CERN

Distributed Deep Learning
with Apache Spark and Keras

Joeri Hermans (Technical Student)
Maastricht University

Introduction

- CMS is exploring the possibility for a deep learning model in the HLT.
- Use-case requires a lot of training data (~ 1 TB).
- Application of **distributed optimization algorithms**.

- **Can be extended to more general use-cases!**

Distributed Deep Learning?

- Training with millions of parameters takes a lot of time and expensive operations like convolutions make the task even more computationally expensive.
  - How do we **decrease** the **training time?**

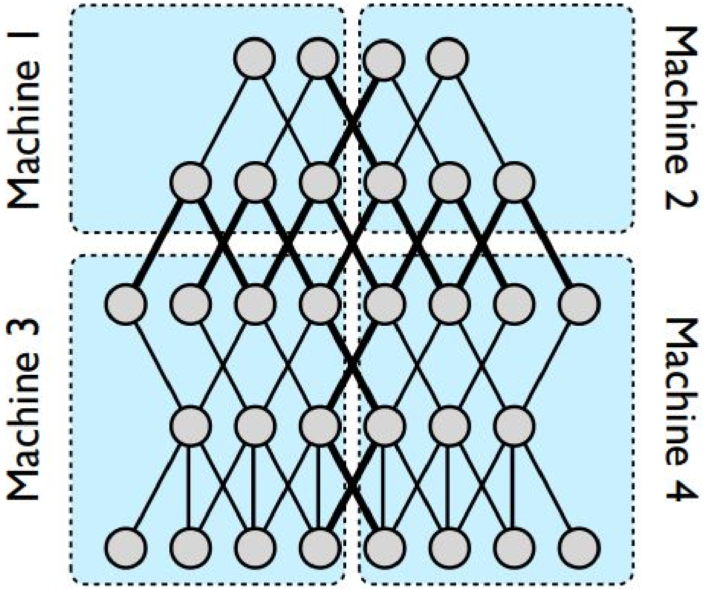
Distributed Deep Learning

- 2 main paradigms introduced by Dean et al. (Google)
    - **Model parallelism**
    - **Data parallelism**

Note: hybrids of these are possible.

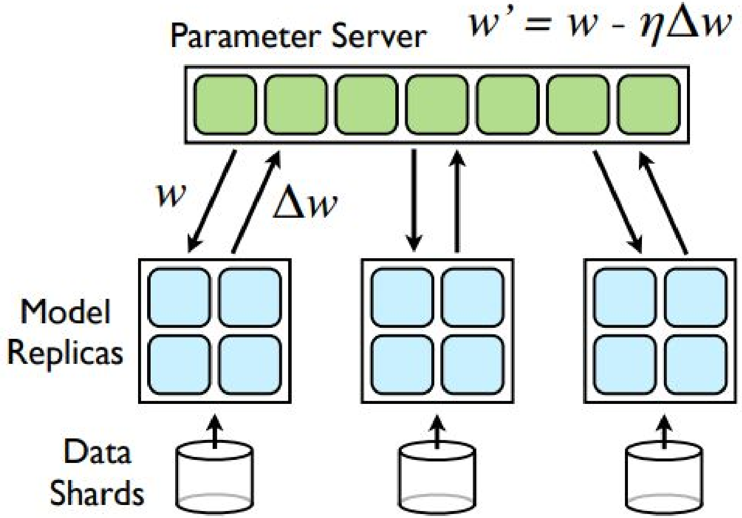
Model Parallelism

- Big network spread over multiple machines.
- Usually due to memory requirements.
- Not only for training, faster propagation.
- Sharing of updates.



