers



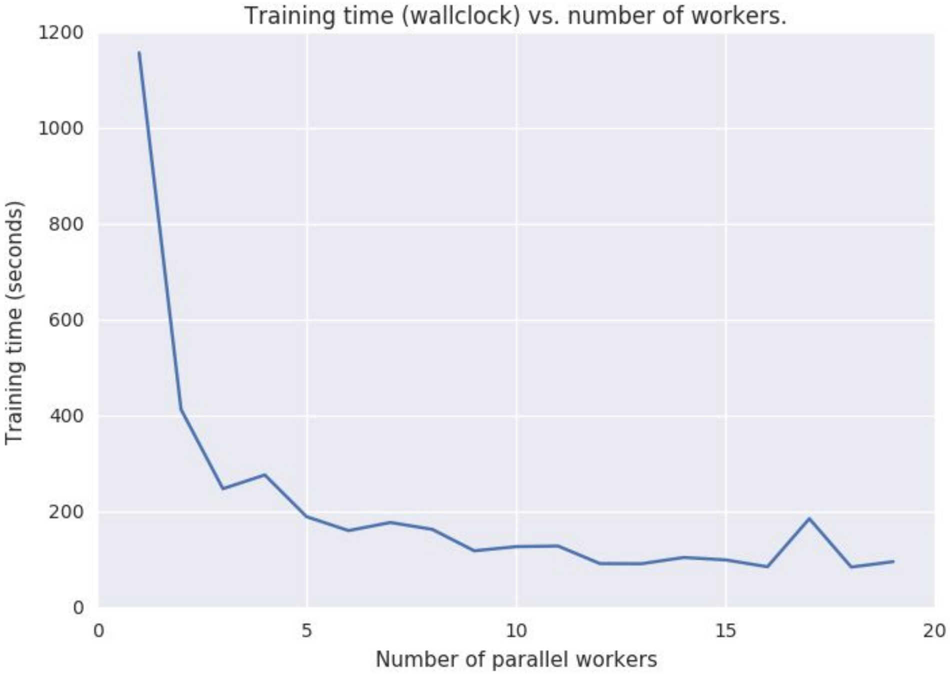
20 parallel workers (convergence)

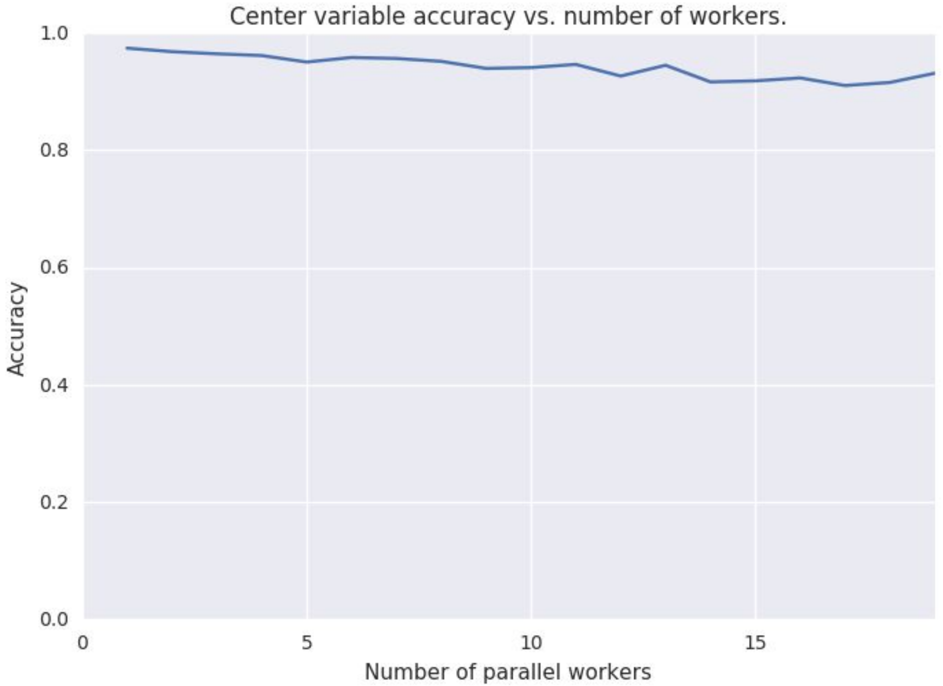


40 parallel workers (divergence)

Experiments (3)