Data Parallelism

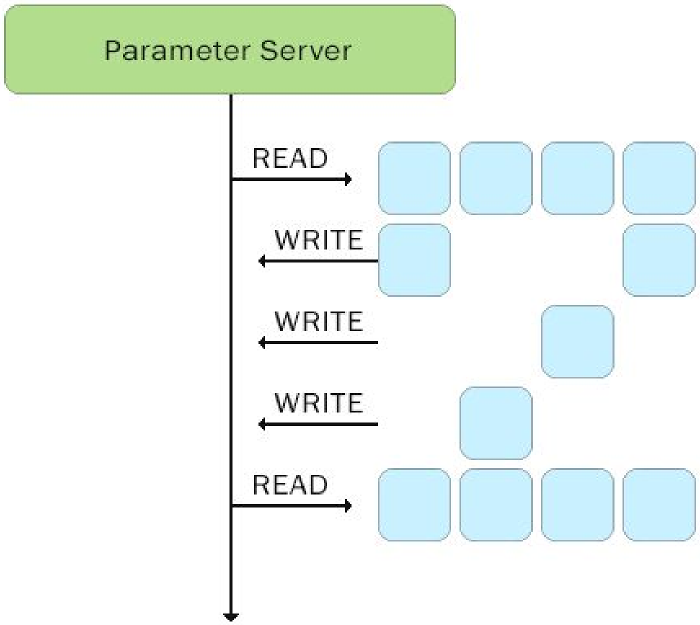
- Model is replicated over different machines.
- Every worker gets own shard (partition) of the dataset.
- Every update is reported to **PS**.
- **PS:** Parameter Server
- **We will be focussing on this!**



- Two approaches:
    - **Synchronous data parallelism**
    - **Asynchronous data parallelism**

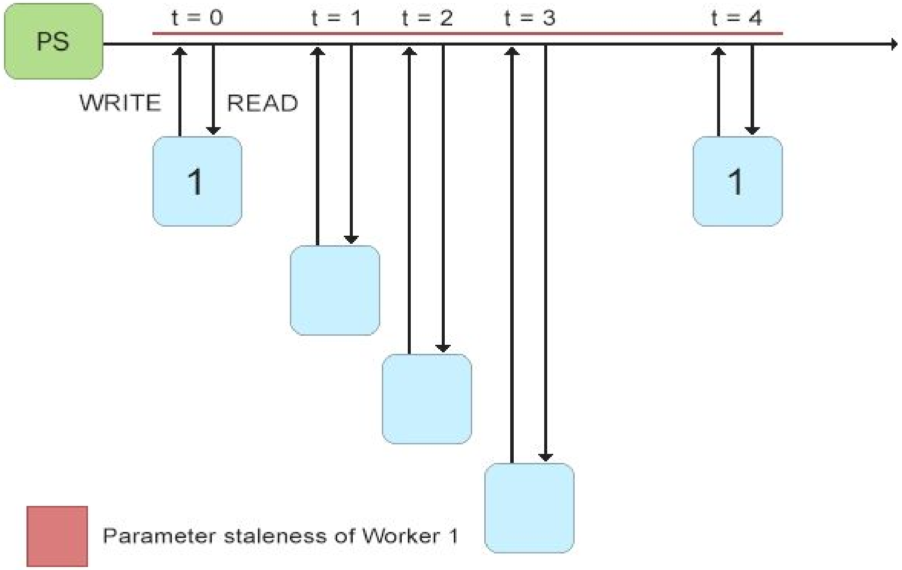
Synchronous Data Parallelism

- All workers synchronize on the update variable.
- Synchronization mechanism required.
  - What in the case of a **slow** node?
  - But updates will be consistent!

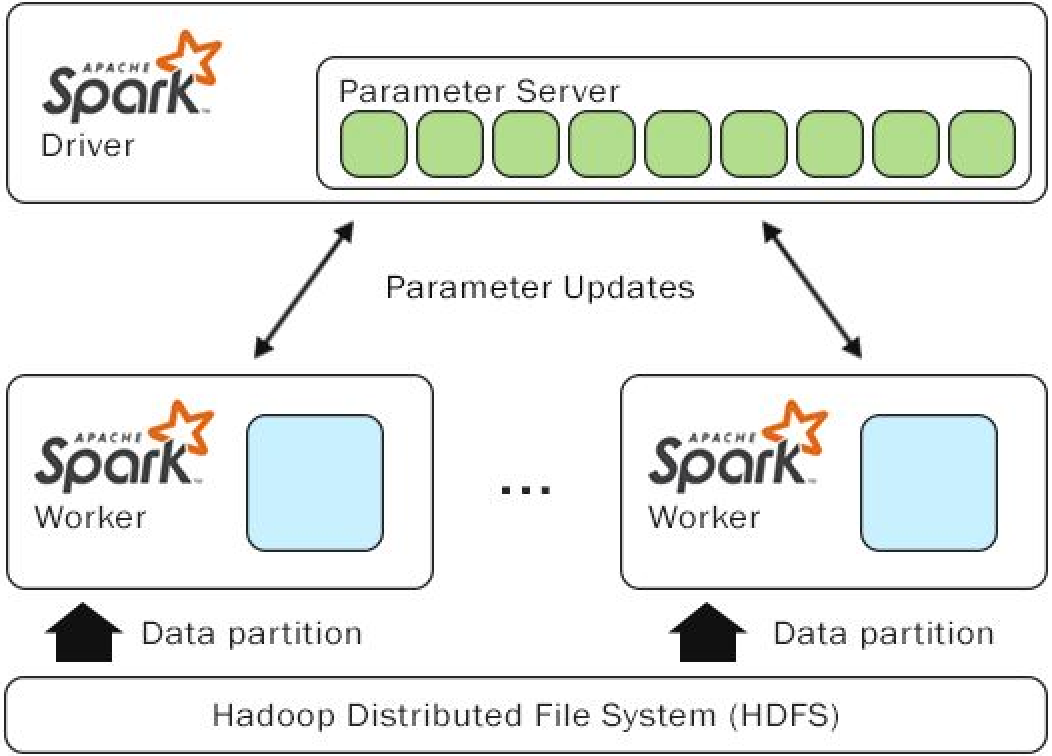


Asynchronous Data Parallelism

- All workers read and write whenever they want.
- **Problem:** stale gradient updates.
- **But:** sparse updates!