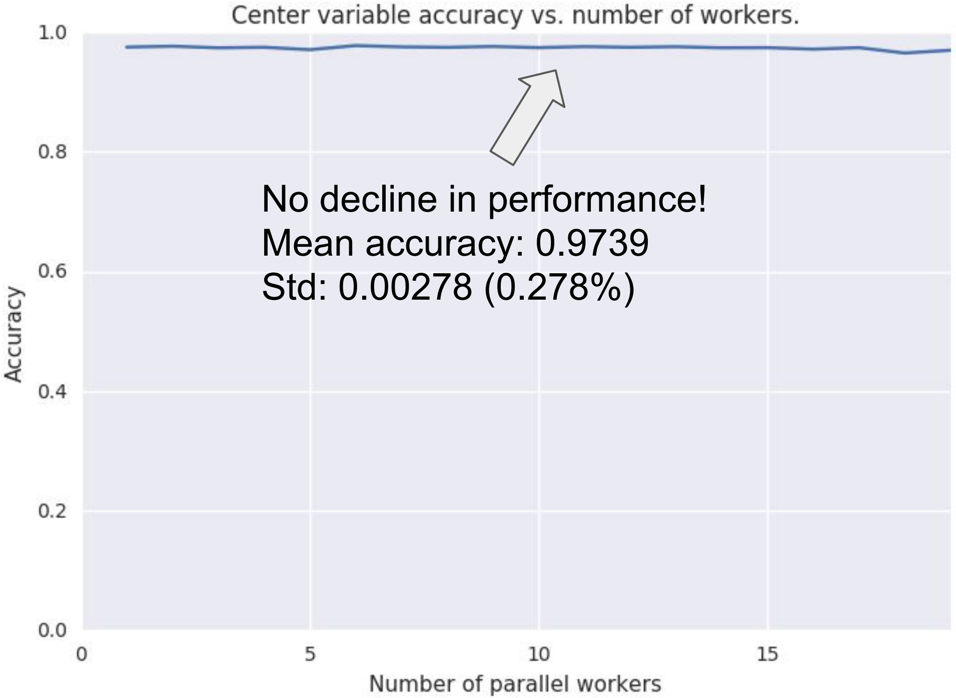
Optimization algorithm: Asynchronous EASGD (rho = 5.0)





Experiments (4)

Optimization algorithm: ADAG



Problems we encountered

- Convolutional layers expect matrices to be in a specific format (reshape).

```
reshape_transformer = ReshapeTransformer("features_normalized", "matrix", (28, 28, 1))
dataset = reshape_transformer.transform(dataset)
```

- Adding a column to a distributed DataFrame based on other columns proved be non-trivial to do efficiently.

```
def new_dataframe_row(old_row, column_name, column_value):
    """Constructs a new Spark Row based on the old row, and a new column name and value."""
     row = Row(*(old_row.__fields__ + [column_name]))(*(old_row + (column_value, )))
    return row
```

- **Strugglers**. Some workers are idle because they completed their data shard way faster.
  - *Parallelism factor*: a data-shard is segmented in “tasks” w.r.t. this factor. If a w
    then it will take tasks from other workers in order to get the job done faster.

Future Work

- Further theoretical understanding.
- Steps have been / will be made to build an optimizer (ADAG).
  - Combine EASGD like communication windows (to ensure scaling).
  - Staleness compensation.
  - ...
- In-depth performance tests (including CIFAR-10(0)).
- Some work needs to be done to improve throughput of parameter server.
  - PS doesn’t scale that well when using models with a -very- high number of parameters.
  - Initially, weight sharing was done using a REST API, now custom protocol.
  - Random communication windows to lower “spiking” load of PS?

Questions?

https://github.com/cerndb/dist-keras

https://github.com/cerndb/dist-keras/blob/master/examples/mnist.ipynb

Appendices

Code example

```
trainer = DOWNPOUR(keras_model=convnet, worker_optimizer=optimizer_convnet, loss=loss_convnet,
                   num_workers=num_workers, batch_size=8, communication_window=5, learning_rate=0.1,
                   num_epoch=1, features_col="matrix", label_col="label_encoded")

trainer.set_parallelism_factor(1) # default value (more on this later)

trained_model = trainer.train(training_set)

print("Training time: " + str(trainer.get_training_time()))
print("Accuracy: " + str(evaluate_accuracy(trained_model, test_set, "matrix")))

def evaluate_accuracy(model, test_set, features="features_normalized_dense"):
    evaluator = AccuracyEvaluator(prediction_col="prediction_index", label_col="label")
    predictor = ModelPredictor(keras_model=model, features_col=features)
    transformer = LabelIndexTransformer(output_dim=nb_classes)
    test_set = test_set.select(features, "label")
    test_set = predictor.predict(test_set)
    test_set = transformer.transform(test_set)
    score = evaluator.evaluate(test_set)

    return score
```

ADAG (research idea)

- Our 10 Gbps network allows for fast parameter transfers.
- As a result, no communication constraints (assumption).
- In order to reduce communication overhead ever further:
  - Random communication windows in specific range. E.g., [2-6]
- This reduces computational overhead introduced by EASGD.
- Instead of averaging the gradients, divide the gradient residual by the communication window. -> Empirically proved to be better than DOWNPOUR