Distributed Keras



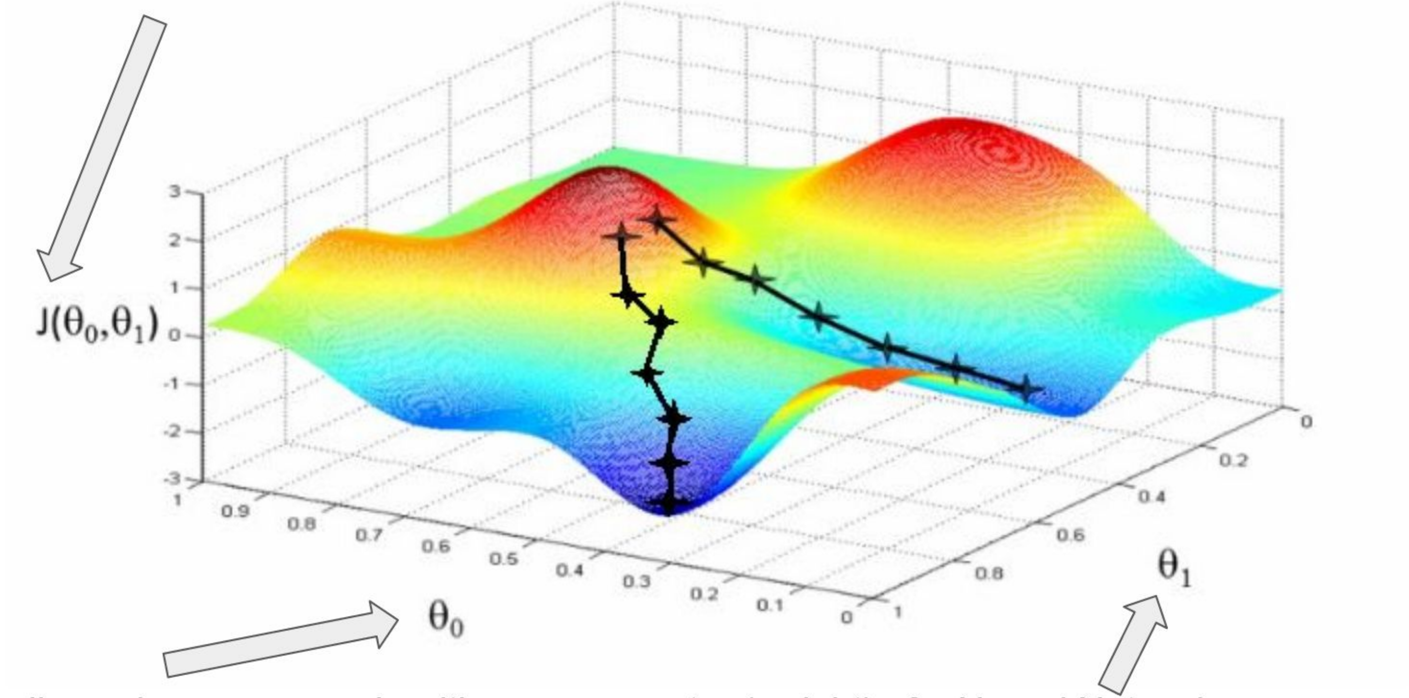
Optimizers

- **Recap:** an optimizer will modify the weights of the NN in such a way that it will (try to) minimize the error of the prediction.
- Most optimizers follow a **Gradient Descent** (next slide) approach.
- 3 classes of distributed optimizers in Distributed Keras:
  - **SingleTrainer** (1 worker, for benchmarking)
  - **EASGD** (Elastic Averaging, ***global variable consensus***)
  - **DOWNPOUR** (Explicit gradient updates are sent to the PS)

Gradient Descent

Goal: optimize parameters in such a way that it minimizes the error.
Problem: Gradient can only be evaluated locally.
Solution: Iteratively, add gradient (slope) until convergence.

Additional dimension to indicate error of expected value, and predicted value.



A single dimension corresponds with one parameter (weight) of a Neural Network.

Spark Driver Implementation

```python
def train(self, data, shuffle=False):
    self.start_service()
    worker = self.allocate_worker()
    numPartitions = data.rdd.getNumPartitions()
    if numPartitions > self.num_workers:
        data = data.coalesce(self.num_workers)
    else:
        data = data.repartition(self.num_workers)
    if shuffle:
        data = shuffle(data)
    for i in range(0, self.num_epoch):
        self.reset_variables()
        data.rdd.mapPartitionsWithIndex(worker.train).collect()
    self.stop_service()
```

Repartition depending on number of workers.

Data iterations.

Spark Worker Implementation

```
def train(self, index, iterator):
    # Deserialize and compile the Keras model.
    try:
        while True:
            batch = [next(iterator) for _ in range(self.batch_size)]
            feature_iterator, label_iterator = tee(batch, 2)
            X = np.asarray([x[self.features_column] for x in feature_iterator])
            Y = np.asarray([x[self.label_column] for x in label_iterator])
            if self.iteration % self.communication_period == 0:
                # Compute Elastic Average and send to the Parameter Server.
            model.train_on_batch(X, Y)
            self.iteration += 1
    except StopIteration:
        pass
```

Interface for Spark's mapPartitionsWithIndex(index, iterator)

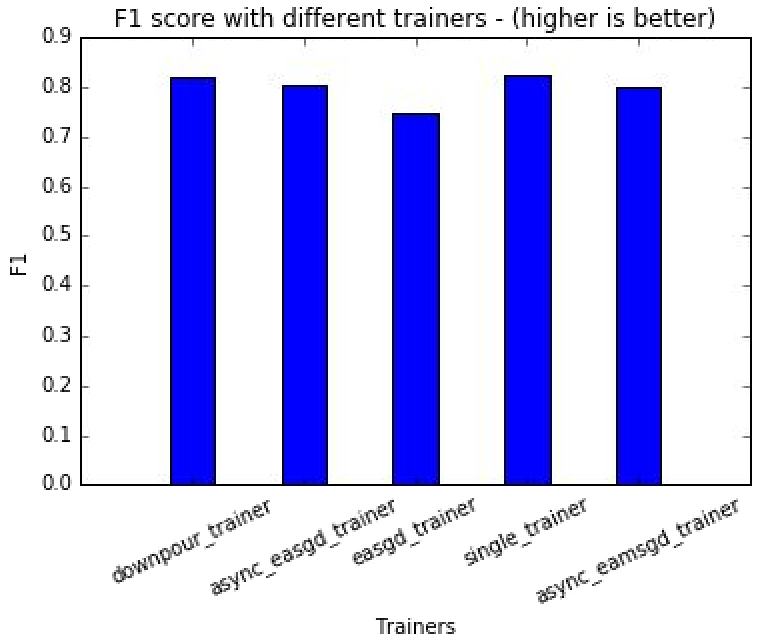
Fetch mini-batch from partition iterator.

Duplicate mini-batch.

Train local replica of model on mini-batch.

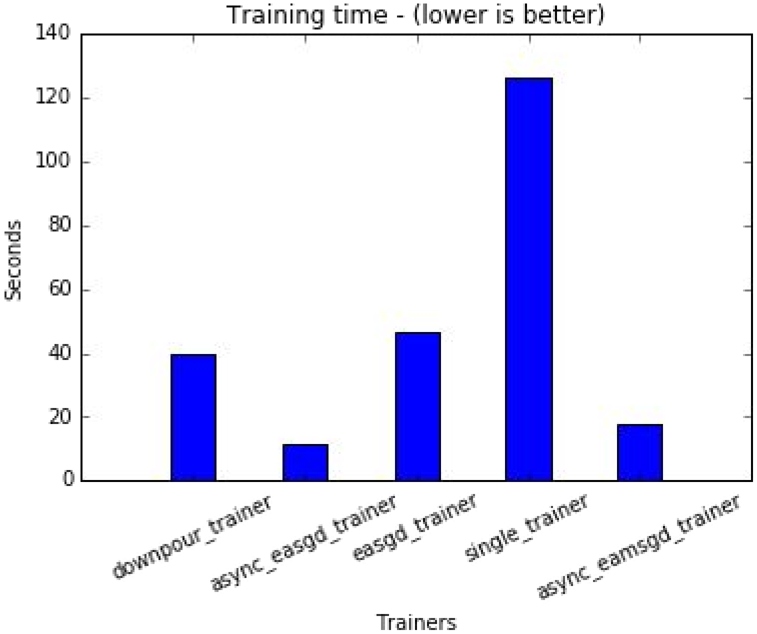
Results (1)

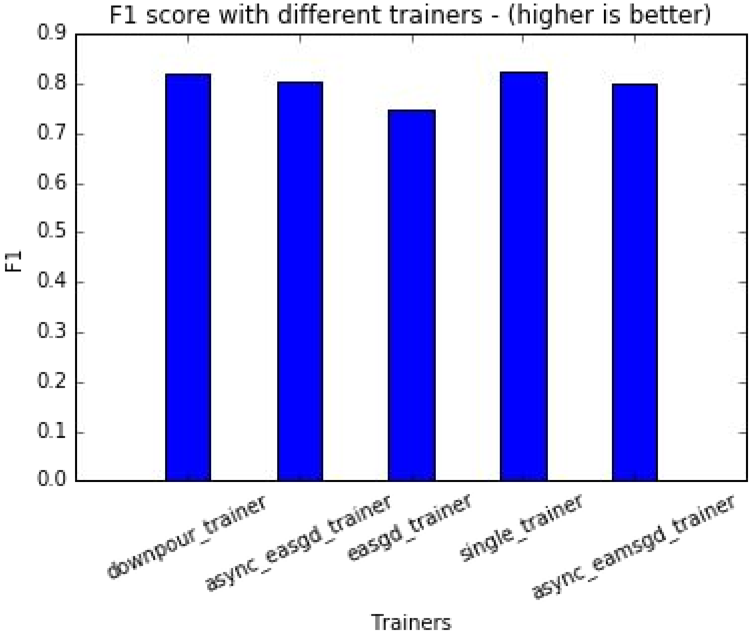
- Similar statistical performance among trainers.
- But single process is still “the best”.
- **But!**



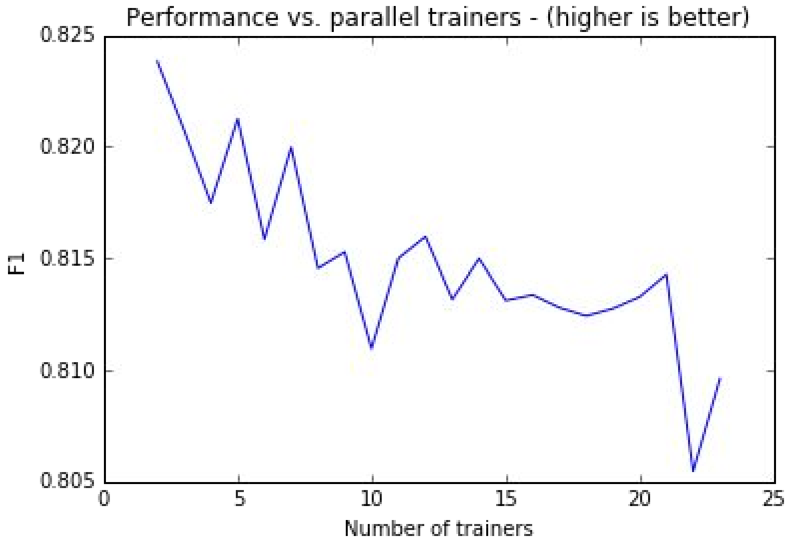
Results (2)

- **Model is computed a lot (10x) faster!**
- But using 24 times the amount of computational resources… (tradeoff)



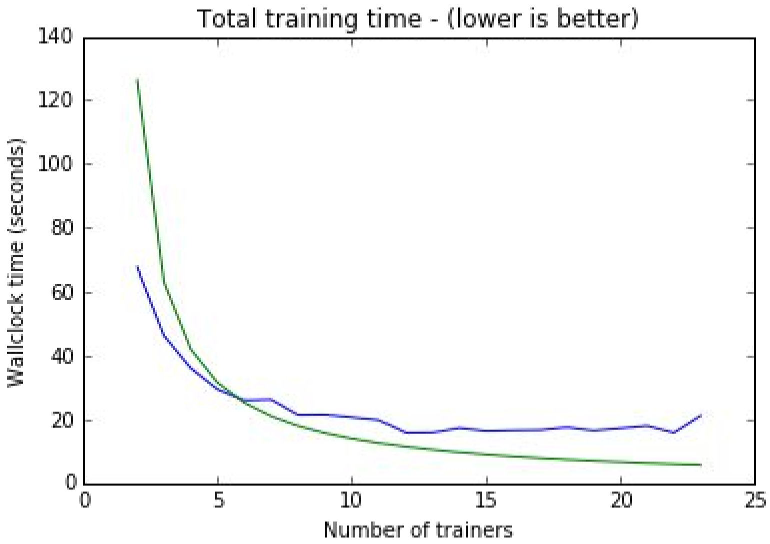


Scaling: statistical model performance



Scaling: training speedup

- Green line depicts the expected value (2 workers, 2 times faster, …)



Future Work

- Stop on target loss (early stopping)
- Training metrics
- Improve parameter transmission performance:
    - Threaded parameter queue
    - Compression
    - HashIndexing (e.g., non-zero values)
    - Gradient residuals

Summary

- We presented an architecture for Distributed Deep Learning on Apache Spark and shown the gain in performance.
- Prototype of “production” environment is being built (dml.cern.ch).
- Found incorrectly derived equation in EASGD research paper during implementation.

Notebook

Complete Apache Spark workflow with more details and experimental findings:

https://github.com/JoeriHermans/dist-keras/blob/master/examples/workflow.ipynb

Feedback + Issues are welcome!

Questions?